UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA DE SISTEMAS

DISEÑO E IMPLEMENTACIÓN DE UN PROVEEDOR DE INFRAESTRUCTURA EN LA NUBE INP, MEDIANTE SEGMENTACIÓN DE RECURSOS

Trabajo de titulación previo a la obtención del

titulo de Ingeniero de Sistemas

AUTORES: WILMER XAVIER CAMAS MAINATO
HERNÁN XAVIER RIERA TAZA
TUTOR: ING. ERWIN JAIRO SACOTO CABRERA, PhD

CUENCA - ECUADOR

2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Wilmer Xavier Camas Mainato con documento de identificación N° 0302603493 y Hernán Xavier Riera Taza con documento de identificación N° 0106805591; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 20 marzo del 2022

Atentamente,

Wilmer Xavier Camas Mainato

0302603493

Hernán Xavier Riera Taza

0106805591

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Wilmer Xavier Camas Mainato con documento de identificación N° 0302603493 y Hernán Xavier Riera Taza con documento de identificación N° 0106805591, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "Diseño e implementación de un proveedor de infraestructura en la nube INP, mediante segmentación de recursos", el cual ha sido desarrollado para optar por el título de: Ingeniero de Sistemas, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 20 marzo del 2022

Atentamente,

Wilmer Xavier Camas Mainato

0302603493

Hernán Xavier Riera Taza

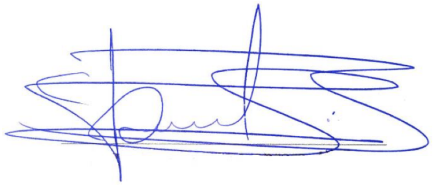
0106805591

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Erwin Jairo Sacoto Cabrera con documento de identificación N° 0301185229, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DISEÑO E IMPLEMENTACIÓN DE UN PROVEEDOR DE INFRAESTRUCTURA EN LA NUBE INP, MEDIANTE SEGMENTACIÓN DE RECURSOS, realizado por Wilmer Xavier Camas Mainato con documento de identificación N° 0302603493 y por Hernán Xavier Riera Taza con documento de identificación N° 0106805591, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 20 marzo del 2022

Atentamente,



Ing. Erwin Jairo Sacoto Cabrera, PhD

0301185229

Agradecimiento

En primer lugar, deseo expresar mi agradecimiento al tutor de esta tesis, PhD. Erwin J. Sacoto Cabrera, por el apoyo brindado durante este proyecto de titulación.
Gracias a mis padres, a mi hermana por el amor y apoyo recibido para culminar mi carrera universitaria. Gracias a mi familia por su apoyo incondicional.

Wilmer Camas

En primer lugar quiero agradecer a Dios por bríndame sabiduría, fortaleza y poder culminar esta etapa de mi vida con éxito, a y de manera especial a PhD. Erwin J. Sacoto Cabrera por permitirme realizar el proyecto de titulación en el grupo de investigación GIHP4C, por su tiempo, apoyo y enseñanzas a lo durante el desarrollo de este proyecto y a mis padres Margarita y Raúl por brindarme su apoyo incondicional durante todos estos años.

Xavier Riera

Dedicatoria

El presente trabajo se los dedico a mis padres Sarita y José quienes con su amor, paciencia y esfuerzo me han permitido llegar a cumplir hoy un meta más para formarme como un profesional.

A mi abuelita Hilda por inculcar en mí el ejemplo de esfuerzo y valentía, de no temer las adversidades. A mi hermana Mayra por su cariño apoyo incondicional, durante todo este proceso, a mis prima Erika por estar conmigo en todo momento. Y a toda mi familia porque con sus oraciones, consejos y palabras de aliento hicieron de mí una mejor persona.

A todos mis amig@s y seres queridos, por acompañarme y apoyarme cuando más los necesite, de manera especial a mi amigo y compañero en este proyecto Xavier Riera, sin olvidar a Luis, Vinicio, Damián, Magna que formamos un grupo no solo de estudio sino de amistad.

Wilmer Camas

A mis padres Margarita y Raul por su tiempo, paciencia, amor, consejos y por ser mi inspiración de lucha y perseverancia para conseguir mis sueños.

A mi familia por el apoyo a lo largo de mi vida, en especial a mis hermanos Ernesto y Franklin por compartir momentos de felicidad, por bríndame animo y motivación.

A mis amigos en especial a mi compañero de proyecto Wilmer por la confianza puesta en mí. A Luis, Juan Pablo, Damian, Vinicio, Magda por su amistad, apoyo y sentido de humor durante todo este trayecto de vida.

Y por ultimo a todas la personas que de alguna manera contribuyeron a hacer posible cumplir esta etapa.

Xavier Riera

Índice de Contenido

Índice de Tablas

Índice de Figuras

Resumen

En la actualidad, las organizaciones buscan una manera eficiente para desplegar y administrar su infraestructura de TI en una nube de forma automatizada. Con los avances en la tecnología de automatización facilitan la creación de una infraestructura mediante la programación de scripts describiendo los requerimientos de hardware y software en forma de código. De esta manera utilizando el enfoque de Infraestructura como código. Es necesario del uso de varias herramientas, como puede ser Terraform que es una poderosa herramienta para el aprovisionamiento y Ansible se utiliza para la gestión de la configuración e implementación de aplicaciones, ya que cada herramienta proporciona varios módulos para simplificar y facilitar las tareas.

Los scripts brindan la capacidad de crear, configurar y administrar la infraestructura como puedes ser servidores, bases de datos, balanceadores de carga, topología de red, etc., en archivos de configuración de Terraform y enviar esos archivos a un sistema de control de versiones. Luego ejecutar ciertos comandos de Terraform, como puede ser: "terraform apply", para implementar la infraestructura requerida. El binario terraform analiza el código, lo traduce en una serie de llamadas a la API de la nube de OpenStack, para implementar estos requerimientos al mismo tiempo, la herramienta Ansible permite la configuración de las instancias desplegadas desde cero para implementar sus aplicaciones sobre estos, permitiendo la administración de la configuración y la implementación de aplicaciones.

El presente proyecto de tesis propone el "Diseño e Implementación de un proveedor de infraestructura en la nube InP, mediante segmentación de recursos", tiene como objetivo principal implementar y desplegar una infraestructura basado en la nube privada en OpenStack mediante la herramienta Terraform. Para esto se propone un metodología para la construcción de infraestructuras aplicando la Infraestructura como Código bajo un enfoque guiado por escenarios empleando distintas topologías con diferentes requerimientos y configuraciones en los que se

puede utilizar la implementación automatizada, introduciendo los conceptos y enfoques aplicados sobre proyectos de infraestructura definiendo un modelo genérico y aplicable a cualquier organización.

Palabras claves: Infraestructura como Código, Aprovisionamiento, Terraform, Ansible, OpenStack.

Abstract

Today, organizations are looking for an efficient way to deploy and manage their IT infrastructure in a cloud in an automated manner. With advances in automation technology they facilitate the creation of an infrastructure by programming scripts describing the hardware and software requirements in the form of code. Thus using the Infrastructure as code approach. It is necessary to use various management tools. Such as Terraform which is a powerful tool for provisioning and Ansible is used for configuration management and application deployment, as each tool provides several modules to simplify and facilitate tasks.

The scripts provide the ability to create, configure and manage infrastructure such as servers, databases, load balancers, network topology, etc., in Terraform configuration files and send those files to a version control system. Then run certain Terraform commands, such as terraform apply, to deploy the required infrastructure. The terraform binary parses the code, translates it into a series of API calls to the OpenStack cloud, to implement these requirements at the same time, the Ansible tool allows configuration of deployed instances from scratch to deploy their applications on top of those servers, allowing configuration management and application deployment.

This thesis project proposes the "Design and Implementation of an InP cloud infrastructure provider, through resource segmentatio", its main objective is to implement and deploy an infrastructure based on the private cloud in OpenStack using the Terraform tool. For this, a model for building infrastructures is proposed applying Infrastructure as Code under a scenario-driven approach using different topologies with different requirements and configurations in which the automated implementation can be used, introducing the concepts and approaches applied to infrastructure projects defining a generic model applicable to any organization.

Keywords:

Infrastructure as Code, Provisioning, Terraform, Ansible, OpenStack.

Índice de abreviaturas

VM	Virtual Machine,(Máquinas Virtual).
SO	Sistema Operativo.
TI	Tecnología de la información.
IP	Protocolo de Internet.
IaC	Infrastructure as Code,(Infraestructura como código).
API	Interfaz de Programación de Aplicaciones.
SSH	Secure SHell.
SaaS	Software as a Service,(Software Como Servicio).
PaaS	Platform as a Service,(Plataforma como servicio).
IaaS	Infrastructure as a Service,(Infraestructura Como Servicio).
CD	Continuous Deployment.
DNS	Sistema de Nombres de Dominio
CPU	Unidad Central de Procesamiento.
RAM	Memoria de Acceso Aleatorio.
CLI	Interfaz de Linea de Comandos.

Introducción

La construcción de infraestructura es un arte complejo y en evolución, que exige mejoras repetitivas que involucran aspectos como la mantenibilidad, la escalabilidad, la tolerancia a fallas y el rendimiento. En el entorno tradicional construir e implementar componentes de infraestructura era una tarea manual y tediosa que se traduce en retrasos y disminución de la agilidad organizacional, ya que para desplegar una aplicación se debía implementar servidores físicos, instalar y configurar manualmente el software necesarios para el correcto funcionamiento. Este proceso puede llevar días o incluso semanas. Por lo que surgió un nuevo paradigma llamado Infraestructura como código (IaC por sus siglas en ingles) para enfrentar la implementación y configuración manual de la misma. IaC ayuda al tratar la infraestructura como parte de la solución general al convertirla en código y ser interpretado por alguna herramienta. Permitiendo aprovisionar los recursos/componentes de una forma automatizada. Terraform es una herramienta que ayuda a poner en práctica lo antes dicho, mediante sus implementaciones idempotentes y su lenguaje basado en la configuración, para ello emplea la interfaz de línea de comandos de Terraform (Terraform CLI).

La IaC permitir que las organizaciones abstraigan de la capa física del sistema e incorporen las mejores prácticas de software en la gestión de la infraestructura. Incluyendo implementar un control de versiones de código.

Este proyecto se enfoca en el estudio y aplicación del paradigma de IaC que permite el aprovisionamiento de la infraestructura de TI. El proyecto se encuentra estructurado de la siguiente manera. En el capítulo 1 se presenta el problema que se a resolver con la solución planteada, en el capítulo 2 los objetivos a alcanzar, capitulo 3 marco teórico, capitulo 4 diseño de la arquitectura para el aprovisionamiento y desplegué de la nube privada, capitulo 5 metodología implementada para el desarrollo del proyecto, finalmente, capítulo 6 presenta un los resultados

del uso de estas herramientas para aprovisionar una instancia de hardware en OpenStack.

Capítulo 1

Problema

1.1 Definición del problema

Hoy en día una empresa van creciendo de una manera exponencial por lo que tienen que adaptase a las necesidades que lo requieran por lo cual tienen la necesidad de adquirir diferentes Infraestructuras para virtualizar los servicio y elaborar los procesos que conlleva una organización de una manera automatizada, esto implica el incremento de servidores virtuales, mayor gestión y control de la infraestructura provocando un crecimiento tanto en la administración, mantenimiento y gestión de las máquinas virtuales (VM).

De acuerdo a lo descrito en [1], la gestión de la infraestructura de tecnología de la información(TI) es un proceso que se lo ha venido haciendo manualmente. Comúnmente las empresas instalan físicamente los servidores y los configuran uno a uno.

Como era de esperar, este es un proceso manual que a menudo daría lugar a varios problemas, como por ejemplo en [1, 2], se describen algunos problemas como la escalabilidad y la disponibilidad pero al final, todo se reduce a la velocidad de respuesta. Dado que la configuración manual es lenta, las aplicaciones a menudo tienen problemas en horas pico para el acceso, mientras que los administradores del sistema intentan desesperadamente configurar los servidores para administrar la carga entre los servidores.

Otro problema es el costo, tendría que contratar a muchos profesionales para realizar las tareas necesarias en cada paso del proceso, desde ingenieros de redes hasta técnicos de mantenimiento

de hardware físico.

Para dar solución a estos problemas se se crea la virtualización este permite desplegar maquinas virtuales de forma mas rápida, así solucionando alguno de los problemas mencionados pero el proceso de creación de máquinas virtuales es tedioso cuando se tiene gran cantidad de MV. Para ejecutar este procedimiento, se debe realizar la instalación de un Sistema Operativo y luego para generar un procedimiento para desplegar un tipo de servidor en específico. Además, se debe realizar la configuración de red obtener o asignar una dirección IP(Protocolo de Internet), definir el almacenamiento, habilitar puertos, etc. y este no se adapta automáticamente a cambios que se realizan en la Infraestructura.

De lo anterior se resume que las empresas tienen dificultades en varios procesos sean más eficientes has lograr procesos automatizados de aprovisionamiento y gestión de infraestructura TI, así sustituyendo los procedimientos estándar de operación.

1.2 Justificación

Los autores en [3], indican que a existido un gran avance y evolución durante estos años en cuanto a las practicas del desarrollo del software, mientras que en la practica de operaciones de TI no a evolucionado al mismo ritmo. Hoy en día existen muchos equipos de TI que utilizan una combinación de imágenes maestras, configuraciones manuales, scripts personalizados o herramientas antiguas para la administración de infraestructura. Esto provoca que el desplegué de entornos sea lento y con errores.

En [3] se describe la diferencia entre la automatización de la infraestructura y la infraestructura como código (IaC).

La automatización de la infraestructura posibilita llevar a cabo acciones de manera repetida

a través de un largo número de nodos mientras que, la segunda utiliza técnicas, prácticas y herramientas del desarrollo de software para asegurar que esas acciones son probadas a fondo antes de ser aplicadas a los sistemas de alta criticidad para el negocio.

La IaC, se refiere a la práctica de scripts para configurar la infraestructura de manera de código en vez de configurar VM en forma manual, lo descrito permite convertir a la infraestructura altamente elástica (repetible y escalable). Considerando lo descrito, es necesario diseñar una metodología e implementar una nube privada, con la finalidad de determinar si el uso de IaC permite cumplir los desafíos que enfrentan los equipos de TI en la gestión y administración de la infraestructura.

A partir el surgimiento de la virtualización de servidores y en la actualidad la adopción de la Cloud Computing en las empresas, han surgido una gran cantidad de herramientas para la automatización de crear infraestructura, estas herramientas lo hacen con la programación de scripts así generando un entorno mas rápido, seguro y consistente para alojar aplicaciones.

El manejo de estas tecnologías debería ser implementadas en todas las empresas que manejan MV, e incluso en uso interno. Esto llevara a ser mas eficientes en la gestión de servicios para los clientes , así como para el desplegué de aplicaciones en producción, algo que agrega estas tecnologías es la adopción de la documentación así logrando automatizar procesos en la infraestructura.

Capítulo 2

Objetivos Generales y Específicos

2.1 Objetivo General

Diseñar e implementar la infraestructura de una Nube Privada utilizando IaC, con la finalidad de mejorar el despliegue, aprovisionamiento, gestión y administración de la creación de maquinas virtuales.

2.2 Objetivos Específicos

- Explicar los principios y estudiar las diferentes soluciones de implementar una Infraestructura como Código.
- Listar y analizar los desafíos que enfrentan los equipos de TI en la gestión y administración de la infraestructura.
- Realizar la implementación de OpenStack como nube privada.
- Elaboración de scripts mediante Terraform que permitan desplegar la infraestructura
- Demostrar la automatización mediante despliegue de servicios en la nube privada.

Capítulo 3

Revisión de la literatura o fundamentos teóricos

3.1 Cloud Computing

Cloud Computing (Computación en la Nube) se define en [4, 5, 6], como una tecnología que permite a un grupo de servidores brinden la capacidad de compartir sus recursos informáticos con otros dispositivos a través de Internet.

Para proporcionar a los clientes recursos, servicios en la nube y satisfacer sus necesidades, la computación en la nube utiliza tecnologías de virtualización y asignación de recursos. Proporcionando a los dispositivos que necesiten recursos virtualizados compartidos (hardware, software y plataforma) tal como indica el autor en [7].

De acuerdo con el Instituto Nacional de Estándares y Tecnología (NIST) [8], "La computación en la nube es un modelo para permitir el acceso conveniente a la red bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con un mínimo esfuerzo de administración o interacción del proveedor de servicios".

3.1.1 Características.

Según los autores en [5, 9, 10, 11, 12] mencionan las siguientes características de Cloud Computing, basados en el NIST:

Autoservicio bajo demanda: de acuerdo con el autor [13] se refiere al acceso de los servicios o recursos por parte del cliente sin necesidad de una interacción con el proveedor. Dicho en otras palabras, permite el aprovisionamiento de recursos de la nube del proveedor a necesidad de los clientes, como puede ser CPU (Contents Under Pressure), almacenamiento, accedo a la red,etc.

Amplio acceso a la red: según el autor en [11] describe que el amplio acceso a la red brinda a los clientes la posibilidad de acceder a los servicios alojados en la nube del proveedor a través de Internet desde cualquier ubicación por medio de una amplia gama de dispositivos como, por ejemplo, estaciones de trabajo, tabletas, computadoras portátiles, teléfonos móviles, etc.

Agrupación de recursos: A través del modelo de infraestructura compartida, los recursos del proveedor de la nube se comparten en sus servidores permitiendo un modelo de uso de múltiples clientes. A los consumidores se les asignan de forma dinámica los recursos que pueden ser físicos o virtuales como el almacenamiento, procesamiento, memoria y otros recursos, según las necesidades que lo requieran. Generalmente, los usuarios no tienen idea de la ubicación exacta de los recursos que se les proporcionan, como lo plantea el autor en [13] .

Elasticidad rápida: En [5], se define como elasticidad a la capacidad que permite configurar y liberar de manera flexible, y en algunos casos de manera automática los recursos del proveedor, para expandirse y contraerse rápidamente según la demanda de requerimientos de los usuarios, ya sea con potencia de cómputo o espacio de almacenamiento de una sola VM o expandiendo las capacidades de una red completa de servidores.

Servicios medidos: los sistemas de un proveedor de una nube están diseñados para gestionar y monitorizar el uso de recursos proporcionados al cliente automáticamente, por ejemplo, procesamiento, ancho de banda y cuentas de usuarios activos. Con la finalidad de optimizar el uso de recursos manteniendo un control a través de la capacidad de medición, proporcionando trans-

parencia tanto para el proveedor como para el consumidor del servicio utilizado. Este sistema generalmente se modela según el paradigma de pago por uso, como destaca el autor en [12].

3.1.2 Tipos de Servicios de Cloud Computing

De acuerdo con el NIST, define tres tipos de modelos de servicios de computación en la nube: infraestructura como servicio (IaaS), plataforma como servicio (PaaS) y software como servicio (SaaS), como se visualiza en la Figure 3.1.

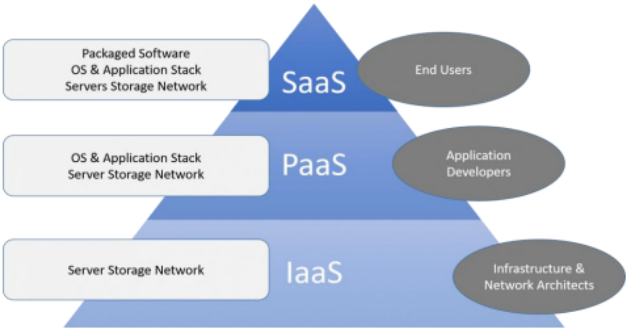


Figura 3.1

Modelos de Cloud Computing [5].

3.1.2.1 Software como Servicio (SaaS).

En base a la Figure 3.1 este modelo en la nube, según lo que definen los autores en [14, 9], se centra en los usuarios finales, brindando la oportunidad de acceder a las aplicaciones que proporciona el proveedor en cualquier momento si lo solicitan, estas aplicaciones se ejecutan en los servidores en la nube del proveedor. Dicho de otras palabras, el proveedor debe encargarse del mantenimiento, disponibilidad de los servicios, instalando las aplicaciones y el software necesarios, ofreciendo una aplicación como servicio basado en la nube a un cliente y disponible a través de una red, en lugar de instalar el software en su computadora.

Los usuarios del servicio no administran ni controlan la infraestructura de la nube subyacente, incluida la gestión de software y hardware.

Ejemplos de este modelo como servicio con los documentos y Gmail, Microsoft Office 365, etc. Se puede acceder a las aplicaciones a través de una interfaz de un navegador web.

3.1.2.2 Plataforma Como Servicio (PaaS).

En base a la Figure 3.1 y tal como se indica en [14], PasS permite a los consumidores utilizar lenguajes de programación, servicios y herramientas compatibles con el proveedor para implementar sus propias aplicaciones en un entorno basado en la nube. EL consumidor no tiene control de la infraestructura subyacente, incluido la redes, los servidores, el almacenamiento o el sistema operativo; sin embargo, el usuario tiene control total sobre las aplicaciones implementadas y los ajustes de configuración para el entorno que aloja el código del cliente.

Ejemplos de proveedores de PaaS son App Engine de Google, Azure de Microsoft, Makara de Red Hat, AWS (Amazon Web Services) Elastic Beanstalk, AWS Cloud Formation, etc. Como plantea [15] estos proveedores de servicios en la nube (CSP) tienen la capacidad de admitir diferentes sistemas operativos en el mismo entorno físico servidor.

3.1.2.3 Infraestructura Como Servicio (IaaS).

De acuerdo con [5, 16], IaaS es la capa más baja de servicio en la nube como se visualiza en la Figure 3.1. Este modelo servicio del proveedor provee a los consumidores un conjunto de recursos informáticos físicos o virtualizados como la capacidad de procesamiento, redes, memoria, almacenamiento, máquinas virtuales con su sistema operativo pre-configurado, etc. según sea las especificaciones que sean necesarias. En este modelo, el usuario final no gestiona la infraestructura

de nube subyacente, pero tiene control sobre el entorno implementado con las configuraciones necesarias, además de controlar su software, sistema de almacenamiento y las aplicaciones

Los usuarios pueden ampliar y reducir los recursos en función de sus requisitos en cualquier momento. Según [11], IaaS utiliza tecnología de virtualización para convertir recursos físicos en recursos lógicos que los clientes pueden aprovisionar y liberar dinámicamente según sea necesario.

"Algunas de las principales empresas que ofrecen infraestructura como servicio incluyen Rackspace Cloud Servers, Microsoft Azure, Amazon Web Sercice (AWS) y Google Compute Engine (GCP), OpenStack, OpenNebula, Eucalyptus, IBM" como describe el autor en [17].

3.1.3 Modelos de Infraestructura de Cloud Computing

3.1.3.1 Cloud Privada.

Es una infraestructura en la nube en [18] el autor define que es íntegramente gestionada por una sola organización ya que sus recursos computacionales no están disponibles para el público, es decir, sus servicios de TI y recursos se aprovisionan para el uso solo a sus usuarios y/o terceros autorizados que están dentro de los límites de la organización.

Es un modelo de implementación más adecuado, tiene el potencial de darle a la organización mayor control sobre la infraestructura y los recursos computacionales. La nube privada generalmente utiliza tecnologías de virtualización para optimizar la utilización del hardware (componentes de computación, memoria, red y almacenamiento) de los usuarios.

3.1.3.2 Cloud Publica.

Como nos indica [5, 16], la infraestructura física de un proveedor está a disposición al público o

cualquier empresa a través de internet; es decir, la infraestructura puede ser propiedad de una organización. El proveedor ofrece sus recursos y servicios de manera libres o son vendidos con la modalidad de pago por uso (pay as you go). Esto permite al cliente el uso de los servidores y demás servicios que se utilizan de forma compartida. En [19] se define como nube plública como aquella que "está alojada, operada y gestionada por un proveedor desde uno o más centros de datos".

3.1.3.3 Cloud Híbrida.

Este modelo de infraestructura en la nube surge de combinar una nube privada con una nube pública o acoplar múltiples nubes (privadas o públicas). Interoperando de maneja conjunta, proporcionando una mayor capacidad con los recursos, mejorando el rendimiento de los servicios y/o aplicaciones desplegadas evitando perder calidad de servicio, de acuerdo a lo expuesto en [5, 16].

Con base a lo descrito en [19],se destaca que la nube híbrida tiene la cualidad de ínter operar de "Con una nube híbrida las organizaciones pueden ejecutar aplicaciones no fundamentales en una nube publica, mientras mantienen las aplicaciones fundamentales y los datos sensibles internos en una nube privada".

El estudio de la computación en la nube brinda soporte a varias tecnologías y redes como 5G tal como se describe en [20, 21, 22, 23], así como permiten del desarrollo y soporte de proyectos basados en Internet de las Cosas (IoT), como se describe por los autores en [24, 25, 26, 27, 28].

3.2 Gestores de Infraestructura IaaS (Nubes Open Source)

3.2.1 OpenNebula

De acuerdo con [29], los autores describen que OpenNebula es una solución de código abierto simple pero potente para crear y administrar cualquier tipo de nube: privada, pública o híbrida. Combina tecnologías de virtualización y contenedores con tenencia múltiple, provisión automática y elasticidad para ofrecer bajo demanda aplicaciones y servicios.

Esta plataforma IaaS permite la implementación dinámica, permite asignar recursos físicos a un grupo de servicios virtualizados como señala el autor en [30].

3.2.2 OpenStack

En [31, 32], el autor expone que OpenStack es una plataforma de código abierto que permite implementar y configurar nubes tanto privadas como públicas en los centros de datos dueña de la infraestructura. Esta plataforma despliega una infraestructura virtual sobre los recursos físicos, así permitiendo el control de los contenedores, maquinas, sistema de almacenamiento, redes e incluso recursos de aplicaciones. OpenStack se gestiona a través de panel web, línea de comandos (CLI) y una interfaz de programación de aplicaciones (API).

En base a planteado en [33] señala que es un proyecto global para crear una plataforma de cómputo open source en el cloud, que cumpla con las necesidades de los proveedores de servicios en nube pública, privada e híbrida, independientemente de su tamaño, que sea fácil de implementar y escalable

3.2.2.1 Arquitectura De OpenStack.

De acuerdo con [6, 34, 35, 36], los componentes básicos de la arquitectura de OpenStack son los siguientes:

Cómputo OpenStack (Nova): es un controlador original de la plataforma. Es el encargado de instanciar y administrar máquinas virtuales, contenedores de aplicaciones que un usuario lo a solicitado, así como de la gestión de una red virtual y asignar IP para cada instancia. La implementa se puede realizar sobre diferentes hipervisores, incluyendo Quick Emulator(QEMU) y Kernel-based Virtual Machine(KVM) de acuerdo a lo planteado en [34].

Almacenamiento de objetos (Swift): como describe el autor en [6] este servicio permite el almacenamiento redundante, tolerante a fallos y escalabre para un sistema de objetos. Swift está orientado principalmente a datos estáticos, como copias de máquinas virtuales, almacenamiento simple de ficheros, almacenamiento de streamings de audio/vídeo, copias de seguridad y archivos.

Redes (Neutrón): de acuerdo con [35] señala que es un componente es responsable de la conectividad de asociar dispositivos de red con redes. Los administradores pueden gestionar, configurar y asignar direcciones IP a los dispositivos.

OpenStack Dashboard (Horizon): es una interfaz web que incorpora todas las herramientas para la para administración de OpenStack, dicho de otras palabras, los administradores pueden modificar y ampliar para una nube específica, basado en lo que plante el autor en [35].

Servicio de identidad (Keystone): con referencia a lo planteado en [35] expresa que es un servicio que permite gestionar la autenticación de los usuarios de OpenStack, Es decir, acceder a todos los componentes que ofrece la plataforma.

Almacenamiento en bloque (Cinder): ofrece un servicio de almacenamiento de datos en bloques. Este componente es responsable de crear, destruir, asignar, etc. volúmenes de datos

persistentes en las instancias, además de crear una copia de seguridad de ese volumen como describe el autor en [36].

3.3 Infraestructura de la TI

Según definen los autores en [37, 38] la infraestructura de tecnología de la información (TI) consiste en contar con varios elementos necesarios para una gestionar y operar el departamentos del TI, así logrando ejecutar las operaciones relacionadas con la tecnología y el almacenamiento de los datos de la empresa. La infraestructura de TI se puede dar en una cloud computing o bien en el data center de la empresa. Además la infraestructura del TI se constituye por componentes como Redes, Hardware, Software y todo lo relacionado con el control, desarrollo, monitoreo y brindar el soporte de los servicios que ofrecidos por el departamento de TI.

3.3.1 Tipos de modelo de infraestructura de TI

De acuerdo con [39, 40] describen los siguientes tipos de infraestructuras de TI:

3.3.1.1 Infraestructura tradicional. La infraestructura tradicional citando a [39] manifiesta que esta compuesta por componentes que son de propiedad de la empresa como el hardware y software como son servidores, sistema de almacenamiento de datos, hardware de red, etc. Estos tipos de infraestructura soy costosas, tienen un alto consumo de energía y requieren un amplio espacio físico para el funcionamiento.

3.3.1.2 Infraestructura de nube. La infraestructura de la nube en [41] el autor señala que tiene un parecido a la infraestructura tradicional con la diferencia de que se puede acceder de a los recursos a través de internet y permite usar la virtualización para emplear recursos compu-

tacionales sin la necesidad de instalaciones locales. Se puede implementar una nube privada por nuestra cuenta o bien se puede usar un proveedor de nube publica.

3.3.1.3 Infraestructura hiper-convergente. La infraestructura hiper-convergente de acuerdo a lo planteado en [42] indica que ofrece un enfoque centralizado y unificada que permite gestionar y administrar la red, los recursos informáticos, el almacenamiento. Ofreciendo la virtualización de la maquinas, redes virtuales y almacenamiento datos definido por software, todo esto desde una sola interfaz.

3.3.2 Desafíos que enfrentan los equipos de TI en la gestión y administración de la infraestructura.

Con base a lo planteado en [38, 43] se indica que en la actualidad empresas cuentan con su propio centros de datos o con varios centros de datos para operar, lo dicho aplica para grandes empresas que no pueden o que no quieren aumentar su presencia a nivel global en cuento a sus servicios.

Estas empresas vienen trabajando con un infraestructura tradicional, para la ejecución de sus aplicaciones y para brindar servicios a los clientes, el departamento de TI en estas infraestructuras cuentan con una Base de Datos de Gestión de la Configuración(CMDB por sus siglas en ingles, Configuration Manager database) para el inventario de los componentes de la infraestructura, esta información es tomada por una aplicación de gestión de servicio(ITSM por sus siglas en ingles IT Service Manager), que es usado para la gestionar los cambios en la infraestructura, la comunicación entre los equipos que dan soporte al igual que la comunicación entre los usuarios.

En [44] el autor describe que en la infraestructura tradicional para realizar la creación o mantenimiento se necesita realizar una petición del usuario que necesita dichos recursos, esta petición

pasaba por diferentes equipos que cumplían con la definida tarea para completar la petición, aun que el flujo presentado en la infraestructura tradicional servía para organizar y gestionar grandes entornos, a continuación según los autores [44, 38] se presentaba los siguientes desafíos para IT:

- **Mantenimiento:** como se conoce los S.O tienen mejoras continuas, parches para solucionar problemas, al igual se presentan vulnerabilidades que pueden afectar a la infraestructura dichos problemas deben ser solucionados de la manera mas rápida, lo que ocasiona que diferentes aplicaciones dejen de funcionar durante el mantenimiento.
- **Tiempo de desplegué:** el desplegué de una aplicación en este tipo de infraestructura puede durar de un día como se podrán extender dos semanas, ya que el equipo de operaciones llegan a recibir varias peticiones para el desplegué de alguna aplicación, además las misma va a pasar por diferentes equipos como por ejemplo va a ser aprovisionado por un equipo y la configuración por otro, todo esto genera retrasos.
- **Incidentes:** se pueden producir incidentes tanto de infraestructura como por fallos en la aplicaciones, lo cual produce un fallos en la continuidad del servicio que tienen que ser resueltos de la forma mas eficiente.
- **Documentación:** mantener la documentación actualizada de la configuración y instalaciones, ya que las mismas están en constante cambio, y no se actualizan con las nuevas cambio en la configuración o nuevas programas.

3.4 Infraestructura como Código

En [45] el autor define que la "Infraestructura como código es un enfoque para administrar la infraestructura de TI para la era de la nube, los microservicios y la entrega continua que se basa en prácticas del desarrollo de software". Esto nos indica que la IaC permite describir

la configuración de la infraestructura a través de software, en este caso código en archivos de configuración.

Según los autores [2, 46, 47, 48, 49] la IaC es un paradigma que permite que todos los aspectos de la infraestructura de TI y sus configuraciones se conviertan en código. Dicho de otras palabras, permite definir y controla la infraestructura como si fuese código, incluyendo redes, máquinas virtuales, configuración de aplicaciones, bases de datos, etc. lo que conlleva a reducir el tiempo de despliegue de toda la infraestructura en cuestión de minutos ejecutando un solo archivo. Lo que permite la gestión granular de cambios, la uniformidad de los servidores y el potencial de escalabilidad rápida. La infraestructura se define en un formato de código basado en la sintaxis adecuada en un archivo de configuración que se puede compartida y reutilizada. IaC permite el uso de un Sistema de Control de Versión (VCS) para mantener el control de calidad del código mediante la ejecución de pruebas (pruebas unitarias, de integración y de aceptación); realizar depuración de códigos, revisiones de códigos; Tener versionado el código no permitirá volver a una versión estable en caso de que una versión actual cause un error o en sí no funcione. De esta manera se tiene un controlado por versiones para las diferentes configuraciones implementadas.

En [50] el autor expone que "la herramienta Terraform realiza un despliegue controlado de los principales recursos principales como Azure Kubernetes Service (AKS), API Manager, virtual networks (vnet) y sus correspondientes subnets, como recursos de carga y balanceadores de carga, y baúles de contraseñas/claves de despliegues y realiza una comparación entre los diferentes métodos que se pudo aprovisionar".

Como se expone en la Figure 3.2 se evidencia que al hacerlo de una forma manual tanto la implantación como el desplegué, puede generar errores y por ende toma un tiempo importante, considerando que la implementación de estos proyectos utiliza metodologías ágiles con una cul-

Método					
Portal	Recurso	Tiempode implementación (Minutos)	Errores en la implemntación	Tiempo de despliegue (Minutos)	Errores en el despliegue
	AKS	50	10	40	2
	APIM	45	15	30	2
	VNET	25	2	15	1
AZ CLI	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	35	10	35	1
	APIM	25	12	27	1
	VNET	15	4	12	1
Script	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	40	15	25	2
	APIM	25	10	24	3
	VNET	15	5	10	2
Terraform	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	25	2	15	0
	APIM	15	2	10	0
	VNET	5	1	1	0

Figura 3.2

Comparativa de tiempos en los despliegues de infraestructura en diferentes proveedores [50].

tura DevOps (términos ingleses development y operations) para minimizar el tiempo, utilizando IaC usando Terraform se tiene un desplegué con un menor tiempo y con menos errores y tenemos un control de versiones de la infraestructura.

3.4.1 Principios de la Infraestructura como Código

En [46, 48, 49] los autores describe estos principios para que pueda comprender mejor cómo implementarlos y cómo ayudan a su proceso de IaC.

- **Todo sistema debe ser reproducible:** Debido a que el sistema está definido en código en un archivo legible, la infraestructura se puede aprovisionada fácilmente varias veces. Todos los detalles sobre el nombre de equipo, el nombre de la red, etc. deben estar en el archivo de configuración utilizado para el IaC para permitir una fácil repetición los describe el autor en [51].
- **Todo sistema debe ser desechable:** Cuando hablamos de IaC el autor en [52] expone que esto implica dinamismo en el sentido de que todas las facetas de la infraestructura como como

se puede cambiar el tamaño, eliminar, etc. Estos cambios no pueden influir en el correcto funcionamiento de la infraestructura. La infraestructura debe ser confiable y consistente en todos los cambios. Se vuelve particularmente importante contar con una infraestructura dinámica de la que se pueda disponer fácilmente en caso de falla sin crear un problema para otros módulos de infraestructura.

- **Todo sistema debe ser consistente:** De acuerdo con el autor en [52] menciona que cuando se crea un sistema usando IaC, es esencial, por ejemplo, que dos servidores configurados con el mismo script sean iguales. La única diferencia puede ser la dirección IP o el nombre del servidor. Este principio evita inconsistencias, como cuando creamos un servidor en OpenStack, GCP o AWS. Definimos el servidor usando una plantilla; esta plantilla se utiliza para definir el tamaño de almacenamiento del disco duro, la cantidad de CPU, etc. Esta consistencia es importante para evitar la interrupción de los servicios.
- **Todo sistema debe ser repetible:** Cuando definimos un archivo IaC, necesitamos definir un archivo que se pueda repetir y el resultado debe ser idéntico para cada ejecución de ejecución. Si el proceso no es repetible, como en los cambios que se manifiestan durante la ejecución, tendremos inestabilidad como lo expone el autor en [48].
- **El diseño del sistema siempre cambia:** en [48] el autor expresa que con una infraestructura convencional, todos los cambios generan sobre costos innecesarios y demoras. Debido a la naturaleza de TI, nadie puede prever la frecuencia con la que será necesario realizar cambios en la infraestructura. IaC, especialmente junto con un entorno de nube, puede resultar en tiempos de respuesta más rápidos. La automatización con configuraciones comprobadas probadas en entornos más bajos refuerza esto.

3.4.2 Categorías de la Infraestructura como Código

El principal objetivo de la IaC es administrar casi todo utilizando código, incluidos redes, servidores, bases de datos, archivos de logs, etc. En este contexto, de acuerdo con [53, 48] existe varias de herramientas IaC que se describen a continuación:

- **Script Ad Hoc**

  La forma más sencilla de automatizar cualquier cosa de acuerdo con el autor en [53] es escribir un script con una secuencia de comandos y ejecutar esa secuencia en un servidor. La administración se complica cuando intentamos ejecutar el script en una docena de servidores, bases de datos, balanceadores de carga, configuraciones de red, etc. Los scripts ad hoc son excelentes para tareas pequeñas y únicas, pero si va a administrar toda su infraestructura como código, debe usar una herramienta de IaC diseñada específicamente para el trabajo.

- **Herramientas de gestión de la configuración**

  Son herramientas de administración de configuración, lo que significa que están diseñados para instalar y administrar software en servidores aprovisionados en un proveedor en la nube como lo señala en [53].

- **Herramientas de plantillas de servidor**

  Desde el punto de vista del autor en [48] destaca que una alternativa popular a la gestión de la configuración son herramientas de plantillas de servidor cuyo objetivo es de crear una imagen de un servidor que captura un "instantánea" del sistema operativo (SO), el software, los archivos y todos los demás detalles relevantes. En lugar de lanzar varios servidores y configurarlos uno por uno ejecutando el mismo código.

- **Herramientas de orquestación**

  Las herramientas de plantillas de servidor de acuerdo en [48] se expone que son excelentes para

crear máquinas virtuales y contenedores, pero ¿qué hacer con ellas? Para la mayoría de los casos de uso del mundo real, necesita una forma de hacer lo siguiente: Implemente máquinas virtuales y contenedores, haciendo un uso eficiente de su hardware.

- **Herramientas de aprovisionamiento**

  Las herramientas de administración de configuración, plantillas de servidor y orquestación definen el código que se ejecuta en cada servidor, las herramientas de aprovisionamiento como Terraform, Ansible ,etc. son responsables de crear sus propios servidores. De hecho, puedes usar herramientas de aprovisionamiento no solo para crear servidores, sino también bases de datos, cachés, balanceadores de carga, colas, monitoreo, configuraciones de subred, configuración de firewall, reglas de enrutamiento, Secure Sockets Certificados de capa (SSL) y casi todos los demás aspectos de su infraestructura de acuerdo a lo que señala el autor en [53].

3.5 Herramientas para el despliegue de Infraestructura como código

Las herramientas de infraestructura como código crean oportunidades para trabajar en una variedad de formas para ayudarlo a entregar cambios con mayor frecuencia, rapidez y confiabilidad, mejorando la calidad general de su sistema.

3.5.1 Vagrant

Según lo que plantea el autor en [54] es una herramienta de administración de máquinas virtuales para desarrolladores que a menudo se usa para simular un conjunto de servidores que se parecen mucho al entorno de producción en el que se implementará una aplicación. Entre otras cosas, Vagrant facilita la ejecución del software en estaciones de trabajo basadas en Linux, macOS y Windows.

Vagrant ofrece una interfaz de línea de comandos para administrar las máquinas virtuales. Es herramienta es compatible con diferentes plataformas de virtualización como: Hyper-V, VMWare y Virtual Box,. Además, brinda compatibilidad con otras herramientas de configuración de software más populares, incluidas Ansible, Chef, Puppet y Salt. Debe considerarse una herramienta de uso general para trabajar con máquinas virtuales. Esta herramienta debe estar ejecutar en la maquina del administrador como en los nodos gestionados de cuerdo a lo que expone en [54].

3.5.2 Chef

Como expresa el autor en [55] Chef fue desarrollada por la comunidad Opscode en año 2008. Esta herramienta de gestión de configuración se utiliza para describir y administrar la infraestructura a través de código. Chef automatiza el proceso de despliegue de instancias con una configuración mínima. Esta herramienta trabaja con un modelo Maestro-Cliente en donde se requiere un nodo maestro. Chef es compatible con unos diferentes entornos como: Linux, Windows 7 y Windows Server. Chef fue desarrollada en Ruby y Erlang. Y se integra con plataformas en nube como Amazon, GoogleCloud, Azure, OpenStack. Para asegurarse que esta herramienta funcione, debe estar instalada tanto en el host del administrador como en los nodos gestionados. Los agentes pueden ser instalados desde el cliente usando knife que usa SSH para desplegar.

3.5.3 Terraform

Según su web oficial manifiesta [56] que "Terraform permite que la infraestructura se exprese como código. Utiliza un lenguaje simple y legible HCL (lenguaje de configuración de HashiCorp). Lee archivos de configuración y proporciona un plan de ejecución de cambios, que puede revisarse por seguridad y luego aplicarse y aprovisionarse". Los proveedores de infraestructura en la nube permiten que Terraform administre una amplia gama de recursos, incluidos IaaS, PaaS, SaaS y

servicios de hardware.

Terraform es una herramienta de código abierto creada por HashiCorp e introducida al mercado en 2014 como software IaC (escribir código) que se utiliza principalmente para construir, cambiar y administrar la infraestructura de manera segura y eficiente.

Terraform desarrollada en el lenguaje de programación Go. El código se compila en un binario dependiendo del sistema operativo base del administrador. El archivo binario se utiliza para desplegar, cambiar y versionar infraestructura de forma segura y eficiente en un servidor, el binario hace peticiones a las API de los proveedores de nube como AWS, GCP, OpenStack, UPCLOUD, etc. Esto significa que Terraform es independiente de la nube del proveedor puede aprovisionar o administrar su infraestructura en cualquier plataforma en la nube como se visualiza en la Figure 3.3 . Lo que significa que tiene una gran ventaja frente a otras herramientas de IaC. También proporciona soporte para otros tipos de proveedores de soluciones de tipo PAAS o SAAS. La infraestructura que administra Terraform se puede alojar en nubes públicas como las antes mencionadas, o en las instalaciones en nubes privadas como VMware vSphere, OpenStack o CloudStack. Terraform maneja IaC, por lo que nunca tendrá que preocuparse de que su infraestructura se aleje de la configuración deseada, como lo propone el autor en [57].

Interfaz de Línea de Comandos de Terraform (o Terraform CLI por sus siglas en inglés) citando a [58] plantea que la herramienta que ejecutan comandos como terraform init, terraform plan, terraform apply, etc., estos comandos son compatibles con todas las nubes. Esta herramienta es imprescindible ya que analiza el código y lo traduce en una serie de llamadas a las API de los diversos proveedores en la nube. Este fragmento indica a Terraform que realice llamadas API a AWS para implementar un servidor como se muestra en la Figure 3.3. Terraform utiliza principalmente archivos que terminan en .tf o .tf.json que contienen información detallada sobre

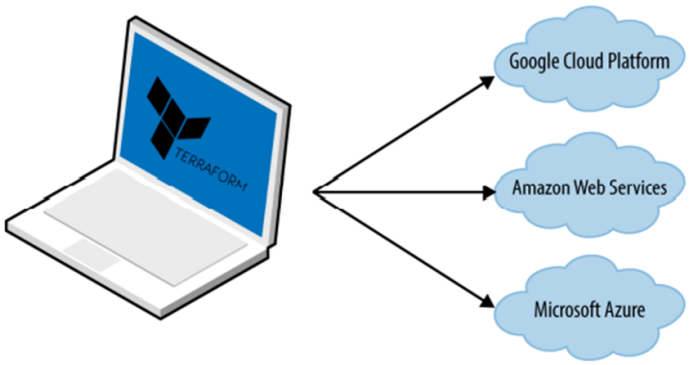


Figura 3.3

Terraform es un binario que traduce el contenido de sus configuraciones en llamadas API a proveedores de la nube [5].

qué componentes de infraestructura se requieren para ejecutar o administrar los componentes de la infraestructura como: servidores, bases de datos, balanceadores de carga, red topología, etc. Terraform desarrolla un plan de acciones, que describe la secuencia de pasos que se va a realizar para lograr el estado deseado y luego lo ejecuta para construir la infraestructura descrita. En la Figure 3.4 se visualiza algunos proveedores que ofrecen la API para Terraform.

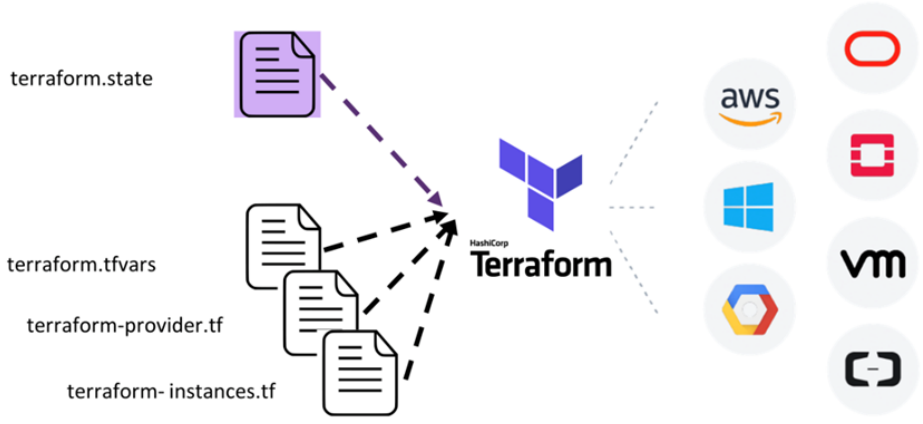


Figura 3.4

Terraform es un binario que traduce el contenido de sus configuraciones en llamadas API a proveedores de la nube [5].

Una configuración de Terraform se compone de uno o más archivos en un directorio, binarios

de proveedor, archivos de plan y archivos de estado una vez que Terraform ha ejecutado la configuración. Terraform es simple y fácil de escribir, y la sintaxis HLC utilizada en los archivos de configuración es muy poderosa, lo permite hacer referencia a variables, atributos, etc. Y tener bucles (count, for each y for) que hacen que algunas cosas sean más fáciles y otras un poco difíciles y alienta a los desarrolladores y administradores a comenzar a explorar Terraform.

- **Archivo de configuración (archivos *.tf):** Aquí declaramos el proveedor y los recursos que se implementarán junto con el tipo de recurso y todas las configuraciones específicas de los recursos
- **Archivo de declaración de variables (variables.tf o variables.tf.json):** Aquí declaramos las variables de entrada necesarias para aprovisionar recursos
- **Archivos de definición de variables (terraform.tfvars):** Aquí asignamos valores a las variables de entrada
- **Archivo de estado (terraform.tfstate):** se crea un archivo de estado una vez después de ejecutar Terraform. Almacena el estado sobre nuestra infraestructura administrada.

3.5.3.1 Ciclo de vida de Terraform.

El ciclo de vida de Terraform consiste en iniciar, planificar, aplicar y destruir como se muestra en la Figure 3.5.

Figura 3.5

Ciclo de vida de Terraform [5].

1. **Terraform init:** Inicializa el entorno terraform (local). Por lo general, se ejecuta solo una

vez por sesión.

2. **Terraform plan:** Compara el estado Terraform con el estado tal cual en la nube, construye y muestra un plan de ejecución. Esto no cambia la implementación (solo lectura).
3. **Terraform apply:** Ejecuta el plan. Esto potencialmente cambia la implementación.
4. **Terraform destroy:** Elimina todos los recursos que se rigen por este entorno terraform específico.

3.5.3.2 Arquitectura de Terraform.

Como puede observar en la Figure 3.6 la arquitectura de Terraform tiene dos componentes clave en los que el funcionamiento depende de los complementos Terraform Core y Terraform.

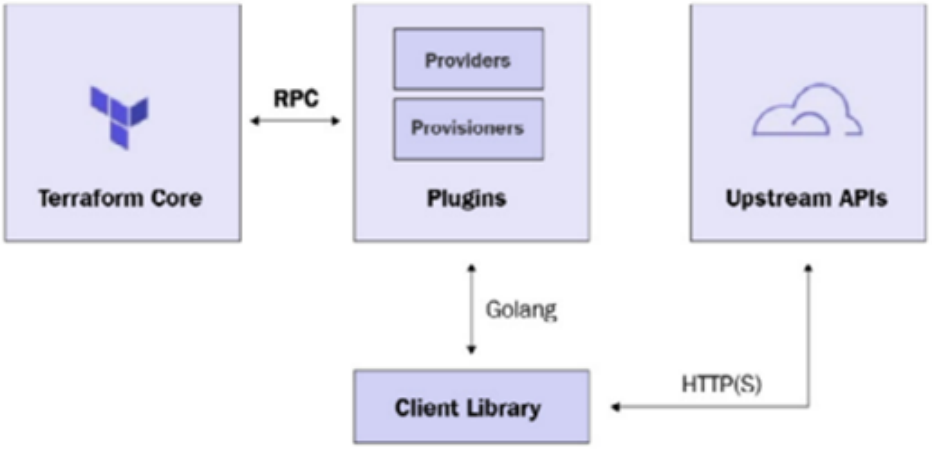


Figura 3.6

Arquitectura de Terraform [5].

- **Terraform Core** el autor en [56] señala que se utiliza el control remoto de Procedimiento de llamadas (RPC) para comunicarse con los complementos de Terraform y ofrece múltiples formas de descubrir y cargar complementos para usar. Terraform Core es un binario compilado estáticamente escrito

en el lenguaje de programación Go. Utiliza RPC para comunicarse con los complementos de Terraform y ofrece múltiples formas de descubrir y cargar complementos para su uso. Las responsabilidades de Terraform Core son las siguientes: Lectura e interpolación de archivos y módulos de configuración, Gestión de estado de recursos, Construcción de gráficos de recursos, Ejecución del plan Comunicación con plugins vía RPC.

- **Terraform Plugins** Son complementos escritos en el lenguaje de programación Go, estos binarios son ejecutables que obtienen invocado por Terraform Core a través de RPC. Cada complemento expone una implementación para un servicio específico, como AWS, GCP. Terraform tiene muchos integrados aprovisionadores, mientras que los proveedores se agregan dinámicamente cuando sea necesario como lo define el autor en [59].

3.5.3.3 Componentes de Terraform.

- **Provisioners** Los provisioners de Terraform según el autor en [57] describe que se utilizan para ejecutar script en la máquina local o en una máquina remota cuando ejecuta Terraform, normalmente para hacer lo siguiente: trabajo de arranque, gestión de configuración o limpieza. Los provisioners pueden modelar acciones específicas en la máquina para preparar servidores u otros objetos de infraestructura para el servicio.
- **Terraform State:** Terraform registra en archivos el estado de la infraestructura y las configuraciones. Los estados son usados para encontrar recursos en una configuración, además para rastrear metadatos, y mejorar el rendimiento de grandes infraestructuras como lo manifiesta el autor en [56]. El estado se almacena en un archivos local llamado “terraform.tfstate”.
- **Recursos (Resource)** En [59] el autor indica que los proveedores de la nube proporcionan

varios servicios en sus plataformas, se hace referencia como recursos en Terraform. Estos recursos se describen como bloques de construcción para crear instancias como máquinas virtuales, redes, bases de datos, etc. Estos recursos se describen en archivos de configuración que Terraform administra.

- **Provider** Es un complemento de Terraform que interactúa con las diversas API necesarias para crear, actualizar y eliminar varios recursos.
- **Módulos** Un módulo como señala el autor en [59] es un contenedor en el que se definen todos los recursos para ser utilizados juntos. Cada código de Terraform tiene un módulo esencial, llamado módulo raíz, que contiene todos los recursos en el archivo de configuración .tf.

3.5.3.4 Características de Terraform.

De acuerdo con los autores en [60, 59] indican las siguientes características:

- **Infraestructura como código:** la IaC se define en archivos con configuraciones avanzadas, que permite lanzar y versionar la infraestructura como otra pieza de software.
- **Planes de Ejecución:** Terraform define un plan de secuencia. En donde se muestra todos los pasos necesarios para ejecutar para la creación y liberar la infraestructura como código.
- **Gráfico de recursos:** Terraform crea un gráfico de recursos con toda la información de las dependencias de la infraestructura. Este gráfico ilustra las dependencias entre máquinas virtuales, subredes, instancias, etc.
- **Automatización de cambios:** Terraform permite aplicar cambios a la infraestructura casi sin la necesidad de interacción humana. Los gráficos de recursos permiten un fácil seguimiento de las dependencias en toda la infraestructura. Esto ayuda a reducir el error humano al mostrar cada etapa con precisión.

3.5.4 Ansible

Ansible es una herramienta libre que permite automatizar el aprovisionamiento de software, además de la gestión de configuraciones y el despliegue de aplicaciones, como nos indica el autor en [61] Ansible "Permite trabajar con los proveedores Cloud para automatizar, tanto el despliegue de instancias, sino también para administrar los diferentes componentes en la nube, como redes, grupos de seguridad, claves públicas y direcciones IP públicas".

En Ansible, según lo descrito en [61] las configuraciones son realizadas en archivos YAML que describen las acciones que se van a realizar a los nodos que están siendo administrados. Estas acciones se ejecutan a través de del protocolo SSH, por lo que no es necesario instalar ningún software adicional en los equipos. Como expresa en [62] define que "dando como resultado una plataforma fácil de usar por medio de los scripts que son una forma muy común de realizar tareas de configuración". Esta herramienta permite instalar aplicaciones, orquestar servicios y tareas más avanzadas como entrega continua (continuous deployment, CD) e integración continua (Continuous Integration, CI).

3.5.4.1 Arquitectura de Ansible.

Hay dos tipos de máquinas en la arquitectura Ansible como lo manifiesta el autor en [63] que son el equipo de control y los equipo gestionados. La herramienta Ansible está instalado en el equipo de control y todos sus componentes se almacenan en él. Los equipos administrados se enumeran en un inventario de host, que es un archivo de texto en el equipo de control que incluye una lista de nombres de equipos administrados o direcciones IP de cada equipo administrado, todos estos componentes se observan en la Figure 3.7.

Los administradores desde el equipo de control inician Ansible, proporcionándole un playbook y

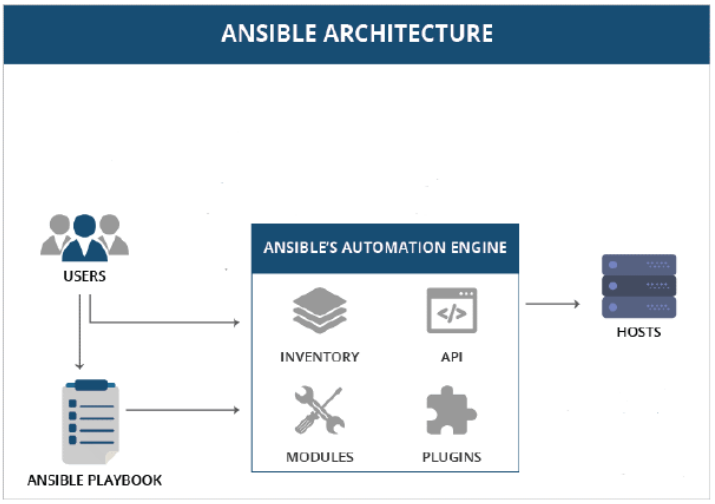


Figura 3.7

Ansible Arquitectura [64].

los equipos a ser administrados. Ansible utiliza SSH como un transporte de red para gestionar los equipos. Ansible requiere tener instalado Python en los equipos remotos en los que se vaya a ejecutar para poder utilizar. Los módulos descritos en un playbook se copian en los equipos remotos ejecutándose en orden, con los comandos específicos.

Los administradores pueden redactar sus propios módulos personalizados según las necesitan específicas, los módulos principales incluidos con Ansible pueden manejar la mayoría de las tareas de administración del sistema.

La Table 3.1 se enumera y se describe los componentes de Ansible que se mantienen en el nodo de control.

3.5.4.2 Inventario (Inventory).

Como expresa en [65], un inventario es el componente más básico de la arquitectura de Ansible. Define los equipos que se deben administrar en su infraestructura. Al mismo tiempo, usando una

Tabla 3.1

Componentes del nodo de control Ansible [61].

Elemento	**Descripción**
Playbook	Son archivos con formato YAML que describen las acciones a realizar en los nodos gestionados.
Play	Lista de tareas (Tasks) a realizar en los nodos especificados por el Playbook.
Host Inventario	Son archivos que contiene una lista estática o dinámica, con los nodos a administrar y la información necesaria de cada uno de ellos.
Nodo	Elemento para administrar: servidor, router u otro elemento administrable.
Tarea	La definición de una acción a realizar en un nodo.
Módulo	Unidades de trabajo que se copian al nodo administrado y ejecuta la tarea requerida. Ansible contiene más de 1500 módulos.
Variable	Valores especificados estática o dinámicamente a ser utilizados por módulos y plantillas.
Custom modules	Los usuarios pueden ampliar la funcionalidad de Ansible escribiendo sus propios módulos y agregándolos a la biblioteca de Ansible.

lista o grupo de listas después de definir el inventario. Ansible ejecuta sus tareas en todos los equipos enumerados en un archivo de inventario. Los inventarios se pueden definir en dos formas. Un inventario de equipos estático y un inventario dinámico como describe el autor en [65].

3.5.4.3 Inventario de equipos estático.

Teniendo en cuenta a [61] considera que un inventario estático de equipo (equipo), puede contener una lista con los nombres de los equipos que pueden o no pertenecer a un grupo. Como observa en la Figure 3.8 un ejemplo de un archivo de inventario estático que describe un solo equipo. La definición de un grupo se hace entre corchetes y la lista de nodos a continuación pertenecerá al mismo.

```
$ cat hosts
example.com
web001
db001
50.16.4.125
```

Figura 3.8

Ejemplo Inventario de un host estático [65].

3.5.4.4 Inventario dinámico.

La información del inventario del equipo también se puede generar de forma dinámica. En el entorno de TI moderno, las fuentes de los nodos pueden provenir de diferentes proveedores, como una cloud privada (AWS, Azure, GCP, DigitalOcean), también de una infraestructura privada (OpenStack), contenedores. Estos sistemas tienen su lista de nodos y Ansible se integra con dicho inventario dinámico externo, según lo planteado en [66]. Los Inventarios dinámicos son una lista de nodos que se administrarán y cambiará durante este período de tiempo, no es necesario editarlo manualmente en el archivo, como describe el autor en [61].

3.5.4.5 Playbooks.

Los Playbooks es el término que Ansible usa para un script de administración de configuración, el cual define una lista de tareas, variables de usuario y detallando las configuraciones que pueda ser necesario que se pueden ejecutar en los equipos definidos en un inventario.

Los Playbooks según lo definido en [67] es el término que Ansible usa para un script de administración de configuración, el cual define una lista de tareas, variables de usuario y detallando las configuraciones que pueda ser necesario que se pueden ejecutar en los equipos definidos en un inventario. En [66] el autor afirma que los Playbooks son scripts escrito en formato YAML como se observa en la Figure 3.9. En un Playbook le da un sabor programable a Ansible por

permitiendo declaraciones variables, bucles y condicionales.

```
---
- name: Conectar a los servidores Debian
  hosts: debian
  tasks:
    - name: Copiar fichero a los servidores
      copy: src=sources.list dest=/etc/apt/sources.list
- name: Conectar a los servidores CentOS
  hosts: centos
  tasks:
    - name: Copiar fichero a los servidores
      copy: src=cntos.repo dest=/etc/yum.repos.d/centos.repo
...
```

Figura 3.9

Ejemplo de un Playbook. [61]

Elementos de un Playbook

Como de observa en la Figure 3.10 un playbook está conformando por diferentes elementos, así como se relaciona cada elemento con los demás [68].

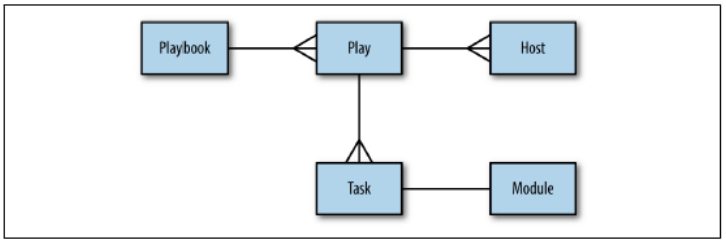


Figura 3.10

Elementos de un Playbook [67].

- **Playbook:** Son archivo de texto con formato YAML que contiene uno o más plays. que se ejecutará en un servidor en particular o un conjunto de servidores.
- **Módulos** Los módulos son scripts empaquetados con Ansible y están incluidos en nodo master. Ansible está compuesto por múltiples módulos, que se encargan de administrar sistemas, dispositivos, usuarios y una gran cantidad de elementos diferentes. Cada tarea del script está asociada a un módulo. Ansible proporciona más de 200 módulos, como Sistema, Red, Base

de datos y Archivos, que puede usar y manejar fácilmente en su infraestructura (Learning Ansible Use Ansible to configure your systems, deploy software, and orchestrate advanced IT tasks. Madhurranjan Mohaan - Ramesh Raithatha).

- **Play:** Un plays son tareas que se deben ejecutar en los hosts administrados.
- **Hosts** Enumera el host o el grupo de hosts donde Ansible ejecutara las tareas.
- **Tasks:** Una tarea representará la ejecución secuencial de una acción con argumentos, ya sean opcionales u obligatorios. Cada acción se declara en una lista de tareas especificando el nombre del módulo además de parámetros opcionales.

3.5.4.6 Módulos.

Los módulos según que expuesto en [69] son scripts empaquetados con Ansible y están incluidos en el anfitrión. Ansible está compuesto por múltiples módulos, que se encargan de administrar sistemas, dispositivos, usuarios y una gran cantidad de elementos diferentes. Cada tarea del script está asociada a un módulo. Ansible proporciona más de 200 módulos, como Sistema, Red, Base de datos y Archivos, que puede usar y manejar fácilmente en su infraestructura.

3.5.4.7 FACTS Hechos (Variables derivadas de la información del sistema).

La primera tarea predeterminada cuando se ejecuta un Playbook es la GATHERING FACTS que recopila información como lo define el autor en [69]. Ansible se conecta al host y lo consulta para obtener metadatos útiles como: IP de host, tipo de CPU, espacio en disco, información del sistema operativo e interfaz de red y más. Esta información se almacena en variables sobre la máquina en el formulario de variables; Estas variables se pueden utilizar para cambiar ciertas tareas cuando se ejecutan, o para cambiar cierta información utilizada en Archivos de configuración como lo expone se lo expone en [67].

3.5.4.8 Variables.

Las variables en [70] el autor describe que se utilizan para almacenar valores que luego se pueden utilizar en los Playbooks. Las variables se pueden establecer por medio de nombres que consisten en una cadena que debe comenzar con una letra y que solo puede contener letras, números y guiones bajos; En un Playbook se establecen mediante la palabra reservada "vars:" Esta clave toma un par clave-valor, donde la clave es el nombre de la variable y el valor es el valor real de la variable.

Ahora veremos cómo establecer una variable en un libro de jugadas usando el ejemplo anterior. Esto se demuestra en la siguiente Figure 3.11:

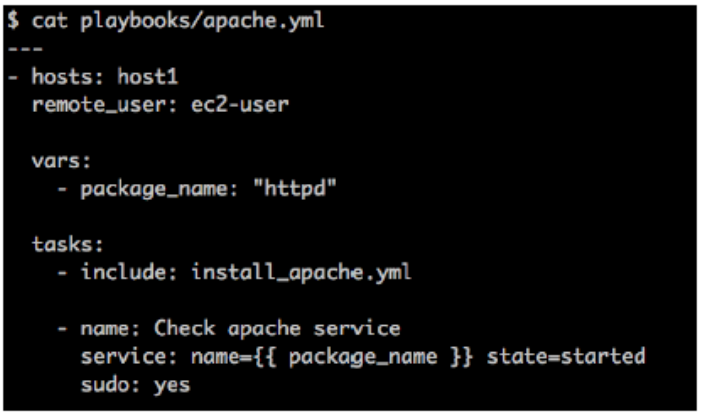


Figura 3.11

Ejemplo Playbook [70].

3.5.4.9 Roles.

Los roles proporcionan un mecanismo para dividir una de colecciones de variables, tareas, archivos, plantillas y módulos totalmente independientes o interdependientes. Cada rol se limita típicamente a un tema en particular o al resultado final deseado, con todos los pasos necesarios para alcanzar ese resultado, ya sea dentro del rol mismo o en otros roles listados como

dependencias según lo descrito en [65].

3.6 Docker

Según el autor en [71] se describe que Docker no es más que un conjunto de plataformas como servicio (PaaS) que proporciona virtualización a nivel de sistema operativo para entregar el software en contenedores. El contenedor comparte el kernel del equipo y por lo tanto, no depende o no necesita ningún invitado sistema operativo. Los desarrolladores y los administradores de sistemas pueden usar la herramienta docker para implementar fácilmente sus aplicaciones en los contenedores y luego pueden ejecutar esas aplicaciones en el sistema operativo del equipo.

Docker es una herramienta Open Source desarrollada en el lenguaje de programación Go que permite aislar y encapsular los requerimientos necesarios de las aplicaciones así ganando una gran popularidad entre los desarrolladores. Es Decir, Docker facilita la ejecución de cualquier aplicación sin tener que preocuparse por los requerimientos necesarios como tales librerías, archivos del sistema y otras dependencias ya que los empaqueta dentro de un contenedor único que se define en un solo archivo; De esta forma, garantiza que los procesos dentro de un contendedor tengan el mismo comportamiento y el correcto independiente del host, como propone el autor en [72].

Reduciendo significativamente la necesidad de un código de gestión de configuración complejo. Aunque Docker puede mejorar significativamente la capacidad de una organización para administrar aplicaciones y sus dependencias, no reemplaza directamente la administración de configuración más tradicional.

3.6.1 Contenedores Docker

Un contenedor de Docker como definió en [73] es una instancia en ejecución de una imagen específica. Se pueden crear uno o más contenedores de una imagen ya definida para tener varias instancias en ejecución de la misma aplicación, como se muestra en la Figure 3.12.

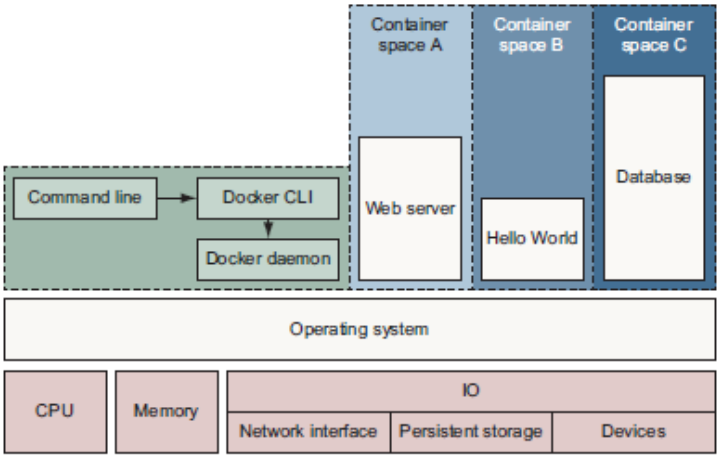


Figura 3.12

Docker ejecutando tres contenedores en un sistema Linux [53].

Los contenedores son autónomos, contienen todo lo que una instancia de aplicación necesita para ejecutarse (sistema operativo, dependencias o lo que sea) y se pueden ejecutar, iniciar, detener, mover y eliminar. En realidad, un contenedor es solo un proceso del sistema operativo que lo utiliza para ejecutar aplicaciones. El servidor Docker asigna algunos recursos de equipo como (memoria RAM, CPU, Almacenamiento, red). Una vez inicializada, una aplicación específica se ejecuta dentro del contenedor como cualquier otra aplicación en el sistema equipo, como propone el autor en [74].

3.6.2 Dockerfile

Los Dockerfile teniendo en cuenta a [72] describe que se usan para cómo se debe ver un contenedor en el momento de la compilación, pero no administran el estado continuo del contenedor y no se pueden usar para administrar el sistema host de Docker.

Pues bien, un Dockerfile es un archivo de secuencia de instrucciones y configuraciones necesarias donde se le indica a Docker que acciones debe realizar para generar la imagen desde cero (o basándose en una ya existente), según lo manifestado en [75].

Además, un Dockerfile en [74] señala que es un archivo de texto simple sin ninguna extensión con el nombre fijo Dockerfile. Un Dockerfile cuenta con multitud de directivas que permiten personalizar y configurar cómo va a funcionar la imagen de Docker. Contiene una serie de comandos (que pueden variar de un archivo a otro según los requisitos) que se ejecutan para crear la imagen de Docker. Esto indica debemos utilizar un Dockerfile para personalizar una imagen según nuestros requisitos específicos.

Algunos de los comandos utilizados en el fichero Dockerfile para su configuración, son los siguientes expresados en la Table 3.2:

3.6.3 Arquitectura de Docker

Docker utiliza una arquitectura cliente servidor. El cliente de Docker se comunica con el demonio de Docker donde envía una solicitud que realiza el trabajo de crear, ejecutar y desplegar los contenedores. Ambos pueden ejecutarse en el mismo sistema, o un cliente puede conectarse a un demonio Docker remoto. El cliente Docker y el daemon de Docker se comunican a través de un socket o mediante la API RESTfull.

Tabla 3.2

Tabla de comandos.

Elemento	Descripción
FROM	define una imagen base de partida sobre la que construir
ADD	copia ficheros y directorios dentro de la imagen del contenedor. Además, acepta URLs como parámetro y las descarga directamente dentro
RUN	añade capas a la imagen base, instalando aplicaciones, librerías y componentes
CMD	especifica qué comandos se deben ejecutar al iniciarse el contenedor. Es importante tener en cuenta que solo puede existir una instrucción CMD en el fichero
ENTRYPOINT	es la instrucción "principal" de un Dockerfile. Especifica el punto de entrada de un contenedor que se quiere que funcione como un ejecutable. Por ejemplo, el siguiente comando "docker run -it –rm -p 80:80 nginx" ejecuta un servidor NGINX de forma interactiva, publica el puerto 80 y, cuando finaliza su ejecución, se elimina
ENV	define las variables del entorno del contenedor que se usarán en tiempo de ejecución
EXPOSE	define en qué puertos la aplicación estará escuchando.
USER	especifica el UID (o el nombre del usuario) que se usará internamente en el contenedor para ejecutar las aplicaciones
VOLUME	define volúmenes de datos a crear o un punto de montaje del contenedor en tiempo de ejecución.
WORKDIR	especifica la ubicación en donde se ejecutará el comando en tiempo de ejecución

3.6.4 Imágenes (Docker)

Según lo descrito en [77] considera que una imagen es una clase definida de contenedor que podríamos crear. Un contenedor es entonces una instancia u objeto de esa clase.

Una imagen de Docker son archivos solo de lectura en la cual incluye los componentes necesarios para ejecutar la aplicación como el sistema operativo base (por lo general cualquier distribución de linux) dónde correrá nuestra aplicación, la aplicación en sí, las bibliotecas necesarias, variables de entorno y archivos de configuración que se ejecuta dentro de un contenedor, de acuerdo con lo que expresa en [76].

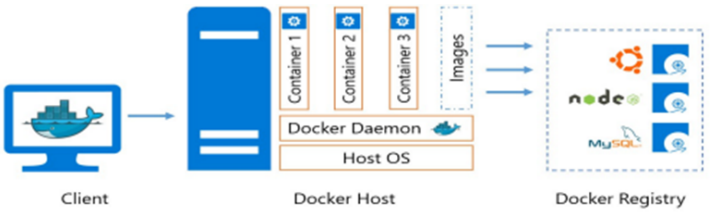


Figura 3.13

Representación gráfica de la Arquitectura de Docker [76].

Como plantea en [78] se describe que se puede crear cualquier número de contenedores a partir de una imagen. Sin embargo, cuando se hace esto, los contenedores lanzados desde la misma imagen no compartirán los cambios de su sistema de archivos. Una sola imagen de Docker se puede replicar en varios hosts. Las imágenes suelen tener nombres y etiquetas. Esta etiqueta se usa a menudo para identificar una versión específica de una imagen.

Imágenes de Docker y contenedores de Docker Las imágenes de Docker son de solo lectura. Esto no significa que no podamos hacer cambios en la imagen, pero una vez que lo hacemos, tenemos un nuevo tipo de imagen. También puede crear una imagen desde cero y, después de instalar todas las dependencias necesarias, puede agregar su aplicación o puede descargar imágenes creadas por otros desde el repositorio de Docker Hub, como señala el autor en [77].

3.6.5 Registro De Docker

El registro de Docker según lo indica en [73] señala que el sistema de almacenamiento y distribución de imágenes de Docker. Está organizado en repositorios de Docker, donde un repositorio contiene todas las versiones de una imagen específica, y las imágenes se pueden cargar y descargar desde una misma ubicación. Hay dos tipos de repositorios: públicos y privados.

En el registro de Docker, un repositorio de Docker contiene todas las imágenes con el mismo nombre, pero con diferentes etiquetas (las etiquetas se usan para identificar diferentes versiones

de una imagen), tal como expone en [79].

3.6.6 Doker Hub

Plataforma SaaS para compartir y gestionar contenedores Docker. Docker Hub es la fuente espejo oficial para las principales tecnologías de código abierto que se utilizan, incluidas Jupyter, PostgreSQL, Redis y Mongo DB, como manifiesta el autor en [77]. Todas las imágenes en este repositorio se pueden compartir en áreas preferidas como pública o privada.

3.6.7 Docker Compose

En [80] el autor expone que Docker Compose es un proyecto open source que se usa para ejecutar múltiples contenedores como un solo servicio de una manera sencilla, además de servir para gestionar entornos de desarrollo y de pruebas o para procesos de integración continua.Por ejemplo, una aplicación requiere contenedores NGNIX y MySQL, podría cree un archivo que iniciaría ambos contenedores como un servicio (docker compose) o iniciar cada uno por separado (docker run).Todos los servicios deben definirse en formato YAML Docker Compose Ellos a menudo Resuelve tus problemas sin necesidad de soluciones más complejas.

Docker Compose es una herramienta para ensamblar microservicios y aplicaciones que consta de múltiples contenedores. Estos microservicios y aplicaciones están diseñados utilizando un único YAML y se puede construir, ejecutar y escalar, cada uno a través de un solo comando.

3.7 Herramientas para el versionado de código

Los sistemas de control de versiones (SCV por sus siglas en español) de acuerdo con [81] expone que es una herramienta esencial para manejar la gestión de múltiples versiones de un proyecto

de software. Para administrar una versión, cada cambio (adición, edición o eliminación) a los archivos en un proyecto debe ser rastreado.

3.7.1 Git

Git como se indica en [82] indica que es un sistema de control de versiones diseñado para la gestión eficiente de flujos de trabajo distribuido no lineales . Git es multiplataforma, operar en entornos como Linux, MacOS y Windows. Git ha conquistado desde entonces la mayor parte del mundo de código abierto y también es utilizado por muchas organizaciones para sus proyectos privados/propietarios.

Git esta implementado como un espacio local para el historial de versiones de un proyecto. Git también es una gran herramienta para desarrolladores de software que trabajan en equipos escribiendo nuevo código fuente y cambiando el que ya existe. En donde el equipo revisar las diferencias entre esas versiones y gestionar el historial de cambios de un archivo. Una de las principales características de Git es su sistema de ramificación. Una rama es una copia de un proyecto. en el que puede trabajar sin alterar el repositorio. Por lo general, se crea una rama para crear o probar una nueva función y vuelva a fusionar esa rama cuando esté satisfecho con el trabajo.

A continuación, vamos a describir los principales comandos de Git (ver Figure 3.14).

3.7.2 GitHub

Con base en [83], se define que es un servicio de almacenamiento donde se alojan los repositorios git remotos. Github es actualmente propiedad de Microsoft, en este servicio se puede observar por una interfaz web, las ramas que existen en el repositorio y sus archivos que contiende el

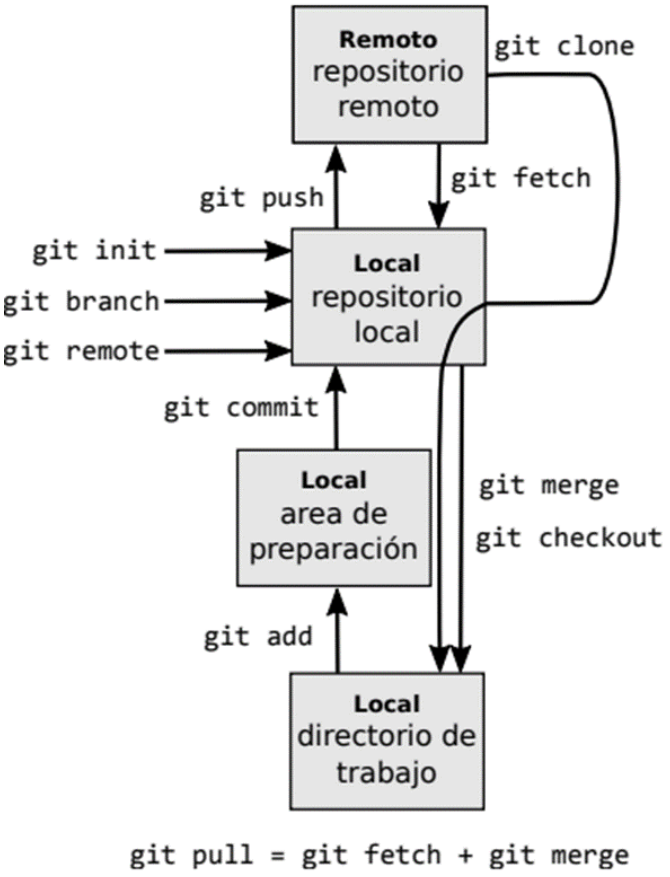


Figura 3.14

Principales comandos de Git [82].

repositorio, además de ver los cambios que se han hecho por cada uno de los que colaboradores del proyecto.

3.7.3 GitLab

Es otro sistema de alojamiento de repositorios Git, pero de código libre el cual también facilita el desarrollo colaborativo entre varios desarrolladores se pueden configurar los proyectos para que sean públicos de libre acceso para todos o privado con diferentes permisos para cada uno de los miembros. GitLab que permiten instalar en servidores propios un servicio de alojamiento de repositorios remotos con herramientas similares a las que ofrece GitHub. Git está disponible de

Tabla 3.3

Tabla de comandos de Git.

Elemento	**Descripción**
git init [project-name]	Crea un nuevo repositorio local con el nombre especificado
git clone [url]	Descarga un proyecto y toda su historia de versión
git status	Enumera todos los archivos nuevos o modificados que se deben confirmar
git add [file]	Toma una instantánea del archivo para preparar la versión
git commit -m "[descriptive message]"	Registra las instantáneas del archivo permanentemente en el historial de versión
git branch	Enumera todas las ramas en el repositorio actual
git branch [branch name]	El comando git branch permite crear y destruir ramas. Por defecto se trabaja en la rama denominada master.
git checkout [branch name]	Cambia a la rama especificada y actualiza el directorio activo Para cambiar de rama en el directorio de trabajo se utiliza el comando git checkout.
git merge [branch]	Combina el historial de la rama especificada con la rama actual
git pull	Descarga el historial del marcador e incorpora cambios Para actualizar el repositorio con los cambios de un repositorio local se utiliza el comando git pull.
git merge [bookmark][branch]	Combina la rama del marcador con la rama local actual
git push [alias] [branch]	Carga todos los commits de la rama local al GitHub Si se desean enviar los cambios a un repositorio remoto se utiliza el comando git push indicando la rama cuyos cambios se quieren enviar.
git remote	Previamente los repositorios remotos se han configurado utilizando el comando

forma gratuita como software de código abierto y es considerado uno de los sistemas de control de versiones más usados en general. GitLab es una de las alternativas a GitHub más populares (cuando Microsoft se hizo con GitHub en 2018, muchos usuarios se cambiaron a GitLab), como describe el autor en [84].

Capítulo 4

Diseño de la arquitectura para el aprovisionamiento y desplegué de la nube privada

En esta capitulo se describe la arquitectura general propuesta para la implementación de la nube privada OpenStack y las herramientas para el aprovisionamiento de recursos Terraform y Ansible, además cuenta con las especificaciones técnicas que se utilizara para este ambiente de pruebas.

La nube privada OpenStack es una de las plataformas mas utilizadas de código abierto, en la cual se aprovisionara los recursos como es el almacenamiento, redes, CPU y memoria RAM, etc.

4.1 Arquitectura

La arquitectura consta de los componentes funcional para el aprovisionamiento de la infraestructura y de la nube privada OpenStack

4.1.1 Componentes funcionales:

En la arquitectura se presentan los siguientes componentes:

- **Maquinas virtuales:** estas son instancias generadas por OpenStack con imágenes de tipo qcow2.
- **Redes virtuales:** ofrece funciones de red, como la compartición de archivos, configuración

de direcciones IP, además posibilita la comunicación entre las instancias de la nube privada con redes internas utilizando un mismo direccionamiento y comparte algunas características comunes como el gateway, servicio de DHCP, etc.

- **Routers virtuales:** son dispositivos que permiten conectar con la red externa y entre las subredes de OpenStack.
- **Docker:** este permite encapsular y aislar aplicaciones con todo lo necesario para el correcto funcionamiento, con desplegar los servicios de una manera mas rápida y funcional.
- **Terraform:** es la herramienta de IaC, se utiliza para la construcción de entornos de desplegué de infraestructura así como para administración.
- **Ansible:** es la herramienta con la que se gestionara la configuración de las instancias.

4.1.2 Arquitectura propuesta

La Figure 4.1 se ilustra la integración de los componentes funcionales, que comienza por el administrador IaC generando el script(.tf) de terraform, en dicho script se define la configuración de Terraform en el cual se define por bloques los recursos como son almacenamiento, red, memoria RAM, CPU, grupos de seguridad, etc. Además se cuenta con Ansible que permite la configuración de las instancias, para ello se genero el script Playbook(.yaml), que cuenta con las configuraciones de inicio para las instancias, los scripts se subirá a la plataforma de versionamiento GitHub para tener un control de las versiones o cambios realizados en la infraestructura. Una vez versionado y aprobado los recursos a desplegar con las herramientas de IaC se procede a aprovisionar los recursos en el proveedor de nube OpenStack.

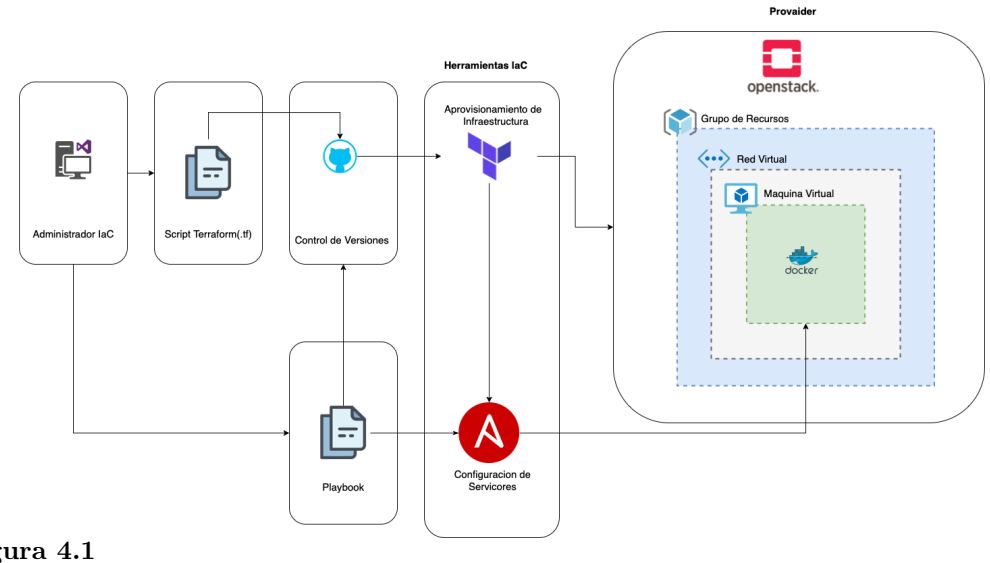


Figura 4.1

Arquitectura de la infraestructura propuesta.

4.1.3 Recursos de Hardware y software utilizados

Para la instalación de OpenStack y las herramientas IaC se utilizo los siguientes recursos de software y hardware los que se muestran en la Table 4.1:

Tabla 4.1

Requerimientos software para entorno de OpenStack y IaC.

Elemento	Nombre	Versión	Sistema Operativo
Herramienta IaC	Terraform	1.0.11	Linux Lite
Herramienta IaC	Ansible	2.0	Linux Lite
Plataforma Cloud	OpenStack	setein	Centos 7

Tabla 4.2

Requerimientos hardware para entorno de OpenStack y IaC.

Elemento	CPU	Procesadores	Memoria RAM	Disco	S.O
OpenStack packstack	Intel Core i7	4 procesador	14 GB	100 GB	Linux
Terraform y Ansible	Intel Core i7	1 procesador	1 GB	20 GB	Linux

4.1.4 Instalación de OpenStack

La instalación de OpenStack se la realizo con Packstack Stein, es una de varias formas de instalar, que su principal objetivo es generar entornos de prueba con OpenStack para SO basados en RPM como Centos. Se instalo sobre una Linux Centos 7, en el Anexo 1 se presenta detallado los pasos realizados para la instalación.

Capítulo 5

Marco metodológico

5.1 Metodología

La metodología propuesta para realizar el aprovisionamiento de la infraestructura , se visualiza en la Figure 5.1 contiene 4 fases: 1) Fase de Diseño del aprovisionamiento, 2) Fase De Codificación, 3) Fase De Planificación , 4) Fase De Aplicación y cuenta con 7 pasos a seguir, los cuales se describe a continuación.

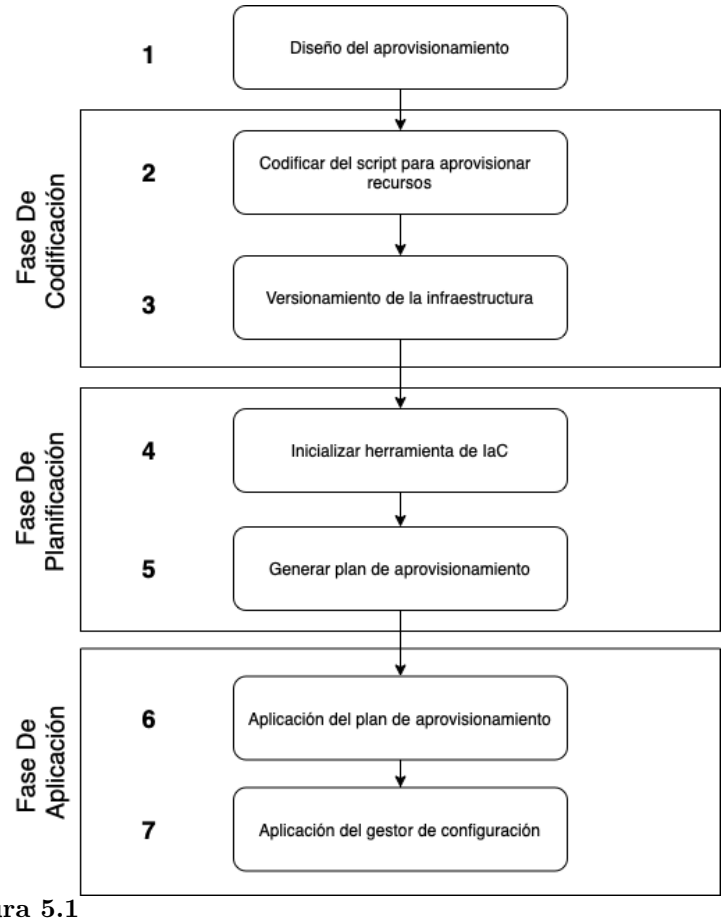


Figura 5.1

Diagrama de la metodología.

- **Fase de Diseño del aprovisionamiento**

  En esta fase se toma la información de los recursos que son necesarios para la infraestructura y tener listo para ser usados.

  - Paso 1 Diseño del aprovisionamiento: En esta paso se tomara los requerimientos como numero de instancias, almacenamiento, memoria RAM, CPU, puertos permitidos, dirección IP de la instancia, con dichos datos datos se procederá a generar una topología para el aprovisionamientos de los recursos. Para el desarrollo de esta fase se utilizo digram.net para diseñar la topología.

- **Fase De Codificación**

  En esta fase se desarrollara los pasos 2 y 3 que se muestran en la Figure 5.1: 2) Codificación script de aprovisionamiento y 3) Versionamiento de la infraestructura que son explicados a continuación:

  - Paso 2 Codificación del script de aprovisionamiento: Se escribe la configuración para aprovisionar los recursos, requerimientos tomados en la fase de diseño. Para este paso se utilizo la herramienta de Visual Studio Code para la redacción de código.
  - Paso 3 Versionamiento de código: Para este paso se utiliza un SVC, para almacenar las distintas versiones de código de la infraestructura. Se utilizo la plataforma GitHub para el almacenar cada versión del código redactado en el paso anterior.

- **Fase De Planificación**

  Esta fase es importante determina los recursos que se va a crear, actualizar o a destruir en la infraestructura, para ello se sigue el paso 4) Inicialización de la herramienta IaC y a continuación el paso 5) Generar el plan de aprovisionamiento que se muestran en la Figure 5.1 que se explica a continuación:

  - Paso 4 Inicialización de herramienta de la herramienta IaC: en este paso se se descar-

gara los pluguis según el proveedor de la cloud seleccionado para conectarse con a su API(Application Programming Interfaces). La herramienta a utilizar es Terraform Cli.

- Paso 5 Generar el plan de aprovisionamiento: En este paso se genera un plan de ejecución que mostrara la secuencia de recursos planificados para el aprovisionamiento de la infraestructura. En este paso se utiliza la nube privada OpenStack.

- **Fase De Aplicación**

  En esta fase se aplica el plan de ejecución para cambiar del estado actual de la infraestructura y dar paso al estado definido en fase 1, se efectúa los siguientes pasos 6) Aplicación del plan aprovisionamiento y 7) Aplicación del gestor de configuración como se muestran en la Figure 5.1. Se detallan a continuación:

  - Paso 6 Aplicación del plan aprovisionamiento: este paso se aplica el plan de aprovisionamiento en la nube OpenStack con la herramienta Terraform Cli.
  - Paso 7 Aplicación del gestor de configuración: una vez desplegado la nueva/modificada infraestructura se procede a ejecutar el script para la configuración de las instancias.Se utilizo la herramienta Ansible.

5.2 Implementación de la IaC

Para la implementación de la metodología propuesta se a generado dos escenarios que serán explicados a continuación:

- **Escenario A:** en este escenario se presenta un desplegué de una maquina virtual con contenedores(Docker) el cual contiene un servidor web(Nginx).
- **Escenario B:** este contara con el desplegué de varias VM con servicios las mismas serán administras y configuras por Ansible.

En la implementación de la metodología se eligió al proveedor de nube OpenStack por ser uno de los nubes privadas mas utilizadas pro grandes empresas. En cuanto a las herramientas IaC se eligió para el desplegué de la infraestructura Terraform la instalación se encuentra en el Anexo 3 y Ansible para la configuración de las instancias la instalación se encuentra en el Anexo 4, estas herramientas ayudan a automatizar y gestionar la infraestructura de manera eficiente.

5.2.1 Implementación del Escenario A

5.2.1.1 Diseño del aprovisionamiento.

A este escenario se le tomara como un entorno para realizar pruebas que se requiera un servidor web. Dentro de este escenario se desplegará un contenedor Docker con un servidor web Ngnix, la misma cuenta con conexión a la red exterior para acceder al servicio web en la Figure 5.2 se presenta el diagrama que de la arquitectura utilizara.

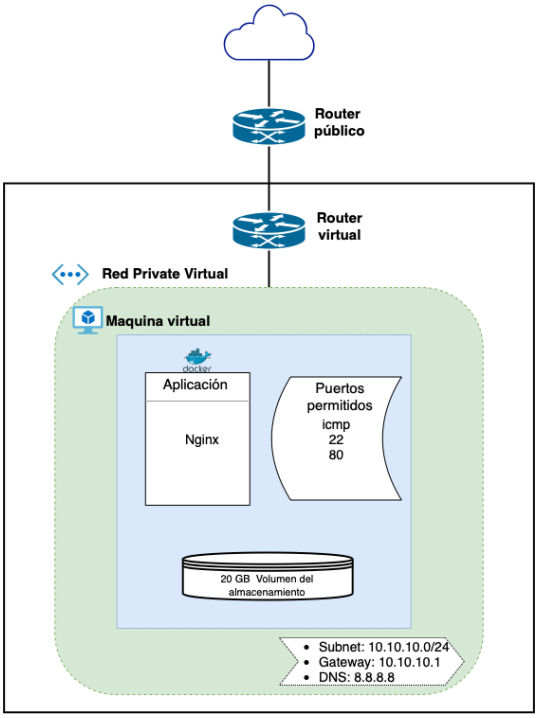


Figura 5.2

Topología del escenario A.

En la topología de la Figure 5.2 se muestra una VM que posee un almacenamiento de 20 GB(Gigabyte), permite solo el acceso de ciertos puertos como el puerto 80(http) para el servicio web, el puerto 22(SSH) y el IMCP para probar conectividad, con respecto al contenedor Docker se descargo desde DockerHub la imagen de Nginx como servidor web, por otro lado se tiene una red virtual privada solo para este escenario la cual se conecta con el router virtual que proporciona una salida así internet. Hay que considerar que OpenStack proporciona dos direccionamientos IPs, la dirección de IP flotante que es la dirección IP publica con la que se accede a los servicios de dicha VM y una dirección IP privada que proporciona la red privada virtual para la comunicación entres VM que pertenecen a la misma red virtual privada.

5.2.1.2 Codificación del scripts de aprovisionamiento.

En este apartado se explica parte de la estructura del proyecto de Terraform, al ser un lenguaje declarativo no importa el orden en que se definan los recursos dentro del archivo, cabe decir que para llevar buenas practicas del proyecto se dividió en varios scripts para llevar un orden, en la Figure 5.3 se presenta la estructura del proyecto de Terraform, mas adelante se explicara cada uno de los scripts que se muestran.

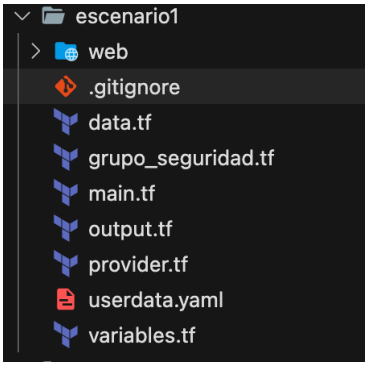


Figura 5.3

Estructura proyecto terraform.

Proveedor.

Se definió el proveedor este proporciona una colección de tipos de recursos, atributos que se van a exportar y además descarga los plugins para conectar con la API remota de OpenStack.

En la Figure 5.4 se muestra el fragmento de código del archivo `provider.tf`, en el primer bloque de código ya se proporciona la documentación de Terraform, en la cual contiene la versión de Terraform y el proveedor de la nube elegido.En el segundo bloque de código contiene las credenciales necesarias para autorizar a la API. Por seguridad las credenciales se encuentra como variables de entorno esto se explica en el Anexo 2, las misma están llamando a las variables que se encuentra en el script de `variables.tf`.

```
# Definir proveedor
terraform {
  required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.35.0"
    }
  }
}

# Configurar el proveedor de OpenStack
provider "openstack" {
  user_name   = var.USUARIO
  tenant_name = var.USUARIO
  password    = var.PASSWORD
  auth_url    = var.AUTH_URL
  region      = "RegionOne"
  domain_name = "Default"
}
```

Figura 5.4

Archivo de configuración del proveedor.

Variables de entrada.

Las variables de entrada se encuentra en el script `variables.tf`, que son parámetros que se toman como valores para recursos que serán llamados desde el `main.tf` o a la hora de ejecución.

Las variables tiene una estructura, `variable` que es el nombre que tomara, una descripción que puedes ser opcional, y `default` que es parámetro, en la Figure 5.5 se muestra las variables de

```
variable "keypair_publica" {
    description = "Nombre de la clave publica en openstack."
    default  = "clave"
}
variable "volumen_tamano" {
    description = "El tamaño del volumen utilizado para crear una instancia de la instancia."
    default = 30
}
variable "red_externa" {
    description = "Red que se conectara con el internet."
    default  = "RedExterna"
}
variable "nombre_red_interna" {
    description = "Red que se conectara con el internet."
    default  = "RedTerraform"
}
variable "subred_interna" {
    description = "Nombre de la subred interna."
    default  = "RedSubTerraform"
}
variable "red_ip4" {
    description = "Red ip4."
    default  = "10.1.0.0/24"
}
variable "router" {
    description = "Nombre del router"
    default  = "r1_terrafrom"
}
variable "instancia" {
    description = "Nombre instalancia"
    default  = "tesis"
}
variable "imagen_id" {
    description = "El id de la imagen  que utilizaremos."
    default  = "cc7c5431-b296-43b5-9ee3-caf1e0132c1a"
}
variable "usuario" {
    description = "Nombre se usuario con  el cual nos conectaremos por ssh."
    default  = "ubuntu"
}
variable "sabor" {
    description ="Id de el sabor (Cantidad de VCPU Y memoria Ram)"
    default = "2"
}
```

Figura 5.5

Fragmento de código de variables.

tamaño del disco, la dirección IP, se define los nombres de los recursos de la instancia, router y de la red virtual privada.

En la Figure 5.6 el bloque `keypair_private` se envía el parámetro ssh-key que hace referencia el archivo que se genero de la clave privada en este caso hace referencia a la ruta de la clave, el bloque `path_file` se envía una ruta de la carpeta que contiene los archivos de la pagina web a desplegar en el servidor Nginx.

El bloque de código `user_date` se envía como parámetro el archivo userdata.yaml, que es un tipo de archivo que utiliza Cloud-init que es un método para inicializar las instancias en múltiples

```
variable "keypair_private" {
    description = "Ruta de la clave privada ."
    default  = "/home/tesis/.ssh/id_rsa"
}
variable "user_date" {
    description = "Archivo de canfiguracion de inicio."
    default  = "userdata.yaml"
}
variable "path_file" {
    description = "Carpeta con archivos de la pagina web."
    default  = "web/"
}
```

Figura 5.6

Fragmento de código de variables.

nubes, esta lee los meta-dados que se le envíen a la hora que se inicializa el S.O.

En el fragmento de código de la Figure 5.7 en la línea 4 se manda a actualizar los paquetes, en la línea 5 se define la instalación el paquete de Docker como se ve en la línea 6 que va ser descargado con Docker.io que son paquetes que proporciona Ubuntu para la instalación, el siguiente comando `runcmd` porcina ejecutar comandos en el bash, en la línea 9 se manda a ejecutar Docker en modo detach, se envía la ruta de la ubicación de los archivos de la pagina web que se encuentra en el directorio /tmp, y que se monten en la carpeta de Nginx, además se expone el puerto 80 y se envía el nombre de la imagen Nginx, recuerde que este archivo se ejecutara al iniciar la instancia.

```
#cloud-config
# Documento acerca de este lenguaje:
# https://cloudinit.readthedocs.io/en/latest/topics/examples.html
package_update: true
packages:
  - docker.io
runcmd:
  - cd /root
  - docker run -d -v /tmp:/usr/share/nginx/html -p 80:80 nginx
```

Figura 5.7

Valores de salida archivo userdata.yaml.

Para finalizar definimos las variables de las credenciales de OpenStack, que llamaran a las varia-

bles de entorno, que se definió en el Anexo 2. Para llamarlas solo es necesario poner el nombre de la variable de entorno sin el prefijo TF_VAR_ como se observa en la Figure 5.8.

```
58  variable "AUTH_URL" {}
59  variable "USUARIO" {}
60  variable "PASSWORD" {}
```

Figura 5.8

Varibles de credencias de OpenStack.

Valores de salida.

Los valores de salida son información que ya devuelve Terraform después de la ejecución, estos son de utilidad para consultar atributos de los recursos de la infraestructura, en la Figure 5.9 se muestra el fragmento de código de la recuperación de información de la instancia, al finalizar la aplicación del plan se mostrara en consola dicha información.

```
1  output "ip_flotante" {
2    value = openstack_networking_floatingip_v2.ip_flotante.address
3  }
4  output "nombre_instancia" {
5    value = openstack_compute_instance_v2.servidor_web.name
6  }
7  output "disco_duro_GB" {
8    value = openstack_blockstorage_volume_v2.volumen.size
9  }
```

Figura 5.9

Valores de salida archivo output.tf.

Fuente de datos.

La fuente de datos permite recuperar datos fuera de Terraform es decir que no sea haya generado con Terraform por ejemplo que se haya generado directamente desde OpenStack, para luego ser utilizo en las configuraciones como en la Figure 5.10 se observa en el bloque `openstack_networking_network_v2` recupera los datos de la red externa en el bloque `openstack_compute_keypair_v2` este tipo de recurso contiene la clave publica del cliente, como

anterior mente fue ingresada véase en Anexo 3, en el siguiente bloque se define un archivo de tipo text con el va a iniciar la instancia luego de que se haya instalado el S.O.

El fragmento de código `template_cloudinit_config` que se encuentra en la Figure 5.10, permite enviar un tipo de plantilla con el cual se inicia la instancia, en la instancia se utiliza cloud-init como se explico anterior mente, este recurso de Terraform permite enviar diferentes formatos de datos, con `part` ayuda a separar los formatos de datos se envió el tipo de dato `text/cloud-config` que un texto de cloud-init además agregar el contenido que es un archivo, que se declaro en el script de variables, con `file` indica que es un tipo de archivo que se recupera de la variable `user_data`. Cabe destacar que solo se utiliza un tipo de formato de datos, si se desea agregar otro tipo de datos se tendría que agregar otra parte.

Figura 5.10

Archivo de fuente de datos (data.tf).

Recursos.

Los recursos son los bloques mas importantes ya que estos los fundamentales para el desplegué de la infraestructura como la red, volumen, CPU, memoria RAM, etc. Esta configuración se encuentra en el script `main.tf`.

En la Figure 5.11 se muestran la configuración de la red, el primer bloque de código se muestra

la creación de la red virtual privada, se le asigna un nombre con `var.nombre_red_interna`, se llama a la variable `nombre_red_interna` que se encuentra en el script `variables.tf`, el `var` se utiliza varias veces para llamar a diferentes variables, en el segundo bloque se crea la sub red virtual privada en la cual se le agrega a la red que se creo en el bloque anterior, en `cidr` se coloca la dirección IP de la red, en `ip_version` se utilizo IPv4 y el DNS de Google.

```
#Creamos la red interna
resource "openstack_networking_network_v2" "redInternaterraform" {
  name           = "${var.nombre_red_interna}"
  admin_state_up = "true"
}

#Creamos la sub red interna
resource "openstack_networking_subnet_v2" "subRedTerraform" {
  name            = "${var.subred_interna}"
  network_id      = openstack_networking_network_v2.redInternaterraform.id
  cidr            = "${var.red_ip4}"
  ip_version      = 4
  dns_nameservers = ["8.8.8.8", "8.8.4.4"]
}
```

Figura 5.11

Configuración de la red virtual privada.

Los router se encargan de enrutar la conexión entre la red externa el internet y la red virtual privada, en la Figure 5.12 en el primer bloque de código se crea el router se le asigna un nombre y además se conecta la red externa con el router, `external_network_id` es el ID de la red externa, que se recupera de los datos de el script `data.tf`. En el segundo bloque de código se conecta la interfaz de la subred interna que se creo anteriormente con el router.

```
#creamos el router
resource "openstack_networking_router_v2" "router_1" {
  name                = "${var.router}"
  external_network_id = data.openstack_networking_network_v2.network.id
}

#Realizamos la conecion el router con la sub red
resource "openstack_networking_router_interface_v2" "router_interface_1" {
  router_id = openstack_networking_router_v2.router_1.id
  subnet_id = openstack_networking_subnet_v2.subRedTerraform.id
}
```

Figura 5.12

Configuración de router.

Previo a la creación de la instancia para el servidor web, se crea un script de Terraform para el grupo de seguridad, este básicamente permite abrir puertos según se los requiera, para el servidor web se abre el puerto 80 para la pagina web, el puerto 22 para conexión SSH y además el puerto IMCP para pruebas de conexión ping.

El grupo de seguridad por defecto todos los puertos se encuentran cerrados, en la Figure 5.13 en la línea 8 se crea una regla para el grupo de seguridad, el primer parámetro que se solicita es si la regla es de ingreso o salida, además se ingresa el protocolo de la capa 3 (IPv4,IPv6) y el protocolo de la capa cuatro, se coloca el rango del protocolo máximo y mínimo se se utiliza el puerto 80, `remote_ip_prefix` es la dirección IP que puede ingresar, la dirección IP 0.0.0.0/0 se coloco ya que al ser un servicio web van a poder acceder todas las direcciones IP que lo necesiten, pero para el caso de SSH solo se colocara la dirección IP que va gestionar el servidor, con `security_group_id` agrega la regla al grupo de seguridad creado anteriormente. En la Figure 5.13 se da el ejemplo para el puerto 80, de mas misma forma se lo realiza para el puerto 22 y IMCP.

```
1  ####Grupo de seguridad ######
2  resource "openstack_networking_secgroup_v2" "grupo_seguridad" {
3    name        = "puertosWeb"
4    description = "Grupo de seguridad para servidodr web."
5  }
6
7  # Permitir puerto 80
8  resource "openstack_networking_secgroup_rule_v2" "puerto_80" {
9    direction         = "ingress"
10   ethertype         = "IPv4"
11   protocol          = "tcp"
12   port_range_min    = 80
13   port_range_max    = 80
14   remote_ip_prefix  = "0.0.0.0/0"
15   security_group_id = "${openstack_networking_secgroup_v2.grupo_seguridad.id}"
16 }
```

Figura 5.13

Configuración de instancia.

En este apartado se explica como se creo la instancia para el servidor web, en la Figure 5.14 se observa los parámetros como nombre el ID de la imagen es una imagen Ubuntu, estos datos

están definidos en el script de variables como se ve en Figure 5.5, otro parámetro `flavor_id` es el ID de sabor que es los VCPU y memoria RAM que se le asigna, por lo general OpenStack tiene varios sabores creados o también se los puede crear dependiendo de lo requerido, se elige el sabor con ID 2 el cual proporciona 1 VCPU y 2 GB de memoria RAM, con `security_groups` se pasa el grupo de seguridad creado anteriormente, la llave publica es muy importante permite iniciar sesión con SSH de manera remota, con `key_pair` se envía el nombre de clave que fue ingresada a OpenStack, todo esto fue explicado en el apartado de Fuente de datos, además se envía los datos de configuración de inicio para la instancia con `user_data` se envia el script user_data que se explico en el apartado de Fuente de datos y para finalizar se agrega el nombre la red virtual interna.

```
28  #Creacion  de instancia
29  resource "openstack_compute_instance_v2" "servidor_web" {
30    name            = "${var.instancia}"
31    image_id        = "${var.imagen_id}"
32    flavor_id       = "${var.sabor}"
33    security_groups = ["${openstack_networking_secgroup_v2.grupo_seguridad.name}"]
34    key_pair        = data.openstack_compute_keypair_v2.clave.name
35    user_data       ="${data.template_cloudinit_config.init_script.rendered}"
36
37    network {
38      name = openstack_networking_network_v2.redInternaterraform.name
39    }
40  }
```

Figura 5.14

Configuración de instancia.

Para finalizar con todos los componentes principales de la instancia se crea el volumen que es el almacenamiento de la instancia, en la Figure 5.15 el primer bloque de código muestra la creación del volumen dándole un nombre y tamaño que vine dado en Gigabyte. El segundo bloque asocia el volumen a la instancia que se lo hace con el ID de la instancia y el ID del volumen.

En la Figure 5.16 se presenta la asignación de una dirección IP flotante a la instancia del pool de direcciones IP de OpenStack. En la línea 56 se le asigna la red externa la que proporcionara la dirección IP, y en el siguiente bloque de código se asocia la dirección IP flotante con la instancia.

```
#Crear Volumen
resource "openstack_blockstorage_volume_v2" "volumen" {
  name = "volumen"
  size = "${var.volumen_tamano}"
}

#Asociar volumen a la instancia
resource "openstack_compute_volume_attach_v2" "attached" {
  instance_id = openstack_compute_instance_v2.servidor_web.id
  volume_id   = openstack_blockstorage_volume_v2.volumen.id
}
```

Figura 5.15

Generar y asignar volumen a la instancia.

```
#Definimos el pool que usaremos para la ipflotante
resource "openstack_networking_floatingip_v2" "ip_flotante" {
  pool = "${var.red_externa}"
}

#Asociar una ip flotante a la instancia
resource "openstack_compute_floatingip_associate_v2" "myip" {
  floating_ip = openstack_networking_floatingip_v2.ip_flotante.address
  instance_id = openstack_compute_instance_v2.servidor_web.id
}
```

Figura 5.16

Asignar de la dirección IP flotante a la instancia.

Para finalizar se crea el bloque de código `null_resource` este es un bloque sin recursos, es decir que no esta ligado a ningún recurso en especifico.En este bloque se tiene aprovisionares para modelar acciones especificas en la instancia como por ejemplo para preparar un servicio, ejecutar scripts, ejecutar línea de comando y copiar archivos una vez finalizada la instalación de la instancia.

En la Figure 5.17 el primer bloque se configura para un acceso con SSH a la instancia, en que se ingresa el tipo de conexión, la dirección IP de host que es la dirección IP flotante que se asigno, el usuario que es Ubuntu y la llave privada, que se le paso el archivo con la llave.

En el siguiente bloque de código con `provisioner ‘‘file’’` permite que una vez accedido a

la instancia se copia los archivos de la maquina local hacia la instancia creada, se requiere dos valores el origen y el destino de los archivos a copiar.

```
65  resource "null_resource" "aprovisionar" {
66    # Conección SSH
67    connection {
68      host        = openstack_networking_floatingip_v2.ip_flotante.address
69      user        = "${var.usuario}"
70      private_key = file("${var.keypair_private}")
71    }
72    #Copiar archivos a la instancia
73    provisioner "file" {
74      source      = "${var.path_file}"
75      destination = "/tmp/"
76    }
77  }
```

Figura 5.17

Configuración SSH y copiar archivos de pagina web.

5.2.1.3 Versionamiento de la infraestructuras.

Una vez escrito el script con las configuraciones se sube a la plataforma de versionamiento GitHub, para tener un repositorio es necesario tener un cuenta en dicha plataforma además tener instalado Git, con esto se seguir los siguientes comandos para subir los script a GitHub:

```
cd escenario1

git init

git add . git branch -M main

git remote add origin https://github.com/xriera/terraform-escenario1.git

git push -u origin main
```

Un vez finalizado debe de tener un resultado parecido a la Figure 5.18

5.2.1.4 Inicialización de herramienta de la herramienta IaC.

```
[MacBook-Pro-de-Xavier:escenario1 xavier$ git push -u origin main
Enumerating objects: 61, done.
Counting objects: 100% (61/61), done.
Delta compression using up to 12 threads
Compressing objects: 100% (52/52), done.
Writing objects: 100% (61/61), 22.49 MiB | 3.23 MiB/s, done.
Total 61 (delta 5), reused 0 (delta 0)
remote: Resolving deltas: 100% (5/5), done.
To https://github.com/xriera/terraform-escenario1.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Figura 5.18

Repositorio GitHub

Ahora que se cuenta con los archivos de la configuración de Terraform versionado antes de ejecutar el plan, primero se descarga el proveedor que se encuentra definido en `provaider.tf`. Se abre el terminar en la ubicación donde se encuentre el script `provaider.tf`, se ejecuta el comando `terraform init` lo que hace es descargar el plugin de OpenStack tendrá un resultado parecido a la Figure 5.19.

```
tesis tesis ~ cd terrfrom-srv-web/
tesis tesis main ~ terrfrom-srv-web terraform init

Initializing the backend...

Initializing provider plugins...
- Finding terraform-provider-openstack/openstack versions matching "~> 1.35.0"...
- Finding latest version of hashicorp/null...
- Finding latest version of hashicorp/template...
- Installing terraform-provider-openstack/openstack v1.35.0...
- Installed terraform-provider-openstack/openstack v1.35.0 (self-signed, key ID 4F80527A391BEFD2)
- Installing hashicorp/null v3.1.0...
- Installed hashicorp/null v3.1.0 (signed by HashiCorp)
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
tesis tesis main ~ terrfrom-srv-web
```

Figura 5.19

Resultado de comando terraform init.

En la Figure 5.19 se muestra la descarga y la iniciación los plugins para el proveedor de OpenStack, lo descargado se encuentra de forma predeterminada en un archivo llamado .terraform,

este comando se utiliza cada vez que se tiene nuevo código de Terraform.

5.2.1.5 Generar el plan de aprovisionamiento.

Una vez iniciado, el siguiente paso es lanzar el plan de implementación de los recursos, pero primero se tiene que validar la configuración, con el comando `terraform validate`, si la sintaxis de los script se encuentran correcto se tendrá un resultado como la Figure 5.20 caso contrario ejecute el plan para ubicar el posible error.

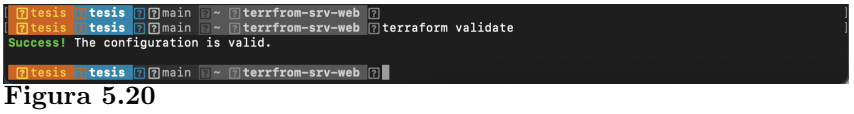


Figura 5.20

Resultado de comando terraform validate.

El comando `terraform plan` permite ver la configuración de los recursos de Terraform antes de aplicarlos, la salida de el comando es simple de interpretar los recursos que se crearan tiene un signo mas (+), mientras que los recursos con el signo menos (-) se eliminarán, en la Figure 5.21 se visualiza la ejecución del comando `terraform plan`.

La Figure 5.21 muestra un fragmento del resultado de plan que se aplica, en el cual se visualiza la creación de un volumen(disco de almacenamiento), dirección IP flotante y la instancia, como se nota algunos valores se generan una vez creada la instancia como por ejemplo ID de la instancia, pero hay otros valores que se envía mediante parámetros como el nombre de la instancia.

Una buena practica es asegurar que la configuración de los recursos vayan bien, este se lo va a realizar de forma manual para revisar el plan propuesto, bien de la modificación si existe alguna o creación de los recursos. La aprobación de el plan es el paso para la aplicación, un plan mal aprobado produciría gastos económicos ya que se tendría recursos innecesarios en la

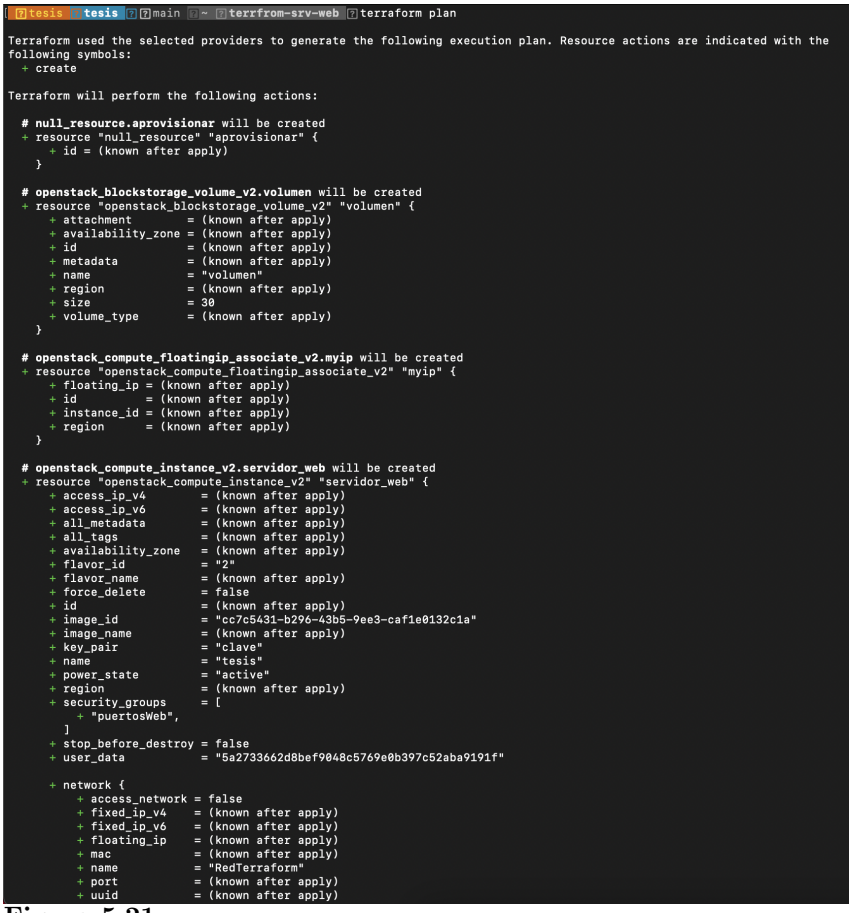


Figura 5.21

Fragmento de resultado de comando `terraform plan`.

nube privada, para garantizar un plan es recomendable tener un solo plan pendiente a la vez, al generar un nuevo plan invalidara a todos los anteriores con ese plan, en caso de que se haya modificado el estado del código se tendrá que volver calcularse un plan para el nuevo código.

5.2.1.6 Aplicación del plan de aprovisionamiento.

Una vez aprobado el plan de lo deseado a implementar, lo siguiente es aplicar la configuración se la realiza con el comando `terraform apply` este muestra el mismo resultado al del `plan` con las mismas parámetros ya explicadas y al final muestra un mensaje que si desea confirmar el plan y aplicarlo como se muestra en la Figure 5.22.

Figura 5.22

Ejecución del comando terraform apply.

Es necesario recalcar que con `terraform init` y `terraform plan` no se a realizado ningún cambio en OpenStack, solo el comandó `terraform apply` sera el encargado de realizar cambios en el entorno de OpenStack. De igual forma el comando `terraform apply` internamente valida la sintaxis del código así como ejecuta el plan de nuevo antes de aplicar la configuración en la nube, esto es una forma de comprobar si se a realizado algún cambio desde la ultima ejecución de del comando `terraform plan`.

Además Terraform realiza un seguimiento a los recursos ya desplegados de esta configuración, por lo tanto ya sabe que recursos ya se encuentran creados, así que al modificar el plan solo tomara esos cambios para aplicarlos, es decir que no creara de nuevo todos los recursos de la configuración.

Una vez aceptado se aplicara la configuración en OpenStack, en la Figure 5.23 se muestra cada

uno de los recursos que se crearon en total 14 recursos nuevos, en aprovisionar se muestra que tomo mas tiempo por la conexión SSH y la copia de los archivos de la paginas web, y por ultimo se tiene un `output` que es la dirección IP flotante.

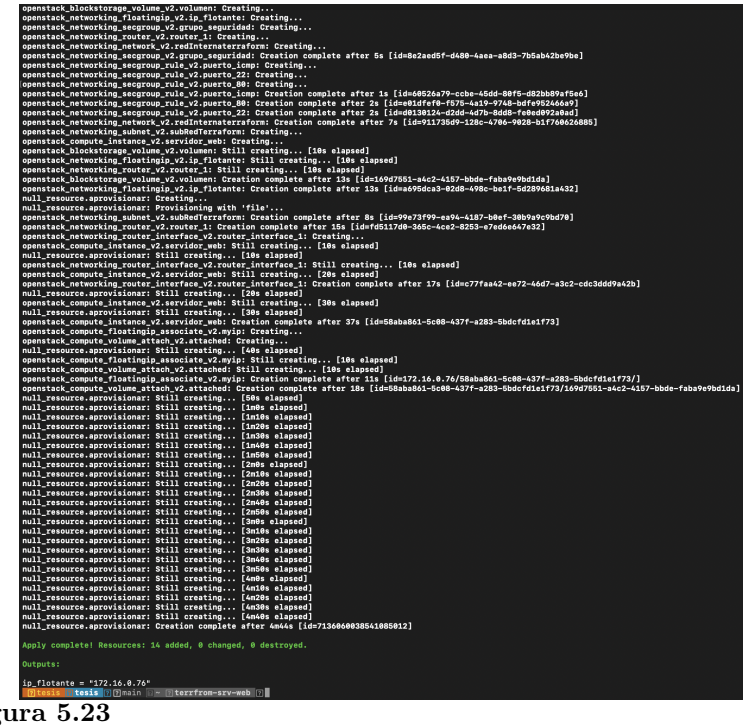


Figura 5.23

Resultado del comando terraform apply.

Una vez finalizado de la ejecución del plan, se espera unos momentos hasta que ejecute la configuración de arranqué que se configuro, con un navegador web y la dirección IP flotante se visualiza la pagina web como se observa en la Figure 5.24.

Para ingresar a la instancia de formar remota se lo realiza por SSH con el comando `ssh -i /home/tesis/.ssh/id_rsa ubuntu@ip-instancia` para lo cual se utiliza la ruta de la ubicación de la llave primaria, usuario y la dirección IP flotante, en la Figure 5.25 se visualiza la conexión a la instancia.

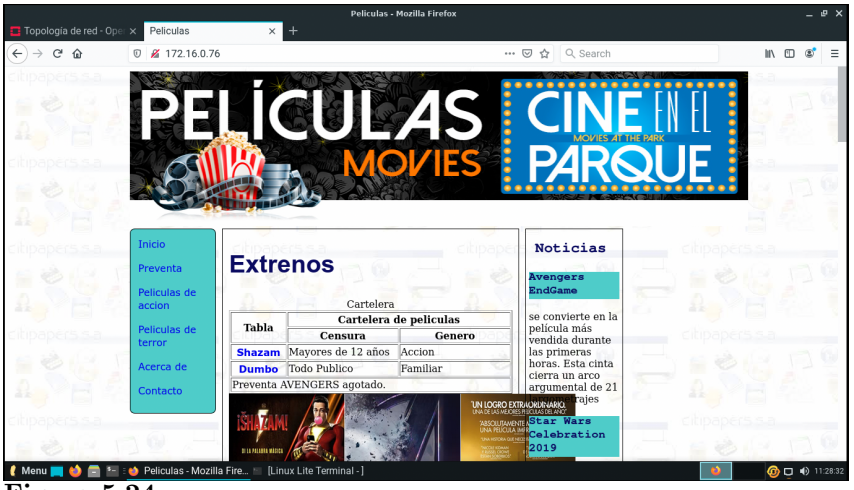


Figura 5.24

Pagina web desplegada con Docker.

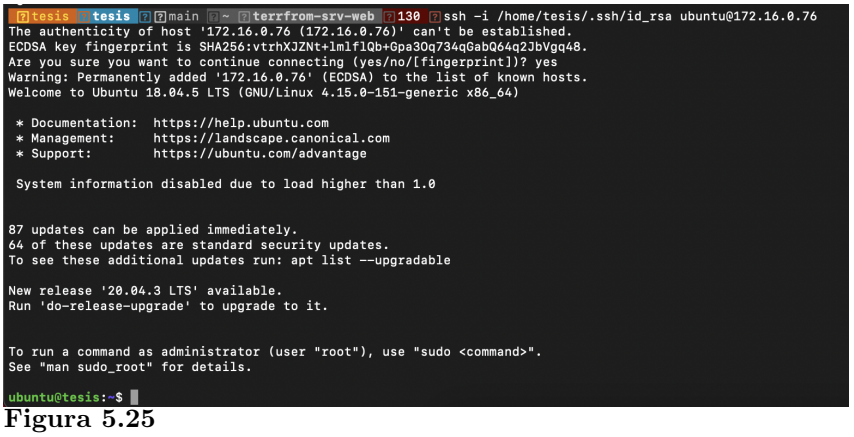


Figura 5.25

Conexión remota con SSH.

5.2.2 Implementación del Escenario B

5.2.2.1 Diseño del aprovisionamiento de la infraestructura.

En este escenario se despliegan tres instancias por lo cual se utiliza para el desplegué de la arquitectura la herramienta Terraform y para gestionar la configuración Ansible, la cual se desplegara con contenedores de Wordpress y base datos, en la Figure 5.26 se muestra la topología del escenario.

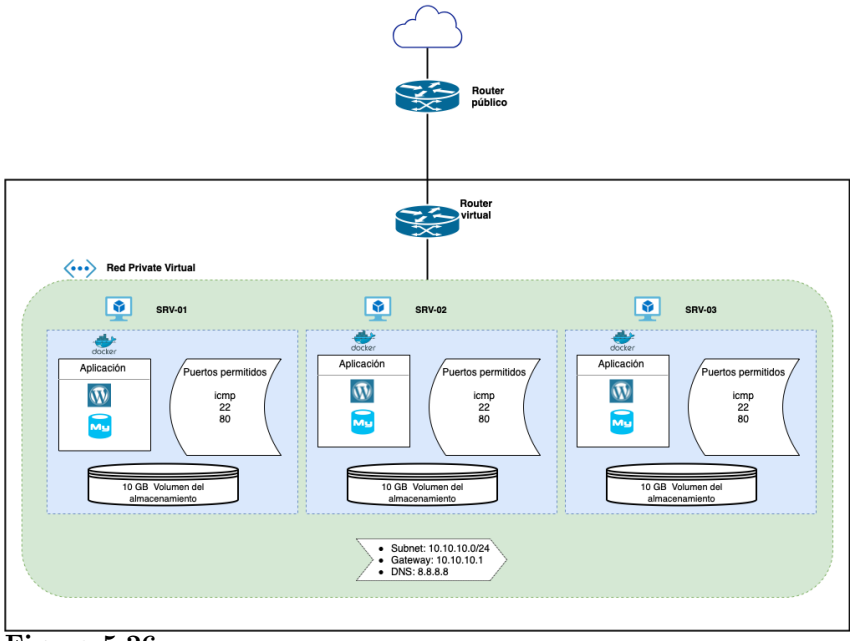


Figura 5.26

Topología de escenario B.

5.2.2.2 Codificación de scripts para el aprovisionamiento.

Para el desarrollo del escenario B, se tendrá en cuenta varios conceptos que ya fueron explicados en el escenario A, ya que partes del código serán reutilizados, por lo tanto algunos de los scritps que se presentaran a continuación no se detallaran a profundidad estos se los puede revisar en Desarrollo de Script de Escenario A. De acuerdo a lo mencionado en la Figure 5.27 se presenta la estructura del proyecto para el escenario B.

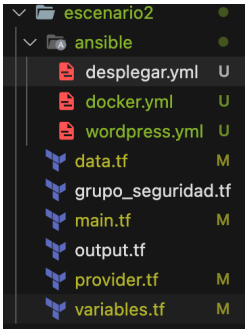


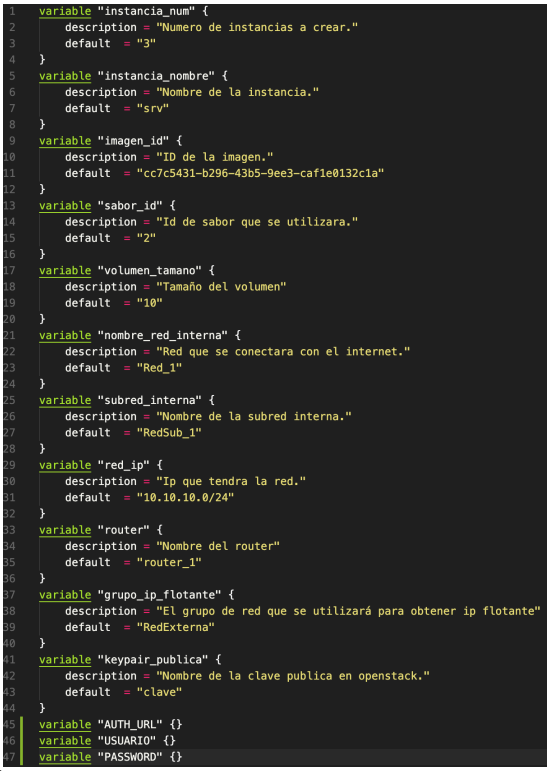
Figura 5.27

Estructura del proyecto Terraform para el escenario B.

Como se muestra en la Figure 5.27 se mantiene una estructura similar a la presentada en el escenario A, con la mismas funcionalidad pero con diferente objetivo de desplegué de la arquitectura, pero por ejemplo el script `provider.tf` no tendrá ninguna modificación debido a que el proveedor sera el mismo,que en el escenario anterior como es OpenStack.

Valores de entrada.

Las variables de entrada que se presenta en la Figure 5.28 son para crear una o mas instancias, dicho valor sera ingresado en `instancia_num`, los de mas valores de entradas son para la red, router, tamaño de volumen, sabor, etc.



```
variable "instancia_num" {
    description = "Numero de instancias a crear."
    default  = "3"
}
variable "instancia_nombre" {
    description = "Nombre de la instancia."
    default  = "srv"
}
variable "imagen_id" {
    description = "ID de la imagen."
    default  = "cc7c5431-b296-43b5-9ee3-caf1e0132c1a"
}
variable "sabor_id" {
    description = "Id de sabor que se utilizara."
    default  = "2"
}
variable "volumen_tamano" {
    description = "Tamaño del volumen"
    default  = "10"
}
variable "nombre_red_interna" {
    description = "Red que se conectara con el internet."
    default  = "Red_1"
}
variable "subred_interna" {
    description = "Nombre de la subred interna."
    default  = "RedSub_1"
}
variable "red_ip" {
    description = "Ip que tendra la red."
    default  = "10.10.10.0/24"
}
variable "router" {
    description = "Nombre del router"
    default  = "router_1"
}
variable "grupo_ip_flotante" {
    description = "El grupo de red que se utilizará para obtener ip flotante"
    default  = "RedExterna"
}
variable "keypair_publica" {
    description = "Nombre de la clave publica en openstack."
    default  = "clave"
}
variable "AUTH_URL" {}
variable "USUARIO" {}
variable "PASSWORD" {}
```

Figura 5.28

Script `varibles.tf`

Valores de salida.

En la Figure 5.29 se mostrara las direcciones IPs flotantes de las instancias, esto se lo realiza

mediante un for recorriendo la lista de recursos de las direcciones IP flotantes.

```
output "float_ips" {
    value = {
       for fip in openstack_networking_floatingip_v2.ip_flotante:
          fip.fixed_ip => fip.address
    }
}
```

Figura 5.29

Script ***output.tf***

Fuente de datos.

En la fuente de datos se recupera los datos de la red externa y la clave publica que fue ingresada en OpenStack la cual sera utilizado para ingresar por medio de SSH a la instancia, en la Figure 5.30 se observa el fragmento de código utilizado.

```
#Recuperar id de la Red Externa
data "openstack_networking_network_v2" "network" {
  name = "${var.red_externa}"
}
#Recuperamos la clave publica
data "openstack_compute_keypair_v2" "clave" {
  name = "${var.keypair_publica}"
}
```

Figura 5.30

Script ***data.tf***

Recursos.

Los recursos son la parte fundamentan de la composición de la infraestructura, en la Figure 5.31 se muestra el fragmento de código de lo referente a la red virtual y el router.

En cuanto a la creación de la instancia en este escenario se desplegara 3 instancias en la Figure 5.32, línea 25 con `count` es un meta-argumento que recibe números enteros, en donde se

```
#Creamos la red interna
resource "openstack_networking_network_v2" "red_interna" {
  name           = "${var.nombre_red_interna}"
  admin_state_up = "true"
}
#Creamos la sub red interna
resource "openstack_networking_subnet_v2" "sub_red" {
  name            = "${var.subred_interna}"
  network_id      = openstack_networking_network_v2.red_interna.id
  cidr            = "${var.red_ip}"
  ip_version      = 4
  dns_nameservers = ["8.8.8.8", "8.8.4.4"]
}
#creamos el router
resource "openstack_networking_router_v2" "router_1" {
  name                = "${var.router}"
  external_network_id = data.openstack_networking_network_v2.network.id
}
#Realizamos la conecion el router con la sub red
resource "openstack_networking_router_interface_v2" "router_interface" {
  router_id = openstack_networking_router_v2.router_1.id
  subnet_id = openstack_networking_subnet_v2.sub_red.id
}
```

Figura 5.31

Fragmento de código de `main.tf` *para creación de redes y router*

definió el numero de instancias a desplegar, en la siguiente línea se proporciona el nombre a la instancia, que al haber mas de una instancia se le debe dar un formato en el nombre, para que no se repitan los nombres.

En el segundo bloque de código se muestra la creación de los volúmenes de la misma forma la variable `count` se emplea para que cree la misma cantidad de volúmenes que de instancias, en el tercer bloque de código asocia los volúmenes creados a las instancias.

```
resource "openstack_compute_instance_v2" "srv_instancia" {
  count = var.instancia_num
  name = format("%s-%02d", var.instancia_nombre, count.index+1)
  image_id = var.imagen_id
  flavor_id = var.sabor_id
  key_pair = data.openstack_compute_keypair_v2.clave.name
  security_groups = ["${openstack_networking_secgroup_v2.grupo_seguridad.name}"]
  network {
    name = openstack_networking_network_v2.red_interna.name
  }
}
#Crear Volumen
resource "openstack_blockstorage_volume_v2" "volumen" {
  count = var.instancia_num
  name = format("volumen-%02d", count.index+1)
  size = "${var.volumen_tamano}"
}
#Asociar volumen a la instancia
resource "openstack_compute_volume_attach_v2" "attached" {
  count = var.instancia_num
  instance_id = element(openstack_compute_instance_v2.srv_instancia.*.id, count.index)
  volume_id   = element(openstack_blockstorage_volume_v2.volumen.*.id, count.index)
}
```

Figura 5.32

Fragmento de codigo de `main.tf`.

Y para finalizar en en cuanto a al desplegué de la infraestructura es la asignación de las direcciones IP flotante para cada instancia como se muestra en la Figure 5.33.

```
48  resource "openstack_networking_floatingip_v2" "fip" {
49    count = var.instancia_num
50    pool = var.grupo_ip_flotante
51  }
52
53  resource "openstack_compute_floatingip_associate_v2" "fip" {
54    count = var.instancia_num
55    floating_ip = element(openstack_networking_floatingip_v2.fip.*.address, count.index)
56    instance_id = element(openstack_compute_instance_v2.srv_instancia.*.id, count.index)
57  }
```

Figura 5.33

Fragmento de código de ***main.tf****.*

5.2.2.3 Versionamiento de la infraestructuras.

Ya codeado el script se procede a subir a la plataforma de versionar para ello se crea un repositorio y se sube los scripts a la plataforma de versionamiento con los siguientes comandos:

```
cd escenario1

git init

git add . git branch -M main

git remote add origin https://github.com/xriera/terraform-esenario2.git

git push -u origin main
```

Se tendrá un resultado parecido a como se muestra en la Figure 5.34.

```
MacBook-Pro-de-Xavier:escenario2 xavier$ git push -u origin main
Enumerating objects: 22, done.
Counting objects: 100% (22/22), done.
Delta compression using up to 12 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (22/22), 10.51 MiB | 1.02 MiB/s, done.
Total 22 (delta 0), reused 0 (delta 0)
To https://github.com/xriera/terraform-esenario2.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Figura 5.34

Resultado de comando git push.

5.2.2.4 Inicialización de herramienta de la herramienta IaC.

Para inicializar Terraform se accede a la carpeta en donde se encuentre los scripts, una vez ingresado se ejecuta el comando `terraform init` para descargar los plugins de OpenStack, se tendrá una resultado simular a la Figure 5.35.

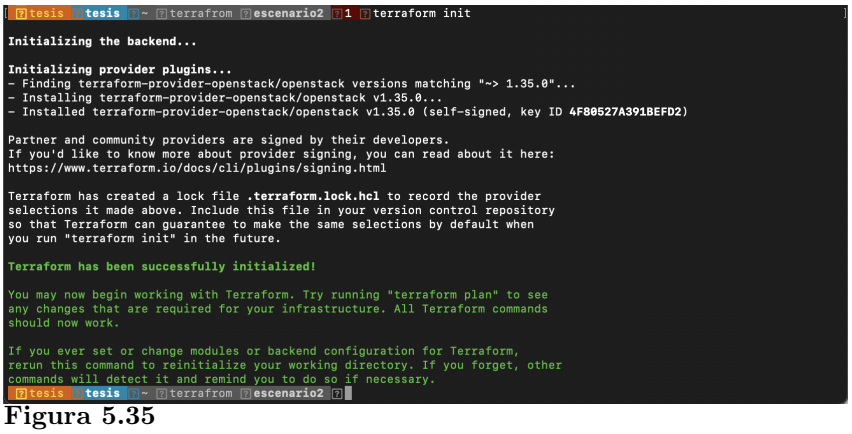


Figura 5.35

Resultado de comando `terraform init`.

5.2.2.5 Generar el plan de aprovisionamiento.

En este paso se muestra el plan de ejecución propuesto, en donde se muestra los recursos previstos antes de ser aplicados, esto se lo realiza con el comando `terraform plan` se tendrá un resultado similar como la Figure 5.36, en este paso es importante verificar los recursos que se van a desplegar que concuerden los recursos planteados en el paso 1.

5.2.2.6 Aplicación del plan de aprovisionamiento.

Al terminar la aprobación del plan, se procede a ejecutar el plan esto se lo realiza con el comando

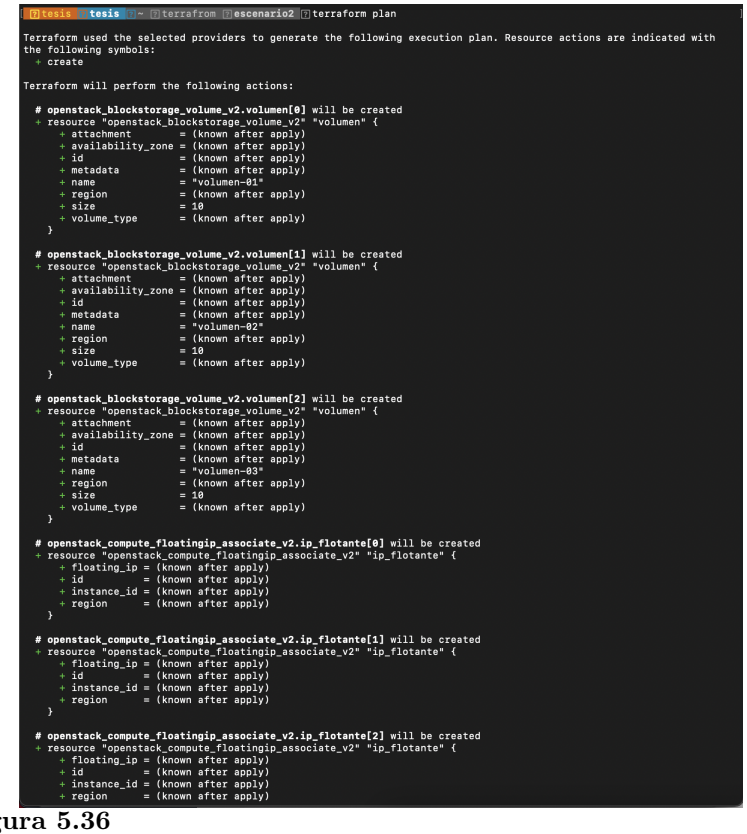


Figura 5.36

Resultado de comando `terraform plan`.

`terraform apply` se tendrá un resultado como en la Figure 5.37 se observa que agregaran 23 recursos nuevos, al final se confirma el plan para el desplegué en OpenStack.

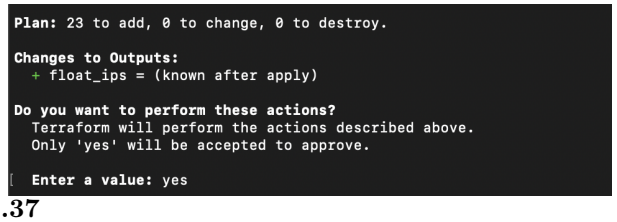


Figura 5.37

Resultado de comando `terraform apply`.

Una vez confirmado el plan, los recursos se comienzan a desplegar como se muestran en la Figure 5.38 además se visualiza los direcciones IPs flotantes que son las variables de salida (Outputs).

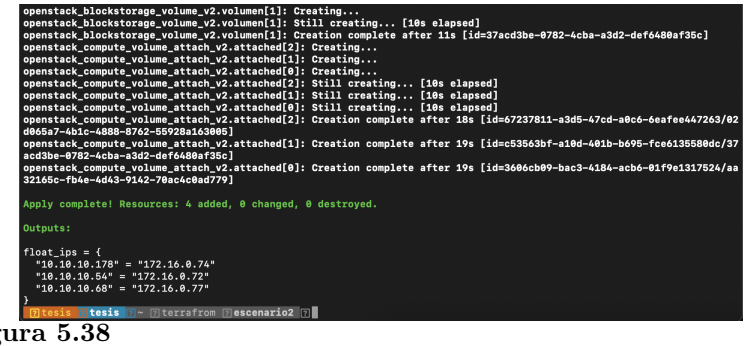


Figura 5.38

Resultado de comando `terraform apply`.

5.2.2.7 Aplicación del gestor de configuración.

Para finalizar se aplica las configuraciones para desplegar las aplicaciones que se definieron el paso 1, para esto se usa Ansible, la misma que permite por medio de un script ejecutar una secuencia de tareas. Lo primero es configurar los host para ello se abre el archivo que se encuentra en `nano /etc/ansible/hosts` en el cual se agrega las direcciones IP de las instancias que se que desea gestionar como muestra la Figure 5.39.

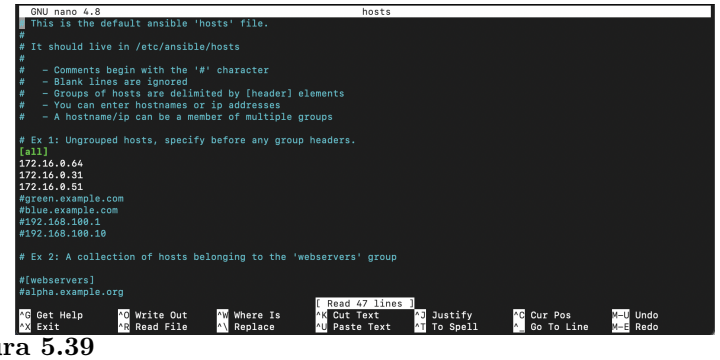


Figura 5.39

Archivo de configuración de hosts.

Una vez configurado los hosts, se ejecuta el playbook con la lista de tareas este se divido en dos playbooks uno para la instalación de docker y el otro para el desplegué de Worpress y las base de datos MYSQL. En la Figure 5.40 se visualiza el código del playbook `docker.yml` que contiene

las tareas para la instalación de Docker.

```yaml
---
- hosts: all
  become: true

  tasks:
    - name: Instalar dependencias
      apt:
        name: "{{item}}"
        state: present
        update_cache: yes
      loop:
        - apt-transport-https
        - ca-certificates
        - curl
        - gnupg-agent
        - software-properties-common
    - name: Agregue la clave GPG oficial de Docker
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present
    - name: Añadir repositorio docker
      apt_repository:
        repo: deb https://download.docker.com/linux/ubuntu bionic stable
        state: present
    - name: Instalar docker
      apt:
        name: "{{item}}"
        state: latest
        update_cache: yes
      loop:
        - docker-ce
        - docker-ce-cli
        - containerd.io
...
```

Figura 5.40

Playbook para instalación de Docker.

Con el comando `ansible-playbook docker.yml -u ubuntu` este permite ejecutar el archivo docker.yml, `-u ubuntu` se define el usuario de la instancia con el que se ejecutara el playbook. En la Figure 5.41 se visualiza el resultado de la ejecución del comando `ansible-playbook docker.yml -u ubuntu`.

```
tesis  tesis  /  etc  ansible  ansible-playbook docker.yml -u ubuntu

PLAY [all] *********************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [172.16.0.64]
ok: [172.16.0.31]
ok: [172.16.0.51]

TASK [Instalar dipendencias] ***************************************************************************************
changed: [172.16.0.51] => (item=apt-transport-https)
changed: [172.16.0.64] => (item=apt-transport-https)
changed: [172.16.0.31] => (item=apt-transport-https)
ok: [172.16.0.51] => (item=ca-certificates)
ok: [172.16.0.64] => (item=ca-certificates)
ok: [172.16.0.31] => (item=ca-certificates)
ok: [172.16.0.51] => (item=curl)
ok: [172.16.0.64] => (item=curl)
ok: [172.16.0.31] => (item=curl)
changed: [172.16.0.51] => (item=gnupg-agent)
changed: [172.16.0.64] => (item=gnupg-agent)
changed: [172.16.0.31] => (item=gnupg-agent)
ok: [172.16.0.51] => (item=software-properties-common)
ok: [172.16.0.31] => (item=software-properties-common)
ok: [172.16.0.64] => (item=software-properties-common)

TASK [Agregue la clave GPG oficial de Docker] **********************************************************************
changed: [172.16.0.31]
changed: [172.16.0.51]
changed: [172.16.0.64]

TASK [Agregar respositorio Docker] *********************************************************************************
changed: [172.16.0.31]
changed: [172.16.0.51]
changed: [172.16.0.64]

TASK [Instalar Docker] *********************************************************************************************
changed: [172.16.0.51] => (item=docker-ce)
changed: [172.16.0.31] => (item=docker-ce)
changed: [172.16.0.64] => (item=docker-ce)
ok: [172.16.0.51] => (item=docker-ce-cli)
ok: [172.16.0.31] => (item=docker-ce-cli)
ok: [172.16.0.64] => (item=docker-ce-cli)
ok: [172.16.0.51] => (item=containerd.io)
ok: [172.16.0.31] => (item=containerd.io)
ok: [172.16.0.64] => (item=containerd.io)

PLAY RECAP *********************************************************************************************************
172.16.0.31                : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.16.0.51                : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
172.16.0.64                : ok=5    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Figura 5.41

Resultado de ejecutar el playbook.

De la misma forma que para el deplegue de Docker, se genero un playbook para el desplegué de Worpress y MYSQL llamado desplegar.yml, que contiene la lista de tareas para instalar y ejecutar docker-compose. En la Figure 5.42 se muestra el script desplegar.yml con las tareas.

```yaml
---
- hosts: all
  become: true
  tasks:
  - name: Install docker-compose
    get_url:
      url : https://github.com/docker/compose/releases/download/1.25.1-rc1/docker-compose-Linux-x86_64
      dest: /usr/local/bin/docker-compose
      mode: 'u+x,g+x'

  - name: Crear directorio de trabajo
    file:
      path: /tmp/wordpress
      state: directory

  - name: Deploying docker-compose template
    template:
      src: /etc/ansible/wordpress.yml
      dest: /tmp/wordpress/wordpress.yml

  - name: Running docker-compose
    shell: /usr/local/bin/docker-compose -f /tmp/wordpress/wordpress.yml up -d
...
```

Figura 5.42

Script de Ansible desplegar.yml.

En la Figure 5.43 se muestra el archivo del Docker Compose en el que se encuentra las configuraciones de la conexión a la base de datos para Wordpress y la configuración de MYSQL para Wordpress.

```yaml
version: '3.1'

services:

  wordpress:
    image: wordpress:php7.1-apache
    ports:
      - 8080:80
    environment:
      WORDPRESS_DB_HOST: mysql
      WORDPRESS_DB_USER: root
      WORDPRESS_DB_PASSWORD: root
      WORDPRESS_DB_NAME: wordpress
    links:
      - mysql:mysql


  mysql:
    image: mysql:8.0.13
    command: --default-authentication-plugin=mysql_native_password
    environment:
      MYSQL_DATABASE: wordpress
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - ~/docker/mysql-data:/var/lib/mysql
```

Figura 5.43

Código de Docker-compose para Worpress y MYQSL.

Se ejecuta el comando `ansible-playbook desplegar.yml -u ubuntu` para ejecutar la lista de tareas del playbook, en la Figure 5.44 se visualiza el resultado de la ejecución del playbook.

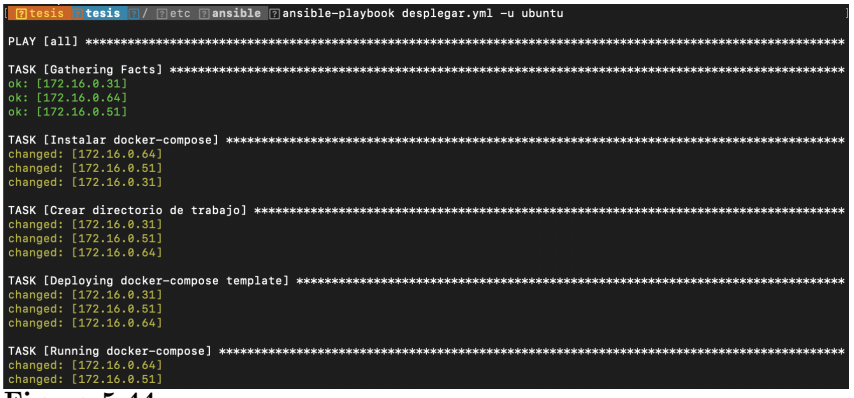


Figura 5.44

Resultado de la ejecución del playbook desplegar.yml.

Una vez ejecutado el playbook del desplegué se ingresa a las instancias mediante SSH con el comando `ssh -i /home/tesis/.ssh/id_rsa ubuntu@ip-instancia`, se ingresa con la ruta la ubicacion de la llave privada, en `ip-instancia` define la dirección IP de la instancia a que se quiere ingresar, una vez ingresado se comprueba la ejecución de los contenedores Docker con la ejecución del comando `docker ps` como se observa en la Figure 5.45.

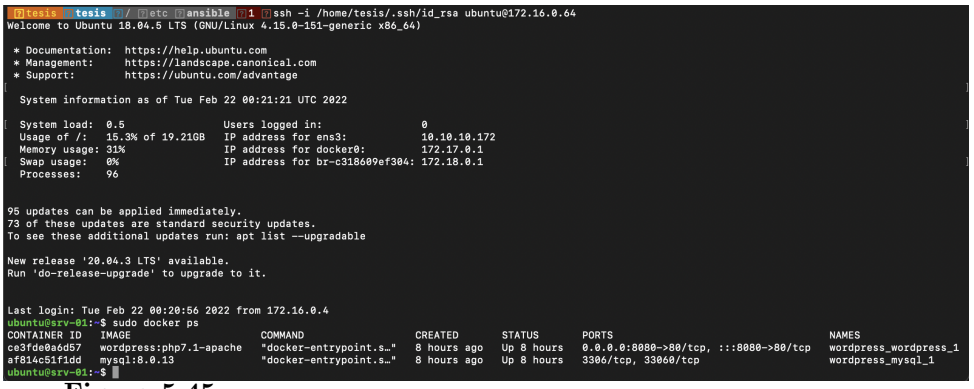


Figura 5.45

Conexión SSH a la instancias SRV-01.

Para constatar el funcionamiento del desplegué de Wordpress se ingresa al navegador Web con la dirección IP de una de las instancias, en la Figure 5.46 se visualiza la pagina web de Wordpress para configurar.

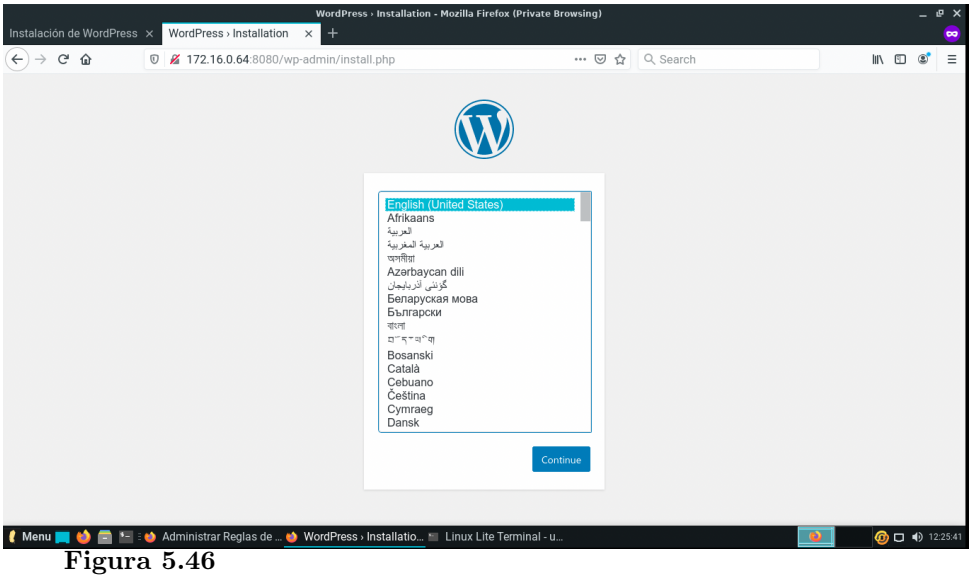


Figura 5.46

Wordpress desplegado.

Capítulo 6

Resultados

En este capitulo se muestra los resultados y pruebas obtenidos de los diferentes escenarios presentados en le capitulo anterior.

6.1 Resultados del Escenario A

El escenario A se presenta el desplegué de una arquitectura para un servidor web con Docker, como se muestra en la Figure 5.2, una vez analizada la topología a desplegar se escribió los script de Terraform con dichos requerimientos, en la sección 5.2.1 se muestra el desarrollo de cada script que se realizara las pruebas en esta sección, cabe señalar que siguió la metodología planteado para que no existe inconvenientes a la hora de realizar el desplegué de esta arquitectura.

En este apartado se muestran los resultado de los recursos que se generaron de los scripts de Terraform en la nube privada OpenStack. En la Sección 5.2.1.2 se detallo la construcción de cada uno de los recursos que se de plegaron en OpenStack y de igual manera ya se explico los comandos para la aplicación de los script de Terraform. De acuerdo con la topología presentada para este escenario que se muestra en la Figure 5.2, los scripts desplegaron la siguiente topología en OpenStack.

La Figure 6.1 muestra la red privada(RedTerraform) para la instancia en la cual se encuentra levantando el servidor de Ngnix dentro de Docker, el router virtual(r1_terraform) cuenta con dos interfaces que conectan la red virtual privada con la red externa.

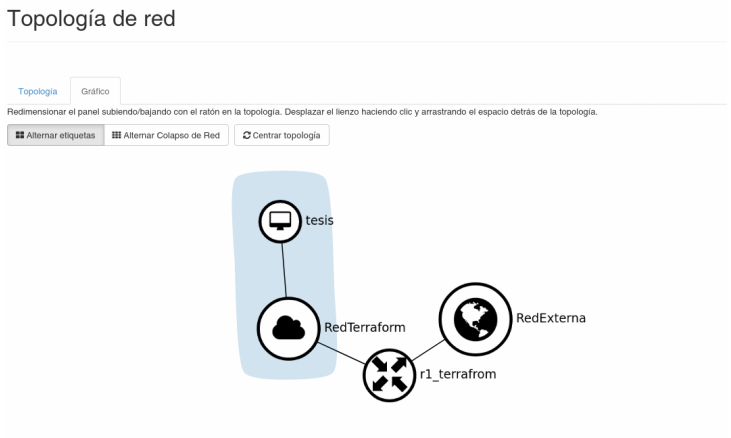


Figura 6.1

Topología desplegada en OpenStack con Terraform.

Como fue definido en el script la red virtual cuenta con la red `10.1.0.0/24`, la cual esta conectada a la instancia donde OpenStack media DHCP(Protocolo de configuración dinámica de host) proporciona una IP en el rango de la red virtual, que esta conectado a la red externa que es la red física con la que tendremos salida a internet, en la muestra Figure 6.2 la topología que se genero con la aplicación del script de Terraform.

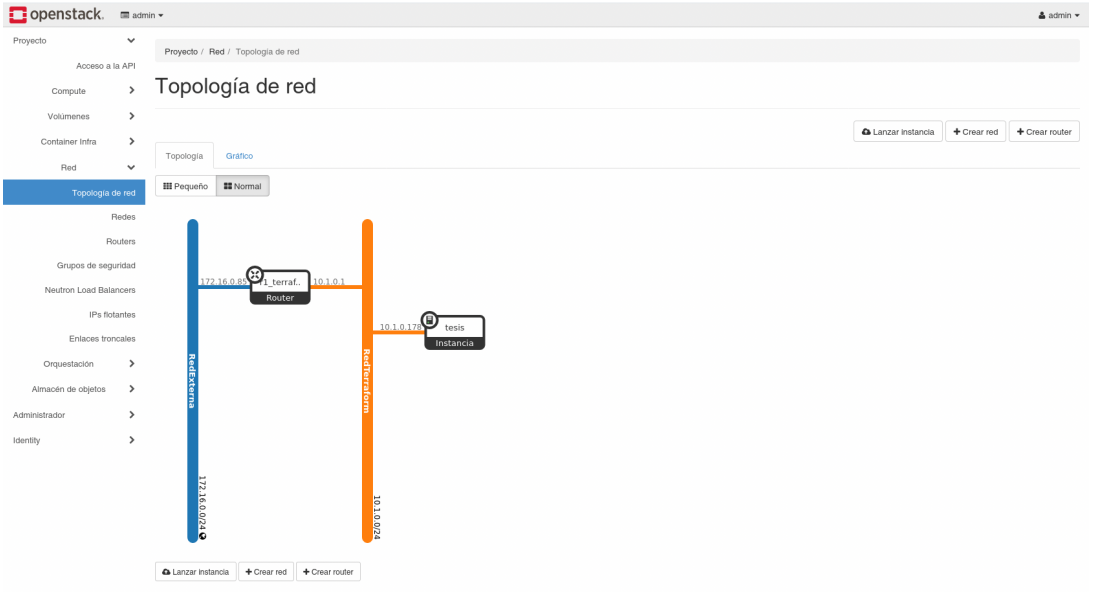


Figura 6.2

Topología de red en OpenStack

Con relación a otro recurso que se genero con el script es el grupos de seguridad, que permite el ingreso de ciertos puertos hacia las instancias.En la Figure 6.3 se muestra los puertos permitidos.

Figura 6.3

Grupo de seguridad en OpenStack.

Además se desplegó y se le asigno a la instancia un almacenamiento que en OpenStack a este recursos se le llama volumen, en el script de configuración de Terraform se le asigno un tamaño de 30 GB de almacenamiento. En la Figure 6.4 se muestra el volumen desplegado.

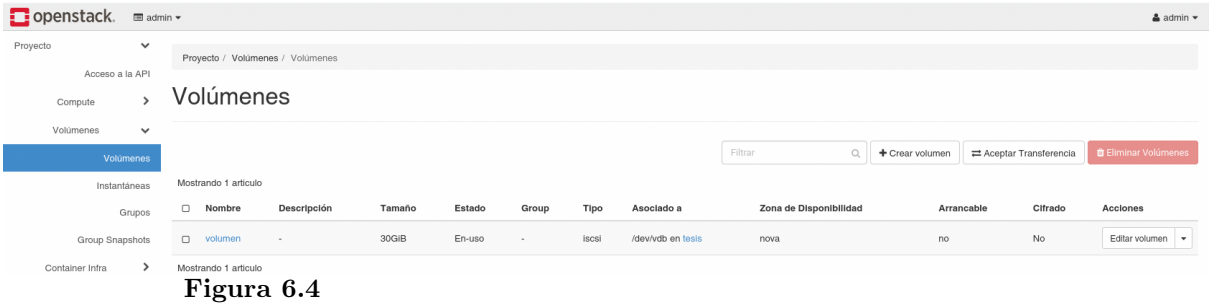


Figura 6.4

Volumen asignado a la instancia.

La Figure 6.5 se muestra las instancias creadas con el script de Terraform con las características especificadas en el mismo.

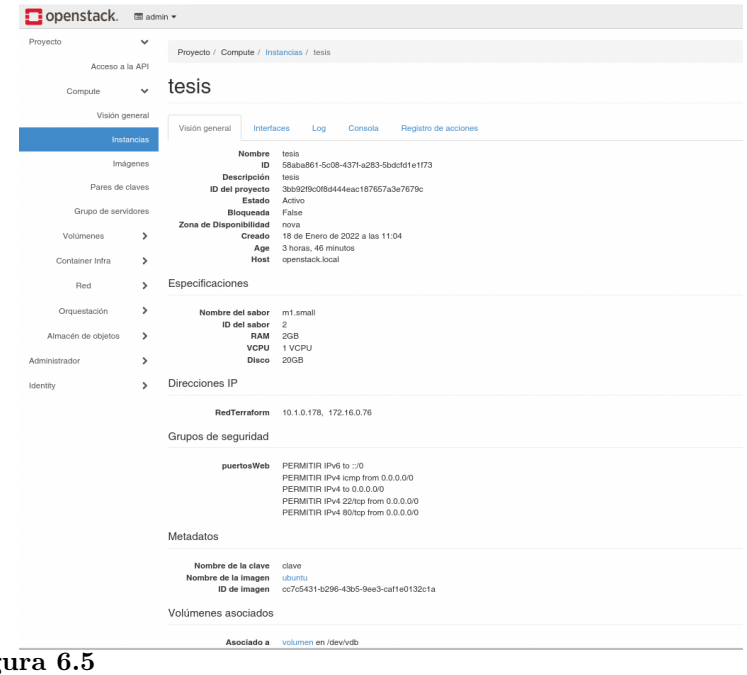


Figura 6.5

Características de la instancia.

6.2 Resultados del Escenario B

En el escenario B se presenta el desplegué de tres instancias las cuales se encuentran gestionadas por Ansible que en base a un script se instalo Docker con un contenedor de servidor Wordpress y MySQL. A continuación en la Figure 6.6 se presenta la topología desplegada con Terraform en la nube OpenStack .

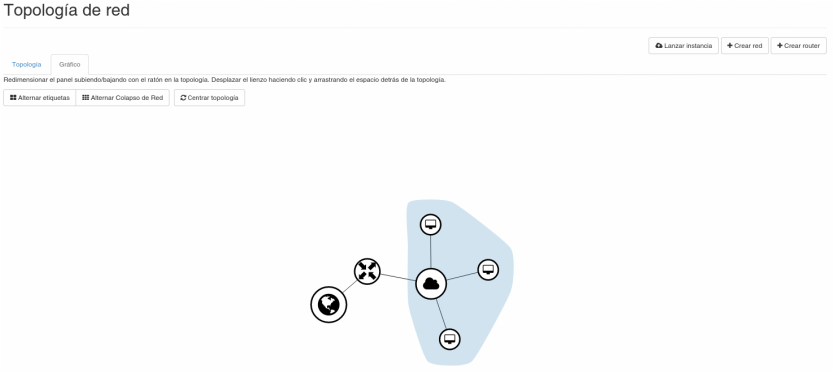


Figura 6.6

Topología desplegada en OpenStack.

Además en el script de configuración se genero la red virtual la misma que consta de la red privada que genera las diferentes direcciones IP para cada instancia, que se encuentra conectado con un router hacia la red externa que permite salir al internet, en la Figure 6.7 se muestra las diferentes direcciones IP asignadas para cada recurso de la red.

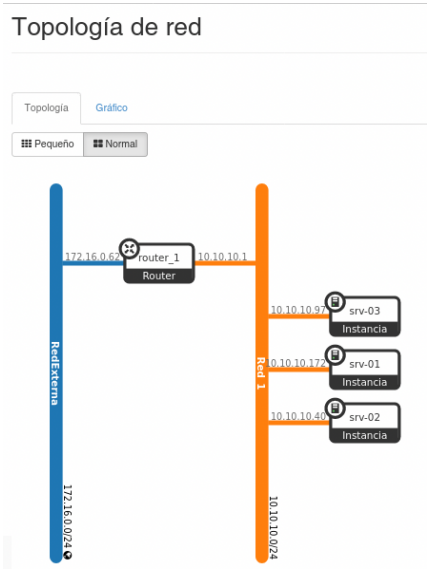


Figura 6.7

Topología desplegada en OpenStack.

Continuando con el desplegué se creo el almacenamiento para las tres instancias, el script de configuración de Terraform se definió asignar un volumen con un almacenamiento de 10 Gb, en la Figure 6.8 se muestra los volúmenes creados y asignados a las instancias.

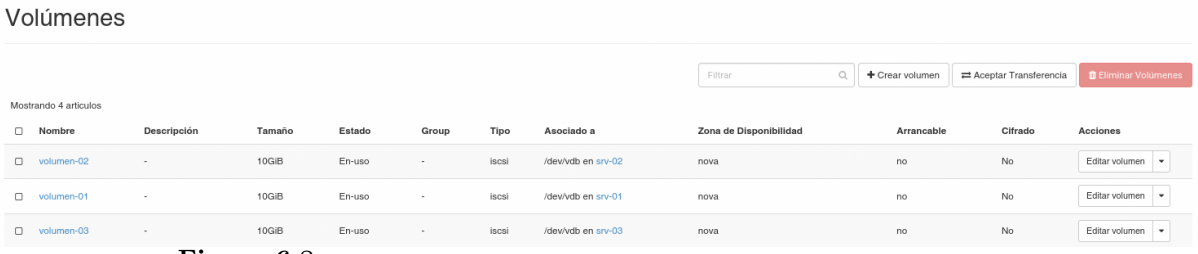


Figura 6.8

Volúmenes desplegados en OpenStack.

Para tener conexión con el internet se asigno direcciones de IP flotante a las instancias desplegadas en la Figure 6.9 se mustras las direcciones IP flotantes asignado a las instancias.

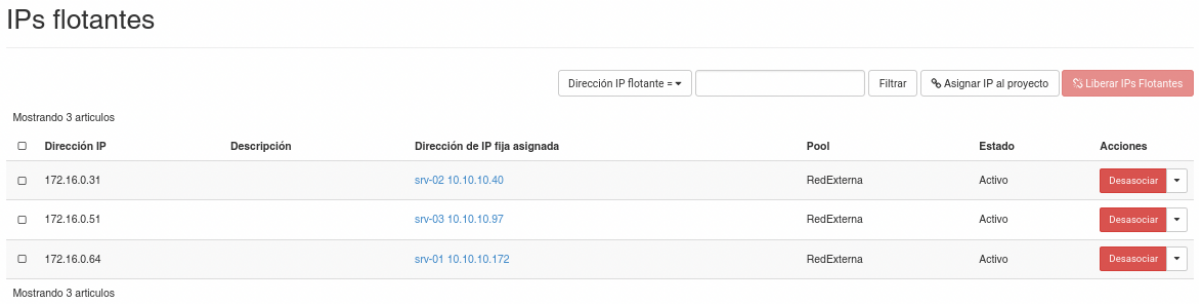


Figura 6.9

Direcciones de IPs flotante.

En la Figure 6.10 se presenta las instancias desplegadas en OpenStack con el script de configuración de Terraform, como se muestra la imagen del sistema operativo utilizada para la instancia,las direcciones IP, sabor que es la memoria RAM Y CPU, el nombre la clave publica utilizado.

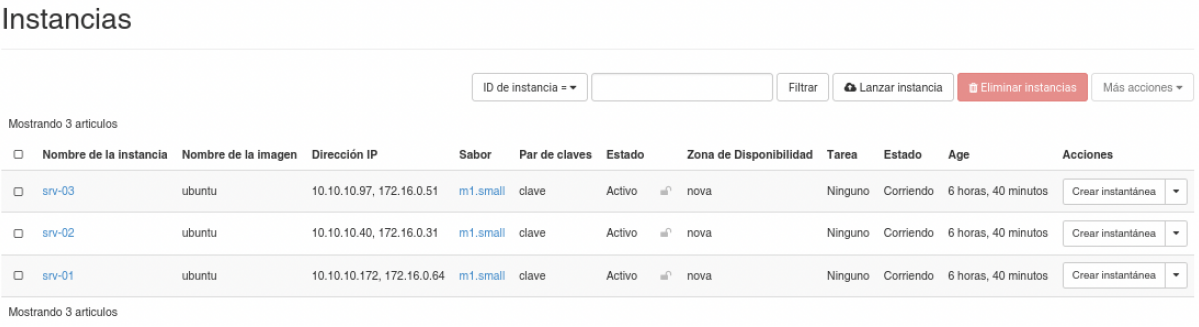


Figura 6.10

Instancias en OpenStack.

Para concluir en cuánto se refiere desplegué en la Figure 6.11 se muestra a detalle la características de las tres instancias que se desplegaron en la cloud OpenStack.

Figura 6.11

Características de las instancias.

Capítulo 7

Cronograma

En Table 7.1, se observa el cronogramas de actividades seguido para el desarrollo del proyecto.

Tabla 7.1

Cronograma de Actividad.

N°	**Actividad desarrollada**	**Fecha**	**Horas**
1	Estudio fundamentos de la infraestructura como Código (IaC).	06/04/2021	8 Horas
2	Estudiar las diferentes herramientas de laC.	08/04/2021	20 Horas
3	Estudio de las soluciones de laC.	12/04/2021	10 Horas
4	Estudio proceso de implementación de IaC.	16/04/2021	20 Horas
5	Estudiar técnicas de despliegue de infraestructura.	05/05/2021	8 Horas
6	Estudiar los fundamentos de OpenStack.	10/05/2021	12 Horas
7	Definir los requerimientos de hardware y software.	14/05/2021	12 Horas
8	Implementación de la nube privada con OpenStack.	18/05/2021	40 Horas
9	Diseño de una metodología para la implementación la parte técnica de IaC.	03/06/2021	45 Horas

Tabla 7.1

(Continuación)

N°	**Actividad desarrollada**	**Fecha**	**Horas**
10	Definir las variables necesarias para la creación de scripts.	21/06/2021	20 Horas
11	Instalación de Terraform en un servidor Linux.	28/06/2021	14 Horas
12	Elaboración y configuración de scripts que permitan la automatización en la Nube Privada.	05/07/2021	50 Horas
13	Implementación de la automatización para la creación de máquinas virtual en la nube privada.	02/08/2021	80 Horas
14	Estudiar las características de Docker y sus funcionalidades.	23/08/2021	40 Horas
15	Recolección de datos de los equipos de TI.	07/09/2021	40 Horas
16	Estudiar los métodos tradicionales de los equipos del TI.	15/09/2021	48 Horas
17	Especificar la gestión y administración de los equipos TI.	27/09/2021	40 Horas
18	Instalación de Docker en las maquinas creadas con Ansible.	11/10/2021	96 Horas
19	Implementar un control de versiones para el manejo más eficiente de los scripts.	22/11/2021	40 Horas

Tabla 7.1

(Continuación)

N°	**Actividad desarrollada**	**Fecha**	**Horas**
20	Prueba de los escenarios con Terraform.	29/11/2021	24 Horas
21	Prueba de la gestión de configuración con Ansible en las instanciadas creadas.	13/12/2021	30 Horas
22	Pruebas de funcionamiento de los Servicios con Docker.	27/12/2021	36 Horas
23	Elaboración del informe del proyecto de Titulación.	10/01/2022	96 Horas

- Horas totales de Wilmer Camas 400 horas.
- Horas totales de Xavier Riera 400 horas.

Capítulo 8

Presupuesto

En la Table 8.1, se detalla el presupuesto utilizado para el desarrollo del proyecto.

Tabla 8.1

Presupuesto aplicado para el desarrollo del proyecto.

DENOMINACIÓN	**CANT.**	**COSTO UNITARIO**	**COSTO TOTAL**
	Unidades	Dólares	Dólares
Bienes			
Copias	100	0.05	5.00
Servicios			
Servicio de transporte	130	1.50	195.00
Alimentación	130	2.50	325.00
Cursos de capacitación	4	25.00	100.00
Tecnológico			
Computadora portátil	2	1000	2000.00
Personal			
Estudiante investigador	2	1500.00	3000.00
Otros			
Imprevistos	1	150.00	150.00
TOTAL			5800.00

Conclusiones

Se desarrollo una metodología propia como soporte conceptual al procedimiento que se empleó para realizar esta tesis. Contando de cuatro fases: Fase de Diseño del aprovisionamiento, Fase De Codificación, Fase De Planificación, Fase De Aplicación.

La IaC apunta a automatizar lo máximo posible, ya que permite abstraer la infraestructura en forma de código y almacenarlo en un sistema de control de versiones. Sin duda con Terraform se puede automatizar procesos de configuración manual en la consola del proveedor de nube, aunque es importante integrarlo con otras herramientas como Ansible para gestionar la configuración de las instancias creadas. De esta forma, una organización dispone de varias herramientas para el aprovisionamiento y configuración de la infraestructura.

Se demostró que Terraform es una herramienta para poder crear recursos en la nube privada de OpenStack que se utilizó para esta investigación, ya que no hubo mayor complicación en realizar la parte práctica, debido a que en la documentación fue lo suficientemente clara para poder avanzar, además de concisa de todo lo que se desea saber acerca de usar Terraform para crear instancias en la nube.

Se desarrollo scripts con los requerimientos necesarios para levantar una infraestructura en la nube. Se automatizo la configuración de un servidor de Docker funcional y la instalación de paquetes. Cabe destacar que cada una de estas configuraciones es genérica, motivo por lo cual se puede aplicar directamente en un servidor recién instalado sin necesidad de realizar posteriores configuraciones manuales.

Para crear un servidor, se ejecutó la herramienta Terraform Cli. El lenguaje facilita la descripción exacta de la infraestructura que desea crear. El comando plan le permite verificar sus

cambios y capturar errores antes de implementarlos. Las variables, referencias y dependencias permiten le permite eliminar la duplicación de su código y hacerlo altamente configurable. Para implementar un proyecto de Terraform, primero debe escribir el código de configuración, luego configurar los proveedores y otras variables de entrada, inicializar Terraform y finalmente aplicar los cambios. La limpieza se realiza con un comando de destrucción.

Ansible proporciona su propio lenguaje declarativo para definir las actividades que se realizarán en el servidor creado, y la herramienta funciona sin un agente a través de conexiones remotas SSH. Utiliza archivos de texto de inventario configurables, así como libros de jugadas YAML para expresar la configuración. Para el caso especial de este estudio, es necesario administrar de manera efectiva los catálogos de Ansible y los scripts de configuración en hosts que ejecutan sistemas operativos basados en Linux.

De las pruebas realizadas se concluye que la implementación del enfoque IaC a través de las herramientas Terraform y Ansible ha influenciado de manera positiva en sistemas que empleen OpenStack como proveedor de nube privada, ayudando a disminuir los tiempos de despliegue y configuración.

De las pruebas de realizadas se concluye que la implementación del enfoque IaC ha influido de manera positiva al proceso de expansión de servidores y gestión de configuraciones de los componentes para sistemas de información basados en la nube privada de OpenStack.

Recomendaciones

Se recomienda ampliamente el uso de Terraform para crear recursos en la nube y sobre todo que se use en conjunto con Ansible ya que entre estas herramientas se complementan para agilizar la creación de toda una infraestructura que es lo que se busca, si se desea replicar el experimento de esta investigación se recomienda emplear el proveedor antes mencionado ya que es el que más facilidades le brinda al usuario.

Al momento de implementar los scripts con los diferentes requerimientos en las distintas topologías, se debería de llevar una documentación bien estructura y ordenada de todos los playbooks y plantillas. Una de las herramientas que podría utilizar para llevar dicha documentación es Visual Studio Code con la extensión YAML que brinda la facilidad de escribir código de manera sencilla y ordenada. Para implementar de la infraestructura en la nube privada de OpenStack, se recomienda muchos recursos de hardware para un óptimo funcionamiento.

Para el caso particular de este proyecto, fue necesario administrar efectivamente los directorios y los scripts de configuración de Terraform y Ansible en una estación de Trabajo que opera con un sistema operativo basado en Linux.

Para establecer una comunicación entre la infraestructura de nube privada de OpenStack con la herramienta Terraform, se recomienda agregar las credenciales como el usuario y contraseña, además de la dirección IP del servidor de OpenStack para poder desplegar los recursos deseados.

Se recomienda tomar en cuenta habilitar los puertos requeridos en la infraestructura de modo que, al exponer los servicios creados en las máquinas virtuales, sean accesibles desde el exterior de OpenStack.

Referencias

[1] Mike Chan. *Infrastructure as Code*. https://www.thorntech.com/infrastructureascodebenefits/. 2018.

[2] Carlos Schults. *What Is Infrastructure as Code? How It Works, Best Practices*. https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/. 2019.

[3] K. Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, 2016. ISBN: 9781491924358. URL: https://books.google.com.ec/books?id=kOnurQEACAAJ.

[4] Mostafa Noshy, Abdelhameed Ibrahim y Hesham Arafat Ali. «Optimization of live virtual machine migration in cloud computing: A survey and future directions». En: *Journal of Network and Computer Applications* 110 (2018), págs. 1-10. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1084804518300833.

[5] Chnar Mustafa Mohammed, Subhi RM Zebaree y col. «Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review». En: *International Journal of Science and Business* 5.2 (2021), págs. 17-30.

[6] Andrey Markelov. *Certified OpenStack Administrator Study Guide*. Springer, 2016.

[7] Hanan Shukur, Subhi Zeebaree, Rizgar Zebari, Diyar Zeebaree, Omar Ahmed y Azar Salih. «Cloud computing virtualization of resources allocation for distributed systems». En: *Journal of Applied Science and Technology Trends* 1.3 (2020), págs. 98-105.

[8] Peter Mell, Tim Grance y col. «The NIST definition of cloud computing». En: (2011).

[9] Christine Miyachi. «What is "Cloud"? It is time to update the NIST definition?» En: *IEEE Cloud computing* 5.03 (2018), págs. 6-11.

[10] Sabiyyah Sabir. «Security Issues in Cloud Computing and their Solutions: A Review». En: *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS* 9.11 (2018), págs. 343-346.

[11] Shane Callanan. «An Industry-Based Study on the Efficiency Benefits of Utilising Public Cloud Infrastructure and Infrastructure as Code Tools in the IT Environment Creation Process». En: (2018).

[12] Roberto Ernesto Guevara y col. «Servicios de cómputo en la nube (cloud computing).» En: (2018).

[13] Francisco Javier Ruiz del Olmo. «Conocimiento en la nube: características sociocomunicativas del cloud computing». En: *Razón y palabra* 73 (2010).

[14] Dan C Marinescu. *Cloud computing: theory and practice*. Morgan Kaufmann, 2017.

[15] Naresh Kumar Sehgal, Pramod Chandra P Bhatt y John M Acken. *Cloud Computing with Security*. Springer, 2020.

[16] Sunilkumar S Manvi y Gopal Krishna Shyam. «Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey». En: *Journal of network and computer applications* 41 (2014), págs. 424-440.

[17] Aaqib Rashid y Amit Chaturvedi. «Cloud computing characteristics and services: a brief review». En: *International Journal of Computer Sciences and Engineering* 7.2 (2019), págs. 421-426.

[18] Borja de Luque Yarza y Jorge Martínez García. «Diseño de una cloud privada basada en software OpenStack». En: (2017).

[19] Luis Joyanes Aguilar. «Computación en la nube: Notas para una estrategia española en cloud computing». En: *Revista del Instituto Español de Estudios Estratégicos* 00 (2012).

[20] Juan Aranda, Erwin J Sacoto-Cabrera, Daniel Haro-Mendoza y Fabián Astudillo-Salinas. «Redes 5G: una revisión desde las perspectivas de arquitectura, modelos de negocio, ciberseguridad y desarrollos de investigación». En: *Revista Digital Novasinergia* 4.1 (2021), págs. 6-41.

[21] Erwin Sacoto Cabrera. «Análisis basado en teoría de juegos de modelos de negocio de operadores móviles virtuales en redes 4G y 5G». Tesis doct. Universitat Politècnica de València, 2021.

[22] Erwin J Sacoto-Cabrera, Luis Guijarro, Jose R Vidal y Vicent Pla. «Economic feasibility of virtual operators in 5G via network slicing». En: *Future Generation Computer Systems* 109 (2020), págs. 172-187.

[23] Erwin J Sacoto-Cabrera, Angel Sanchis-Cano, Luis Guijarro, José Ramón Vidal y Vicent Pla. «Strategic interaction between operators in the context of spectrum sharing for 5g networks». En: *Wireless Communications and Mobile Computing* 2018 ().

[24] Erwin J Sacoto-Cabrera, Gabriel León-Paredes y Walter Verdugo-Romero. «LoRaWAN: Application of Nonlinear Optimization to Base Stations Location». En: *Communication, Smart Technologies and Innovation for Society*. Springer, 2022, págs. 515-524.

[25] Erwin J Sacoto Cabrera, Sonia Palaguachi, Gabriel A León-Paredes, Pablo L Gallegos-Segovia y Omar G Bravo-Quezada. «Industrial Communication Based on MQTT and Modbus Communication Applied in a Meteorological Network». En: *The International Conference on Advances in Emerging Trends and Technologies*. Springer. 2020, págs. 29-41.

[26] Victor Vimos y Erwin J Sacoto Cabrera. «Results of the implementation of a sensor network based on Arduino devices and multiplatform applications using the standard OPC UA». En: *IEEE Latin America Transactions* 16.9 (2018), págs. 2496-2502.

[27] Erwin Sacoto-Cabrera, Jorge Rodriguez-Bustamante, Pablo Gallegos-Segovia, Gabriela Arevalo-Quishpi y Gabriel León-Paredes. «Internet of Things: Informatic system for metering with communications MQTT over GPRS for smart meters». En: *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. IEEE. 2017, págs. 1-6.

[28] V Vimos, E Sacoto y DX Morales. «Conceptual architecture definition: Implementation of a network sensor using Arduino devices and multiplatform applications through OPC UA». En: *2016 IEEE International Conference on Automatica (ICA-ACCA)*. IEEE. 2016, págs. 1-5.

[29] OpenNebula Support Team. «DATASHEET OpenNebula Overview.» En: (2019). URL: https://support.opennebula.pro/hc/en-us/article_attachments/360017307377/OpenNebula_Overview-Rev20210226.pdf.

[30] Chao-Tung Yang, Jung-Chun Liu, Ching-Hsien Hsu y Wei-Li Chou. «On improvement of cloud virtual machine availability with virtualization fault tolerance mechanism». En: *The Journal of Supercomputing* 69.3 (2014), págs. 1103-1122.

[31] Cody Bumgardner. *OpenStack in action*. Simon y Schuster, 2016.

[32] Diego P Rodriguez Alvarado y Erwin J Sacoto-Cabrera. «Implementation and Analysis of the Results of the Application of the Methodology for Hybrid Multi-cloud Replication Systems». En: *The International Conference on Advances in Emerging Trends and Technologies*. Springer. 2021, págs. 273-286.

[33] Nelson R Rodríguez, María Antonia Murazzo, Susana Beatriz Chávez y Miguel José Guevara. «Arquitectura de cloud computing híbrida basada en tecnología open source». En: *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*. 2014.

[34] Egle Sigler Kevin Jackson Cody Bunch. *OpenStack Cloud Computing Cookbook Fourth Edition*. Packt Publishing Ltd, 2018.

[35] Uchit Vyas. *Applied OpenStack Design Patterns: Design solutions for production-ready infrastructure with OpenStack components*. Apress, 2016.

[36] Walter Bentley. *OpenStack Administration with Ansible 2*. 2.ª ed. Packt Publishing Ltd, 2016.

[37] Yurley Constanza Medina Cárdenas y Dewar Willmer Rico Bautista. «Modelo de gestión basado en el ciclo de vida del servicio de la Biblioteca de Infraestructura de Tecnologías de Información (ITIL)». En: *Revista Virtual Universidad Católica del Norte* 27 (2009), págs. 1-21.

[38] K.C. Laudon, J.P. Laudon y A.V.R. Elizondo. *Sistemas de información gerencial*. Área: Computación. Pearson Educación, 2016. ISBN: 9786073236966. URL: https://books.google.com.ec/books?id=61s5ngAACAAJ.

[39] RedHat. *What is IT Infrastructure?* https://www.redhat.com/es/topics/cloud-computing/what-is-it-infrastructure. 2019.

[40] IBM. *What is IT Infrastructure?* https://www.ibm.com/topics/infrastructure. 2020.

[41] O Mejıa. «Computación en la nube». En: *ContactoS* 80 (2011), págs. 45-52.

[42] Matías David Moglia. «Infraestructura hiperconvergente». Tesis doct. Universidad Nacional de La Plata, 2020.

[43] Nils Urbach y Frederik Ahlemann. «Infrastructure as Commodity: IT Infrastructure Services Are Traded on Free Markets and Purchased as Required». En: *IT Management in the Digital Age: A Roadmap for the IT Department of the Future*. Cham: Springer International Publishing, 2019, págs. 75-84. DOI: 10.1007/978-3-319-96187-3_8. URL: https://doi.org/10.1007/978-3-319-96187-3_8.

[44] Carlos Martino Ojeda. «Automatización de infraestructura IT con IaC». En: (2020).

[45] Kief Morris. *Infrastructure as Code*. 2.ª ed. O'Reilly Media, 2021.

[46] Pierluigi Riti y David Flynn. *Beginning HCL Programming*. Springer, 2021.

[47] Ritesh Modi. «Infrastructure as Code». En: *Deep-Dive Terraform on Azure*. Springer, 2021, págs. 1-8.

[48] Kief Morris. *Infrastructure as code*. O'Reilly Media, 2020.

[49] Navin Sabharwal, Sarvesh Pandey y Piyush Pandey. «Getting Started with HashiCorp Automation Solutions». En: (2021), págs. 1-10. DOI: 10.1007/978-1-4842-7129-2_1. URL: https://doi.org/10.1007/978-1-4842-7129-2_1.

[50] John Alexander Galeano Ospina. «Línea base de infraestructura en la nube». En: (2020).

[51] Luis Dominguez-Quintero y Miguel Vargas-Lombardo. «Herramientas de infraestructura como código: ansible, terrafom, chef, puppet». En: *I+ D Tecnológico* 17.2 (2021).

[52] Angel Francisco Ybarhuen Manrique. «Influencia del enfoque de infraestructura como código en sistemas de información basados en la nube de Amazon Web Services». En: (2018).

[53] Yevgeniy Brikman. *Terraform: Up & Running: Writing Infrastructure as Code*. O'Reilly Media, 2019.

[54] Wlodzimierz Gajda. *Pro Vagrant*. Apress, 2015.

[55] Asaf Yigal. *Chef vs. Puppet: Methodologies, Concepts, and Support*. 2017. URL: https://logz.io/blog/chef-vs-puppet/.

[56] HashiCorp. *Deliver Infrastructure as Code*. URL: https://www.terraform.io/.

[57] Kirill Shirinkin. *Getting Started with Terraform*. Packt Publishing Ltd, 2017.

[58] Ritesh Modi. «CI/CD with Terraform». En: *Deep-Dive Terraform on Azure: Automated Delivery and Deployment of Azure Solutions*. Berkeley, CA: Apress, 2021, págs. 163-190. ISBN: 978-1-4842-7328-9. DOI: 10.1007/978-1-4842-7328-9_7. URL: https://doi.org/10.1007/978-1-4842-7328-9_7.

[59] Bradley Campbell. «Terraform In-Depth». En: *The Definitive Guide to AWS Infrastructure Automation : Craft Infrastructure-as-Code Solutions*. Berkeley, CA: Apress, 2020, págs. 123-203. DOI: 10.1007/978-1-4842-5398-4_4. URL: https://doi.org/10.1007/978-1-4842-5398-4_4.

[60] Oscar Medina y Ethan Schumann. «Provisioning the SharePoint Farm to Azure Using Terraform». En: *DevOps for SharePoint: With Packer, Terraform, Ansible, and Vagrant*. Berkeley, CA: Apress, 2018, págs. 85-153. DOI: 10.1007/978-1-4842-3688-8_4. URL: https://doi.org/10.1007/978-1-4842-3688-8_4.

[61] Alberto GONZÁLEZ RODRÍGUEZ. *ANSIBLE Automatización para todos*. RC Libros, 2018.

[62] Nathan Hull. *Starting Ansible Easy guide for beginners*. 2016.

[63] Michael Heap. «Getting Started». En: *Ansible: From Beginner to Pro*. Berkeley, CA: Apress, 2016, págs. 1-17. ISBN: 978-1-4842-1659-0. DOI: 10.1007/978-1-4842-1659-0_1. URL: https://doi.org/10.1007/978-1-4842-1659-0_1.

[64] myservername. *11 Best Software Configuration Management Tools*. URL: https://es.myservername.com/11-best-software-configuration-management-tools.

[65] Jesse Keating. *Mastering Ansible*. Packt Publishing Ltd, 2015.

[66] Waqas Irtaza. *IT Infrastructure Automation Using Ansible: Guidelines to Automate the Network, Windows, Linux, and Cloud Administration*. BPB Publications, 2022.

[67] Lorin Hochstein y Rene Moser. *Ansible: Up and Running: Automating configuration management and deployment the easy way*. " O'Reilly Media, Inc.", 2017.

[68] Andrew Mallett. «Working with the Ansible Configuration». En: (2021). DOI: 10.1007/978-1-4842-6861-2_2. URL: https://doi.org/10.1007/978-1-4842-6861-2_2.

[69] Madhurranjan Mohaan y Ramesh Raithatha. *Learning Ansible*. Packt Publishing Ltd, 2014.

[70] Gourav Shah. *Ansible Playbook Essentials*. Packt Publishing Ltd, 2015.

[71] HY Vani y col. «Building, Deploying and Validating a Home Location Register (HLR) using Jenkins under the Docker and Container Environment». En: *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE. 2020, págs. 1278-1282.

[72] Sean P Kane y Karl Matthias. *Docker: Up & Running: Shipping Reliable Containers in Production*. O'Reilly Media, 2018.

[73] Kunchala Vikram. *Introduction to Docker & Docker Swarm*. URL: https://github.com/kunchalavikram1427/Docker_public/blob/master/Cheatsheets/docker_reference_01.pdf.

[74] Pramod Singh. «Machine Learning Deployment Using Docker». En: *Deploy Machine Learning Models to Production: With Flask, Streamlit, Docker, and Kubernetes on Google Cloud Platform*. Berkeley, CA: Apress, 2021, págs. 91-126. ISBN: 978-1-4842-6546-8. DOI: 10.1007/978-1-4842-6546-8_4. URL: https://doi.org/10.1007/978-1-4842-6546-8_4.

[75] Darío Dosantos Moreno. «Virtualización de una aplicación web para su uso con contenedores docker». Tesis de mtría. 2017.

[76] María Cabrera Gómez de la Torre. «GESTIÓN DE CONTENEDORES DOCKER». En: ().

[77] Joshua Cook. *Docker for Data Science: Building Scalable and Extensible Data Infrastructure Around the Jupyter Notebook Server*. Apress, 2017.

[78] Jeffrey Nickoloff y Stephen Kuenzli. *Docker in action*. Simon y Schuster, 2019.

[79] Alexander Kropp y Roberto Torre. «Docker: containerize your application». En: *Computing in Communication Networks*. Elsevier, 2020, págs. 231-244.

[80] Karl Matthias y Sean P Kane. *Docker: Up & Running: Shipping Reliable Containers in Production*. " O'Reilly Media, Inc.", 2015.

[81] Michael Hüttermann. *DevOps for developers*. Apress, 2012.

[82] Francisco J Lopez-Pellicer, Rubén Béjar, Miguel A Latre, Javier Nogueras-Iso y Francisco Javier Zarazaga-Soria. «GitHub como herramienta docente». En: *Actas de las XXI Jornadas de la Enseñanza Universitaria de la Informática*. Universitat Oberta La Salle. 2015, págs. 66-73.

[83] Rosabel Roig-Vila, Jordi M Antolí Martínez, Josefa Eugenia Blasco Mira, Asunción Lledó Carreres y Neus Pellín Buades. «Redes colaborativas en torno a la docencia universitaria». En: (2017).

[84] Scott Chacon y Ben Straub. *Pro git*. Springer Nature, 2014.

Anexos

Anexo 1. Instalación de OpenStack.

En esta sección se indica paso a paso la instalación de OpenStack. Características de MV:

- Centos 7
- 12 Gb de memoria ram
- 4 procesadores

1. Para comenzar la instalación de OpenStack se actualiza el S.O, con los siguientes comandos se actualiza los repositorios y dependencias.

   ```
   yum -y update
   ```

   ```
   yum -y upgrade
   ```

2. Se asigna una ip estática a la maquina virtual, para ello ingresamos al archivo ifcfg-enp0s3 que se encuentra en `/etc/sysconfig/network-scripts/ifcfg-enp0s3`. También se lo puede

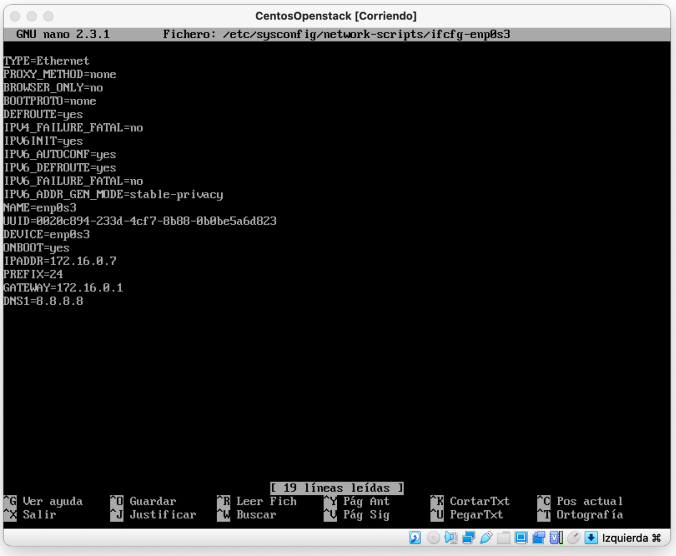


Figura 8.1

Configuración de la ip estática en la MV.

hacer la configuración con el comando nmtui.

3. Agrega el nombre de host con `nano /etc/hostname` , si no se la coloca a la hora de la instalación de S.O. En este caso el nombre de host es openstack.local

Figura 8.2

Configuración hostname

4. Coloca la IP y host en el archivo `/etc/hosts`, esto se lo realizo para que resuelva tanto por la dirección IP así como por el nombre de host.
5. Se para y desactiva el firewalld, se lo hace para mas adelante no se presente errores en la instalación.

   ```
   systemctl stop firewalld

   systemctl disable firewalld

   systemctl stop NetworkManager

   systemctl disable NetworkManager
   ```

6. Deshabilitamos el selinux, se remplaza en enforcing por disabled como se muestra en la figura.

   ```
   nano /etc/selinux/config
   ```

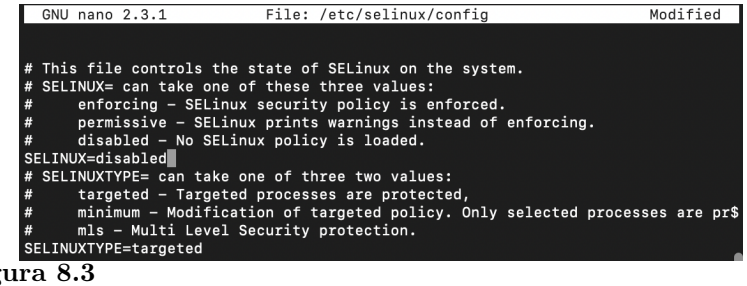


Figura 8.3

Deshabilitar selinux.

7. Luego de deshabilitar selinux se debe reiniciar con le comando `reboot`

8. Ahora se descarga los paquetes para la instalación de OpenStack Packstack, la versión de stein desde los repositorios de centos. Lo siguiente es actualizar los repositorios para que tome este nuevo repositorio y finamente reiniciar la M.V.

   ```
   yum install -y centos-release-openstack-stein
   yum -y update
   reboot
   ```

9. Ejecutaremos el siguiente comando para la instalación de OpenStack Packstack.Y mandaremos a actualizar los repositorios por si tenemos paquetes por actualizar.

   ```
   yum install -y openstack-packstack
   yum -y update
   ```

10. Finalmente se instala los componentes de OpenStack packstack, que consiste en instalar todos los componentes en un solo nodo, este no se instalara con el pryecto demo de OpenStack, pero contara con heat que sirve para el desplegué de orquestación de contenedores, la instalación dura alrededor de unos 25 minutos.

    ```
    packstack --allinone --os-neutron-l2-agent=openvswitch
    --os-neutron-ml2-mechanism-drivers=openvswitch
    --os-neutron-ml2-tenant-network-types=vxlan
    --os-neutron-ml2-type-drivers=vxlan,flat,vlan --provision-demo=n
    --os-neutron-ovs-bridge-mappings=extnet:br-ex
    --os-neutron-ovs-bridge-interfaces=br-ex:enp0s3
    --os-heat-install=y --os-magnum-install=y
    ```

    Cuando finalice la instalacion tendremos una mensaje de que fue instalado correctamente como se muestra en la siguiente figura.

    Además se presentara información adicional en la nos indicara la dirección ip a la que para

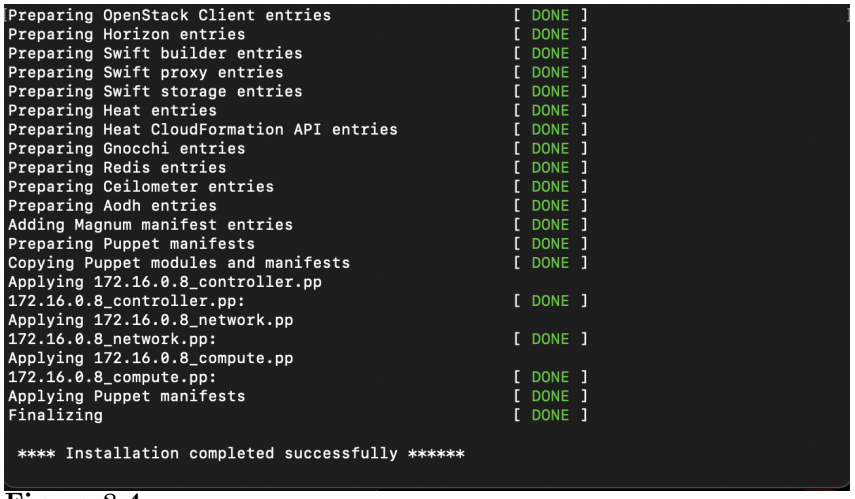


Figura 8.4

Instalación Openstack Packstack.

acceder al dashboard.

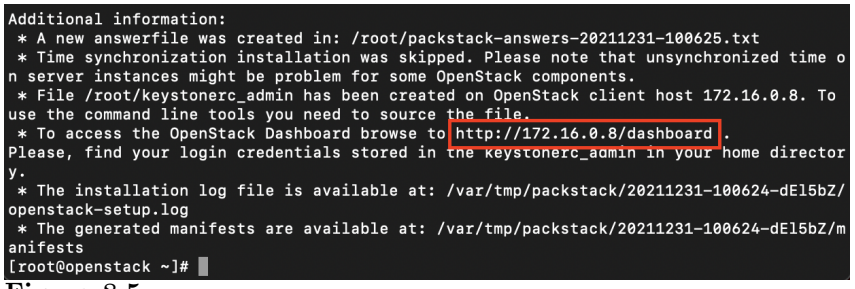


Figura 8.5

Instalación Openstack Packstack.

Se puede ingresar también de linea de comandos que es OpenStack Cli

`source keystonerc_admin`

11. Un vez finalizada la instalación, para ingresar a la interfaz de OpenStack necesitamos un usuario y contraseña, para ello ingresaremos a `cat keystonerc_admin` en donde se encuentran dichos datos.
12. Una vez finalizado la instalación, crearemos una red externa que nos permite tener conexión con internet, para ello creamos un red tipo flat, ingresaremos a OpenStack cli.

    `source keystonerc_admin`

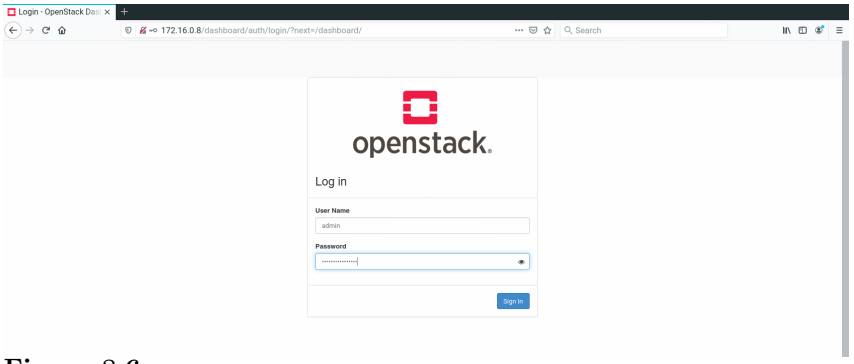


Figura 8.6

Interfaz gráfica de OpenStack.

Crear la red flat:

```
neutron net-create RedExterna --provider:network_type flat
--provider:physical_network extnet --router:external
```

El siguiente comando crea una subred que se le asignara a la red creada, la subred debe tener la ip y gateway de la red local si no tendrá una conexión con las instancias de OpenStack.

```
neutron subnet-create --name public_subnet --enable_dhcp=False
--allocation-pool=start=172.16.0.10,end=172.16.0.200 --gateway=172.16.0.1
RedExterna 172.16.0.0/24
```

Anexo 2. Variables de entorno.

Terraform utiliza variables de entorno para ingresar variables de forma predeterminada y el mismo este a disposición en cualquier configuración que requiera Terraform. Para que una variable de entornó la tome Terraform se antepone el prefijo TF_VAR_ seguido de el nombre deseado.

A continuación se generan las variables de entorno para el la URL, usuario, contraseña de OpenStack:

1. Se ingresa al Shell donde se instalo OpenStack e ingresamos el comando `cat keystonerc_admin`

para visualizar los datos de OpenStack que necesitamos para las variables de entorno.

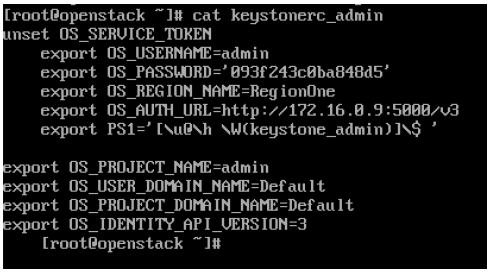


Figura 8.7

Resultado del comando `cat keystonerc_admin`.

2. En el terminal configuramos las variables con los datos ya descritos:

Figura 8.8

Comandos para las variables de entorno.

Anexo 2. Llaves de acceso

Para la generar las llave privada y publica para la conexión SSH se utiliza el comando `ssh-keygen`.

Las llaves se encuentra en el directorio `/home/tesis/.ssh/`

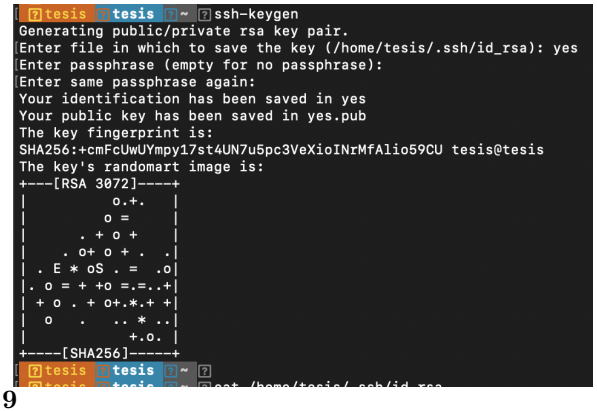


Figura 8.9

Resultado del comando `ssh-keygen`.

Figura 8.10

Directorio de las llaves.

Una vez generado las llaves para SSH, del equipo con el que se va conectar a OpenStack, mas adelante es necesario para realizar la conexión a la instancia, para ello importaremos el archivo con la llave publica nos dirigimos a la pestaña Proyecto >Computo >Pares Claves >Importar clave publica que se encuentra en el directorio `/home/tesis/.ssh/`.

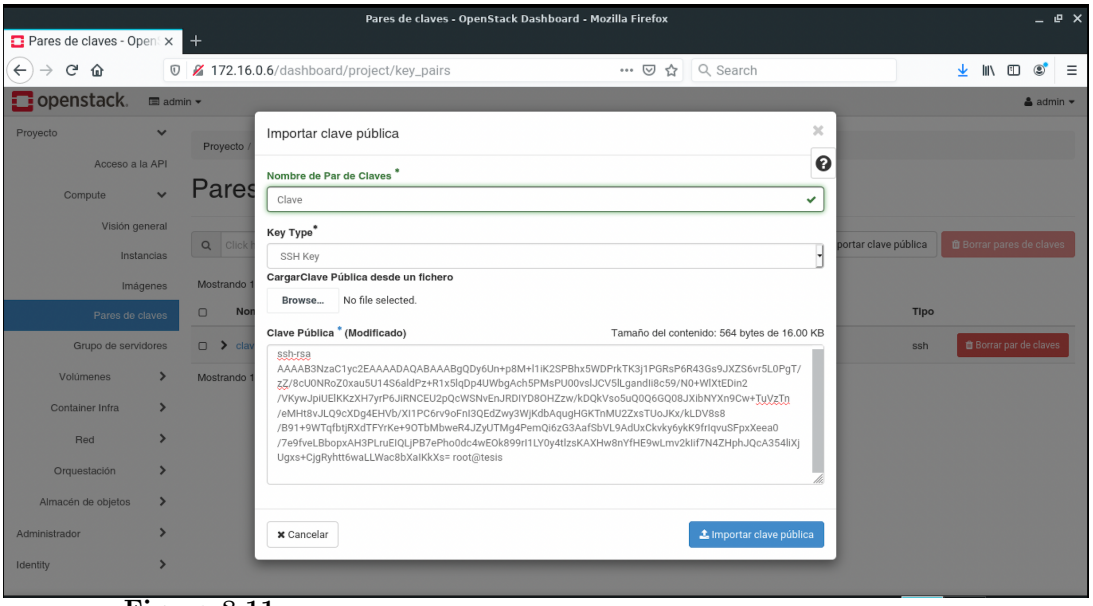


Figura 8.11

Importar clave publica en OpenStack.

Anexo 3. Instalación de Terraform

Para la instalación de terraform se utilizo distribución Linux Lite basada en Ubuntu, ya que HashiCorp mantiene y firma oficialmente paquetes de Terraform.

1. Primero se actualiza el S.O y que tenga instalado los paquetes gnupg, software-properties-common y curl con el siguiente comando:

   ```
   sudo apt-get update

   sudo apt-get install -y gnupg software-properties-common curl
   ```

2. Se añade la llave HashiCorp GPG

   ```
   curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
   ```

3. Se añade el repositorio de HashiCorp Linux

   ```
   sudo apt-add-repository ‘‘deb [arch=amd64] https://apt.releases.hashicorp.

   com \$(lsb_release -cs) main’’
   ```

4. Finalmente se actualiza y se instala CLI de Terraform.

   ```
   sudo apt-get update

   sudo apt-get install terraform
   ```

Anexo 4. Instalación de Ansible

La instalación se la realizo en distribución Linux Lite basada en Ubuntu, que es la maquina que es la maquina que funcionara como nodo central para Ansible.

1. En la maquina que funciona como nodo central se descarga los paquetes oficiales de Ansible.

   ```
   sudo apt-add-repository ppa:ansible/ansible
   ```

2. Se procede a actualizar los paquetes.

   ```
   sudo apt update
   ```

3. Y finalmente se instala Ansible

   ```
   sudo apt install ansible
   ```