UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"OPTIMIZACIÓN DEL RENDIMIENTO EN IMPLEMENTACIONES DE INFRAESTRUCTURA COMO SERVICIO (IAAS) UTILIZANDO OPENSTACK"

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética)

AUTOR:

Raúl Andrés Díaz Morillo

DIRECTOR:

Msc. Carlos Alberto Vásquez Ayala

Ibarra, Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	0401329073		
APELLIDOS Y NOMBRES:	Díaz Morillo Raúl Andrés		
DIRECCIÓN:	Tulcán N2-77 y 9 de agosto		
EMAIL:	radiazm@utn.edu.ec / andreezrr@gmail.com		
TELÉFONO FIJO:	2827673	**TELÉFONO MÓVIL:**	0998048963

DATOS DE LA OBRA	
TÍTULO:	OPTIMIZACIÓN DEL RENDIMIENTO EN IMPLEMENTACIONES DE INFRAESTRUCTURA COMO SERVICIO (IAAS) UTILIZANDO OPENSTACK
AUTOR (ES):	DÍAZ MORILLO RAÚL ANDRÉS
FECHA: DD/MM/AAAA	05/02 /2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR:	MSC. CARLOS ALBERTO VÁSQUEZ AYALA
ASESOR:	MSC. FABIÁN GEOVANNY CUZME ROGRIGUEZ

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 05 días del mes de febrero del 2026.

EL AUTOR:

Raúl Andrés Díaz Morillo

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 05 de febrero del 2026

MSC. CARLOS ALBERTO VÁSQUEZ AYALA

DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f)……

Msc. Carlos Alberto Vásquez Ayala

C.C.: 1002424982

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DEDICATORIA

Este trabajo está dedicado a mis padres, por su apoyo incondicional brindado a lo largo de toda mi formación académica y personal, por el esfuerzo constante, la confianza depositada y por impulsarme siempre a seguir adelante incluso en los momentos más difíciles.

De igual manera, dedico este logro a mi hermano, quien ha sido un apoyo permanente, ofreciéndome ayuda, comprensión y paciencia en cada etapa de mi proceso, convirtiéndose en un pilar fundamental durante mi desarrollo académico.

Asimismo, dedico este trabajo a mis amigos "The Fuck", "El Alpha", "La Señora", "Manuelito B", "More", "Don Raymi", entre otros, quienes con su compañía, apoyo y buen ánimo lograron que los momentos difíciles de mi camino universitario fueran más llevaderos y enriquecedores.

Finalmente, dedico este trabajo a Alejandra, quien con su paciencia, apoyo y comprensión ha contribuido de manera significativa a su crecimiento personal, motivándome a mejorar constantemente y a afrontar los retos con mayor madurez y compromiso.

Díaz Morillo Raúl Andrés

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

AGRADECIMIENTO

Expreso mi sincero agradecimiento a la Universidad Técnica del Norte y a la carrera de Ingeniería en Telecomunicaciones por la formación académica brindada a lo largo de mi etapa universitaria, así como a todos los docentes que contribuyeron a mi desarrollo profesional. De manera especial agradece a mi tutor MSc. Carlos Vásquez por su guía constante apoyo académico y orientaciones oportunas durante el desarrollo del presente trabajo de titulación, así como a mi asesor MSc. Fabián Cuzme por el acompañamiento la disposición y las recomendaciones técnicas que permitieron fortalecer y culminar satisfactoriamente este proyecto.

Díaz Morillo Raúl Andrés

RESUMEN EJECUTIVO

El trabajo de titulación se enfoca en la optimización del rendimiento de implementaciones de Infraestructura como Servicio IaaS mediante el uso de la plataforma OpenStack con el objetivo de mejorar la gestión de recursos de cómputo red y almacenamiento en un entorno virtualizado de carácter académico. El proyecto parte del análisis del estado inicial de la infraestructura utilizada en prácticas universitarias donde se identifican limitaciones asociadas al control del tráfico la asignación de recursos y la ausencia de mecanismos de automatización y calidad de servicio a partir de lo cual se diseña e implementa una nube privada basada en OpenStack desplegada bajo una topología de nodo único utilizando DevStack como método de instalación.

Durante la implementación se configuran los principales servicios de OpenStack permitiendo la creación y administración de instancias virtuales redes definidas por software y almacenamiento además se aplican estrategias de optimización orientadas al control de recursos la gestión del tráfico mediante políticas de calidad de servicio y el monitoreo del rendimiento. Los resultados obtenidos evidencian un uso más eficiente de los recursos una reducción de la latencia y una mayor estabilidad del entorno virtualizado aportando finalmente una guía práctica que fortalece la comprensión y aplicación de conceptos de IaaS OpenStack y optimización del rendimiento en la formación académica del estudiante.

Palabras clave: Infraestructura como Servicio, OpenStack, DevStack, Optimización del rendimiento, Computación en la nube, QoS.

ABSTRACT

This degree project focuses on performance optimization in Infrastructure as a Service IaaS implementations using the OpenStack platform with the aim of improving the management of computing network and storage resources within a virtualized environment oriented to academic use. The project development includes an analysis of the initial state of the infrastructure used for university practices identifying limitations related to traffic control resource allocation and the absence of automation and quality of service mechanisms Based on this diagnosis a private cloud architecture based on OpenStack is designed and implemented under a single node topology using DevStack as the deployment method.

During the implementation process the main OpenStack services were configured enabling the creation and management of virtual instances software defined networks and storage Performance optimization strategies were applied focusing on resource control network traffic management through quality of service policies and performance monitoring using analysis tools. The obtained results demonstrate an improvement in efficient resource usage reduced latency in simulated traffic and greater stability of the virtualized environment Finally this project provides a practical guide that allows students to better understand and apply concepts related to IaaS OpenStack and performance optimization strengthening their academic and technical training.

Keywords: Infrastructure as a Service, OpenStack, DevStack, Performance Optimization, Cloud Computing, Quality of Service (QoS).

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

ÍNDICE DE CÓDIGO

CAPÍTULO I: Antecedentes

1.1. Tema

OPTIMIZACIÓN DEL RENDIMIENTO EN IMPLEMENTACIONES DE INFRAESTRUCTURA COMO SERVICIO (IAAS) UTILIZANDO OPENSTACK.

1.2. Problema

En la actualidad, el uso de la infraestructura como servicio (IaaS) en la nube ha aumentado mucho porque permite tener recursos informáticos que se pueden adaptar fácilmente y escalar según sea necesario. Sin embargo, aunque este modelo tiene muchas ventajas, no se presta mucha atención a cómo optimizarlo y hacerlo más eficiente (Conzultek, n.d.).

La mayoría de las soluciones de IaaS que hay en el mercado se concentran en ofrecer espacio para alojar aplicaciones, pero no se preocupan mucho por mejorar la forma en que se maneja el tráfico de datos ni por garantizar una buena calidad del servicio. Esta falta de atención a la optimización del tráfico hace que los recursos no se usen de la mejor manera, lo que resulta en costos más altos o perdida de ellos y una experiencia de usuario no muy amigable (Brismark, 2020).

Además, en el ámbito educativo, existe una falta de cubrir algunos temas. En la carrera de telecomunicaciones hasta el séptimo nivel no se profundiza en cómo funcionan los servicios virtuales, el cloud computing o cómo optimizar la infraestructura en la nube. Esto hace que no se cubran algunos temas requeridos por las empresas, que se dedican a gestionar y mejorar la infraestructura en la nube de manera eficiente.

Por eso, es importante investigar y encontrar formas de hacer que las implementaciones de IaaS sean más eficientes, especialmente utilizando tecnologías como OpenStack, que ofrecen formas flexibles y modulares de gestionar la infraestructura en la nube (García, 2020).

En este anteproyecto, se responde a la pregunta: ¿Cómo hacer que las implementaciones de IaaS sean más eficientes utilizando OpenStack para gestionar el tráfico, mejorar la calidad del servicio y reducir los costos operativos?

1.3. Objetivos

1.3.1. Objetivo General

Optimizar el rendimiento aplicando técnicas de gestión del tráfico, calidad del servicio (QoS) y reducción de costos operativos, en implementaciones de infraestructura como servicio (IaaS) utilizando OpenStack.

1.3.2. Objetivos Específicos

- Realizar un estudio del estado actual de las implementaciones de Infraestructura como Servicio (IaaS) en la nube, identificando las limitaciones y áreas de mejora en términos de optimización y eficiencia.
- Establecer un diseño optimizado de infraestructura como servicio (IaaS) utilizando OpenStack, basado en los requerimientos específicos de hardware y software, con el fin de generar nodos de comunicación que permitan prácticas efectivas.
- Desplegar una infraestructura como servicio basado en OpenStack utilizando la metodología en cascada para optimizar el rendimiento en entornos de IaaS, cumpliendo con requisitos específicos para prácticas en SDN.

- Realizar pruebas de rendimiento y guías que permitan a los estudiantes poder realizar prácticas sobre temáticas de optimización del rendimiento en implementaciones de infraestructura como servicio (IaaS), permitiendo la comprensión de conceptos y generar un criterio.

1.4. Alcance

El proyecto propuesto se enfocará en optimizar el rendimiento en implementaciones de infraestructura como servicio (IaaS) utilizando OpenStack. El objetivo es mejorar la eficiencia y la escalabilidad de la infraestructura, centrándose en la optimización de recursos y la maximización del rendimiento de los servicios proporcionados. El proceso seguirá una metodología en cascada, mediante el establecimiento de cinco fases para el desarrollo y cumplimiento del objetivo general, considerando las siguientes:

En primera instancia se estableció la fase de planificación, donde se va a llevar a cabo la recolección de requisitos de rendimiento, en donde se identificará los requisitos específicos de rendimiento para la infraestructura de IaaS con OpenStack, considerando métricas como capacidad de procesamiento, latencia de red y utilización de recursos. También se buscará la definición de objetivos de rendimiento, en donde se realizará el establecimiento de objetivos claros de rendimiento que cumplan con los requisitos identificados, con métricas cuantificables y alcanzables para evaluar el rendimiento de la infraestructura.

Como siguiente fase, tenemos la de diseño. Veremos el diseño de arquitectura para optimización del rendimiento, donde empieza la elaboración de una arquitectura de infraestructura basada en OpenStack que permita la optimización del rendimiento, considerando la distribución de recursos, la escalabilidad y la redundancia. También veremos la selección de componentes y configuración, aquí iniciamos con la selección de los componentes de OpenStack más adecuados para cumplir con los objetivos de rendimiento establecidos, y configuración de parámetros para optimizar el rendimiento.

Seguimos con la fase de implementación. Iniciamos con el despliegue de la infraestructura, veremos la implementación de la infraestructura de OpenStack según el diseño establecido, asegurando una configuración correcta y consistente, y automatizando el proceso de despliegue para mejorar la eficiencia. Aquí también nos toparemos con la configuración de servicios, entramos con la configuración de los servicios de OpenStack, especialmente los relacionados con el rendimiento, como la asignación de recursos y la gestión de la red.

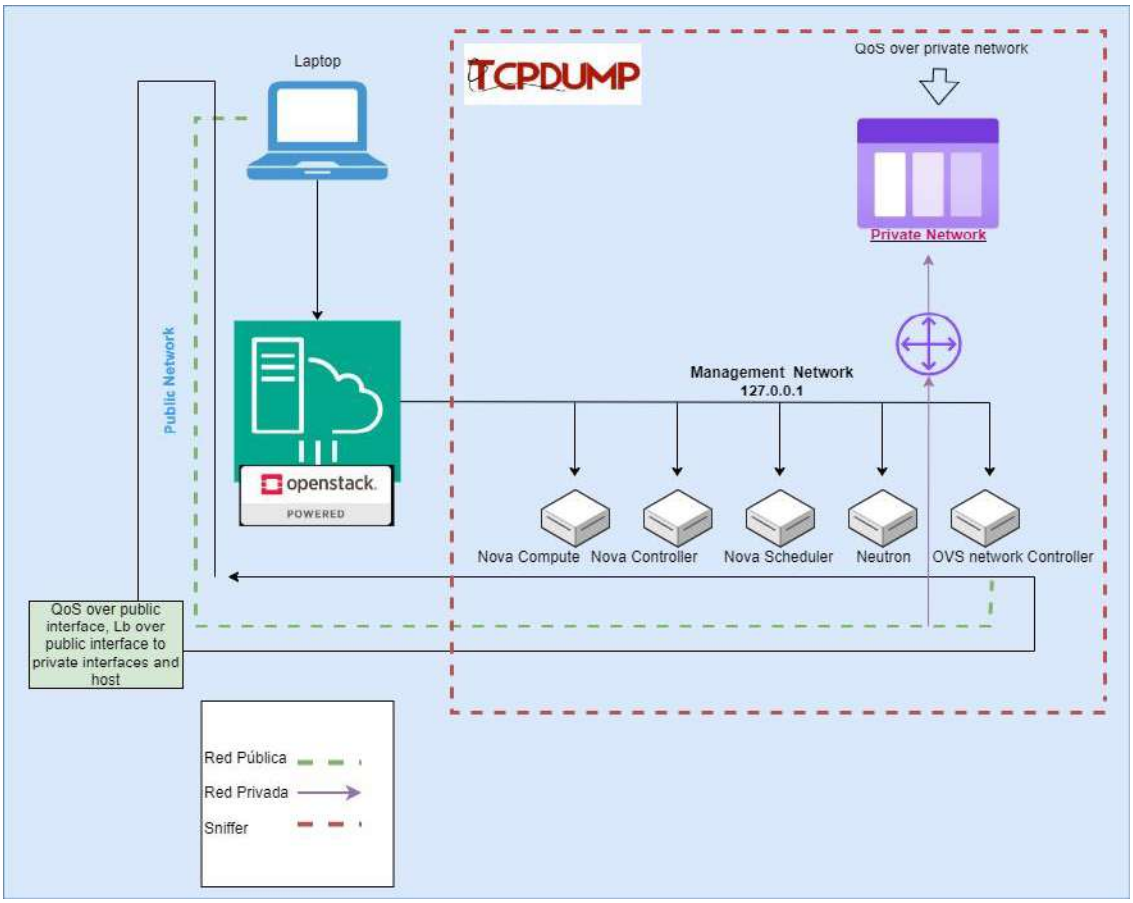
En la fase de pruebas. Se realizará las pruebas de Rendimiento, con la realización de pruebas exhaustivas de rendimiento para evaluar el cumplimiento de los objetivos establecidos, utilizando métricas como la velocidad de procesamiento, la latencia de red y la utilización de recursos. También procederemos con la optimización basada en resultados de pruebas, se empleará un análisis de los resultados de las pruebas de rendimiento y ajuste de la configuración de OpenStack según sea necesario para mejorar el rendimiento.

En la Fase final, despliegue y operación. Se buscará la implementación en producción, en donde se llevará a cabo el despliegue de la infraestructura optimizada en un entorno de producción una vez que se hayan completado las pruebas de rendimiento satisfactoriamente. También se comenzará con la gestión y mantenimiento continuos, en donde el monitoreo continuo del rendimiento de la infraestructura y los servicios de OpenStack en producción, implementando actualizaciones y parches de seguridad de manera planificada para mantener y mejorar el rendimiento. Por último, se diseñará guías que permitan a los estudiantes poder realizar prácticas sobre temáticas de optimización del rendimiento en implementaciones de infraestructura como servicio (IaaS), permitiendo la comprensión de conceptos y generar un criterio.

La siguiente arquitectura describe la infraestructura de prueba, en la que se evidencia las implementaciones de infraestructura como servicio (IaaS) utilizando OpenStack. ***Figura 1***

Figura 1

Arquitectura planteada para la realización del proyecto.



Nota: Este grafico representa una guía para la arquitectura que se va a realizar en este proyecto de titulación, nos muestra el número y tipo de red que se utilizara. También se muestra los nombres de las herramientas, como OpenStack y tcpdumk.

1.5. Justificación

La elección de este tema de proyecto de tesis surge del tema de la computación en la nube ya que es un elemento fundamental para las empresas modernas debido a sus ventajas en flexibilidad y liberación de carga sobre los departamentos de TI (Smart Government Solutions, 2022). Al adoptar servicios como SaaS, PaaS o IaaS, las empresas obtienen modelos de facturación y escalado más eficientes, delegando tareas de instalación y mantenimiento de software y hardware a los proveedores de servicios en la nube. Es particularmente en el servicio

de IaaS, donde se alquila una infraestructura informática completa, donde estas ventajas son más evidentes y beneficiosas (Know How, 2024).

La computación en la nube se vislumbra como la próxima revolución tecnológica del siglo XXI, según muchos líderes del sector. Este avance está potenciando el desarrollo de tecnologías móviles, aplicaciones OTT y otras como inteligencia artificial, cadena de bloques e Internet de las cosas (Orozco & Jacobs, 2016). A pesar de que en el período de estudios 2014-2017 se analizó el acceso a la computación en la nube en países en desarrollo, la rápida evolución de las tendencias en este ámbito hace necesario reexaminar tanto las tecnologías como los modelos de negocio, inversiones y adopción, así como los instrumentos y marcos para su desarrollo en estos países (ITU, 2021).

Las empresas buscan el desarrollar nuevas estrategias de TI con presupuestos de inversión y operaciones más reducidos, que puedan alinear a los requerimientos cambiantes del negocio, de una manera rápida y flexible. Cloud Computing permite a las empresas dimensionar e implementar servicios rápidamente, cambiar con las estrategias del mercado y el negocio, sin tener que invertir en TIC's de manera constante, cuya implementación cae sobre el proveedor de servicios (Mejia, 2015). Desde el punto de vista académico tenemos que Cloud Computing ofrece una gran variedad de herramientas a los estudiantes ya que son de múltiples propósitos. Facilitando así el proceso de enseñanza-aprendizaje ofreciendo nuevos e innovadores servicios tanto por parte del sector privado como público. Se determinó que con la utilización de estas herramientas que nos ofrece Cloud Computing los estudiantes podrán disponer de nuevas y variadas herramientas con múltiples servicios en línea que ayudaran a sector educativo y no solo los estudiantes se pueden beneficiar de etas herramientas, también los docentes podrán beneficiarse (Menéndez, 2017).

La elección de una plataforma de administración de la nube depende de las necesidades empresariales y técnicas específicas, como el uso de contenedores o la integración con proveedores de servicios en la nube existentes (Aguirre, 2016). Estas plataformas ofrecen soluciones que van desde Plataforma como Servicio (PaaS) hasta Infraestructura como Servicio (IaaS), proporcionando un panel de control centralizado para gestionar todas las cargas de trabajo y recursos en la nube. La utilización de una plataforma de administración de la nube ayuda a las organizaciones de TI a gestionar servicios en la nube, optimizar recursos, virtualizar servidores y cumplir con normativas (Sigcha, 2016). OpenStack se destaca como una excelente opción para empresas que buscan una plataforma de nube basada en código abierto en arquitectura Intel®, permitiendo la orquestación dinámica de recursos de computación, redes y almacenamiento. Intel ha sido un colaborador clave en el desarrollo de OpenStack, contribuyendo a mejorar su rendimiento, disponibilidad y seguridad, y continúa ofreciendo tecnologías de plataforma de vanguardia como Intel® (intel.la, 2024).

OpenStack es una plataforma de tecnología open source que facilita la creación y gestión de nubes privadas y públicas mediante recursos virtuales agrupados. Sus herramientas, denominadas "proyectos", se encargan de los principales servicios de cloud computing, como informática, redes, almacenamiento, identidades e imágenes. Además, permite la incorporación de proyectos opcionales para crear nubes personalizadas. En la virtualización, OpenStack utiliza un conjunto uniforme de interfaces de programación de aplicaciones (API) para extraer recursos virtuales de diversos proveedores y distribuirlos según las necesidades a través de hipervisores, potenciando así las herramientas estándar de cloud computing utilizadas por administradores y usuarios (Red Hat, 2021).

La implementación de estrategias efectivas de optimización y eficiencia en las implementaciones de IaaS puede traducirse en una mejor utilización de recursos, una mayor disponibilidad del servicio, una experiencia de usuario mejorada y una reducción de los costos operativos. Por lo tanto, este estudio tiene el potencial de generar beneficios tangibles y proporcionar una ventaja competitiva a las empresas que adopten estas prácticas (WOW! Customer Experience, 2023).

Este estudio se justifica por su relevancia en el contexto empresarial actual, su contribución a cerrar la brecha en habilidades en el mercado laboral. Y en el ámbito educativo, para que los estudiantes salgan preparados para la vida laboral.

La importancia de OpenStack como plataforma de gestión y los potenciales beneficios que puede generar.

CAPÍTULO II: Fundamentación teórica

En el desarrollo de cualquier proyecto, la fundamentación teórica constituye un pilar esencial que sostiene y guía todo el proceso. Este capítulo se enfoca en la optimización del rendimiento en implementaciones de Infraestructura como Servicio (IaaS) utilizando OpenStack, proporcionando un marco conceptual que respalde y contextualice las diversas estrategias y técnicas empleadas. La Infraestructura como Servicio ha revolucionado la manera en que organizaciones gestionan y despliegan sus recursos tecnológicos, ofreciendo flexibilidad, escalabilidad y eficiencia. OpenStack, como una de las plataformas de código abierto más robustas y versátiles, juega un papel crucial en esta transformación. La fundamentación teórica abordará los conceptos clave de IaaS, destacando sus beneficios y desafíos, así como el papel de OpenStack en la implementación y optimización de estas infraestructuras. Además, se explorarán diversas metodologías y enfoques de optimización, analizando su relevancia y aplicación en el contexto de OpenStack. Este marco teórico no solo proporcionará una comprensión profunda de los elementos técnicos y conceptuales involucrados, sino que también establecerá una base sólida para las propuestas de optimización que se desarrollarán y evaluarán a lo largo del proyecto.(Madni et al., 2016)

2.1. Infraestructura como Servicio (IaaS)

2.1.1. Arquitecturas de IaaS

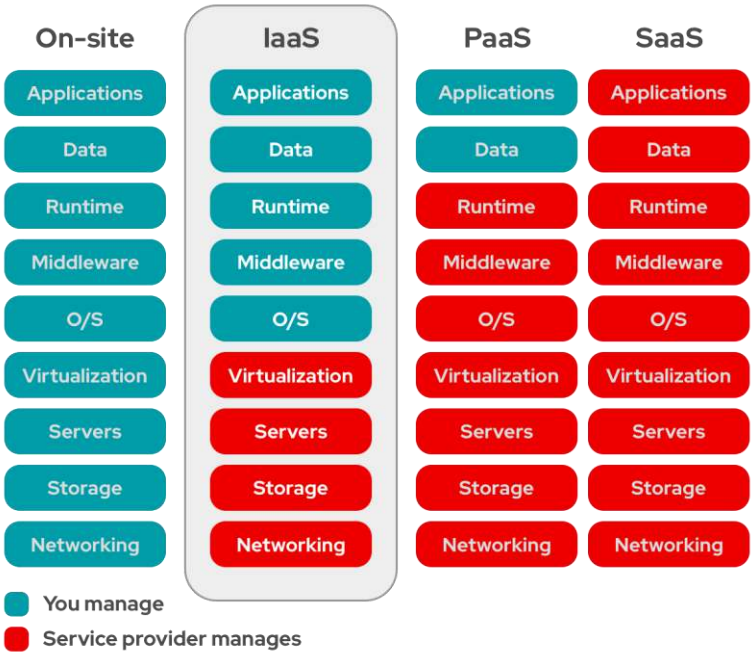
La Infraestructura como Servicio (IaaS) es un modelo de servicio en la computación en la nube que proporciona recursos de infraestructura virtualizados a través de Internet. Estas infraestructuras incluyen componentes fundamentales como servidores, almacenamiento y redes, que pueden ser aprovisionados y gestionados de manera remota. La arquitectura de IaaS se compone de varios elementos clave que permiten su funcionamiento eficiente y escalable(King, 2021).

1. **Virtualización:** La virtualización es la base de IaaS. Permite la creación de máquinas virtuales (VMs) que pueden ejecutar múltiples sistemas operativos y aplicaciones en un solo servidor físico. Esta capa de virtualización es administrada por un hipervisor, que gestiona los recursos físicos y asigna capacidades a las VMs según sea necesario (Bakshi Akash, 2021).
2. **Servidores:** En una arquitectura de IaaS, los servidores físicos son divididos en múltiples VMs. Los proveedores de IaaS utilizan servidores de alta capacidad y rendimiento para asegurar la disponibilidad y escalabilidad de los servicios ofrecidos. Estos servidores pueden ser configurados y gestionados a través de una consola de administración en línea (Team EMB, 2023).
3. **Almacenamiento:** El almacenamiento en IaaS se ofrece generalmente en dos formas: almacenamiento en bloque y almacenamiento en objetos. El almacenamiento en bloque se utiliza para datos estructurados que requieren acceso rápido y secuencial, mientras que el almacenamiento en objetos es ideal para datos no estructurados, como archivos multimedia y copias de seguridad.
4. **Redes:** La conectividad de red en una arquitectura de IaaS incluye componentes como routers, switches y firewalls virtuales. Los proveedores de IaaS proporcionan redes virtuales que permiten a los usuarios configurar y gestionar sus propias subredes, establecer reglas de seguridad y asegurar la conectividad entre sus recursos virtuales (Team EMB, 2023).
5. **Administración y Orquestación:** La administración y orquestación de los recursos de IaaS se realiza a través de plataformas de gestión en la nube. Estas plataformas permiten a los usuarios desplegar, monitorizar y escalar sus recursos según las necesidades de sus aplicaciones y servicios. Herramientas de

automatización y orquestación, como OpenStack y Kubernetes, juegan un papel crucial en la eficiencia operativa de los entornos IaaS (Team EMB, 2023).

Figura 2

Diferencias entre Iaas, PaaS y SaaS (Red Hat, 2023)



Nota. Aquí se muestra los servicios que ofrece cada arquitectura como servicio que existe, donde el verde representa los servicios que el usuario puede manejar y en color rojo los servicios que el proveedor gestiona.

La arquitectura de IaaS proporciona una base flexible y escalable para la implementación de infraestructuras TI, permitiendo a las organizaciones reducir costos operativos y aumentar la agilidad en el despliegue de sus servicios (Team EMB, 2023).

2.1.2. Tecnologías en IaaS

Las tecnologías en Infraestructura como Servicio (IaaS) abarcan una variedad de herramientas y plataformas que permiten la creación, gestión y optimización de recursos virtualizados. Estas tecnologías facilitan el aprovisionamiento dinámico de servidores, almacenamiento y redes, ofreciendo a las organizaciones la flexibilidad necesaria para adaptarse rápidamente a las demandas cambiantes del mercado.(Mestas Yucra, 2015)

1. **Virtualización:** La tecnología de virtualización es fundamental en IaaS, y se implementa a través de hipervisores como VMware vSphere, Microsoft Hyper-V y KVM (Kernel-based Virtual Machine). Los hipervisores permiten la abstracción de los recursos físicos, creando múltiples máquinas virtuales (VMs) en un solo hardware físico, optimizando el uso de los recursos (Slingerland, 2024).
2. **Almacenamiento en la nube:** Las soluciones de almacenamiento en la nube son cruciales para IaaS. Entre ellas se incluyen Amazon S3 (Simple Storage Service), Google Cloud Storage y Microsoft Azure Blob Storage. Estas tecnologías permiten el almacenamiento de datos en la nube, proporcionando alta disponibilidad, durabilidad y escalabilidad. Además, servicios como Elastic Block Store (EBS) de AWS permiten almacenamiento en bloque, ideal para bases de datos y sistemas de archivos (Khorolets, 2024).
3. **Redes definidas por software (SDN):** SDN es una tecnología que permite la gestión centralizada y programable de la red. Plataformas como OpenFlow y Cisco ACI (Application Centric Infrastructure) permiten la creación de redes virtuales que pueden ser configuradas y gestionadas de manera dinámica, mejorando la eficiencia y seguridad de la red (Khorolets, 2024).
4. **Plataformas de gestión y orquestación:** Herramientas como OpenStack, CloudStack y VMware vCloud Director son esenciales para la administración y orquestación de entornos IaaS. Estas plataformas permiten a los usuarios desplegar y gestionar recursos en la nube, proporcionando interfaces de usuario intuitivas y APIs para la automatización y control de los servicios en la nube (Khorolets, 2024).
5. **Contenedores y orquestadores:** Tecnologías de contenedores, como Docker, y orquestadores de contenedores, como Kubernetes, han revolucionado la forma en

que se gestionan las aplicaciones en IaaS. Los contenedores permiten empaquetar aplicaciones y sus dependencias en unidades ligeras y portátiles, mientras que los orquestadores facilitan el despliegue, escalado y gestión de estos contenedores en un entorno distribuido (Slingerland, 2024).

6. **Automatización y DevOps:** Herramientas de automatización como Ansible, Chef y Puppet, y prácticas de DevOps, juegan un papel crucial en la gestión eficiente de infraestructuras IaaS. Estas tecnologías permiten la automatización de tareas repetitivas, la implementación de infraestructura como código (IaC) y la integración continua/del despliegue continuo (CI/CD), mejorando la velocidad y consistencia en la entrega de servicios (Khorolets, 2024).
7. **Seguridad en la nube:** Tecnologías de seguridad, como firewalls virtuales, sistemas de detección y prevención de intrusiones (IDS/IPS) y soluciones de gestión de identidades y accesos (IAM), son fundamentales para proteger los entornos IaaS. Plataformas como AWS IAM, Azure Active Directory y Google Cloud Identity proporcionan controles granulares para gestionar el acceso y asegurar los datos y recursos en la nube (Khorolets, 2024).

Las tecnologías en IaaS continúan evolucionando, ofreciendo cada vez más funcionalidades y mejoras en términos de rendimiento, escalabilidad y seguridad. La adopción de estas tecnologías permite a las organizaciones aprovechar al máximo los beneficios de la computación en la nube, optimizando sus operaciones y respondiendo eficientemente a las necesidades del negocio(Ruiz Quispe, 2016).

2.1.3. Protocolos Utilizados en IaaS

Los protocolos utilizados en Infraestructura como Servicio (IaaS) son esenciales para la comunicación, gestión y seguridad de los recursos en la nube. Estos protocolos permiten la interoperabilidad entre diferentes sistemas y servicios, garantizando un funcionamiento eficiente y seguro de las infraestructuras virtualizadas. A continuación, mediante la **Tabla1** se describen algunos de los protocolos más comunes y su relevancia en el contexto de IaaS(Espino, 2009).

Tabla 1
Protocolos Utilizados en IaaS

Protocolo	Función	Casos de Uso en IaaS	Descripción Técnica
Hypertext Transfer Protocol (HTTP/HTTPS)	Comunicación segura entre servidores y clientes web	Acceso a consolas de administración de IaaS, API de gestión remota	HTTPS es una versión segura de HTTP que utiliza SSL/TLS para encriptar la comunicación entre el cliente y el servidor, asegurando la integridad de los datos.
Simple Object Access Protocol (SOAP)	Intercambio de mensajes en servicios web	Protocolo de comunicación en la API de gestión de recursos de IaaS, integración con otros servicios en la nube	Protocolo basado en XML que facilita el intercambio de datos estructurados en redes distribuidas.
Representational State Transfer (REST)	Interfaz para la interacción con servicios web	API de IaaS para operaciones CRUD (Create, Read, Update, Delete) de recursos virtuales, como máquinas virtuales y almacenamiento	Basado en HTTP, REST proporciona una forma ligera y escalable de comunicarse con servicios web, usando URLs y

			métodos HTTP como GET, POST, PUT y DELETE.
Secure Shell (SSH)	Acceso remoto seguro a sistemas virtuales	Acceso a servidores virtuales (VMs) en IaaS para gestión, mantenimiento y configuración de sistemas	SSH utiliza criptografía para proporcionar un canal seguro sobre una red insegura, comúnmente utilizado para acceder de manera remota a servidores y dispositivos.
Virtual Extensible LAN (VXLAN)	Protocolo de encapsulación de red para redes virtuales	Creación de redes virtuales superpuestas (overlay networks) para la conexión de recursos distribuidos geográficamente en centros de datos IaaS	VXLAN extiende redes LAN tradicionales para permitir la conectividad entre máquinas virtuales en diferentes segmentos de red dentro de una infraestructura de nube.
OpenFlow	Protocolo de control de red en SDN	Gestión dinámica de redes virtualizadas en IaaS, facilitando la automatización del flujo de tráfico y la administración de red	Permite a los controladores SDN interactuar directamente con los dispositivos de red (switches, routers) para gestionar el tráfico de manera programática.
Internet Protocol Security (IPsec)	Cifrado y autenticación de datos en redes IP	Proteger la transferencia de datos entre servidores virtuales y entre centros de datos IaaS	IPsec ofrece un conjunto de protocolos para cifrar y autenticar paquetes IP, asegurando la transmisión segura de datos en redes IP.

Dynamic Host Configuration Protocol (DHCP)	Asignación automática de direcciones IP	Provisión automática de direcciones IP a máquinas virtuales creadas en entornos IaaS	Protocolo que asigna dinámicamente direcciones IP y otros parámetros de red a dispositivos en la red.
Domain Name System (DNS)	Resolución de nombres de dominio	Gestión de dominios y direccionamiento en IaaS para la creación de nombres de host accesibles a través de Internet o redes privadas virtuales	DNS convierte nombres de dominio legibles por humanos en direcciones IP numéricas necesarias para identificar dispositivos en redes.
Lightweight Directory Access Protocol (LDAP)	Administración de identidades y acceso	Autenticación y autorización de usuarios en entornos IaaS, integrándose con sistemas de gestión de identidades para gestionar el acceso a recursos virtuales	Protocolo ligero para el acceso y mantenimiento de directorios distribuidos que contienen información sobre usuarios, grupos y recursos de red.

Nota. Esta tabla muestra todos los protocolos utilizados en IaaS y una pequeña descripción de cada uno de estos.

2.2.1. Diversas Plataformas

2.2.1.1. Amazon Web Services (AWS)

Amazon Web Services (AWS) es una de las plataformas de IaaS más populares y ampliamente utilizadas. AWS ofrece una amplia gama de servicios, incluyendo computación, almacenamiento, bases de datos y redes, entre otros. La plataforma es conocida por su escalabilidad, flexibilidad y robustez, proporcionando soluciones adecuadas tanto para startups como para grandes empresas (AWS, 2023).

2.2.1.2. Microsoft Azure

Microsoft Azure es otra plataforma líder en el mercado de IaaS. Azure proporciona una extensa lista de servicios, que incluyen máquinas virtuales, almacenamiento en la nube, redes

y bases de datos. La integración con otros productos de Microsoft y su compatibilidad con diversas tecnologías la hacen una opción atractiva para empresas que buscan una solución integral y escalable (Taylor, 2023).

2.2.1.3. Google Cloud Platform (GCP)

Google Cloud Platform (GCP) ofrece una infraestructura global altamente segura y escalable. GCP proporciona servicios de computación, almacenamiento y bases de datos, así como herramientas avanzadas de análisis de datos e inteligencia artificial. La plataforma es conocida por su enfoque en la innovación y la eficiencia, siendo una opción preferida por empresas tecnológicas y de datos (Haranas, 2023).

2.2.1.4. OpenStack

OpenStack es una plataforma de código abierto para la creación y gestión de infraestructuras en la nube. OpenStack proporciona una suite de componentes modulares que permiten la implementación de entornos IaaS personalizados y escalables. Su naturaleza de código abierto y la amplia comunidad de desarrolladores hacen de OpenStack una opción flexible y económica para organizaciones que buscan evitar el bloqueo de proveedores (openstack, 2024).

2.2.1.5. VMware vCloud

VMware vCloud es una plataforma de IaaS que se centra en proporcionar soluciones de virtualización y gestión de la nube. vCloud permite a las organizaciones crear nubes híbridas, combinando recursos locales con la infraestructura de nube pública. La plataforma es conocida por su fuerte enfoque en la seguridad y la compatibilidad con las soluciones de virtualización de VMware (VM, 2024).

2.2.1.6. IBM Cloud

IBM Cloud ofrece una amplia gama de servicios de IaaS, incluyendo computación, almacenamiento y redes, junto con herramientas avanzadas para inteligencia artificial y análisis

de datos. La plataforma se destaca por su enfoque en la integración de tecnologías de nube híbrida y su capacidad para soportar cargas de trabajo empresariales críticas (IBM, 2024).

Cada una de estas plataformas ofrece características únicas y ventajas específicas, permitiendo a las organizaciones seleccionar la que mejor se adapte a sus necesidades y objetivos estratégicos. La elección de la plataforma adecuada puede influir significativamente en el rendimiento, la seguridad y la eficiencia operativa de la infraestructura en la nube (Wawira & Hougen, 2024).

2.2.2. Comparativa entre las principales plataformas para Iaas

La elección de una plataforma de Infraestructura como Servicio (IaaS) es una decisión crítica para las organizaciones que desean aprovechar las ventajas de la computación en la nube. Las principales plataformas de IaaS ofrecen diferentes características, beneficios y costos.

AWS y Azure son ideales para organizaciones que buscan una amplia gama de servicios y escalabilidad global. GCP es una opción sólida para aquellas que necesitan capacidades avanzadas de análisis de datos y machine learning (NAKIVO, 2024). OpenStack es perfecto para organizaciones que buscan flexibilidad y evitar el bloqueo de proveedores, mientras que VMware vCloud es adecuado para entornos que ya utilizan ampliamente las soluciones de VMware (Hystax, 2023). En la **Tabla 2** muestra una comparativa de las plataformas IaaS:

Tabla 2
Características de Plataformas para IaaS

Características	AWS	Azure	GCP	OpenStack	VMware vCloud
Escalabilidad	Alta	Alta	Alta	Alta	Alta
Flexibilidad	Alta	Alta	Alta	Muy Alta	Media

Costo	Variable, generalmente alto	Variable, generalmente alto	Competitivo	Bajo	Alto
Facilidad de uso	Media	Alta	Media	Baja	Alta
Soporte y Comunidad	Excelente, gran ecosistema	Excelente, integración con Microsoft	Buena, enfoque en datos y AI	Excelente, comunidad activa	Buena, fuerte enfoque en virtualización
Servicios Adicionales	Amplia gama de servicios avanzados	Amplia gama de servicios avanzados	Amplia gama de servicios avanzados	Limitada en comparación	Limitada en comparación
Modelo de Facturación	Pago por uso, opciones de ahorro	Pago por uso, opciones de ahorro	Pago por uso, descuentos por compromiso	Sin costos de licencia	Costos de licencia altos
Integración con Herramientas	Amplia integración con terceros	Integración con herramientas de Microsoft	Integración con herramientas de Google	Alta, depende de configuración	Alta con soluciones VMware
Seguridad	Alto nivel de seguridad	Alto nivel de seguridad	Alto nivel de seguridad	Depende de la configuración	Enfoque fuerte en seguridad
Actualización y Mantenimiento	Automático, gestionado por AWS	Automático, gestionado por Azure	Automático, gestionado por Google	Manual, requiere administración	Automático, gestionado por VMware

Interoperabilidad	Alta	Alta	Alta	Muy Alta	Media
Soporte para DevOps	Amplio soporte (AWS DevOps, herramientas)	Amplio soporte (Azure DevOps, herramientas)	Amplio soporte (Google Cloud Build, etc.)	Buen soporte, depende de configuración	Amplio soporte (vRealize Suite, etc.)
Casos de Uso Comunes	Todo tipo de aplicaciones y servicios	Aplicaciones empresariales y híbridas	Análisis de datos, AI, aplicaciones web	Nubes privadas y públicas personalizadas	Entornos de virtualización híbridos
Ventajas	Amplia gama de servicios, escalabilidad	Integración con Microsoft, facilidad de uso	Potente análisis de datos, precios competitivos	Código abierto, altamente customizable	Integración con VMware, seguridad
Desventajas	Costos elevados, curva de aprendizaje	Costos elevados, posibles problemas de compatibilidad	Menor cantidad de servicios, ecosistema más pequeño	Necesidad de mayor conocimiento técnico	Costos de licencias elevados, menor flexibilidad

Nota. En esta tabla refleja todas las características que posee las infraestructuras IaaS

2.2.3. OpenStack

OpenStack es una plataforma de código abierto para la creación y gestión de nubes privadas y públicas, que se ha convertido en una solución popular para la implementación de Infraestructura como Servicio (IaaS). Diseñada para ser flexible y escalable, OpenStack permite a las organizaciones construir entornos de nube a medida, adaptados a sus necesidades

específicas. A continuación, se detallan los componentes clave y características de OpenStack (Haff, 2017).

2.2.3.1. Componentes Principales de OpenStack

- **Nova (Computación):** Nova es el componente de OpenStack encargado de la gestión de instancias de máquinas virtuales y recursos de computación. Proporciona la capacidad para crear, modificar y eliminar instancias, además de gestionar el ciclo de vida de las máquinas virtuales. Nova es compatible con diversos hipervisores, como KVM y VMware, permitiendo a las organizaciones utilizar la tecnología que mejor se adapte a sus necesidades (Dautovic, 2023).
- **Swift (Almacenamiento en Objetos):** Swift es el sistema de almacenamiento en objetos de OpenStack, diseñado para almacenar grandes cantidades de datos no estructurados. Ofrece alta disponibilidad y redundancia, permitiendo la replicación de datos en múltiples ubicaciones para garantizar su durabilidad. Swift es ideal para aplicaciones que requieren almacenamiento de datos masivos, como archivos multimedia y copias de seguridad (Paternó, 2015).
- **Cinder (Almacenamiento en Bloque):** Cinder proporciona almacenamiento en bloque para instancias de máquinas virtuales, permitiendo a los usuarios crear y gestionar volúmenes de almacenamiento que pueden ser adjuntados a las máquinas virtuales según sea necesario. Ofrece funcionalidades avanzadas como instantáneas y replicación, mejorando la flexibilidad y la gestión del almacenamiento (Dautovic, 2023).
- **Neutron (Redes):** Neutron gestiona la conectividad de red dentro de OpenStack, proporcionando servicios de redes y administración de direcciones IP. Permite a los usuarios crear redes virtuales, configurar firewalls, balanceadores de carga y

VPNs. Neutron ofrece una arquitectura modular que se puede extender con plugins para soportar diferentes tecnologías de red (Dautovic, 2023).

- **Horizon (Interfaz de Usuario):** Horizon es el panel de control basado en web de OpenStack, que permite a los usuarios gestionar sus recursos de nube a través de una interfaz gráfica intuitiva. Proporciona acceso a las funcionalidades de Nova, Swift, Cinder, Neutron y otros componentes, facilitando la administración y monitoreo de la infraestructura en la nube (Dautovic, 2023).
- **Keystone (Identidad y Autenticación):** Keystone es el servicio de gestión de identidades y autenticación de OpenStack. Maneja la autenticación de usuarios y servicios, así como la autorización y gestión de permisos. Keystone proporciona un sistema centralizado para el control de acceso y la integración con otras soluciones de gestión de identidades (Dautovic, 2023).
- **Glance (Imágenes de Disco):** Glance es el servicio de gestión de imágenes de disco en OpenStack. Permite a los usuarios almacenar, recuperar y gestionar imágenes de disco y snapshots que pueden ser utilizados para crear nuevas instancias. Glance soporta múltiples formatos de imagen y ofrece funcionalidades para la conversión y almacenamiento de imágenes (Paternó, 2015).

2.2.3.2. Características Clave de OpenStack

- **Código Abierto y Comunidad Activa:** OpenStack es una plataforma de código abierto, lo que significa que su código fuente está disponible públicamente para su revisión y modificación. La comunidad activa de desarrolladores y usuarios contribuye continuamente al desarrollo de nuevas características y mejoras (cloudification, 2024).
- **Escalabilidad y Flexibilidad:** OpenStack está diseñado para escalar horizontalmente, permitiendo la adición de nuevos nodos y recursos según sea

necesario. Su arquitectura modular permite a las organizaciones personalizar y extender la plataforma para satisfacer sus necesidades específicas (cloudification, 2024).

- **Interoperabilidad:** OpenStack es compatible con una amplia gama de tecnologías y herramientas, facilitando la integración con soluciones de terceros y la interoperabilidad con otros entornos de nube. Su soporte para diferentes hipervisores, sistemas de almacenamiento y redes proporciona flexibilidad en la elección de tecnologías (cloudification, 2024).
- **Automatización y Gestión:** OpenStack incluye herramientas y APIs para la automatización y gestión de recursos, permitiendo a los usuarios definir y desplegar infraestructuras a través de scripts y plantillas. Herramientas como Heat (orquestación) y Trove (bases de datos) amplían las capacidades de gestión y automatización de la plataforma (Pure Storage, 2024).

2.3. Optimización en redes SDN

Las Redes Definidas por Software (SDN, por sus siglas en inglés) representan una de las innovaciones más significativas en el ámbito de las redes y la infraestructura de TI. SDN permite una gestión más flexible y eficiente de las redes al separar el plano de control del plano de datos (Microsoft, 2023). Esto facilita la optimización de los recursos de red y la implementación de políticas de gestión más avanzadas. En este tema, exploraremos los conceptos fundamentales de SDN, las técnicas de optimización y los beneficios que aportan a las infraestructuras IaaS(Díaz Sánchez, 2017).

Tabla 3
Clasificación de los posibles beneficios de SDN (Cisco, 2014).

Categoría	Esencial	Importante, pero no esencial	Útil, pero no importante	Para nada importante
Uso optimizado de recursos	35.8	42.0	16.0	3.7
Innovación para redes y servicios	30.5	46.3	15.9	2.4
Implementación de hardware de menor costo	36.8	39.0	14.6	4.9
Reducción de costos operativos, especialmente para mantener los equipos distribuidos	41.5	34.1	18.3	3.7
Automatización u organización de servicios con más velocidad de servicios	36.8	39.0	19.5	4.7
Coordinación de asignación de recursos de proveedores de servicios de red con redes IP/MPLS y DC	37.0	38.3	17.3	4.9
Rentabilización de servicios y nuevos servicios	39.6	35.8	17.3	2.5

Nota. En esta figura se muestra una tabla de los beneficios de usar SDN, obtenido del estudio de Heavy Reading sobre operadores de red.

2.3.1. Aplicación de Calidad de servicio

La Calidad de Servicio (QoS, por sus siglas en inglés) es un aspecto crucial en la gestión de redes, especialmente en entornos de Infraestructura como Servicio (IaaS) donde la eficiencia y la fiabilidad del tráfico de red son fundamentales. La aplicación de QoS en Redes Definidas por Software (SDN) permite una gestión más precisa y flexible del tráfico de red, asegurando que las aplicaciones críticas reciban el nivel de servicio necesario.(Kalaiselvan & Venkatakrishna, 2013)

2.3.1.1. Conceptos Fundamentales de QoS en SDN

QoS en el contexto de SDN se refiere a la capacidad de gestionar y priorizar el tráfico de red para garantizar un rendimiento consistente y fiable. Esto incluye la gestión del ancho de banda, la latencia, la variación de la latencia (jitter) y la pérdida de paquetes. La arquitectura SDN permite implementar políticas de QoS de manera centralizada, simplificando la administración y aumentando la eficiencia.(Kalaiselvan & Venkatakrishna, 2013)

- Ancho de Banda Garantizado: Asignación de una cantidad fija de ancho de banda para aplicaciones críticas para asegurar que siempre dispongan de los recursos necesarios (Reyes, 2024).
- Control de Latencia: Reducción y gestión de la latencia para aplicaciones sensibles al tiempo, como servicios de voz y video en tiempo real (Reyes, 2024).
- Minimización de Jitter: Control de la variación de la latencia para asegurar la entrega uniforme de paquetes, crucial para aplicaciones de tiempo real (Salazar, 2016).
- Prevención de Pérdida de Paquetes: Gestión de colas y buffers para evitar la congestión de la red y la pérdida de datos (Salazar, 2016).

2.3.2. Monitoreo

El monitoreo en Redes Definidas por Software (SDN) es un componente esencial para asegurar la eficiencia, seguridad y rendimiento de las infraestructuras de red. En un entorno de Infraestructura como Servicio (IaaS), donde la disponibilidad y la calidad del servicio son críticas, el monitoreo continuo permite a los administradores de red detectar y resolver problemas de manera proactiva. Este subtema explora las técnicas y herramientas de monitoreo en SDN, así como sus beneficios y casos de uso (Reyes, 2024).

2.3.2.2. Técnicas de Monitoreo en SDN

El monitoreo en SDN se lleva a cabo utilizando diversas técnicas y herramientas que proporcionan datos en tiempo real sobre el estado de la red lo cual se verá en la **¡Error! No se encuentra el origen de la referencia.**:

Tabla 4
Técnicas de Monitoreo en SDN.

Técnica de Monitoreo	Descripción
Telemetry y Streaming Analytics	Recopilación continua de datos de rendimiento y eventos de red, que se transmiten a un sistema de análisis para su procesamiento y visualización.
Sondeo de Red (Polling)	Solicitud periódica de datos a los dispositivos de red para obtener información sobre su estado y rendimiento.
Registro de Eventos (Logging)	Captura de eventos significativos en la red, como cambios en la configuración o errores, para su análisis posterior.
Análisis de Tráfico (Flow Analysis)	Inspección detallada del tráfico de red para identificar patrones y comportamientos anómalos.
Supervisión de Integridad (Health Monitoring)	Evaluación continua del estado de los dispositivos de red y su capacidad operativa.

Nota. En esta tabla describe las técnicas de monitoreo que utiliza SDN.

2.4. Tendencias y futuro

La evolución de las Redes Definidas por Software (SDN) y su integración con Infraestructura como Servicio (IaaS) está en constante transformación. A medida que la tecnología avanza, surgen nuevas tendencias que prometen mejorar aún más la eficiencia, seguridad y capacidad de gestión de las redes. En este tema, exploraremos las tendencias actuales y las posibles direcciones futuras en el ámbito de SDN y IaaS (Worldstream, 2023).

2.4.1. Tendencias Actuales

Integración con Inteligencia Artificial y Machine Learning: La integración de SDN con tecnologías de Inteligencia Artificial (IA) y Machine Learning (ML) está ganando terreno

rápidamente. Estas tecnologías permiten a las redes aprender y adaptarse de manera autónoma a las condiciones cambiantes, optimizando el rendimiento y mejorando la capacidad de respuesta ante incidentes (Fiveable, 2024).

- Optimización Autónoma: Algoritmos de ML pueden analizar patrones de tráfico y ajustar las políticas de QoS en tiempo real para maximizar la eficiencia.
- Detección de Anomalías: IA puede identificar comportamientos anómalos y posibles amenazas de seguridad, permitiendo una respuesta rápida y precisa.

2.4.1.1. Aumento de la Adopción de Contenedores y Microservicios

La tendencia hacia el uso de contenedores y arquitecturas de microservicios está transformando la manera en que se despliegan y gestionan las aplicaciones. SDN juega un papel crucial en la provisión de redes flexibles y seguras que soporten estas arquitecturas (Vijay K, 2021).

Kubernetes y SDN: Integraciones como CNI (Container Network Interface) permiten a Kubernetes gestionar redes definidas por software, proporcionando conectividad eficiente y escalable para contenedores (Vijay K, 2021).

2.4.1.2. Redes 5G y SDN

El despliegue de redes 5G está impulsando nuevas oportunidades para SDN, permitiendo la creación de redes más rápidas, flexibles y personalizables. La combinación de 5G y SDN promete mejorar significativamente la capacidad de gestionar y orquestar redes a gran escala (Li et al., 2017).

Network Slicing: SDN facilita la implementación de network slicing en redes 5G, permitiendo a los operadores crear múltiples redes virtuales sobre una única infraestructura física (Li et al., 2017).

2.4.1.3. Seguridad Mejorada mediante SDN

La creciente preocupación por la seguridad en la era digital ha llevado a un enfoque renovado en cómo SDN puede mejorar la postura de seguridad de las redes. La capacidad de gestionar la red de manera centralizada permite una implementación más coherente y efectiva de políticas de seguridad.(Lluco et al., 2023)

Microsegmentación: SDN permite la microsegmentación de la red, mejorando el control sobre el tráfico y reduciendo la superficie de ataque.

Respuesta Automatizada a Incidentes: La integración con sistemas de detección de intrusos y respuestas automatizadas permite una defensa más rápida y precisa contra ciberataques.

2.4.2. Direcciones Futuras

A medida que la tecnología de redes definidas por software (SDN) y las infraestructuras como servicio (IaaS) continúan evolucionando, surgen nuevas posibilidades que prometen transformar aún más el entorno de las telecomunicaciones y la computación en la nube. Estas direcciones futuras no solo apuntan a mejorar el rendimiento y la eficiencia, sino también a facilitar la gestión autónoma, la interoperabilidad entre plataformas y la expansión hacia arquitecturas distribuidas como el edge computing, como se ve en la **¡Error! No se encuentra el origen de la referencia.**. En este apartado se analizan algunas de las proyecciones más relevantes que podrían marcar el rumbo de estas tecnologías en los próximos años, considerando tanto avances técnicos como tendencias de integración con nuevas soluciones.

Tabla 5
Direcciones Futuras

Direcciones Futuras	Descripción	Casos	Descripción
Redes Autonómicas	Las redes autonómicas son redes capaces de auto-configurarse, auto-optimizarse y auto-	Autonomía Completa	El objetivo es crear redes que puedan gestionar todas las operaciones de manera

	protegerse sin intervención humana. SDN habilita esta visión proporcionando flexibilidad y control necesarios.		autónoma, desde la configuración hasta la resolución de problemas.
Mayor Enfoque en la Interoperabilidad y Estándares Abiertos	La interoperabilidad entre soluciones y el uso de estándares abiertos permite la combinación de tecnologías de múltiples proveedores sin problemas de compatibilidad. Las iniciativas como ONF y proyectos de código abierto seguirán impulsando estos estándares.	Open Networking	Iniciativas como ONF (Open Networking Foundation) y proyectos de código abierto seguirán promoviendo estándares y plataformas interoperables.
Expansión del Edge Computing	La expansión del edge computing acerca la computación y el almacenamiento al lugar donde se generan los datos. SDN es clave para gestionar estas infraestructuras distribuidas de manera eficiente y centralizada.	Edge y SDN	La combinación de SDN con edge computing permitirá redes más dinámicas, con mejoras en la latencia y el rendimiento, lo que es crucial para aplicaciones sensibles al tiempo de respuesta.
Blockchain y Seguridad de Redes	La tecnología blockchain mejora la seguridad y transparencia en la gestión de redes SDN mediante la creación de registros inmutables de eventos, lo que refuerza la confianza y la seguridad en la red.	Gestión Descentralizada	Blockchain permite una gestión descentralizada y más segura de políticas de red, lo que mejora la resiliencia y la transparencia en la gestión de redes SDN.

Nota. Esta tabla describe que tendencias puede existir al utilizar este tipo de tecnologías y en qué casos podemos aplicar.

CAPÍTULO III: Desarrollo del Proyecto

El presente capítulo describe el desarrollo técnico del proyecto titulado "Optimización del rendimiento en implementaciones de infraestructura como servicio (IaaS) utilizando OpenStack". En esta sección se detalla cada una de las fases que conforman el proceso de implementación, desde la planificación inicial hasta la puesta en marcha del entorno funcional, siguiendo la metodología de desarrollo en cascada.

Durante el desarrollo del proyecto se realizaron actividades enfocadas en la definición de requerimientos, diseño de la arquitectura, instalación y configuración de los servicios de OpenStack, además de la ejecución de pruebas orientadas a evaluar el rendimiento del sistema implementado. Todo este proceso se llevó a cabo con el objetivo de cumplir con los objetivos específicos planteados y de garantizar una solución funcional, escalable y orientada al entorno educativo.

La estructura de este capítulo permite comprender cómo se abordó la implementación técnica del proyecto, mostrando los procedimientos seguidos, las herramientas utilizadas y los criterios aplicados para asegurar la optimización de los recursos disponibles. Asimismo, se documentan las decisiones tomadas en cada fase con el fin de proporcionar una guía útil para futuras implementaciones similares.

3.1. Diagnóstico de la Infraestructura Actual

Antes de proceder con el despliegue de OpenStack y realizar las optimizaciones necesarias, es fundamental realizar un diagnóstico exhaustivo de la infraestructura existente. Este análisis tiene como objetivo identificar los puntos débiles y las limitaciones del entorno actual, permitiendo así comprender las necesidades de mejora y los requisitos técnicos para

implementar una solución eficaz. Un diagnóstico adecuado nos brinda una visión clara sobre los recursos físicos y lógicos disponibles, las capacidades actuales del sistema y las dificultades que se presentan al realizar tareas de virtualización y simulación de redes.

En este apartado, se analiza tanto el hardware como el software que conforman la infraestructura previa utilizada para las prácticas académicas, y se evalúan sus limitaciones en términos de rendimiento, escalabilidad y gestión de recursos. Además, se revisa la configuración de las redes virtuales, ya que la correcta gestión del tráfico y la asignación dinámica de direcciones IP son factores clave para la creación de una infraestructura eficiente y optimizada.

Este diagnóstico permite identificar áreas en las que la infraestructura actual no es capaz de soportar de manera eficiente los servicios y tareas propuestas en el proyecto, como la falta de herramientas de automatización, la ausencia de políticas de calidad de servicio (QoS) y la deficiencia en la gestión del almacenamiento y las redes. Con base en estos hallazgos, se establecerán los requisitos necesarios para la instalación y puesta en marcha de OpenStack, y se podrá diseñar una infraestructura más robusta y preparada para realizar simulaciones avanzadas de redes y servicios en la nube.

3.1.1. Descripción del entorno físico y lógico previo

Previo a la implementación de OpenStack, la infraestructura se encontraba basada en la instalación directa de un sistema operativo base Ubuntu Server 20.04 LTS, configurado como plataforma inicial para la provisión de servicios, sobre el cual se disponían recursos de cómputo, red y almacenamiento administrados de forma centralizada, permitiendo la operación

continua de servicios sin mecanismos avanzados de orquestación ni gestión automatizada de infraestructura como servicio.

En este contexto, la administración de los recursos se realizaba mediante configuraciones manuales del sistema operativo y de los servicios de red, sin contar con herramientas especializadas para la virtualización integral, la asignación dinámica de direcciones IP, la segmentación lógica de redes o la aplicación de políticas de calidad de servicio, lo que limitaba la escalabilidad, la eficiencia operativa y la capacidad de adaptación a cargas variables propias de entornos de producción basados en IaaS.

3.1.2. Recursos de hardware y software disponibles

La implementación del entorno se llevó a cabo sobre un servidor con hardware de gama media-alta, diseñado para operar de forma continua en un contexto de producción y con soporte completo para virtualización por hardware mediante tecnologías VT-x, dicho equipamiento permitió disponer de recursos de cómputo, memoria y almacenamiento dimensionados de manera equilibrada, garantizando estabilidad operativa, capacidad de procesamiento y condiciones adecuadas para la ejecución sostenida de los servicios de OpenStack bajo carga real.

Los recursos físicos fueron asignados considerando criterios de rendimiento, disponibilidad y eficiencia, con el objetivo de asegurar una correcta operación de la plataforma IaaS y una adecuada gestión de instancias, redes y servicios, en la **¡Error! No se encuentra el origen de l a referencia.** se presentan los componentes de hardware disponibles y la distribución de recursos utilizada para la ejecución de OpenStack en el entorno productivo establecido:

Tabla 6

Tabla Recursos de hardware disponibles

Componente	Recurso disponible (host físico)	Observaciones
Procesador (CPU)	Procesador multinúcleo de 12 núcleos lógicos	Capacidad suficiente para la ejecución simultánea de servicios de cómputo, red y almacenamiento garantizando estabilidad operativa
Memoria RAM	24 GB DDR5	Permite la ejecución continua de los servicios principales de OpenStack y la creación de múltiples instancias con un rendimiento sostenido
Almacenamiento	300 GB SSD	Proporciona alta velocidad de lectura y escritura favoreciendo la carga de servicios, almacenamiento de imágenes y volúmenes persistentes
Tarjeta de red	Adaptador de red Ethernet configurado en red personalizada	Permite conectividadcción controlada del tráfico y comunicación eficiente entre los servicios de red de la infraestructura

Nota: Se detallan los componentes físicos del sistema y los recursos asignados para la operación de la plataforma OpenStack en un entorno de producción

En cuanto al software utilizado, en la **¡Error! No se encuentra el origen de la r eferencia.** se presenta el conjunto de herramientas y plataformas empleadas para la instalación, configuración, monitoreo y operación de OpenStack:

Tabla 7

Tabla Recursos de software disponibles

Software	Versión o descripción	Función dentro del entorno

Ubuntu Server	22.04.5 LTS	Sistema operativo base
OpenStack	Última versión estable	sistema de control y orquestación de recursos de cloud computing
Git y Python	Git 2.34 / Python 3.10+	Requisitos para método de instalación de OpenStack y gestión de scripts
htop, tcpdump	Utilidades Linux	Monitoreo de tráfico, consumo de recursos y análisis de red
Wireshark	Utilidades Linux y Windows	Monitoreo de tráfico
Navegador web	Firefox o Chrome	Acceso a Horizon (dashboard web de OpenStack)

Nota: Se detalla el conjunto de herramientas y plataformas empleadas para la instalación

3.1.3. Evaluación del rendimiento inicial

Al realizar pruebas previas sin una plataforma de orquestación como OpenStack, se observaron las siguientes limitaciones:

- Sobrecarga de CPU y RAM al ejecutar más de tres máquinas virtuales de forma simultánea.
- Altos tiempos de respuesta en aplicaciones y servicios simulados.
- Falta de control sobre el tráfico de red, afectando pruebas relacionadas con calidad de servicio (QoS).
- Gestión manual de IPs y servicios, lo cual generaba errores recurrentes y tiempos prolongados de configuración.

Estas observaciones evidencian la necesidad de adoptar una solución más escalable y automatizada como OpenStack, incluso en contextos educativos con recursos limitados.

3.1.4. Análisis de la arquitectura de red actual

La arquitectura de red considerada para el desarrollo del proyecto se basa en un esquema de nodo único, en el cual todos los servicios principales de la infraestructura se ejecutan de forma centralizada sobre un mismo servidor, este enfoque responde a la arquitectura soportada por OpenStack y permite implementar una plataforma IaaS completamente funcional manteniendo un control directo sobre los componentes de red, cómputo y almacenamiento.

Previo a la integración de los servicios avanzados de OpenStack, la red operaba bajo una estructura básica de conectividad, orientada a permitir la comunicación entre los servicios del sistema, pero sin mecanismos de segmentación lógica ni control dinámico del tráfico, esta configuración inicial limitaba la aplicación de políticas propias de entornos productivos, como la priorización de servicios, el aislamiento de redes o la gestión centralizada del flujo de datos.

- Capa 2 (enlace de datos): En la capa de enlace, la arquitectura no contemplaba la definición explícita de VLANs ni esquemas de aislamiento entre segmentos de red, el tráfico era manejado de forma uniforme dentro del nodo, sin diferenciación por tipo de servicio ni prioridad, lo que impedía aplicar políticas de control a nivel de enlace y limitaba la eficiencia en la gestión del dominio de broadcast.

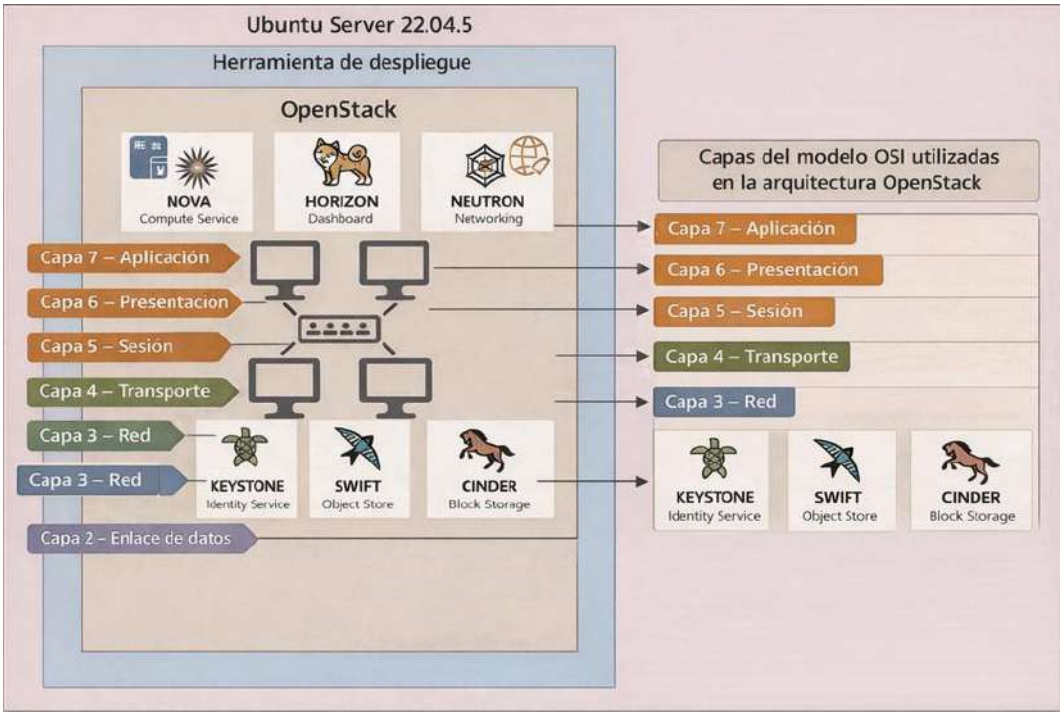
- Capa 3 (red): En la capa de red, el direccionamiento IP se gestionaba de forma estática, sin servicios de asignación dinámica ni políticas internas de control de acceso, la ausencia de reglas de enrutamiento y filtrado restringía la segmentación lógica de la red y dificultaba la implementación de esquemas de seguridad y control de tráfico acordes a un entorno productivo.

Con la incorporación de OpenStack bajo un esquema de nodo único, se integra el servicio Neutron como componente central de gestión de red, permitiendo la creación de redes lógicas, la asignación dinámica de direcciones IP, la definición de reglas de seguridad y la segmentación del tráfico dentro del mismo nodo, todo ello administrado desde un plano de control centralizado, lo que habilita una arquitectura de red organizada, controlada y alineada con los requerimientos de operación de una infraestructura como servicio en producción.

3.1.5. Representación de la arquitectura

La **¡Error! No se encuentra el origen de la referencia.** muestra la arquitectura p lanteada para este proyecto. Esta estructura parte del sistema operativo base Ubuntu Server, que contiene la herramienta de despliegue y todos los servicios de OpenStack. Desde allí se gestionan instancias virtuales y se habilita el acceso a redes internas y externas mediante una red virtual en modo puente o NAT.

Figura 3
Arquitectura para OpenStack sobre Ubuntu Server



Nota: Se representa la ejecución del entorno OpenStack desplegado mediante su método de despliegue sobre un SO base Ubuntu Server

3.2. Requisitos del Sistema

Para asegurar que la implementación y simulación de la plataforma OpenStack se realice de forma exitosa, es indispensable definir con precisión todos los requisitos del sistema que garanticen un entorno funcional, estable y acorde a los objetivos planteados en este proyecto. Los requisitos se dividen en funcionales, técnicos, de hardware, software y diseño de la topología de red, buscando cubrir todos los aspectos necesarios para un despliegue adecuado en un entorno académico.

Para definir correctamente los requisitos necesarios para implementar la plataforma OpenStack, se ha tomado como base la norma IEEE 29148-2018, que proporciona una guía para la especificación de requisitos en sistemas y software. Esta norma propone clasificar los requisitos en grupos como: requisitos de usuario, requisitos del sistema, requisitos funcionales y requisitos no funcionales, lo cual ayuda a organizarlos de manera clara y verificable. Aplicar esta estructura no solo mejora la comprensión de lo que se necesita para el proyecto, sino que también permite asegurar que todos los componentes del sistema estén alineados con los objetivos y estándares aceptados a nivel internacional.

3.2.1. Requisitos operativos del sistema

Los requisitos funcionales especifican las características y capacidades que el sistema debe ofrecer para cumplir con los objetivos del proyecto y permitir la realización de pruebas de optimización del rendimiento. En detalle, estos requisitos incluyen:

- Gestión completa de Infraestructura como Servicio (IaaS): El sistema debe permitir el aprovisionamiento, configuración, monitoreo y desmantelamiento de recursos virtuales como máquinas virtuales (instancias), redes virtuales y almacenamiento.

- Creación y administración de redes virtuales: Se debe posibilitar la creación de múltiples redes virtuales aisladas, con asignación dinámica de direcciones IP mediante DHCP, reglas de firewall y políticas de calidad de servicio (QoS) para priorizar el tráfico.
- Servicios de almacenamiento: El sistema debe soportar almacenamiento en bloque (mediante Cinder) y almacenamiento en objetos (mediante Swift), para cubrir diferentes necesidades de persistencia y acceso a datos por parte de las instancias.
- Interfaz gráfica de usuario: Disponibilidad de una consola web (Horizon) que permita la gestión sencilla de todos los recursos, facilitando el aprendizaje y el manejo de la infraestructura.
- Automatización del despliegue: Uso de un método de despliegue para acelerar el proceso de instalación y configuración, permitiendo reinicios rápidos y ajuste flexible de componentes.
- Monitoreo y análisis: Incorporación de herramientas que permitan monitorear en tiempo real el uso de recursos, rendimiento de instancias y tráfico de red para detectar cuellos de botella o fallos.
- Pruebas de rendimiento y escalabilidad: Capacidad para simular múltiples cargas de trabajo, evaluar el comportamiento de la infraestructura y aplicar técnicas de optimización.

Este conjunto de funcionalidades garantiza que el entorno no solo sea operativo, sino también versátil y apto para la experimentación académica.

3.2.2. Requisitos técnicos y de hardware

El correcto desempeño de la plataforma depende directamente de los recursos físicos disponibles. Es necesario asignar recursos suficientes para evitar problemas de rendimiento y asegurar la estabilidad del sistema base. A continuación, se detallan las especificaciones mínimas y recomendadas de hardware en la **¡Error! No se encuentra el origen de la r eferencia.**:

Tabla 8
Requisitos de hardware

Componente	Especificación mínima (OpenStack producción)	Especificación recomendada (OpenStack producción)	Especificación utilizada	Justificación
Procesador (CPU)	8 núcleos físicos	12 núcleos o más físicos	12 núcleos lógicos	Permite ejecutar de forma concurrente los servicios de control, red, cómputo y almacenamiento de OpenStack
Memoria RAM	16 GB	32 GB	24 GB DDR4	Necesaria para sostener los servicios del plano de control, agentes de red y creación de instancias
Almacenamiento	200 GB SSD	500 GB SSD o más	300 GB SSD	Requerido para imágenes, volúmenes, registros y bases de datos de OpenStack
Red	Ethernet 1 Gbps	Ethernet 10 Gbps	Ethernet 1 Gbps	Garantiza conectividad estable entre servicios y tráfico de instancias

Nota: Detalla las especificaciones mínimas, recomendadas y utilizadas de hardware.

Las especificaciones mínimas, recomendadas y utilizadas del hardware se definieron considerando los requerimientos propios de una infraestructura OpenStack en operación bajo una arquitectura de nodo único, en la cual convergen de forma simultánea los servicios de control, cómputo, red y almacenamiento, a diferencia de un servidor convencional, OpenStack demanda mayores recursos debido a la ejecución concurrente de múltiples servicios como Nova, Neutron, Keystone, Cinder y los sistemas de mensajería y bases de datos asociados, por lo que los valores establecidos responden a criterios técnicos documentados en guías oficiales

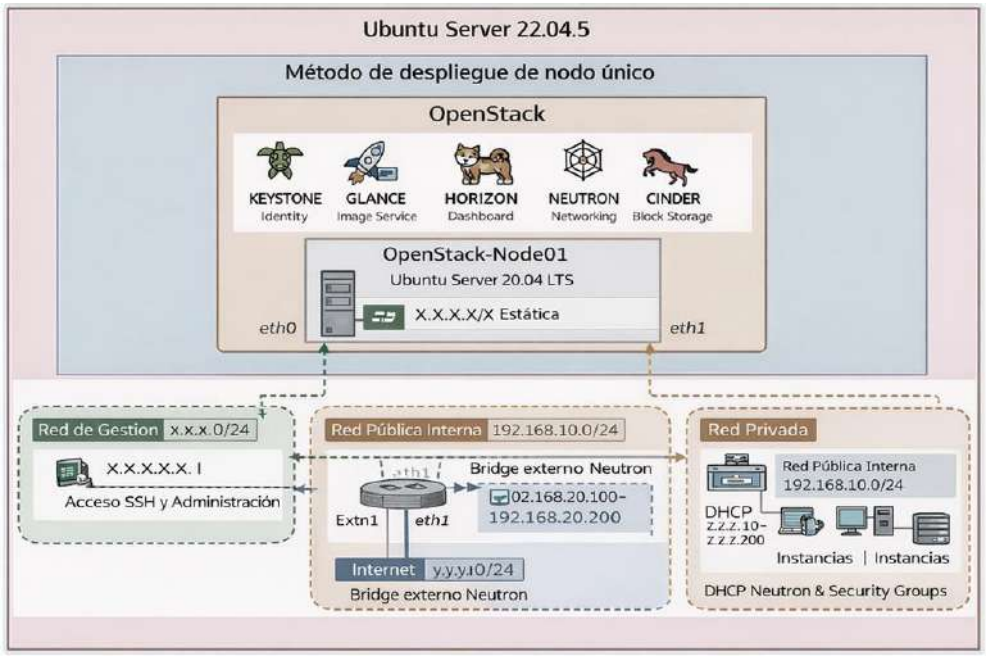
de despliegue y operación de OpenStack para entornos productivos, el dimensionamiento de CPU permite el paralelismo necesario para la gestión de instancias y tráfico de red, la memoria RAM garantiza la estabilidad y continuidad de los servicios del plano de control, el almacenamiento en estado sólido asegura un acceso eficiente a imágenes y volúmenes persistentes, mientras que la capacidad de red definida permite una comunicación estable entre los componentes de la plataforma, asegurando así un funcionamiento adecuado, confiable y alineado con las exigencias de una infraestructura como servicio en producción.

3.2.3. Diseño preliminar de la topología

Debido a las limitaciones del entorno académico y de hardware disponible, la arquitectura de despliegue escogida para este proyecto es una topología de nodo único. Esto significa que todos los componentes principales de OpenStack, incluyendo computación, redes, almacenamiento y gestión de identidad, se ejecutan en una sola máquina virtual, como se ve en la **¡Error! No se encuentra el origen de la referencia.**.

Figura 4

Diagrama de topología del entorno de pruebas Nodo único con servicios integrados



Nota: Diagrama simplificado y claro de OpenStack, mostrando los servicios principales (Nova, Neutron, Cinder/Swift, Keystone, Horizon) y cómo gestionan las instancias virtuales y la red virtual.

Esta aproximación permite un equilibrio entre la complejidad técnica y la factibilidad, facilitando la instalación y administración del entorno sin necesidad de múltiples servidores físicos, como vemos en la **¡Error! No se encuentra el origen de la referencia.**. Características d e la topología:

- Nodo único: Consolida todos los servicios esenciales en una única instancia virtualizada, lo que simplifica la gestión y reduce los requerimientos de hardware.
- Red virtual en modo puente: La máquina virtual está configurada para usar un adaptador en modo puente que permite la comunicación directa con la red local e Internet, permitiendo la interacción realista con otros dispositivos y servicios.
- Redes virtuales internas: OpenStack, mediante Neutron, implementa VLANs y redes segmentadas para aislar el tráfico y asignar direcciones IP dinámicamente a las instancias virtuales.
- Servicios integrados: Keystone para autenticación y control de acceso, Horizon para la interfaz gráfica, Nova para la gestión de cómputo, Cinder y Swift para almacenamiento, y Neutron para redes.

Tabla 9
Comparativa de topologías para entornos OpenStack

Topología	Características	Ventajas	Desventajas	Aplicación en este proyecto
Nodo único	Todos los servicios en un host	Fácil de administrar, bajo costo	Limitado en escalabilidad y redundancia	Escogida para pruebas y simulación local
Multinodo	Servicios distribuidos en varias host	Mayor escalabilidad, alta disponibilidad	Complejidad de configuración y mayor costo	No viable por limitaciones de hardware

Nota: Compara las características entre topologías de nodo único y multimodo.

El diseño del Nodo Único es suficiente para el objetivo académico del proyecto, que es la simulación, prueba y optimización de un entorno IaaS con OpenStack, sin comprometer la capacidad del equipo anfitrión ni la estabilidad del sistema.

3.3. Diseño de la Arquitectura del Sistema

El diseño de la arquitectura del sistema es un paso fundamental para garantizar que la plataforma OpenStack desplegada cumpla con los objetivos de rendimiento, escalabilidad y funcionalidad planteados en esta tesis. Esta etapa implica la definición clara de la estructura lógica y física del entorno, así como la planificación de la distribución de recursos, la configuración de redes y la integración de los diversos servicios que componen la nube privada. En este apartado, se presenta un análisis detallado del diseño conceptual y práctico de la arquitectura, tomando en cuenta las capas del modelo OSI, los escenarios de implementación y la justificación técnica de las decisiones adoptadas.

3.3.1. Selección del método de instalación (DevStack)

La elección de DevStack como entorno de implementación para esta tesis se fundamenta en su simplicidad, rapidez de despliegue y orientación educativa, lo cual lo convierte en una herramienta ideal para entornos de laboratorio y pruebas académicas. Al tratarse de un proyecto mantenido por la comunidad de OpenStack, DevStack permite desplegar una nube funcional en cuestión de minutos, facilitando la experimentación con los distintos servicios que componen una infraestructura como servicio (IaaS).

Tabla 10
Tabla comparativa de métodos de instalación de OpenStack

Criterio	DevStack	MicroStack	Packstack	Instalación Manual	Kolla-Ansible
Requiere pocos recursos de hardware	Sí	Sí	-	-	-

Fácil instalación para entornos académicos	Sí	Sí	Parcial	-	-
Permite modificar o controlar servicios fácilmente	Sí	-	-	Sí	Sí
Recomendado para pruebas de laboratorio	Sí	Parcial	Parcial	-	-
Nodo Único	Sí	Sí	-	-	-

Nota: Se eligió DevStack como método de instalación porque es el más adecuado para entornos educativos y hardware limitado. Su despliegue rápido, facilidad de personalización y bajo consumo de recursos lo hacen ideal para el entorno del proyecto. Además, permite trabajar con los servicios principales de OpenStack sin requerir una infraestructura compleja de múltiples nodos.

DevStack es especialmente útil en investigaciones como esta, donde se busca explorar técnicas de optimización del rendimiento, ya que permite modificar configuraciones de red, computación y almacenamiento de forma ágil. Gracias a su diseño modular, los estudiantes pueden activar o desactivar servicios específicos, entender cómo interactúan entre sí y realizar ajustes en tiempo real, lo cual enriquece el proceso de aprendizaje práctico como se puede ver en la **¡Error! No se encuentra el origen de la referencia.**.

Aunque no está pensado para entornos de producción, DevStack ofrece una plataforma flexible y controlada para realizar pruebas, documentar procesos y desarrollar guías que otros estudiantes podrán seguir para comprender mejores conceptos como la calidad de servicio (QoS), la gestión del tráfico o la implementación de redes definidas por software (SDN).

En resumen, DevStack se alinea perfectamente con los objetivos académicos de esta tesis al ofrecer un entorno accesible, versátil y didáctico, que facilita el estudio y la aplicación de técnicas de optimización en IaaS basadas en OpenStack.

3.3.2. Diseño conceptual y topológico

El diseño conceptual parte de la necesidad de representar un entorno IaaS completo, que permita el despliegue y gestión de recursos virtualizados a través de OpenStack, operando sobre una infraestructura física limitada y bajo un esquema de nodo único.

Características del diseño conceptual:

- Modularidad: La arquitectura se basa en módulos o servicios independientes que se comunican entre sí mediante APIs. Esto permite mayor flexibilidad y facilita futuras ampliaciones o actualizaciones.
- Centralización: Dado que se utiliza una topología de nodo único, todos los servicios esenciales de OpenStack (computación, red, almacenamiento y gestión) se concentran en un solo nodo lógico, alojado en una máquina virtual.
- Automatización: El uso de DevStack permite automatizar la instalación, configuración y puesta en marcha del entorno, reduciendo errores y agilizando el proceso.
- Separación de capas: Se sigue el modelo OSI para diseñar las redes y servicios, garantizando la correcta segmentación y comunicación entre capas.

El diseño topológico, basado en estos principios, contempla:

- Un nodo único que actúa como controlador y proveedor de recursos.
- Instancias virtuales que se crean y administran mediante el servicio Nova.
- Redes virtuales gestionadas por Neutron, con soporte para DHCP.
- Servicios de almacenamiento accesibles para las instancias.
- Interfaz gráfica Horizon para la administración y gestión por parte del usuario.

3.3.3. Escenarios de implementación propuestos

Para asegurar la validación y flexibilidad del diseño, se plantean tres escenarios de implementación, cada uno con características y objetivos específicos, como se puede ver en la **¡Error! No se encuentra el origen de la referencia.**.

Tabla 11
Escenarios de implementación propuestos

Escenario	Descripción	Objetivo

Escenario 1: Entorno base	Despliegue inicial de OpenStack en nodo único con servicios mínimos.	Validar la instalación y configuración básica.
Escenario 2: Simulación de carga	Creación de múltiples instancias y redes para evaluar rendimiento y QoS.	Analizar el comportamiento bajo carga y optimizar recursos.
Escenario 3: Entorno optimizado	Aplicación de configuraciones avanzadas de QoS, balanceo y monitoreo.	Mejorar la eficiencia, reducir latencia y garantizar estabilidad.

Nota: Esta tabla muestra los tres escenarios que se va a trabajar, con una pequeña descripción y objetivo.

Estos escenarios permiten una progresión lógica y ordenada en la implementación, facilitando la identificación y corrección de problemas en etapas tempranas, y garantizando un entorno funcional y eficiente para la simulación.

3.3.4. Análisis por capas del modelo OSI

El modelo OSI es una herramienta fundamental para el diseño de redes y servicios, ya que permite entender y gestionar cada aspecto de la comunica

ción de forma segmentada.

- **Capa 2 (Enlace de Datos):** En esta capa se diseñan las redes virtuales que conectan las instancias dentro del nodo único. Se implementan VLANs para segmentar el tráfico y evitar interferencias, además de bridges para interconectar las máquinas virtuales con la red física. OpenStack Neutron gestiona estos aspectos, permitiendo configuraciones flexibles y dinámicas.

- **Capa 3 (Red):** Aquí se definen las políticas de direccionamiento IP, enrutamiento y asignación dinámica de direcciones mediante DHCP. Además, se aplican reglas de

firewall y seguridad para controlar el acceso y proteger la infraestructura. La correcta gestión en esta capa es vital para asegurar conectividad eficiente y segura.

Capa 7 (Aplicación):

A nivel de aplicación se encuentran los servicios de OpenStack (Nova, Cinder, Keystone, Horizon, etc.), que proporcionan las funcionalidades de gestión, control y usuario final. Esta capa es la interfaz con el usuario y la que orquesta todos los procesos internos de la nube.

3.3.5. Justificación de la arquitectura seleccionada

La elección de una arquitectura de nodo único para este proyecto se justifica por varios factores:

- Dado que la infraestructura OpenStack se implementa bajo una arquitectura de nodo único, los recursos disponibles se gestionan de forma centralizada en un solo servidor, por lo que la distribución de los servicios en múltiples nodos no resulta necesaria para los objetivos definidos, este enfoque permite optimizar el uso de CPU, memoria, almacenamiento y red, garantizando un funcionamiento estable y controlado de los servicios del plano de control y de datos sin introducir complejidad adicional en la administración de la infraestructura.
- Facilidad de administración: Centralizar los servicios en un solo nodo simplifica la instalación, configuración y mantenimiento, aspectos importantes en un proyecto con plazos y recursos definidos.

- Flexibilidad y escalabilidad: Aunque el nodo único tiene limitaciones para producción, permite aplicar y probar muchas de las funcionalidades y configuraciones avanzadas de OpenStack, siendo un buen punto de partida para futuros proyectos más complejos.

- Compatibilidad con herramientas educativas: La arquitectura se alinea con el uso de DevStack, una herramienta que facilita el aprendizaje y experimentación, permitiendo a estudiantes y docentes comprender el funcionamiento de OpenStack sin la complejidad de un despliegue multinodo.

3.4. Configuración y Despliegue de OpenStack

La configuración y despliegue de OpenStack constituyen una etapa fundamental dentro del desarrollo del proyecto, ya que permiten materializar la arquitectura definida y habilitar la operación de la infraestructura como servicio en un entorno productivo, en este subtema se describen los procedimientos y criterios técnicos aplicados para la instalación de los servicios principales de OpenStack, considerando aspectos de estabilidad, compatibilidad y correcto funcionamiento del plano de control y de datos, asegurando así una plataforma operativa capaz de gestionar recursos de cómputo, red y almacenamiento de manera centralizada y eficiente.

3.4.1. Prestaciones del entorno OpenStack implementado

Dentro del desarrollo del proyecto, se definen las prestaciones del entorno OpenStack implementado con el objetivo de describir las capacidades técnicas que ofrece la infraestructura antes de su fase de operación y evaluación, estas prestaciones se establecen a partir de la configuración real del sistema y del dimensionamiento de los recursos de hardware y software, permitiendo caracterizar el comportamiento esperado de la plataforma bajo condiciones de carga propias de un entorno de producción

La infraestructura se encuentra diseñada bajo una arquitectura de nodo único, lo cual permite una gestión centralizada de los servicios del plano de control y de datos, optimizando el uso de

los recursos disponibles y reduciendo la complejidad administrativa, la capacidad de procesamiento y memoria asignada permite la ejecución simultánea de los servicios principales de OpenStack y la creación de múltiples instancias, garantizando estabilidad operativa y continuidad del servicio

Asimismo, la utilización de almacenamiento en unidades de estado sólido proporciona un acceso eficiente a imágenes, volúmenes persistentes y registros del sistema, mientras que la gestión de red mediante Neutron permite la segmentación lógica del tráfico, la asignación dinámica de direcciones IP y la aplicación de reglas de seguridad, estas prestaciones establecen la base técnica sobre la cual se evaluará posteriormente el rendimiento y comportamiento de la infraestructura durante su operación.

Tabla 12
Prestaciones de la infraestructura OpenStack

Prestación	Valor implementado	Descripción técnica
Capacidad de procesamiento	12 núcleos lógicos (3 físicos * 4 virtuales)	Permite la ejecución concurrente de los servicios del plano de control y la gestión de múltiples instancias
Capacidad de memoria	24 GB RAM	Garantiza estabilidad operativa de los servicios principales de OpenStack bajo carga sostenida
Almacenamiento persistente	300 GB SSD	Proporciona acceso rápido a imágenes, volúmenes y registros del sistema
Gestión de red	Neutron (automático)	Permite segmentación del tráfico, direccionamiento dinámico y control de acceso
Conectividad externa	Ethernet 1 Gbps	Habilita comunicación estable con redes externas y acceso controlado a instancias
Gestión centralizada	Horizon	Facilita la administración y supervisión de la infraestructura desde una interfaz unificada
Seguridad lógica	Grupos de seguridad (Horizon)	Controla el tráfico permitido entre instancias y hacia redes externas

Nota: Las prestaciones descritas corresponden a la configuración real del entorno OpenStack implementado y definen las capacidades técnicas del sistema previo a su fase de evaluación.

Las prestaciones definidas en este apartado permiten establecer un marco técnico claro para el desarrollo del proyecto, ya que describen las capacidades operativas del entorno OpenStack antes de la ejecución de pruebas y mediciones, estas características sirven como referencia para comprender el comportamiento esperado de la infraestructura y facilitan la interpretación de los resultados que serán analizados en etapas posteriores.

3.4.1.1.Parámetros funcionales de referencia para entornos reales

Con el objetivo de contar con una referencia técnica que permita realizar comparaciones posteriores y aplicar fórmulas de ajuste, se establecen parámetros funcionales reales correspondientes a una infraestructura OpenStack en producción, estos valores representan un escenario operativo adecuado en el cual los servicios del plano de control, red, cómputo y almacenamiento pueden ejecutarse de forma estable y continua, la definición de estos parámetros se realizó considerando las recomendaciones oficiales para despliegues OpenStack de nodo único tal como se muestra en la **¡Error! No se encuentra el origen de la referencia.**, y a que este tipo de arquitectura concentra múltiples servicios en un mismo sistema y requiere un dimensionamiento adecuado de recursos para evitar degradación del rendimiento, de esta manera, los parámetros definidos sirven como base técnica para evaluar el comportamiento del entorno implementado y relacionarlo con condiciones reales de operación (openstack r, 2013).

Tabla 13
Parámetros funcionales reales de referencia para infraestructura OpenStack

Parámetro	Valor funcional de referencia	Descripción técnica
Procesador (CPU)	12 núcleos físicos	Capacidad de procesamiento considerada adecuada para la ejecución concurrente de los servicios del plano de control, red, cómputo y almacenamiento en una arquitectura OpenStack de nodo único
Memoria RAM	32 GB	Capacidad de memoria utilizada como referencia para garantizar estabilidad operativa de los servicios principales de OpenStack bajo carga sostenida

Parámetro	Valor funcional de referencia	Descripción técnica
Almacenamiento	500 GB SSD	Capacidad funcional destinada a la gestión de imágenes, volúmenes persistentes, registros del sistema y crecimiento controlado de la infraestructura
Tipo de almacenamiento	SSD	Tecnología de almacenamiento seleccionada por su baja latencia y alto rendimiento en operaciones de lectura y escritura
Conectividad de red	Ethernet 10 Gbps	Capacidad de red considerada funcional para soportar tráfico administrativo, comunicación entre servicios y tráfico generado por instancias

Nota: Los parámetros funcionales definidos en esta tabla representan un escenario de referencia para una infraestructura OpenStack en producción y serán utilizados posteriormente para la aplicación de fórmulas de ajuste y comparación con el entorno implementado.

Los valores funcionales definidos para CPU, memoria RAM, almacenamiento y conectividad se establecen debido a que una infraestructura OpenStack en producción concentra servicios que permanecen activos de forma continua y que operan de manera concurrente, en una arquitectura de nodo único el mismo sistema ejecuta componentes del plano de control, servicios de red y procesos asociados al cómputo y al almacenamiento, por lo cual el dimensionamiento debe garantizar estabilidad operativa y capacidad de respuesta ante variaciones de carga, en este sentido, el valor funcional de doce núcleos permite distribuir procesos y atender múltiples operaciones simultáneas, la referencia de treinta y dos gigabytes de RAM asegura margen suficiente para servicios como bases de datos, mensajería y agentes de red, el almacenamiento SSD con quinientos gigabytes se considera adecuado para sostener imágenes, volúmenes y registros sin comprometer tiempos de acceso, mientras que una conectividad de diez gigabits por segundo representa un escenario consistente con operación productiva donde la red no se convierte en el principal cuello de botella.

3.4.1.2. Comparación con la infraestructura implementada

Una vez definidos los parámetros funcionales reales, se realiza una comparación directa con los recursos disponibles en la infraestructura implementada, esta comparación permite

identificar la relación de escalamiento entre el entorno implementado y el escenario funcional de referencia, lo cual resulta necesario para interpretar resultados y aplicar posteriormente factores de ajuste, la comparación se enfoca en CPU, RAM, almacenamiento y red, ya que son los recursos que condicionan de forma directa el rendimiento de instancias, el comportamiento del tráfico y la respuesta de los servicios de OpenStack, a partir de esta relación se obtiene un factor por recurso que permitirá normalizar métricas como throughput, latencia, uso de CPU o consumo de memoria en función de condiciones de operación equivalentes como se indica en la **¡Error! No se encuentra el origen de la referencia.**.

Tabla 14
Comparación de parámetros funcionales reales frente a parámetros implementados

Recurso	Valor funcional real de referencia	Valor implementado	Relación de escalamiento
CPU	12 núcleos físicos	12 núcleos lógicos (3 físicos y 4 virtuales)	12 / 3 = 4.00
RAM	32 GB	24 GB	32 / 24 = 1.33
Almacenamiento	500 GB SSD	300 GB SSD	500 / 300 = 1.67
Red	10 Gbps	1 Gbps	10 / 1 = 10.00

Nota: La relación de escalamiento representa el factor necesario para llevar el recurso implementado a condiciones equivalentes al parámetro funcional real definido como referencia.

3.4.1.3. Justificación de los parámetros funcionales reales

Los parámetros funcionales reales definidos previamente se establecen con el propósito de representar un escenario operativo acorde a una infraestructura OpenStack en producción. Estos valores permiten disponer de una referencia técnica clara para interpretar los resultados obtenidos durante el desarrollo del proyecto y evitar que las limitaciones propias del entorno implementado afecten la validez del análisis.

La selección de estos parámetros responde a la necesidad de asegurar la ejecución simultánea de los servicios del plano de control, red, cómputo y almacenamiento. En una arquitectura de nodo único, todos estos componentes se concentran en un mismo sistema, por lo que un dimensionamiento adecuado resulta fundamental para garantizar estabilidad y continuidad del servicio.

- **Modelo de normalización y escalamiento de métricas de rendimiento**

Para relacionar las mediciones obtenidas con el escenario funcional real de referencia, se utiliza un modelo de normalización y escalamiento de métricas de rendimiento. Este modelo permite ajustar los valores medidos considerando la diferencia existente entre los recursos implementados y los parámetros funcionales definidos.

El factor de escalamiento proporcional de recursos se calcula mediante una relación de normalización basada en principios de análisis de rendimiento de sistemas, expresada como el cociente entre el valor funcional real de referencia y el valor del recurso implementado (Malkawi, 2013).

El factor de escalamiento para cada recurso se calcula mediante la **¡Error! No se encuentra el o rigen de la referencia.**:

Ecuación 1

Factor de escalamiento proporcional de recursos

$$\boldsymbol{F_R} = \frac{R_{funcional}}{R_{implementado}}$$

Nota: F_R representa el factor de escalamiento del recurso, $R_{funcional}$ corresponde al valor funcional real de referencia y $R_{implementado}$ al valor del recurso disponible en la infraestructura utilizada.

Esta **¡Error! No se encuentra el origen de la referencia.** ya fue aplicada en la **¡Error! No se en cuentra el origen de la referencia.** en el apartado de Relación de escalamiento, para cada recurso utilizado.

- **Ajuste proporcional de métricas de capacidad**

Con el fin de estimar el comportamiento de la infraestructura bajo condiciones funcionales reales de producción, se aplica un ajuste proporcional de las métricas de capacidad obtenidas durante la implementación. Este procedimiento permite transformar los valores medidos en el entorno utilizado a un escenario de referencia, considerando la diferencia de capacidad existente entre ambos entornos y utilizando el factor de escalamiento previamente definido.

Este ajuste resulta especialmente útil en métricas de capacidad, como el throughput o la cantidad de datos procesados por unidad de tiempo, ya que estas variables tienden a escalar de manera directamente proporcional al recurso dominante, siempre que no existan cuellos de botella adicionales que limiten el rendimiento del sistema.

La capacidad ajustada se calcula mediante la **¡Error! No se encuentra el origen de la r eferencia.**:

Ecuación 2

Fórmula de ajuste proporcional de métricas de capacidad

$$C_{ajustada} = C_{media} \times F_R$$

Nota: $C_{ajustada}$ representa la capacidad estimada bajo condiciones funcionales reales, C_{media} corresponde a la capacidad medida en la infraestructura implementada y F_R es el factor de escalamiento proporcional del recurso dominante.

Esta Ecuación nos va a servir para después hallar los resultados reales, los cuales los veremos en el apartada de CAPÍTULO IV: Operación y Resultados.

3.4.2. Preparación del entorno y configuración inicial

La preparación del entorno y la configuración inicial son pasos fundamentales para asegurar que el sistema sea capaz de ejecutar OpenStack de forma eficiente. Esta fase incluye la instalación de todos los componentes necesarios en el sistema operativo base (Ubuntu Server 22.04 LTS), la configuración de los recursos del sistema y la instalación de las dependencias

necesarias para garantizar que OpenStack funcione correctamente. La correcta ejecución de esta fase es clave, ya que sienta las bases para el despliegue de los servicios de OpenStack.

3.4.2.1 Instalación del sistema operativo

El primer paso para preparar el entorno de pruebas fue la instalación del sistema operativo Ubuntu Server 22.04 LTS. Ubuntu fue seleccionado debido a su compatibilidad con las últimas versiones de OpenStack y su estabilidad, lo que lo convierte en una excelente opción para este tipo de implementaciones. La instalación del sistema operativo se realizó siguiendo los siguientes pasos:

- **Descarga de la imagen ISO:**

Se descargó la última versión de Ubuntu Server 22.04 LTS desde el sitio web oficial de Ubuntu https://ubuntu.com/download/server . Esta versión está optimizada para servidores y entornos de nube especialmente porque tiene en vigencia el soporte para OpenStack.

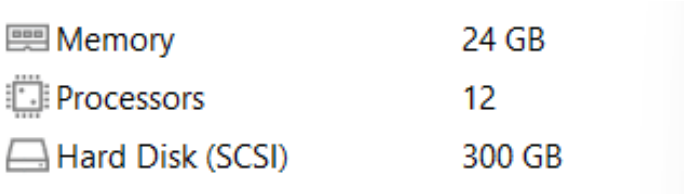
- **Configuración del Host para que sea un sistema de control y orquestación de recursos de cloud computing (Hardware):**

La infraestructura se implementó a partir de la instalación directa de Ubuntu Server 20.04 LTS como sistema operativo base, sobre el cual se configuraron los servicios necesarios para el despliegue de OpenStack, garantizando estabilidad, compatibilidad y soporte extendido para una operación continua en un entorno de producción.

1. Memoria RAM: Se asignaron 12 GB de RAM para permitir que OpenStack y todos los servicios se ejecuten de manera eficiente.
2. CPU: Se asignaron 12 núcleos lógicos de CPU para que el sistema tenga suficiente capacidad de procesamiento de los cuales son 3 físicos por 4 virtuales.
3. Almacenamiento: Se creó un disco duro virtual con 100 GB de espacio, que fue particionado adecuadamente para instalar Ubuntu y los componentes de OpenStack.

Como podemos ver en los recursos de la máquina **¡Error! No se encuentra el origen de la r eferencia.**.

Figura 5
Recursos asignados de la máquina



Nota: Aquí vemos claramente los recursos que le se le puedo asignar a la maquina con las prestaciones actuales.

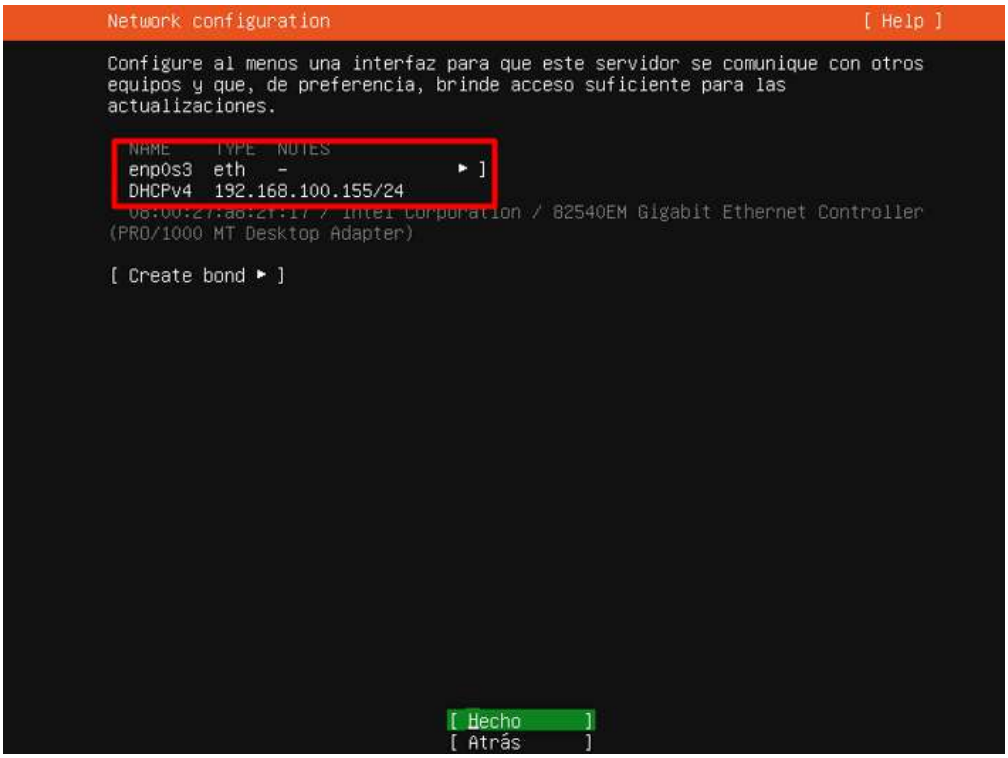
- **Instalación del sistema operativo a nivel de Software:**

Durante la instalación de Ubuntu, se seleccionaron las opciones mínimas necesarias, incluyendo:

1. Configuración de la red: Se configuró el sistema para usar una red estática **¡Error! No s e encuentra el origen de la referencia.**, ya que al ser un sistema de control y orquestación de recursos de cloud computing de un solo nodo, su salida a internet debe ser nítida, en otras palabras, esta debe ser única.

Figura 6

Configuración de la tarjeta de red del SO base



Nota: En este apartado de la instalación es en donde se pone una IP estática ya que es el punto de salida de OpenStack.

2. SSH: Es muy importante activar esta opción al instalar el SO ya que nos permitirá por medio de consola conectarnos de una forma rápida y segura **¡Error! No se encuentra el origen de la referencia.**

Figura 7

Activación del servidor SSH

Nota: Seleccionar la opción de Instalar Servidor OpenSSH de una vez, ya que después se requerirá.

3. Actualización de paquetes: Una vez instalado, se configuraron las actualizaciones automáticas para garantizar que el sistema esté siempre al día con los últimos parches de seguridad.

3.4.2.2. Actualización y preparación del sistema:

Después de completar la instalación de Ubuntu, el siguiente paso fue asegurarse de que el sistema estuviera completamente actualizado y preparado para recibir los componentes de OpenStack. Este paso es esencial para evitar problemas de compatibilidad y asegurar que todas las dependencias estén satisfechas.

- Actualización del sistema operativo:

Se ejecutaron los siguientes comandos para actualizar todos los paquetes a las versiones más recientes:

- *sudo apt update*
- *sudo apt upgrade -y*
- Instalación de herramientas esenciales:

Para facilitar la instalación de OpenStack y sus dependencias, se instalaron algunas herramientas básicas que serían necesarias durante el proceso de despliegue:

- Git: Para clonar el repositorio de DevStack desde GitHub.
- Python: OpenStack requiere Python 3.x, por lo que se instaló la versión más reciente compatible con Ubuntu Server 22.04 LTS.

Otros paquetes de desarrollo: Algunos paquetes adicionales fueron instalados para asegurar que el sistema tuviera las librerías necesarias para compilar e instalar ciertos componentes.

Los comandos para instalar estas herramientas fueron los siguientes:

- *sudo apt install python3-pip python3-dev git -y*

- **Verificación de la red y la conectividad:**

Antes de continuar con la instalación de OpenStack, se verificó que el sistema operativo tuviera acceso a Internet y pudiera resolver nombres de dominio. Se realizaron pruebas de conectividad utilizando comandos como ping y nslookup para asegurar que el sistema tuviera una conexión de red activa:

- *ping google.com*
- *ping 8.8.8.8*

En el caso que nuestro sistema operativo no pueda hacer ping a Google, es por falta de salida de dominios, en este caso tenemos que configurar el archivo *resolf.conf,* aumentando las líneas *nameserver 8.8.8.8* y *nameserver 8.8.4.4.* Necesarios para que el SO pueda resolver por dominio.

3.4.3. Configuración e Instalación de servicios principales

Una vez que el sistema operativo base (Ubuntu Server 22.04.5 LTS) está instalado y actualizado, y se han cumplido los requisitos de software (Python, Git y herramientas de desarrollo), se procede a la instalación de los servicios principales de OpenStack utilizando DevStack. Esta herramienta automatiza el proceso de despliegue, simplificando significativamente la instalación de una nube funcional para fines académicos o de prueba.

- **Preparación del entorno con DevStack**

Primero, se creó un nuevo usuario llamado stack, que es requerido por DevStack para ejecutar todos los servicios. Esto se realiza con los siguientes comandos:

- *sudo useradd -s /bin/bash -d /opt/stack -m stack*
- *echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack*

Luego se inició sesión con dicho usuario:

- *sudo su - stack*

Después, se clonó el repositorio oficial de DevStack desde OpenDev:

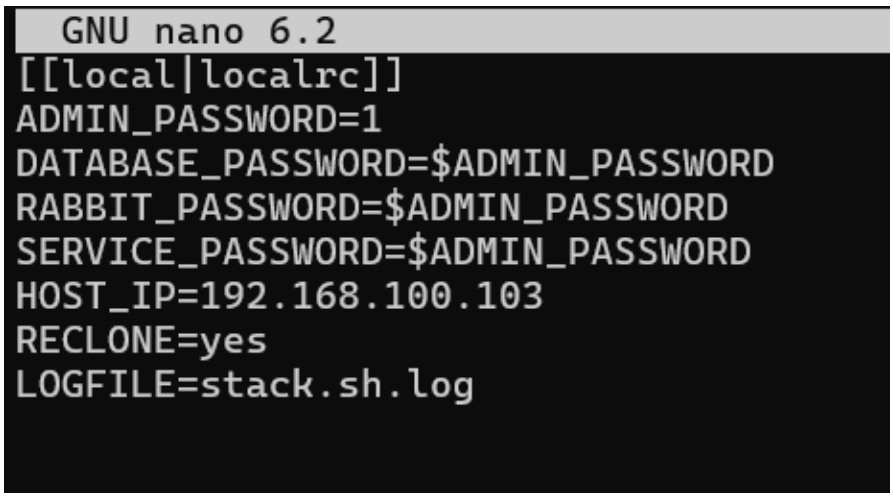
- *git clone https://opendev.org/openstack/devstack*
- *cd devstack*

Se creó el archivo de configuración local.conf, en el cual se define qué servicios se instalarán, así como los usuarios y contraseñas:

- *nano local.conf*

Para este caso el contenido del archivo fue el siguiente **¡Error! No se encuentra el origen de l a referencia.** para un entorno sin balanceador:

Figura 8
Contenido del archivo local.conf sin balanceador



Nota: Aquí se ve el contenido que debe ir en el archivo local.conf. Sin balanceador.

Finalmente, se ejecutó el script principal de instalación:

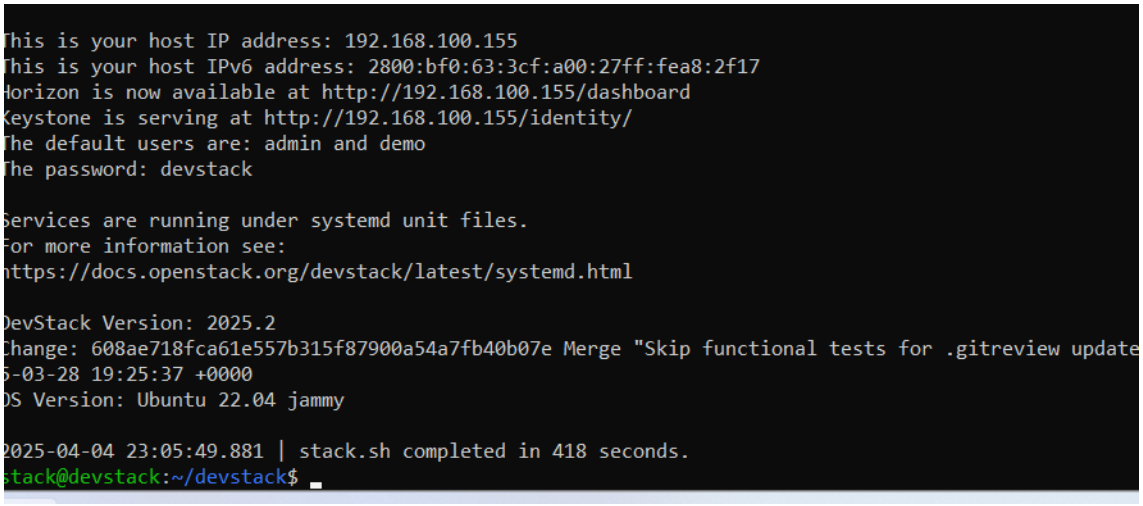
- ./stack.sh

Este proceso puede demorar entre 30 y 45 minutos dependiendo del hardware y la velocidad de red, ya que se descargan paquetes, se crean servicios y se configuran automáticamente.

Una vez hecho esto, al final de ejecutar el comando, aparecerá lo siguiente:

Figura 9

Fin de instalación de devstack



Nota: Mensaje de que ya se puede utilizar OpenStack.

Una vez probado esto, ingresamos por medio del dashboard. Aquí hacemos un paréntesis, ya que esta no es la instalación final. En el archivo local.conf debe cargarse el balanceador de carga, pero como recomendación se debe correr sin este primero para ver si nuestra maquina es compatible y lo mas importante, si con los recursos que posee, se podrá trabajar.

Al poder ingresar, no es necesario probar, ya que automáticamente este se levanta solo, ahora vamos a ejecutar estos comandos:

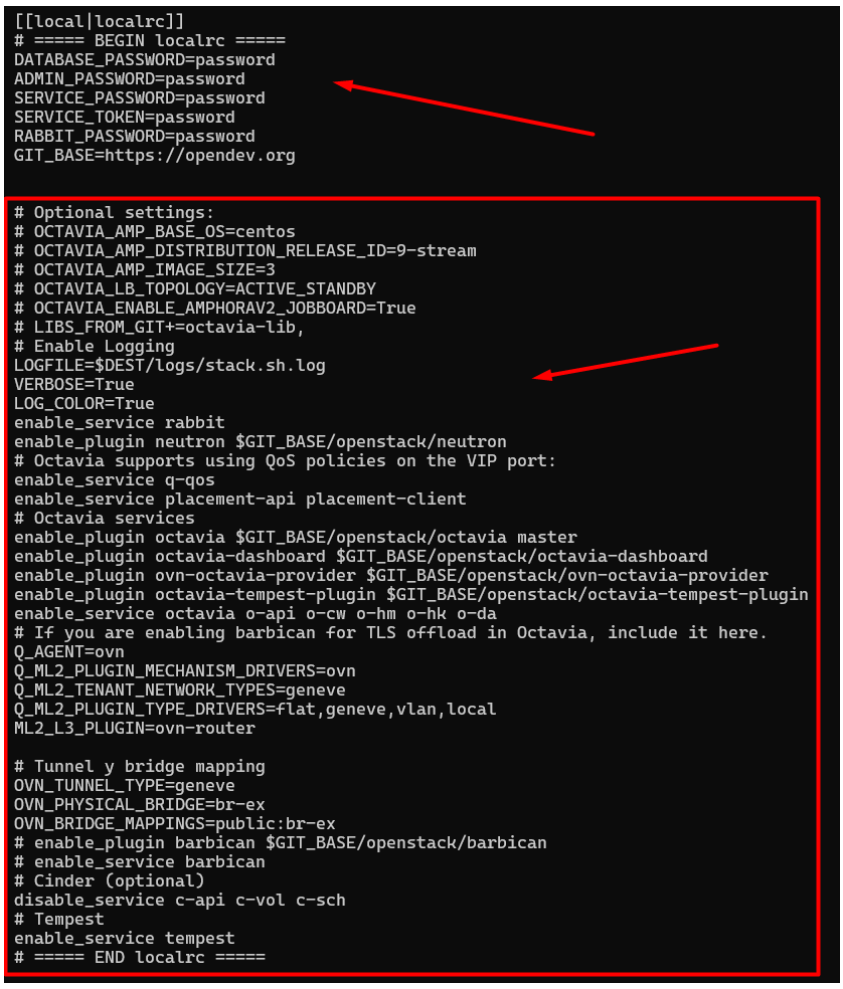
- *./unstack.sh* y *./clean.sh*

Con esto paremos la ejecución actual y podremos configurar el archivo local.conf, para que instale el balanceador.

No es recomendable manipular el archivo local.conf cuando está en ejecución ya que después se tendrá problemas, primero paramos ejecución y limpiamos memoria cache. Para poder aplicar los cambios necesarios. De esta forma obtenemos este nuevo archivo **¡Error! No se encuentra el origen de la referencia.**, todo esto está en la documentación oficial de OpenStack https://docs.openstack.org/2025.2/ .

Figura 10

Contenido del archivo local.conf con balanceador



Nota: Se observa las librerías necesarias para que el balanceador pueda funcionar, esto depende del sistema operativo y del tipo de instalación de OpenStack.

Con este archivo modificado lo siguiente el volver a ejecutar ./stack.sh.Ahora todo esto es muy complejo y tarda mucho, por esta razón se optó por realizar un scrit, que nos permita hacer todo

esto de un solo paso, para esto ejecutamos hacemos ejecutable el archivo de instalación breve, en este caso se utilizó:

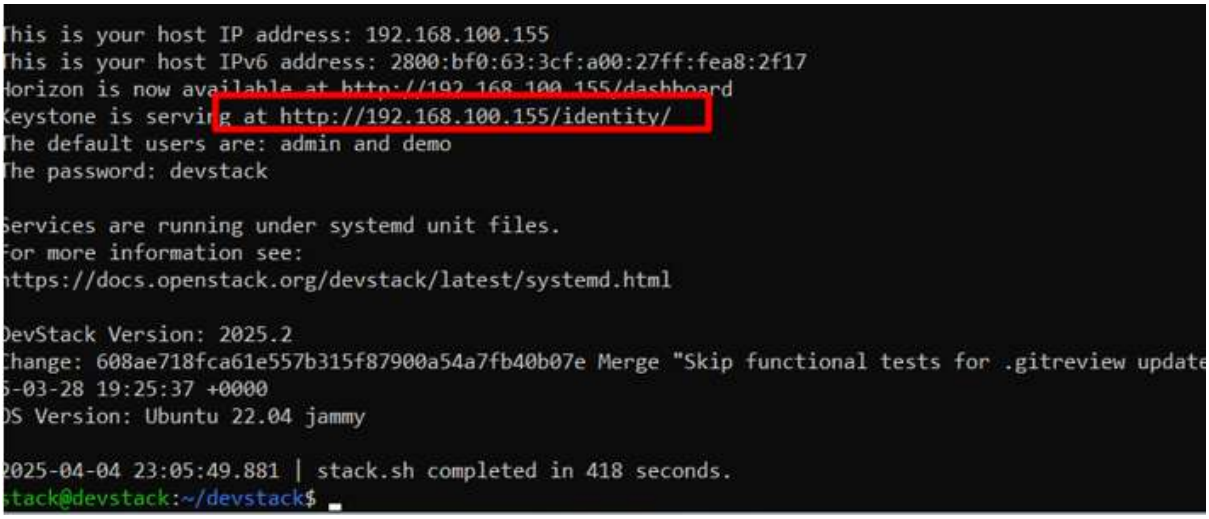
- chmod +x instalación.sh
- ./ instalación.sh

El contenido de este archivo lo encontramos en Contenido del scrit de instalación rápida: instalacion.sh.

Acabado el proceso, manualmente ingresamos al usuario stack y después en la carpeta devstack **¡Error! No se encuentra el origen de la referencia.**, ejecutamos denuevo ./stack. Al final v erificamos en la terminal

Figura 11

Contenido de que la ejecución de devstack, ya está operativo



Nota: Este es el mensaje que debe aparecer, sin errores.

Si sale algún error, la instalación y ejecución no valdrá. Así podamos entrar a dashboard, de preferencia siempre bajar las imágenes del SO de la página oficial.

- **Funcionamiento de cada servicio principal**

1. Nova (Servicio de cómputo)

Durante la ejecución de ./stack.sh, Nova se instala como uno de los servicios básicos. DevStack descarga sus componentes y configura el hipervisor por defecto (KVM). La verificación de que Nova funciona correctamente se puede hacer desde la terminal con:

- *openstack compute service list*

2. Neutron (Servicio de red)

Neutron también es desplegado automáticamente y se encarga de gestionar redes virtuales, routers y direcciones IP. DevStack crea por defecto una red private y un router1 para conexión externa. Desde la terminal se puede validar con:

- *openstack network list*
- *openstack router list*

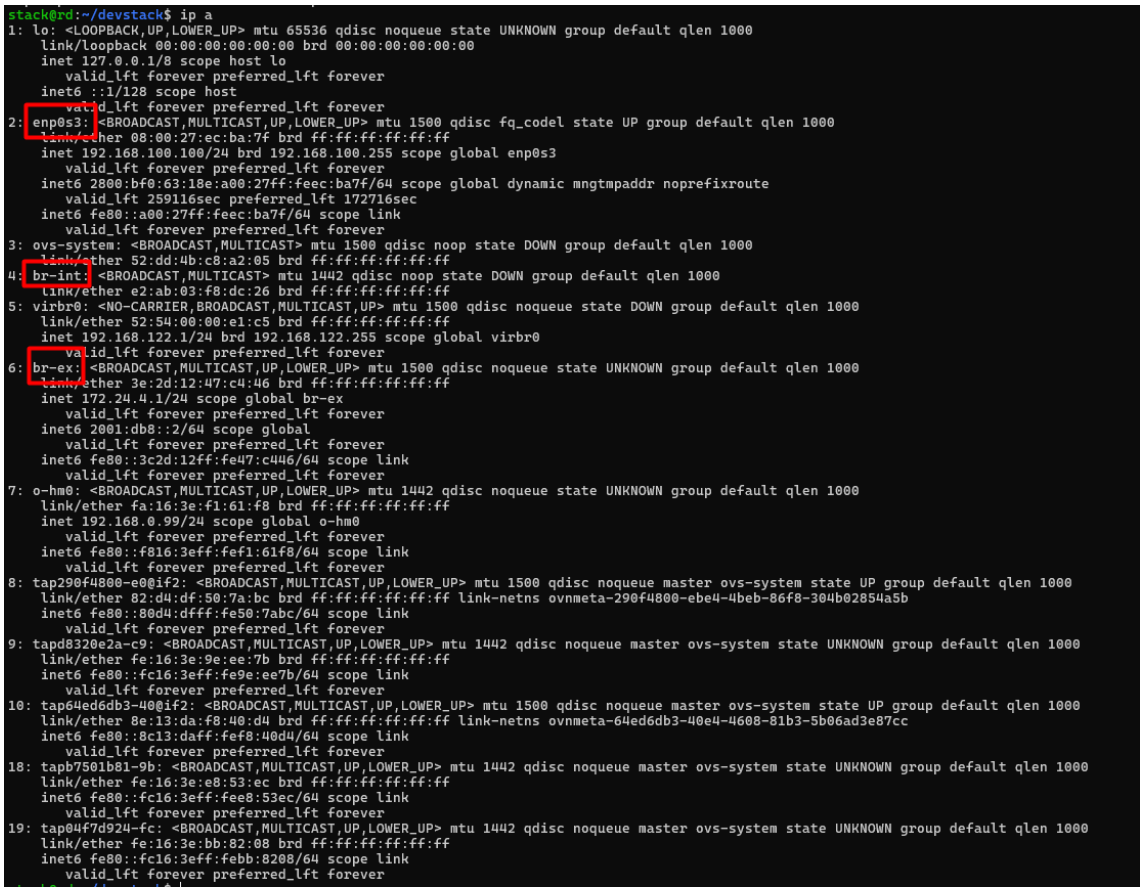
También se puede modificar la configuración de Neutron en el archivo local.conf si se requieren VLANs o redes externas más específicas. De recomendación esto se aplica cuando es multi nodo. Por lo general se establece las subredes de la siguiente forma:

- Red interna Privada 10.0.0.0/24
- Red interna Publica 192.168.233.0/24
- Red de IP flotante 172.24.4.0/24

Red externa, es la red que nos da salida a internet, esta se le asigna una IP estatica al momento de la configuración del equipo principal, para tener una mejor comprensión, en la **¡Error! No s e encuentra el origen de la referencia.** vemos la creación virtual de las redes.

Figura 12

Configuración de las redes y puentes automática en DevStack



Nota: Se observa todas las redes y puentes creados al momento de ejecutar la instalación de OpenStack, mediante DevStack

3. Cinder (Almacenamiento en bloque)

Cinder permite a las instancias usar volúmenes persistentes. Su instalación también es automática con DevStack. Los volúmenes se prueban con:

- *openstack volume create --size 1 test-volume*

Se debe asegurar que el backend de almacenamiento esté funcionando, por lo general como LVM o disco loop local.

4. Keystone (Identidad y autenticación)

Keystone es el primer servicio que se instala, ya que gestiona usuarios y tokens de autenticación. Los usuarios admin y demo se crean por defecto, y desde la terminal se puede comprobar con:

- *openstack user list*
- *openstack project list*

Desde local.conf se define la contraseña y parámetros de Keystone. También se puede extender con más dominios o roles personalizados.

5. Horizon (Interfaz gráfica web)

Horizon es el panel de administración web. Una vez finalizada la instalación, se accede desde un navegador en el host físico mediante la IP estática de nuestra máquina:

- http://192.168.100.35/dashboard

Las credenciales por defecto son:

- Usuario: admin
- Contraseña: password

Desde Horizon se pueden crear instancias, volúmenes, redes y usuarios de manera visual, ideal para prácticas educativas.

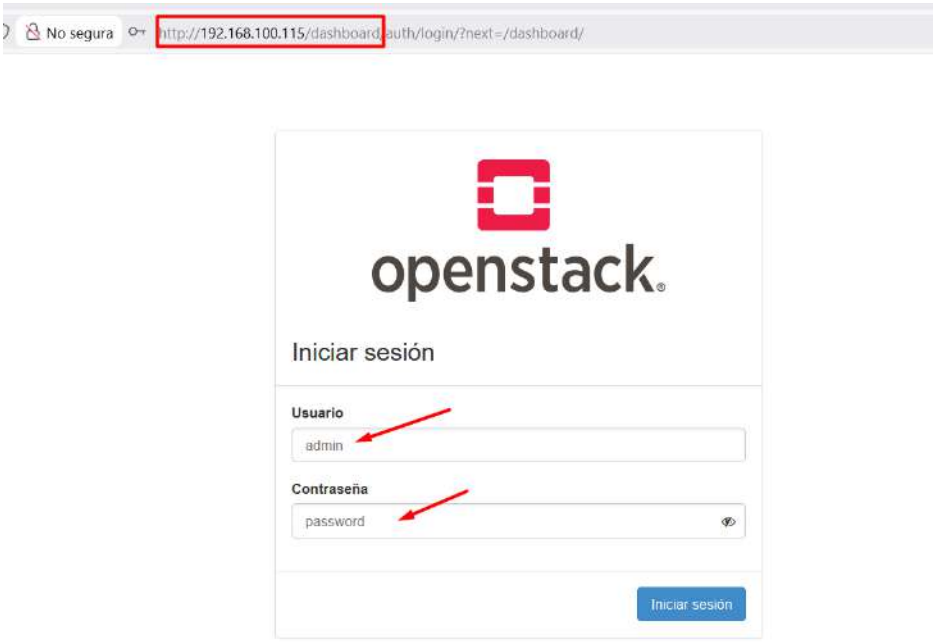
En conjunto, estos servicios forman la base para una nube IaaS funcional. Gracias al uso de DevStack, se logró configurar este entorno en un nodo único, ideal para realizar pruebas,

simular escenarios de red y explorar técnicas de optimización del rendimiento como QoS, balanceo de carga o monitoreo del tráfico.

3.4.4. Horizon (Dashboard).

El dashboard de OpenStack, conocido como Horizon, constituye la interfaz gráfica de administración de la infraestructura como servicio implementada **¡Error! No se encuentra el o rigen de la referencia.**, a través de este panel es posible gestionar de manera centralizada los recursos de cómputo, red y almacenamiento, facilitando la interacción del usuario con los distintos servicios que conforman la plataforma. El uso del dashboard permite realizar operaciones de administración sin necesidad de acceder directamente a la línea de comandos, lo cual simplifica la supervisión y el control del entorno.

Figura 13
Dashboard de OpenStack

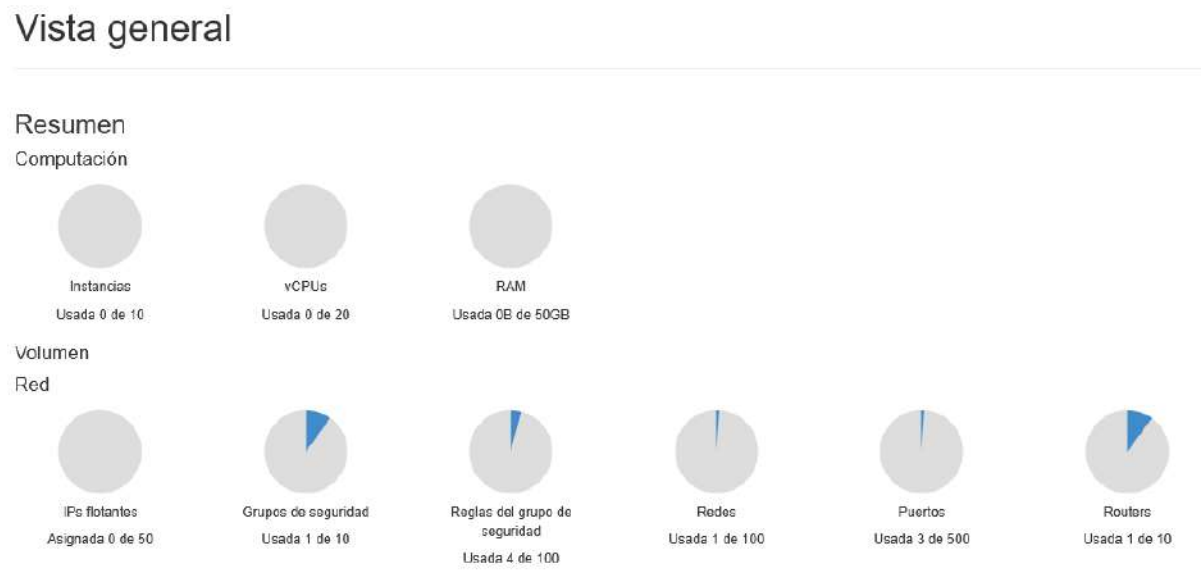


Nota: Este es el inicio y una forma de controlar que la ejecución es un éxito.

Dentro del dashboard se pueden visualizar y administrar proyectos, usuarios y roles, así como crear y gestionar instancias, imágenes, volúmenes de almacenamiento y configuraciones de red. Además, se dispone de información relacionada con el estado de los recursos, el consumo

de CPU, memoria y almacenamiento, y la asignación de direcciones IP, lo que permite tener una visión general del funcionamiento de la infraestructura. De esta manera **¡Error! No se encuentra el origen de la referencia.**, el dashboard se convierte en una herramienta fundamental para la administración operativa de OpenStack, proporcionando un entorno intuitivo y organizado para la gestión de la plataforma en producción. Donde se observa los recursos disponibles que tenemos.

Figura 14
Recursos disponibles vistos desde el Dashboard



Nota: La imagen nos muestra la cantidad de recursos que puede almacenar OpenStack, con la instalación de DevStack.

3.4.6. Configuración de servicios adicionales

En esta etapa del proyecto, se realizó la configuración de servicios adicionales dentro del entorno OpenStack, enfocados principalmente en el despliegue y dimensionamiento de instancias virtuales. Esta tarea fue esencial para simular un entorno realista de infraestructura como servicio (IaaS), permitiendo validar la funcionalidad del sistema, así como ejecutar pruebas de rendimiento, monitoreo y calidad de servicio (QoS). La selección adecuada de imágenes, recursos y sabores (flavors) permitió optimizar el uso del hardware disponible, garantizando estabilidad y eficiencia durante el proceso de pruebas.

3.4.6.1. Dimensionamiento de las instancias

Una vez configurado el entorno base de OpenStack mediante DevStack, se procedió al despliegue de las instancias necesarias para las pruebas. Estas instancias fueron dimensionadas cuidadosamente para garantizar un uso eficiente de los recursos, manteniendo la estabilidad del entorno y permitiendo prácticas de optimización, QoS y monitoreo.

En el apartado de computación **¡Error! No se encuentra el origen de la referencia.**, buscamos e l apartado instancia y al final lanzar instancia.

Figura 15

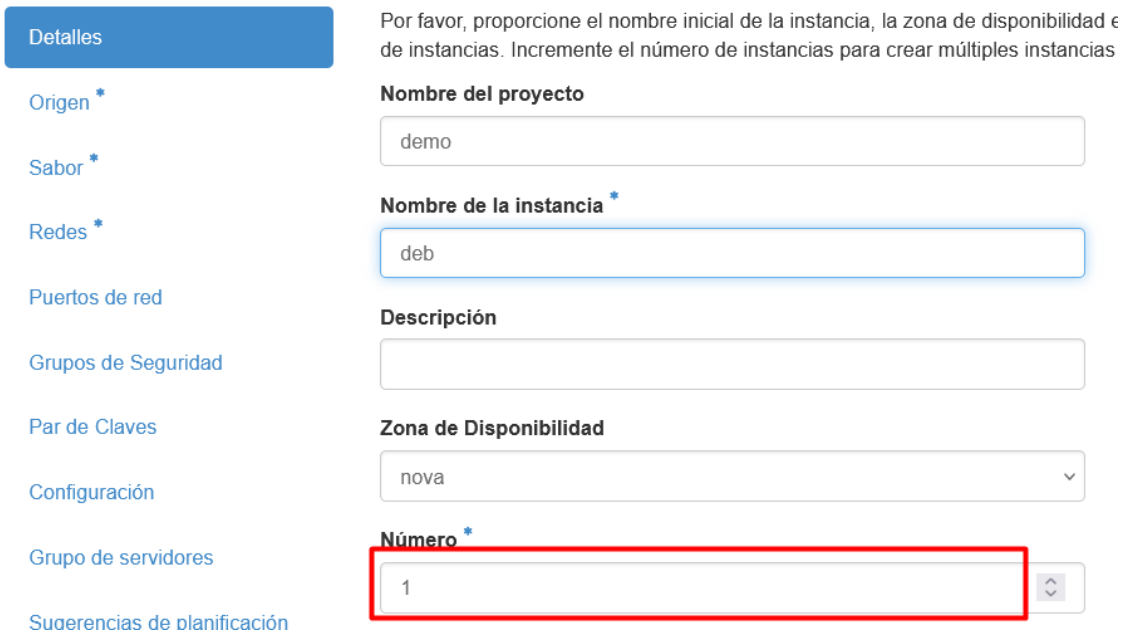
Dimensionamiento de una instancia, lanzamiento

Nota: Para poder crear una nueva instancia, nos dirigimos al botón Lanzar Instancia.

Se desplegará lo siguiente **¡Error! No se encuentra el origen de la referencia.**, aquí vamos a p oner el nombre de la instancia y también nos da la opción de multiplicar la instancia una vez creada

Figura 16

Dimensionamiento de una instancia, Detalles.

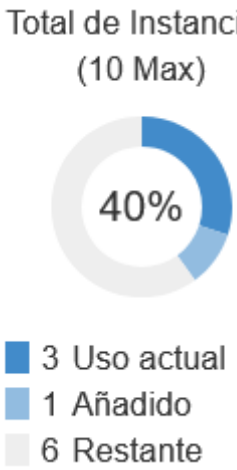


Nota: Aquí pondremos cualquier nombre a la instancia.

Importante, debemos escoger con cuidado cada elección para poder crear nuestro servidor **¡Error! No se encuentra el origen de la referencia.**, no olvidemos que en la parte derecha t enemos un indicador de cuantas instancias podemos crear.

Figura 17

Capacidad de OpenStack para instancias



Nota: La capacidad máxima que tenemos es crear 10 instancias.

- **Cargar una nueva imagen**

Para la sección de instancias, OpenStack por defecto viene solo cargado con una imagen liviana que es Cirros, pero para este caso se va a utilizar imágenes robustas para realizar las pruebas

ya que Cirros es muy limitado. Todo esto se hace mediante terminal **¡Error! No se encuentra e l origen de la referencia.**. Se debe crear una carpeta con el siguiente comando, dentro del usuario stack.

- *mkdir -p ~/images && cd ~/images*

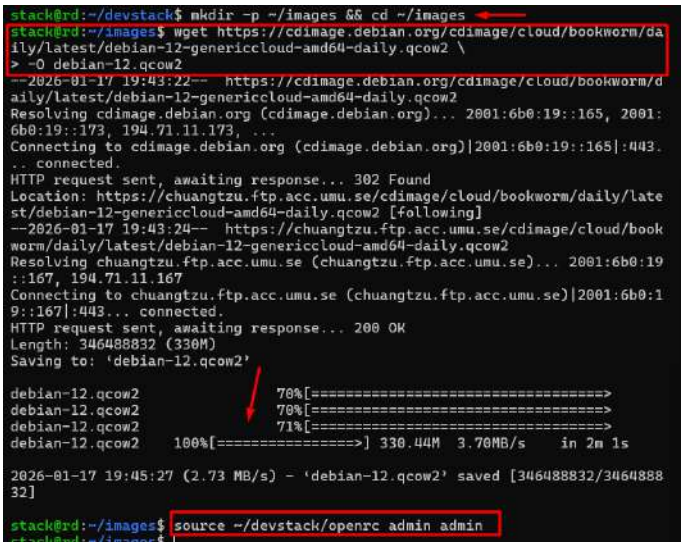
Siempre que vayamos a modificar algo, tenemos que tener permiso, para eso utilizaremos

- *source ~/devstack/openrc admin admin*

Déspues debemos optener el comando que tendrá el link de descarga de la imagen, en este caso será debian

- *wget https://cdimage.debian.org/cdimage/cloud/bookworm/daily/latest/debian-12-genericcloud-amd64-daily.qcow2 \*
- *-O debian-12.qcow2*

Figura 18
Configuraciones para cargar una imagen nueva



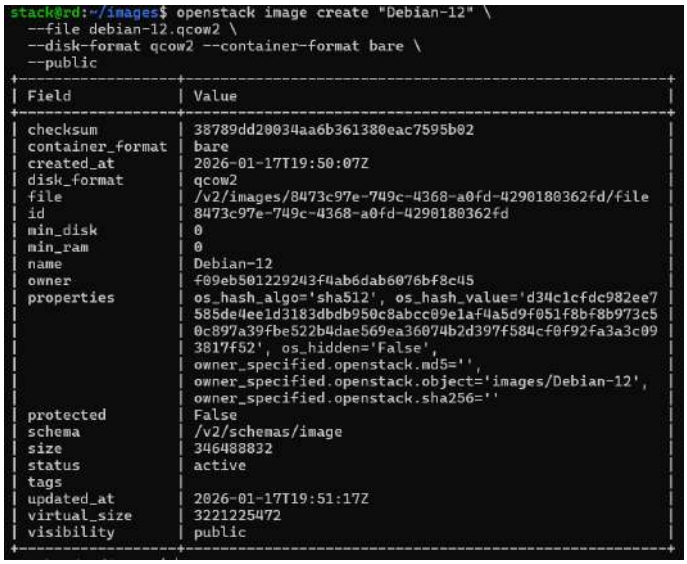
Nota: No olvidemos que a ser DevStack, no se va a poder cargar nuevas imágenes desde el dashboard, se lo hará mediante terminal

Una vez copiado la imagen del repositorio, cargaremos al Glance de OpenStack, en esta parte ya podremos observar la imagen cargada en el dashboard.

- *openstack image create "Debian-12" *

- *--file debian-12.qcow2 *
- *--disk-format qcow2 --container-format bare *
- *--public*

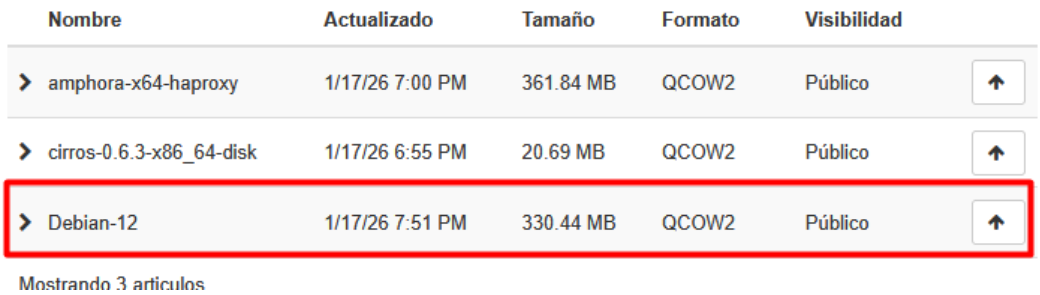
Figura 19
Comando para cargar la imagen al dashboard



Nota: Se debe cargar esta imagen al dashboard, para poder seguir configurando las instancias.

Y también podemos verificar desde el dashboard **¡Error! No se encuentra el origen de la referencia.**.

Figura 20
Comprobación de la imagen cargada correctamente



Nota: Para tener mayor flujo de trabajo siempre es necesario recargar las páginas.

Para ver cargado desde consola, se utiliza el siguiente comando, que nos dará la lista de las imágenes cargadas y el estado en que están:

- *openstack image list*

Figura 21

Comprobación de la imagen cargada correctamente terminal

```
stack@rd:~/images$ openstack image list
+--------------------------------------+---------------------------+--------+
| ID                                   | Name                      | Status |
+--------------------------------------+---------------------------+--------+
| 8473c97e-749c-4368-a0fd-4290180362fd | Debian-12                 | active |
| 1dc160f5-733f-46da-980c-4c19e9596ac1 | amphora-x64-haproxy       | active |
| 4e28be31-f4bc-4cc0-9be7-306047e3d14d | cirros-0.6.3-x86_64-disk  | active |
+--------------------------------------+---------------------------+--------+
```

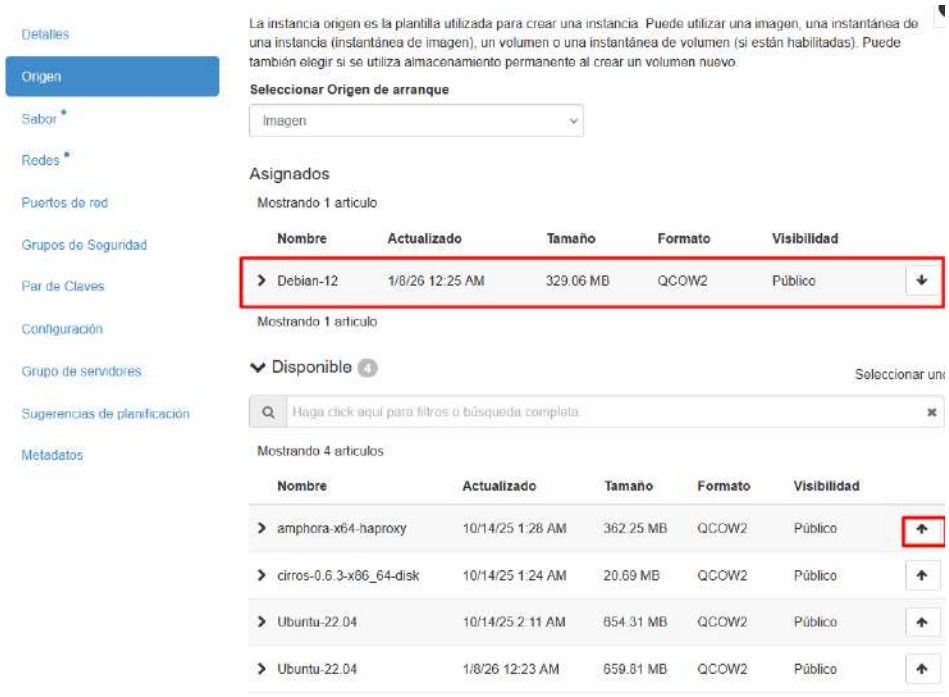
Nota: Lista de imágenes cargadas en terminal.

- **Configuración de la instancia**

Por medio del dashboard es muy sencillo crear una instancia ya que también dispone de comentarios de que debemos escoger, como por ejemplo en la parte de origen vamos a escoger la imagen que queremos que la instancia sea **¡Error! No se encuentra el origen de la r eferencia.**, en este caso se escogió Debian.

Figura 22

Dimensionamiento de una instancia, Origen.



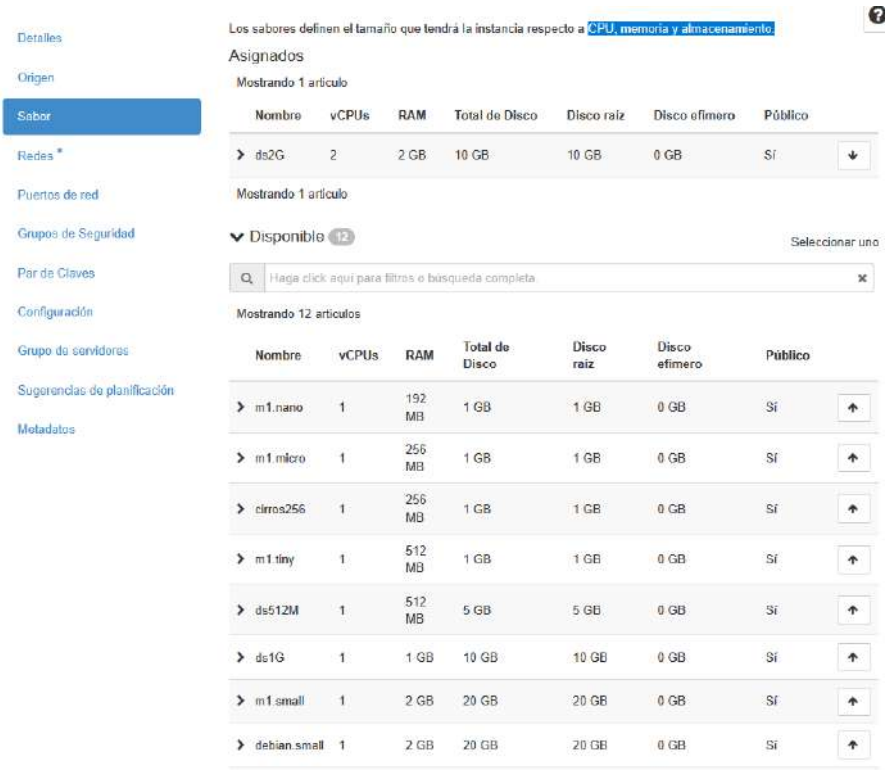
Nota: En este apartado se debe escoger la imagen de formato qcow2 para la instancia.

En el apartado de sabor es muy importante ya que es donde debemos escoger el tamaño de disco virtual, esto quiere decir que se dimensionara los siguiente CPU, memoria y almacenamiento. Cabe recalcar que no todo es compatible, y todo depende de la imagen y del servidor a lanzar. Por ejemplo, existe sabores muy pequeños que solo son para pruebas y solo

para la imagen Cirros, la cual es automáticamente cargada al instalar openstack. Esteo es muy visible en cada sabor.

En la siguiente imagen vemos como se dimensionó para debian **¡Error! No se encuentra el o rigen de la referencia.**, mencionar que los sabores pequeños no permitirán que esta se despliegue.

Figura 23
Dimensionamiento de una instancia, Sabor.

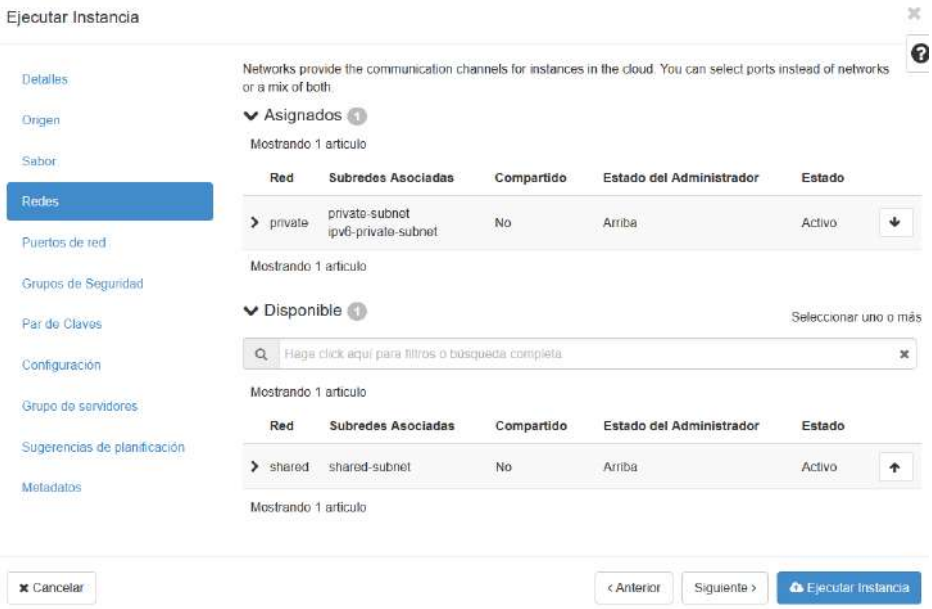


Nota: Aquí se escoge el tamaño de la instancia, en cuanto a procesador y memoria.

En el apartado redes **¡Error! No se encuentra el origen de la referencia.**, solo debemos e scoger a cuál Red pertenecerá:

Figura 24

Dimensionamiento de una instancia, Redes.



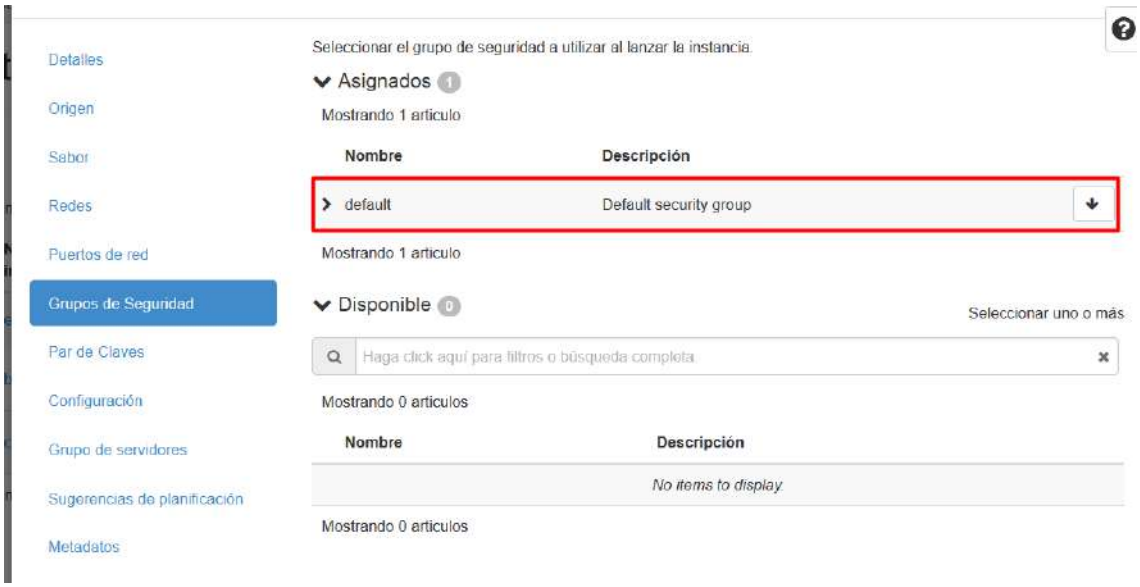
Nota: No olvidar que todas las redes deben estar conectadas a el router virtual para tener salida a internet.

En los apartados Puerto de red, Grupo de servidores, Sugerencia y planificaciones y Metadatos. No vamos a configurar nada ya que no aplica para el proyecto, de igual forma no es difícil, ya que solo debemos escoger opciones.

En el apartado Grupos de Seguridad **¡Error! No se encuentra el origen de la referencia.**, d ebemos escoger aquel grupo al que configuramos previamente, ya que en este podremos dar acceso o no a cada servidor o restringir ciertos protocolos como icmp o ssh.

Figura 25

Dimensionamiento de una instancia, Grupos de Seguridad.

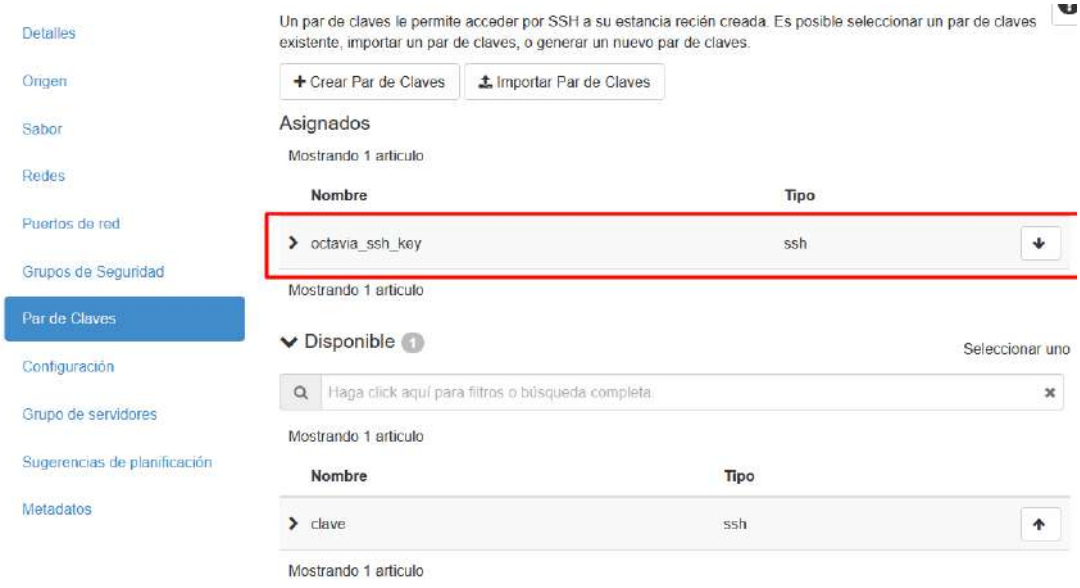


Nota: Es recomendable para cada tipo de servidores crear diferentes grupos de seguridad por sus diferentes protocolos que utilizan.

En el apartado par de claves **¡Error! No se encuentra el origen de la referencia.**, podemos e scoger claves de conexión, es este caso tenemos ssh.

Figura 26

Dimensionamiento de una instancia, Par de Claves.



Nota: Este es muy importante para tener conexión vía terminal entre la instancia y la maquina física.

Uno de los apartados más importantes es el de Configuración ya que aquí debemos poner un código que sea compatible con la instancia y que nos permita poner el usuario de la instancia y la contraseña para poder dar acceso al super usuario **¡Error! No se encuentra el origen de l**

a referencia.. No olvidemos que al momento de escoger la imagen de tipo QCOW2, esta no vine libre y debemos poner estos comandos para poder acceder a la terminal:

Código 1

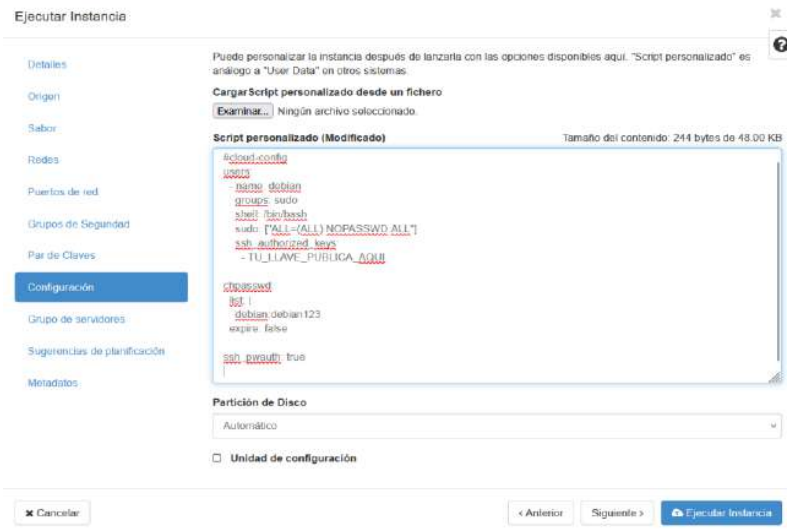
Código que va en el apartado Configuración

```
#cloud-config

% users:
      - name: debian
% groups: sudo
% shell: /bin/bash
% sudo: ["ALL=(ALL) NOPASSWD:ALL"]
% ssh_authorized_keys:
      - TU_LLAVE_PUBLICA_AQUI
% chpasswd:
% list: |
% debian:debian123
% expire: false


% ssh_pwauth: true
```

Nota: Este código se personaliza para cada imagen que se ocupa, también existe imágenes que no necesitan de este, como Cirros.

Figura 27

Dimensionamiento de una instancia, Configuración.



Nota: Código personalizado para cada instancia

De esta forma se hace cada instancia, claro que cada una cambia debido al servicio que instalemos en ella, cambiaria el tipo de imagen, el sabor y el tamaño de disco. Entonces se obtuvo las siguientes instancias **¡Error! No se encuentra el origen de la referencia.**, que r epresentaran los servicios y clientes.

Tabla 15

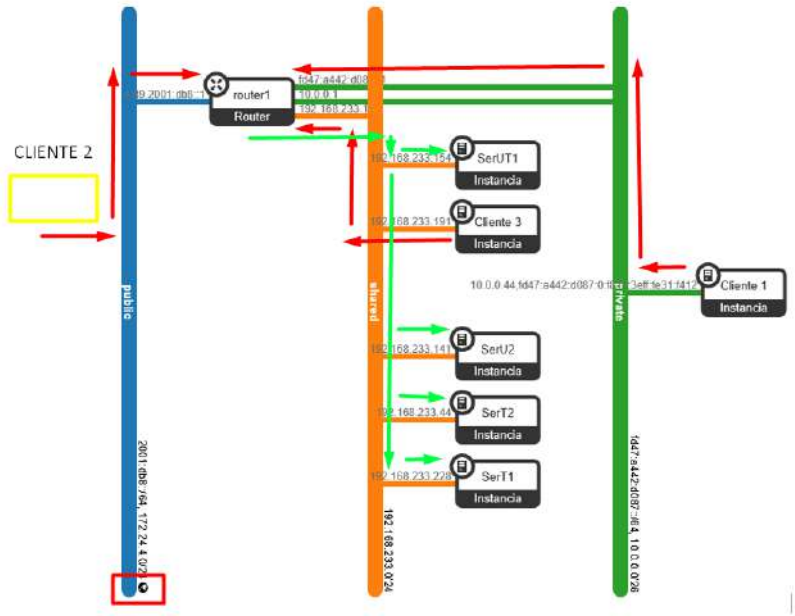
Instancias desplegadas, sabores e imagen utilizada

Instancia	Rol dentro de la infraestructura	Imagen utilizada	vCPU	Memoria RAM	Almacenamiento	Descripción funcional
Web	HTTP y TCP	CirrOS	1	1 GB	5 GB	Proporciona servicios web accesibles mediante protocolo HTTP, utilizada para pruebas de conectividad y análisis de tráfico TCP
TCP	HTTP y TCP	CirrOS	1	512 MB	5 GB	
UDP 1	Servidor de servicios UDP	Debian	1	512 MB	5 GB	Primera y segunda instancia dedicada al manejo de tráfico UDP, utilizada para pruebas de comunicación sin conexión, herramienta netcat
UDP 2	Servidor de servicios UDP	Debian	1	512 MB	5 GB	
Cliente 1	Generador de tráfico	CirrOS	1	256 MB	1 GB	Utilizada para generar solicitudes y tráfico hacia los servidores, empleando una imagen liviana para minimizar consumo de recursos

Nota: Las imágenes y sabores asignados a cada instancia se seleccionaron considerando el rol que desempeñan dentro de la infraestructura, priorizando el uso de imágenes livianas en los clientes para reducir el consumo de recursos y concentrar la capacidad del sistema en los servidores de servicio.

Figura 28

Topología implementada y funcional



Nota: Se observa la topología utilizada, adicional con las flechas rojas se indica el camino de las solicitudes que siempre se dirigirán la router 1 desde los clientes y con la flecha verde indica que estas peticiones salen desde el router 1 hacia los servidores.

La infraestructura OpenStack implementada se organiza en instancias con roles claramente definidos de cliente y servidor, con el objetivo de estructurar adecuadamente la generación y

recepción de tráfico dentro del entorno como de evidencia en la **¡Error! No se encuentra el o rigen de la referencia.**. Las instancias servidor se encargan de proveer servicios específicos, como servicios web y servicios de transporte basados en los protocolos TCP y UDP, mientras que las instancias cliente se utilizan exclusivamente para la generación de solicitudes y tráfico hacia dichos servidores, permitiendo evaluar el comportamiento de la infraestructura desde el punto de vista del consumo y la respuesta del sistema.

La utilización de imágenes livianas en las instancias cliente permite minimizar el consumo de recursos del nodo OpenStack, asegurando que la mayor parte de la capacidad de procesamiento, memoria y almacenamiento se destine a las instancias servidor. De esta manera, se garantiza que las mediciones y el análisis posterior se centren en el desempeño de los servicios y no en procesos innecesarios del sistema operativo de las instancias cliente.

Tabla 16

Consumo total de recursos según el tipo de instancia

Tipo de instancia	Cantidad	vCPU totales	Memoria RAM total	Almacenamiento total
Servidor Web	1	1 vCPU	1 GB	10 GB
Servidores TCP	1	1 vCPU	512 MB	5 GB
Servidores UDP	2	2 vCPU	1 GB	10 GB
Clientes	2	2 vCPU	512 MB	2 GB
Total asignado	**6**	**6 vCPU**	**3 GB**	**27 GB**

Nota: El consumo total de recursos refleja la distribución de instancias desplegadas dentro del entorno OpenStack, permitiendo visualizar el uso global de CPU, memoria y almacenamiento en función del rol de cada instancia.

3.4.6.2. Configuración de red interna con red externa

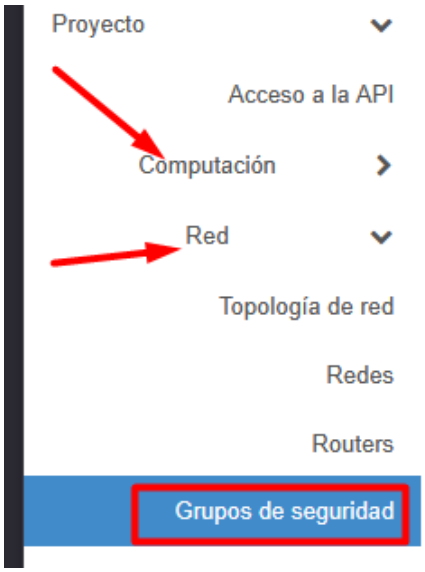
Para poder tener comunicación externa entre una instancia y cualquier maquina fuera de OpenStack, se debe tener en cuenta con dos observaciones:

Tener una ip flotante, esta ayuda a la instancia a comunicarse al exterior, Neutron tiene una red destinada para esto que está en el Rango 172.24.4.0/24, por lo tanto, debemos crear una ruta desde el dispositivo exterior asegurando que tengamos ping a la IP Estatica de OpenStack, estos son los comandos utilizados:

1. *Para Windows: ROUTE ADD 172.24.4.0 MASK 255.255.255.0 192.168.100.100*
2. *Para Linux: sudo ip route add 172.24.4.0/24 via 192.168.100.100*

La otra observación es tener habilitado todos los puertos que necesitemos utilizar, para la conexión o respuesta se debe tener habilitado los protocolo SSH y ICMP, respectivamente con sus puertos, en OpenStack existe la facilidad de habilitar los protocolos mediante el Dashboard **¡Error! No se encuentra el origen de la referencia.**, en el apartado Grupo de Seguridad.

Figura 29
Ubicación del apartado Grupos de seguridad

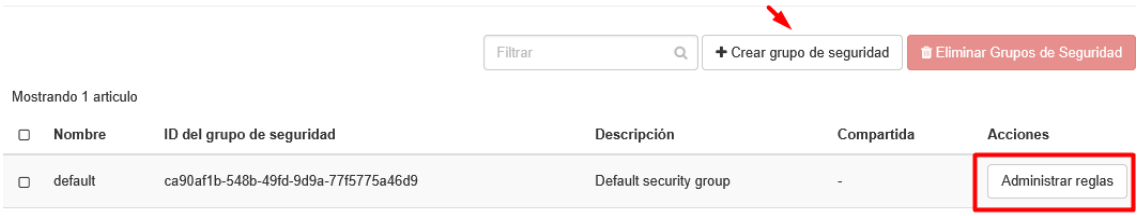


Nota: En este apartado se habilitarán o no protocolos, para configuraciones futuras

Dentro de este apartado se puede crear diferentes grupos de seguridad para cada necesidad o simplemente aumentar reglas a un grupo ya existente que es lo que está hecho **¡Error! No se encuentra el origen de la referencia.**:

Figura 30

Creación de grupo de seguridad



Nota: De preferencia utilizar el grupo ya definido, para ahorrar recursos.

En administrar reglas, podemos ver las reglas existentes, pero como vemos no tenemos habilitado para la conexión SSH y es lo que se realizara, nos dirigimos a Agregar regla **¡Error! No se encuentra el origen de la referencia.**:

Figura 31

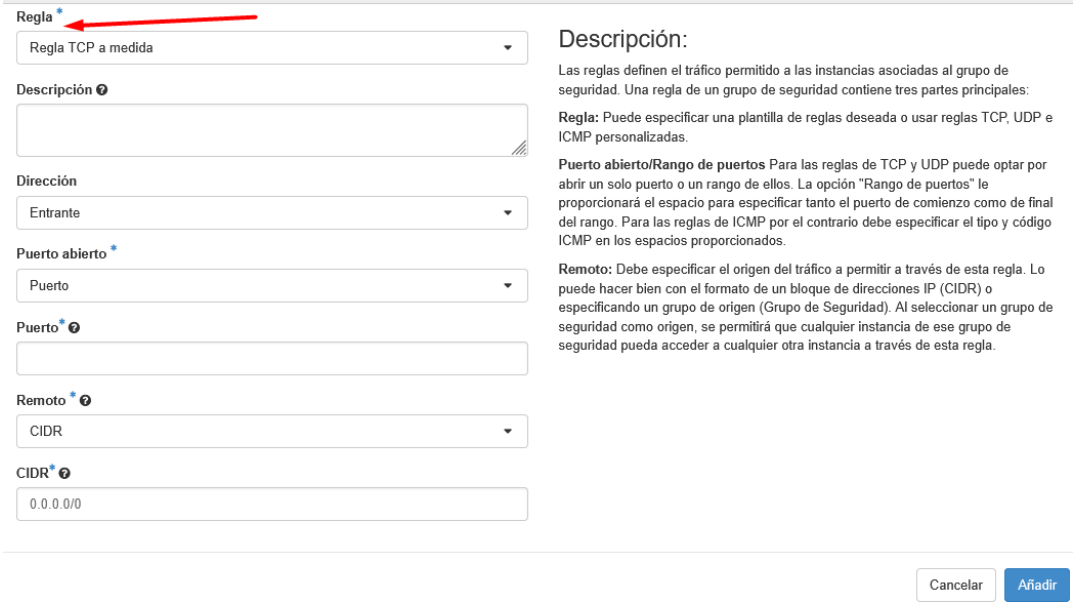
Creación de nueva regla

Nota: Antes de agregar algo, leer las reglas que ya tenemos, para evitar conflictos.

Dentro de esto vamos al apartado que dice Regla y se desplegara el protocolo que se habilitara **¡Error! No se encuentra el origen de la referencia.**.

Figura 32

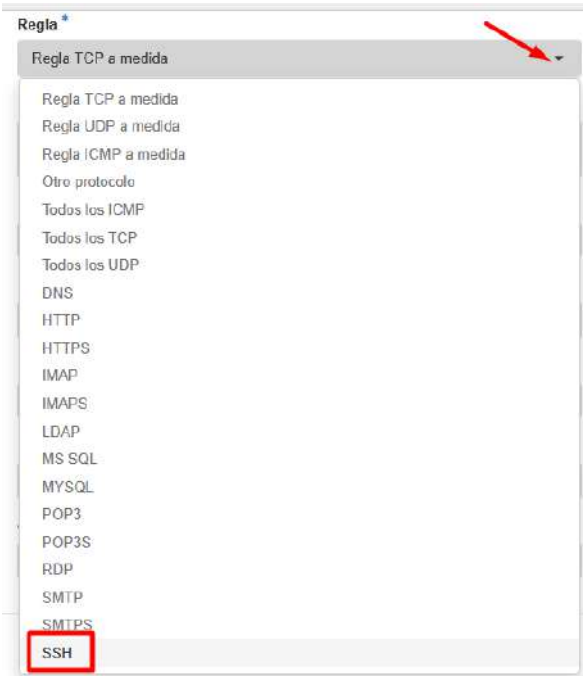
Configuración interna de cada regla basada en un protocolo



Nota: En la parte derecha se encuentra una leyenda, que especifica lo que se necesita.

Se desplegará un listado de los diferentes protocolos que podemos dar o no permiso, como se ve, se encuentra la mayoría. En esta ocasión se selecciona SSH **¡Error! No se encuentra el origen de la referencia.**.

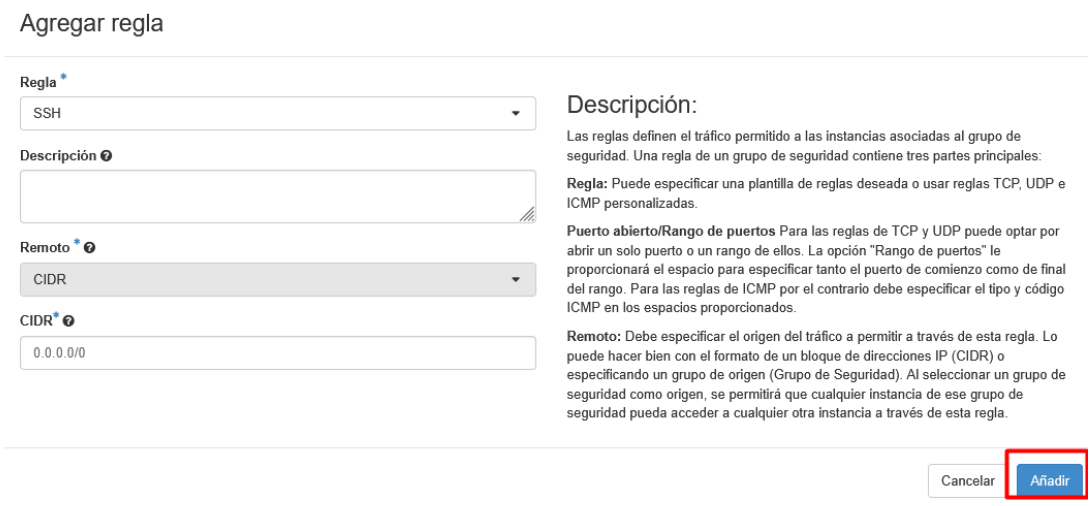
Figura 33
Protocolos disponibles



Nota: Esta lista es útil para después poder configurar lo demás.

Automáticamente, se muestra en los campos que se ira a aplicar **¡Error! No se encuentra el origen de la referencia.**.

Figura 34
Configuración de la regla de acuerdo al protocolo



Nota: Automáticamente se llenan los campos, en ocasiones como utilización de otros puertos, estos se llenarán manualmente.

Verificamos la creación de la regla en él listado y comprobamos en las respectivas salidas **Figura *35***.

Figura 35

Aparición de la nueva regla

Mostrando 9 articulos

	Dirección	Tipo Ethernet	Protocolo IP	Rango de puertos	Prefijo de IP Remota	Grupo de Seguridad Remoto	Description	Acciones
☐	Saliente	IPv4	Cualquier	Cualquier	0.0.0.0/0	-	-	Eliminar Regla
☐	Saliente	IPv4	ICMP	Cualquier	0.0.0.0/0	-	-	Eliminar Regla
☐	Saliente	IPv4	TCP	8080	0.0.0.0/0	-	-	Eliminar Regla
☐	Saliente	IPv6	Cualquier	Cualquier	::/0	-	-	Eliminar Regla
☐	Entrante	IPv4	Cualquier	Cualquier	-	default	-	Eliminar Regla
☐	Entrante	IPv4	ICMP	Cualquier	0.0.0.0/0	-	-	Eliminar Regla
☐	Entrante	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Eliminar Regla
☐	Entrante	IPv4	TCP	8080	0.0.0.0/0	-	-	Eliminar Regla
☐	Entrante	IPv6	Cualquier	Cualquier	-	default	-	Eliminar Regla

Nota: En el listado siempre debe reflejar la nueva regla, a veces este proceso se puede demorar un poco.

3.4.6.3. Dimensionamiento de Servicios

- **Servicios con tráfico TCP**

Para poder realizar pruebas a nivel de producción, se escogieron métodos de generar tráfico que se asemeje con servicios reales, dependiendo del tipo de resultados que se encontraran a futuro, se configuro dos servidores Web y en estos mismos servidores se utilizó la herramienta netcat, que genera tráfico TCP en respuestas verdaderas para después ser analizados, de esta forma este es el dimensionamiento de los servidores TCP:

- **WEB**

Se utilizo dos imágenes Cirros, al ser liviana y permite demostrar fácilmente, solo creamos una carapeta, en este caso www y un archivo index.html, que contendrá el contenido de la página web, personada para cada servidor y diferenciar.

1. sudo sh
2. mkdir -p /var/www
3. vi /var/www/index.html

Figura 36

Configuración de los servidores WEB

Nota: Se muestra la ejecución de los códigos dentro de cirros, este proceso se hace en cada servidor.

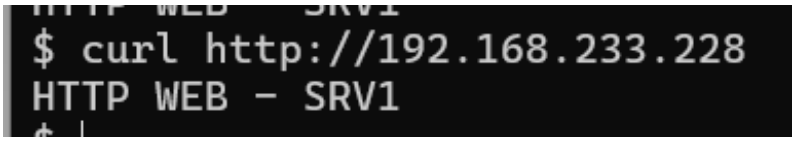
Al final ponemos la siguiente línea para poner los servidores en modo escucha, aquí se especifica el puerto, como en este caso el 80.

- busybox httpd -f -p 80 -h /var/www &

Para poder comprobar, existen dos métodos desde el cliente, el primero es acceder desde el navegador y poner http:// "IP servidor" **¡Error! No se encuentra el origen de la referencia.** o desde terminal hacer un curl **¡Error! No se encuentra el origen de la referencia.** y esperar respuesta del servidor.

Figura 37

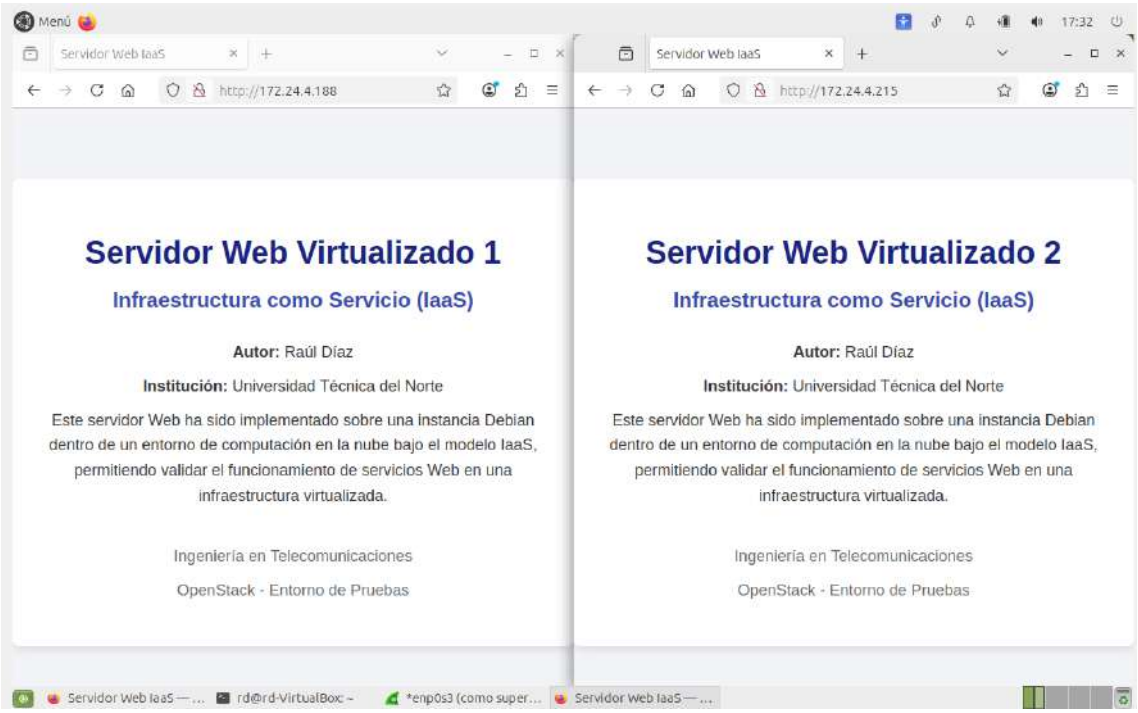
Respuesta desde el servidor por medio de terminal



Nota: Esta es la respuesta del servidor Web aplicado un curl desde un terminal.

Figura 38

Respuesta desde el servidor por medio de un navegador



Nota: En un navegador ingresamos vía web al servidor

- **TCP utilizando netcat**

Se utiliza las mismas instancias que los servidores WEB, ya que cirros ya tiene preinstalado la herramienta netcat, para esto es muy sencillo el cada servidor se pone los comandos:

- *sudo sh*
- *while true; do echo "RESPUESTA SERVIDOR TCP" | nc -l -p 5000; done &*

Este comando pone al servidor el modo escucha y genera trafico TCP, el cliente es el que solicita este tráfico, de la siguiente manera:

- *while true; do echo "TRAFICO TCP QoS" | nc 172.24.4.188 5000; sleep 0.5; done*

De esta manera ya solo filtramos tráfico, mediante TCPDUMP o Wireshark para después poder analizarlo.

No olvidar que, al utilizar un puerto, se tiene que crear la regla para este **¡Error! No se e ncuentra el origen de la referencia.**, para ello se crea una regla para tráfico TCP en puerto 5000.

Figura 39

Creación de la regla TCP puerto 5000

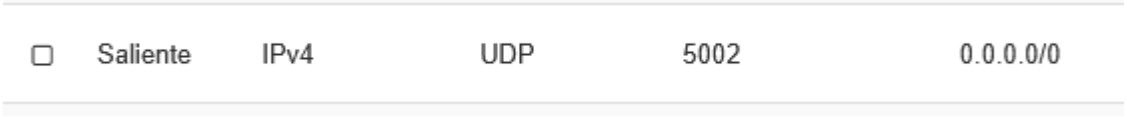
Nota: Para permitir tanto la salida como la entrada de tráfico TCP por el puerto 5000, debemos crear una regla en el grupo de seguridad.

- **UDP utilizando netcat**

El primer paso es habilitar el puerto tanto de entrada y salida para el protocolo UDP **¡Error! No se encuentra el origen de la referencia.**, en este caso por el puerto 5002.

Figura 40

Creación de la regla UDP puerto 5002



Nota: Para permitir tanto la salida como la entrada de tráfico UDP por el puerto 5002, debemos crear una regla en el grupo de seguridad.

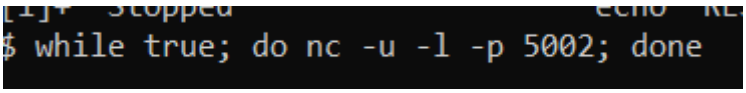
En nuestro servidor utilizaremos netcat, para que tráfico se asemeje a un entorno real.

En el servidor se utilizará se envía carga UDP continua con tamaño fijo **¡Error! No se encuentra el origen de la referencia.**.

- *while true; do nc -u -l -p 5002; done*

Figura 41

Aplicación de tráfico UDP servidor

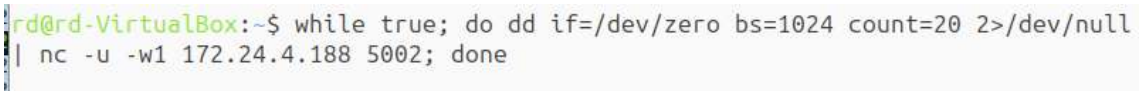


Nota: Se utiliza la herramienta netcat que viene preinstalada en Cirros, para generar tráfico UDP.

Mientras que, para el cliente, por medio de este se establece una solicitud de paquetes UDP para el servidor **¡Error! No se encuentra el origen de la referencia.**.

- *while true; do dd if=/dev/zero bs=1024 count=20 2>/dev/null | nc -u -w1 172.24.4.188 5002; done*

Figura 42
Solicitud de tráfico UDP



Nota: El cliente solicita tráfico UDP, este comando asegura solicitudes largas de UDP, para poder apreciar en la captura de paquetes.

3.5. Estrategias de Optimización del Rendimiento

La optimización del rendimiento en infraestructuras como servicio constituye una etapa clave dentro del desarrollo de este proyecto, ya que permite evaluar cómo el uso adecuado de mecanismos de control y gestión impacta directamente en la eficiencia de los recursos disponibles, en este contexto se aplicaron estrategias orientadas al control del tráfico de red mediante calidad de servicio, al reparto equilibrado de solicitudes a través del balanceo de carga y a la combinación de ambas técnicas para simular escenarios reales utilizados en entornos productivos.

Aplicación de Calidad de Servicio en el entorno IaaS

La calidad de servicio fue aplicada con el objetivo de priorizar determinados flujos de tráfico y limitar el consumo de ancho de banda de instancias específicas, esta técnica resulta especialmente útil cuando existen servicios críticos que requieren mayor estabilidad frente a otros que generan tráfico de menor prioridad.

Aplicación de balanceo de carga en el entorno virtualizado

El balanceo de carga fue utilizado como una estrategia para distribuir el tráfico de red entre múltiples instancias que ofrecen el mismo servicio, evitando la sobrecarga de un único servidor y mejorando la disponibilidad general del sistema.

En el entorno de pruebas se implementó un escenario donde dos instancias actuaban como servidores y una tercera instancia funcionaba como cliente generador de solicitudes, al aplicar balanceo de carga se pudo observar que las peticiones eran atendidas de forma alternada por ambos servidores, reduciendo el consumo excesivo de recursos en una sola máquina, este ejemplo permitió demostrar cómo el balanceo de carga contribuye a una utilización más eficiente del cómputo y mejora la experiencia del usuario final.

Aplicación conjunta de Calidad de Servicio y balanceo de carga

La combinación de calidad de servicio y balanceo de carga permitió evaluar un escenario más avanzado de optimización del rendimiento, donde no solo se distribuyen las solicitudes entre múltiples servidores, sino que además se prioriza el tráfico según su importancia

En este caso se configuró un entorno donde el tráfico hacia el balanceador tenía políticas de calidad de servicio asociadas, garantizando que las solicitudes críticas mantuvieran prioridad incluso cuando el número de conexiones aumentaba, al mismo tiempo el balanceo de carga se encargó de repartir dichas solicitudes entre los servidores disponibles, logrando un comportamiento más estable y predecible del sistema

Este escenario combinado permitió evidenciar cómo ambas técnicas se complementan y representan una solución más cercana a las implementaciones reales utilizadas en infraestructuras empresariales.

3.5.1. Definición de escenarios de prueba para la optimización del rendimiento

Con el fin de evaluar el impacto real de las estrategias de optimización aplicadas sobre la infraestructura como servicio implementada, se definieron distintos escenarios de prueba que permitieron analizar el comportamiento del sistema bajo condiciones controladas, estos escenarios fueron diseñados para simular situaciones comunes en entornos de red virtualizados y facilitar la comparación de resultados. Todo esto se aplicó a la topología trabajada **¡Error! No se encuentra el origen de la referencia.**.

La definición de escenarios diferenciados permite analizar de manera progresiva cómo cada técnica de optimización influye en el rendimiento del sistema, comenzando por configuraciones simples y avanzando hacia escenarios más complejos que integran múltiples mecanismos de control.

Para este proyecto se establecieron tres escenarios principales de prueba, el primero enfocado únicamente en la aplicación de calidad de servicio, el segundo centrado en el uso de balanceo de carga y el tercero orientado a la aplicación conjunta de ambas técnicas, estos escenarios fueron ejecutados bajo condiciones similares para permitir una comparación objetiva de los resultados obtenidos.

3.5.1.1. Escenario de prueba con aplicación exclusiva de Calidad de Servicio (QoS)

En este primer escenario se aplicaron políticas de calidad de servicio con el objetivo de priorizar el tráfico generado. Dentro del entorno OpenStack se configuraron límites de ancho de banda, al ejecutar solicitudes concurrentes se pudo observar que las instancias priorizadas mantenían tiempos de respuesta más estables incluso cuando el tráfico aumentaba.

Este escenario permitió evidenciar cómo la calidad de servicio contribuye a controlar el uso de la red y a garantizar un nivel mínimo de rendimiento.

3.5.1.2. Escenario de prueba con aplicación exclusiva de balanceo de carga

El segundo escenario se diseñó para analizar el comportamiento del sistema cuando se aplica balanceo de carga sin políticas de calidad de servicio, en este caso el objetivo principal fue distribuir las solicitudes de red entre múltiples instancias que ofrecen el mismo servicio.

Para este escenario se desplegaron dos instancias servidoras que respondían a las solicitudes de una o varias instancias cliente.

3.5.1.3. Escenario de prueba con aplicación conjunta de Calidad de Servicio y balanceo de carga

El tercer escenario integró tanto políticas de calidad de servicio como mecanismos de balanceo de carga, representando un entorno más cercano a una implementación real de infraestructura como servicio, este escenario permitió evaluar el comportamiento del sistema bajo condiciones de mayor complejidad.

En este caso se aplicaron políticas de QoS al tráfico que ingresaba al balanceador, garantizando prioridad a ciertos flujos, mientras que el balanceo de carga se encargó de distribuir dichas solicitudes entre los servidores disponibles, los resultados mostraron un comportamiento más estable, con menor congestión y mejor aprovechamiento de los recursos.

3.5.1.4. Relación de los escenarios con las herramientas de análisis

Los tres escenarios definidos fueron analizados mediante herramientas de monitoreo y captura de tráfico, lo que permitió validar el impacto de cada estrategia sobre el rendimiento del sistema, estas herramientas se detallan en los apartados siguientes y constituyen un apoyo fundamental para la evaluación objetiva de los resultados.

3.5.2. Herramientas de medición y validación del rendimiento

Para evaluar de manera objetiva el impacto de las estrategias de optimización aplicadas en los diferentes escenarios de prueba, fue necesario utilizar herramientas de medición y análisis que permitieran observar el comportamiento del sistema tanto a nivel de tráfico de red como de consumo de recursos, estas herramientas constituyen un elemento clave para validar los resultados obtenidos y respaldar el análisis técnico del proyecto.

Las herramientas empleadas fueron TCPDUMP, HTOP y Wireshark, cada una aplicada de forma específica según el escenario de prueba y el tipo de información que se requería analizar.

3.5.1.1. TCPDUMP

Se utiliza como herramienta principal para la captura y análisis del tráfico de red generado por las instancias virtuales, su aplicación permitió observar los paquetes que circulaban entre clientes y servidores, así como verificar el correcto funcionamiento de las configuraciones de red implementadas en OpenStack

En el entorno de pruebas se utilizó TCPDUMP para capturar tráfico TCP y UDP generado durante las simulaciones de calidad de servicio y balanceo de carga, por ejemplo, al ejecutar solicitudes continuas desde una instancia cliente hacia servidores web se pudo identificar el flujo de paquetes, los puertos utilizados y la frecuencia de transmisión, lo cual permitió comprobar si las políticas de priorización estaban siendo aplicadas correctamente

3.5.1.2. HTOP

Se empleada como herramienta de monitoreo para analizar el consumo de CPU y memoria del nodo que aloja OpenStack, su uso permitió visualizar en tiempo real el impacto que tienen las instancias virtuales y los servicios del sistema durante la ejecución de pruebas de carga.

Durante las pruebas se observó el aumento del consumo de recursos al generar tráfico simultáneo desde múltiples instancias, especialmente cuando se aplicaban escenarios de balanceo de carga, esta observación permitió identificar qué procesos de OpenStack demandaban mayor capacidad de cómputo y ajustar la asignación de recursos para evitar la saturación del sistema.

3.5.1.3. WIRESHARK

Se utiliza como una herramienta complementaria a TCPDUMP para realizar un análisis más detallado del tráfico de red capturado durante las pruebas, su interfaz gráfica permitió examinar de forma visual los paquetes, facilitando la interpretación de los resultados obtenidos.

En el entorno implementado se empleó Wireshark para analizar flujos de tráfico TCP y UDP asociados a servicios priorizados mediante calidad de servicio, permitiendo observar métricas como tiempos de respuesta, retransmisiones y comportamiento de los protocolos bajo diferentes niveles de carga, estos análisis permitieron validar de forma visual el efecto de las políticas de optimización aplicadas.

3.5.2. Parámetros de optimización de la Red y Herramientas.

El análisis del rendimiento de la red dentro de una infraestructura como servicio requiere la evaluación de parámetros que permitan medir de forma objetiva el comportamiento del sistema bajo diferentes condiciones de carga, en este proyecto se seleccionaron parámetros fundamentales que reflejan la eficiencia, estabilidad y capacidad de respuesta de la red virtualizada implementada en OpenStack.

Los parámetros analizados fueron **¡Error! No se encuentra el origen de la referencia.**, los c uales se evaluaron mediante herramientas de monitoreo y captura de tráfico utilizadas a lo largo del desarrollo del proyecto.

Tabla 17
Tabla de parámetros de optimización de la red

Parámetro	Descripción	Importancia en el proyecto
Throughput	Cantidad efectiva de datos transmitidos por unidad de tiempo	Permite evaluar la capacidad real de transmisión de la red
Latencia	Tiempo que tarda un paquete en viajar del origen al destino	Indica el nivel de respuesta de los servicios
Jitter	Variación del retardo entre paquetes consecutivos	Refleja la estabilidad del tráfico especialmente en tiempo real
Escalabilidad	Capacidad del sistema para mantener el rendimiento ante mayor carga	Permite analizar el comportamiento al aumentar usuarios o tráfico

Nota: Aquí se muestras todos los parámetros a evaluar en este proyecto.

Throughput

El throughput fue determinado a partir de la cantidad total de datos transmitidos entre instancias durante un intervalo de tiempo definido, este valor se obtuvo mediante la observación del tráfico capturado con tcpdump y analizado posteriormente en Wireshark.

La expresión utilizada para calcular el throughput es:

- *Throughput = Datos transmitidos / Tiempo*

Donde los datos transmitidos se expresan en bits y el tiempo en segundos.

Por ejemplo, si durante una prueba se transmitieron 10 Mbits en un intervalo de 5 segundos el throughput resultante sería.

Throughput = 10 Mbits / 5 s = 2 Mbps

Este parámetro permitió observar que al aplicar balanceo de carga el valor de throughput se mantenía más estable al distribuir el tráfico entre varios servidores.

Latencia

La latencia se obtuvo midiendo el tiempo transcurrido entre el envío de un paquete desde el cliente y la recepción de la respuesta por parte del servidor, este valor fue observado mediante marcas de tiempo capturadas en Wireshark.

La latencia se expresa como:

- *Latencia = Tiempo de llegada − Tiempo de envío*

Este parámetro permitió evaluar el impacto de la calidad de servicio ya que las instancias priorizadas presentaron tiempos de respuesta menores en comparación con aquellas sin prioridad.

Jitter

El jitter se calculó como la variación del retardo entre paquetes consecutivos, este parámetro fue especialmente relevante para analizar la estabilidad del tráfico bajo escenarios de congestión.

La expresión utilizada para el cálculo del jitter es:

- $Jitter = | Latencia_n - Latencia_{n-1} |$

Donde $Latencia_n$ corresponde al retardo del paquete actual y $Latencia_{n-1}$ al retardo del paquete anterior.

Un valor bajo de jitter indicó un comportamiento más estable de la red, observándose mejoras significativas cuando se aplicaron políticas de calidad de servicio.

Escalabilidad

La escalabilidad no se expresó mediante una fórmula única, sino mediante la observación del comportamiento del sistema al incrementar progresivamente el número de instancias y la carga de tráfico.

Este parámetro se evaluó comparando métricas como throughput y latencia antes y después del aumento de carga, determinando si el sistema era capaz de mantener un rendimiento aceptable.

Desde el punto de vista académico, la escalabilidad permitió analizar cómo la combinación de balanceo de carga y calidad de servicio contribuye a sostener el rendimiento del entorno IaaS ante escenarios de mayor demanda.

3.5.2.1. Aplicación de Calidad de Servicio.

En este apartado vamos a aplicar calidad de servicio directamente al tráfico, significa que no habrá etiquetas, todo esto lo haremos mediante terminal, ya que el apartado en dashboard de

QoS, no esté habilitado completamente debido a que DevStack se basa en un solo nodo **¡Error! No se encuentra el origen de la referencia.**.

- *openstack network qos policy create qos_tcp_limit*

Con este comando se crea una política

- *openstack network qos policy list*

Se comprueba que aparezca en la lista.

Dentro de la política debemos crear una regla, es aquí donde vamos a aplicar directo al tráfico, en este caso, se va a aplicar un limitante al Ancho de Banda, no olvidar que la regla siempre va aplicado al servidor mas no al cliente.

Se aplica un limitante de 1mbps de salida.

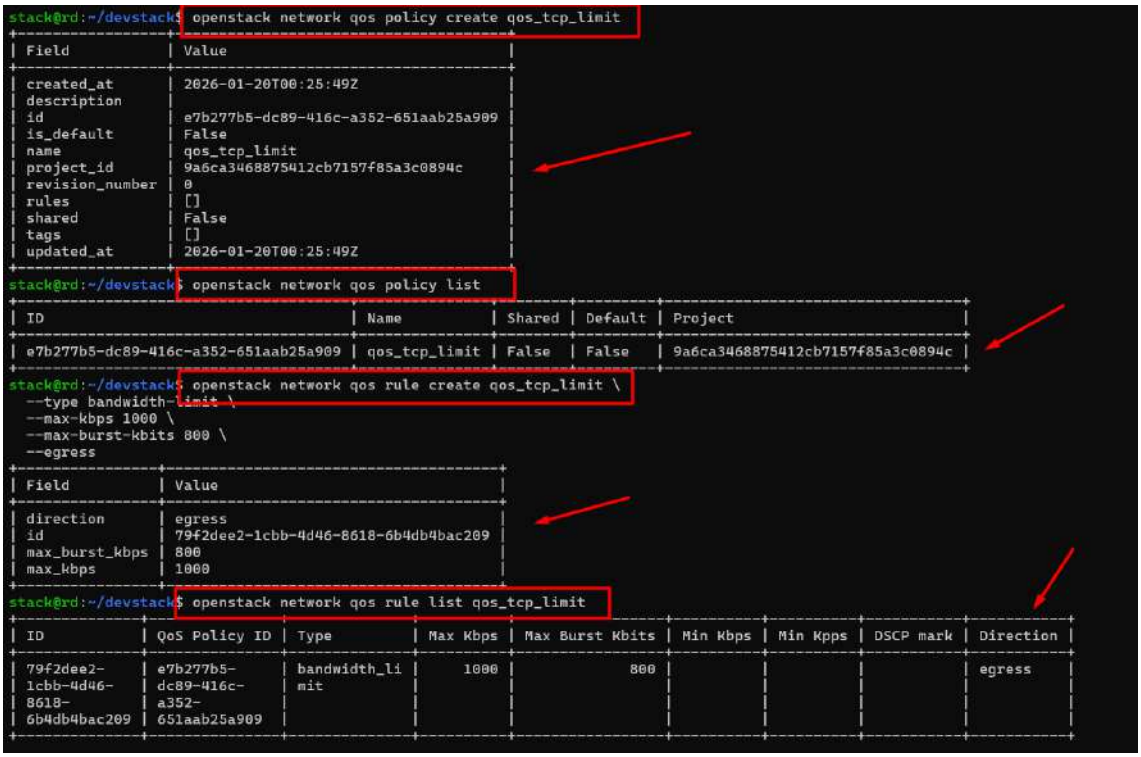
- *openstack network qos rule create qos_tcp_limit* \
- *--type bandwidth-limit* \
- *--max-kbps 1000* \
- *--max-burst-kbits 800* \
- *--egress*

Después se verifica la regla creada.

- *openstack network qos rule list qos_tcp_limit*

Figura 43

Comandos para crear una política



Nota: En la figura se muestra el orden correcto de los comandos para crear una política para QoS.

El paso siguiente es fundamental, ya que debemos obtener el ID del servidor para poder asociar la política, el siguiente comando filtra directamente ID asociada con la IP del servidor.

- openstack port list --fixed-ip ip-address=172.24.4.215
- openstack port list --fixed-ip ip-address=172.24.4.188

Una vez se obtenga la ID del servidor se aplica la política **¡Error! No se encuentra el origen de la referencia.**.

- openstack port set --qos-policy qos_tcp_limit PORT_ID_172_24_4_215

Figura 44

ID de los diferentes servicios

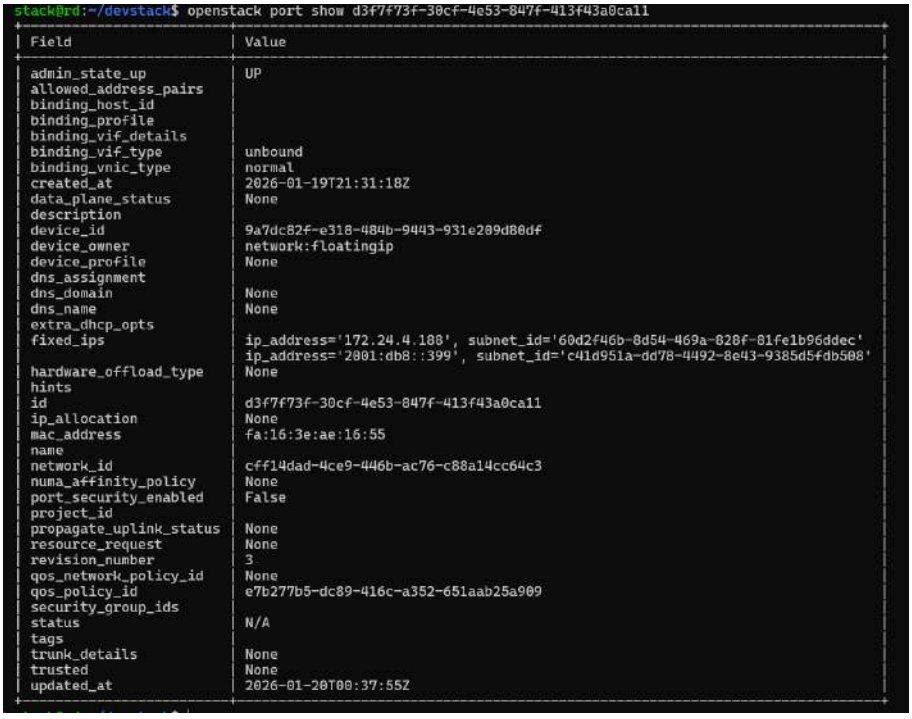
Nota: Esta parte es muy importante ya que nos muestra como filtrar para encontrar el ID correcto.

Comprobar es una buena opción, para evitar fallos **¡Error! No se encuentra el origen de la r eferencia.**.

- openstack port show PORT_ID_172_24_4_215

Figura 45

Comprobación de la política creada

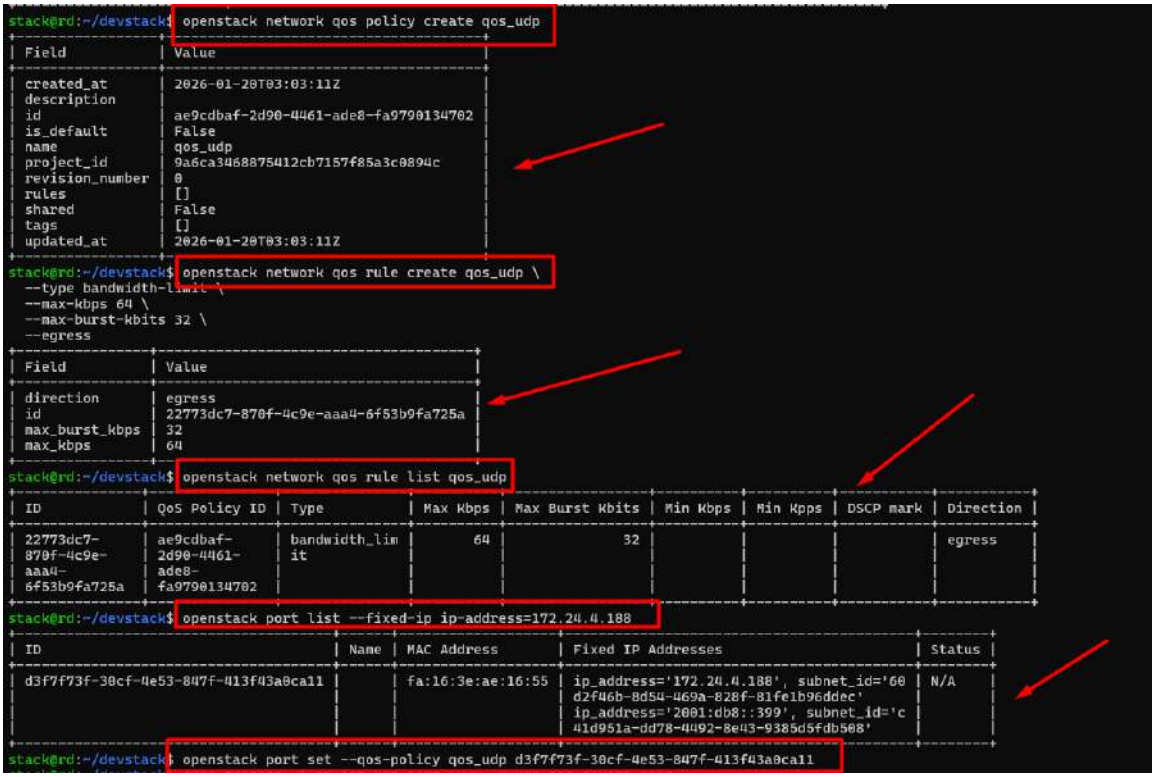


Nota: Como en pasos anteriores, siempre tenemos que comprobar. Ya que DevStack es muy inestable y a veces falla.

Para la aplicación de QoS con tráfico UDP en el puerto 5002. Se sigue los pasos anteriores modificando nombre, es importante saber que en DevStack, no tenemos manejo de QoS se tiene manejo directo con esto, ya que nos limita, en este caso solo se lo puede utilizar para regular el tráfico, poniendo límites al AB **¡Error! No se encuentra el origen de la referencia.**.

- openstack network qos policy create qos_udp
- openstack network qos rule create qos_udp \
- --type bandwidth-limit \
- --max-kbps 64 \
- --max-burst-kbits 32 \
- --egress
- openstack network qos rule list qos_udp
- openstack port list --fixed-ip ip-address=172.24.4.188
- openstack port set --qos-policy qos_udp d3f7f73f-30cf-4e53-847f-413f43a0ca11

Figura 46
Configuración de QoS para UDP



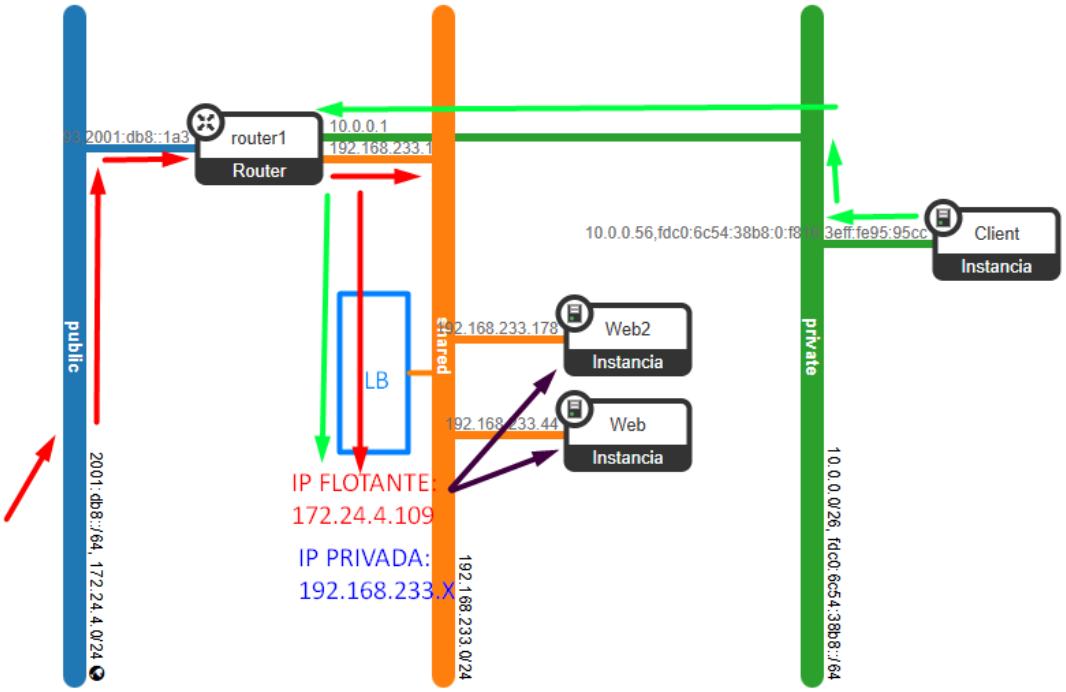
Nota: Como se observa, son los mismos comandos para TCP, cambiando el nombre de la regla y el ID de cada servidor.

3.5.2.2. Balanceo o distribución de carga.

Para entender el balanceo de carga, vamos a ver la siguiente topología **¡Error! No se encuentra el origen de la referencia.**. Donde se muestra el funcionamiento del balanceador, su

procedencia es de una balanceador existente Octavia, este funciona como una instancia, la cual no se la puede ver, pero tiene un sabor, Amphora. Y su funcionamiento es similar a una red normal, que hace sus peticiones y solicitudes al router y después al servidor, el único cambio es el aumento del balanceador, una vez que se hagan las solicitudes al router, este las envía al balanceador y este es el que responde, ya no router.

Figura 47
Funcionamiento del balanceador de carga

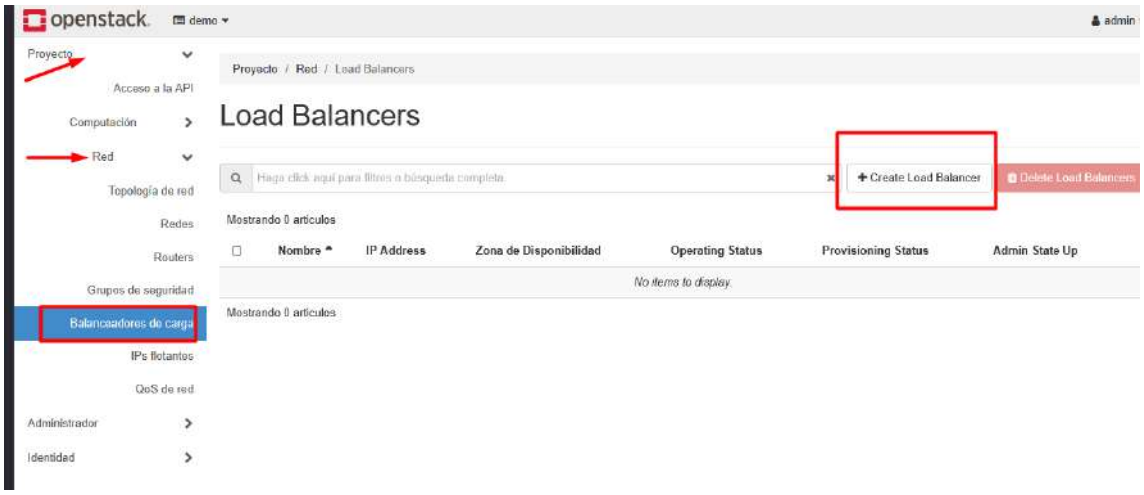


Nota: La líneas verdes y rojas nos indica el camino de las solicitudes que llegan al balanceador y las líneas negras representan las respuestas.

Para la configuración del balanceador, en el apartado de Red **¡Error! No se encuentra el origen de la referencia.**, existe una subsección que es Balanceadores de carga.

Figura 48

Creación del Balanceador de Carga

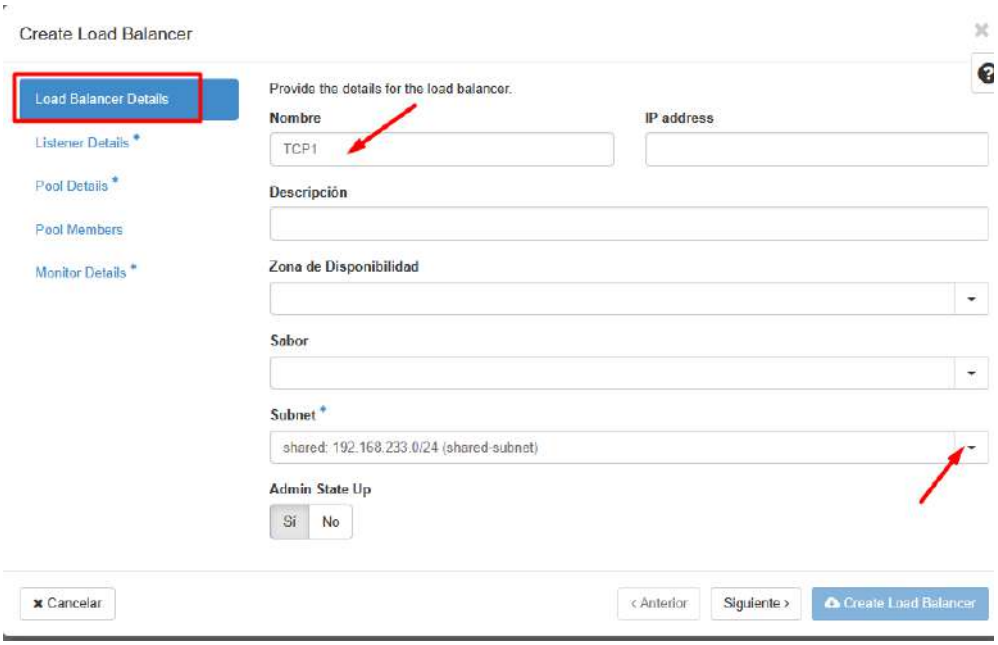


Nota: Esta es la vista del dashboard para encontrar la ubicación del balanceador.

Realizar o configurar es muy fácil, ya que solo se llena los campos necesarios **¡Error! No se encuentra el origen de la referencia.**.

Figura 49

Configuración interna del balanceador

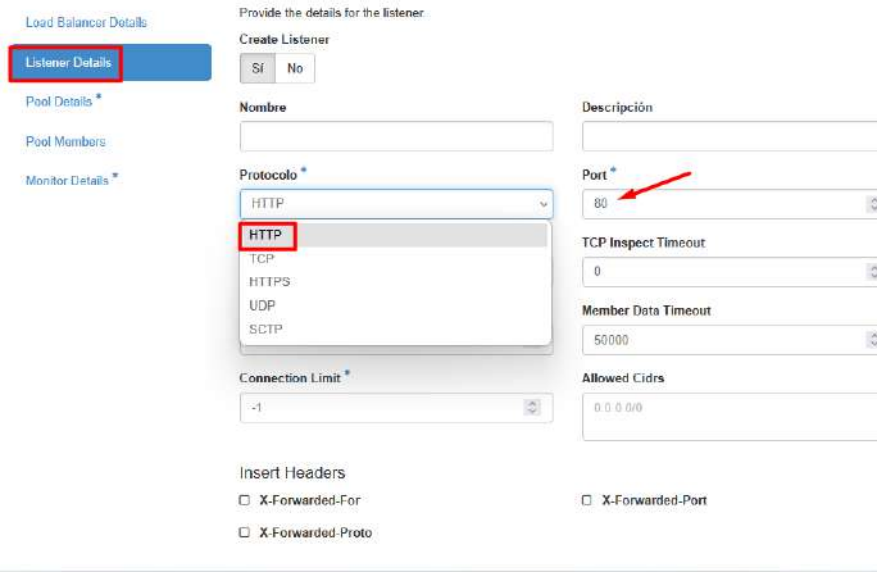


Nota: En la imagen se muestra las configuraciones del balanceador.

Para poder establecer en concordancia con los servidores Web, este va a manejar el protocolo HTTP **Figura *50***.

Figura 50

Configuración interna del balanceador

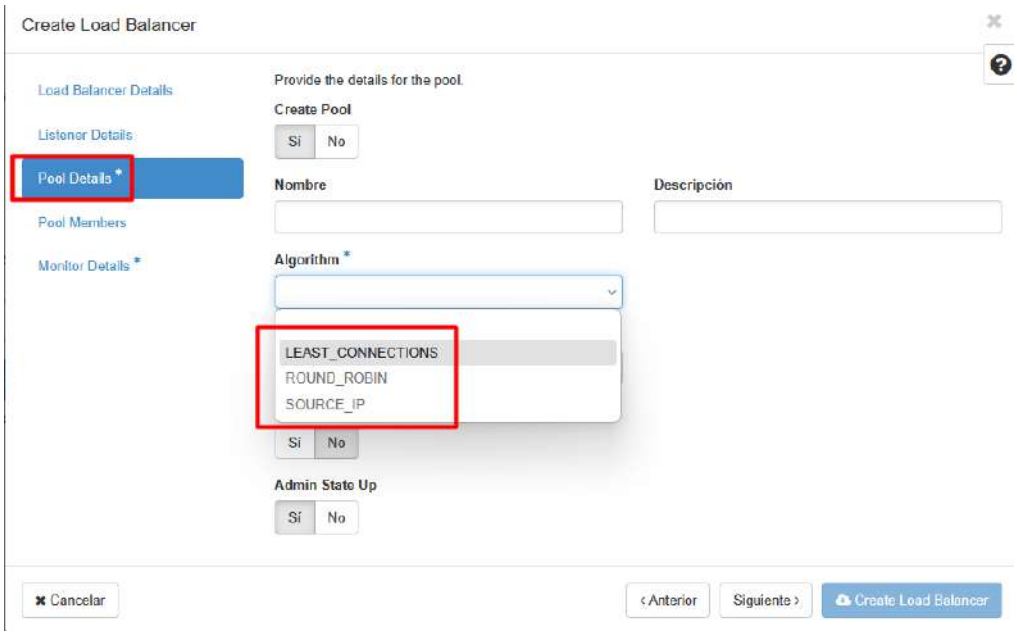


Nota: En la imagen se muestra las configuraciones del balanceador de acuerdo al protocolo.

Una de las desventajas que maneja el balanceador es que solo puede trabajar bajo 3 criterios. Para esta parte, se trabaja bajo el criterio de ROUND_ROBIN, el cual consiste en cambiar de servidor al momento de que se termina con uno **Figura *51***.

Figura 51

Tres métodos de aplicar el balanceador



Nota: Tres métodos de configurar el balanceador, se tiene que aplicar de acuerdo al tráfico aplicado.

El balanceo de carga es una técnica fundamental en infraestructuras como servicio debido a que permite distribuir las solicitudes de red entre múltiples servidores, evitando la saturación de un único recurso y mejorando la disponibilidad del sistema, en este proyecto se analizaron tres métodos de balanceo ampliamente utilizados en entornos virtualizados y de nube **¡Error! No se encuentra el origen de la referencia.**, los cuales presentan comportamientos distintos según el tipo de tráfico y la carga del sistema.

Tabla 18

Métodos de balanceo

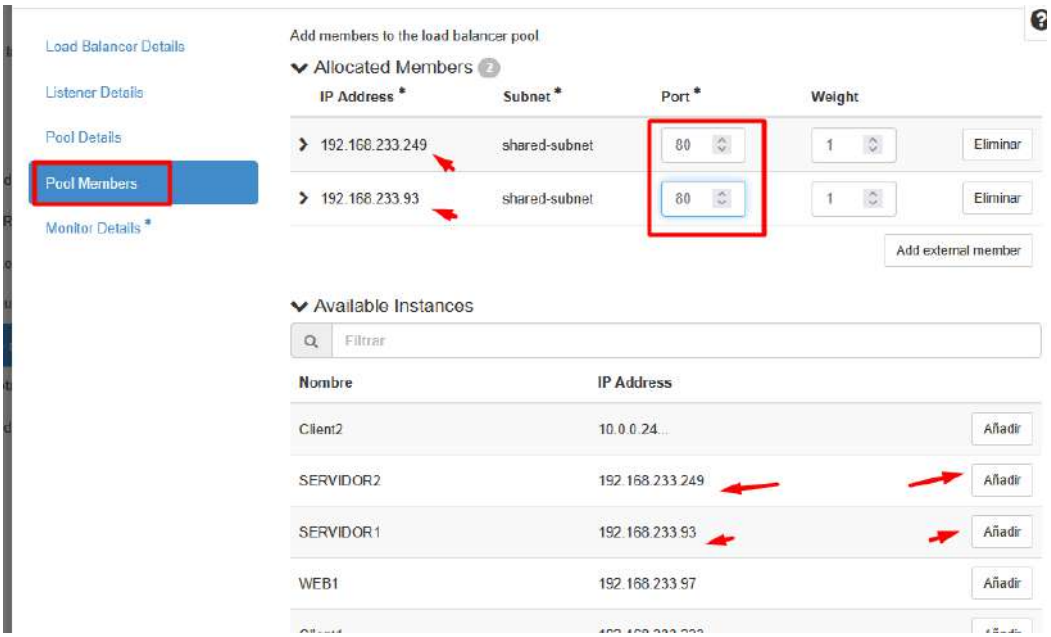
Método	Descripción	Cuando se utiliza	Ventaja principal
Round Robin	Distribuye las solicitudes de manera secuencial entre los servidores disponibles	Cuando los servidores tienen capacidades similares y las solicitudes son homogéneas	Simplicidad y reparto equitativo del tráfico
Least Connections	Asigna la nueva solicitud al servidor con menor número de conexiones activas	Cuando las solicitudes tienen duraciones variables y cargas diferentes	Mejor adaptación a cargas dinámicas
Source IP	Asigna las solicitudes según la dirección IP de origen del cliente	Cuando se requiere que un mismo cliente sea atendido siempre por el mismo servidor	Persistencia de sesión y estabilidad

Nota: La tabla indica los tres métodos disponibles para aplicar el balanceo de carga.

Este parte es muy importante es en donde se escoge a los servidores y se establece el puerto **¡Error! No se encuentra el origen de la referencia.**.

Figura 52

Configuración interna del balanceador por puerto

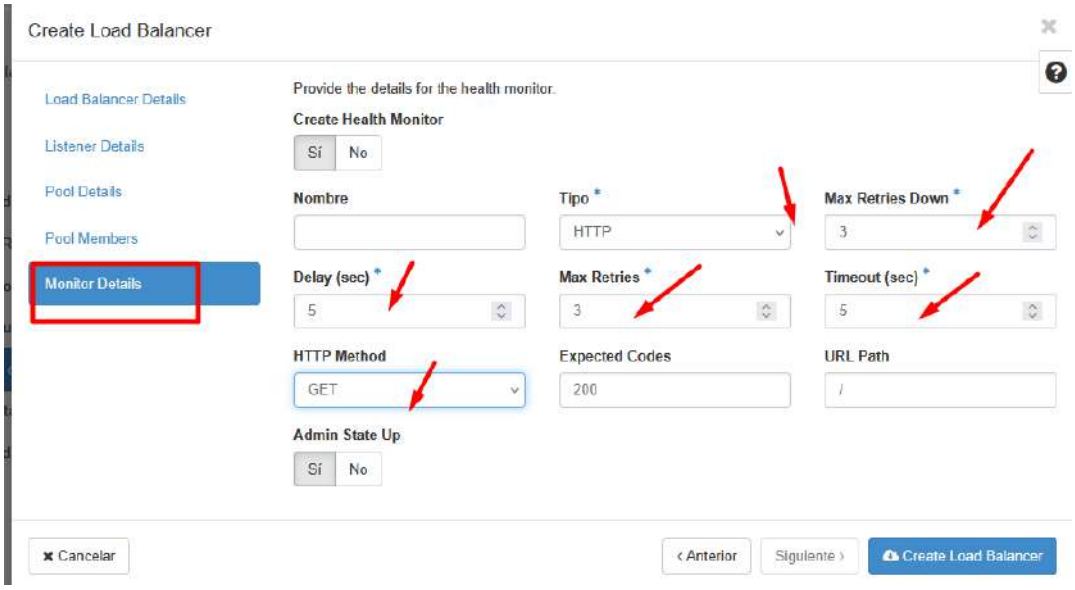


Nota: Aquí asociamos cada servidor a un solo balanceador y a un solo protocolo.

Y por último configuramos los detalles, con tiempo, etc.**Figura *53***.

Figura 53

Configuración interna del balanceador por protocolo



Nota: Aquí se especifica el protocolo.

Se crea el balanceador y esperamos a que este nos aparezca, este proceso puede demorar e incluso fallar, ya que este crea por dentro una programación interna **Figura *54***.

Figura 54

Creación del balanceador

Nota: Este proceso puede tardar y a veces incluso fallar.

Aquí la espera es fundamental, ya que al ser DevStack, este puede ser estable y a su vez no crearse, no es falla de la creación, sino más bien del propio OpenSatck.

3.5.2.3. Calidad de Servicio en el balanceador de carga

La aplicación de calidad de servicio en el balanceador de carga se justifica debido a que este componente actúa como punto central de recepción y distribución del tráfico dentro de la infraestructura, por lo tanto, cualquier congestión o mal manejo en esta etapa puede afectar directamente el rendimiento de todos los servidores asociados.

Al implementar políticas de calidad de servicio en el balanceador se logra priorizar determinados flujos de tráfico antes de que sean distribuidos hacia los servidores backend, garantizando que las solicitudes críticas mantengan tiempos de respuesta estables incluso cuando el volumen de tráfico aumenta.

En otras palabras, el QoS se aplicará después de aplicar el balanceo de carga.

CAPÍTULO IV: Operación y Resultados

En este capítulo se presenta la operación de la infraestructura OpenStack implementada y el análisis de los resultados obtenidos durante la ejecución de las pruebas definidas en el desarrollo del proyecto. A partir del entorno configurado y de las instancias desplegadas, se evalúa el comportamiento de los servicios y de los recursos del sistema bajo diferentes condiciones de carga, considerando métricas relacionadas con el uso de CPU, memoria, almacenamiento y red.

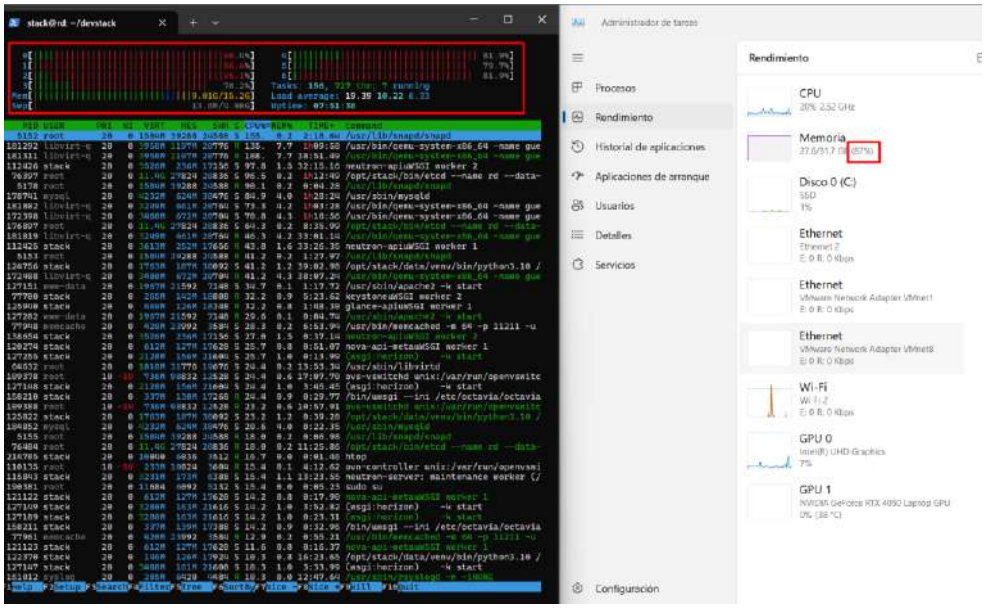
Asimismo, se exponen los resultados obtenidos a partir de las mediciones realizadas, los cuales son interpretados tomando como referencia los parámetros funcionales reales y los modelos de escalamiento definidos previamente. De esta manera, el capítulo permite analizar de forma objetiva el desempeño de la infraestructura y establecer una relación directa entre la configuración implementada y el comportamiento observado, sentando las bases para la discusión técnica y las conclusiones del proyecto.

4.1. Consumo de recursos durante las pruebas de creación, configuración y manejo de instancias

Se va a analizar este caso ya que la creación de instancias, es el punto más alto en donde los procesadores están al límite, incluso mayor, en cuanto a estar prendida toda la topología. Esto fue medido después de tener la topología final, ya que si tomamos el punto de pardita que es la creación de una arquitectura con consumo de recursos super elevados, llegando al 97% de proceso del host físico, este solamente llegara al fallo y fallara.

Figura 55

Resultados del Consumo de recursos al crear una instancia, mediante HTOP.



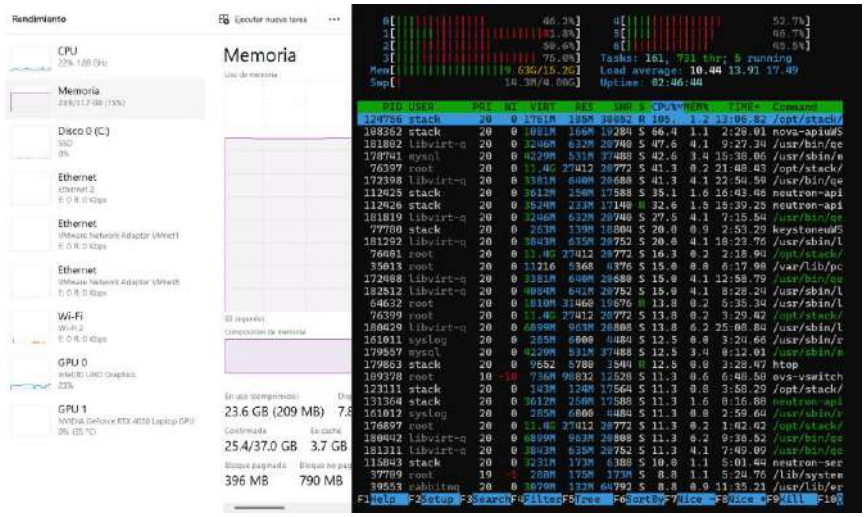
Nota: Este es el escenario en donde se utiliza mayor procesamiento.

La **¡Error! No se encuentra el origen de la referencia.** evidencia el comportamiento del c onsumo durante el aprovisionamiento de una instancia, observándose el incremento temporal de carga debido al arranque del sistema operativo, la inicialización de servicios y la asignación de recursos por el hipervisor, tras la estabilización se identifica el estado operativo esperado

para el perfil de instancia seleccionado. Estos valores no varían casi nada, si una instancia tiene diferente sabor o tamaño de CPU.

Figura 56

Resultados del Consumo de recursos estables, mediante HTOP



Nota: Se muestra un consumo de recursos estables.

En la **Figura *56***, se muestra un consumo de procesamiento moderado, en donde toda la topología esta encendida, y probando los servidores. Sin importar en escenario al que se encuentra sometida.

De esta forma podemos medir el parámetro de escalabilidad, mediante el consumo de recursos, este no depende del tipo de escenario, sino al dimensionamiento de recursos asignados.

Tabla 19

Tabla para comprobar la escalabilidad del proyecto

Configuración	Porcentaje host físico	Porcentaje	Media	Descripción
Baja	% 75	% 80	% 77.5	Arquitectura con bajos recursos asignados
Media	% 87	% 85	% 86	Arquitectura con recursos propiamente distribuidos
Alta	% 90	% 95	% 92.5	Arquitectura con recursos altos, lenta respuesta

Fuera de rango	% 97	% 100	% 98.5	Arquitectura aplicando los recursos mas altos. Respuesta nula.

Nota: La tabla presenta las diferentes configuraciones, sin importar en escenario, toma de referencia el mayor consumo de procesos.

En la **¡Error! No se encuentra el origen de la referencia.**, se compara los diferentes tipos de c onfiguración para la creación y rendimiento de las arquitecturas, en este vemos que el porcentaje de procesamiento tanto del fisco como del virtual. Llegando a la conclusión que estos son similares, y depende de los recursos que el host físico asigna al virtual, El valor de media, representa un entorno real, en el que se expresa que, si el procesamiento de la arquitectura es mayo a % 90, esta deja de funcionar, el tipo de configuraciones debe ser menor a este valor para poder trabajar sin ningún problema.

De esta manera se llega a la conclusión que este proyecto si es escalable, ya que depende mucho de los recursos que se le asigne, los cuales son expandibles, con la condición de balancear estos.

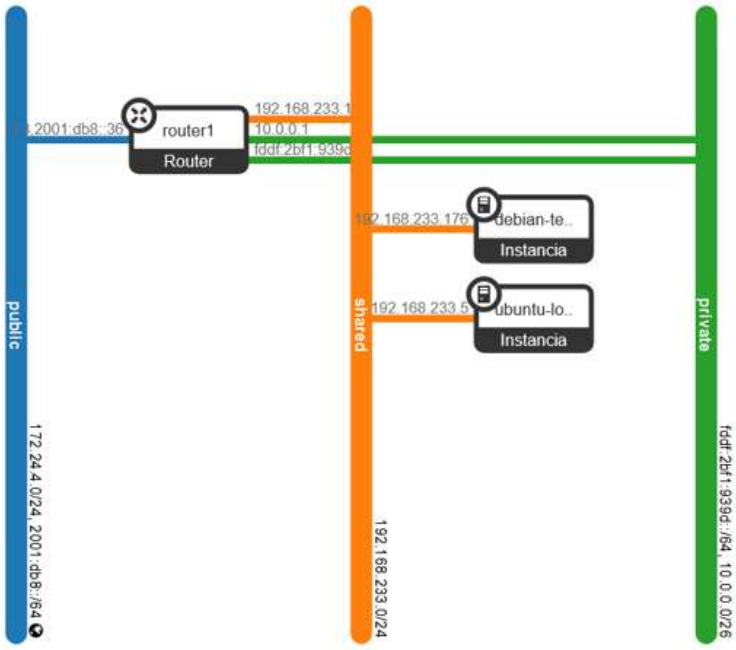
4.2. Topología inicial de servidores y clientes – Selección de arquitectura.

Para el resultado de este apartado se tuvieron en cuenta diversos factores, que permitan trabajar sin ningún problema a la topología.

Al principio se optó por utilizar solo dos instancias robustas como servidores finales, utilizando Ubuntu y Debian **¡Error! No se encuentra el origen de la referencia.**, utilizando sabores g randes, para poder configurar todos los servicios necesarios, pero el tiempo de reacción de estas instancias era demasiado, e incluso no alcanzaban a crearse, debido a los recursos del host físico. Legando a consumir un 97% del procesador y dejando sin respuesta al mismo.

Figura 57

Primera topología de prueba



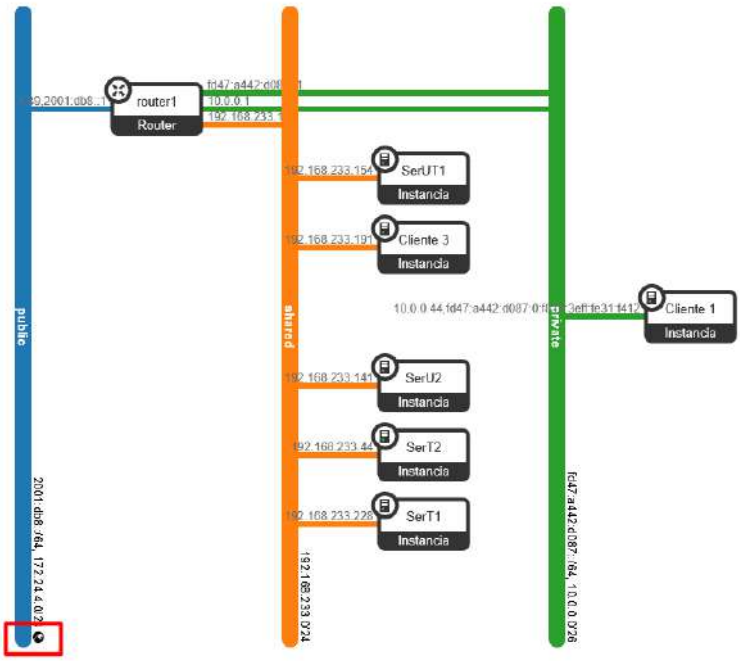
Nota: En esta topología se observa solo dos instancias que simulan todos los tráficos necesarios, para hacer pruebas de rendimiento.

Buscando soluciones acordes con el objetivo específico dos de este proyecto, se probó diferente configuración de instancias y se logró llegar al diseño **¡Error! No se encuentra el origen de l a referencia.**, en este se muestra 2 servidores para tráfico TCP y dos Para Trafico UDP, más dos clientes.

El principal cambio con el primer diseño, fue aumentar el número de instancias, pero disminuir sus características al momento de iniciar los sabores, en total se utilizaron 5 instancias de base Cirros, con un sabor muy pequeño del mismo nombre cirros. Y para pruebas un poco más fuertes se optó por una instancia de base Debian, con el sabor small. Toda esta configuración permitió trabajar al host físico con un 87% de proceso. Haciendo esta arquitectura la deseada para este entorno.

Figura 58

Topología final



Nota: Esta es la topología mejora, con la que se trabajó para encontrar los resultados con medición de los parámetros ya establecidos.

Partiendo de la distribución de la topología, se llego a establecer la arquitectura deseada, donde con los recursos que tenemos disponibles y con configuración adecuada de cada servicio. De esta manera también vemos que si existe escalabilidad en este proyecto. Todo se basa en la distribución correcta.

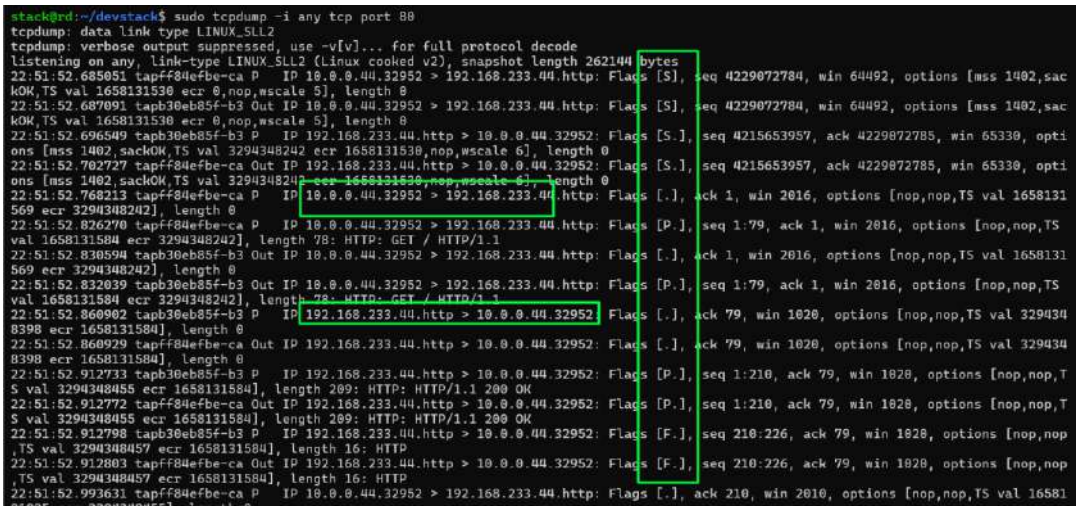
4.3. Análisis de tráfico implementado en los diferentes escenarios

Como se logró describir en el capítulo III, de este proyecto, para poder realizar la optimización del rendimiento en entorno IaaS, se planteó hacer 3 tipos de escenarios, donde se puede evaluar los parámetros latencia, Jitter y Throughput. Y de esta forma verificar si es o no factible optimizar el rendimiento con técnica de optimización que es balanceo de carga QoS.

El punto de partida es analizar el tráfico de la arquitectura sin aplicar ninguna técnica de optimización, en este punto no imparta si el tráfico es UDP o TCP. Ya que solo vemos que exista estos.

Figura 59

Trafico TCP sin aplicar técnicas de optimización

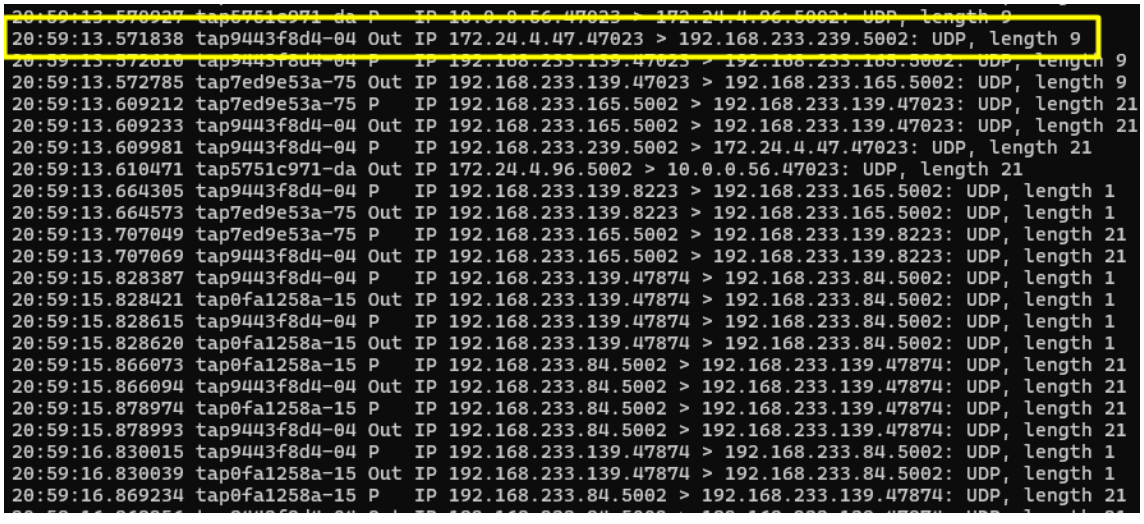


Nota: Para ello se utilizó la herramienta TCPDUMD, la cual nos permite visualizar que si existe tráfico.

En la **¡Error! No se encuentra el origen de la referencia.** vemos la captura de paquetes sin u tilizar ningún tipo de optimización, como diferenciamos si es TCP o UDP, por las banderas [S], [P] o [F] que indican el estado del protocolo, también vemos las IP´s tanto de origen como destino y el tamaño de ventana. Esta imagen se basó en las capturas del servidor WEB.

Figura 60

Trafico UDP sin aplicar técnicas de optimización



Nota: Para ello se utilizó la herramienta TCPDUMD, la cual nos permite visualizar que si existe tráfico.

En la **¡Error! No se encuentra el origen de la referencia.**, se observa el tráfico UDP generado p or la herramienta netcat en cirros, la clara diferencia entre TCP, es que esta no lleva las banderas que indican estado, pero si vemos tanto la IP de origen como de destino.

Una vez establecidos estos dos puntos vamos a analizar cada escenario:

- **Solo aplicando QoS**

Para que se evidencia de mejor manera la aplicación del QoS, se utilizara la herramienta de Wireshark ya que es más intuitiva y nos da los resultados en forma de graficas.

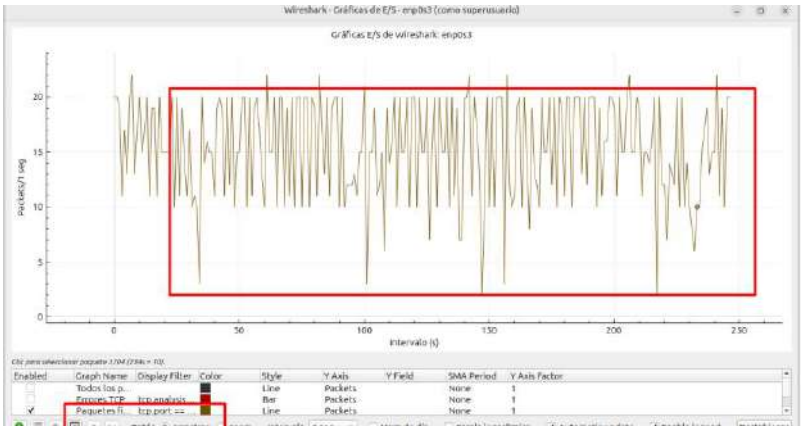
A diferencia de una implementación de QoS normal que agrega etiquetas, OpenStack, no lo hace, ya que funciona como una llave de flujo aplicada directamente al tráfico, al ser una distribución pequeña, DevStack, no permite realizar muchos tipos de implementación de QoS, pero para este proyecto es mas que suficiente, ya que nos deja controlar el AB, permitiendo que trafico llegue con mayor o menor fuerza, aplicando un margen mas grande a un tráfico más necesario.

De esta forma obtenemos los siguiente, se creó una política en donde el tráfico tendrá estas condiciones:

- *max-kbps 128*
- *max-burst-kbits 64*

Figura 61

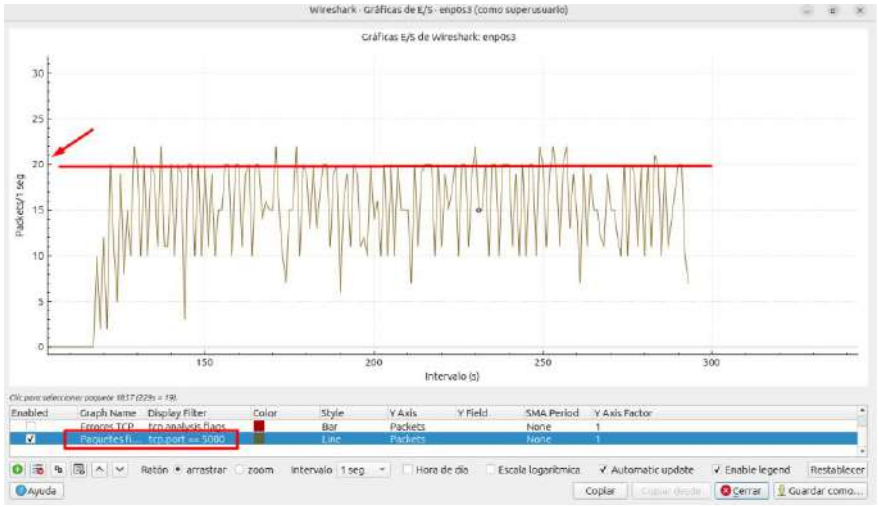
Grafica de comportamiento de tráfico TCP sin QoS



Nota: Este es el comportamiento del tráfico sin ningún cambio.

Figura 62

Grafica de comportamiento de tráfico TCP con QoS

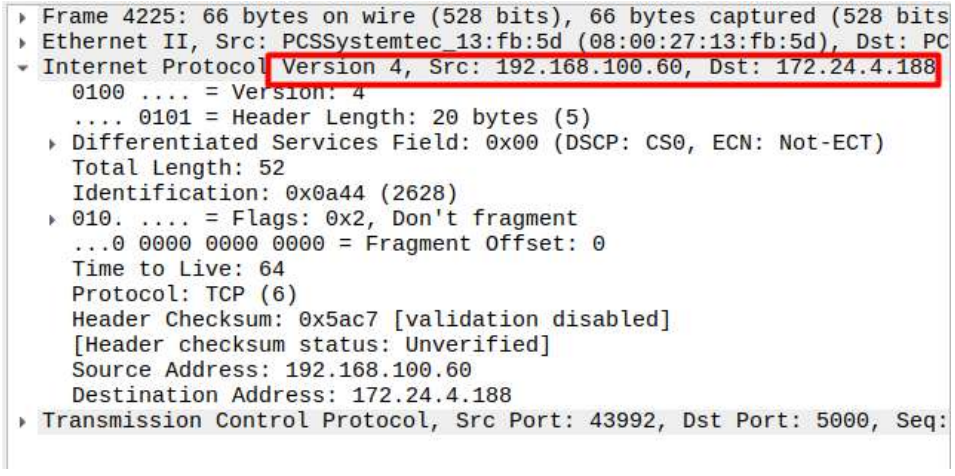


Nota: Este es el comportamiento del tráfico aplicado QoS, controlando el AB.

Se va a comparar la **¡Error! No se encuentra el origen de la referencia.** con la **¡Error! No se encuentra el origen de la referencia.**, donde se observa la grafica del comportamiento, en la primera no está aplicado el QoS y en el segundo sí. A simple vista si se logra ver que la señal es mas esta cuando se ocupa QoS, manteniéndole dentro de un solo rango, donde las pérdidas se pueden controlar, teniendo una comunicación continua.

Figura 63

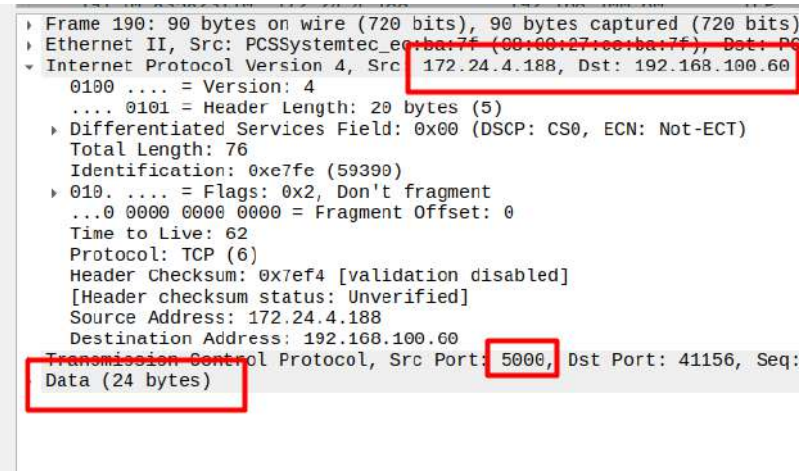
Análisis del protocolo TCP con QoS



Nota: Se observa el comportamiento del paquete mandado.

Figura 64

Análisis del protocolo TCP sin QoS



Nota: Se observa el comportamiento del paquete mandado sin QoS.

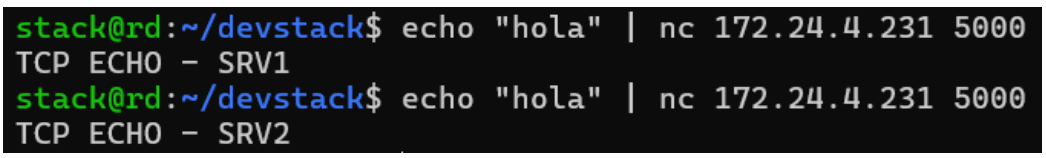
Para el análisis en la **¡Error! No se encuentra el origen de la referencia.** y **¡Error! No se en cuentra el origen de la referencia.** se observa claramente como se comporta el paquete, se ve el puerto y el protocolo que es el mismo y las IP´s de origen y destino. Se ve el cambio en el tamaño de ventada, donde el uno esta dentro del rango establecido.

- **Solo aplicado Balanceo de Carga**

Aquí se limita mucho, ya que solo tenemos tres métodos de balanceo de carga, en este caso se utilizó lo que es un Round-robin, este método ayuda a que los servicios no se saturen. Funciona de la siguiente manera, se tiene servidores del mismo tipo y para que no exista cuello de botella, el balanceador manda solicitudes alternadas entre los servicios. Y manda una respuesta a la misma IP, como se ve la **¡Error! No se encuentra el origen de la referencia.**.

Figura 65

Respuesta del balanceador de carga

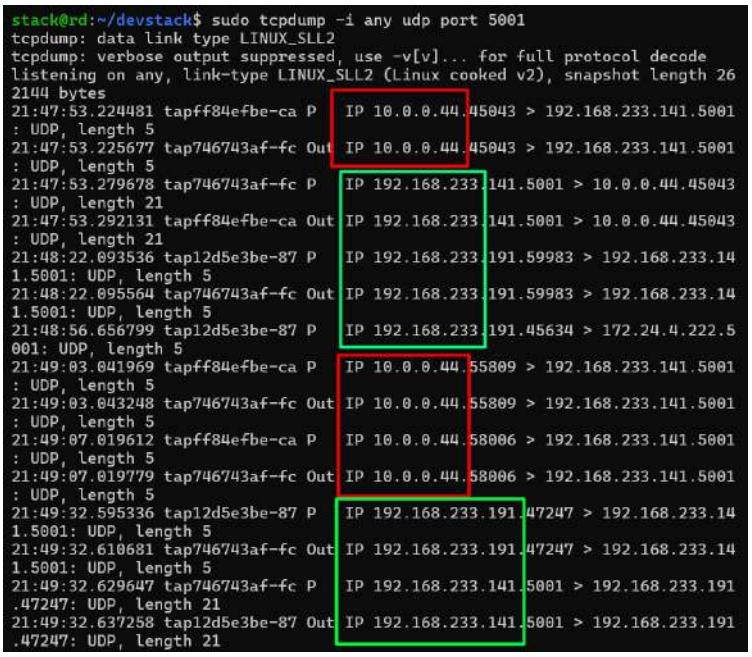


Nota: El comando demuestra que existe una respuesta de cada servidor, pero proveniente de la misma IP

Para tener una mejor se observa como el protocolo actúa sobre el balanceador, el protocolo no se daña, siempre tiene un inicio y fin. Pero la respuesta proviene de las diferentes IP´s, sin dañar el protocolo como se observa en la **¡Error! No se encuentra el origen de la r eferencia.**.

Figura 66

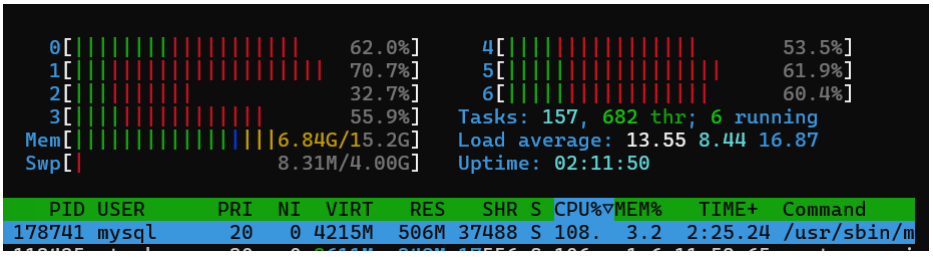
Captura de paquetes con el balanceador de carga



Nota: TCPDUMP logra capturar el tráfico, dando evidencia que no se daña el protocolo.

Figura 67

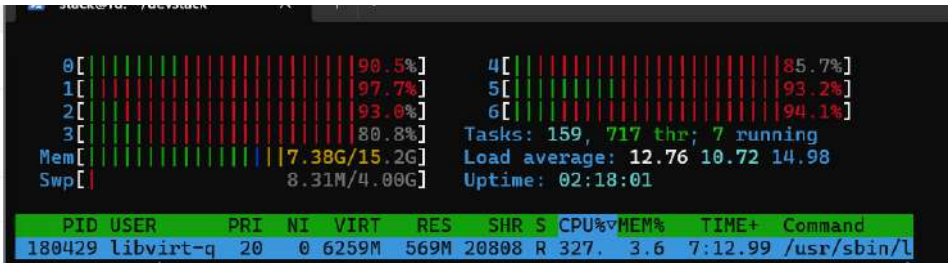
Resultados del procesamiento con Balanceador de carga



Nota: Se observa el comportamiento del procesamiento después de balancear las cargas

Figura 68

Resultados del procesamiento sin Balanceador de carga



Nota: Se observa el comportamiento del procesamiento antes de balancear las cargas

Entre la **¡Error! No se encuentra el origen de la referencia.** y la **¡Error! No se encuentra el origen de la referencia.** vemos el claro comportamiento del procesamiento al aplicar un balanceo de carga adecuado, sin dejar que la CPU colapse y distribuyendo esto, teniendo una mejor respuesta por el comportamiento de este balanceo. En conclusión, aparte de evitar un cuello de botella, este impide que los recursos asignados se colapsen en tanto a rendimiento.

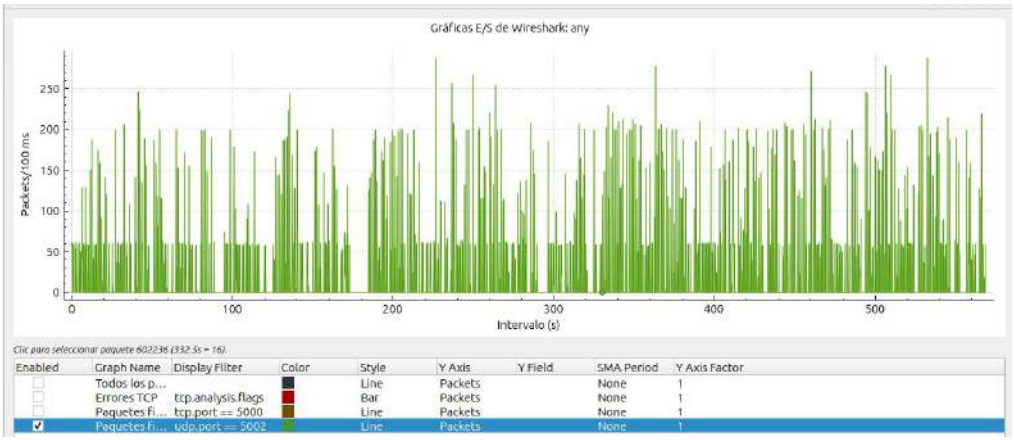
- **Aplicando Balanceo de y Calidad de Servicio**

Para este apartado se debe tener un orden lógico, para sacarle el mayor provecho posible la optimización de recursos, si se aplica primero la calidad de servicio y después el después el balanceo de carga, si funcionara, pero la pérdida de tiempo y de procesos internos iría en contra de la optimización, lo adecuado será aplicar el balanceador y después la calidad de servicio ya que de esta forma al aplicar la calidad se lo hará de forma general y no solo de un punto. Casos en el que se puede aplicar primero la calidad es solo si un servidor de los muchos que se puede llegar a programar necesita un trato especial.

Otra ventaja de aplicar primero el balanceo es el resultado en sí, ya que por medio del analizador de tráfico se verá que la carga se distribuye y no es necesario aplicar calidad de servicio, esta conclusión se la obtuvo mediante medida de los escenarios presentados.

Figura 69

Aplicando Balanceo de y Calidad de Servicio en tráfico UDP

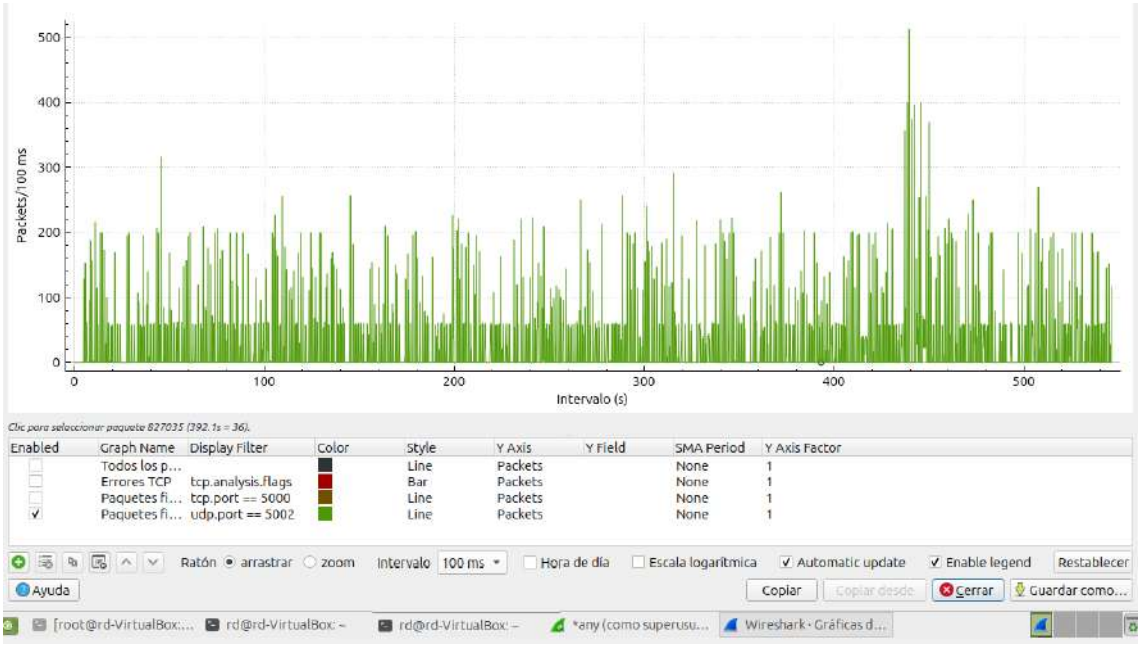


Nota: Esta imagen es el tráfico de entrada y salida optimizado.

Para tener una mejor apreciación se compara la **¡Error! No se encuentra el origen de la r eferencia.** y **¡Error! No se encuentra el origen de la referencia.** las cuales se ve clara mente el comportamiento del tráfico, este se lo realizo por UDP ya que es donde existe mayor apreciación de predida de paquetes, ya que no existe como tal cuna conexión y se logra distinguir el Throughput, Latencia y Jitter claramente sin necesidad de aplicar formulas. De todas maneras, en las siguientes figuras se logrará apreciará de mejor forma. Como se ve la transmisión de paquetes no es regular y dependiendo de donde provenga esta puede tener mayor perdida.

Figura 70

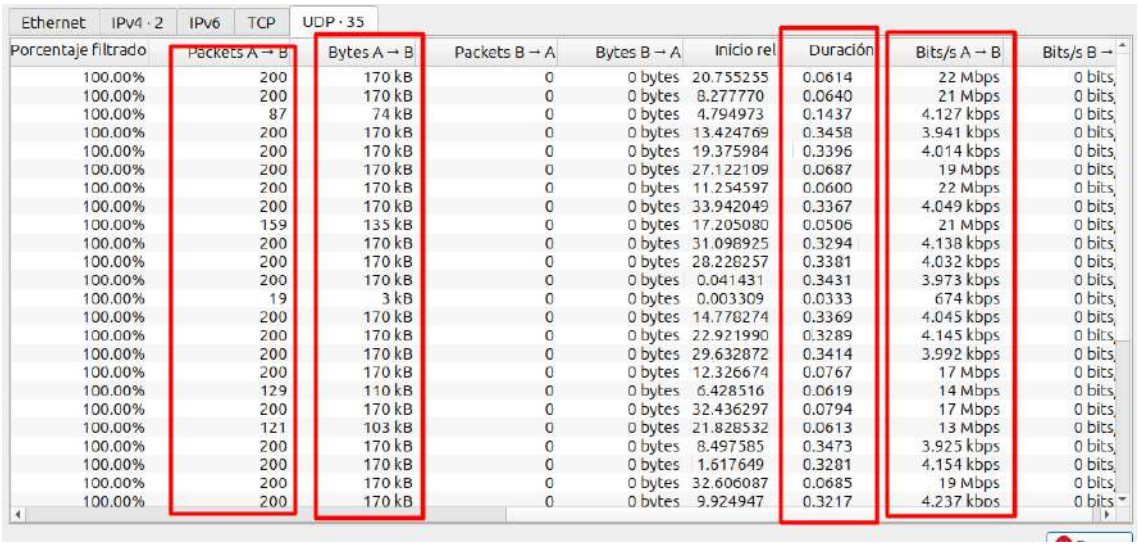
Trafico UDP



Nota: Esta imagen es el tráfico de entrada y salida no optimizado.

Figura 71

Trafico UDP en tiempos



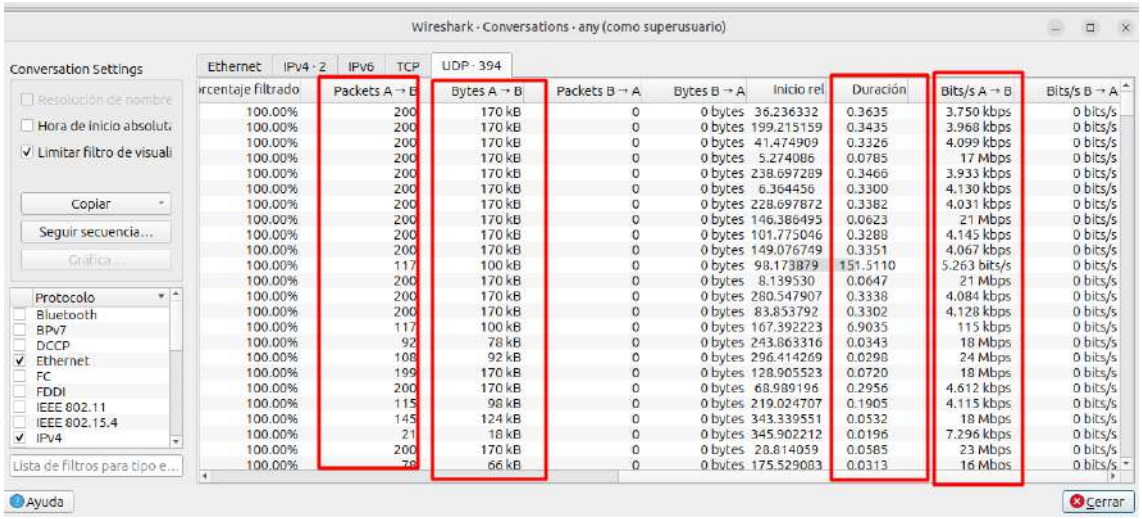
Porcentaje filtrado	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Inicio rel	Duración	Bits/s A → B	Bits/s B →
100.00%	200	170 kB	0	0 bytes	20.755255	0.0614	22 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	8.277770	0.0640	21 Mbps	0 bits
100.00%	87	74 kB	0	0 bytes	4.794973	0.1437	4.127 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	13.424769	0.3458	3.941 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	19.375984	0.3396	4.014 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	27.122109	0.0687	19 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	11.254597	0.0600	22 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	33.942049	0.3367	4.049 kbps	0 bits
100.00%	159	135 kB	0	0 bytes	17.205080	0.0506	21 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	31.098925	0.3294	4.138 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	28.228257	0.3381	4.032 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	0.041431	0.3431	3.973 kbps	0 bits
100.00%	19	3 kB	0	0 bytes	0.003309	0.0333	674 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	14.778274	0.3369	4.045 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	22.921990	0.3289	4.145 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	29.632872	0.3414	3.992 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	12.326674	0.0767	17 Mbps	0 bits
100.00%	129	110 kB	0	0 bytes	6.428516	0.0619	14 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	32.436297	0.0794	17 Mbps	0 bits
100.00%	121	103 kB	0	0 bytes	21.828532	0.0613	13 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	8.497585	0.3473	3.925 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	1.617649	0.3281	4.154 kbps	0 bits
100.00%	200	170 kB	0	0 bytes	32.606087	0.0685	19 Mbps	0 bits
100.00%	200	170 kB	0	0 bytes	9.924947	0.3217	4.237 kbps	0 bits

Nota: En esta captura vemos el dimensionamiento por paquetes y tiempos.

Para evaluar el comportamiento de la red en condiciones de congestión se generó tráfico UDP de carácter agresivo mediante el envío continuo de bloques de datos nulos desde un cliente Ubuntu MATE hacia instancias Debian, incrementando de forma gradual el volumen y la frecuencia del tráfico con el propósito de representar escenarios de alta demanda y observar el desempeño de la infraestructura virtualizada **¡Error! No se encuentra el origen de la r eferencia.**.

La política de calidad de servicio fue implementada a través de Neutron QoS mediante la aplicación de reglas de limitación de ancho de banda y marcaje DSCP sobre el puerto correspondiente a la instancia servidor, tras lo cual se repitió la generación de tráfico UDP permitiendo identificar una disminución del jitter y una mayor estabilidad en el flujo de datos como resultado directo de la aplicación de QoS en OpenStack **¡Error! No se encuentra el o rigen de la referencia.**.

Figura 72
Trafico UDP con QoS



Nota: En esta captura vemos el dimensionamiento por paquetes y tiempos con QoS.

4.4. Resultados obtenidos del análisis al implementar una optimización en todos los escenarios.

El análisis de la calidad de servicio en redes virtualizadas resulta fundamental para comprender el impacto que tienen las políticas de gestión del tráfico sobre el rendimiento de la red, especialmente en entornos de Infraestructura como Servicio donde múltiples aplicaciones comparten los mismos recursos, por lo que en este estudio se evaluaron parámetros como throughput, latencia y jitter bajo diferentes escenarios de tráfico y aplicación de políticas de QoS utilizando protocolos UDP y TCP con el fin de identificar su comportamiento ante condiciones de congestión y priorización en diferentes escenarios planteados en la **¡Error! No s e encuentra el origen de la referencia.** y donde también se aplicó un balanceo de carga adecuado.

De esta manera llegamos a la **¡Error! No se encuentra el origen de la referencia.**, la que nos m uestra en resumen los resultados obtenidos, ya que antes de esto se ha ido desglosando el proceso y la justificación adecuada. En esta tabla tenemos una comparación por escenario y vemos que el objetivo de una optimización adecuada.

Para tener mejores resultados se aplicó únicamente donde se diferencia de verdad, de esta manera el tráfico UDP, se distingue más para la aplicación de QoS y trafico UDP para Balanceo.

Tabla 20
Resultados obtenidos del análisis de calidad de servicio en tráfico de red

Protocolo	Escenario	Throughput promedio	Latencia promedio	Jitter promedio	Comportamiento observado
UDP	Escenario 1 - Sin QoS – Sin Balanceo	Alto e inestable, con picos elevados	Variable y no controlada	Elevado	El tráfico presenta picos abruptos de transmisión, variaciones significativas en el retardo y pérdida de paquetes bajo congestión
UDP	Escenario 2 - Con QoS (límite de ancho de banda)	Controlado y estable	Reducida respecto al escenario sin QoS	Bajo	El flujo de datos se mantiene constante, disminuyendo la congestión y estabilizando el comportamiento temporal
TCP	Escenario 2 - Con QoS (Marcaje)	Variable según control interno del protocolo	Moderada	Media	El protocolo ajusta dinámicamente su velocidad mediante la ventana deslizante, ocultando parcialmente los efectos de la red
TCP	Escenario 3 - Con QoS y Balanceo	Estable	Baja y uniforme	Muy Bajo	El marcado prioriza el tráfico sin interferir con el control de congestión, mejorando la regularidad del flujo

Los resultados obtenidos evidencian que la aplicación de políticas de calidad de servicio mejora de manera significativa el desempeño de la red, particularmente en el tráfico UDP donde la limitación de ancho de banda permitió reducir el jitter y estabilizar el flujo de datos, mientras que en el tráfico TCP el uso de marcaje DSCP contribuyó a una transmisión más uniforme y predecible, demostrando que la correcta selección de políticas de QoS y balanceo de carga,

evitando saturación de servicio, en función del protocolo utilizado optimiza el uso de los recursos de red y garantiza un comportamiento más eficiente en entornos virtualizados.

CAPÍTULO V: Practicas de Laboratorio

El presente capítulo describe el desarrollo de las prácticas de laboratorio realizadas en el entorno OpenStack, las cuales permitieron aplicar de forma práctica los conceptos teóricos relacionados con la gestión de recursos y el rendimiento de red en infraestructuras de tipo IaaS, abordando desde el acceso y monitoreo de los recursos disponibles hasta la evaluación del desempeño de las instancias virtuales mediante la ejecución de pruebas controladas que facilitan la comprensión del funcionamiento interno de la plataforma.

5.1. Guía 1- Acceso y Monitoreo de Recursos en OpenStack

Esta guía tiene como objetivo familiarizar al estudiante con el entorno OpenStack mediante el acceso a la plataforma y la observación de los recursos disponibles, permitiendo identificar el estado de las instancias virtuales, redes y servicios activos como base para la correcta administración y supervisión de la infraestructura IaaS.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TEMA: Guías Acceso y Monitoreo de Recursos en OpenStack

AUTOR: RAÚL DÍAZ

GUÍA DE LABORATORIO 1

Objetivo

Identificar y analizar el estado de los recursos de cómputo, red y almacenamiento en una infraestructura IaaS mediante la plataforma OpenStack.

Descripción

En esta guía se establece el procedimiento para acceder a la plataforma OpenStack a través de la interfaz web Horizon, permitiendo reconocer los principales indicadores de rendimiento del sistema, como uso de CPU, memoria RAM y almacenamiento.

Recursos necesarios

- Infraestructura OpenStack operativa

- Navegador web
- Credenciales de acceso a la plataforma

Procedimiento

1. El usuario accede a la plataforma OpenStack mediante la interfaz Horizon.

   En una MV, instalar el SO Ubuntu Server LTS 24.04

   https://ubuntu.com/download/server

   Instalar Devstask:

   Se creó el usuario stack requerido por DevStack mediante los comandos sudo useradd -s /bin/bash -d /opt/stack -m stack y echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack para permitir la ejecución de servicios con privilegios administrativos. Posteriormente se inició sesión con el usuario creado utilizando el comando sudo su - stack para continuar con el proceso de instalación.

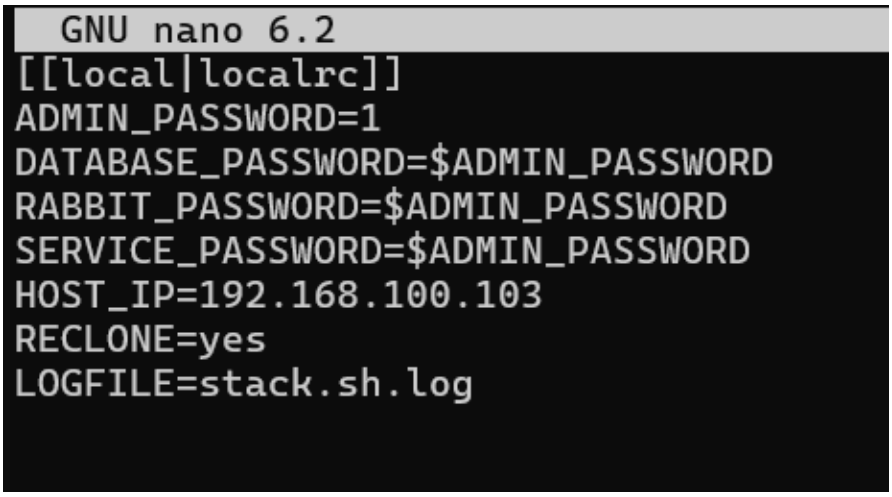
A continuación, se clonó el repositorio oficial de DevStack desde OpenDev empleando los comandos git clone https://opendev.org/openstack/devstack y cd devstack.

Luego se creó el archivo de configuración local.conf mediante el comando nano local.conf, en el cual se definieron los servicios a instalar para un entorno sin balanceador según se muestra en la **Figura *73***.**Figura *73***

Contenido del archivo local.conf sin balanceador

Figura 73

Contenido del archivo local.conf sin balanceador



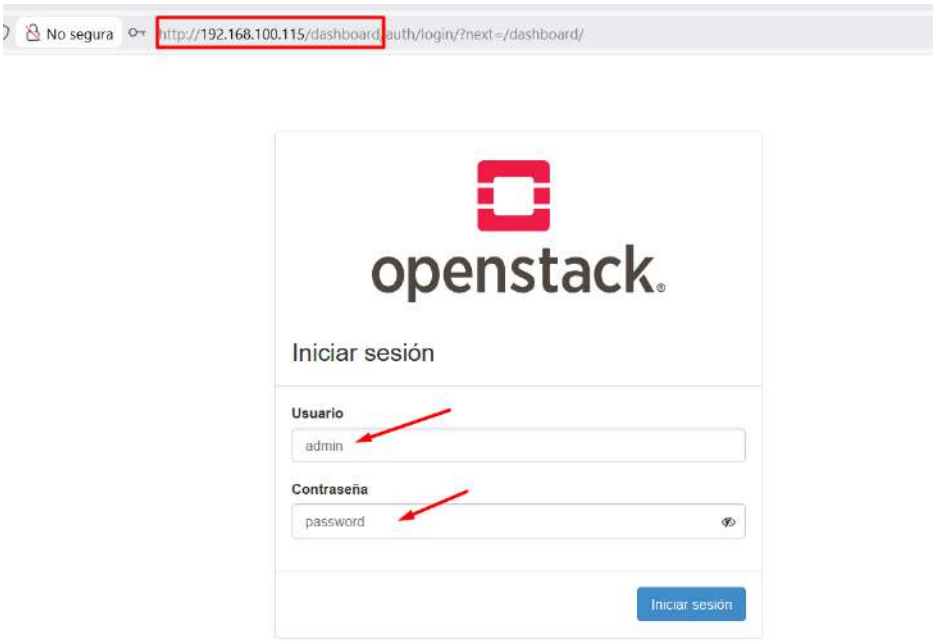
Nota: Aquí se ve el contenido que debe ir en el archivo local.conf. Sin balanceador.

Finalmente se ejecutó el script principal de instalación de OpenStack utilizando el comando ./stack.sh

2. Se autentica utilizando las credenciales asignadas.

Figura 74

Dashboard de OpenStack



Nota: Este es el inicio y una forma de controlar que la ejecución es un éxito.

3. Se navega por los paneles de administración para identificar instancias activas.
4. Se revisa el consumo de recursos de cada instancia.

Figura 75

Recursos disponibles vistos desde el Horizon

Nota: La imagen nos muestra la cantidad de recursos que puede almacenar OpenStack, con la instalación de DevStack.

5. Se registran los valores observados para su posterior análisis.

Resultados esperados

Se obtiene una visión general del estado de la infraestructura, permitiendo identificar posibles cuellos de botella en el uso de recursos.

Acceso y Monitoreo de Recursos en OpenStack

Recurso Monitoreado	Valor Observado	Estado del Recurso	Observación
CPU (%)		Óptimo / Alto	
Memoria RAM (%)		Óptimo / Alto	
Almacenamiento (%)		Disponible / Crítico	
Número de instancias activas		Normal / Elevado	

Permite identificar el nivel de uso de los recursos y posibles puntos de saturación en la infraestructura IaaS.

5.2. Guía 2- Gestión y Asignación de Recursos en Instancias Virtuales

En esta guía se aborda la gestión y asignación de recursos en instancias virtuales, analizando cómo la configuración de parámetros como memoria, procesamiento y red influye en el desempeño de las máquinas virtuales, con el propósito de comprender la importancia de una correcta planificación de recursos dentro de un entorno virtualizado.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TEMA: Gestión y Asignación de Recursos en Instancias Virtuales

AUTOR: RAÚL DÍAZ

GUÍA DE LABORATORIO 2

Objetivo

Configurar y administrar instancias virtuales en OpenStack, evaluando la asignación de recursos de cómputo según la carga de trabajo.

Descripción

La guía describe el proceso de creación, configuración y administración de instancias virtuales, considerando la asignación eficiente de CPU, memoria y almacenamiento.

Recursos necesarios

- Plataforma OpenStack
- Imagen de sistema operativo
- Red virtual configurada

Procedimiento

1. El usuario ingresa al panel de administración de OpenStack.
2. Se selecciona una imagen base para la creación de la instancia.

Mediante el dashboard de OpenStack se facilita la creación de instancias al guiar al usuario en cada apartado requerido, iniciando con la selección de la imagen en la sección de origen donde se eligió Debian en formato qcow2 como se muestra en la **Figura *76***.

Figura 76

Dimensionamiento de una instancia, lanzamiento

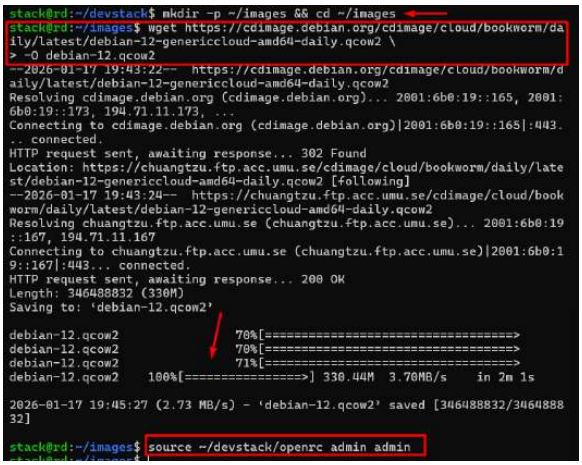
Nota: Para poder crear una nueva instancia, nos dirigimos al botón Lanzar Instancia.

3. Se asignan los recursos de cómputo requeridos.

Posteriormente se configuró el sabor de la instancia, definiendo los recursos de CPU memoria y almacenamiento, considerando que no todos los sabores son compatibles con todas las imágenes, ya que los más pequeños están destinados únicamente a pruebas con Cirros, razón por la cual se seleccionó un sabor adecuado para Debian como se observa en la **Figura *77***.

Figura 77

Configuraciones para cargar una imagen nueva



Nota: No olvidemos que a ser DevStack, no se va a poder cargar nuevas imágenes desde el dashboard, se lo hará mediante terminal

En el apartado de redes se seleccionó la red a la que pertenecería la instancia asegurando su conexión al router virtual para permitir salida a Internet, tal como se presenta en la Figura 24, mientras que las secciones de puertos grupos de servidores sugerencias planificación y metadatos no fueron configuradas por no ser necesarias para el proyecto

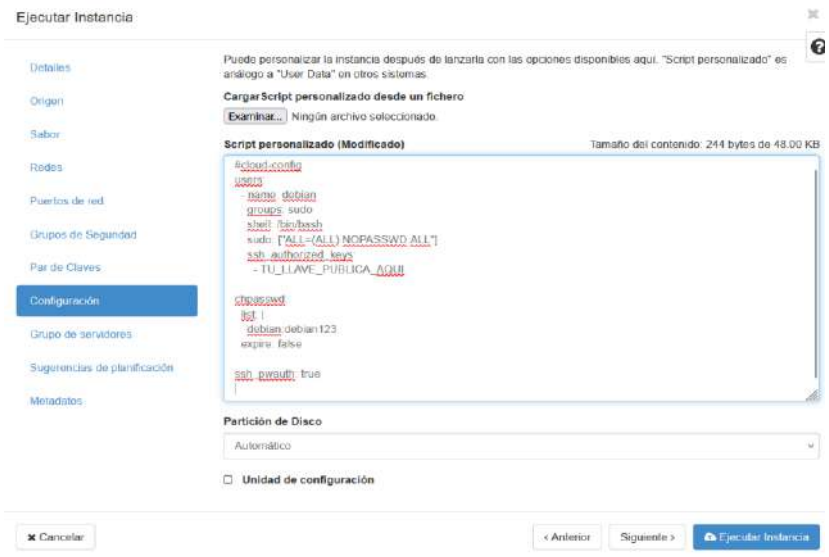
En la sección de grupos de seguridad se eligió el grupo previamente configurado para controlar el acceso y los protocolos permitidos como se aprecia, recomendándose la creación de grupos específicos según el tipo de servidor

A continuación, se seleccionó el par de claves para habilitar el acceso mediante SSH, siendo este un paso esencial para la administración remota de la instancia.

Finalmente, en el apartado de configuración se incluyó un script compatible con la imagen Debian que permitió definir el usuario credenciales y privilegios de superusuario necesarios para el acceso al sistema, tal como se muestra, considerando que este tipo de configuración es obligatoria en imágenes qcow2 y varía según la imagen utilizada.

Figura 78

Dimensionamiento de una instancia, Configuración.



Nota: Código personalizado para cada instancia

4. Se inicia la instancia virtual.

5. Se verifica su correcto funcionamiento mediante pruebas básicas.

Resultados esperados

Las instancias quedan operativas con recursos adecuados, permitiendo analizar el impacto de la asignación en el rendimiento del sistema.

Aquí debemos jugar con nuestros recursos de la MV, y sabremos cuantas instancias y el tipo alcanzan a trabajar al mismo tiempo.

Gestión y Asignación de Recursos en Instancias Virtuales

Instancia	CPU Asignada	RAM Asignada	Estado de la Instancia	Rendimiento
VM-01			Activa / Inactiva	Adecuado / Bajo
VM-02			Activa / Inactiva	Adecuado / Bajo
VM-03			Activa / Inactiva	Adecuado / Bajo

Facilita evaluar si la asignación de recursos es adecuada según la carga de trabajo, apoyando la optimización del sistema.

5.3. Guía 3- Evaluación del Rendimiento de Red en Entornos IaaS

La presente guía se enfoca en la evaluación del rendimiento de la red en un entorno IaaS mediante la generación y análisis de tráfico de red, permitiendo medir parámetros como throughput, latencia y jitter, así como observar el efecto de la aplicación de políticas de calidad de servicio sobre el comportamiento de la infraestructura virtual.

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN TELECOMUNICACIONES

TEMA: Evaluación del Rendimiento de Red en Entornos IaaS

AUTOR: RAÚL DÍAZ

GUÍA DE LABORATORIO 3

Objetivo

Analizar el rendimiento de la red en una infraestructura IaaS mediante pruebas de conectividad y latencia entre instancias virtuales.

Descripción

La guía establece un conjunto de pruebas de red entre instancias cliente y servidor para evaluar la calidad del servicio y el comportamiento de la infraestructura bajo diferentes condiciones.

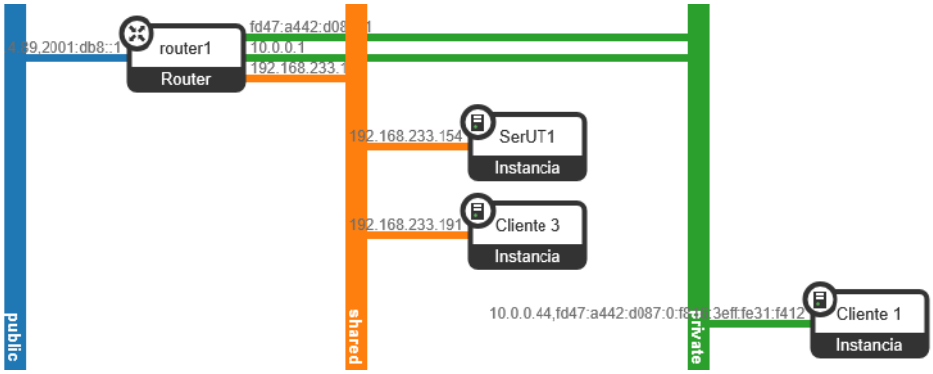
Recursos necesarios

- Dos instancias virtuales activas
- Herramientas de prueba de red (ping, netcat)
- Red virtual configurada

Procedimiento

1. Se verifica la conectividad entre las instancias virtuales.

Figura 79
Topología Planteada



Nota: De preferencia utilizar la red Shared

2. Se ejecutan pruebas de latencia y pérdida de paquetes.
3. Se registran los valores obtenidos.
4. Se comparan los resultados con valores de referencia.
5. Se identifican posibles mejoras en la configuración de red.

Resultados esperados

Se obtiene información cuantitativa sobre el rendimiento de la red, permitiendo detectar problemas de latencia o congestión.

Evaluación del Rendimiento de Red en Entornos IaaS

Prueba Realizada	Latencia (ms)	Pérdida de Paquetes (%)	Estado de la Red
Ping Cliente-Servidor			Estable / Inestable
Transferencia TCP			Óptimo / Deficiente
Transferencia UDP			Óptimo / Deficiente

Permite medir el desempeño de la red virtual y detectar problemas que afecten la calidad del servicio.

Conclusiones y Recomendaciones

Conclusiones

El análisis inicial de la infraestructura permitió identificar limitaciones relacionadas con la falta de control y priorización del tráfico, así como una gestión manual de recursos lo que evidenció la necesidad de una plataforma de orquestación como OpenStack para mejorar el rendimiento y la escalabilidad.

El diseño de una arquitectura de nodo único demostró que es posible implementar un entorno OpenStack funcional incluso con recursos limitados permitiendo la operación estable de instancias virtuales redes segmentadas y políticas de calidad de servicio.

La aplicación de una metodología en cascada facilitó la organización del proyecto mientras que las pruebas de rendimiento permitieron observar el impacto de la optimización sobre el comportamiento de la infraestructura y comprender el uso de los recursos de red.

El desarrollo de pruebas y guías prácticas permitió a los estudiantes interactuar directamente con un entorno IaaS fortaleciendo la comprensión de conceptos de optimización del rendimiento y fomentando un criterio técnico basado en el análisis de resultados.

Recomendaciones

- Se recomienda que en futuras implementaciones se realice un diagnóstico previo más detallado incluyendo métricas históricas de uso de recursos con el objetivo de facilitar la toma de decisiones técnicas y seleccionar configuraciones óptimas desde las etapas iniciales del diseño.
- Se recomienda evaluar en trabajos futuros la migración hacia arquitecturas multi-nodo con el fin de analizar escenarios de alta disponibilidad y tolerancia a fallos ampliando así el alcance del aprendizaje hacia entornos empresariales más complejos.
- Se recomienda complementar la metodología utilizada con enfoques iterativos o ágiles que permitan realizar ajustes más frecuentes durante la fase de pruebas especialmente en escenarios donde se evalúan múltiples configuraciones de red y tráfico.
- Se recomienda formalizar las guías de laboratorio desarrolladas como material de apoyo académico incorporando escenarios progresivos de complejidad que permitan a los estudiantes fortalecer sus competencias prácticas en redes virtuales y computación en la nube.

Bibliografía

AWS. (2023). *¿Qué es la IaaS? Explicación de la infraestructura como servicio: AWS*. https://aws.amazon.com/what-is/iaas/

Bakshi Akash. (2021). Introduction to virtualization and resource management in IaaS | CNCF. *Member Post*. https://www.cncf.io/blog/2021/04/16/introduction-to-virtualization-and-resource-management-in-iaas/

Cisco. (2014). *Enhanced Reader*. moz-extension://68f10084-afcc-485a-b905-21bc45a388d9/enhanced-reader.html?openApp&pdf=https%3A%2F%2Fwww.cisco.com%2Fc%2Fdam%2Fglobal%2Fes_es%2Fassets%2Fpdf%2Fnetworking_sdn_enhance_operator_monetization_wp.pdf

cloudification. (2024, May 16). *Introducing OpenStack*. The Leading Open Source Cloud • Cloudification - We Build Clouds. https://cloudification.io/cloud-blog/introducing-openstack-leading-open-source-cloud-platform/

Dautovic, E. (2023, May 2). *Deep dive into the main OpenStack components part 1 - Fairbanks*. https://fairbanks.nl/openstack-nova-neutron-and-cinder/

Díaz Sánchez, F. (2017). *Optimización de la ubicación de controladores en redes SDN*. https://repository.usta.edu.co/handle/11634/9950

Espino, L. (2009). *CLOUD COMPUTING COMO UNA RED DE SERVICIOS*.

Fiveable. (2024, August 9). *Integration of SDN with AI and machine learning | Software-Defined Networking Class Notes | Fiveable*. https://library.fiveable.me/software-defined-networking/unit-15/integration-sdn-ai-machine-learning/study-guide/2k3YP1A82PqytiSQ

Haff, G. (2017, December 17). *OpenStack: private cloud Infrastructure-as-a-Service (IaaS)*. https://www.redhat.com/en/blog/private-cloud-infrastructure-service-iaas

Haranas, M. (2023, August 29). *Google Cloud Next 2023: The 20 Biggest Product Launches | CRN*. Google Cloud Next 2023: The 20 Biggest Product Launches. https://www.crn.com/news/cloud/google-cloud-next-2023-the-20-biggest-product-launches

Hystax. (2023, December 18). *OpenStack versus VMware: cost efficiency showdown | Hystax*. OpenStack versus VMware: Cost Efficiency Showdown. https://hystax.com/openstack-versus-vmware-cost-efficiency-showdown/

IBM. (2024, October 23). *Cloud Infrastructure Solutions | IBM*. https://www.ibm.com/cloud/infrastructure

Kalaiselvan, K., & Venkatakrishna, P. (2013). Performance-based QoS for cloud's infrastructure as a service. *International Journal of Cloud Computing*, *2*(4), 325. https://doi.org/10.1504/IJCC.2013.058096

Khorolets, A. (2024). *Top Hypervisors Compared: VMware, Hyper-V, Azure Stack HCI, KVM*. *1*. https://www.starwindsoftware.com/blog/top-enterprise-hypervisors-compared-vmware-hyper-v-azure-stack-hci-kvm/

King, B. (2021, November 5). *What is IaaS? Infrastructure as a Service Explained | DigitalOcean*. What Is IaaS? Infrastructure as a Service Explained. https://www.digitalocean.com/blog/what-is-iaas

Li, X., Samaka, M., Chan, H. A., Bhamare, D., Gupta, L., Guo, C., & Jain, R. (2017). *Network Slicing for 5G: Challenges and Opportunities*.

Lluco, L., Javier Tutor, J., & Barquero, G. (2023). *Evaluación y Comparativa de controladores SDN para despliegues de redes 5G*. https://riunet.upv.es/handle/10251/197672

Madni, S. H. H., Latiff, M. S. A., Coulibaly, Y., & Abdulhamid, S. M. (2016). Resource scheduling for infrastructure as a service (IaaS) in cloud computing: Challenges

and opportunities. *Journal of Network and Computer Applications*, *68*, 173–200. https://doi.org/10.1016/J.JNCA.2016.04.016

Malkawi, M. I. (2013). The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP). *Human-Centric Computing and Information Sciences*, *3*(1), 1–17. https://doi.org/10.1186/2192-1962-3-22/TABLES/2

Mestas Yucra, E. E. (2015). Modelo Basado en Tecnología de Cloud Computing para Ofrecer Servicio de Infraestructura (IaaS) en el Centro de Computo e Informática de la Universidad Nacional del Altiplano 2014. *Universidad Nacional Del Altiplano*. https://renati.sunedu.gob.pe/handle/sunedu/3577898

Microsoft. (2023). *Optimización del rendimiento de redes definidas por software | Microsoft Learn*. https://learn.microsoft.com/es-es/windows-server/administration/performance-tuning/subsystem/software-defined-networking/

NAKIVO. (2024). AWS vs Azure vs Google Cloud: Comparación de plataformas. *NAVICO*, *1*. https://www.nakivo.com/es/blog/aws-vs-azure-vs-google-cloud-comparison/

openstack. (2024, October 23). *Open Source Cloud Computing Infrastructure - OpenStack*. https://www.openstack.org/

openstack r. (2013, February 1). *OpenStack Operations Guide* . https://docs.openstack.org/operations-guide/

Paternó, G. (2015, November 30). *OpenStack - Nova and Glance*. OpenStack - Nova and Glance. https://www.guruadvisor.net/en/cloud/118-openstack-nova-and-glance

Pure Storage. (2024, October 18). *VMware vs. OpenStack* . Pure Storage Blog. https://blog.purestorage.com/purely-educational/vmware-vs-openstack/

Red Hat. (2023, August 4). *iaas_focus-paas-saas-diagram-1200x1046.png (Imagen PNG, 1638 × 1046 píxeles) - Escalado (69 %)*. ¿Qué Es La IaaS? https://www.redhat.com/rhdc/managed-files/iaas_focus-paas-saas-diagram-1200x1046.png

Reyes, L. (2024). *Calidad de servicio (QOS)*. https://saberpunto.com/tecnologia/calidad-de-servicio-qos/#google_vignette

Ruiz Quispe, B. V. (2016). *Implementación de Prototipo de Tecnología de Cloud Computing para Servicios de Infraestructura (Iaas) en la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato*. https://repositorio.uta.edu.ec:8443/jspui/handle/123456789/23660

Salazar, G. (2016, September 30). *Fundamentos de QoS - Calidad de Servicio en Capa 2 y Capa 3 - Cisco Community*. https://community.cisco.com/t5/blogs-routing-y-switching/fundamentos-de-qos-calidad-de-servicio-en-capa-2-y-capa-3/ba-p/3103715

Slingerland, C. (2024). *9 VMware Alternatives To Consider In 2024*. https://www.cloudzero.com/blog/vmware-alternatives/

Taylor, A. (2023, December 6). *Microsoft named a Leader in 2023 Gartner® Magic QuadrantTM for Strategic Cloud Platform Services (SCPS) | Microsoft Azure Blog*. https://azure.microsoft.com/en-us/blog/microsoft-named-a-leader-in-2023-gartner-magic-quadrant-for-strategic-cloud-platform-services-scps/

Team EMB. (2023, November 27). *Cloud Computing's Backbone: Infrastructure as a Service (IaaS)*. Cloud Computing's Backbone: Infrastructure as a Service (IaaS). https://blog.emb.global/cloud-computings-backbone-iaas/

Vijay K, K. (2021, April 29). *A brief overview of the Container Network Interface (CNI) in Kubernetes*. https://www.redhat.com/en/blog/cni-kubernetes

VM. (2024, October 23). *VMware Named a Leader in the 2023 Gartner Magic Quadrant for Distributed Hybrid Infrastructure : @VMblog*. https://vmblog.com/archive/2023/10/19/vmware-named-a-leader-in-the-2023-gartner-magic-quadrant-for-distributed-hybrid-infrastructure.aspx

Wawira, M., & Hougen, A. (2024, October 17). *11 Popular Cloud Computing Platforms Compared in 2024*. https://www.cloudwards.net/cloud-computing-platforms/

Worldstream. (2023, June 21). *These 5 Industry Trends are Driving SDN Adoption* . Worldstream. https://www.worldstream.com/en/news/these-5-industry-trends-are-driving-sdn-adoption

ANEXOS

Contenido del scrit de instalación rápida: instalacion.sh

- #!/usr/bin/env bash
- set -e

- export DEBIAN_FRONTEND=noninteractive

- echo "[+] Actualizando el sistema y paquetes..."
- sudo systemctl stop ufw || true
- sudo systemctl disable ufw || true
- sudo apt update && sudo apt -y upgrade

- echo "[+] Instalando dependencias básicas para DevStack..."
- sudo apt install -y git vim net-tools \
- python3-pip python3-dev build-essential \
- libssl-dev libffi-dev libxml2-dev libxslt1-dev zlib1g-dev \
- libmysqlclient-dev libpq-dev curl openvswitch-switch

- echo "[+] Creando usuario stack..."
- if ! id "stack" &>/dev/null; then
- sudo useradd -s /bin/bash -d /opt/stack -m stack
- echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
- sudo chmod 0440 /etc/sudoers.d/stack
- fi

- echo "[+] Asegurando que el home de stack exista..."
- sudo mkdir -p /opt/stack
- sudo chown -R stack:stack /opt/stack

- sudo apt-get -y purge cloud-init || true
- sudo netplan apply

- echo "[+] Esperando 5 segundos para asegurar que la red esté lista..."
- sleep 5

- echo "[+] Clonando DevStack (rama estable 2024.1) en /opt/stack/devstack..."
- sudo -u stack bash -c "
- cd /opt/stack
- for i in {1..3}; do
  - git clone --progress https://opendev.org/openstack/devstack devstack && break
  - echo 'Intento \$i fallido, esperando 5s antes de reintentar...'
  - sleep 5
- done
- "
- sudo chown -R stack:stack /opt/stack/devstack

- echo "[+] Creando local.conf adaptado a enp0s3 (192.168.1.15/24)..."
- sudo -u stack tee /opt/stack/devstack/local.conf > /dev/null <<'EOF'

- [[local|localrc]]
- # ===== BEGIN localrc =====
- DATABASE_PASSWORD=password
- ADMIN_PASSWORD=password
- SERVICE_PASSWORD=password
- SERVICE_TOKEN=password
- RABBIT_PASSWORD=password

- GIT_BASE=https://opendev.org

- # Optional settings:
- # OCTAVIA_AMP_BASE_OS=centos
- # OCTAVIA_AMP_DISTRIBUTION_RELEASE_ID=9-stream
- # OCTAVIA_AMP_IMAGE_SIZE=3
- # OCTAVIA_LB_TOPOLOGY=ACTIVE_STANDBY
- # OCTAVIA_ENABLE_AMPHORAV2_JOBBOARD=True
- # LIBS_FROM_GIT+=octavia-lib,
- # Enable Logging
- LOGFILE=$DEST/logs/stack.sh.log
- VERBOSE=True
- LOG_COLOR=True
- enable_service rabbit
- enable_plugin neutron $GIT_BASE/openstack/neutron
- # Octavia supports using QoS policies on the VIP port:
- enable_service q-qos
- enable_service placement-api placement-client
- # Octavia services
- enable_plugin octavia $GIT_BASE/openstack/octavia master
- enable_plugin octavia-dashboard $GIT_BASE/openstack/octavia-dashboard
- enable_plugin ovn-octavia-provider $GIT_BASE/openstack/ovn-octavia-provider
- enable_plugin octavia-tempest-plugin $GIT_BASE/openstack/octavia-tempest-plugin
- enable_service octavia o-api o-cw o-hm o-hk o-da
- # If you are enabling barbican for TLS offload in Octavia, include it here.
- Q_AGENT=ovn
- Q_ML2_PLUGIN_MECHANISM_DRIVERS=ovn
- Q_ML2_TENANT_NETWORK_TYPES=geneve
- Q_ML2_PLUGIN_TYPE_DRIVERS=flat,geneve,vlan,local
- ML2_L3_PLUGIN=ovn-router

- # Tunnel y bridge mapping
- OVN_TUNNEL_TYPE=geneve
- OVN_PHYSICAL_BRIDGE=br-ex
- OVN_BRIDGE_MAPPINGS=public:br-ex
- # enable_plugin barbican $GIT_BASE/openstack/barbican
- # enable_service barbican
- # Cinder (optional)
- disable_service c-api c-vol c-sch
- # Tempest
- enable_service tempest
- # ===== END localrc =====
- EOF

- echo "[+] Preparando script post-stack para cargar Debian Cloud con contraseña habilitada..."
- sudo -u stack tee /opt/stack/devstack/load_debian_image.sh > /dev/null <<'EOS'
- #!/usr/bin/env bash
- set -e
- echo "[+] Descargando imagen Debian 12 Cloud..."
- mkdir -p ~/images
- cd ~/images
- curl -LO https://cloud.debian.org/images/cloud/bookworm/latest/debian-12-genericcloud-amd64.qcow2

- echo "[+] Cargando imagen en Glance..."
- source /opt/stack/devstack/openrc admin admin
- openstack image create "debian-12-cloud" \
- --disk-format qcow2 \

- --container-format bare \
- --public \
- --file debian-12-genericcloud-amd64.qcow2

- echo "[+] Creando flavor m1.small si no existe..."
- openstack flavor show m1.small || openstack flavor create m1.small --ram 2048 --disk 20 --vcpus 2

- echo "[+] Creando keypair temporal..."
- test -f ~/.ssh/id_rsa || ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
- openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey || true

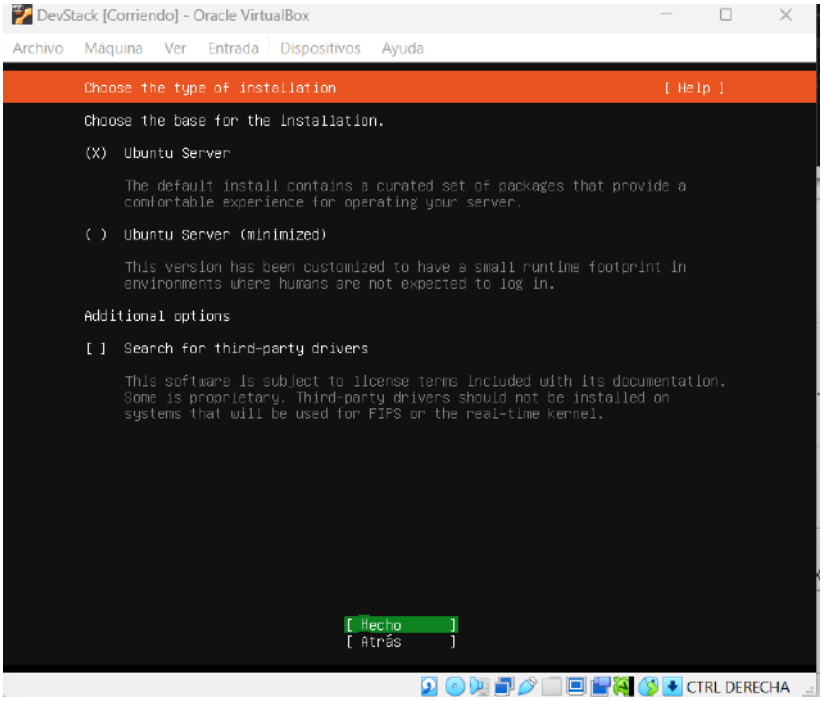
- echo "[+] Listo. Para lanzar Debian con login por contraseña usa:"
- echo "openstack server create --flavor m1.small --image debian-12-cloud --network public --key-name mykey --user-data <(cat <<EOF
- #cloud-config
- password: debian123
- chpasswd: { expire: False }
- ssh_pwauth: True
- EOF
-) debian-test"
- EOS
- sudo chmod +x /opt/stack/devstack/
- sudo chmod +x /opt/stack
- sudo chmod +x /opt/stack/devstack/load_debian_image.sh

- echo "[+] Preparación finalizada ☑"
- echo "Ahora inicia sesión como usuario stack y ejecuta:"
- echo " cd /opt/stack/devstack && ./stack.sh"
- echo "Cuando termine, carga la imagen Debian con:"
- echo " /opt/stack/devstack/load_debian_image.sh"

Configuración de la MV

Figura 80

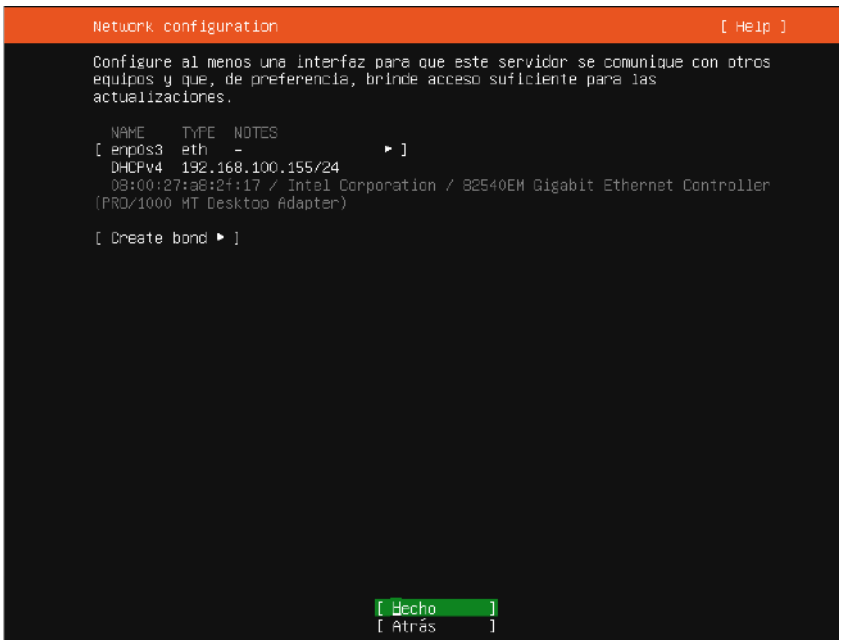
Instalación del SO Ubuntu Server 22.04.5 LTS



Nota: Opción de server

Figura 81

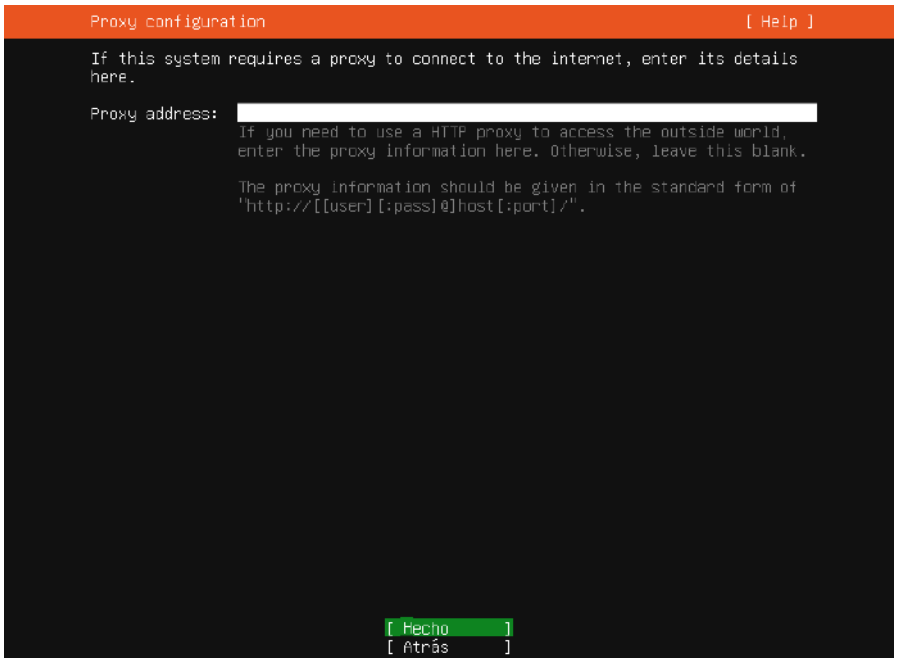
Instalación del SO Ubuntu Server 22.04.5 LTS - Configuración de red.



Nota: Poner una IP fija, siempre en el mismo rango que nos de si es puente o Nat

Figura 82

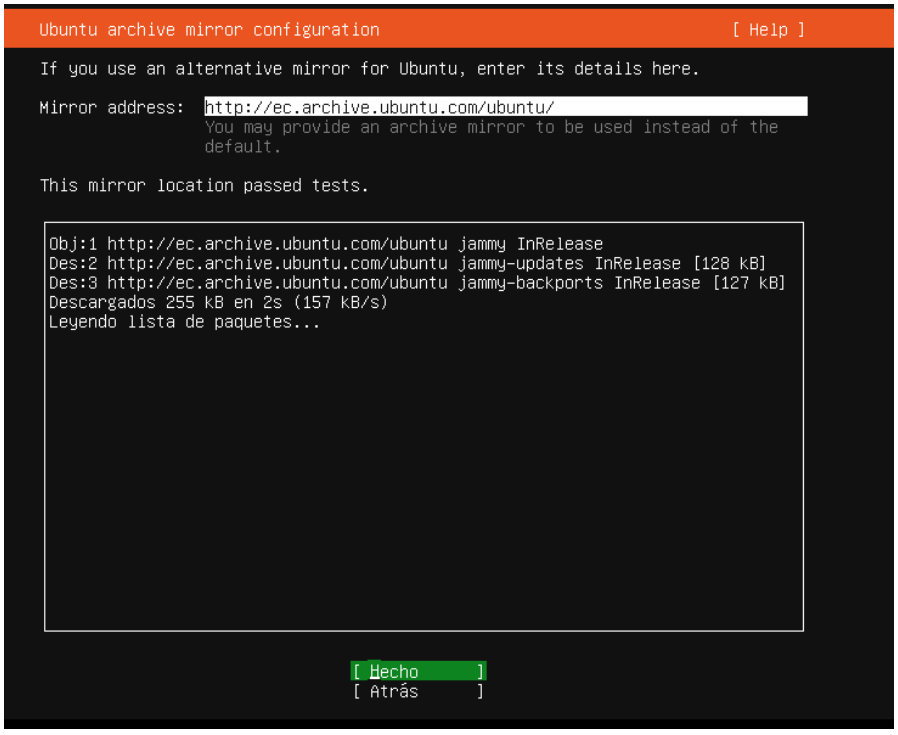
Instalación del SO Ubuntu Server 22.04.5 LTS - Configuración de Proxy



Nota: De preferencia no poner nada de proxy

Figura 83

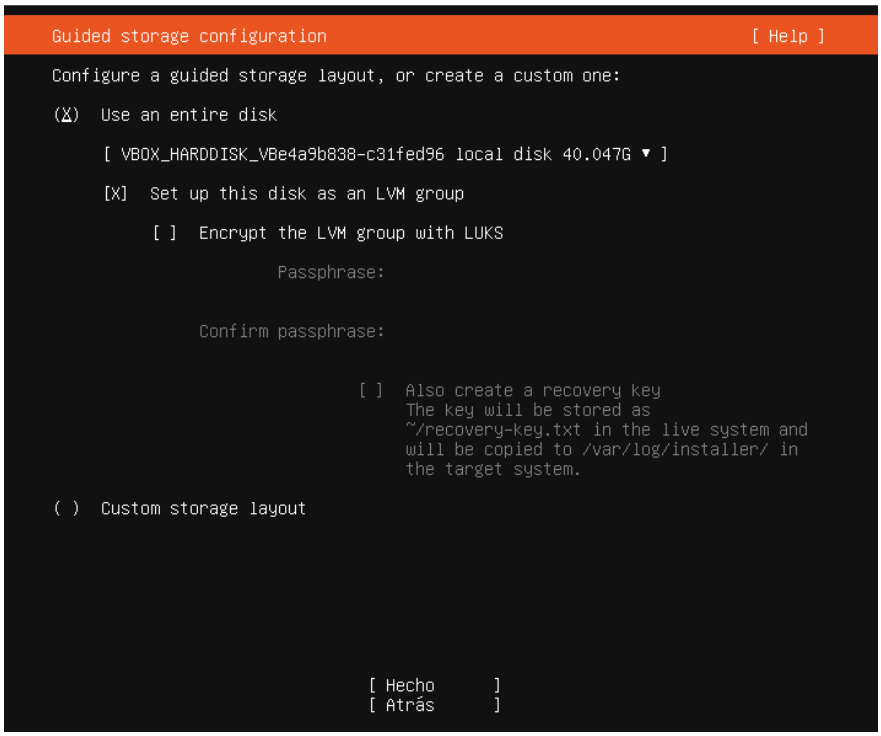
Instalación del SO Ubuntu Server 22.04.5 LTS - Descarga de repositorios



Nota: Esta parte es muy fundamental, toca que se descargue una versión estable

Figura 84

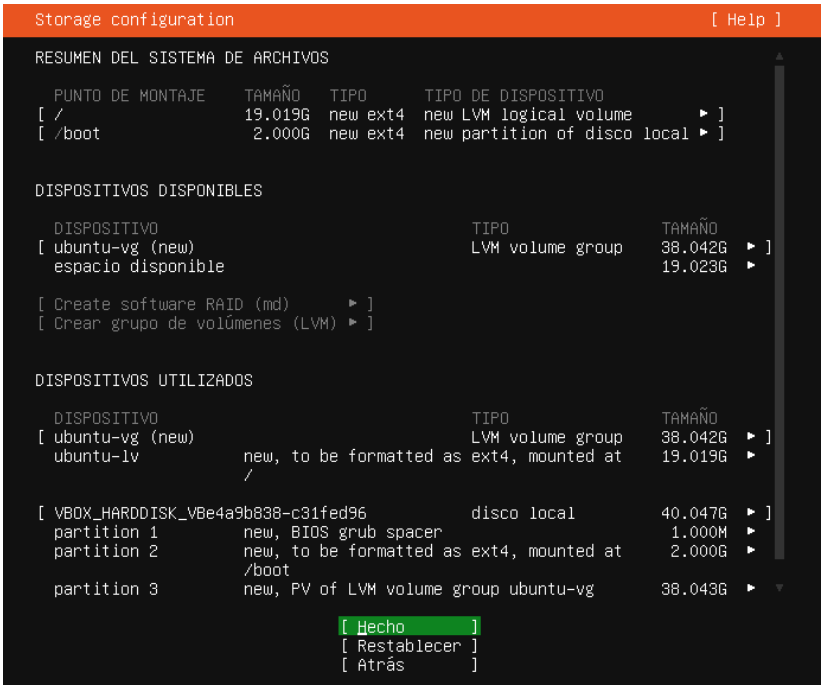
Instalación del SO Ubuntu Server 22.04.5 LTS - 1



Nota: En esta parte solo ponemos siguiente

Figura 85

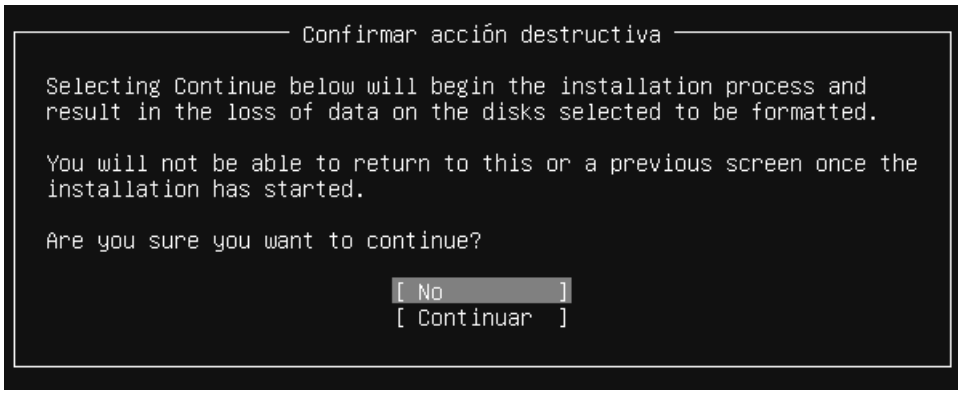
Instalación del SO Ubuntu Server 22.04.5 LTS – Asignación de Espacios



Nota: Al ser Devstack no se necesita crear discos específicos o espacios

Figura 86

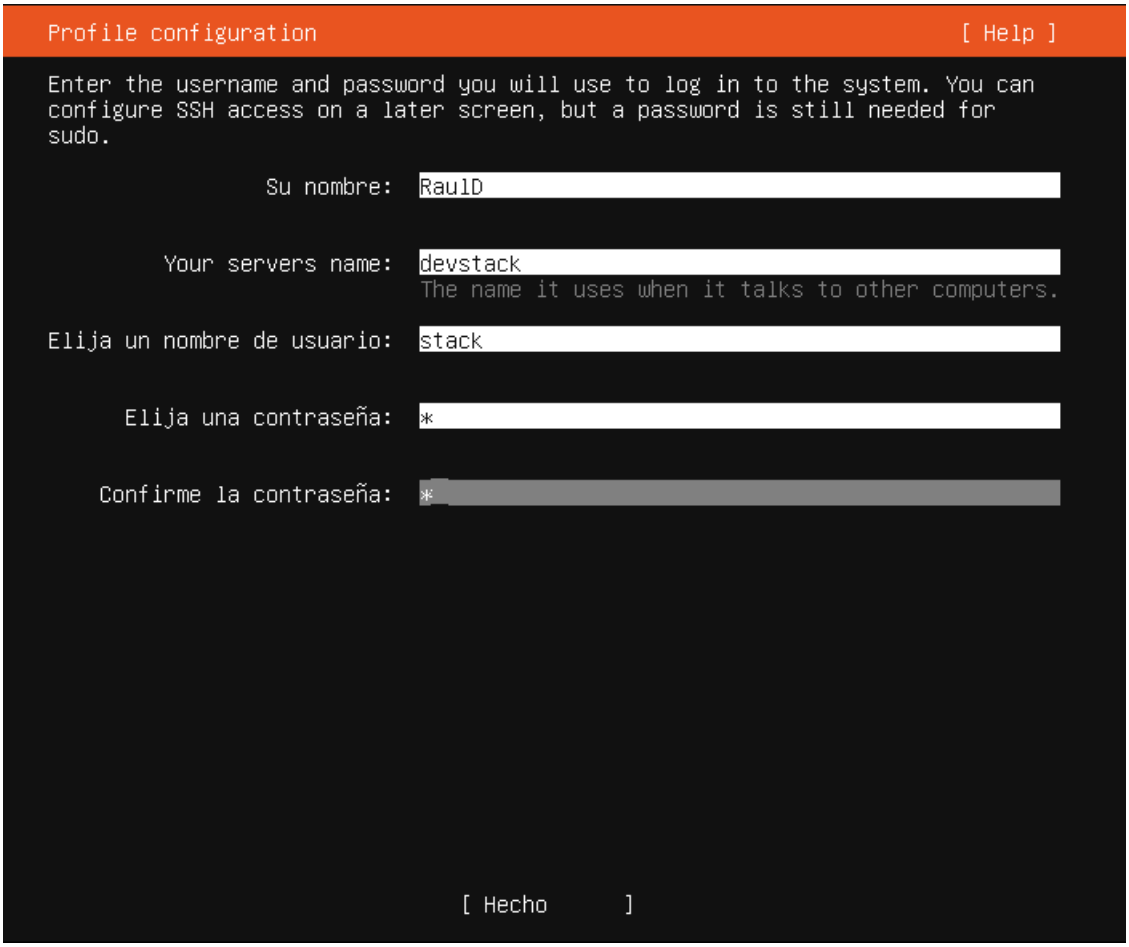
Instalación del SO Ubuntu Server 22.04.5 LTS - 2



Nota: Ponemos Continuar.

Figura 87

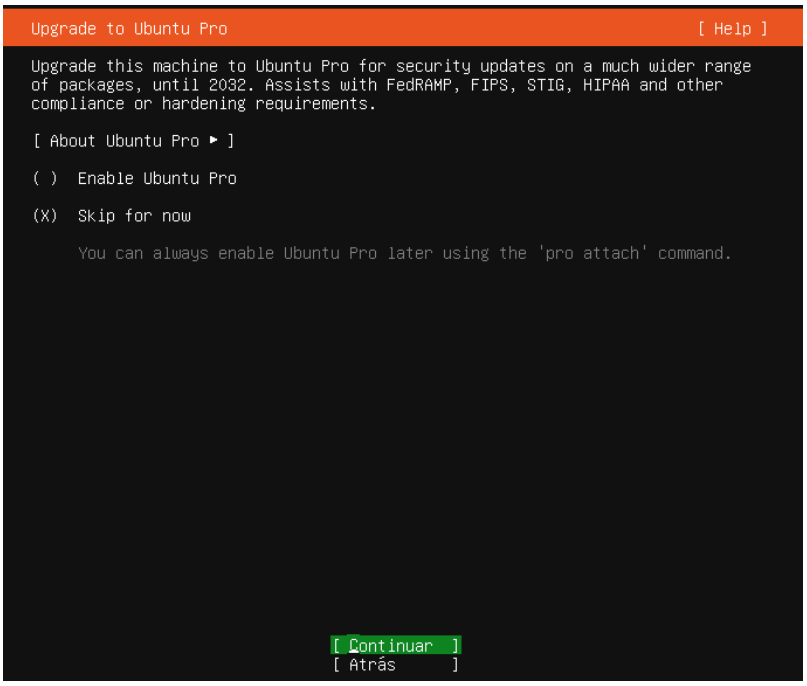
Instalación del SO Ubuntu Server 22.04.5 LTS – Creación de Usuarios



Nota: Existe dos opciones, podemos o no crear el usuario Stack, de preferencia no, ya que después lo crearemos con los permisos necesarios.

Figura 88

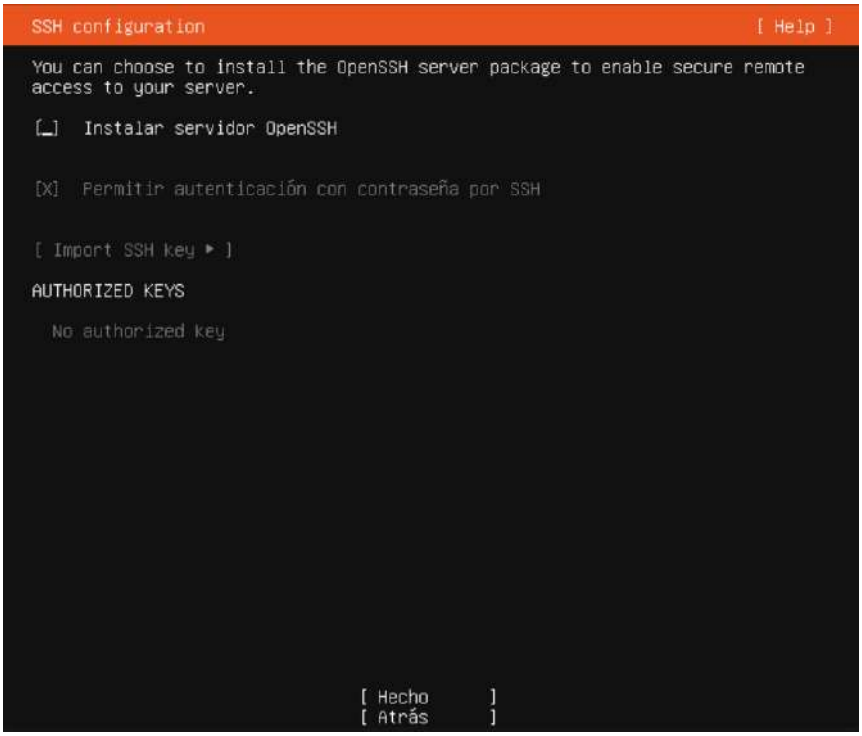
Instalación del SO Ubuntu Server 22.04.5 LTS - 3



Nota: Ponemos continuar

Figura 89

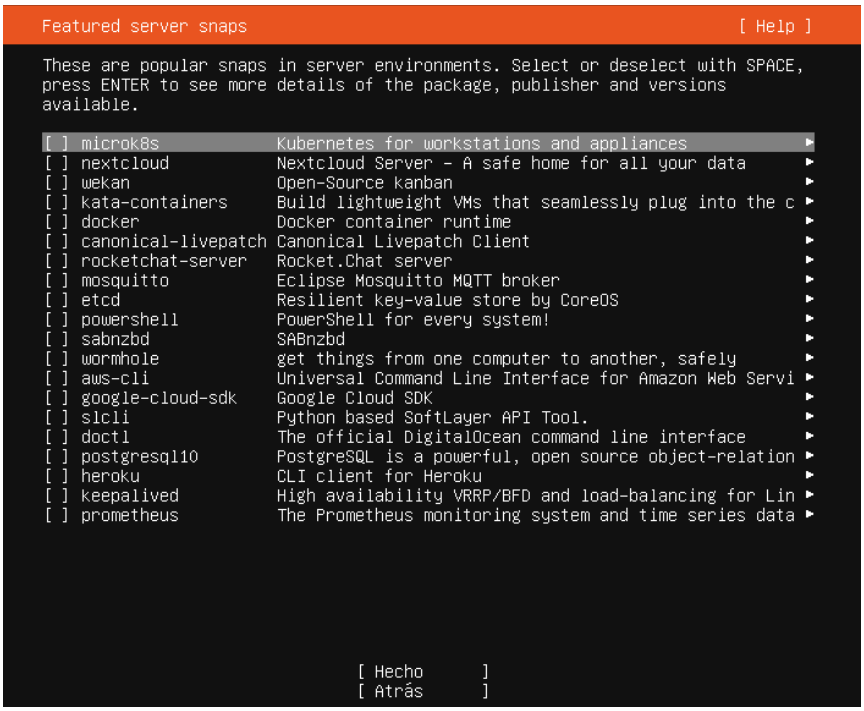
Instalación del SO Ubuntu Server 22.04.5 LTS – Instalación de servicio SSH



Nota: Siempre poner servidor SSH, ya que este será nuestro medio directo de conexión, entre la MV y el host físico.

Figura 90

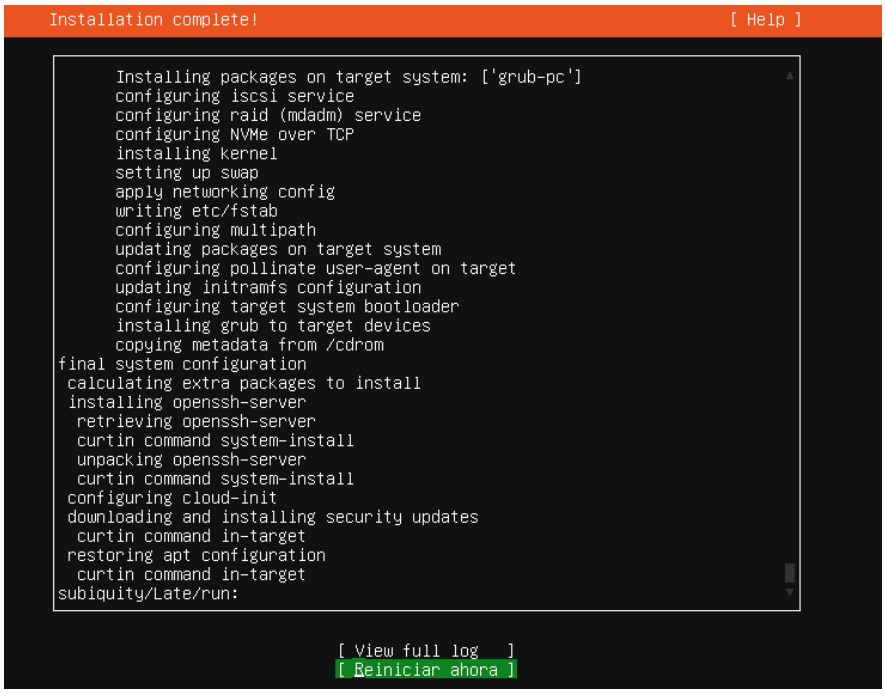
Instalación del SO Ubuntu Server 22.04.5 LTS



Nota: Se puede instalar herramientas complementarias, pero para no consumir recursos, solo instalaremos mediante vayamos utilizando, debido a las versiones.

Figura 91

Instalación del SO Ubuntu Server 22.04.5 LTS – Fin de instalación



Nota: Este proceso puede tardar, dependiendo de los recursos que tengan la MV

Figura 92

Instalación del SO Ubuntu Server 22.04.5 LTS – Reinicio necesario

Nota: Tal como nos indica en mensaje dar ENTER, para reiniciar la MV y ponernos en marcha