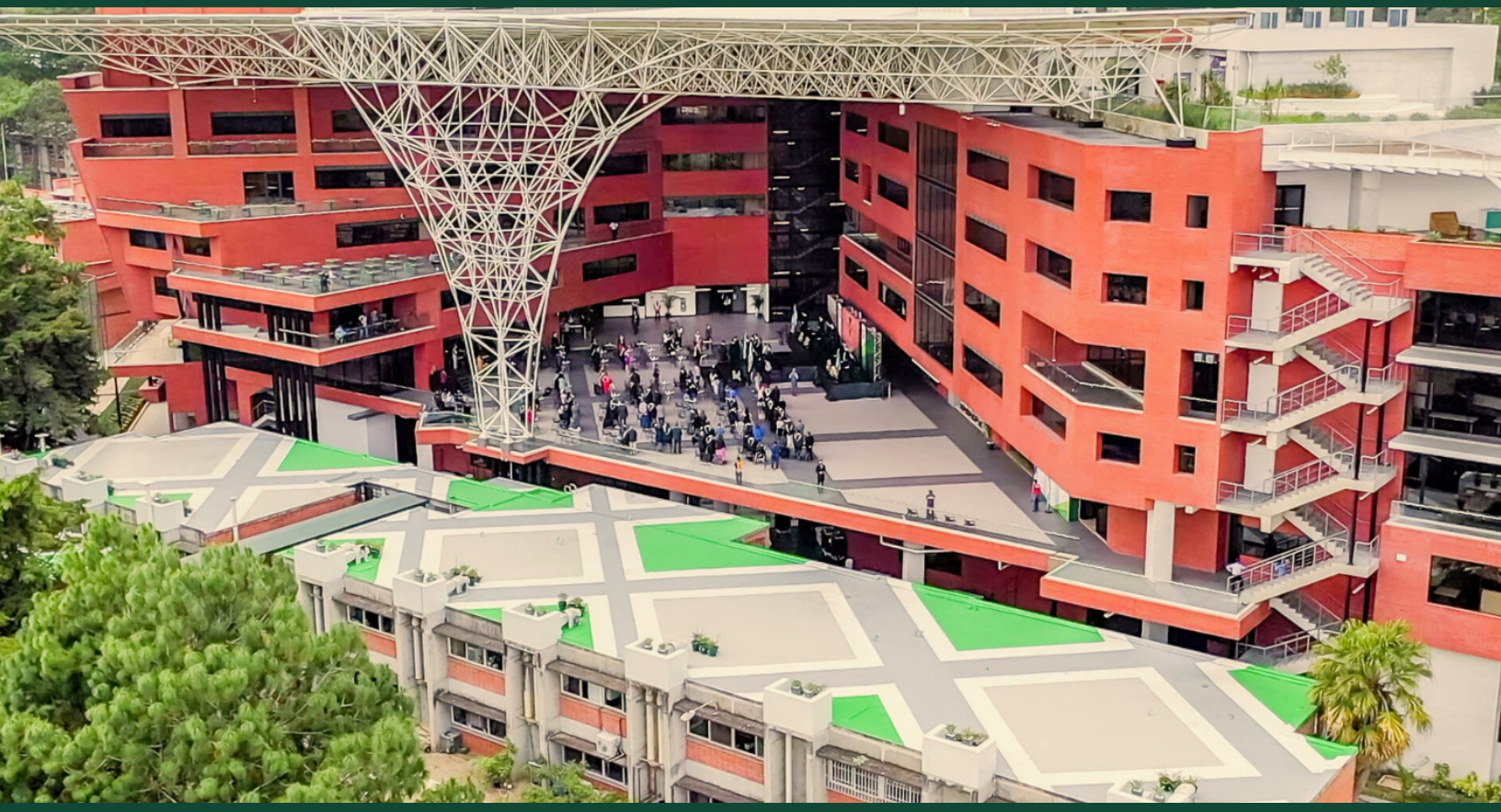
Automatización de despliegue de plataforma de *cloud computing* Openstack: una estrategia escalable empleando infraestructura como código

Estuardo Alejandro Castillo García



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Automatización de despliegue de plataforma de *cloud computing* Openstack: una estrategia escalable empleando infraestructura como código

Trabajo de graduación presentado por Estuardo Alejandro Castillo García para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

Vo.Bo.:

(f) 

M.Sc. Jonathan de los Santos

(f) 

M.Sc. Carlos Esquit

Fecha de aprobación: Guatemala, 20 de noviembre de 2025.

Prefacio

El presente trabajo de graduación representa la culminación de un proceso de aprendizaje y dedicación que ha abarcado varios años de formación académica en la Universidad del Valle de Guatemala. Todo este esfuerzo no habría sido posible sin el apoyo incondicional de diversas personas a lo largo de este camino. Quiero expresar mi más profundo agradecimiento a mi madre, Ruth, cuyo apoyo incondicional impulsó mi perseverancia y dedicación en cada etapa de mi formación. Su sacrificio y confianza en mis capacidades han sido una fuente constante de motivación.

Agradezco a mi asesor de tesis, el ingeniero Jonathan de los Santos, que con su guía en aspectos técnicos y de metodología, me ayudaron a comprender de forma profunda los conceptos y prácticas necesarios para llevar a cabo este proyecto con éxito. Agradezco al equipo de profesores de la Universidad del Valle de Guatemala, quienes compartieron sus conocimientos y experiencias, enriqueciendo mi formación académica y profesional. Al equipo técnico que brindó soporte durante la implementación del proyecto, su ayuda fue invaluable para tener acceso a los recursos necesarios y resolver problemas técnicos que surgieron durante el desarrollo del trabajo. Finalmente, agradezco a mis compañeros de clase por su apoyo y colaboración durante este proceso. Las discusiones y el intercambio de ideas con ellos han sido fundamentales para enriquecer mi perspectiva y mejorar la calidad de este trabajo.

Estuardo Alejandro Castillo García
Guatemala, 2025

Índice general

Índice de figuras

Índice de cuadros

Resumen

La creciente necesidad de entrega de contenido y servicios en la nube ha impulsado la adopción de infraestructuras más flexibles, escalables y orientadas a la automatización. La gestión de recursos en estos entornos se ha visto limitada por la complejidad operacional inherente a la configuración, despliegue y mantenimiento del plano de control y de los servicios distribuidos. En el presente trabajo se propone una estrategia para agilizar el despliegue de infraestructuras en la nube mediante el uso de herramientas de automatización y contenedorización. Se implementa una solución basada en Openstack, una plataforma de computación en la nube de código abierto, utilizando Kolla-Ansible para la orquestación, gestión de contenedores y provisión automatizada de servicios. La metodología empleada incluye la configuración de un entorno capaz de aprovechar los recursos físicos disponibles en la red de laboratorio, implicando la definición de los planos de red y de almacenamiento, la gestión de las librerías y dependencias necesarias para los servicios del plano de control, y la habilitación de la comunicación segura entre los nodos. Posteriormente, se lleva a cabo la automatización del despliegue mediante *playbooks* de Ansible, seguida de la validación integral del funcionamiento y la interoperabilidad del entorno de nube. Los resultados demuestran que la automatización mediante Kolla-Ansible reduce significativamente la complejidad administrativa y la intervención manual, sin comprometer la funcionalidad del sistema e incluso mejorando la visibilidad y comprensión de las interdependencias entre los servicios que interactúan. Al emplear Kolla-Ansible para el despliegue del entorno de nube, los operadores pueden ofrecer a los usuarios finales una plataforma completa para la prestación de soluciones basadas en infraestructura como servicio. Se concluye que la adopción de infraestructuras como código constituye una estrategia efectiva para optimizar la gestión de entornos de nube, facilitando su escalabilidad y capacidad de adaptación frente a entornos dinámicos.

Palabras clave: infraestructura como código, Openstack, Kolla-Ansible, virtualización, contenedores.

Abstract

The growing need for delivering cloud-based content and services has driven the adoption of more flexible, scalable, and automation-oriented infrastructures. Resource management in these environments has been constrained by the operational complexity inherent to the configuration, deployment, and maintenance of the control plane and distributed services. This work proposes a strategy to streamline cloud infrastructure deployment through the use of automation and containerization tools. A solution based on Openstack, an open-source cloud computing platform, is implemented using Kolla-Ansible for container orchestration, management, and automated service provisioning. The methodology includes configuring an environment capable of leveraging the physical resources available in the laboratory network, involving the definition of network and storage planes, the management of libraries and dependencies required by control-plane services, and the enablement of secure communication between nodes. Subsequently, the infrastructure deployment is automated through Ansible playbooks, followed by a comprehensive validation of the cloud environment's operation and interoperability. The results demonstrate that automation through Kolla-Ansible significantly reduces administrative complexity and manual intervention, without compromising system functionality and even improving the visibility and understanding of service interdependencies. By employing Kolla-Ansible for cloud deployment, operators can provide end users with a complete platform for delivering Infrastructure-as-a-Service solutions. It is concluded that adopting infrastructure as code constitutes an effective strategy for optimizing cloud-environment management, enhancing scalability and adaptability in dynamic environments.

Keywords: infrastructure as code, Openstack, Kolla-Ansible, virtualization, containers.

CAPÍTULO 1

Introducción

El sector de tecnologías en la nube ha experimentado un crecimiento significativo en los últimos años, impulsado por la creciente demanda de servicios digitales y la necesidad de infraestructuras flexibles y escalables. En este contexto, la adopción de prácticas como la infraestructura como código (IaC) se ha convertido en una estrategia clave para optimizar la gestión y el aprovisionamiento de recursos en entornos de nube. Dentro de este marco, la adopción de herramientas que faciliten la automatización y la estandarización de procesos resulta fundamental para garantizar la entrega de capacidades a usuarios finales en instituciones educativas.

La finalidad de este trabajo es implementar una estrategia para gestionar infraestructura de nube privada con mínima intervención humana, brindando al Departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala una solución que permita el aprovisionamiento automático de recursos en la red de laboratorios del campus. La propuesta se basa en el uso de la herramienta moderna de gestión de infraestructura Ansible para desplegar y administrar una nube privada basada en Openstack. Esta solución busca no solo mejorar la eficiencia operativa, sino también proporcionar un entorno escalable y adaptable a las necesidades cambiantes de los usuarios y proyectos académicos. El método empleado para llevar a cabo el despliegue de la nube abarca la preparación del entorno para asociar equipos físicos a elementos que pueden ser abstraídos en capas de *software*, la configuración de la herramienta Ansible para gestionar la infraestructura como código, y la implementación de Openstack como plataforma de nube privada. Se desarrolló la modificación de archivos de configuración relevantes para adaptar la solución a las características específicas del entorno de la universidad, asegurando así una integración fluida con la infraestructura existente.

Se obtiene como principal resultado una plataforma de nube lista para ser operada por administradores y usuarios finales, con la capacidad de aprovisionar recursos de manera automática y eficiente. Además la solución implementada permite la escalabilidad y capacidad de despliegue rápido de nuevos recursos según las necesidades de usuarios finales. El uso de la herramienta permite explotar el uso de recursos de cómputo que de alguna forma se ven subutilizados por la complejidad operativa que conlleva su administración manual.

CAPÍTULO 2

Antecedentes

La utilización de herramientas para despliegue de nubes privadas no es un concepto nuevo para el ambiente de cómputo empleado en la Universidad del Valle de Guatemala. Se tiene el precedente de casos de éxito en dónde se ha explorado el despliegue modular de Openstack [1] para aprovisionar máquinas virtuales en la red privada de laboratorios. En la prueba de concepto se emplean las conexiones a interfaz de aplicación de los diversos proyectos para poder lograr el despliegue de los servicios básicos. En el trabajo se presentan las consideraciones necesarias para poder configurar el ambiente usando un servidor Linux, aprovechando recursos de cómputo subutilizados en la red de laboratorios. La construcción de módulos de la plataforma se realiza sin *frameworks* para construir la infraestructura como código (IaC). Otros trabajos exploran el uso de herramientas de infraestructura como código (como Ansible) para poder automatizar procesos y asegurar la repetibilidad en tareas que son susceptibles al error humano. En el trabajo desarrollado sobre la red de laboratorios de La Universidad de Brno en Chequia [2] se ha demostrado mediante el uso de *playbooks* la forma en que pueden ser abstraídas tareas que hacen uso de herramientas en el sistema operativo para aprovisionar servicios y aplicaciones, brindando capacidades a los usuarios que hacen uso de los equipos físicos o virtualizados en infraestructuras de red locales y públicas. El escenario planteado en el trabajo presenta cómo puede emplearse la herramienta para la orquestación de diversas estaciones de trabajo, independientemente del sistema operativo que empleen, utilizando distribuciones populares de Linux y versiones recientes de Windows.

En ambos casos se presentan escenarios de despliegue que se realizan en ambientes de laboratorio, donde el uso de herramientas de automatización y despliegue de infraestructura son utilizados para asegurar la consistencia y repetibilidad de los procesos. Un caso particular fue explorado en el ambiente de supercomputación de TianHe, donde se emplea un *deployment* inteligente de registros distribuidos (IDRD) basado en la nube de Openstack [3]. La arquitectura de IDRD está basada en Kolla-Ansible, proveyendo capacidad a los usuarios dentro del ambiente de laboratorios para aprovisionar recursos de cómputo y almacenamiento de forma rápida y eficiente. El trabajo presenta el uso de herramientas para despliegue contenerizado en la orquestación de los servicios y la automatización del despliegue de los mismos, asegurando la consistencia en el aprovisionamiento de recursos. Con esto se presenta

un caso de éxito en el empleo de herramientas de desarrollo y operaciones para el despliegue del servicio de Openstack en ambientes listos para producción y deja como resultado el uso de tecnologías de contenedores con excelentes características de escalabilidad y portabilidad para aplicaciones en escenarios reales y demandantes.

CAPÍTULO 3

Justificación

La adopción de tecnologías en la nube es una tendencia en constante crecimiento. A medida que más dispositivos requieren conexión a la red, el aprovisionamiento de infraestructura capaz de satisfacer la alta demanda de solicitudes se vuelve crucial. Surge así la necesidad de herramientas que aseguren la estabilidad, la repetibilidad y que permitan acortar los tiempos de desarrollo. La tendencia actual es abstraer funciones que tradicionalmente se realizaban con equipos especializados y basar las soluciones en *software*, con el objetivo de reducir el espacio físico, el tiempo y el esfuerzo humano requeridos. Además, una utilización eficiente de los recursos disponibles podría representar beneficios económicos para los proveedores de servicios [4].

Sin embargo, muchas organizaciones aún enfrentan el reto de montar la infraestructura una sola vez, sin considerar el desarrollo continuo e integración a futuro. Esta práctica puede generar problemas de escalabilidad, mantenimiento y actualización, ya que cualquier cambio o ampliación requiere intervención manual significativa. La falta de automatización limita la capacidad de adaptación ante nuevas demandas y puede incrementar los costos operativos a largo plazo. Por ello, es fundamental justificar la inversión en herramientas y procesos que permitan automatizar el aprovisionamiento y la gestión de la infraestructura, asegurando así la sostenibilidad y eficiencia de las operaciones.

En este contexto, la construcción de arquitecturas basadas en *software* se vuelve esencial para garantizar la repetibilidad de los entornos y minimizar la intervención humana. El uso de servicios *agentless*, que no requieren la instalación adicional en los sistemas gestionados, facilita la administración remota y reduce la complejidad operativa [5]. Herramientas de infraestructura como código, como Ansible, permiten definir y desplegar configuraciones de manera automatizada, asegurando que los entornos sean consistentes y fácilmente replicables independientemente de la distribución de sistema operativo que se disponga en el ambiente. Estas soluciones contribuyen a una gestión más eficiente de los recursos, mejorando la capacidad de respuesta ante cambios y optimizando el uso del tiempo y esfuerzo del personal técnico.

En este trabajo se introduce una plataforma para el manejo de ambientes de desarrollo

completamente gestionables y escalables, adaptada a las necesidades de la red de laboratorios en campus. La solución propuesta está diseñada para operar con mínima intervención de los administradores locales, permitiendo el aprovisionamiento de recursos según la demanda y facilitando la administración a largo plazo mediante la automatización y el uso de herramientas modernas de gestión de infraestructura.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Automatizar el despliegue y configuración de nodos de Openstack mediante *scripts* de Ansible, garantizando una implementación consistente, rápida y escalable de la infraestructura de nube privada.

4.2. Objetivos específicos

- Automatizar la instalación y configuración de los servicios de Openstack necesarios para despliegue de máquinas virtuales utilizando Ansible.
- Implementar la exposición de *endpoints* de Openstack en base a la configuración de inventarios de Ansible.
- Instalar archivos de inventario para *hosts* locales, permitiendo la gestión centralizada de los nodos de Openstack en un solo servidor *bare-metal*.
- Modificar archivos de inventario de Ansible para la configuración de distintos nodos de Openstack en diferentes servidores *bare-metal*.
- Emplear Ansible para la generación de claves para la autenticación segura entre nodos y servicios de Openstack.
- Aprovisionar y configurar servicio para gestión Openstack *service* Horizon.
- Configurar *playbooks* de Ansible para mantenimiento, chequeo de contenedores y actualización de sus imágenes.

CAPÍTULO 5

Alcance

El proyecto consiste en desplegar plataforma de *cloud computing* Openstack aprovechando la infraestructura como código con la herramienta de automatización Ansible sobre un *cluster* de 2 HPC que cuentan con los siguientes recursos: Servidor Dell Precision 7920 (88 vCPU, 376GB RAM, 12 TB SDD) y Servidor Dell PowerEdge T560 (112 vCPU, 256GB RAM, 2 TB SDD) en la red de la Universidad del Valle de Guatemala dentro del período comprendido entre enero y noviembre de 2025. Se cuentan con módulos de Openstack que proporcionan servicios de computación, almacenamiento y redes. Los servicios clave para levantar la infraestructura son: Keystone, Glance, Placement, Nova, Neutron y Cinder. Además se emplean nodos importantes como Horizon para gestionar y controlar recursos, Ceilometer para monitorización.

La automatización con Ansible garantiza un despliegue reproducible y escalable, permitiendo a los usuarios crear máquinas virtuales, contenedores y recursos de almacenamiento en una infraestructura optimizada y eficiente. Como entregable, se proporciona una guía completa del proceso de instalación de la herramienta que permite la automatización del despliegue en contenedores de Openstack y su futura operación y mantenimiento. Se realiza la configuración de los servicios clave de Openstack mencionados previamente así como la configuración de *backends* empleados para su operación. El aprovisionamiento de servicios adicionales de Openstack está fuera del alcance de este proyecto.

CAPÍTULO 6

Marco teórico

6.1. Tecnologías en la nube

La creciente demanda en recursos tecnológicos han promovido el uso y desarrollo de soluciones que puedan ser provisionados de forma sencilla a través de internet. La computación en la nube permite el acceso a la tecnología de información sin la necesidad de gestión de infraestructura física, dando lugar a modelos de servicio completamente manejables y con operabilidad escalable; los servicios se vuelven flexibles a la necesidad del usuario [6].

6.2. Infraestructura como código

La definición de los diversos modelos de servicio que pueden existir en el ambiente de cómputo en la nube pueden ayudar a comprender el alcance que cada uno de los mismos puede ofrecer a los usuarios y como abstraen tareas que se realizan con equipos *bare metal* en la infraestructura clásica. La abstracción de los modelos de servicio en *software* permite también el manejo y aprovisionamiento de cada uno con metodologías de infraestructura como código para la automatización de la configuración de ambientes de computación. El uso de *frameworks* para la definición de tareas resulta una herramienta esencial para asegurar la consistencia y portabilidad de ambientes de desarrollo a diferentes niveles y permite la definición completa de la infraestructura de forma organizada y con archivos de configuración. Este acercamiento para el despliegue de servicios, emplea prácticas utilizadas en desarrollo de *software* para asegurar rutinas que sean consistentes y repetibles . Algunas prácticas comunes para la aseguraración de una infraestructura resiliente incluyen desarrollo conducido por pruebas (TTC), integración continua (CI) y desarrollo continuo (CD) [7]. El surgimiento de equipos de desarrollo y operaciones vienen a cumplir la necesidad de organizaciones para unificar el trabajo implicado en diseño de infraestructura, dimensionamiento y optimización de costos con consideraciones dentro del ciclo de vida del desarrollo de *software* [8]. la infraestructura como código promueve gestionar el conocimiento y la experiencia de una

gran cantidad de subsistemas como una única fuente de verdad comúnmente disponible, en lugar de reservarla tradicionalmente solo para los administradores de sistemas o aquellos encargados de operaciones.

6.3. Plataformas de infraestructura

Mediante la arquitectura de computación en la nube, los administradores de red pueden habilitar el uso de diversos recursos de cómputo proveyendo a los usuarios finales con sus capacidades de forma rápida y con mínimo esfuerzo en el manejo. En la infraestructura de nube moderna resulta útil agrupar las capas de servicio en tres partes fundamentales [9].

6.3.1. *Software* como servicio

Las aplicaciones que existen encima de las capas de infraestructura y plataforma conforman este nivel de abstracción encargado principalmente de brindar capacidades a los usuarios en la organización. En este nivel típicamente se hallan paquetes de aplicaciones, instancias de contenedores.

6.3.2. Plataforma como servicio

El siguiente nivel de abstracción se encarga de aprovisionar recursos para que las aplicaciones de la capa superior puedan ser ejecutadas, esencialmente brindando un ambiente de computación, también se le conoce como *application runtimes*. Dentro de esta capa se hallan componentes esenciales para que corran aplicaciones, como lo son *servers* de aplicación, bases de datos, ambientes virtuales y en tecnologías más modernas empleando *clusters* de contenedores.

6.3.3. Infraestructura como servicio

La plataforma de infraestructura contiene a los recursos y el set de herramientas y servicios que los manejan. En este caso, se trata de la capa que contiene los elementos primitivos que ayudan a construir la infraestructura sobre la que corren plataformas y aplicaciones. Los servicios de nube y plataformas de virtualización son responsables del despliegue de recursos como cómputo, almacenamiento y conexiones de dispositivos en redes [9].

Figura 1. Capas de plataformas de infraestructura

Nota. La imagen muestra solo algunos de los componente que pueden estar en cada una de las capas. Esta imagen fue recreada de [7].

6.4. Recursos de infraestructura

En su concepción tradicional, la infraestructura de tecnologías de la información está compuesta por una variedad de recursos físicos y virtuales que soportan las operaciones de una organización. La adopción de tecnologías en la nube ha llevado a una evolución en la forma en que se gestionan estos recursos, dando lugar a la infraestructura como código. Entre los recursos se incluyen equipo de cómputo, redes, bases de datos, balanceador de carga, almacenamiento y otros servicios que pueden ser aprovisionados y gestionados de manera sistémica. Los recursos de infraestructura son elementos que pueden ser gestionados

a través de una interfaz de aplicación, permitiendo su creación, modificación y eliminación de forma automatizada. Los proveedores de nubes públicas como AWS, Azure y GCP ofrecen una API [7] para gestionar recursos como instancias de cómputo, redes virtuales y volúmenes de almacenamiento. Herramientas de IaC como Terraform, AWS CloudFormation y Ansible permiten definir estos recursos en archivos de configuración, facilitando la reproducibilidad y el control de versiones.

La automatización de la infraestructura es esencial para la entrega continua y el despliegue confiable de aplicaciones. Además, la gestión declarativa de recursos, donde se describe el estado deseado en lugar de los pasos para alcanzarlo, es una práctica recomendada para mantener la coherencia y facilitar la colaboración entre equipos [10].

6.5. Recursos de almacenamiento

Se refiere a volúmenes de datos que pueden ser accedidos mediante un sistema de archivos. En las plataformas de infraestructura normalmente se encuentran en forma de almacenamiento en bloque, los cuales son utilizados para almacenar datos persistentes que pueden ser montados en instancias de cómputo. Por otra parte, el almacenamiento en objetos es un tipo de almacenamiento que permite guardar datos no estructurados, como archivos multimedia o documentos. Este tipo de almacenamiento es común en servicios de nube pública y privada, proporcionando escalabilidad y durabilidad para grandes volúmenes de datos [7].

6.6. Recursos de redes

En redes tradicionales, la capacidad de una red viene determinada por la disponibilidad de dispositivos físicos como lo son *routers* y *switches*, que son responsables de dirigir el tráfico de datos entre diferentes segmentos de la red. En su concepción tradicional el ancho de banda disponible en la red es limitado por la capacidad de estos dispositivos y la infraestructura física subyacente, lo que puede llevar a cuellos de botella y limitaciones en el rendimiento. La idea de virtualización de redes propone combinar los recursos de redes disponibles, separando el ancho de banda disponible en canales, cada uno independiente del otro, asignables a diversos servicios y aplicaciones en tiempo real; los recursos de red virtualizados son reasignables [11]. La infraestructura en la nube hace uso de recursos virtualizados para asegurar la flexibilidad y escalabilidad de las redes.

6.6.1. Redes definidas por *software* (SDN) y funciones de red virtualizadas (VNF)

El plano de control y el plano de datos son dos componentes fundamentales en la arquitectura de redes. El plano de datos es responsable de la transmisión real de paquetes a través de la red, mientras que el plano de control gestiona las decisiones sobre cómo y dónde se deben enviar esos paquetes. Tradicionalmente, ambos planos están integrados en los dispositivos de red físicos, lo que puede limitar la flexibilidad y escalabilidad de la red [12]. Las

redes definidas por *software* separan el plano de control del plano de datos en los dispositivos de red, permitiendo una gestión centralizada y programable de la infraestructura de red. Esto facilita la automatización, la flexibilidad y la optimización dinámica de los recursos de red. Por otro lado, las funciones de red virtualizadas (VNF) implementan servicios de red tradicionalmente dependientes de *hardware*, como *firewalls* o balanceadores de carga, en entornos virtualizados; mejorando la eficiencia y reduciendo costos [13]. La integración de SDN y VNF es fundamental en arquitecturas modernas de nube y telecomunicaciones.

6.7. Recursos de cómputo

Los recursos de cómputo constituyen el elemento central de cualquier aplicación embebida moderna; interactúan y coordinan el uso de recursos del sistema para lograr la ejecución de aplicaciones [14]. Las plataformas modernas ofrecen la posibilidad de realizar tareas de cómputo en servidores físicos *bare-metal*, máquinas virtuales o contenedores. Siendo estos dos últimos la opción empleada en entornos de nube.

6.7.1. Máquinas virtuales

Las máquinas virtuales son entornos virtualizados que emulan *hardware* físico o servidores *bare-metal*. Una máquina virtual (VM) incluye todos los recursos que tiene una máquina física, como CPU, memoria, almacenamiento y red, pero se ejecuta sobre un hipervisor, ocultando la complejidad física a los usuarios finales. Las VM son una parte fundamental de la arquitectura de computación en la nube, pues permite entregar capacidades en forma de *hardware* virtualizado a través de internet [15]. Comparación en tecnologías de virtualización en entornos de Linux [16] muestran que los contenedores ofrecen características iguales o mejores a VM en términos de rendimiento.

6.7.2. Contenedores

Los contenedores son una forma de virtualización que aseguran la compatibilidad de aplicaciones y sus ambientes al encapsular con estas las librerías y binarios requeridos para la ejecución. A diferencia de las máquinas virtuales, los contenedores comparten el mismo núcleo (*kernel*) del sistema operativo, lo que los hace más ligeros y eficientes en términos de recursos [17].

Docker

El proyecto Docker es una plataforma de código abierto que automatiza la implementación, el escalado y la gestión de aplicaciones mediante la contenedorización. Utiliza componentes de *software* ligero y compatibles con API, responsables de ejecutar y gestionar contenedores, proporcionando entornos aislados a nivel de *kernel* [18]. La amplia adopción de Docker en las tecnologías de la nube se debe a su capacidad para encapsular aplicaciones y sus dependencias en unidades portátiles y consistentes, lo que permite una implementación

fluida en diversos entornos. Al tratarse de una tecnología de virtualización es importante resaltar como Docker interatúa con los recursos de cómputo, almacenamiento y redes del sistema anfitrión.

Sistema de archivos de Docker

Docker utiliza un sistema de archivos en capas que permite la creación eficiente de imágenes de contenedores. Cada capa representa una modificación o adición a la imagen base, y estas capas son apiladas para formar la imagen final del contenedor. Este enfoque en capas facilita la reutilización de componentes comunes entre diferentes imágenes, reduciendo el espacio de almacenamiento y mejorando la velocidad de despliegue. Se destaca entonces que las imágenes de Docker son genéricas y portátiles, ya que pueden ser ejecutadas en cualquier entorno que soporte Docker, independientemente de las diferencias en el sistema operativo subyacente.

El controlador de almacenamiento recomendado para Docker en sistemas Linux modernos es `overlay2`, que utiliza OverlayFS, un sistema de archivos de unión (*union filesystem*). OverlayFS permite superponer dos directorios (llamados `lowerdir` y `upperdir`) y presentar una vista unificada a través de un tercer directorio (`merged`). Las capas inferiores (`lowerdir`) son de solo lectura y corresponden a las capas de la imagen, mientras que la capa superior (`upperdir`) es de lectura-escritura y almacena los cambios realizados por el contenedor. El directorio `workdir` es utilizado internamente por el controlador para operaciones de gestión.

Cada imagen y contenedor tiene su propio directorio bajo `/var/lib/docker/overlay2`. Las imágenes descargadas se almacenan como capas en directorios separados, y los contenedores crean una nueva capa superior para sus cambios. Docker utiliza enlaces simbólicos y archivos auxiliares para gestionar la relación entre capas y optimizar el uso de espacio en disco. El punto de montaje final, visible dentro del contenedor, es el directorio `merged`, que combina todas las capas.

Networking en Docker

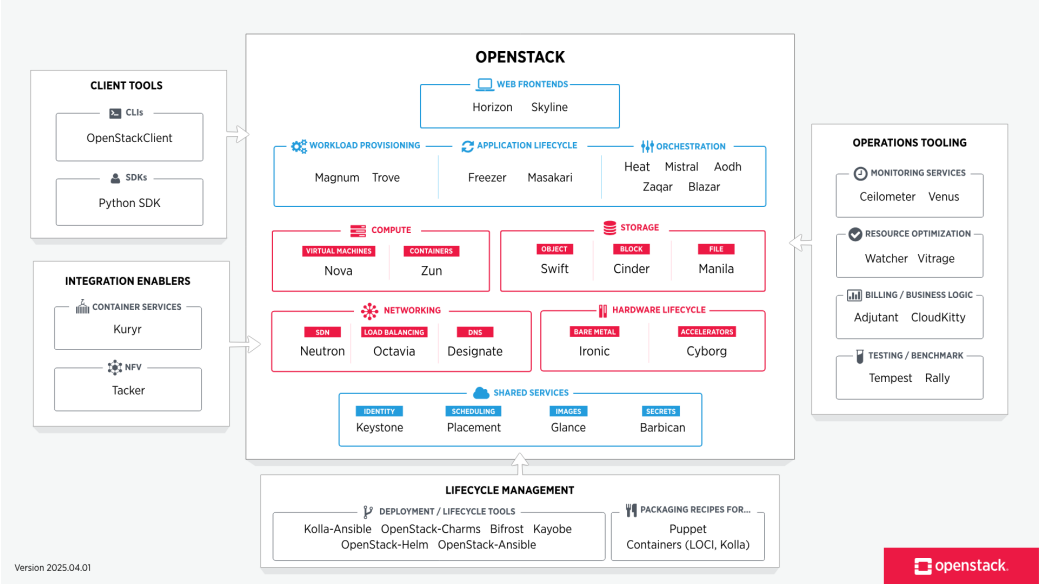
Al igual que otras tecnologías de virtualización, Docker cuenta con las herramientas para armar un *stack* de redes virtualizadas haciendo uso de *network namespaces*, *virtual ethernet interfaces* (*veth*) y puentes de red (*bridges*). Docker crea por defecto una red puente llamada `bridge` que permite la comunicación entre contenedores en el mismo *host*. Cada contenedor recibe una dirección IP única dentro de esta red, y Docker configura las reglas de *iptables* para gestionar el tráfico de red entre contenedores y hacia el exterior. El uso de *namespaces* de red de la ilusión de tener una pila de red dedicada para cada contenedor, aislando su tráfico de red del *host* y otros contenedores.

6.8. Proyecto Openstack

Openstack es una plataforma de código abierto para la gestión de infraestructuras de nube pública y privada. Permite la provisión y administración de recursos de cómputo, al-

macenamiento y red a través de una interfaz unificada, este proyecto es en esencia una manifestación pura de infraestructura como servicio (IaaS). Entre sus servicios principales se encuentran Nova (gestión de instancias de cómputo), Neutron (gestión de redes), Glance (gestión de imágenes), Keystone (autenticación y autorización), y Horizon (interfaz gráfica de usuario) [19]. Openstack facilita la automatización y escalabilidad de los recursos, permitiendo a las organizaciones construir nubes privadas robustas y flexibles

Figura 2. Ecosistema de Openstack para versión 2025.1



Nota. Algunos de estos servicios no son parte del núcleo de Openstack, pero pueden ser integrados para ampliar las capacidades de la plataforma. Esta imagen fue obtenida de [19].

6.8.1. Bibliotecas comunes de Openstack (Oslo)

Oslo es el nombre del proyecto que agrupa las bibliotecas comunes utilizadas a través de todos los servicios de Openstack. Este proyecto como tal no es un servicio de Openstack, sino un set de librerías que contienen código compartido entre los demás servicios de Openstack. Estas bibliotecas proporcionan funcionalidades base como configuración (`oslo.config`), registro (`oslo.log`), mensajería (`oslo.messaging`), y caché (`oslo.cache`), entre otras. La importancia de Oslo radica en que establece un conjunto estandarizado de herramientas y patrones que aseguran la consistencia y interoperabilidad entre los diferentes componentes de Openstack. Por ejemplo, `oslo.messaging` implementa el sistema de comunicación asíncrona entre servicios, permitiendo que componentes como el servicio de computación y servicio de redes intercambien mensajes de manera eficiente y confiable. Esta capa de abstracción facilita el desarrollo y mantenimiento de los servicios de Openstack, ya que los desarrolladores pueden concentrarse en la lógica específica de cada servicio sin preocuparse por reimplementar funcionalidades comunes [19].

6.9. Arquitectura de Openstack

Impulsada por tecnologías de virtualización y contenedorización, la arquitectura de microservicios cambia el paradigma monolítico tradicional, dividiendo aplicaciones complejas en componentes más pequeños e independientes. Cada microservicio se enfoca en una función específica, lo que facilita el desarrollo, despliegue y escalabilidad de aplicaciones en la nube [20]. El modelo de aplicación de Openstack se basa en una arquitectura de microservicios, donde cada componente o servicio es responsable de una función específica dentro del ecosistema de Openstack. La comunicación entre los servicios debe responder a la naturaleza cambiante de la nube y por ende, debe contar con los mecanismos para asegurar consistencia en la transmisión de datos. Para cumplir con este objetivo, cada uno de los servicios de Openstack expone una API RESTful que permite la transferencia de datos de forma eficiente entre los diversos componentes. Esta arquitectura modular facilita la escalabilidad y el mantenimiento del sistema, ya que los servicios pueden ser actualizados o reemplazados de manera independiente sin afectar la funcionalidad global del entorno de nube. Cada despliegue de Openstack expone las capacidades de los servicios a través de la API correspondiente, estas son consumidas por clientes que realizan peticiones dirigidas a *endpoints* específicos. Los *endpoints* son URLs que representan la ubicación de un recurso o servicio dentro de la infraestructura de Openstack. Los clientes pueden interactuar con estos *endpoints* para realizar operaciones como crear, leer, actualizar o eliminar recursos, utilizando los métodos HTTP estándar. Es responsabilidad del administrador la segregación de los *endpoints* según las necesidades de la organización, pudiendo definir *endpoints* públicos o internos [19].

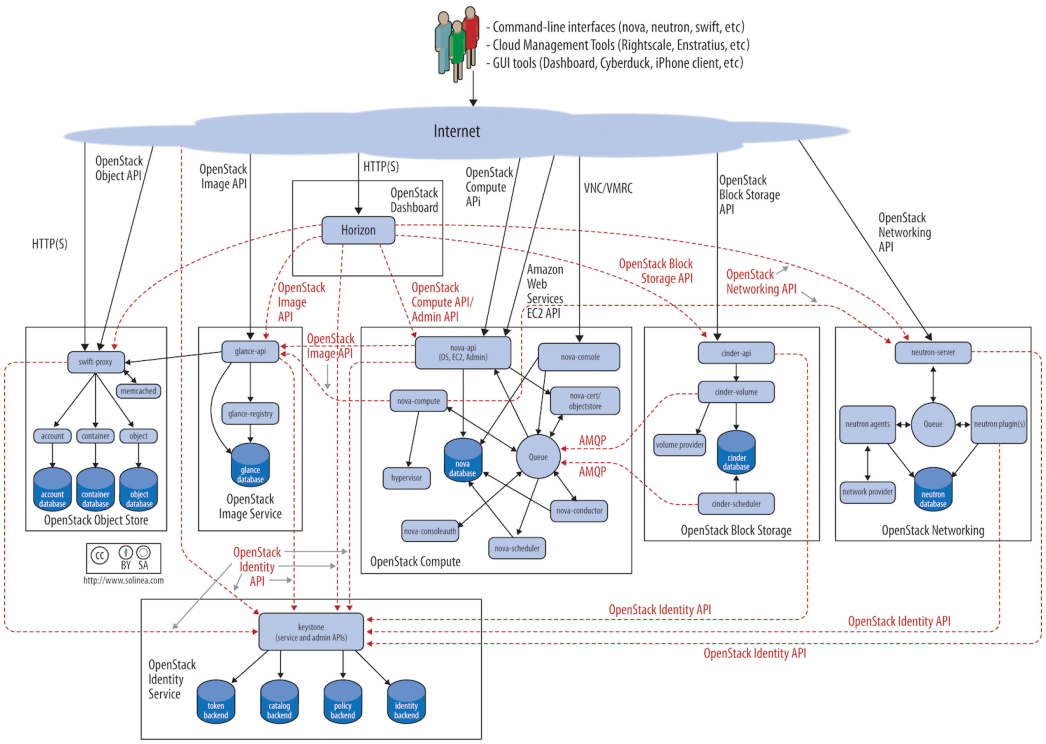
6.9.1. Interfaces de aplicación de transferencia de estado representacional (REST)

El estilo arquitectónico *representational state transfer* (REST) nace como una abstracción de los elementos que conforman un sistema distribuido de hipermedios, siendo la base conceptual del diseño de la red moderna. REST no se centra en los detalles de implementación de componentes ni en la sintaxis de protocolos, sino en los roles de los componentes, las restricciones en su interacción y la interpretación de los datos significativos que circulan entre ellos [21].

REST se fundamenta en una serie de restricciones arquitectónicas que, en conjunto, buscan optimizar la escalabilidad de las interacciones, la independencia en el despliegue de componentes y la visibilidad de las comunicaciones. Estas restricciones derivan de estilos arquitectónicos previos, tales como cliente-servidor, *stateless*, *cacheable*, interfaz uniforme, sistema por capas y código bajo demanda. Su combinación define los principios de la red y de las APIs modernas:

- **Cliente-Servidor**: separa las responsabilidades entre cliente (interfaz de usuario) y servidor (gestión de datos), mejorando la portabilidad y escalabilidad.
- ***Stateless***: cada petición HTTP debe contener toda la información necesaria para ser procesada, sin depender del estado almacenado en el servidor. Esto mejora fiabilidad, visibilidad y escalabilidad.

Figura 3. Arquitectura de microservicios de Openstack



Nota. Los servicios de Openstack se comunican entre sí a través de llamadas a sus respectivas interfaces de aplicación. Destacar que cada servicio está ligado al catálogo de servicios de Keystone para la autenticación y autorización previo a su uso. Esta imagen fue obtenida de [19].

- **Caché**: las respuestas pueden ser etiquetadas como almacenables en caché o no, lo que permite reutilizar información y reducir latencia.
- **Interfaz Uniforme**: los recursos se identifican mediante URI, se manipulan a través de representaciones estandarizadas y se accede a ellos mediante un conjunto reducido de métodos.
- **Sistema por capas**: permite introducir intermediarios (proxies, gateways, firewalls) sin alterar la interacción básica entre cliente y servidor, facilitando seguridad, traducción de protocolos y balanceo de carga.
- **Código bajo demanda (opcional)**: posibilita ampliar las capacidades del cliente mediante la descarga de applets o scripts.

En REST, los datos fundamentales son los **recursos**, que se identifican de manera única mediante URI. Los clientes no acceden al recurso "crudo", sino a una **representación** de este. Estas representaciones son transferidas entre componentes junto con metadatos que permiten al receptor interpretarlas y procesarlas.

La interacción en REST sigue el modelo **CRUD** (*create, read, update, delete*), mapeado directamente a los verbos HTTP. Así, una petición `GET` permite leer un recurso, `POST` crear

uno nuevo, `PUT/PATCH` actualizarlo y `DELETE` eliminarlo. Además, se pueden añadir parámetros de consulta en la URL para filtrar o paginar los resultados, lo que resulta clave en API de gran escala como las de Openstack.

REST proporciona un marco de principios y restricciones que hacen posible construir interfaces de aplicación escalables, independientes y fácilmente consumibles. En el contexto de Openstack, entender REST es fundamental porque cada servicio expone sus capacidades a través de una **API RESTful**, que se convierte en el medio estándar para autenticar, consultar, crear y administrar recursos en la nube.

6.10. Servicios de Openstack

Openstack está compuesto por una serie de servicios modulares que trabajan en conjunto para proporcionar una plataforma completa de infraestructura en la nube. Cada servicio es responsable de una función específica dentro del ecosistema de Openstack, y todos ellos se comunican entre sí a través de una API RESTful.

6.10.1. Openstack Keystone

El servicio de autenticación y autorización de la nube Openstack, Keystone, proporciona un sistema centralizado para gestionar identidades y permisos de acceso a los recursos de la nube. El servicio proporciona una interfaz de programación de aplicaciones para el manejo de *tokens* de acceso, usuarios, roles y proyectos, permitiendo a los administradores controlar quién puede acceder a qué recursos dentro de la infraestructura de Openstack. Esto hace al servicio de Keystone un componente crítico para ambientes de producción, donde la seguridad y el control de acceso son esenciales. La convención de transferencia de estado representacional (REST) es utilizada para la comunicación con el servicio, permitiendo una integración sencilla con otros componentes de Openstack y aplicaciones externas. La arquitectura modular de Openstack hace uso extensivo de Keystone para la autenticación de usuarios y servicios. Cada servicio de Openstack debe autenticarse con Keystone para obtener un *token* que le permita interactuar con otros servicios. Este enfoque centralizado simplifica la gestión de identidades y mejora la seguridad al reducir la necesidad de múltiples sistemas de autenticación [22].

6.10.2. Openstack Glance

El servicio de imágenes de Openstack, Glance, permite a los usuarios almacenar, recuperar y gestionar imágenes de máquinas virtuales. Glance expone su API RESTful para interactuar con las imágenes, permitiendo a los usuarios crear instancias de cómputo a partir de estas imágenes. Además, Glance soporta múltiples formatos de imagen y almacenamiento, lo que facilita la integración con diferentes sistemas de almacenamiento y la reutilización de imágenes en diferentes entornos. Este servicio es esencial para la creación y gestión eficiente de instancias en Openstack, permitiendo a los usuarios desplegar rápidamente entornos de desarrollo y producción [23].

6.10.3. Openstack Placement

El servicio de colocación de Openstack, Placement, es responsable de la gestión y asignación de recursos dentro de la infraestructura de nube. Este servicio permite a los usuarios y administradores definir políticas de colocación (en coordinación con otros servicios de autenticación) para optimizar el uso de recursos, como lo son procesadores, memoria y almacenamiento, en función de las necesidades específicas de las cargas de trabajo. Placement proporciona una API que permite a los servicios de Openstack consultar y reservar recursos disponibles, facilitando la planificación y ejecución eficiente de instancias y otros servicios en la nube [24].

6.10.4. Openstack Nova

El servicio de cómputo de Openstack, Nova, es responsable de la gestión de instancias de máquinas virtuales y recursos de cómputo. Nova proporciona una API RESTful que permite a los usuarios crear, eliminar y gestionar instancias, así como asignar recursos como CPU, memoria y almacenamiento. Además, Nova soporta múltiples hipervisores, lo que permite a los usuarios elegir la tecnología de virtualización que mejor se adapte a sus necesidades. Este servicio es una pieza fundamental para la provisión de recursos de cómputo en Openstack, permitiendo a las organizaciones escalar sus infraestructuras de nube de manera eficiente [25]. Dentro de la interfaz de programación de aplicaciones de Nova, se puede destacar el rol de *scheduler* que se encarga de asignar las instancias a los nodos de cómputo disponibles, optimizando el uso de recursos y garantizando un rendimiento adecuado para las cargas de trabajo.

6.10.5. Openstack Neutron

El servicio de redes de Openstack, Neutron, proporciona una infraestructura de red flexible y escalable para la gestión de redes virtuales. Neutron permite a los usuarios crear y gestionar redes, subredes, enrutadores y otros componentes de red a través de una API RESTful. Este servicio soporta múltiples tecnologías de virtualización de redes, lo que permite a los usuarios implementar redes complejas y personalizadas según sus necesidades. Neutron es esencial para la provisión de conectividad entre instancias y otros servicios en Openstack, facilitando la creación de entornos de red seguros y eficientes [26]. El servicio interactúa con diversos componentes de la infraestructura que actúan como *switches* y otros controladores de red para proporcionar una solución integral de redes virtualizadas. Esto viene a cumplir con la necesidad de brindar conectividad dinámica a entornos que han acogido diversas tecnologías para la virtualización de redes.

Neutron emplea redes de proveedor y redes de proyecto para segmentar el tráfico de la red. Las redes de proveedor aseguran la conexión con la infraestructura física, mientras que las redes de proyecto emplean *namespaces* de Linux para aislar el tráfico de diferentes proyectos, asegurando la seguridad y eficiencia en la gestión del tráfico de red [26]. Al tratarse del proyecto que brinda conectividad a las instancias de cómputo, Neutron es un componente crítico en la arquitectura de Openstack. Para el funcionamiento de la red *core* de Openstack se emplean diferentes elementos como lo son `neutron-server`, `neutron-dhcp-agent`,

`neutron-l3-agent` y `neutron-openvswitch-agent`, cada uno con funciones específicas para asegurar la provisión y gestión de redes virtuales en el entorno de nube [26].

neutron-server

Es el componente principal del servicio y encargado de procesar solicitudes de la API RESTful, gestionar la base de datos de red y coordinar las operaciones entre los diferentes agentes de Neutron.

neutron-dhcp-agent

Es responsable de proporcionar servicios DHCP a las instancias dentro de las redes virtuales, asignando direcciones IP y gestionando la configuración de red.

neutron-l3-agent

Gestiona las funciones de enrutamiento y NAT (traducción de direcciones de red) para las redes virtuales, permitiendo la comunicación entre diferentes subredes y con redes externas.

neutron-openvswitch-agent

Se encarga de gestionar los puentes y puertos de OVS en los nodos de cómputo, facilitando la conectividad de red para las instancias de máquinas virtuales.

neutron-plugins

Son componentes opcionales que permiten la integración de Neutron con diferentes tecnologías de virtualización de redes, proporcionando soporte para diversas arquitecturas y protocolos de red que puedan ofrecer diversos proveedores de servicios.

6.10.6. Openstack Cinder

El servicio de almacenamiento en bloque de Openstack, Cinder, proporciona una solución para la gestión de volúmenes de almacenamiento persistente. Cinder permite a los usuarios crear, adjuntar y gestionar volúmenes de almacenamiento que pueden ser montados en instancias de cómputo. El proyecto permite asignar diversos *backends* de almacenamiento, lo que facilita la integración con diferentes sistemas de almacenamiento y la optimización del rendimiento según las necesidades específicas de las cargas de trabajo. Cinder expone una API RESTful que permite a los usuarios interactuar con los volúmenes de almacenamiento de manera eficiente y flexible [27].

6.10.7. Openstack Horizon

El servicio de interfaz gráfica de usuario de Openstack, Horizon, proporciona un panel de control web para gestionar los recursos y servicios de la nube. Horizon permite a los usuarios interactuar con Openstack a través de una interfaz intuitiva, facilitando la creación y gestión de instancias, redes, volúmenes y otros recursos. Este servicio es esencial para usuarios que prefieren una experiencia visual en lugar de interactuar directamente con las APIs RESTful. Horizon también permite a los administradores personalizar la apariencia y funcionalidad del panel de control, adaptándolo a las necesidades específicas de su organización [28].

6.11. Operación de Openstack

6.11.1. Comunicación entre Servicios

Los servicios de Openstack se comunican entre sí mediante un sistema de mensajería asíncrona implementado con RabbitMQ. Este *broker* de mensajes actúa como intermediario central, permitiendo la comunicación desacoplada entre los diferentes componentes. De forma complementaria, los servicios también exponen APIs RESTful para operaciones síncronas y acceso externo. Esta combinación de mensajería asíncrona y APIs RESTful asegura una comunicación eficiente, escalable y flexible dentro del ecosistema de Openstack [19].

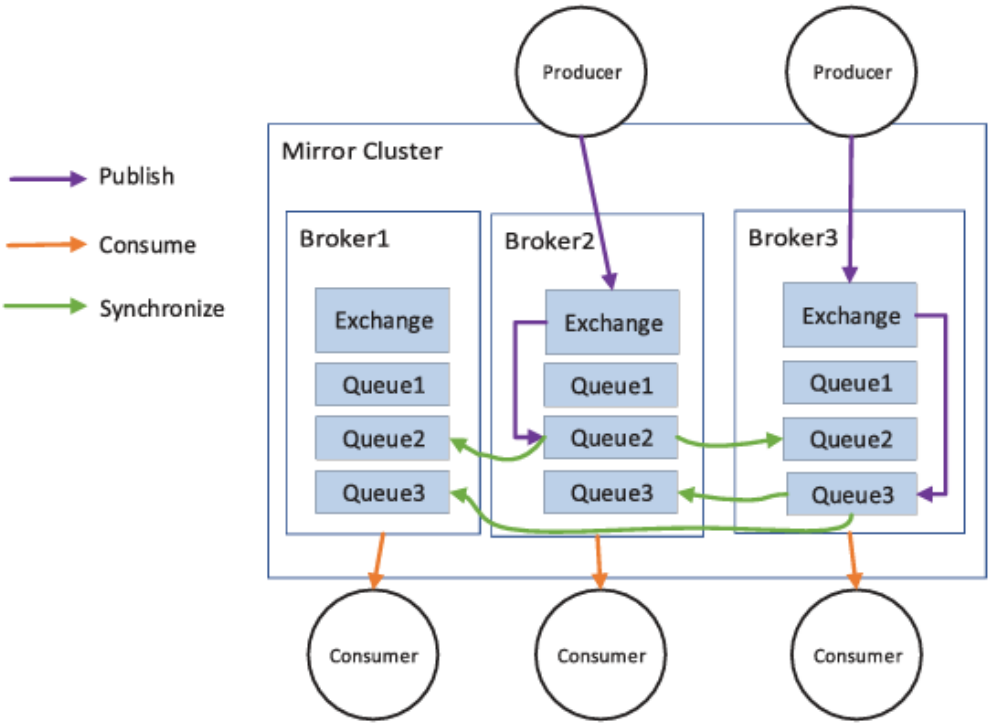
Flujo de comunicación via RabbitMQ (AMQP)

RabbitMQ es un proyecto de código abierto que implementa el protocolo *advanced message queuing protocol* (AMQP) para la mensajería asíncrona entre servicios [29]. En Openstack, RabbitMQ facilita la comunicación entre los diversos servicios mediante el intercambio de mensajes a través de colas. En términos prácticos, este servicio actúa como soporte de *backend* para la librería `oslo.messaging`, que es utilizada por los servicios de Openstack para enviar y recibir mensajes de manera eficiente y confiable.

El flujo de comunicación sigue una secuencia estándar que garantiza la persistencia de los mensajes y alta disponibilidad.

1. **Publicación de mensajes**: los productores envían mensajes a un *exchange*, que actúa como agente de reenvío.
2. **Enrutamiento**: el glsexchange distribuye los mensajes a las colas correspondientes según:
   - Reglas de vinculación (*binding rules*) específicas
   - El tipo de glsexchange utilizado
   - Puede enviar un mensaje a múltiples colas o a una cola específica
3. **Almacenamiento en cola**: los mensajes se almacenan en colas que implementan un sistema de alta disponibilidad:

Figura 4. Arquitectura de sistema de mensajería



Nota. La imagen ilustra el flujo de mensajes entre productores, glsexchange, colas y consumidores en RabbitMQ. Esta imagen fue obtenida de [29].

- Cada cola tiene un nodo maestro y uno o más espejos
- Las operaciones se aplican primero al nodo maestro
- Los cambios se propagan a los nodos espejo

4. **Consumo de mensajes**: los consumidores pueden recibir mensajes mediante:
   - Modo Push: RabbitMQ envía activamente los mensajes
   - Modo Pull: los consumidores solicitan mensajes

Alta Disponibilidad

La alta disponibilidad en RabbitMQ se logra mediante el espejado de colas *mirroring*. En caso de fallo del nodo maestro, los consumidores pueden ser redirigidos automáticamente a una cola espejo en otro *broker*, garantizando la continuidad del servicio.

Flujo de comunicación via HTTP REST

Además del sistema de mensajería asíncrona, los servicios de Openstack complementan su comunicación mediante solicitudes HTTP RESTful. Este método de comunicación es

utilizado principalmente para operaciones que requieren una respuesta inmediata o para la interacción con clientes externos. Cada servicio expone sus API de acuerdo a la arquitectura implementada. Estas API permiten a los usuarios y otros servicios realizar operaciones CRUD (crear, leer, actualizar, eliminar) sobre los recursos gestionados por Openstack. La comunicación vía HTTP REST sigue el modelo cliente-servidor, donde los clientes envían solicitudes HTTP a los *endpoints* de los servicios de Openstack, y estos responden con los datos solicitados o el resultado de la operación.

6.11.2. Autenticación y ciclo de vida de *tokens*

Cuando un usuario o servicio solicita acceso, Keystone genera un token que sigue el siguiente ciclo:

- **Emisión**: el usuario presenta credenciales válidas y recibe un token
- **Validación**: el token es verificado en cada solicitud de API
- **Almacenamiento**: los *tokens* activos se mantienen en caché (típicamente usando Memcached)
- **Revocación**: los *tokens* pueden ser invalidados antes de su expiración
- **Expiración**: los *tokens* caducan automáticamente después de un tiempo configurado

6.11.3. Estado y Persistencia

Openstack utiliza un esquema de bases de datos relacionales para mantener el estado de los recursos. Cada servicio mantiene su propia base de datos, típicamente implementada en MariaDB, con tablas específicas para sus recursos:

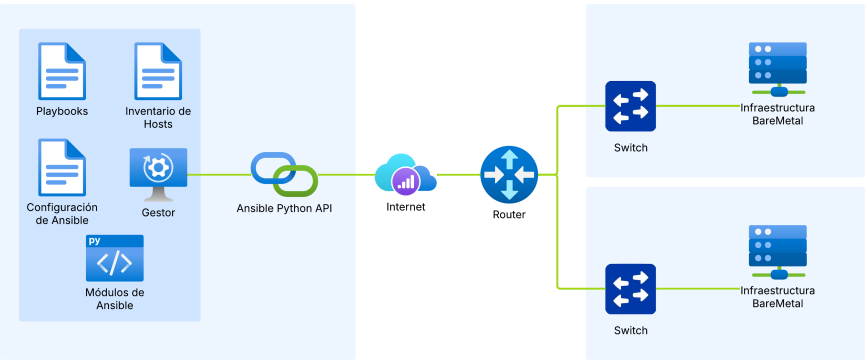
- **Keystone**: almacena usuarios, roles, proyectos y asignaciones.
- **Glance**: guarda metadatos de imágenes y su ubicación.
- **Placement**: registra disponibilidad y asignaciones de recursos físicos.
- **Nova**: almacena información sobre instancias, sabores, asignaciones de recursos.
- **Neutron**: mantiene configuración de redes, subredes, puertos y reglas de seguridad.
- **Cinder**: almacena volúmenes, snapshots y sus estados.
- **Horizon**: no mantiene estado persistente.

El esquema de base de datos se gestiona mediante migraciones automatizadas, permitiendo actualizaciones sin pérdida de datos. Para mejorar el rendimiento, se utiliza Memcached para almacenar en caché datos frecuentemente accedidos como *tokens* y catálogo de servicios.

6.12. Ansible

Ansible es una herramienta de automatización y orquestación de infraestructura que utiliza archivos de configuración escritos en YAML (*yet another markup language*) para la configuración de entornos. Ansible es un proyecto desarrollado en Python y puede emplear Docker para la creación de contenedores, lo que permite una fácil integración con otras herramientas y servicios, así como el monitoreo de recursos. Está diseñado para el despliegue en múltiples *hosts* y es compatible con diversos sistemas operativos que incluyen distribuciones populares de Linux e incluso versiones de Windows. Permite definir tareas y flujos de trabajo de manera declarativa, facilitando la gestión de servidores, redes y aplicaciones de forma reproducible y escalable [30]. Para su despliegue, Ansible emplea tres elementos clave, *inventory files*, *playbooks* y *modules* que permiten definir los recursos a gestionar, las tareas a ejecutar y las acciones específicas a realizar en cada recurso, respectivamente [31]. El uso de YAML como lenguaje de definición proporciona una sintaxis sencilla e interpretable tanto por administradores como usuarios. La sintaxis sencilla reduce la complejidad y los errores en la automatización de infraestructuras.

Figura 5. Arquitectura básica de Ansible con separación de dominios de *broadcast*



Nota. La imagen muestra dos infraestructuras de *baremetal*, esto es ilustrativo y puede ser extendido a la demanda. Esta imagen fue recreada de [32].

6.12.1. Lenguajes de representación de datos

En entornos de automatización y orquestación como Ansible, la representación de datos juega un papel crucial en la definición y gestión de configuraciones. Uno de los lenguajes más utilizados para este propósito es YAML, que se destaca por su simplicidad y legibilidad. Es un formato de serialización de datos legible por humanos, diseñado para ser fácil de leer y escribir. Al igual que lenguajes como Python, emplea indentación para definir la estructura de los datos, por lo que los espacios en blanco son significativos. Ansible usa YAML para definir sus *playbooks*. Un ejemplo de un *playbook* puede contener una lista de tareas «también conocidas como *plays*» que se ejecutarán en un conjunto de *hosts*. La sintaxis de YAML es simple y clara, lo que facilita la comprensión y edición de los archivos de configuración [33].

A continuación, se muestra un ejemplo básico de un archivo YAML:

Cuadro 6.1. Ejemplo de un *playbook* de Ansible para instalar un paquete

```
- name: Mi primera tarea
  hosts: all
  become: true
  tasks:
    - name: Instalar paquete bash-completion
      package:
        name: bash-completion
        state: present
```

En este ejemplo, se define un *playbook* que ejecuta una tarea en todos los *hosts* especificados. El *playbook* como tal es el archivo de YAML y puede contener múltiples *plays* (tareas). Cada tarea se define con un nombre y un módulo, en este caso, el módulo `package` se utiliza para instalar el paquete `bash-completion`. La opción `state: present` dicta que el paquete debe estar instalado en el sistema.

6.12.2. Tareas de Ansible

Los *playbooks* de Ansible son archivos de configuración escritos en YAML que definen un conjunto de tareas a ejecutar en uno o más *hosts*. Estos archivos permiten describir de manera declarativa el estado deseado de la infraestructura, especificando qué acciones deben realizarse y en qué orden. Los *playbooks* son fundamentales para la automatización de tareas repetitivas y la gestión de configuraciones complejas, ya que permiten definir flujos de trabajo que pueden ser ejecutados de manera consistente y reproducible. Además, los *playbooks* pueden incluir variables, condiciones y *bucles*, lo que proporciona flexibilidad y adaptabilidad a diferentes entornos y escenarios.

Al integrar diversas llamadas a módulos, (que a su vez invocan comandos del sistema operativo), el *playbook* contiene una lista de tareas que se ejecutarán en los *hosts* especificados en el inventario. Véase el caso para instalar un paquete de *software* en Linux en el siguiente fragmento de código:

Cuadro 6.2. Ejemplo de tarea en un *playbook* de Ansible para instalar Apache HTTPD en sistemas basados en Debian

```
- name: Install apache httpd
  apt:
    name: apache2
    state: present
```

El fragmento representa una tarea en un *playbook* de Ansible, donde la sección `- name: Install apache httpd` es simplemente una descripción legible para el usuario. La tarea en sí se encuentra en la sección `apt:` que indica que se utilizará el módulo `apt` para gestionar paquetes en sistemas basados en Debian. La opción `name: apache2` especifica el paquete que se desea instalar o gestionar, mientras que `state: present` indica que el paquete debe estar presente en el sistema, es decir, instalado. Aunque `state=present` es opcional en muchos casos, su inclusión aclara que se busca garantizar la instalación del paquete, para este caso `apache2`.

6.12.3. Inventarios de Ansible

Los inventarios de Ansible son archivos que definen los *hosts* y grupos de *hosts* sobre los cuales se ejecutarán las tareas especificadas en los *playbook*. Estos archivos pueden ser estáticos, donde se enumeran explícitamente los *hosts*, o dinámicos, donde se generan automáticamente a partir de una fuente externa, como una base de datos o un servicio de nube como en Kolla-Ansible. Los inventarios permiten organizar los *hosts* en grupos según su función, grupos de usuarios o dominios; un inventario facilita la gestión y ejecución de tareas en múltiples servidores de manera eficiente. Además, los inventarios pueden incluir variables específicas para cada *host* o grupo, lo que proporciona flexibilidad y personalización en la configuración de la infraestructura. Puede ser especificado en el inventario que los *hosts* que se pretende manejar pueden ser localizados en el directorio que contiene la configuración específica del sistema. Esto funciona para resolución de nombres que contengan una dirección estática. Para sistemas UNIX, este archivo se halla en el directorio `/etc` con la información estática de nombres a resolver en el archivo `hosts` [34].

Cuadro 6.3. Ejemplo de un archivo de inventario estático de Ansible

```
127.0.0.1   localhost
# Openstack Virtual Nodes
192.168.1.2 node1
192.168.1.3 node2
```

Los inventarios deben ser descritos de forma que la identificación de los nodos sea clara y concisa. Se debe permitir a administradores identificar de forma rápida los nodos que serán utilizados, como es el caso de usar nombres de nodos para la resolución de direcciones IP.

6.12.4. Módulos de Ansible

Los módulos de Ansible son componentes reutilizables que encapsulan una tarea específica, como la instalación de un paquete, la configuración de un servicio o la gestión de archivos. Estos módulos se ejecutan en los *hosts* gestionados y pueden ser invocados desde los *playbooks* para realizar acciones específicas. Ansible incluye una amplia variedad de módulos predefinidos para tareas comunes, y también permite a los usuarios crear sus propios módulos personalizados según sus necesidades. La modularidad de Ansible facilita la reutilización de código y la colaboración entre equipos, ya que los módulos pueden ser compartidos y utilizados en diferentes proyectos y entornos. El administrador puede escoger escribir sus propios módulos en cualquier lenguaje de programación, pero Python es el más utilizado y es el lenguaje en que está escrito Ansible y los módulos predefinidos. El proceso de ejecución de un módulo implica la lectura de un *playbook*, Ansible se encarga de analizar y convertir el contenido de las tareas en llamadas en un entorno que pueda ser ejecutado. Los módulos hacen uso de herramientas de Python que interactúan con el sistema operativo del nodo, permitiendo la ejecución de comandos en línea de código o dentro de un *script*. Véase el siguiente ejemplo de la función `install` del módulo `apt`, utilizado para manejar paquetes en sistemas basados en Debian [30].

Cuadro 6.4. Fragmento del módulo de manejo de paquetes de Ansible en sistemas basados en Debian

```
def install(m, pkgspec, cache, upgrade=False, default_release=None,
            install_recommends=None, force=False, ...):
    # Preparación de la lista de paquetes
    pkgspec = expand_pkgspec_from_fnmatches(m, pkgspec, cache)
    package_names = []
    for package in pkgspec:
        name, version_cmp, version = package_split(package)
        ...
        installed, installed_version, version_installable, has_files
            = package_status(
            m, name, version_cmp, version, default_release, cache,
                ...)
        # Lógica para determinar qué paquetes instalar o actualizar
        ...
        # Construcción del comando apt
        cmd = "%s -y %s ..." % (APT_GET_CMD, dpkg_options)
        if build_dep:
            cmd += " build-dep %s" % (packages)
        else:
            cmd += " install %s" % (packages)
        # Opciones adicionales
        if default_release:
            cmd += " -t '%s'" % (default_release,)
        if install_recommends is False:
            cmd += " -o APT::Install-Recommends=no"
        elif install_recommends is True:
            cmd += " -o APT::Install-Recommends=yes"
        # Opciones condicionales
        if allow_unauthenticated:
            ...
        if allow_change_held_packages:
            ...
        # Ejecución del comando
        with PolicyRcD(m):
            rc, out, err = m.run_command(cmd)
        # Análisis del resultado
        ...
```

En este fragmento, el método principal para ejecutar la instalación o actualización es la llamada a `m.run_command(cmd)`. Esta función ejecuta en el sistema operativo el comando construido dinámicamente en función de los argumentos analizados del archivo YAML (esta tarea comúnmente se conoce como *parsing*).

Cuadro 6.5. Definición de la variable global para herramienta de paquetes avanzada de Ansible

```
global APT_GET_CMD
APT_GET_CMD = module.get_bin_path("apt-get")
```

Este fragmento de código realiza dos acciones. Primero, la declaración `global APT_GET_CMD` indica que la variable puede ser accedida desde otras funciones o bloques en el módulo. Luego, la línea `APT_GET_CMD = module.get_bin_path("apt-get")` llama a la fun-

ción `get_bin_path`, cuyo propósito es localizar la ruta completa del ejecutable `apt-get` en el sistema operativo. Esto garantiza que la variable `APT_GET_CMD` contiene la ubicación exacta del binario, por ejemplo, `/usr/bin/apt-get`, que luego será utilizada en la construcción de comandos para gestionar paquetes, asegurando así que se invoca la versión correcta de `apt-get` independiente de dónde se encuentre el *playbook* o el módulo de Ansible.

Se combina la variable `APT_GET_CMD` (una variable global en el entorno) con diversas opciones y paquetes, incluyendo configuraciones como la decisión si realizar una actualización completa, construir dependencias, forzar la instalación, entre otras. En el fragmento se omite la lógica para determinar si el paquete ya se encuentra instalado o requiere actualización. La salida de `run_command` se descompone en tres variables: `rc` (código de retorno), `out` (salida estándar) y `err` (error estándar). Posteriormente, el código analiza el valor de `rc` para determinar si la operación fue exitosa y, en caso negativo, genera mensajes de error apropiados que serán procesados por el *playbook* correspondiente. La lógica de la función simula la ejecución de un comando en la línea de comandos de Debian/Ubuntu, permitiendo gestionar paquetes desde un *playbook* en YAML, donde las variables y argumentos del *playbook* serían traducidos a los valores de entrada de esta función.

Así el módulo de Ansible hace una traducción de las tareas definidas en el *playbook* a comandos específicos del sistema operativo (que a su vez están orquestados por Python), permitiendo la automatización de la gestión de paquetes en este ejemplo.

6.13. Kolla-Ansible

Kolla-Ansible es un proyecto de código abierto que proporciona herramientas y plantillas para implementar Openstack utilizando contenedores Docker. Kolla-Ansible utiliza Ansible para orquestar la configuración y el despliegue de los servicios de Openstack, facilitando la automatización de tareas repetitivas y la gestión de configuraciones complejas. Kolla Ansible tiene soporte para múltiples proyectos de Openstack [35], incluyendo Nova, Neutron, Glance, Keystone y otros. Además Kolla-Ansible despliega contenedores para elementos de infraestructura como lo son bases de datos, servidores de mensajería y servicios de almacenamiento. Algunos servicios destacados son:

- Collectd, Telegraf, InfluxDB, Prometheus y Grafana: utiliza su servicio de recopilación de métricas y monitoreo de rendimiento.
- OpenSearch: un motor de búsqueda y análisis de datos.
- Etcd: un almacén de claves-valor
- Fluentd: un servicio de recopilación y procesamiento de registros.
- Gnocchi: un servicio de almacenamiento y consulta de métricas.
- HAProxy y Keepalived: proporciona balanceo de carga y alta disponibilidad.
- MariaDB Galera Cluster: un servicio de base de datos distribuida y replicada.
- Memcached: un servicio de almacenamiento en caché en memoria.

- OVS: un conmutador virtual para redes definidas por *software*.
- RabbitMQ: un servicio de mensajería para la comunicación entre servicios de Openstack.
- Redis: un almacén de datos en memoria utilizado para almacenamiento en caché y mensajería.

6.14. Operación de Kolla-Ansible

Entre las herramientas de Kolla se encuentran comandos que facilitan la gestión de un entorno Openstack utilizando contenedores. Los comandos de Kolla-Ansible pueden aprovecharse para renovar el estado de contenedores, eliminarlos o actualizar las imágenes de contenedores.

- `deploy`: este comando se utiliza para desplegar e iniciar todos los contenedores de Kolla en el entorno especificado.
- `destroy`: permite eliminar todos los contenedores y volúmenes creados, útil para reinicios desde cero.
- `prechecks`: verifica que todos los requisitos estén satisfechos antes de proceder con el despliegue. Esto incluye la validación de paquetes, *hosts* alcanzables y con direcciones resolvibles.
- `post-deploy`: ejecuta tareas posteriores al despliegue, como la obtención del archivo utilizado para la autenticación.
- `pull`: descarga todas las imágenes necesarias de contenedores sin iniciar servicios. Esta función es sumamente útil para asegurar que se usen las imágenes más recientes, véase como una rutina de actualización en una máquina tradicional.

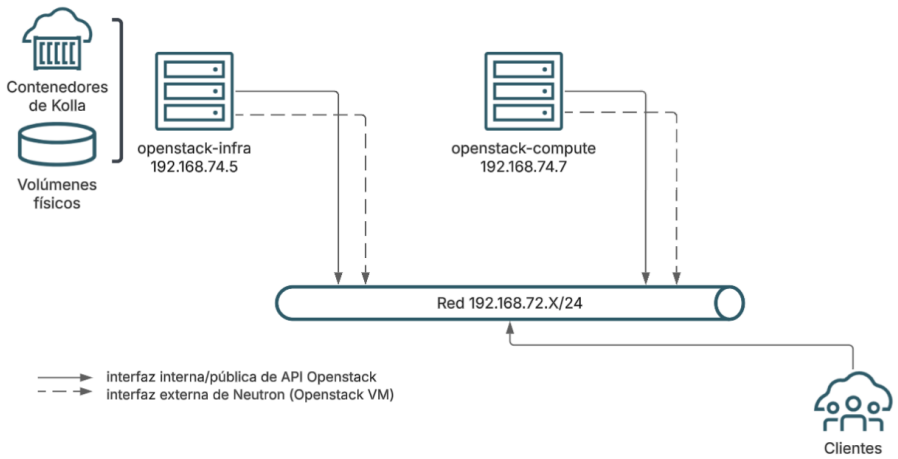
CAPÍTULO 7

Configuración y despliegue de OpenStack como infraestructura como código

La filosofía del proyecto Kolla-Ansible se basa en la automatización del despliegue de Openstack utilizando contenedores. Para lograr esto, es fundamental preparar adecuadamente el entorno del *host* donde se realizará el despliegue. Esta preparación incluye la configuración del sistema operativo para que la capa de control pueda gestionar recursos físicos de manera eficiente.

7.1. Definición de topología

Figura 6. Diagrama de topología física del laboratorio



Nota. Se obvia la infraestructura física subyacente en capa 2 del laboratorio, enfatizando la conexión entre nodos físicos y acceso a la *subnet* de laboratorio

La configuración del entorno debe obedecer al estado físico del *hardware* disponible. Para los recursos de red es necesario contar con interfaces de red para manejar el tráfico externo

y para recursos de almacenamiento contar con los volúmenes físicos que se atarán en la configuración. En cuanto a recursos de cómputo (CPU y RAM), Kolla-Ansible se encargará de su gestión a través de contenedores, por lo que el problema de compatibilidad se reduce a la instalación de un sistema operativo compatible y un interpretador de Ansible.

Antes de definir la asignación de recursos y configuración global de Kolla-Ansible es necesario preparar el sistema operativo del *host* que se utilizará para el despliegue. Definir la topología del laboratorio resulta fundamental para asignar correctamente direcciones IP únicas y estáticas para realizar las configuraciones correspondientes en cada uno de los equipos físicos. Los equipos físicos, también llamados nodos, pueden tener diferentes roles dentro del entorno de laboratorio, como nodo de control, nodo de cómputo o nodo de almacenamiento. Cada uno de estos roles puede requerir configuraciones específicas en cuanto a red y almacenamiento. Para el despliegue actual se distinguen dos nodos físicos:

- **Nodo 1 (control y cómputo):** Este nodo administra los servicios principales de Openstack y cuenta con todos los recursos necesarios para gestionar un entorno de nube. En este nodo se encuentran los archivos de configuración de Kolla-Ansible y cuenta con la tarea de orquestar el despliegue.

  Hostname: openstack-infra

- **Nodo 2 (cómputo dedicado):** Este nodo está dedicado exclusivamente a las tareas de cómputo, ejecutando instancias de máquinas virtuales y gestionando recursos de CPU y RAM. Cumple el propósito de distribuir carga de trabajo y dar la pauta a la escalabilidad del entorno.

  Hostname: openstack-compute

La matriz de soporte de Openstack define los sistemas operativos compatibles para el despliegue de Openstack mediante Kolla-Ansible. Entre las opciones recomendadas se encuentran distribuciones populares como Ubuntu, CentOS y Red Hat Enterprise Linux (RHEL). El administrador puede decidir si el despliegue se realizará en máquinas físicas o virtuales, siempre y cuando se cumplan los requisitos mínimos de *hardware* y *software*.

Aunque la configuración de los *hosts* es mínima, es importante asegurarse de que el sistema operativo esté actualizado y cuente con las herramientas necesarias para la gestión de contenedores y la automatización mediante Ansible. Para este despliegue se distingue configuraciones propias del nodo de control y del nodo de cómputo, las cuales se detallan en las siguientes secciones.

7.2. Instalación del sistema operativo

Para este despliegue se ha seleccionado Ubuntu Server 24.04 LTS debido a su estabilidad, amplia comunidad de soporte y compatibilidad con las últimas versiones de Openstack y Kolla-Ansible. La instalación del sistema operativo debe realizarse en ambos nodos físicos. La instalación del sistema operativo se realiza en función de los recursos físicos disponibles y necesidades del entorno de la nube. Siempre que sea posible y se cuente con controladores, es recomendable utilizar discos en configuración RAID para asegurar la redundancia y alta

disponibilidad de los datos. La administración de volúmenes lógicos mediante LVM también es aconsejable para facilitar la gestión del almacenamiento y permitir una mayor flexibilidad en la asignación de espacio a los diferentes servicios de Openstack. En lo que a manejo de usuarios y contraseñas respecta, se recomienda crear usuarios con privilegios administrativos limitados para las tareas diarias, reservando el usuario `root` únicamente para tareas de mantenimiento y configuración avanzada que serán necesarias durante el despliegue y operación del entorno de nube. El manejo del usuario `root` cae en responsabilidad del administrador del sistema, quien debe asegurarse de que las credenciales sean seguras y se mantengan confidenciales. Para esta prueba de concepto se emplean estas credenciales de usuario:

- Nombre de usuario hpc1: `hpc1`
- Nombre de usuario hpc2: `hpc2`
- Contraseña: `openstack`

7.2.1. Actualizar el sistema

Al crear un usuario y contraseña para usar el servidor, se procede a actualizar paquetes del sistema a su última versión disponible. durante el despliegue se hará uso extensivo de la línea de comandos tanto para propósitos de configuración como operación del entorno. Todos los comandos a continuación se ejecutan con privilegios de superusuario, se obviará el prefijo `sudo` por simplicidad. Si no se desea usar el usuario `root`, se debe anteponer `sudo` a cada comando o utilizar un usuario con privilegios administrativos.

Cuadro 7.1. Actualización de paquetes

```
apt update && apt upgrade -y
```

7.2.2. Instalar herramientas necesarias

Se instalan herramientas básicas que serán útiles para la administración del sistema y diagnóstico de red.

Cuadro 7.2. Instalación de herramientas del sistema

```
apt install net-tools bind9-dnsutils inetutils-traceroute -y
```

7.2.3. Establecer zona horaria

Es importante configurar la zona horaria correcta para asegurar que los registros de eventos y servicios del sistema tengan coherencia entre nodos.

Cuadro 7.3. Habilitar sincronización de tiempo

```
systemctl status systemd-timesyncd.service
systemctl enable systemd-timesyncd.service
systemctl restart systemd-timesyncd.service
```

Cuadro 7.4. Configuración de zona horaria

```
timedatectl set-timezone America/Guatemala
timedatectl set-ntp on
timedatectl status
```

Verificar que la hora del sistema sea congruente a través de todos los nodos involucrados en el despliegue.

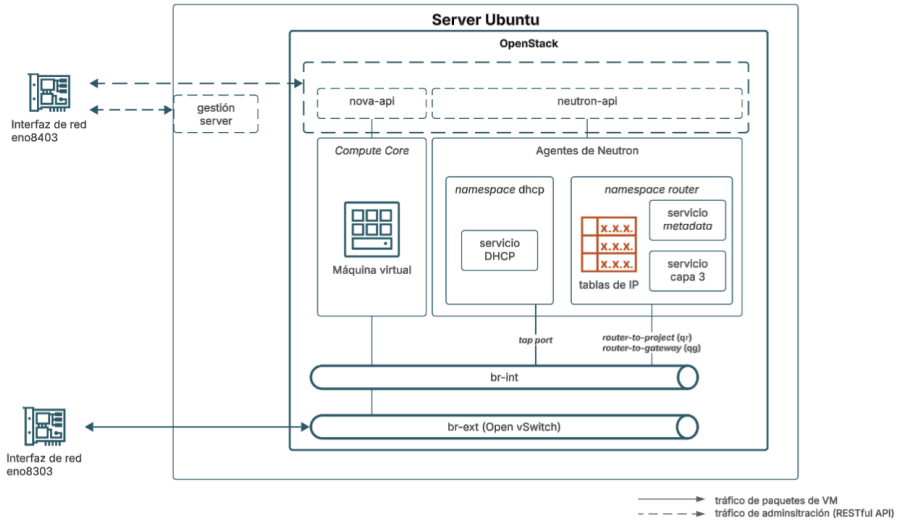
Cuadro 7.5. Salida de comandos de hora

```
               Local time: ddd aaaa-mm-dd hh:mm:ss CST
           Universal time: ddd aaaa-mm-dd hh:mm:ss UTC
                 RTC time: ddd aaaa-mm-dd hh:mm:ss
                Time zone: America/Guatemala (CST, -0600)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

7.2.4. Definición del plan de red (*networking*)

Una configuración de red adecuada es crucial para el correcto funcionamiento de un entorno de nube. Cada nodo debe tener una dirección IP estática para garantizar la comunicación constante entre los diferentes componentes de Openstack. Además, es importante definir correctamente el nombre del *host* y asociar las direcciones IP a las interfaces de red físicas correspondientes.

Figura 7. Escenario de despliegue de red con OVS



Nota. Imagen basada en casos descritos en documentación de Neutron para distintos escenarios de despliegue de red en Openstack [26].

7.2.5. Establecer el nombre del *host*

El nombre del *host* es un identificador único para cada nodo dentro de la red. Es importante definir nombres descriptivos que reflejen el rol de cada nodo en el entorno de nube. Estos nombres luego serán traducidos a direcciones IP mediante el archivo `/etc/hosts` o un servidor de resolución de nombres DNS.

Desde nodo 1 (control y cómputo)

Cuadro 7.6. Establecer el nombre del *host* en nodo 1

```
1 hostnamectl set-hostname openstack-infra
```

Desde nodo 2 (cómputo dedicado)

Cuadro 7.7. Establecer el nombre del *host* en nodo 2

```
1 hostnamectl set-hostname openstack-compute
```

7.2.6. Asociar direcciones IP a interfaces físicas

En sistemas basados en Linux, el método de configuración de red puede variar según la distribución y versión del sistema operativo. Ubuntu Server 24.04 utiliza Netplan como sistema de configuración de red. En esta sección se define una dirección IP estática para la interfaz de red del nodo de despliegue. Esto es especialmente importante en entornos de laboratorio o producción donde se requiere una red estática para que no sea modificada por DHCP, pues el acceso a las rutas de red incide directamente en la disponibilidad de los servicios desplegados. Las direcciones IP asignadas a cada nodo se utilizan para la modificación de archivos de configuración en los *hosts*. La interfaz de red física utilizada para la gestión del equipo en este despliegue es la misma que se emplea para exponer la API de Openstack a clientes externos. Se utiliza una interfaz física adicional para darle conectividad a las máquinas virtuales desplegadas en el entorno de nube como se ilustra en la Figura 7. Según la especificación de Kolla-Ansible, dicha interfaz debe estar activa sin una IP asociada, por lo que se obvia la configuración de red para esta interfaz adicional. Es posible verificar si la interfaz está activa mediante el comando `ip a`.

El archivo de configuración que se modificará normalmente se encuentra en `/etc/netplan/50-cloud-init.yaml`. Se utilizará el editor de texto `nano` para cambiar su configuración.

Desde nodo 1 (control y cómputo)

Se edita el archivo que contiene la configuración de red en línea de comando.

Cuadro 7.8. Edición del archivo de configuración de red

```
nano /etc/netplan/50-cloud-init.yaml
```

Al ingresar al archivo se deben modificar los campos correspondientes a la interfaz de red física, asignando el método de obtención de IP y la dirección deseada.

Cuadro 7.9. Configuración de red estática en nodo 1

```
network:
  version: 2
  ethernets:
    eno8403:
      dhcp4: false
      addresses:
        - 192.168.74.5/22
      routes:
        - to: default
          via: 192.168.72.1
      nameservers:
        addresses:
          - 1.1.1.1
```

Explicación de los campos:

- `version`: indica la versión del esquema Netplan.
- `ethernets`: define las interfaces de red físicas. Dentro de esta sección es posible definir múltiples interfaces si el sistema cuenta con más de una o configurar VLAN.
- `eno8403`: nombre de la interfaz de red (puede variar según el *hardware*).
- `dhcp4`: false: desactiva DHCP, ya que se utilizará una IP estática.
- `addresses`: dirección estática asignada al host, con su máscara de red (CIDR).
- `routes`: define la ruta por defecto y su *gateway*.
- `nameservers`:servidores DNS que utilizará el sistema.

Una vez editado y guardado el archivo, es necesario aplicar la configuración de red para que los cambios surtan efecto. Si la configuración se realiza desde una consola remota (SSH), es recomendable utilizar el comando `netplan try` en lugar de `netplan apply`. Esto permite probar la nueva configuración y revertirla automáticamente si se pierde la conexión, evitando quedar bloqueado fuera del sistema.

Cuadro 7.10. Probar la configuración de red

```
netplan try
```

Si se tiene certeza que la configuración es correcta, se puede aplicar directamente en consola el comando `netplan apply`.

Cuadro 7.11. Aplicar la configuración de red

```
netplan apply
```

Desde nodo 2 (cómputo dedicado)

Se repiten los pasos anteriores para editar el archivo de configuración de red en el segundo nodo físico, con una dirección IP diferente acorde a la topología definida. En este punto el administrador debe decidir si los nodos adicionales tendrán asociados el mismo nombre de interfaz física para el despliegue de Kolla-Ansible. Para el despliegue actual Kolla asume que se utiliza una interfaz física con el nombre `eno8403`. La alternativa es definir nombres distintos para cada nodo, lo cual no afecta el despliegue siempre y cuando se mantenga la coherencia en la configuración global de Kolla-Ansible y se realice la configuración específica para cada nodo. Esto es deseable cuando otros servicios o aplicaciones dependen del nombre por defecto de la interfaz física. Para el primer caso es necesario hallar el nombre de la interfaz física y dirección MAC asociada en el segundo nodo físico. Esto se logra ejecutando el comando `ip a` o `ifconfig` en la consola del nodo de cómputo dedicado.

Cuadro 7.12. Hallar nombre de interfaz física y dirección MAC

```
ip a
```

La salida del comando mostrará las interfaces de red disponibles, junto con sus direcciones MAC. Se debe identificar la interfaz que se utilizará para la conexión de red y anotar su nombre y dirección MAC.

Cuadro 7.13. Ejemplo de salida de comando de configuración de red

```
2: enp0s21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
    fq_codel state UP group default qlen 1000
     link/ether aa:bb:cc:dd:ee:ff brd ff:ff:ff:ff:ff:ff
     inet
```

Explicación de los campos de interés:

- `enp0s21`: nombre de la interfaz de red en el segundo nodo físico.
- `aa:bb:cc:dd:ee:ff`: estructura de 12 dígitos hexadecimales que representa dirección MAC de la interfaz física en el segundo nodo físico. Es única para cada dispositivo de red.

Cuadro 7.14. Configuración de red estática en nodo 1

```
network:
  version: 2
  ethernets:
    enp0s21:
      dhcp4: true
      addresses:
      - 192.168.74.7/22
```

```
8       routes:
9       - to: default
10        via: 192.168.72.1
11      nameservers:
12          addresses:
13          - 1.1.1.1
14      match:
15        macaddress: aa:bb:cc:dd:ee:ff
16      set-name: eno8403
```

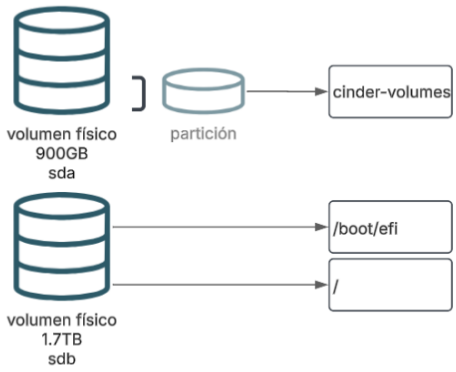
Explicación de los campos modificados:

- `match`: se utiliza para identificar la interfaz física mediante su dirección MAC.
- `set-name`:: permite renombrar la interfaz física al nombre deseado para mantener coherencia en la configuración. Para este despliegue se renombra a `eno8403` para que coincida con el nombre de la interfaz en el nodo de control.

7.3. Plan de almacenamiento

Tanto Openstack como Docker soportan múltiples *backends* de almacenamiento. La elección del *backend* adecuado depende de factores como el rendimiento, la escalabilidad y la facilidad de administración. Para este despliegue se utilizará almacenamiento local en ambos nodos físicos, aprovechando los discos disponibles en cada equipo. Es importante asegurarse de que los discos estén correctamente configurados y tengan suficiente espacio para manejar las imágenes de contenedores, volúmenes de datos y otros recursos necesarios para el funcionamiento de Openstack. Para entornos de producción se deben considerar alternativas de almacenamiento robustas como Ceph, NFS o almacenamiento en la nube de terceros (como Amazon S3) para consideraciones de escalabilidad. Para esta prueba se utilizará el soporte de LVM en ambos nodos físicos, creando volúmenes lógicos dedicados para Docker y Openstack. Al tratarse de configuraciones que afectan directamente la disponibilidad de recursos de sistema, es necesario emplear el usuario `root` para realizar las configuraciones de almacenamiento. También se destaca que la configuración de arranque del sistema no puede hacer parte de los volúmenes lógicos, por lo que se reserva una partición dedicada para este propósito.

Figura 8. Plan de almacenamiento propuesto para despliegue



Nota. Imagen creada para reflejar la asignación de particiones y volúmenes lógicos para el despliege

Desde nodo 1 (control y cómputo)

7.3.1. Verificación de discos físicos

Se crean volúmenes lógicos dedicados al almacenamiento en bloque empleado por Cinder. Se inicia con la verificación de los discos físicos disponibles en el sistema mediante el comando `lsblk`. La salida esperada para este comando debe mostrar una estructura de discos y particiones con este formato

Cuadro 7.15. Ejemplo de salida del comando para discos

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   900G  0 disk
sdb      8:16   0   1.7T  0 disk
|-sdb1   8:17   0     1G  0 part /boot/efi
|-sdb2   8:18   0   1.7T  0 part /
```

En esta salida se asume que se cuenta con un disco adicional `dev/sda` que será utilizado para crear un volumen lógico dedicado Si se tuvieran particiones existentes en el disco que se desea utilizar (`dev/sdaX`, dónde X corresponde al número de partición), es recomendable eliminarlas antes de proceder con la creación del volumen físico. Para este propósito se utiliza la herramienta `fdisk`.

Cuadro 7.16. Eliminación de particiones existentes

```
fdisk /dev/sda
d [número de partición]
```

Dentro de la herramienta `fdisk`, se utiliza el comando `p` para listar las particiones existentes y el comando `d` para eliminar cada partición. Una vez eliminadas todas las particiones, se utiliza el comando `w` para guardar los cambios y salir de la herramienta.

7.3.2. Creación de volumen físico

Tras cerciorarse de la disponibilidad del disco físico y que su formato sea adecuado, se procede a crear un volumen físico (PV) utilizando el comando `pvcreate`. Este comando inicializa el disco para su uso en LVM.

Cuadro 7.17. Creación de volumen físico

```
pvcreate /dev/sda1
vgcreate cinder-volumes /dev/sda1
```

Para particionar el disco se emplea la utilidad `parted`, la cual permite crear una tabla de particiones GPT y definir una partición que ocupe todo el espacio disponible en el disco. Se inicia la herramienta GPT fdisk

Cuadro 7.18. Particionado del disco físico

```
gdisk /dev/sda
```

Dentro de la herramienta existen diversos comandos para gestionar la tabla de particiones. Para listar todos los comandos disponibles, se utiliza el comando `?` dentro de la herramienta. A continuación, se crean las particiones necesarias utilizando los comandos adecuados.

Cuadro 7.19. Creación de partición en disco

```
n          # Nueva partición
2          # Número de partición
[Enter]    # Primer sector (valor por defecto)
+400G      # Último sector (tamaño de la partición)
w          # Escribir cambios y salir
```

Para que los cambios surtan efectos es necesario volver a cargar la tabla de particiones del sistema. En la consola se ingresa el comando `partprobe`.

Cuadro 7.20. Recarga de tabla de particiones

```
partprobe /dev/sda
```

7.3.3. Verificación de volumen físico

Tras realizar la configuración del almacenamiento, es recomendable verificar que el volumen físico se haya creado correctamente. Se utiliza el comando `pvdisplay`

Cuadro 7.21. Verificación de volumen físico

```
pvdisplay /dev/sda1
```

La salida esperada del comando muestra el estado del volumen físico, incluyendo su tamaño, estado y otros detalles relevantes.

Cuadro 7.22. Ejemplo de salida del comando para despliegue de discos

```
  --- Physical volume ---
  PV Name               /dev/sda1
  VG Name               cinder-volumes
  PV Size               400.00 GiB / not usable 3.00 MiB
  Allocatable           yes
  PE Size               4.00 MiB
  Total PE              102400
  Free PE               102400
  Allocated PE          0
  PV UUID               xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

7.4. Generación de claves SSH

Ansible emplea SSH como método de comunicación predeterminado para gestionar nodos remotos. Para permitir la conexión sin contraseña entre el nodo de control y los nodos remotos, es necesario generar un par de claves SSH y copiar la clave pública a cada nodo remoto. Esto es esencial para la automatización del despliegue y gestión de Openstack mediante Kolla-Ansible.

7.4.1. Generación de claves en el nodo de control

Desde el nodo de control se genera un par de claves SSH (pública y privada) sin contraseña, especificado como un *string* vacío (`""`). La clave privada se almacena en el directorio `/root/.ssh/` del usuario.

Cuadro 7.23. Generación de claves SSH

```
ssh-keygen -t rsa -N "" -f /root/.ssh/id_rsa
```

7.4.2. Copia de clave pública a nodos remotos

La clave pública fue copiada a los nodos remotos para permitir la autenticación sin contraseña. Se debe reemplazar `user` por el nombre de usuario de Ubuntu e `IP` por la dirección IP del nodo al que se desea acceder mediante SSH. El comando completo es `ssh-copy-id user@IP`.

Para el entorno de laboratorio, se copia la clave pública al nodo de cómputo dedicado utilizando su dirección IP configurada previamente.

Cuadro 7.24. Ejemplo de copia de clave al nodo de cómputo

```
ssh-copy-id hpc1@192.168.74.7
```

7.5. Instalación de dependencias del sistema

Para el correcto funcionamiento de Kolla-Ansible y sus componentes, es necesario instalar dependencias del sistema en ambos nodos. Estas dependencias incluyen herramientas de desarrollo, librerías del sistema y utilidades para la gestión de contenedores.

7.5.1. Actualización e instalación de paquetes base

Se actualizaron los repositorios del sistema y se instalaron las dependencias necesarias para la compilación y ejecución de componentes de Python y Kolla-Ansible.

Cuadro 7.25. Instalación de dependencias base en ambos nodos

```
apt update
apt install git python3-dev libffi-dev gcc libssl-dev libdbus-glib-1-dev -y
```

Descripción de los paquetes instalados:

- `git`: sistema de control de versiones utilizado para clonar repositorios.
- `python3-dev`: archivos de desarrollo de Python necesarios para compilar módulos.
- `libffi-dev`: librería para interfaces de funciones externas.
- `gcc`: compilador de C requerido para módulos nativos de Python.
- `libssl-dev`: librería de desarrollo para SSL/TLS.
- `libdbus-glib-1-dev`: librería para comunicación con D-Bus.

7.6. Instalación de Docker

Docker es el motor de contenedores utilizado por Kolla-Ansible para desplegar los servicios de Openstack. La instalación se realiza siguiendo el método oficial de Docker, agregando su repositorio oficial a las fuentes de paquetes del sistema. Los contenedores de Docker interactúan directamente con el sistema operativo y su gestión es importante para manejar sus volúmenes e imágenes, factores que inciden directamente en los recursos físicos de almacenamiento del equipo.

Desde ambos nodos

7.6.1. Instalación de certificados y configuración del repositorio

Se instalaron los paquetes necesarios para gestionar certificados y se agregó la clave GPG del repositorio oficial de Docker.

Cuadro 7.26. Preparación del repositorio de Docker

```
apt install -y ca-certificates curl gnupg lsb-release
mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --
    dearmor -o /etc/apt/keyrings/docker.gpg
```

Se agrega el repositorio de Docker a las fuentes de APT, utilizando la versión correspondiente a la distribución instalada.

Cuadro 7.27. Agregación del repositorio de Docker

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/
     keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.
     list > /dev/null
```

7.6.2. Instalación de componentes de Docker

Se actualizaron los repositorios y se instalaron los componentes de Docker, incluyendo el motor, la interfaz de comando de ínea, `containerd` y los *plugins* de Docker.

Cuadro 7.28. Instalación de Docker

```
apt update
apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-
    plugin docker-compose-plugin
```

7.6.3. Instalación del módulo Python para Docker

Para permitir que Ansible gestione Docker, se instaló el módulo `python3-docker`.

Cuadro 7.29. Instalación del módulo Python para Docker

```
apt install -y python3-docker
```

7.6.4. Configuración de Docker

Al tratarse de una dependencia central de Kolla, la configuración de Docker es concerniente a los operadores de la nube y debe tratarse con el debido cuidado. Las configuraciones pueden variar según las necesidades del entorno y los recursos disponibles. Los valores que se escogen vienen a tratar de optimizar el uso de recursos de almacenamiento.

7.6.5. Volúmenes de Docker

Los proyectos de Openstack almacenan una gran cantidad de datos en los volúmenes de Docker. Por defecto, Docker almacena sus datos en el directorio `/var/lib/docker`, lo cual

puede no ser ideal si se cuenta con un volumen dedicado para este propósito. Se recomienda configurar Docker para utilizar un volumen específico que ofrezca mayor rendimiento (en velocidad de acceso a archivos) y capacidad de almacenamiento para manejar datos generados por *logging*. Considerando una ubicación con las características idóneas, los volúmenes de Docker pueden ser redirigidos a dicha ubicación mediante la edición del archivo de configuración `/etc/docker/daemon.json`. Si el archivo no existe, se debe crear.

Cuadro 7.30. Configuración de volúmenes de Docker

```
{
  "data-root": "/mnt/docker-data",
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m",
    "max-file": "3"
  }
}
```

7.6.6. Alta disponibilidad de contenedores

Las consideraciones de alta disponibilidad vienen implícitas en la arquitectura de Openstack y no es necesario configurar Docker para este propósito.

7.7. Configuración de entorno virtual de Python

Para aislar las dependencias de Kolla-Ansible del sistema operativo, se creó un entorno virtual de Python. Esto permite instalar paquetes específicos sin afectar las librerías del sistema y facilita la gestión de versiones. A partir de este momento todos los comandos relacionados con Kolla-Ansible se ejecutan dentro del entorno virtual activado y pueden ser diferenciados por el prefijo del entorno virtual en el *prompt* de la terminal.

7.7.1. Instalación de herramientas para entornos virtuales

Se instala el paquete `python3-venv` que permite la creación de entornos virtuales de Python. De esta forma se asegura que las configuraciones específicas que Kolla-Ansible emplea para el despliegue no interfieran con otras versiones de librerías de Python. Kolla-Ansible descubre automáticamente el entorno virtual activo y utiliza sus binarios para la instalación de dependencias y ejecución de tareas. Aunque no es estrictamente necesario utilizar la misma ruta de entorno virtual en todos los nodos, se recomienda para asegurar consistencia en el despliegue remoto. Si se altera la ruta del entorno virtual, existen diversas estrategias para informar a Kolla-Ansible sobre la ubicación de los binarios de Python, como la modificación de variables de entorno en la configuración de `ansible.cfg` o la edición de archivos de inventario (en este despliegue el inventario `multinode`) que apunten a nodos específicos con el par valor-clave `ansible_python_interpreter`. También es posible instalar un intérprete de

Python específico en el nodo remoto y configurar Kolla-Ansible para automatizar la instalación de dependencias en dicho intérprete. Sin embargo, esta configuración puede complicar el despliegue por dependencias adicionales y debe asegurar que el sistema operativo tenga soporte para uso de módulos como `yum`, `apt`, o `selinux`, que no están disponibles en PyPI.

Desde ambos nodos

Cuadro 7.31. Instalación de paquete de ambiente virtualizado Python

```
apt install python3-venv -y
```

7.7.2. Creación y activación del entorno virtual

Aunque los binarios dentro del entorno virtual pueden ubicarse en cualquier ruta del sistema, es recomendable crear un directorio específico para alojar el entorno virtual. Esta ruta será referenciada constantemente como el **directorio de trabajo**. Además se empleará para almacenar configuraciones globales del proyecto Kolla-Ansible. Se crea una carpeta en `/root/openstack` para alojar el entorno virtual. El propio entorno virtual se crea mediante el módulo `venv` de Python y se le otorga un nombre a dicho entorno.

Cuadro 7.32. Creación del entorno virtual

```
cd /root/openstack
python3 -m venv os-venv
```

Una vez creado, el entorno virtual fue activado. Al activarse, el *prompt* de la terminal muestra el nombre del entorno virtual entre paréntesis. **Los comandos ejecutados en el entorno virtual se muestran con este prefijo, indicando que se está trabajando dentro del entorno aislado.**

Cuadro 7.33. Activación del entorno virtual

```
source /root/openstack/os-venv/bin/activate
```

7.7.3. Actualización de gestor de paquetes e instalación de módulos Python

Dentro del entorno virtual activado, se actualizó el gestor de paquetes `pip` a su última versión y se instalaron los módulos necesarios para la gestión de contenedores Docker y comunicación con D-Bus. Notar que estos binarios pertenecen exclusivamente al entorno virtual y no afectan las librerías del sistema operativo.

Cuadro 7.34. Actualización de gestor de paquetes e instalación de módulos

```
(os-venv) pip install -U pip
(os-venv) pip install docker
(os-venv) pip install dbus-python
```

7.8. Instalación de Kolla-Ansible

Kolla-Ansible es la herramienta de despliegue que automatiza la instalación de Openstack mediante contenedores Docker. La instalación se realiza desde el repositorio oficial de Kolla-Ansible para la versión más reciente.

7.8.1. Instalación desde repositorio Git

Dentro del entorno virtual activado, se instala Kolla-Ansible directamente desde su repositorio en la rama `master`.

Cuadro 7.35. Instalación de Kolla-Ansible

```
(os-venv) pip install git+https://opendev.org/openstack/kolla-ansible@master
```

7.8.2. Creación de directorios de configuración

Se crea el directorio `/etc/kolla` donde se almacenarán los archivos de configuración de Kolla-Ansible. Esta carpeta es de gran utilidad para el operador de Kolla. Se asignó la propiedad del directorio al usuario actual para facilitar la edición de archivos. La creación de directorios y copia de archivos tiene efecto a lo largo de todo el entorno, por lo que no es necesario anteponer el prefijo del entorno virtual en el *prompt*.

Cuadro 7.36. Creación del directorio de configuración

```
mkdir -p /etc/kolla
chown $USER:$USER /etc/kolla
```

7.8.3. Copia de archivos de configuración

Se copiaron los archivos de ejemplo de configuración desde el directorio de instalación del entorno virtual al directorio de configuración. Estos archivos incluyen `globals.yml` y `passwords.yml`, que contienen la configuración global y las contraseñas de los servicios respectivamente.

Cuadro 7.37. Copia de archivos de configuración

```
cp -r /root/openstack/os-venv/share/kolla-ansible/etc_examples/kolla/* /etc/kolla
```

7.8.4. Copia del archivo de inventario

Se copió el archivo de inventario de ejemplo al directorio de trabajo. Por la naturaleza del despliegue que involucra diversos nodos físicos, se utilizó el archivo de inventario `multinode` como base para definir los nodos involucrados en el despliegue. Alternativamente, para despliegues en un solo nodo se puede utilizar el archivo de inventario `all-in-one`, donde todos

los roles de Openstack se despliegan en un solo equipo. Este archivo se encuentra en la misma ubicación que el archivo `multinode`. Los archivos de inventario se pueden ejecutar desde cualquier ubicación, es altamente recomendable mantenerlos en el **directorio de trabajo** para facilitar su gestión.

Cuadro 7.38. Copia del archivo de inventario

```
cd /root/openstack
cp /root/openstack/os-venv/share/kolla-ansible/ansible/inventory/multinode .
```

7.8.5. Instalación de dependencias de Ansible

Se instalan las dependencias de Ansible requeridas por Kolla-Ansible mediante el comando `install-deps`. Al ser instrucciones de kolla, es necesario anteponer el prefijo del entorno virtual en el *prompt*.

Cuadro 7.39. Instalación de dependencias de Ansible

```
source /root/openstack/os-venv/bin/activate
(os-venv) kolla-ansible install-deps
```

7.9. Configuración de archivo de inventario en múltiples nodos

Ansible utiliza un archivo de inventario para definir los nodos que forman parte del despliegue. En este caso, se utilizó el archivo de inventario `multinode` para especificar los nodos de control y cómputo. Se editaron las secciones correspondientes para reflejar la topología definida previamente. Para los *hosts* remotos se especificaron los parámetros de conexión necesarios, incluyendo el usuario, la contraseña de `become` y la ruta a la clave privada SSH generada anteriormente. El *host* local (nodo de control) utiliza la conexión local de Ansible. En esta sección se obvia el detalle del interpretador de Python, ya que Ansible lo detecta automáticamente al momento de la ejecución. La configuración restante del despliegue es específica al nodo 1 (control y cómputo). El resto de configuraciones de nodos remotos se definen en el archivo de inventario y no es necesario interactuar directamente con los nodos remotos para este propósito.

Desde nodo 1 (control y cómputo)

Cuadro 7.40. Configuración del archivo de inventario multinodo

```
[control]
openstack-infra       ansible_connection=local

[network]
openstack-infra       ansible_connection=local
openstack-compute     ansible_user=openstack-compute
   ansible_become_password=openstack ansible_become=True
   ansible_private_key_file=/root/.ssh/id_rsa
```

```
7
8  [compute]
9  openstack-infra       ansible_connection=local
10 openstack-compute     ansible_user=openstack-compute
      ansible_become_password=openstack ansible_become=True
      ansible_private_key_file=/root/.ssh/id_rsa
11 [monitoring]
12 openstack-infra       ansible_connection=local
13
14 [storage]
15 openstack-infra       ansible_connection=local
16
17 [deployment]
18 openstack-infra       ansible_connection=local
```

7.9.1. Verificación de conectividad entre nodos con Ansible

Antes de proceder con el despliegue, se verifica la conectividad entre los nodos definidos en el archivo de inventario utilizando el módulo `ping` de Ansible. Esto asegura que Ansible pueda comunicarse correctamente con todos los nodos y los nombres definidos en el inventario sean resueltos a la dirección asociada. Cuando se invoca el inventario `multinode`, es necesario brindar la ruta completa al archivo de inventario si no se encuentra en el **directorio de trabajo**. Dado que el archivo de inventario se encuentra en dicho directorio, se puede invocar directamente sin especificar la ruta completa.

Cuadro 7.41. Verificación de conectividad con Ansible

```
1 (os-venv) ansible -i multinode all -m ping
```

La salida esperada del comando muestra un mensaje de éxito para cada nodo, indicando que la conexión fue exitosa. Además se indica el intérprete de Python descubierto en cada nodo remoto. Dado que los nodos fueron configurados exactamente igual, el intérprete de Python es el mismo en ambos nodos.

Cuadro 7.42. Ejemplo de salida de comando para conectividad

```
1  openstack-infra | SUCCESS => {
2      "ansible_facts": {
3          "discovered_interpreter_python": "/root/openstack/os-venv/
              bin/python3.12"
4      },
5      "changed": false,
6      "ping": "pong"
7  }
8  openstack-compute | SUCCESS => {
9      "ansible_facts": {
10         "discovered_interpreter_python": "/root/openstack/os-venv/
              bin/python3.12"
11     },
12     "changed": false,
13     "ping": "pong"
14 }
```

7.10. Configuración global de Kolla-Ansible

Antes de realizar el despliegue, es necesario configurar los parámetros globales de Kolla-Ansible en el archivo `/etc/kolla/globals.yml`. Este archivo contiene opciones como interfaces de red, métodos de autenticación y servicios a desplegar. Notar que las configuraciones que se realizan en este archivo afectan a todo el entorno de Openstack y deben ser coherentes con la topología y recursos disponibles.

7.10.1. Configuración básica de imágenes e interfaces de red

El archivo `globals.yml` es probablemente el archivo de configuración más importante de Kolla-Ansible. En este archivo se definen parámetros globales que afectan el comportamiento de todo el entorno de Openstack. Existen múltiples parámetros que pueden ser configurados, se comienza con las configuraciones básicas necesarias para el despliegue. Se emplea el editor de texto `nano` para modificar el archivo.

Cuadro 7.43. Campos editados dentro de archivo de configuración global

```
kolla_base_distro: "ubuntu"
network_interface: "eno8403"
neutron_external_interface: "eno8303"
kolla_internal_vip_address: "192.168.74.69"
```

Descripción de los campos editados:

- `kolla_base_distro`: define la distribución base para los contenedores de Kolla. En este caso, se utiliza Ubuntu.
- `network_interface`: especifica la interfaz de red utilizada para la comunicación interna entre los servicios de Openstack. Debe coincidir con la interfaz configurada en los nodos.
- `neutron_external_interface`: define la interfaz de red que se utilizará para el tráfico externo de Neutron. Esta interfaz puede ser VLAN o *flat* según la topología de red definida. Debe estar activa sin una IP asociada.
- `kolla_internal_vip_address`: dirección IP flotante que será empleada por los servicios de Openstack para la comunicación interna. Esta IP debe ser alcanzable desde todos los nodos y no debe estar en uso por otros dispositivos en la red.

7.10.2. Activación de servicios adicionales

Se habilitaron servicios adicionales en el archivo `globals.yml` para ampliar las capacidades del entorno de Openstack. Estos servicios incluyen soporte para almacenamiento en bloque, balanceo de carga y monitoreo.

Cuadro 7.44. Activación de servicios adicionales en archivo de configuración global

```
...
enable_cinder: "yes"
enable_haproxy: "yes"
enable_ceilometer: "yes"
enable_aodh: "yes"
enable_gnocchi: "yes"
enable_grafana: "yes"
enable_prometheus: "yes"
enable_barbican: "yes"
enable_magnum: "yes"
...
```

Todos estos componentes son opcionales y su activación depende de las necesidades específicas del entorno. Es importante revisar la documentación oficial de Kolla-Ansible para entender las implicaciones de habilitar cada servicio adicional. Para el despliegue actual cabe resaltar que el servicio de almacenamiento en bloque (Cinder) es fundamental para la operación de instancias de cómputo que requieran volúmenes persistentes y es necesario habilitar un *backend* de almacenamiento compatible. En este caso se utiliza LVM como *backend* de Cinder, el cual es compatible con el plan de almacenamiento definido previamente. Para hacerle saber a Kolla-Ansible que utilice LVM como *backend* de Cinder, se añadió la siguiente línea al archivo `globals.yml`.

Cuadro 7.45. Configuración de *backend* de Cinder en archivo de configuración global

```
cinder_backend: "lvm"
```

7.11. Configuraciones específicas de Kolla-Ansible

La configuración global es esencial para definir el comportamiento general de Kolla-Ansible, pero existen configuraciones específicas que pueden ser ajustadas según las necesidades del entorno. Las configuraciones de Kolla-Ansible permiten ser específicas a nivel de servicio o nodo, brindando flexibilidad para adaptar el despliegue a requisitos particulares. Para esta prueba de concepto no resulta necesario realizar configuraciones específicas adicionales, por la alta capacidad de ambos equipos físicos. Sin embargo, con el propósito de brindar una guía para esta clase de configuración, se describe el método para realizar ajustes específicos en Kolla-Ansible.

7.11.1. Creación de archivo de configuración específico

Kolla emplea la configuración global definida en `globals.yml` como base para el despliegue. Sin embargo, es posible crear archivos de configuración específicos para ajustar parámetros particulares y sobrescribirlos. Estos archivos se crean en el directorio `/etc/kolla/config/`. El directorio debe ser referenciado en el archivo `globals.yml` mediante el parámetro `kolla_config_dir`. Si este parámetro no está definido es probable que Kolla ignore las configuraciones adicionales que se hayan especificado.

Cuadro 7.46. Especificación de directorio de configuración en archivo de configuración global

```
kolla_config_dir: "/etc/kolla/config/"
```

Las configuraciones específicas deben seguir una estructura específica:

- Específicos a proyecto → `/etc/kolla/config/<nombre_servicio>/`
- Específicos a nodo→ `/etc/kolla/config/<nombre_servicio>/<nombre_nodo>/`

El nombre de nodo debe de coincidir exactamente con el definido en el archivo de inventario. Por ejemplo, para ajustar configuraciones específicas del servicio de Nova, se crea un archivo dentro del directorio `/etc/kolla/config/nova/`. Si se desea ajustar configuraciones específicas para el nodo de cómputo llamado `localhost`, se crea un archivo dentro del directorio `/etc/kolla/config/localhost/`. Los archivos de configuración deben seguir el formato estándar de archivos INI, donde las secciones y parámetros son definidos según la documentación oficial de cada servicio.

Configuración específica para Nova en nodo 1 (control y cómputo)

Se demuestra un caso práctico para ajustar configuración específica para el nodo principal. En esta configuración se limita el uso de recursos para el servicio de Nova, específico para el anfitrión local (nodo de control y cómputo). Se crea el directorio correspondiente y se edita el archivo `nova.conf` para ajustar los parámetros deseados.

Cuadro 7.47. Ejemplo de configuración específica para Nova

```
mkdir -p /etc/kolla/config/nova/localhost
nano /etc/kolla/config/nova/localhost/nova.conf
```

Se crea el directorio y archivo correspondiente para ajustar la configuración específica de Nova en el nodo local. En este caso, se limita la memoria reservada para el servicio de Nova a 8192 MB (8 GB) para asegurar que no consuma todos los recursos del sistema. Esto es personalizable según las necesidades del entorno y la capacidad del *hardware* disponible.

Cuadro 7.48. Contenido del archivo de configuración de Nova

```
[DEFAULT]
reserved_host_memory_mb = 8192
```

La configuración especificada en este archivo sobrescribe la configuración global definida en `globals.yml` para el servicio de Nova, pero solo para el nodo local. Otros nodos utilizarán la configuración global a menos que se definan configuraciones específicas adicionales para ellos.

Dado que una de las principales misiones de Kolla-Ansible es copiar archivos de configuración a los nodos remotos, es importante que el operador sepa cómo verificar que los archivos se hayan copiado correctamente. Para este propósito, se pueden inspeccionar la salida de los comandos de despliegue, donde se indica la copia de archivos a nodos específicos. La salida del comando muestra el rol que se está ejecutando, seguido de la descripción de

la tarea. Posteriormente, se muestra el estado de la tarea para cada nodo involucrado en el despliegue. Se muestra un ejemplo de salida de comando para identificar la copia de archivos de configuración durante la ejecución. Es posible forzar a Kolla-Ansible a mostrar la salida completa de los comandos empleando el parámetro `-vvv` al momento de la ejecución.

Cuadro 7.49. Ejemplo de salida de comando durante copia de archivos de configuración

```
TASK [nova : Copy nova.conf to compute nodes]**************************************
task path: /root/openstack/os-venv/lib/python3.12/site-packages/kolla_ansible/roles/
    nova/tasks/config.yml:19
ok: [openstack-compute] => (item=nova.conf)
"sources": [
    "/root/openstack/os-venv/share/kolla-ansible/ansible/roles/nova/templates/nova.
        conf.j2",
    "/etc/kolla/config/nova/localhost/nova.conf"
    "/etc/kolla/config/nova/nova.conf"
    ...
]
```

En este ejemplo se pueden apreciar las diferentes fuentes de configuración que ha descubierto Kolla para el archivo `nova.conf`. La segunda línea indica que Kolla ha detectado el archivo de configuración específico para el nodo local y lo ha incluido en la lista de fuentes. Esto confirma que la configuración específica ha sido reconocida y será aplicada durante el despliegue.

7.11.2. Generación de contraseñas

Kolla-Ansible utiliza el comando `kolla-genpwd` para generar contraseñas aleatorias para todos los servicios de Openstack el archivo `/etc/kolla/passwords.yml` contiene las contraseñas generadas. Es importante ejecutar este comando antes del despliegue para asegurar que todos los servicios tengan contraseñas seguras.

Cuadro 7.50. Generación de contraseñas

```
(os-venv) kolla-genpwd
```

Las contraseñas generadas con *strings* de texto aleatorios y cuentan con características de seguridad adecuadas para entornos de producción. La administración de estas contraseñas debe ser minuciosa, ya que son necesarias para la autenticación de los servicios desplegados. Para acceder a las contraseñas generadas, se puede consultar el archivo `/etc/kolla/passwords.yml`. En varias casos se requiere la contraseña de administador de Keystone, la cual se encuentra en el campo `keystone_admin_password`. Se puede acceder a este valor empleando el comando `grep`.

Cuadro 7.51. Consulta de contraseña de administrador de Keystone

```
grep keystone_admin_password /etc/kolla/passwords.yml
```

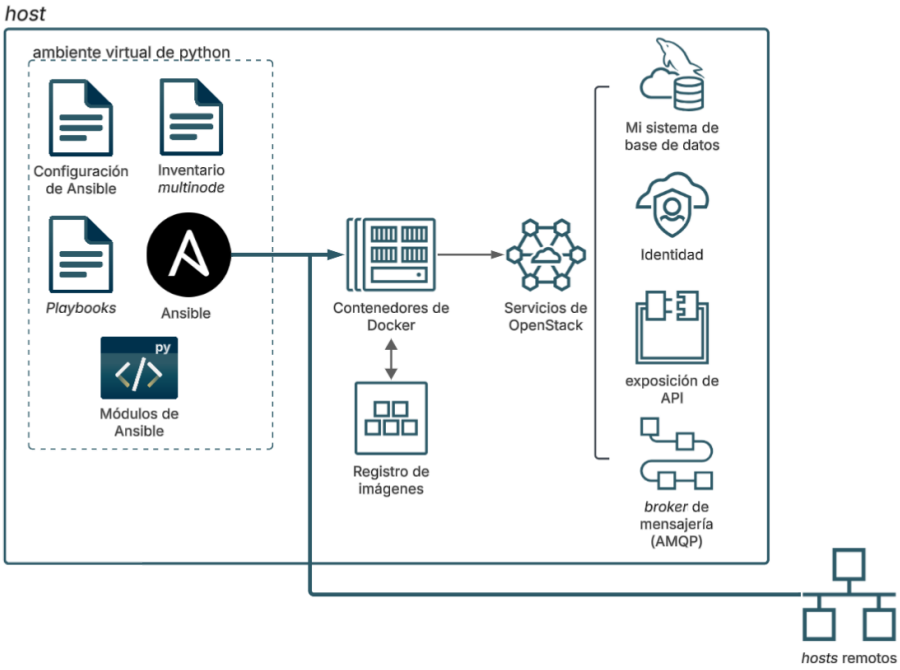
Alternativamente, se puede cambiar la contraseña de administrador de Keystone editando directamente el archivo `/etc/kolla/passwords.yml` y modificando el valor del campo `keystone_admin_password`. Es necesario contar con privilegios de superusuario para editar este archivo. La modificación de esta contraseña debe ser realizada con precaución, ya que afecta la autenticación de todos los servicios que dependen de Keystone. Adicionalmente,

el cambio de esta contraseña surte efecto únicamente si se realiza antes del despliegue de Openstack. Para la rotación de contraseñas en un entorno ya desplegado consultar sección de Operación de Openstack 8.1.

7.12. Despliegue de Openstack

Con la configuración completada, se procede al despliegue de Openstack mediante una serie de comandos de Kolla-Ansible que preparan los servidores, verifican requisitos previos y ejecutan el despliegue.

Figura 9. Organización de archivos de configuración para despliegue de Openstack con Kolla-Ansible



Nota. Imagen creada para reflejar estrategias de despliegue de Openstack con Kolla-Ansible e integración de servicios.

Durante la ejecución de los comandos, Ansible evoca una serie de *playbooks* que automatizan las tareas necesarias para preparar el entorno y desplegar los servicios de Openstack. Una de las misiones principales de estos *playbooks* es garantizar que la información de configuración proporcionada sea coherente y que el administrador pueda identificar posibles errores basado en la naturaleza de los comandos. La serie de tareas que se ejecutan en estos pasos suelen ser sumamente extensas y proporcionan detalles importantes para resolución de problemas en caso de fallos. El administrador debe prestar atención a puntos relevantes en la salida de los comandos. Se muestra un ejemplo de salida de comando para identificar campos clave durante la ejecución.

Cuadro 7.52. Ejemplo de salida de comando durante despliegue

```
TASK [loadbalancer : Checking if kolla_internal_vip_address and
   kolla_external_vip_address are not pingable from any node] ***
ok: [localhost]
ok: [openstack-compute]

TASK [loadbalancer : Fail if kolla_internal_vip_address is pingable
   from any node] ***
skipping: [localhost]
skipping: [openstack-compute]
```

En este ejemplo, se observa que la tarea de verificación de la dirección IP flotante interna (`kolla_internal_vip_address`) se ejecuta correctamente en ambos nodos, indicando que la dirección no es accesible desde ningún nodo, lo cual es el comportamiento esperado. Como se puede apreciar, la salida indica en la tarea el rol que se está ejecutando, en este caso `loadbalancer`, seguido de la descripción de la tarea. Posteriormente, se muestra el estado de la tarea para cada nodo involucrado en el despliegue. En este caso, ambas tareas resultaron en `ok`. Si alguno de los nodos hubiera podido acceder a la dirección IP flotante, la salida habría indicado un estado de `failed`, lo cual señala un error asociado a un nodo específico. Este tipo de información es crucial para la resolución de problemas durante el despliegue.

7.12.1. Verificación de requisitos previos

Se ejecuta el comando `prechecks` para verificar que todos los requisitos estén cumplidos antes del despliegue. Este comando valida configuraciones de red, disponibilidad de recursos (paquetes de aplicaciones disponibles, librerías de Python) y compatibilidad del sistema. La serie de *playbooks* que se ejecutan en este paso constituyen módulos estándar de Ansible y asegura que el entorno esté listo para el despliegue.

Cuadro 7.53. Verificación de requisitos previos

```
(os-venv) kolla-ansible prechecks -i multinode
```

Las tareas de verificación de requisitos previos no son específicas a roles de Openstack y se aplican a todos los nodos definidos en el archivo de inventario. Las tareas ejecutadas en esta sección pueden ser halladas en el directorio de instalación del entorno virtual, en la ruta `<directorio_de_trabajo>/os-venv/lib/python3.12/site-packages/kolla_-ansible/roles/prechecks/tasks/`. Es recomendable revisar estas tareas para entender qué aspectos del entorno están siendo verificados y cómo se relacionan con la configuración global y específica definida previamente, así como verificar que acciones específicas pueden activar un estado de error.

7.12.2. Preparación de servidores

Se ejecuta el comando `bootstrap-servers` para preparar los servidores, instalando dependencias necesarias y configurando Docker en los nodos definidos en el inventario. Aunque Docker y algunas librerías fueron instalados de forma manual, en la práctica se halla que

este comando asegura consistencia a lo largo de todos los ambientes y nodos involucrados en el despliegue. Los *tasks* específicos de paso pueden tener una estructura como la descrita en Cuadro 7.54.

Cuadro 7.54. Ejemplo de tarea de verificación de versión de Docker

```
- name: Checking docker SDK version
  command: "{{ ansible_facts.python.executable }} -c \"import docker; print(docker.
      __version__)\""
  register: result
  changed_when: false
  check_mode: false
  when:
    - inventory_hostname in groups['baremetal']
    - kolla_container_engine == 'docker'
  failed_when: result is failed or result.stdout is version(docker_py_version_min, '<
      ')
```

Como se puede observar en este ejemplo, Ansible emplea instrucciones condicionales para ejecutar tareas específicas basadas en la configuración del entorno. En este caso, la tarea verifica la versión del SDK de Docker instalado en los nodos que pertenecen al grupo `baremetal` y solo si el motor de contenedores configurado es Docker. Si la versión instalada no cumple con el requisito mínimo, la tarea falla, lo cual es crucial para asegurar que el entorno esté preparado adecuadamente para el despliegue de Openstack. Este tipo de verificaciones son comunes en las tareas de preparación de servidores y ayudan a garantizar que todos los nodos estén configurados correctamente antes de proceder con el despliegue.

7.12.3. Preparación de servidores

Se ejecuta el comando `bootstrap-servers` para preparar los servidores, instalando dependencias necesarias y configurando Docker en los nodos definidos en el inventario. Este paso es similar al comando de verificación de requisitos previos, pero se enfoca en la preparación activa de los nodos para el despliegue.

Cuadro 7.55. Preparación de servidores

```
(os-venv) kolla-ansible bootstrap-servers -i multinode
```

7.12.4. Ejecución del despliegue

Una vez verificados los requisitos, se ejecuta el comando `deploy` para iniciar el despliegue de los contenedores de Openstack.

Cuadro 7.56. Despliegue de Openstack

```
(os-venv) kolla-ansible deploy -i multinode
```

En este caso la extensión de la salida en consola viene en función de la cantidad de servicios habilitados y la cantidad de nodos involucrados en el despliegue. La salida del comando es extensa y detalla cada tarea que se ejecuta durante el despliegue, incluyendo la creación de contenedores, configuración de servicios y verificación de estados. Es importante

monitorear esta salida para identificar cualquier error o advertencia que pueda surgir durante el proceso. La descripción de estas tareas viene con especificación de roles de proyecto específico y nodos involucrados, similar a los ejemplos previos. Se nota que si no existen una instalación previa de Openstack en el sistema y se está realizando un despliegue inicial, el despliegue debe ser completamente exitoso sin errores.

Cuadro 7.57. Resumen de cambios realizados durante despliegue

```
PLAY RECAP *********************************************************
localhost                  : ok=456  changed=220  unreachable=0
   failed=0
openstack-compute          : ok=345  changed=150  unreachable=0
   failed=0
```

La salida del comando muestra un resumen de las tareas ejecutadas en cada nodo, indicando la cantidad de tareas que se completaron con éxito (`ok`), las que realizaron cambios en el sistema (`changed`), y si hubo nodos inalcanzables o fallidos. Un despliegue exitoso debe tener cero fallos y todos los nodos alcanzables.

7.12.5. Fallos abruptos durante el despliegue

La interrupción abrupta del proceso de despliegue puede dejar el entorno en un estado inconsistente. Si se interrumpe el despliegue, es recomendable ejecutar el comando `destroy` para limpiar los contenedores y recursos creados hasta ese punto. Esto se hace para evitar errores en sistemas de autenticación y dependencias que hayan quedado a medio configurar.

Cuadro 7.58. Limpieza de contenedores tras fallo abrupto

```
(os-venv) kolla-ansible destroy -i multinode --yes-i-really-really-
   mean-it
```

la especificación del parámetro `-yes-i-really-really-mean-it` es necesaria para confirmar que se desea proceder con la destrucción de los contenedores y recursos asociados.

7.13. Configuración post-despliegue de Openstack

Después de completar el despliegue, fue necesario realizar configuraciones adicionales para poder operar la nube de Openstack. Estas configuraciones incluyen la instalación del cliente de Openstack y la generación de archivos de credenciales que permiten la autenticación en el entorno desplegado. Estos pasos son necesarios para poder interactuar con los proyectos.

7.13.1. Instalación del cliente de Openstack

Se instaló el cliente de línea de comandos de Openstack dentro del entorno virtual, utilizando restricciones de versión compatibles con la rama `master`.

Cuadro 7.59. Instalación del cliente de Openstack

```
(os-venv) pip install python-openstackclient -c https://releases.
   openstack.org/constraints/upper/master
```

7.13.2. Generación de archivos de credenciales

Se ejecuta el comando `post-deploy` para generar el archivo de credenciales administrativas `/etc/kolla/admin-openrc.sh`, necesario para autenticarse en el entorno de Openstack.

Cuadro 7.60. Generación de archivos post-despliegue

```
(os-venv) kolla-ansible post-deploy
```

Para utilizar las credenciales generadas, se debe cargar el archivo `admin-openrc.sh` en la sesión actual de la terminal. Esto establece las variables de entorno necesarias para interactuar con Openstack. Esto se realiza mediante el comando `source`.

Cuadro 7.61. Carga de credenciales de administrador

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

CAPÍTULO 8

Operación de la nube privada Openstack

8.1. Operación de Openstack

Con el despliegue concretado hasta este punto es posible comenzar a operar la nube de Openstack. El estado actual de la nube responde directamente a las configuraciones efectuadas en el archivo de inventario y en la configuración global de Kolla-Ansible. Sin embargo, es deseable realizar verificaciones adicionales para asegurar que los proyectos estén operativos y que los recursos de infraestructura estén disponibles para su uso (como redes, volúmenes y máquinas virtuales). También se evalúa que los proyectos estén accesibles mediante el cliente de Openstack en línea de comando y en la interfaz gráfica Horizon. La operación de la nube se realiza considerando que se hayan cargado las credenciales administrativas en la sesión actual de la terminal y que el entorno virtual de Kolla-Ansible esté activado. También es importante verificar que el motor de Docker esté en ejecución, ya que es el componente central que gestiona los contenedores de Openstack.

8.2. Verificación de contenedores de Docker

La forma de verificar que los contenedores de Openstack estén en ejecución es mediante el comando `docker ps`. Este comando lista todos los contenedores activos en el sistema, mostrando información relevante como el identificador del contenedor, la imagen utilizada, el estado y los puertos expuestos. La actualización de las imágenes de los contenedores puede realizarse directamente empleando Kolla-Ansible, mediante el comando `pull`. Dada la naturaleza extensa del comando `docker ps`, se modifica para mostrar únicamente los nombres y estados de los contenedores activos. Se utiliza el comando `docker` con parámetros adicionales para mostrar estos datos organizados `docker ps --format "table {{.Names}}\t{{.Status}}"`.

Cuadro 8.1. Verificación de contenedores activos de Openstack

```
NAMES                           STATUS
grafana                         Up 2 weeks
prometheus_libvirt_exporter     Up 2 weeks
prometheus_blackbox_exporter    Up 2 weeks
prometheus_openstack_exporter   Up 2 weeks
prometheus_alertmanager         Up 2 weeks
prometheus_cadvisor             Up 2 weeks
prometheus_memcached_exporter   Up 2 weeks
prometheus_mysqld_exporter      Up 2 weeks
prometheus_node_exporter        Up 2 weeks
prometheus_server               Up 2 weeks
proxysql                        Up 2 weeks (healthy)
haproxy                         Up 2 weeks (healthy)
barbican_worker                 Up 6 weeks (healthy)
barbican_keystone_listener      Up 6 weeks (healthy)
barbican_api                    Up 6 weeks (healthy)
aodh_notifier                   Up 10 days (healthy)
aodh_listener                   Up 10 days (healthy)
aodh_evaluator                  Up 10 days (healthy)
aodh_api                        Up 10 days (healthy)
ceilometer_compute              Up 6 weeks (healthy)
ceilometer_central              Up 6 weeks (healthy)
ceilometer_notification         Up 6 weeks (healthy)
gnocchi_metricd                 Up 2 weeks (healthy)
gnocchi_api                     Up 2 weeks (healthy)
magnum_conductor                Up 3 weeks (healthy)
magnum_api                      Up 3 weeks (healthy)
horizon                         Up 6 weeks (healthy)
heat_engine                     Up 10 days (healthy)
heat_api_cfn                    Up 10 days (healthy)
heat_api                        Up 10 days (healthy)
neutron_metadata_agent          Up 6 weeks
neutron_l3_agent                Up 6 weeks (healthy)
neutron_dhcp_agent              Up 6 weeks (healthy)
neutron_openvswitch_agent       Up 6 weeks (healthy)
neutron_server                  Up 6 weeks (healthy)
nova_compute                    Up 2 weeks (healthy)
nova_libvirt                    Up 6 weeks (healthy)
nova_ssh                        Up 6 weeks (healthy)
nova_novncproxy                 Up 2 weeks (healthy)
nova_conductor                  Up 2 weeks (healthy)
nova_metadata                   Up 2 weeks (healthy)
nova_api                        Up 2 weeks (healthy)
nova_scheduler                  Up 2 weeks (healthy)
openvswitch_vswitchd            Up 6 weeks (healthy)
openvswitch_db                  Up 6 weeks (healthy)
placement_api                   Up 6 weeks (healthy)
cinder_backup                   Up 6 weeks (healthy)
cinder_volume                   Up 6 weeks (healthy)
cinder_scheduler                Up 6 weeks (healthy)
cinder_api                      Up 6 weeks (healthy)
glance_api                      Up 6 weeks (healthy)
keystone                        Up 2 weeks (healthy)
keystone_fernet                 Up 2 weeks (healthy)
keystone_ssh                    Up 6 weeks (healthy)
rabbitmq                        Up 2 weeks (healthy)
tgtd                            Up 6 weeks
iscsid                          Up 6 weeks
memcached                       Up 6 weeks (healthy)
mariadb                         Up 6 weeks (healthy)
keepalived                      Up 6 weeks
cron                            Up 2 weeks
kolla_toolbox                   Up 6 weeks
fluentd                         Up 2 weeks
```

8.3. Verificación de servicios activos

El estado de los servicios de Openstack debe reflejar el estado de los contenedores previamente verificados. Cada servicio de Openstack se ejecuta dentro de su respectivo contenedor, por lo que si los contenedores están activos y saludables, los servicios también estarán operativos.

8.3.1. Verificación de *endpoints*

Se utiliza el comando `openstack service list` para verificar que los servicios de Openstack estén activos y funcionando correctamente. La salida esperada del comando muestra una lista de servicios con su estado actual y la URL desde la que se puede consumir el servicio. Notar que la salida puede variar según los servicios habilitados en la configuración global de Kolla-Ansible. Notar que algunos de los campos de la salida han sido truncados para mejorar la legibilidad (ID de proyecto y URL completa de `cinderv3 y heat`).

Cuadro 8.2. Salida esperada para los servicios activos de Openstack

```
+--------+-----------+--------------+-----------------+---------+-----------+----------------------------------+
| ID     | Region    | Service Name | Service Type    | Enabled | Interface | URL                              |
+--------+-----------+--------------+-----------------+---------+-----------+----------------------------------+
| 0213...| RegionOne | barbican     | key-manager     | True    | internal  | http://192.168.74.69:9311        |
| 036b...| RegionOne | aodh         | alarming        | True    | public    | http://192.168.74.69:8042        |
| 0ac3...| RegionOne | glance       | image           | True    | public    | http://192.168.74.69:9292        |
| 0ba2...| RegionOne | heat         | orchestration   | True    | internal  | http://192.168.74.69:8004/v1...  |
| 17d6...| RegionOne | placement    | placement       | True    | internal  | http://192.168.74.69:8780        |
| 2134...| RegionOne | magnum       | container-infra | True    | public    | http://192.168.74.69:9511/v1     |
| 23a9...| RegionOne | glance       | image           | True    | internal  | http://192.168.74.69:9292        |
| 371f...| RegionOne | cinder       | block-storage   | True    | public    | http://192.168.74.69:8776/v3     |
| 4133...| RegionOne | keystone     | identity        | True    | public    | http://192.168.74.69:5000        |
| 49cb...| RegionOne | cinderv3     | volumev3        | True    | public    | http://192.168.74.69:8776/v3...  |
| 5c72...| RegionOne | neutron      | network         | True    | public    | http://192.168.74.69:9696        |
| 6460...| RegionOne | cinder       | block-storage   | True    | internal  | http://192.168.74.69:8776/v3     |
| 69ca...| RegionOne | heat-cfn     | cloudformation  | True    | internal  | http://192.168.74.69:8000/v1     |
| 6bda...| RegionOne | nova         | compute         | True    | internal  | http://192.168.74.69:8774/v2.1   |
| 71ee...| RegionOne | gnocchi      | metric          | True    | public    | http://192.168.74.69:8041        |
| 73b8...| RegionOne | nova         | compute         | True    | public    | http://192.168.74.69:8774/v2.1   |
| 75c2...| RegionOne | magnum       | container-infra | True    | internal  | http://192.168.74.69:9511/v1     |
| 772d...| RegionOne | placement    | placement       | True    | public    | http://192.168.74.69:8780        |
| 85f1...| RegionOne | neutron      | network         | True    | internal  | http://192.168.74.69:9696        |
| a52e...| RegionOne | heat-cfn     | cloudformation  | True    | public    | http://192.168.74.69:8000/v1     |
| b212...| RegionOne | aodh         | alarming        | True    | internal  | http://192.168.74.69:8042        |
| c147...| RegionOne | heat         | orchestration   | True    | public    | http://192.168.74.69:8004/v1...  |
| de97...| RegionOne | barbican     | key-manager     | True    | public    | http://192.168.74.69:9311        |
| e7b2...| RegionOne | cinderv3     | volumev3        | True    | internal  | http://192.168.74.69:8776/v3...  |
| e824...| RegionOne | gnocchi      | metric          | True    | internal  | http://192.168.74.69:8041        |
| f052...| RegionOne | keystone     | identity        | True    | internal  | http://192.168.74.69:5000        |
+--------+-----------+--------------+-----------------+---------+-----------+----------------------------------+
```

En este comando también revela la URL de acceso a cada servicio, la cual debe ser coherente con la configuración realizada en el campo `kolla_internal_vip_address` del archivo `globals.yml`, editado en la sección 7.10.

8.3.2. Verificación de proveedor de recursos

Se utiliza el comando `openstack resource provider list` para verificar que los proveedores de recursos estén activos y funcionando correctamente. La salida esperada del comando muestra una lista de proveedores de recursos con su estado actual y UUID.

Cuadro 8.3. Salida esperada para los proveedores de recursos activos de Openstack

```
+--------+-------------------+------------+--------------------+----------------------+
| uuid   | name              | generation | root_provider_uuid| parent_provider_uuid |
+--------+-------------------+------------+--------------------+----------------------+
| ab8... | openstack-infra   |        162 | ab80d639-5aa7-... | None                 |
| e97... | openstack-compute |          1 | e97ac6d1-8308-... | None                 |
+--------+-------------------+------------+--------------------+----------------------+
```

Los UUID de los proveedores de recursos también pueden ser aprovechados para mostrar información detallada sobre los recursos disponibles en cada proveedor, utilizando el comando `openstack resource provider show <UUID> VCPU`.

Cuadro 8.4. Salida esperada para los detalles de un proveedor de recursos

```
+-------------------+-------+
| Field             | Value |
+-------------------+-------+
| allocation_ratio  | 4.0   |
| min_unit          | 1     |
| max_unit          | 112   |
| reserved          | 0     |
| step_size         | 1     |
| total             | 112   |
| used              | 5     |
+-------------------+-------+
```

8.3.3. Verificación de hipervisores

Se utiliza el comando `openstack hypervisor list` para verificar que los hipervisores de Openstack estén activos y funcionando correctamente. La salida esperada del comando muestra una lista de hipervisores con su estado actual, tipo y *host* asociado.

Cuadro 8.5. Salida esperada para los hipervisores activos de Openstack

```

+-------+----------------------+------------------+--------------+-------+
| ID    | Hypervisor Hostname  | Hypervisor Type  | Host IP      | State |
+-------+----------------------+------------------+--------------+-------+
| ab8...| openstack-infra      | QEMU             | 192.168.74.5 | up    |
| e97...| openstack-compute    | QEMU             | 192.168.74.7 | down  |
+-------+----------------------+------------------+--------------+-------+
```

8.3.4. Verificación de agentes de red

Se utiliza el comando `openstack network agent list` para verificar que los agentes de red de Openstack estén activos y funcionando correctamente. La salida esperada del comando muestra una lista de agentes de red con su estado actual, tipo y host asociado.

Cuadro 8.6. Salida esperada para los agentes de red activos de Openstack

```
+-------+--------------------+--------+-------------------+-------+-------+---------------------------+
| ID    | Agent Type         | Host   | Availability Zone | Alive | State | Binary                    |
+-------+--------------------+--------+-------------------+-------+-------+---------------------------+
| 021...| L3 agent           | deploy | nova              | :-)   | UP    | neutron-l3-agent          |
| 2f9...| Metadata agent     | deploy | None              | :-)   | UP    | neutron-metadata-agent    |
| 7d6...| Open vSwitch agent | deploy | None              | :-)   | UP    | neutron-openvswitch-agent |
| ac8...| DHCP agent         | deploy | nova              | :-)   | UP    | neutron-dhcp-agent        |
+-------+--------------------+--------+-------------------+-------+-------+---------------------------+
```

8.4. Cambio de configuración

En caso de ser necesario realizar cambios en la configuración de Openstack después del despliegue inicial, Kolla-Ansible facilita este proceso. Los cambios pueden involucrar la adición o eliminación de servicios, la modificación de parámetros de configuración o la actualización de imágenes de contenedores. Se distinguen dos tipos de cambios: aquellos que

requieren la destrucción y recreación de contenedores, y aquellos que pueden aplicarse en caliente sin necesidad de reiniciar los servicios.

8.4.1. Cambios en archivos de configuración sin afectar servicios (*test*)

Es posible realizar cambios en los archivos de configuración de Kolla-Ansible y verificar su impacto sin afectar los servicios en ejecución. Esto se logra utilizando el comando `kolla-ansible genconfig` para generar los archivos de configuración actualizados basados en los cambios realizados. Esto le da al operador certeza de que los cambios en archivos de configuración son válidos y no contienen errores de sintaxis. Posteriormente, se puede utilizar el comando `kolla-ansible validate-config` para validar la configuración generada. Este comando verifica que los archivos de configuración cumplan con los requisitos y estándares establecidos por Kolla-Ansible, asegurando que no haya conflictos o inconsistencias que puedan afectar el despliegue de Openstack. La salida de este comando es especificada por medio de mensajes de *log* en el directorio `/var/log/kolla/config-validate`.

Cuadro 8.7. Generación y validación de archivos de configuración

```
(os-venv) kolla-ansible genconfig
(os-venv) kolla-ansible validate-config
nano /var/log/kolla/config-validate/2024-05-01_12-00-00.log
```

8.4.2. Cambios que requieren destrucción de contenedores

Para cambios significativos en la configuración que afectan a los servicios de Openstack, es necesario destruir y recrear los contenedores afectados. Esto se logra utilizando el comando `kolla-ansible destroy` y ejecutando nuevamente la serie de comandos de despliegue: `prechecks`, `pull`, `deploy` y `post-deploy`. Este proceso asegura que los cambios se apliquen correctamente y que los servicios estén actualizados con la nueva configuración.

Cuadro 8.8. Secuencia para aplicar cambios que requieren destrucción de contenedores

```
(os-venv) kolla-ansible destroy --yes-i-really-really-mean-it
(os-venv) kolla-ansible prechecks
(os-venv) kolla-ansible pull
(os-venv) kolla-ansible deploy
(os-venv) kolla-ansible post-deploy
```

8.4.3. Cambios en caliente

Algunos cambios menores en la configuración pueden aplicarse en caliente sin necesidad de destruir los contenedores. Estos cambios incluyen la modificación de parámetros de configuración que no afectan la estructura fundamental de los servicios. Para aplicar estos cambios, se utiliza el comando `kolla-ansible reconfigure`, que actualiza la configuración de los contenedores en ejecución. Este enfoque minimiza el tiempo de inactividad y permite una gestión más ágil de la nube de Openstack, ideal para ambientes que requieren alta

disponibilidad. Además las reconfiguraciones pueden ser aplicadas a servicios específicos utilizando el comando `kolla-ansible reconfigure -tags <servicio>`, donde `<servicio>` es el nombre del servicio que se desea reconfigurar.

Cuadro 8.9. Reconfiguración de un servicio específico

```
(os-venv) kolla-ansible reconfigure --tags nova
```

Para este mismo propósito, es posible utilizar el comando `deploy` con la opción `-tags <servicio>`, lo que permite aplicar cambios específicos sin afectar otros servicios, pero será más lento.

8.4.4. Mantenimiento de contenedores

Kolla-Ansible también proporciona herramientas para el mantenimiento de los contenedores de Openstack. Esto incluye la actualización de las imágenes de los contenedores mediante el comando `kolla-ansible pull`, que descarga las últimas versiones de las imágenes desde el repositorio oficial de Kolla. Además, es posible reiniciar contenedores específicos utilizando el comando `docker restart <nombre_contenedor>`, lo que permite aplicar cambios menores sin necesidad de un despliegue completo. También es posible optimizar el uso de recursos del sistema mediante la limpieza de contenedores y volúmenes no utilizados con el comando `kolla-ansible prune`. Para operaciones de mantenimiento en cuánto al servidor respecta, es recomendable detener el motor de Docker antes de realizar tareas que puedan afectar la integridad de los contenedores. Kolla-Ansible cuenta con el comando `kolla-ansible stop` para detener todos los servicios de Openstack de manera ordenada, seguido del comando `systemctl stop docker` para detener el motor de Docker. Una vez finalizadas las tareas de mantenimiento, se debe volver a desplegar la nube utilizando el comando `kolla-ansible deploy`.

Cuadro 8.10. Secuencia para mantenimiento del servidor

```
(os-venv) kolla-ansible stop
sudo systemctl stop docker
# Realizar tareas de mantenimiento aquí
(os-venv) kolla-ansible deploy
```

8.5. Creación de proyectos y recursos

Con la verificación de los servicios y componentes de Openstack completada, es posible comenzar a crear proyectos y recursos dentro de la nube privada. Esto incluye la creación de proyectos, usuarios, redes, volúmenes y máquinas virtuales, entre otros recursos. La creación y gestión de estos recursos se puede realizar tanto a través de la interfaz gráfica Horizon como mediante el cliente de Openstack en línea de comando. Con fines de simplificación, se ejemplifica el acceso a la interfaz gráfica Horizon para la gestión de todos los recursos de la nube.

La clave de acceso a la interfaz gráfica que fue generada de forma automática puede ser hallada en el archivo `/etc/kolla/passwwords.yml` en el nodo `deploy`, bajo la sección

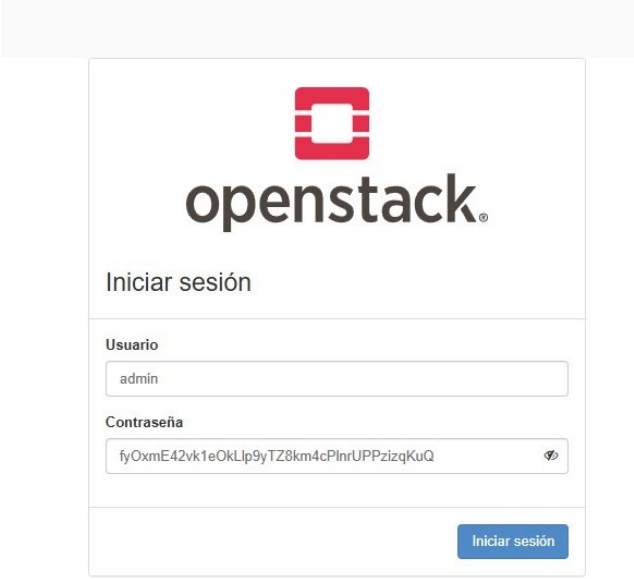
`horizon_keystone_admin_password`. Esta clave es necesaria para iniciar sesión en la interfaz gráfica Horizon con el usuario administrativo `admin`. Resulta conveniente emplear el comando `grep` para localizar rápidamente la clave dentro del archivo de contraseñas. Este comando se corre con privilegios de superusuario para asegurar el acceso al archivo protegido.

Para acceder a la interfaz gráfica Horizon, se abre un navegador web y se ingresa la dirección IP del VIP interno configurado en Kolla-Ansible, seguido de la ruta `/horizon`. Esto redirige al usuario a la página de inicio de sesión de Horizon, donde se ingresan las credenciales administrativas para acceder al panel de control de Openstack. Para el despliegue realizado en este proyecto, la URL de acceso es `http://192.168.74.69/horizon`.

Cuadro 8.11. Extracción de la clave administrativa de Horizon

```
grep horizon_keystone_admin_password /etc/kolla/passwords.yml
horizon_keystone_admin_password: fy0...
```

Figura 10. Inicio de sesión en la interfaz gráfica Horizon de Openstack



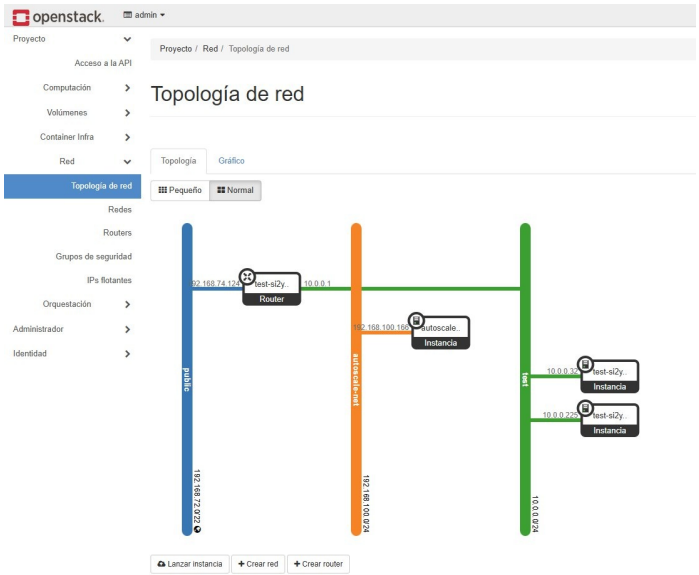
Nota. Captura de pantalla del inicio de sesión en la interfaz gráfica Horizon de Openstack.

Una vez dentro de la interfaz gráfica Horizon, se puede gestionar todos los aspectos de la nube privada Openstack sin necesidad de utilizar la línea de comando.

Figura 11. Interfaz gráfica Horizon de Openstack

Nota. Captura de pantalla de los recursos de computación en la interfaz gráfica Horizon de Openstack.

Figura 12. Vista general de topología de red en Horizon



Nota. Captura de pantalla de la topología de red en la interfaz gráfica Horizon de Openstack.

CAPÍTULO 9

Conclusiones

- Se documenta el proceso de preparación de recursos de infraestructura para un despliegue rápido y escalable de OpenStack empleando *scripts* de automatización con Ansible, garantizando una configuración consistente para futuras implementaciones.
- Se configuran los servicios esenciales de OpenStack para el despliegue de máquinas virtuales. Asegurando la integración adecuada entre los componentes de red, almacenamiento y computación.
- Se configuran recursos de red de los equipos de laboratorio para que los servicios de OpenStack puedan exponer sus funcionalidades a través de *endpoints* especificados en la configuración de Kolla-Ansible y puedan ser accesibles desde la red local.
- Se instalan archivos de configuración necesarios para que la gestión de la nube de OpenStack pueda ser centralizada a partir de un nodo local designado como controlador.
- Se modifican archivos de configuración predeterminados de Kolla-Ansible para la configuración de la nube de OpenStack en diversos servidores *baremetal*.
- Se aprovechan *scripts* de Ansible para la generación automática de claves de acceso a los servicios de OpenStack, facilitando la autenticación y autorización de usuarios y servicios dentro de la nube.
- Se configura el servicio para gestión con interfaz web Horizon, permitiendo a los usuarios administrar recursos de la nube de OpenStack a través de un portal web intuitivo.
- Se aprovechan *playbooks* de Ansible para operaciones de validación de estado de contenedores, actualización de imágenes y optimización de recursos en el *host*.

CAPÍTULO 10

Recomendaciones

1. Dado que se ha adecuado el entorno para demostrar una estrategia escalable empleando infraestructura como código, se recomienda explorar la integración de nodos adicionales en el *cluster* de Openstack utilizando Kolla-Ansible. Esto permitiría evaluar la capacidad de escalabilidad y gestión dinámica de recursos en un entorno de nube privada.

2. Integrar registros de Docker es una práctica común para optimizar la gestión de imágenes de contenedores. Se recomienda configurar un registro privado de Docker para almacenar y distribuir las imágenes personalizadas utilizadas en el despliegue de Openstack. Esto mejorará la eficiencia a medida que se agreguen más nodos al *cluster*.

3. Considerar la implementación de diversos *backends* de almacenamiento, como Ceph o NFS, para evaluar su rendimiento y compatibilidad con los servicios de Openstack desplegados. Esto permitirá optimizar la gestión del almacenamiento en función de las necesidades específicas del entorno y ofrecer soluciones de redundancia.

4. Explorar un despliegue que sea específico a los roles específicos de los nodos, como nodos de control, nodos de cómputo y nodos de almacenamiento aislados. Esta estrategia permitirá optimizar la asignación de recursos y mejorar el rendimiento general del *cluster* de Openstack.

5. Segmentar la red utilizando VLAN o redes superpuestas (VXLAN) para mejorar la seguridad y el aislamiento del tráfico de red entre los diferentes servicios de Openstack. Esta práctica es especialmente relevante en entornos multiusuario o cuando se manejan datos sensibles.

6. Implementar despliegues en entornos virtualizados KVM o VMware para evaluar la flexibilidad y adaptabilidad de la solución en diferentes infraestructuras subyacentes. Esto permitirá validar la portabilidad del despliegue de Openstack y su capacidad para operar en diversos entornos.

CAPÍTULO 11

Referencias

[1] F. Turcios, «Despliegue modular de la plataforma de cloud computing OpenStack en la red de laboratorios del Departamento de Electrónica de la Universidad del Valle de Guatemala,» Universidad del Valle de Guatemala, 2024.

[2] P. Masek, M. Stusek, J. Krejci, K. Zeman, J. Pokorny y M. Kudlacek, «Unleashing Full Potential of Ansible Framework: University Labs Administration,» en *2018 22nd Conference of Open Innovations Association (FRUCT)*, 2018, págs. 144-150. DOI: 10.23919/FRUCT.2018.8468270.

[3] B. Jiang, Z. Tang, X. Xiao, J. Yao, R. Cao y K. Li, «Efficient and Automated Deployment Architecture for OpenStack in TianHe SuperComputing Environment,» *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, n.º 8, págs. 1811-1824, 2022. DOI: 10.1109/TPDS.2021.3127128.

[4] V. Gonzalez, C. Castillo y F. Lopez-Pires, «An Elastic VoIP Solution based on OpenStack,» abr. de 2016. DOI: 10.1109/ICISE.2016.8.

[5] G. Quattrocchi y D. A. Tamburri, « Infrastructure as Code,» *IEEE Software*, vol. 40, n.º 01, págs. 37-40, ene. de 2023, ISSN: 1937-4194. DOI: 10.1109/MS.2022.3212034. dirección: https://doi.ieeecomputersociety.org/10.1109/MS.2022.3212034.

[6] I. Ahmad, «Cloud Computing - A Comprehensive Definiton,» *Journal of Computing and Management Studies*, vol. 1, ene. de 2017.

[7] K. Morris, *Infrastructure as Code*. O'Reilly Media, Inc., 2021, ISBN: 978-1-098-11467-1.

[8] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero y D. A. Tamburri, «DevOps: Introducing Infrastructure-as-Code,» en *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, págs. 497-498. DOI: 10.1109/ICSE-C.2017.162.

[9] P. Mell y T. Grance, «The NIST Definition of Cloud Computing,» National Institute of Standards y Technology, Gaithersburg, MD, inf. téc. Special Publication 800-145, sep. de 2011. dirección: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[10] J. Humble y D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010, ISBN: 9780321601919.

[11] H. Alshaer, «An overview of network virtualization and cloud network as a service,» *International Journal of Network Management*, vol. 25, n.º 1, págs. 1-30, 2015. DOI: https://doi.org/10.1002/nem.1882. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.1882. dirección: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1882.

[12] J. F. Kurose y K. W. Ross, *Computer Networking: A Top-Down Approach (4th Edition)*, 4.ª ed. Addison Wesley, 2007, ISBN: 0321497708. dirección: http://www.amazon.com/Computer-Networking-Top-Down-Approach-4th/dp/0321497708%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0321497708.

[13] Q. Duan, N. Ansari y M. Toy, «Software-defined network virtualization: an architectural framework for integrating SDN and NFV for service provisioning in future networks,» *IEEE Network*, vol. 30, n.º 5, págs. 10-16, 2016. DOI: 10.1109/MNET.2016.7579021.

[14] J. K. Peckol, *Embedded Systems: A Contemporary Design Tool*, 1st. Wiley Publishing, 2007, ISBN: 0471721808.

[15] S. Sotiriadis, N. Bessis, F. Xhafa y N. Antonopoulos, «Cloud Virtual Machine Scheduling: Modelling the Cloud Virtual Machine Instantiation,» en *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, IEEE, 2012. DOI: 10.1109/CISIS.2012.113.

[16] W. Felter, A. Ferreira, R. Rajamony y J. Rubio, «An updated performance comparison of virtual machines and Linux containers,» en *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2015. DOI: 10.1109/ISPASS.2015.7095802.

[17] N. Zhou, Y. Georgiou, L. Zhong, H. Zhou y M. Pospieszny, «Container Orchestration on HPC Systems,» en *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, IEEE, 2020. DOI: 10.1109/CLOUD49709.2020.00017.

[18] C. Negus, *Docker Containers: Build and Deploy with Kubernetes, Flannel, Cockpit, and Atomic*. Packt Publishing, 2015.

[19] «OpenStack cloud operating system,» 2023. dirección: https://www.openstack.org/software/.

[20] Y. Niu, F. Liu y Z. Li, «Load Balancing Across Microservices,» en *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, págs. 198-206. DOI: 10.1109/INFOCOM.2018.8486300.

[21] R. T. Fielding, «Architectural Styles and the Design of Network-based Software Architectures,» Doctoral dissertation, Tesis doct., University of California, Irvine, Irvine, CA, 2000. dirección: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[22] OpenStack Foundation, *OpenStack, Keystone Documentation Release 27.1.0.dev52*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/keystone/latest/.

[23] OpenStack Foundation, *OpenStack, Glance Documentation Release 30.1.0.dev46*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/glance/latest/.

[24] OpenStack Foundation, *OpenStack, Placement Documentation Release 13.1.0.dev5*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/placement/latest/.

[25] OpenStack Foundation, *OpenStack, Nova Documentation Release 31.1.0.dev91*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/nova/latest/.

[26] OpenStack Foundation, *OpenStack, Neutron Documentation Release 26.1.0.dev180*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/neutron/latest/.

[27] OpenStack Foundation, *OpenStack, Cinder Documentation Release 26.1.0.dev50*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/cinder/latest/.

[28] OpenStack Foundation, *OpenStack, Horizon Documentation Release 25.4.1.dev11*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/horizon/latest/.

[29] G. Fu, Y. Zhang y G. Yu, «A Fair Comparison of Message Queuing Systems,» *IEEE Access*, vol. 9, págs. 421-432, 2021. DOI: 10.1109/ACCESS.2020.3046503.

[30] Red Hat, Inc., *Ansible Documentation*, Accedido el 6 de junio de 2025, 2025. dirección: https://www.ansible.com/.

[31] S. Thakur, S. C. Gupta, N. Singh y S. Geddam, «Mitigating and Patching System Vulnerabilities Using Ansible: A Comparative Study of Various Configuration Management Tools for IAAS Cloud,» en *Information Systems Design and Intelligent Applications*, S. C. Satapathy, J. K. Mandal, S. K. Udgata y V. Bhateja, eds., New Delhi: Springer India, 2016, págs. 21-29, ISBN: 978-81-322-2755-7.

[32] M. Mohaan y R. Raithatha, *Learning Ansible*. Packt Publishing, 2014, ISBN: 1783550635.

[33] A. Mallett, «Writing YAML and Basic Playbooks,» en *Red Hat Certified Engineer (RHCE) Study Guide: Ansible Automation for the Red Hat Enterprise Linux 8 Exam (EX294)*. Berkeley, CA: Apress, 2021, págs. 63-77, ISBN: 978-1-4842-6861-2. DOI: 10.1007/978-1-4842-6861-2_5. dirección: https://doi.org/10.1007/978-1-4842-6861-2_5.

[34] T. L. F. LSB Workgroup, *Filesystem Hierarchy Standard*, https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html, Accessed June 5, 2025, 2015.

[35] OpenStack Foundation, *OpenStack, Kolla-Ansible Release 20.1.0.dev27*, Accedido el 6 de junio de 2025, 2025. dirección: https://docs.openstack.org/horizon/latest/.

CAPÍTULO 12

Glosario

AMQP Protocolo abierto para la mensajería asíncrona entre aplicaciones y servicios, facilita la comunicación confiable y escalable.

Ansible Herramienta de automatización de TI que facilita la configuración, gestión y despliegue de aplicaciones y servicios en múltiples sistemas de manera sencilla y eficiente.

API Interfaz de programación de aplicaciones, un conjunto de definiciones y protocolos que permiten la comunicación entre diferentes aplicaciones o servicios.

application runtimes Entornos de ejecución que proporcionan las bibliotecas y servicios necesarios para ejecutar aplicaciones, facilitando la gestión de dependencias y la compatibilidad.

AWS Plataforma de servicios en la nube que ofrece una amplia gama de servicios de computación, almacenamiento y redes para empresas y desarrolladores. Desarrollado por Amazon.

Azure Propuesta de Microsoft para plataforma de servicios en la nube que proporciona soluciones de computación, almacenamiento y redes para empresas y desarrolladores.

backends Parte de una aplicación o sistema que maneja la lógica de negocio, el procesamiento de datos y la interacción con bases de datos, generalmente no visible para los usuarios finales.

balanceador de carga Dispositivo o *software* que distribuye el tráfico de red o las solicitudes de servicio entre múltiples servidores para optimizar la utilización de recursos, mejorar el rendimiento y garantizar la alta disponibilidad.

bare metal Un servidor físico dedicado que no utiliza una capa de virtualización, proporcionando acceso directo al *hardware* para un rendimiento óptimo.

binarios Archivos ejecutables que contienen código máquina que puede ser directamente interpretado por el procesador de una computadora.

bridges Dispositivos de red virtuales que conectan múltiples interfaces de red.

broadcast Dominio de red en el que un mensaje enviado por un dispositivo es recibido por todos los demás dispositivos conectados a la misma red.

broker Intermediario que facilita la comunicación y el intercambio de mensajes entre diferentes aplicaciones o servicios en un sistema distribuido.

***clusters* de contenedores** Conjunto de nodos que ejecutan aplicaciones en contenedores, gestionados de manera coordinada para proporcionar alta disponibilidad, escalabilidad y eficiencia en el uso de recursos.

contenedores Tecnología de virtualización a nivel de sistema operativo que permite ejecutar aplicaciones y sus dependencias en entornos aislados y portátiles, compartiendo el mismo núcleo del sistema operativo.

CRUD Acrónimo de crear, leer, actualizar y borrar, que representa las operaciones básicas que se pueden realizar en una base de datos o sistema de almacenamiento de datos.

Debian Una distribución de Linux conocida por su estabilidad, seguridad y amplia comunidad de usuarios y desarrolladores.

desarrollo y operaciones Conjunto de prácticas que combinan el desarrollo de *software* y las operaciones de TI para mejorar la colaboración, la automatización y la entrega continua de aplicaciones y servicios.

DHCP Protocolo de configuración dinámica de *host* que asigna automáticamente direcciones IP y otros parámetros de red a dispositivos en una red para facilitar la comunicación.

Docker Plataforma de código abierto que automatiza la implementación, escalado y gestión de aplicaciones mediante contenedores, permitiendo a los desarrolladores empaquetar aplicaciones con todas sus dependencias en un entorno aislado y portátil.

endpoints Puntos de acceso a un servicio o aplicación, donde los clientes pueden interactuar con la funcionalidad proporcionada.

exchange Componente de un sistema de mensajería que recibe mensajes de los productores y los enruta a las colas correspondientes según reglas de vinculación.

firewalls Dispositivos o *software* de seguridad que monitorean y controlan el tráfico de red entrante y saliente, protegiendo las redes y sistemas contra accesos no autorizados y amenazas.

frameworks Conjunto de herramientas y bibliotecas que proporcionan una estructura predefinida para desarrollar aplicaciones de manera más eficiente.

GCP Propuesta de Google para una plataforma de servicios en la nube que ofrece soluciones de computación, almacenamiento y redes para empresas y desarrolladores.

hipervisor Paquete de *software* que crea y gestiona máquinas virtuales, permitiendo que múltiples sistemas operativos se ejecuten simultáneamente en un solo *hardware* físico.

HTTP Protocolo de transferencia de hipertexto, un protocolo de comunicación utilizado para la transferencia de datos en la web entre clientes y servidores.

imágenes Archivos que contienen todo lo necesario para ejecutar una aplicación en un contenedor, incluyendo el código, las bibliotecas y las dependencias.

iptables Herramienta de línea de comandos en sistemas Linux que permite configurar y gestionar las reglas del cortafuegos (*firewall*) para controlar el tráfico de red entrante y saliente.

kernel Núcleo del sistema operativo que gestiona los recursos del sistema y facilita la comunicación entre el *hardware* y el *software*.

microservicios Arquitectura de desarrollo de *software* que estructura una aplicación como un conjunto de servicios pequeños, independientes y desplegables de manera autónoma, cada uno enfocado en una funcionalidad específica.

mirroring Técnica utilizada para replicar datos o servicios en múltiples ubicaciones o nodos para mejorar la disponibilidad y la tolerancia a fallos.

máquina virtual Un entorno de computación simulado que emula un sistema informático completo, permitiendo ejecutar sistemas operativos y aplicaciones de manera aislada del *hardware* físico.

namespaces Mecanismo de aislamiento en sistemas operativos que permite separar y gestionar recursos del sistema, como procesos, redes y sistemas de archivos, para diferentes entornos o aplicaciones.

NAT Traducción de direcciones de red, un proceso que permite modificar las direcciones IP en los paquetes de datos para facilitar la comunicación entre redes privadas y públicas.

parsing Proceso de análisis y descomposición de datos o código para extraer información significativa y estructurada.

playbooks Archivos de configuración utilizados en Ansible para definir tareas y automatizar la gestión de sistemas y aplicaciones.

Python Lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su sintaxis clara y legibilidad.

routers Dispositivos de red que encaminan paquetes de datos entre diferentes redes, gestionando el tráfico y asegurando la comunicación eficiente entre dispositivos conectados.

servers Programas o dispositivos que proporcionan servicios, recursos o datos a otros programas o dispositivos, conocidos como clientes, a través de una red.

stack Conjunto de tecnologías, herramientas y servicios que trabajan juntos para desarrollar, implementar y gestionar aplicaciones y sistemas informáticos.

switches Dispositivos de red que conectan múltiples dispositivos dentro de una misma red local (LAN), permitiendo la comunicación eficiente mediante el envío de datos solo al dispositivo destinatario.

Terraform Herramienta de infraestructura como código que permite definir y gestionar recursos de infraestructura en la nube mediante archivos de configuración declarativos.

tokens Elementos de seguridad utilizados para autenticar y autorizar el acceso a sistemas, aplicaciones o servicios, garantizando que solo los usuarios o entidades autorizadas puedan interactuar con ellos.

union filesystem Sistema de archivos que permite combinar múltiples sistemas de archivos en una sola vista lógica, facilitando la gestión y el acceso a los datos.

veth Interfaz de red virtual que conecta dos espacios de nombres de red, permitiendo la comunicación entre contenedores o entre un contenedor y el *host*.

virtualización Tecnología que permite crear una versión virtual de un recurso físico, como un servidor, almacenamiento o red, para mejorar la eficiencia y flexibilidad en la gestión de recursos informáticos.