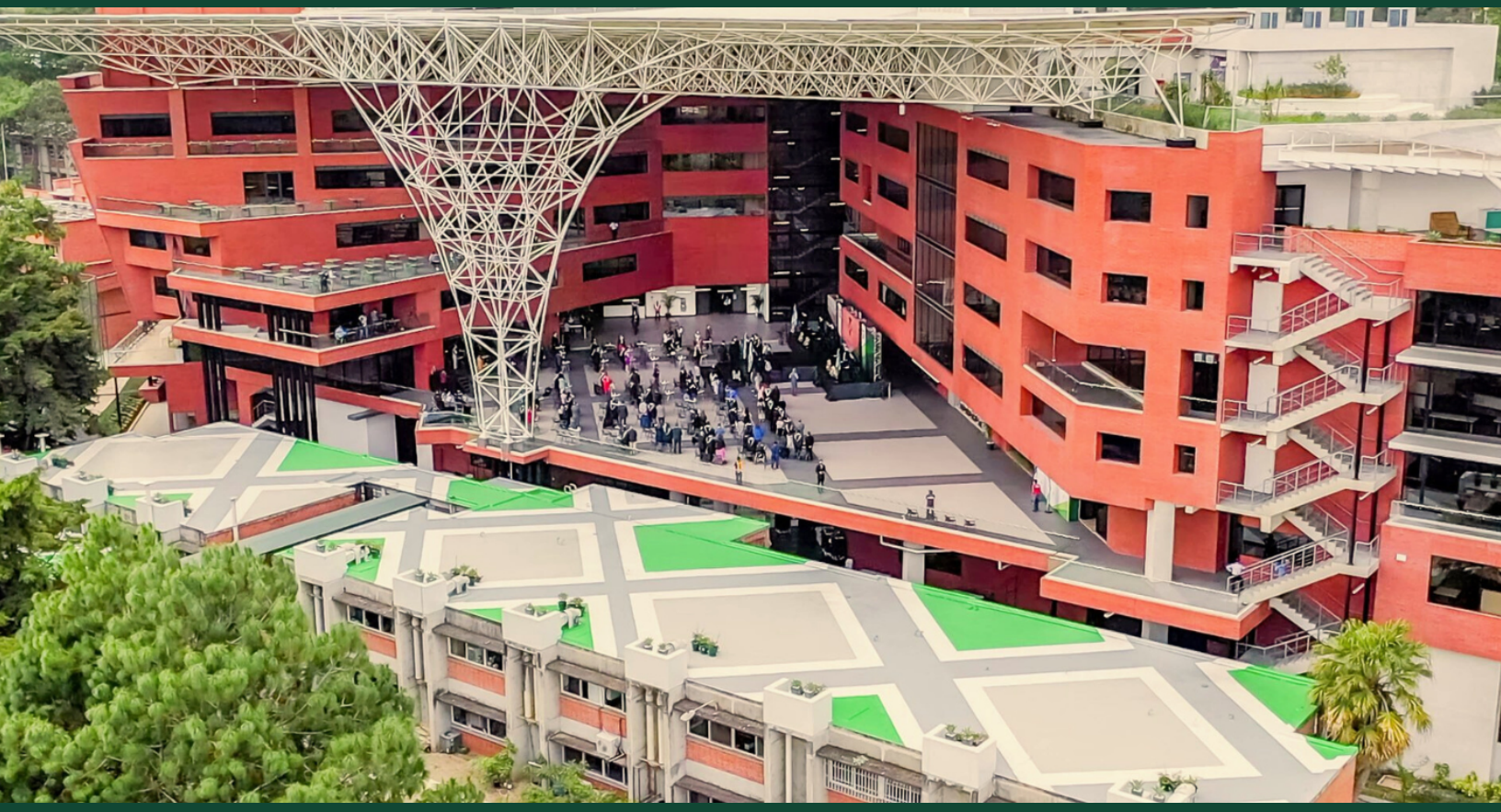
Método alternativo de implementación de un clúster de alto rendimiento de OpenStack en la Universidad del Valle de Guatemala

Pedro Pablo Luna Gutiérrez



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Método alternativo de implementación de un clúster de alto rendimiento de OpenStack en la Universidad del Valle de Guatemala

Trabajo de graduación presentado por Pedro Pablo Luna Gutiérrez 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

Deseo expresar mi más profundo agradecimiento a mi familia, en especial a mi madre, quien ha sido mi mayor sustento y apoyo incondicional a lo largo de este proceso. Su esfuerzo, cariño y ejemplo han sido fundamentales para alcanzar esta meta.

Agradezco también a mis amigos, quienes con su compañía y ánimo me ayudaron a mantener la motivación en los momentos más difíciles.

Finalmente, extiendo mi gratitud a los catedráticos de la Universidad del Valle de Guatemala que contribuyeron significativamente a mi formación académica y profesional.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El presente trabajo tuvo como propósito la implementación de un clúster de cómputo basado en la plataforma OpenStack, con el fin de optimizar el uso de los recursos de cómputo disponibles en el Departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala. Para ello, se replicó el despliegue modular previamente desarrollado en la HPC Precision 7920 dentro de una segunda computadora de alto rendimiento, la HPC PowerEdge T560.

El proyecto se centró en la creación de una infraestructura distribuida administrada por un único *controller node*, que permitió la gestión unificada de múltiples nodos de cómputo bajo una misma instancia de OpenStack. Como parte del proceso, se configuraron los servicios fundamentales de la plataforma —Keystone, Nova, Neutron, Glance, Horizon, Cinder y Placement— y se integró el servicio de gestión de claves Barbican para fortalecer la seguridad de la nube.

La metodología aplicada consistió en la instalación, configuración, validación y documentación de cada componente, siguiendo un enfoque modular que garantizó la replicabilidad del sistema. Se realizaron pruebas de funcionamiento para evaluar el despliegue de instancias, la conectividad de red y la correcta asignación de recursos en el clúster.

Como resultado, se obtuvo una infraestructura funcional capaz de centralizar la administración de recursos de cómputo y habilitar un entorno escalable y seguro para futuras investigaciones y aplicaciones dentro del ámbito académico. Además, se identificaron líneas de trabajo futuro relacionadas con la incorporación de almacenamiento compartido o distribuido, la mejora de la tolerancia a fallos mediante arreglos redundantes, el endurecimiento del *backend* criptográfico de Barbican y la integración de nuevos servicios de OpenStack, lo que abre la puerta a investigaciones posteriores en alta disponibilidad, automatización y seguridad en nubes privadas.

Palabras clave: Openstack, clúster de cómputo, infraestructura en la nube, virtualización, Barbican.

Abstract

The present work aimed to implement a computing cluster based on the OpenStack platform to optimize the use of the computing resources available in the Electronics Engineering Department at Universidad del Valle de Guatemala. To achieve this, the modular deployment previously developed on the Precision 7920 HPC was replicated on a second high-performance computer, the PowerEdge T560.

The project focused on creating a distributed infrastructure managed by a single *Controller Node*, which enabled unified management of multiple compute nodes under a single OpenStack instance. As part of the process, the core platform services —Keystone, Nova, Neutron, Glance, Horizon, Cinder, and Placement— were configured, and the Barbican key management service was integrated to strengthen cloud security.

The methodology consisted of installing, configuring, validating, and documenting each component, following a modular approach that ensured system replicability. Functional tests were carried out to assess instance deployment, network connectivity, and correct resource allocation within the cluster.

As a result, a functional infrastructure was obtained, capable of centralizing the administration of computing resources and enabling a scalable and secure environment for future research and applications in the academic context. Furthermore, future work lines were identified, including the incorporation of shared or distributed storage, improved fault tolerance through redundant disk setups, hardening of Barbican's cryptographic backend, and the integration of additional OpenStack services. These directions open the door to subsequent research on high availability, automation, and security in private cloud environments.

Keywords: Openstack, computing cluster, cloud infrastructure, virtualization, Barbican.

CAPÍTULO 1

Introducción

El presente trabajo se enmarca dentro del campo de la computación en la nube y la infraestructura de alto rendimiento (HPC), áreas que combinan la virtualización de recursos con la optimización de *hardware* especializado para ofrecer entornos flexibles, escalables y eficientes. En particular, el estudio se centra en el uso de OpenStack, una plataforma de código abierto ampliamente utilizada para la gestión de nubes privadas y públicas, la cual permite administrar recursos de cómputo, almacenamiento y red de forma centralizada.

El objetivo principal de este proyecto es implementar un clúster de cómputo basado en OpenStack que integre dos computadoras de alto rendimiento (HPC) pertenecientes al Departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala. Las computadoras son una Precision 7920 y una PowerEdge T560, bajo la administración de un único *controller node*. Con ello, se busca optimizar el uso de los recursos de cómputo disponibles, aumentar la capacidad de procesamiento y facilitar la gestión centralizada de servicios de nube.

El tema se delimita al proceso de replicación del despliegue modular previo de OpenStack, su adaptación al nuevo *hardware* disponible y la posterior configuración del entorno de clúster. No se abordarán implementaciones adicionales de módulos de orquestación o balanceo avanzado, ya que el enfoque se mantiene en la consolidación funcional del clúster y en la incorporación del servicio Barbican, encargado de la gestión segura de claves y datos sensibles dentro de la infraestructura.

La metodología seguida contempla una secuencia de etapas técnicas: la instalación y configuración inicial de los nodos de cómputo y del nodo controlador, la integración de los servicios de OpenStack para formar el clúster, la validación mediante despliegue de instancias virtuales y, finalmente, la documentación detallada de todos los procedimientos realizados. Cada fase se desarrolla de manera modular para permitir la replicabilidad del proceso en otros entornos académicos o de investigación.

Entre las principales conclusiones se destaca que la implementación de un clúster de cómputo a través de OpenStack permite aprovechar de forma más eficiente los recursos de

hardware existentes, mejorando el rendimiento y la disponibilidad de los servicios virtualizados. Asimismo, la incorporación de Barbican refuerza la seguridad de la infraestructura, mientras que la documentación generada constituye una guía práctica para futuros proyectos que busquen expandir o replicar entornos de nube académicos con características similares.

CAPÍTULO 2

Antecedentes

La virtualización y la computación en la nube son pilares para la optimización de recursos computacionales en entornos académicos y científicos. En este contexto, el trabajo desarrollado por Francisco Turcios [1] representa un aporte significativo en la adopción de tecnologías de código abierto en instituciones educativas, mediante la implementación modular de una nube privada basada en OpenStack sobre una infraestructura de cómputo de alto rendimiento (HPC).

En su proyecto, Turcios plantea como objetivo principal la construcción de una plataforma flexible, escalable y replicable, capaz de administrar recursos de cómputo a través de la virtualización, permitiendo así el despliegue de entornos computacionales a demanda mediante máquinas virtuales. Esta propuesta fue desarrollada sobre una *workstation* Dell Precision 7920, configurada como nodo de cómputo y nodo de almacenamiento. El nodo controlador se implementó en una máquina virtual en la cual se realizó la instalación y configuración de los principales módulos que conforman el ecosistema de OpenStack.

Uno de los aspectos clave del trabajo es su enfoque modular, que permite entender y construir cada componente de la infraestructura por separado, facilitando tanto la implementación como la posterior administración de la plataforma. Cada módulo fue instalado de manera manual y documentada, permitiendo un entendimiento detallado de las funciones que cumple cada servicio. Los módulos utilizados fueron Keystone (para la autenticación e identidad de usuarios), Nova (motor de cómputo), Neutron (gestión de redes), Glance (servicio de imágenes), Cinder (almacenamiento en bloques), Horizon (interfaz gráfica) y Placement (asignación de recursos). En cuanto al almacenamiento, se implementó la solución utilizando LVM (*logical volume manager*) para los volúmenes persistentes y el almacenamiento compartido entre servicios. Asimismo, se configuraron servicios auxiliares críticos como la base de datos MariaDB, el sistema de colas RabbitMQ, y memcached, los cuales permiten la operación eficiente y coordinada de los servicios de OpenStack [1].

CAPÍTULO 3

Justificación

Tras demostrar la versatilidad y utilidad de OpenStack para la Universidad del Valle de Guatemala en el trabajo de graduación titulado "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" [1], y aprovechando la adquisición de una segunda computadora de alto rendimiento (servidor PowerEdge T560), se busca potenciar al máximo ambos equipos mediante la implementación de un clúster de cómputo.

Este trabajo se realiza con el propósito de optimizar la gestión de recursos computacionales en el departamento, proporcionando una infraestructura más robusta, escalable y eficiente. La problemática principal que resuelve es la actual fragmentación en el uso de los recursos, ya que con un clúster se podrá administrar desde un único nodo controlador múltiples servicios de *cloud computing*, integrando y centralizando la operación de los equipos disponibles. También llevará a un mejor aprovechamiento del *hardware*, reducirá el riesgo de fallos en los servicios, y habilitará la posibilidad de escalar la infraestructura de manera más sencilla en el futuro.

En términos académicos, este proyecto brindará a estudiantes y docentes una plataforma práctica para la enseñanza, investigación y experimentación en temas de virtualización, redes, seguridad y *cloud computing*, alineándose con las tendencias actuales de la industria tecnológica. Las posibilidades incluyen la ejecución de simulaciones y análisis que requieran grandes capacidades de cómputo, abriendo la puerta a apoyar proyectos multidisciplinarios dentro de la universidad. También fomenta el acceso a tecnología de vanguardia en un entorno académico, preparando a los futuros ingenieros con experiencia práctica en herramientas que tienen alta demanda en el sector laboral [2].

La implementación se llevará a cabo replicando el despliegue modular de OpenStack previamente desarrollado, integrando los servicios principales (Keystone, Nova, Neutron, Glance, Horizon y Placement), y reconfigurándolos para habilitar la administración centralizada de ambos nodos de cómputo. Finalmente, se validará la interoperabilidad mediante la creación de instancias virtuales y la integración de módulos adicionales como Barbican, asegurando el funcionamiento seguro y estable del clúster.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Realizar un clúster de cómputo a través de la plataforma OpenStack, replicando el despliegue modular en la HPC PowerEdge T560 y configurandola para unificarla a la HPC Precision 7920 logrando que funcionen bajo la administración de un solo *controller node*, optimizando el uso de recursos de cómputo en el Departamento de Ingeniería Electrónica.

4.2. Objetivos específicos

- Replicar completamente el *compute node* 1 en el *compute node* 2.
- Documentación del proceso de configuración del clúster en *compute node* 1 y *compute node* 2.
- Configurar un único *controller node* centralizado que administre los recursos de ambos nodos.
- Desplegar instancias virtuales para evaluar el funcionamiento del clúster.
- Implementar y configurar el OpenStack *key manager service* Barbican.

CAPÍTULO 5

Alcance

El proyecto consiste en crear un clúster de cómputo a través de la plataforma OpenStack, integrando las HPC Precision 7920 y PowerEdge T560 bajo la administración de un único nodo controlador. Para ello, se realizará la réplica del despliegue modular previamente documentado [1], asegurando que ambas máquinas funcionen de manera conjunta como parte de una misma infraestructura administrada centralmente. Todo se llevará a cabo entre enero y noviembre del año 2025. En este proceso se busca optimizar el uso de los recursos disponibles en el Departamento de Ingeniería Electrónica y también ampliar las capacidades del despliegue original mediante la incorporación del servicio Barbican para la gestión segura de claves. Además de la réplica completa del nodo de cómputo original, se busca la documentación detallada de todo el proceso de configuración del clúster, la creación y validación de un único nodo controlador, el despliegue de instancias virtuales que permitan verificar el funcionamiento del clúster, y la instalación de Barbican como servicio de seguridad adicional. No se incluye la implementación de servicios adicionales de OpenStack fuera de los definidos, ni la integración de nuevos recursos de hardware que excedan los ya disponibles.

En cuanto a los recursos, el proyecto cuenta con dos computadoras de alto rendimiento propiedad de la universidad: la HPC Precision 7920, que actualmente aloja de manera virtualizada al nodo controlador y a uno de los nodos de cómputo, y la HPC PowerEdge T560, en la cual se instalará el segundo nodo de cómputo a partir de la copia del disco del despliegue original. También se dispone de VMWare Workstation como herramienta para la virtualización de nodos en la Precision 7920. Sin embargo, no se cuenta con discos duros adicionales en la PowerEdge T560 para implementar un arreglo redundante que garantice la alta disponibilidad del nodo, ni con un sistema de almacenamiento externo dedicado, por lo que la tolerancia a fallos será limitada.

CAPÍTULO 6

Marco teórico

6.1. Clúster

6.1.1. Definición

Un clúster es una agrupación de procesadores en configuraciones paralelas. En un clúster, la asignación de recursos es realizada por un sistema centralizado de administración de recursos y planificación de tareas. Todos los nodos de un clúster trabajan cooperativamente como un único recurso unificado. Los nodos son independientes y están interconectados. Todas las interacciones del usuario con un clúster deben pasar a través de un sistema centralizado que gestione la asignación de recursos a los trabajos de las aplicaciones [3].

6.1.2. Componentes de un clúster

Baker et al. [4] señalan algunos de los más importantes componentes en un clúster de cómputo:

- Múltiples computadoras de alto rendimiento.
- Sistemas Operativos de última generación.
- Redes/*switches* de alto rendimiento (como *GigabitEthernet*).
- *Network interface cards* (NIC).
- Protocolos y servicios de comunicación rápidos.
- Ambientes y herramientas de programación paralelos como *compilers*, PVMs (*parallel virtual machines*) y MPI (*message passing interface*).

6.1.3. Clasificación

Hay múltiples tipos de clúster según el uso que se les de y la manera en que se configure. Las principales carácteristicas que ofrecen son las siguientes:

- Alto rendimiento: diseñados para ejecutar tareas que requieren gran capacidad de procesamiento, como simulaciones científicas, análisis de grandes volúmenes de datos y/o cálculos numéricos muy complejos. Estos clústeres emplean procesamiento paralelo entre múltiples nodos para reducir los tiempos de ejecución y maximizar la eficiencia.
- Expansión y escalabilidad: este clúster tiene una arquitectura que permite aumentar las capacidades de cómputo o almacenamiento mediante la incorporación de nuevos nodos, sin afectar la estabilidad o el rendimiento del sistema completo. Es ideal en entornos donde se busca crecimiento progresivo o variaciones en la demanda de recursos.
- Alto rendimiento de transferencia (*throughput*): clúster optimizado para ejecutar un gran número de tareas individuales en paralelo, en lugar de un único proceso distribuido. Es ideal para cargas de trabajo de múltiples tareas independientes, como procesamiento por lotes (*batch*), análisis masivo de archivos o automatización de tareas repetitivas.
- Alta disponibilidad: diseñado para garantizar la continuidad del servicio, incluso ante fallos de hardware o software. Mediante la configuración de redundancias de nodos y mecanismos de conmutación por error (*failover)*, minimizando el tiempo de inactividad, lo que es importante en entornos críticos donde la disponibilidad constante es indispensable.

La tecnología de clústeres permite a organizaciones aumentar su capacidad de procesamiento utilizando tecnología estándar (componentes de hardware y software de uso común) que pueden adquirirse a un costo relativamente bajo. Esto proporciona capacidad de expansión. El rendimiento de las aplicaciones también mejora con el soporte de un entorno de software escalable. Esto implica la capacidad de recuperación ante fallos (*failover*), que permite a una computadora de respaldo asumir las tareas de otra que ha fallado dentro del mismo clúster [4].

6.1.4. Configuración de nodos

Según Heath et al. [5] los nodos de un clúster pueden ser de arquitecturas de dos tipos:

- Clúster homogéneo: todos los nodos tienen una arquitectura similar y mismos sistemas operativos.
- Clúster heterogéneo: todos los nodos tienen una arquitectura distinta y distintos sistemas operativos.

En el caso de este trabajo se tendrá un clúster de tipo heterogéneo en términos de *hardware*, pues las HPC de las que dispone la universidad son completamente diferentes (PowerEdge

T560 y Precison 7920). En términos de *software* eran homogéneos, pues ya que parte del proceso es replicar un trabajo previo [1], se utilizó el mismo sistema operativo (Ubuntu 22.04 LTS) en ambas HPC, sin embargo, dado a un cambio que tuvo que hacerse durante ese proceso de réplica, se optó por cambiar el sistema operativo de la PowerEdge T560 (Ubuntu 24.04 LTS), haciendo que el clúster sea heterogéneo en todo aspecto.

6.1.5. Capacidades de memoria

- Red de RAM: uso de la memoria asociada a cada *workstation* como una memoria caché DRAM (*dynamic random-access memory*) agregada; esto mejora el rendimeinto de la memoria virtual y del sistema de archivos. Esto también da la posibilidad de migrar una tarea de una *workstation* con memoria insuficiente a otra que tenga una menor carga [6].
- *Software/hardware* RAID (*redundant array of inexpensive disks*): uso de los arreglos de discos de las *workstations* para proporcionar almacenamiento de archivos económico, altamente disponible y escalable. Se logra haciendo arreglos redundantes de discos conectados a través de una red LAN (*software*) como respaldo o dentro de un mismo equipo *hardware*. Los RAID de *hardware* y *software* tienen un performance similar, exceptuando para operaciones de escritura que requieren de sincronización de archivos [7].
- *Disk dump*: proceso de clonar o copiar completamente el contenido de un dispositivo de almacenamiento, como un disco duro o una partición, hacia un archivo o hacia otro medio físico. Este procedimiento se utiliza comúnmente para realizar copias de respaldo, migraciones o replicaciones exactas de sistemas operativos, configuraciones y datos, conservando tanto la estructura lógica como los sectores físicos del disco original. En entornos de infraestructura y administración de sistemas, este proceso ayuda a replicar una instalación preconfigurada sin necesidad de reinstalar manualmente el sistema operativo o los servicios, garantizando así la integridad de las configuraciones y la consistencia entre múltiples máquinas [8]. Técnicamente, la operación puede realizarse mediante herramientas como dd, partclone, Clonezilla o qemu-img, que leen los bloques del disco de origen y los escriben en una imagen de salida o en otro dispositivo de almacenamiento. En el contexto de sistemas Linux, el comando dd es uno de los más empleados, ya que permite realizar *disk dumps* bit a bit, generando una réplica exacta del medio fuente [9].

6.2. OpenStack

OpenStack es un sistema operativo de nube de código abierto y altamente escalable, resultado de una colaboración global entre desarrolladores y tecnólogos en computación en la nube, que produce una plataforma de nube abierta y ubicua para nubes públicas y privadas. El objetivo de OpenStack es virtualizar y automatizar todos los aspectos de una infraestructura computacional corporativa. Su propósito es proporcionar acceso en modo autoservicio a todos los recursos que tradicionalmente requerían la intervención de varios equipos para ser gestionados [10]. Controla grandes grupos de recursos de almacenamiento,

cómputo y red en todo un centro de datos, todo ello gestionado a través de un panel de control (Horizon) que da a los administradores el control, mientras permite a los usuarios aprovisionar recursos mediante una interfaz web. Principalmente existen tres proyectos de *software* fundamentales, que son: OpenStack Compute Infrastructure (Nova), OpenStack Object Storage Infrastructure (Swift) y OpenStack Image Service Infrastructure (Glance) [11]. Básicamente, OpenStack entrega una solución del tipo Infraestructura como Servicio (IaaS) a través de sus varios y complementarios módulos. Cada módulo ofrece una interfaz de programación de aplicaciones (API) que facilita la integración [12].

6.2.1. Módulos

- Barbican: servicio de gestión de claves (*key manager*) de OpenStack. Proporciona almacenamiento seguro, aprovisionamiento y gestión de datos secretos. Esto incluye material criptográfico como claves simétricas, claves asimétricas, certificados y datos binarios sin procesar. Proporciona una API REST que permite a los servicios internos de OpenStack —como Cinder, Glance y Nova— almacenar, recuperar y administrar información sensible sin necesidad de exponerla directamente en archivos de configuración o en texto plano [13].

- Cinder: módulo que gestiona el almacenamiento en bloques, permitiendo crear, administrar y adjuntar volúmenes persistentes a las máquinas virtuales. Estos volúmenes funcionan como discos duros adicionales que pueden ser formateados, montados y utilizados por las instancias. Cinder también soporta la creación de instantáneas (*snapshots*) de volúmenes o instancias, las cuales pueden utilizarse para respaldo o como origen de arranque. Es compatible con diversos *backends* de almacenamiento, tanto por *software* (como LVM, Ceph o GlusterFS) como por *hardware* mediante plugins específicos, lo que le otorga gran flexibilidad para adaptarse a diferentes entornos y necesidades de infraestructura.

- Glance: servicio de OpenStack encargado de la gestión de imágenes de disco virtuales utilizadas por las máquinas virtuales dentro del entorno de nube. Funciona como un catálogo de recursos digitales, permitiendo a los usuarios almacenar, registrar, consultar y recuperar imágenes de sistemas operativos y otros archivos asociados [14]. Proporciona una API RESTful que facilita tanto la consulta de metadatos de las imágenes como la descarga de los archivos de imagen propiamente dichos. Las imágenes gestionadas por Glance pueden almacenarse en distintos *backends*, que van desde sistemas de archivos simples hasta soluciones de almacenamiento de objetos [15].

- Horizon: módulo que proporciona una interfaz gráfica basada en la *web* para la gestión de los servicios de la nube. Inicialmente desarrollado como una aplicación para administrar el módulo de cómputo, Horizon evolucionó para interactuar con todos los servicios principales de OpenStack, como Nova, Swift y Keystone, entre otros. Su diseño permite que los administradores y usuarios gestionen recursos y configuraciones de manera intuitiva, centralizando las operaciones para no requerir comandos de consola y conocimientos técnicos avanzados [16]. Horizon organiza las funciones a través de un panel de control (*dashboard*), brindando soporte para crear y administrar máquinas virtuales, redes, volúmenes de almacenamiento y usuarios. Gracias a su estructura extensible y modular se facilita la configuración y el monitoreo de la infraestructura en

la nube [17].

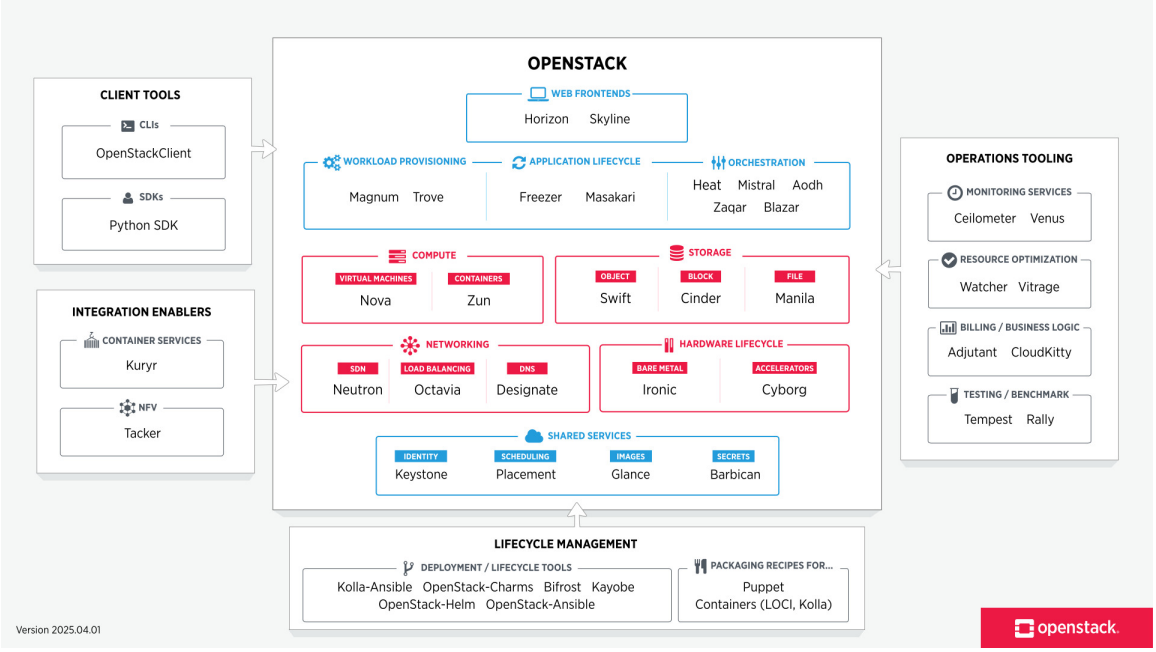
- Keystone: módulo de OpenStack encargado de proporcionar servicios de autenticación, autorización y gestión de identidades dentro del entorno de nube. Funciona como el componente central que permite verificar la identidad de los usuarios, servicios y proyectos (*tenants*), y es responsable de generar y validar los *tokens* de acceso utilizados por los distintos servicios de la plataforma [18].

- Neutron: servicio que se encarga de proporcionar conectividad de red como servicio para la infraestructura de nube. Permite crear y gestionar redes virtuales, subredes y enrutadores, simulando la estructura y funciones de redes físicas dentro del entorno de OpenStack. Neutron habilita la creación de topologías de red avanzadas, incluyendo servicios como firewalls y redes privadas virtuales (VPN), ofreciendo así gran flexibilidad a la hora de diseñar la conectividad de las máquinas virtuales y otros dispositivos gestionados por OpenStack [12]. Además, Neutron soporta el uso de *plugins* para integrar diferentes equipos y soluciones de red, lo que facilita su adaptación a las necesidades específicas de cada infraestructura. En su arquitectura, Neutron gestiona tanto la infraestructura de redes virtuales (VNI) como la capa de acceso de la Infraestructura de redes físicas (PNI), coordinando la comunicación entre las máquinas virtuales y los recursos de red externos [18].

- Nova: módulo de cómputo de OpenStack, encargado de proveer la funcionalidad de aprovisionamiento de instancias de cómputo (también conocidas como servidores virtuales) en la nube. Permite la creación y gestión de máquinas virtuales, servidores *baremetal* (mediante Ironic) y, de forma limitada, contenedores de sistema. Nova opera como un conjunto de procesos que se ejecutan en segundo plano sobre servidores Linux, interactuando con mecanismos de virtualización a través de *drivers* específicos. Este módulo proporciona el control de instancias, redes y acceso de usuarios y proyectos en una plataforma de computación tipo IaaS, exponiendo su funcionalidad mediante una API *web*. Para su funcionamiento básico, Nova depende de otros servicios de OpenStack, como Keystone (autenticación), Glance (repositorio de imágenes), Neutron (provisión de redes) y Placement (inventario y asignación de recursos) [19]. Su diseño busca ser agnóstico en cuanto a *hardware* e hipervisores, lo que le permite adaptarse a distintos entornos y servir de base para nubes de cómputo a gran escala, como la Rackspace Open Cloud [18].

- Placement: servicio encargado de gestionar y rastrear la disponibilidad, asignación y consumo de recursos computacionales dentro de una nube, a través de una API RESTful bien definida [20]. Originalmente desarrollado como parte del módulo Nova, y luego separado como un servicio independiente. Placement permite a diferentes módulos (Nova, Neutron, Cyborg, etc.) registrar sus propios recursos y describirlos mediante clases cuantitativas y características cualitativas (*traits*). Este servicio es esencial en procesos como la programación de instancias, ya que permite a Nova-Scheduler seleccionar *hosts* de destino adecuados para las cargas de trabajo según requisitos de recursos, agrupaciones y topologías [21].

Figura 1. Módulos de OpenStack y sus funcionalidades



Nota. La imagen muestra el *framework* modular de OpenStack y el como se relacionan entre sí los servicios. Esta imagen fue obtenida de [22].

6.2.2. Arquitectura

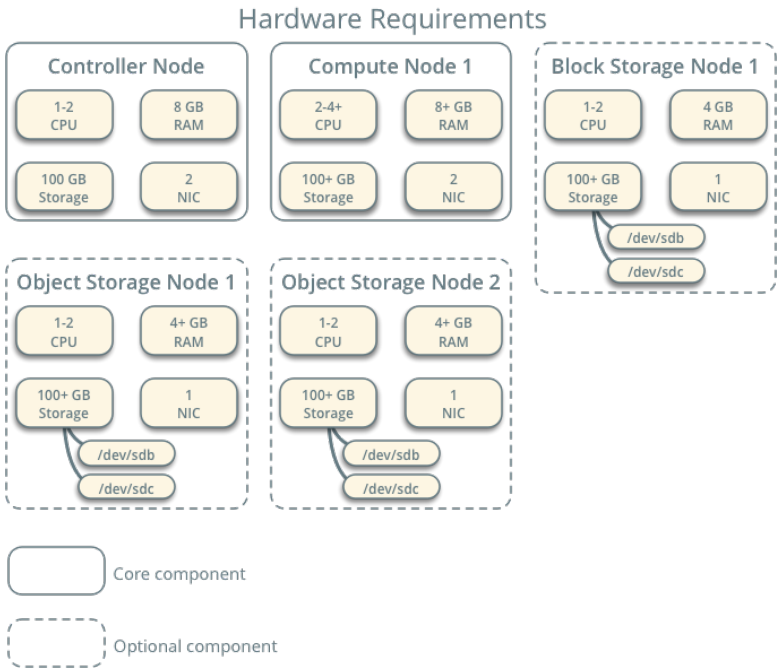
OpenStack usa una arquitectura con nodos, los cuales son máquinas físicas y/o virtuales que son provisionados con sistemas operativos y con algún o algunos *software* ejecutandose encima [23]. Los nodos mínimos para una arquitectura funcional son dos: nodo controlador y nodo de cómputo. También es bastante común agregar distintos tipos de nodos de almacenamiento, especificamente, nodos que almacenan objetos y nodos que almacenan bloques, sin embargo estos son completamente opcionales [24].

- *Controller node*: núcleo de la infraestructura OpenStack. Se encarga de gestionar los principales servicios de administración y control de la nube. Ejecuta servicios como Keystone, Glance, Placement, las partes de administración de Nova y Neutron, así como agentes de red y el *dashboard* Horizon. Además, almacena servicios de soporte como bases de datos SQL, colas de mensajes y sincronización horaria. Opcionalmente, puede incluir partes de los servicios de almacenamiento en bloques, almacenamiento de objetos, orquestación y telemetría. El nodo controlador requiere al menos dos interfaces de red para manejar de manera eficiente las conexiones internas y externas.
- *Compute node*: componente que ejecuta las instancias de máquinas virtuales dentro de la infraestructura. Se encarga de la virtualización utilizando por defecto el hipervisor KVM, aunque es compatible con otros hipervisores. Además, incorpora un agente de red que conecta las instancias a las redes virtuales y proporciona servicios de *firewall*

mediante grupos de seguridad. Este nodo requiere al menos dos interfaces de red para garantizar la conectividad de las instancias y de la propia infraestructura. Se pueden desplegar múltiples nodos de cómputo para escalar la capacidad de procesamiento de la nube.

- *Block storage node*: responsable de alojar los discos que proveen almacenamiento persistente a las instancias a través de los servicios de Cinder y del sistema de archivos compartidos. Aunque en implementaciones sencillas se puede compartir la red de gestión con los nodos de cómputo, en entornos de producción se recomienda una red de almacenamiento dedicada para mejorar el rendimiento y la seguridad. Se pueden desplegar múltiples nodos para aumentar la capacidad y la disponibilidad del almacenamiento.
- *Object storage node*: Almacena los discos utilizados por el servicio de almacenamiento de objetos para almacenar cuentas, contenedores y objetos. Similar al nodo de almacenamiento en bloques, el tráfico de servicios puede compartirse con la red de gestión en implementaciones básicas, aunque en entornos de producción se recomienda una red de almacenamiento separada. Este servicio requiere al menos dos nodos para operar y cada uno necesita al menos una interfaz de red. También pueden desplegarse más de dos nodos para mejorar la capacidad de almacenamiento y la disponibilidad.

Figura 2. Ejemplo de arquitectura OpenStack para levantar una máquina virtual o instancia



Nota. La imagen muestra un ejemplo de una arquitectura común a la hora de usar OpenStack para lanzar máquinas virtuales, mostrando qué nodos son opcionales y cuales son necesarios. Esta imagen fue obtenida de [25].

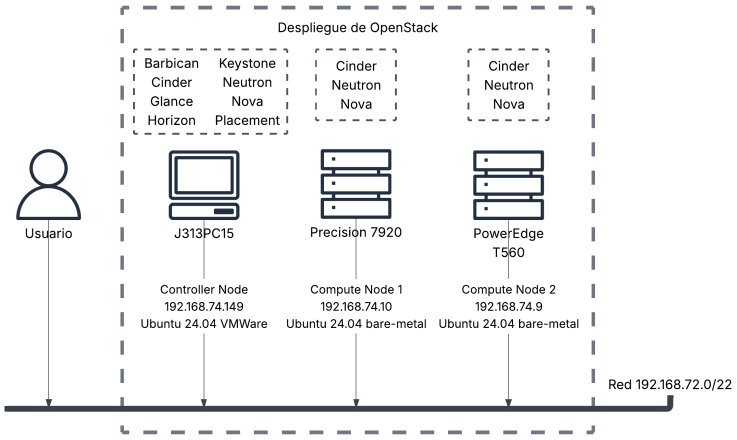
CAPÍTULO 7

Resultados

7.1. Desarrollo del clúster

Para el desarrollo del clúster se replicó completamente el despliegue de OpenStack anterior [1], tanto el nodo controlador, como el nodo de cómputo. El nodo controlador se replicó en VMWare debido a complicaciones con VirtualBox vinculado a limitaciones que impone Windows. El nodo de cómputo se replicó sobre *hardware*, no se virtualizó. La única diferencia contra el nodo de cómputo previamente realizado es la IP, pues esta vez se le asignó la IP 192.168.74.10.

Figura 3. Topología final del despliegue



Nota. La imagen muestra la topología de este despliegue de OpenStack, con los nodos y servicios que cada uno aloja.

Luego se realizó un *disk dump* sobre otro disco duro para generar rápidamente el segundo nodo de cómputo (de IP 192.168.74.9). Este *disk dump* se realizó usando el comando `dd` de Linux [9]. Al realizar el *disk dump* se debe elegir un dispositivo de almacenamiento origen (para este ejemplo será nombrado `/dev/sda`) y uno de destino (en este ejemplo será nombrado `/dev/sdb`).

Cuadro 1. Creación del *disk dump*

```
$ sudo unmount /dev/sda #desmontar dispositvo origen
$ sudo dd if=/dev/sda of=./nombre_archivo_dump.img bs=4M status=
   progress
$ sudo unmount /dev/sdb #desmontar dispositvo destino
$ sudo dd if=./nombre_archivo_dump.img of=/dev/sdb bs=4M status=
   progress oflag=sync
```

Una vez que se logró la copia del nodo de cómputo y se le cambió la IP del nodo original, incluyendo en todos los archivos de configuración, hay que comprobar que el nodo controlador reconoce a ambos nodos. Esto significa que detecte dos servicios de cómputo, de vólumenes y de red.

Cuadro 2. Listar servicios de cómputo desde el nodo controldor

```
$ . openstack_files/admin-openrc
$ openstack compute service list
+------+----------------+----------------------+---------+-------+
| ID   | Binary         | Host                 | Status  | State |
+------+----------------+----------------------+---------+-------+
| f... | nova-scheduler | openstackcontroller  | enabled | up    |
| d... | nova-conductor | openstackcontroller  | enabled | up    |
| 3... | nova-compute   | opcompute1           | enabled | up    |
| 4... | nova-compute   | opcompute2           | enabled | up    |
+------+----------------+----------------------+---------+-------+
```

Nota. Comando para listar recursos de cómputo y salida esperada si ya se cuenta con dos nodos de cómputo.

Cuadro 3. Listar servicios de volúmenes desde el nodo controlador

```
$ openstack volume service list
+------------------+----------------------+------+---------+-------+
| Binary           | Host                 | Zone | Status  | State |
+------------------+----------------------+------+---------+-------+
| cinder-scheduler | openstackcontroller  | nova | enabled | up    |
| cinder-volume    | opcompute1@lvm       | nova | enabled | up    |
| cinder-volume    | opcompute2@lvm       | nova | enabled | up    |
+------------------+----------------------+------+---------+-------+
```

Nota. Comando para listar recursos de volúmenes y salida esperada si ya se cuenta con dos nodos de cómputo.

Cuadro 4. Listar servicios de red desde el nodo controlador

```
$ openstack network agent list
+------+--------------------+----------------------+-------+
| ID   | Agent Type         | Host                 | State |
+------+--------------------+----------------------+-------+
| 2... | DHCP agent         | openstackcontroller  | UP    |
| 4... | Linux bridge agent | opcompute2           | UP    |
| 4... | L3 agent           | openstackcontroller  | UP    |
| s... | Linux bridge agent | openstackcontroller  | UP    |
| 8... | Metadata agent     | openstackcontroller  | UP    |
| a... | Linux bridge agent | opcompute1           | UP    |
+------+--------------------+----------------------+-------+
```

Nota. Comando para listar recursos de red y salida esperada si ya se cuenta con dos nodos de cómputo.

Tras comprobar que el controlador reconoce todos los servicios de los nodos de cómputo, se realiza un *aggregate* para la creación y administración del clúster. En OpenStack, un *aggregate* (o *host aggregate*) es un mecanismo lógico utilizado por el servicio Nova para agrupar uno o varios nodos de cómputo según criterios administrativos, de *hardware* o de disponibilidad. Esta agrupación permite aplicar políticas específicas de programación (*scheduling*) o distinguir capacidades físicas entre los *hosts*, sin requerir una separación física o de red. Los *aggregates* pueden asociarse a *availability zones* (zonas de disponibilidad) visibles para los usuarios, o mantenerse como agrupaciones internas empleadas por el *scheduler* para decidir dónde lanzar las instancias [26].

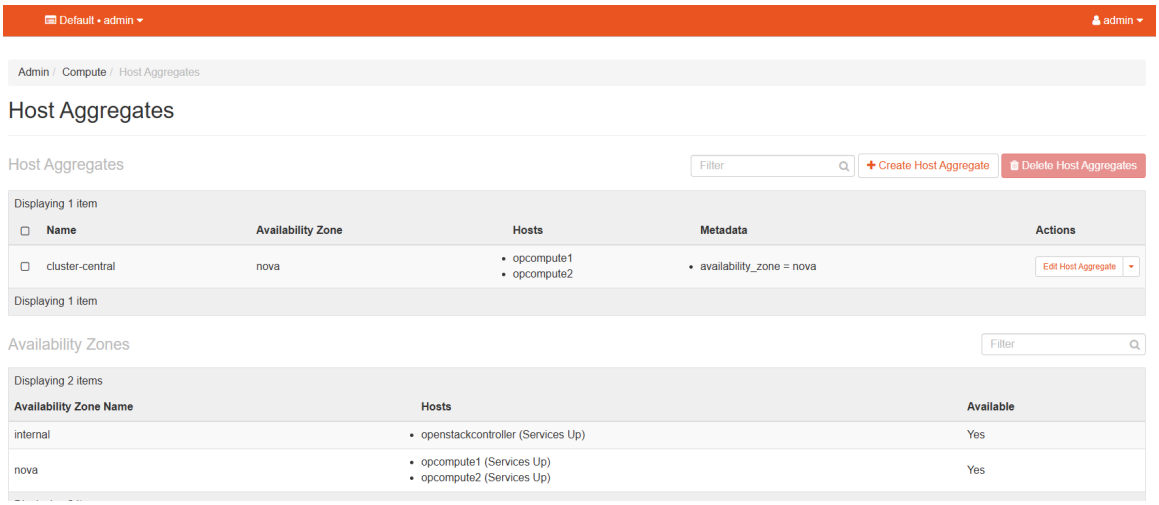
Cuadro 5. Creación del *aggregate* para el clúster desde el nodo controlador

```
$ openstack aggregate create cluster-central --zone nova
$ openstack aggregate add host cluster-central opcompute1
$ openstack aggregate add host cluster-central opcompute2
```

Cuadro 6. Listar *aggregates* desde el nodo controlador

```
$ openstack aggregate show cluster-central
+-------------------+----------------------------------------+
| Field             | Value                                  |
+-------------------+----------------------------------------+
| availability_zone| nova                                    |
| created_at        | 2025-11-05T00:54:26.000000             |
| deleted_at        | None                                   |
| hosts             | ['opcompute1', 'opcompute2']           |
| id                | 1                                      |
| is_deleted        | False                                  |
| name              | cluster-central                        |
| properties        | None                                   |
| updated_at        | None                                   |
| uuid              | 554aaa5d-57ad-4103-9594-1d8c10f9dd4a   |
+-------------------+----------------------------------------+
```

Figura 4. *Aggregate* en Horizon



Nota. La imagen muestra como Horizon ya reconoce el clúster y los *hosts* que hay en el.

7.2. Evaluación del funcionamiento

Para la validación del funcionamiento de este nuevo entorno de nube se crearon redes, *routers*, *flavors*, volúmenes, imágenes e instancias.

Las imágenes que se crearon para este proyecto fueron una imagen de Cirros, una Rocky 8.10 (*cloud image*) y una Ubuntu 24.04.3. La imagen de Ubuntu presentó problemas, y aunque pudo ser lanzada en una instancia, la instalación fallaba poco después de iniciar, incluso si se realizaba una instalación manual. Por otro lado, la imagen de Cirros no presentó

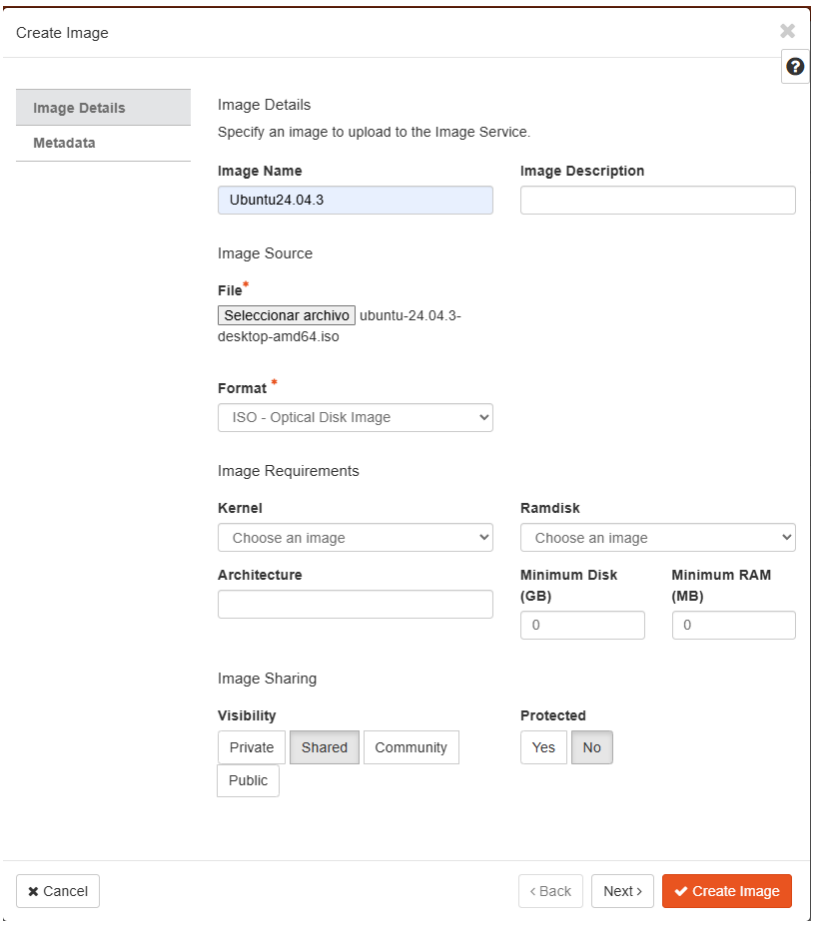
ningún tipo de falla y se pudo lanzar exitosamente al igual que la imagen de Rocky.

La creación de imágenes puede hacerse desde la CLI del controlador o cargar imágenes que se tengan previamente descargadas directamente en Horizon. Ambos métodos son igualmente funcionales para que OpenStack reconozca las imágenes que se tienen disponibles.

Cuadro 7. Creación de cirros desde el CLI del controlador

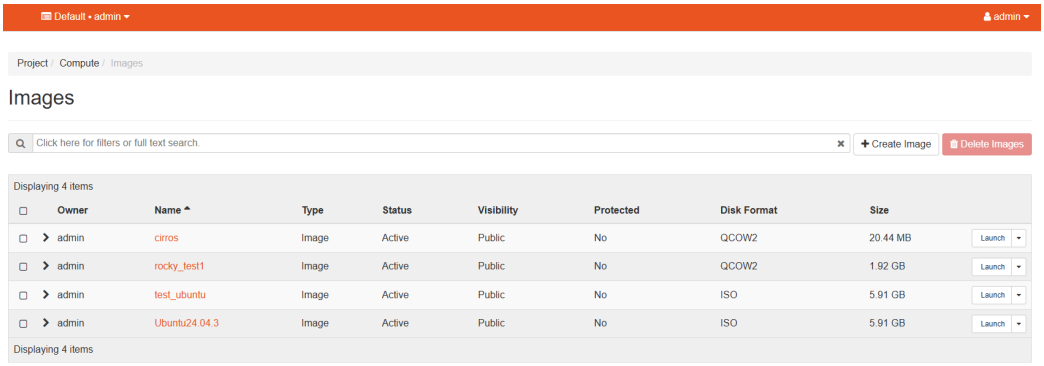
```
$ wget https://download.cirros-cloud.net/0.6.2/cirros-0.6.2-x86_64-disk.img
$ glance image-create --name "cirros" --file cirros-0.6.2-x86_64-disk.img --disk-format qcow2 --container-format bare --visibility=public
```

Figura 5. Creación de imágenes desde Horizon



Nota. La imagen muestra como cargar una imagen en Horizon.

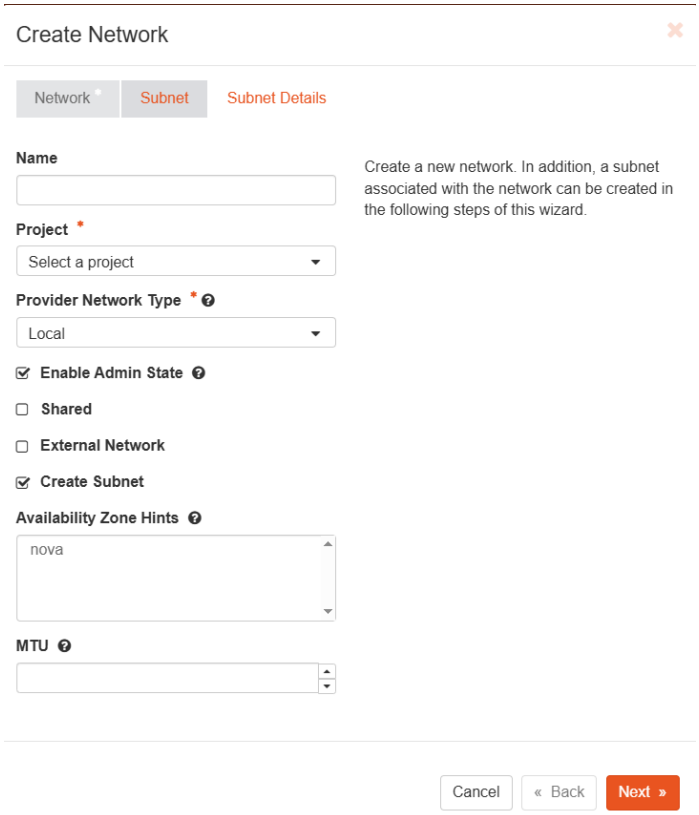
Figura 6. Imágenes disponibles



Nota. La imagen muestra el conjunto de imágenes cargadas y disponibles.

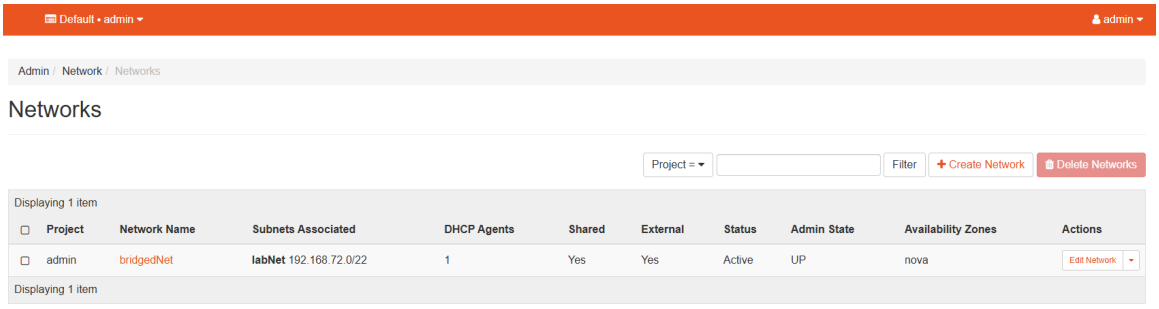
La creación de redes se probó en Horizon y se realizó una red de tipo *flat* con subred de la universidad (192.168.72.0/22) para que las instancias que se conecten a ella tengan acceso a internet así como comunicación con cualquier equipo físico dentro de la red. Es importante que al configurar la red en Horizon se indique que la red es externa. También debe configurarse el DNS (8.8.8.8) en los detalles de subred. También pudieron crearse redes de tipo VXLAN para probar el funcionamiento solo dentro del entorno de OpenStack.

Figura 7. Creación de red



Nota. La imagen muestra como crear una red desde Horizon.

Figura 8. Redes disponibles en Horizon

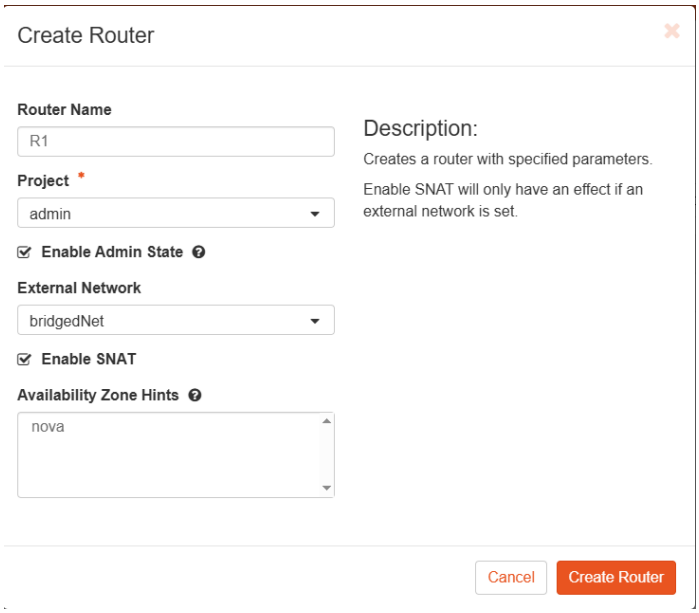


Nota. La imagen muestra las redes disponibles a utilizar.

La creación de un *router* es un proceso igual al de las redes, completamente desde Horizon. Sin embargo requiere de menos configuraciones, la más importante es a qué subred se conecta para tener una correcta asignación de IP y de interfaz. También es posible agregar

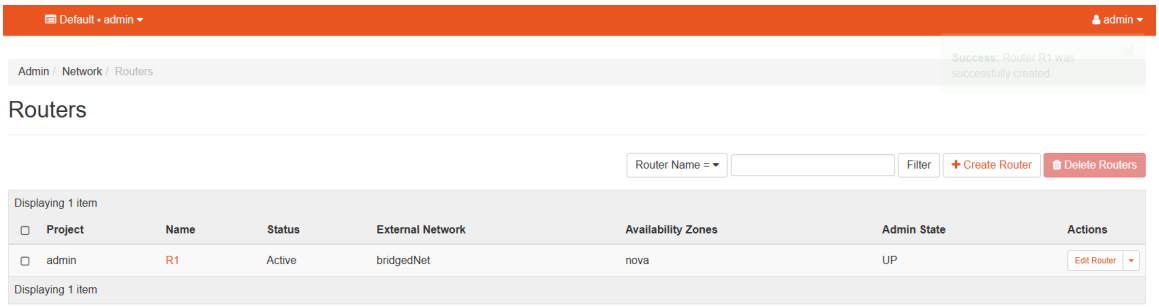
insterfaces para interconectar redes. En este caso, el *router* terminó con interfaces de dos redes con IP 192.168.73.246 (red puente a la red de la universidad) y 192.168.222.1 (red local creada dentro de OpenStack). Estas interfaces pueden verse en la figura 11.

Figura 9. Creación de *router*



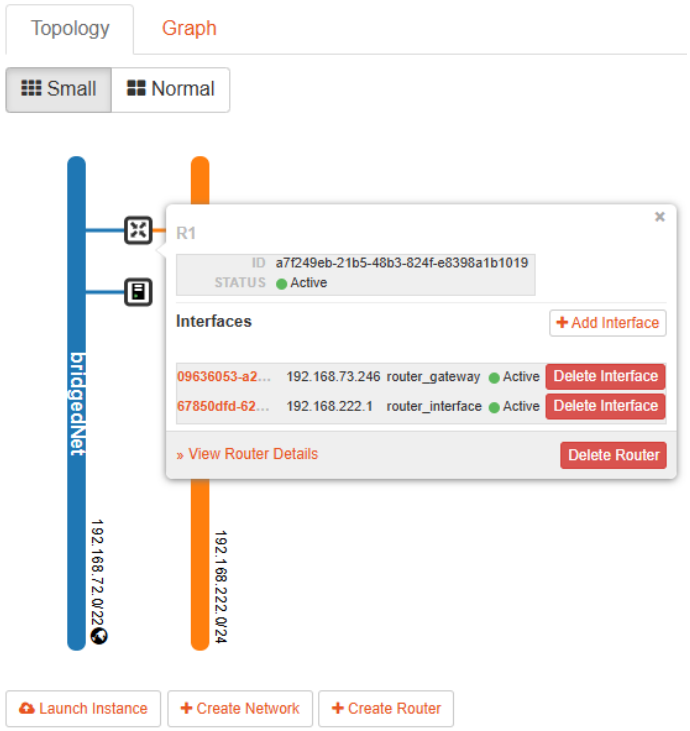
Nota. La imagen muestra como crear un *router* desde Horizon.

Figura 10. *Routers* disponibles en Horizon



Nota. La imagen muestra los *routers* disponibles a utilizar.

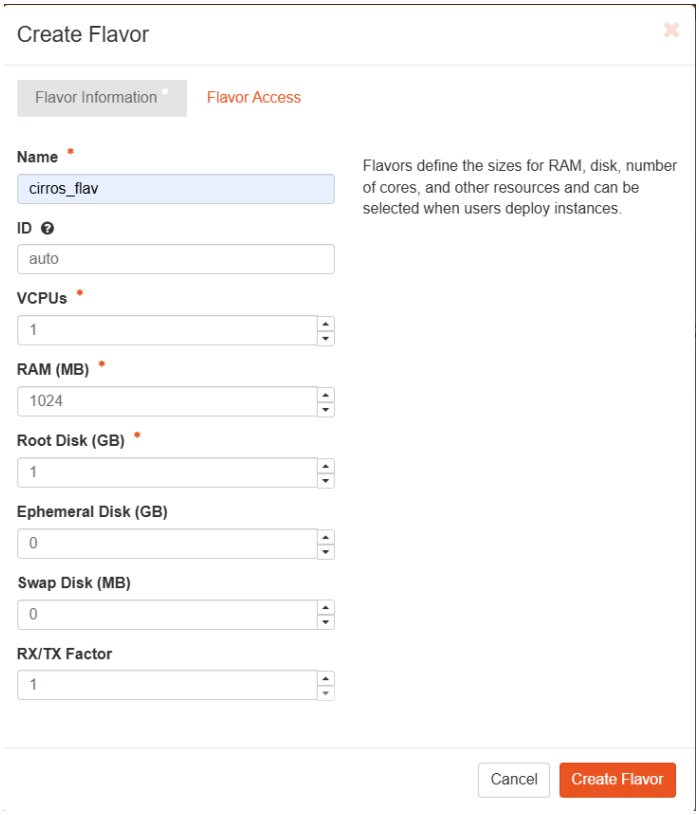
Figura 11. Topología de red funcionando



Nota. La imagen muestra una topología de red funcional incluyendo la conectividad en los puertos del *router* desde Horizon.

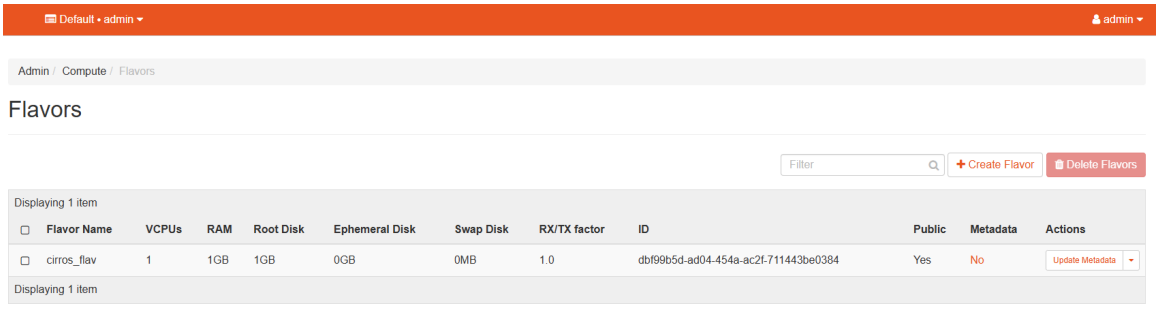
La creación de un *flavor* desde Horizon requiere únicamente de asignar la cantidad de recursos, principalmente de la cantidad de RAM y disco duro o disco raíz que requiera el sistema operativo para la cual se destinará el *flavor*. El resto de configuraciones pueden omitirse para la creación del *flavor*, a menos que el sistema operativo que se desee lanzar tenga requerimientos especiales. Los *flavors* no pueden ser editados tras su creación.

Figura 12. Creación de *flavor*



Nota. La imagen muestra como crear un *flavor* desde Horizon.

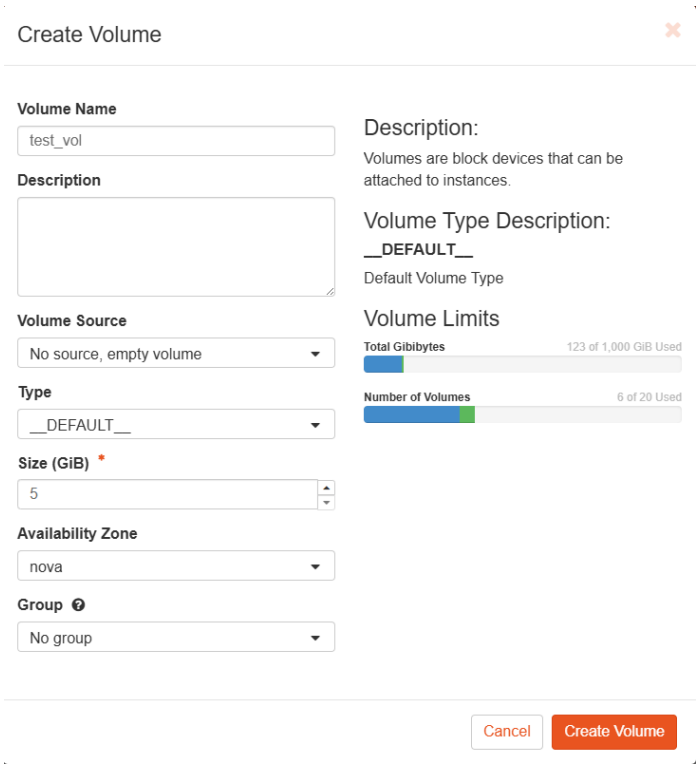
Figura 13. *Flavors* disponibles en Horizon



Nota. La imagen muestra los *flavors* disponibles a utilizar.

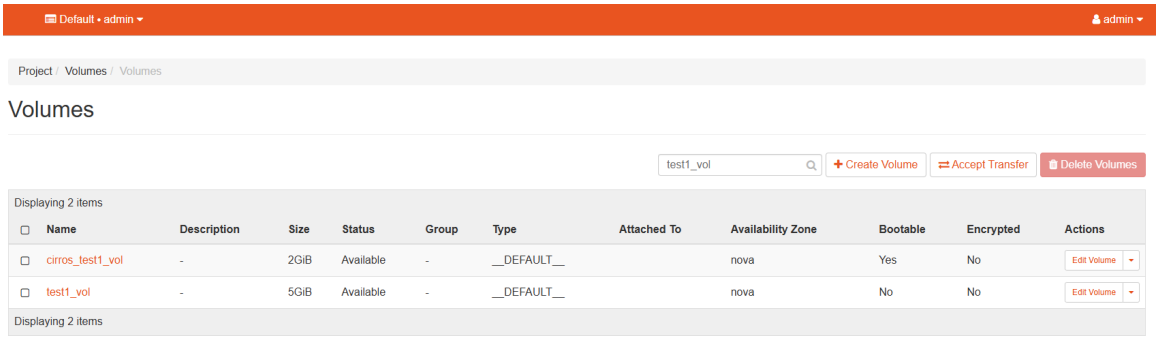
La creación de volúmenes es similar a la creación de *flavors*, pues solo necesita asignación de recursos y tampoco es editable tras su creación. Sin embargo, los volúmenes pueden ser creados desde 0 o basados en imágenes/volúmenes que luego pueden ser lanzados como instancias.

Figura 14. Creación de volúmenes



Nota. La imagen muestra como crear volúmenes desde Horizon.

Figura 15. Volúmenes disponibles en Horizon

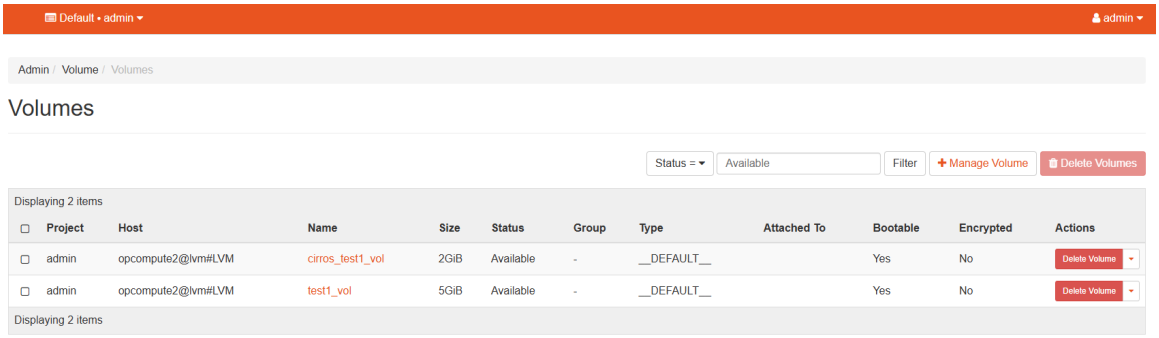


Nota. La imagen muestra los volúmenes disponibles a utilizar.

La creación del clúster permite migrar volúmenes e instancias de un *host* a otro. Este proceso puede hacerse fácilmente desde Horizon. Cabe destacar que la asignación de recursos tanto de Nova como de Cinder es automática gracias a sus *shcedulers*. El Nova-Scheduler evalúa todos los nodos de cómputo disponibles y selecciona el más adecuado para ejecutar una instancia, basándose en la información de recursos (CPU, RAM, almacenamiento) que

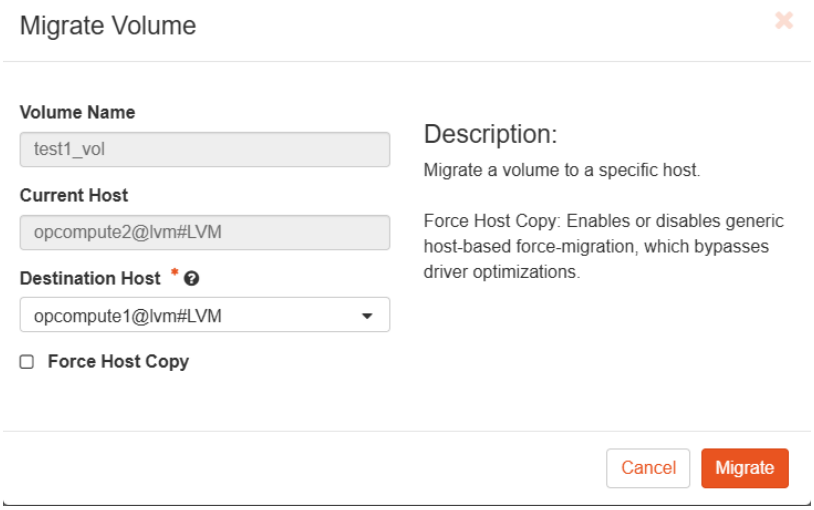
obtiene del servicio Placement. Este proceso se realiza aplicando una serie de filtros (que descartan *hosts* no aptos) y ponderadores (que asignan prioridad a los candidatos restantes). Por defecto, Nova prioriza los *hosts* con más memoria disponible mediante el RAMWeigher, aunque esta política puede personalizarse [27]. En cambio, el Cinder-Scheduler opera de forma independiente, ya que no depende de Placement; utiliza la información reportada por los servicios Cinder-Volume para determinar qué *backend* de almacenamiento posee suficiente capacidad libre o cumple con los criterios definidos por el administrador. Cinder también aplica filtros y pesos (por ejemplo, CapacityFilter y CapacityWeigher) para seleccionar el *backend* más adecuado y coordina la creación del volumen allí [28].

Figura 16. Vólumenes con sus *hosts* actuales



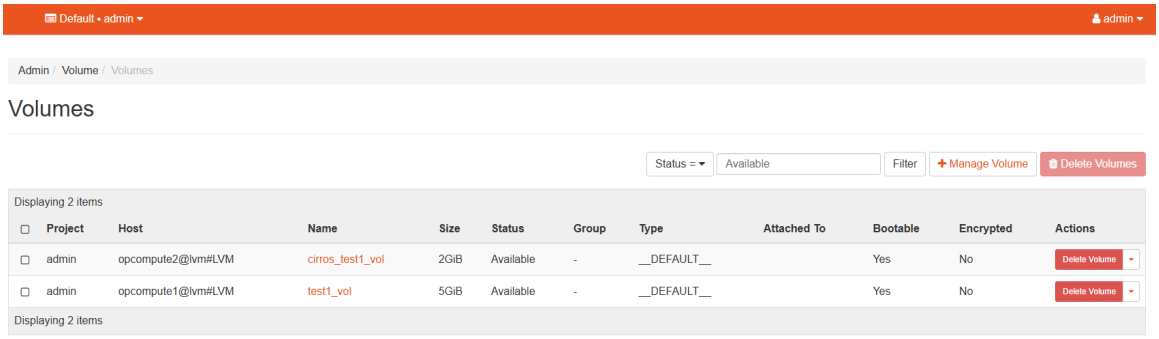
Nota. La imagen muestra los volúmenes disponibles a utilizar incluyendo en qué *host* están alojados.

Figura 17. Migración de volúmenes en Horizon



Nota. La imagen muestra como migrar un volumen desde Horizon de un *host* a otro.

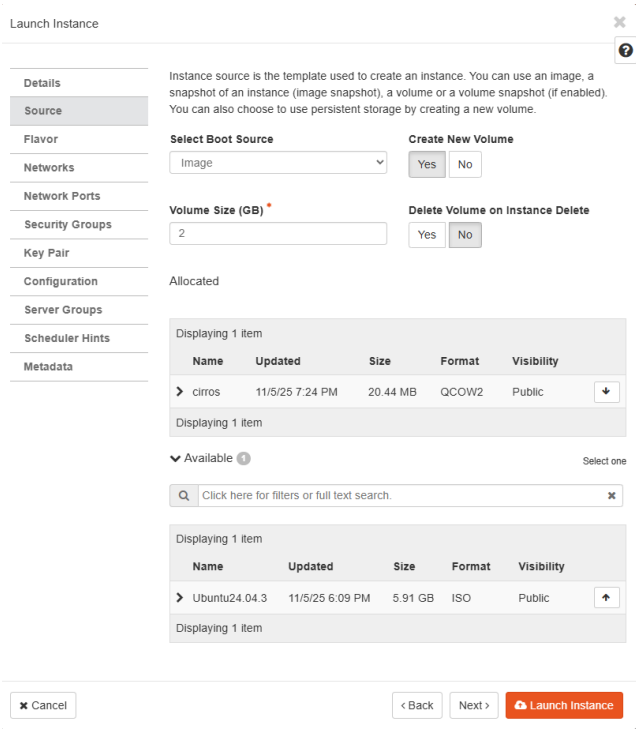
Figura 18. Volúmenes con sus *hosts* actualizados



Nota. La imagen muestra los volúmenes disponibles a utilizar incluyendo los cambios sobre los *host* en los que están alojados.

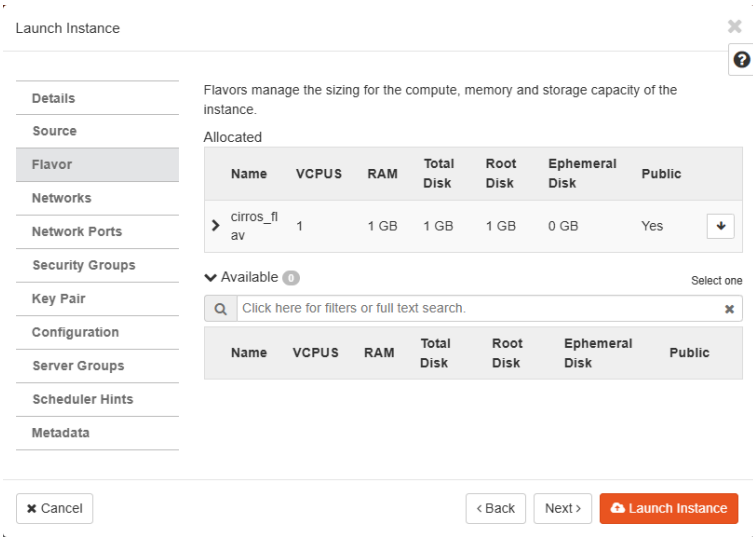
La creación de instancias necesita de todo lo previamente mencionado, es decir, necesita de imágenes, *flavors*, redes y volúmenes. Como se mencionó previamente, la imagen de Ubuntu presentó problemas, por lo que las pruebas se realizaron con Cirros y Rocky. Para lanzar una instancia de Rocky se requiere de una configuración adicional a la asignación de imagen, *flavor*, red y volumen ya que la imagen es para entornos de *Cloud* (formato .qcow2). Normalmente se debe generar un *script* .yml para la instalación, pero Horizon permite escribir el *script* dentro de sus configuraciones sin tener que forzosamente hacer un archivo previamente.

Figura 19. Asignación de imagen/volumen para lanzar una instancia



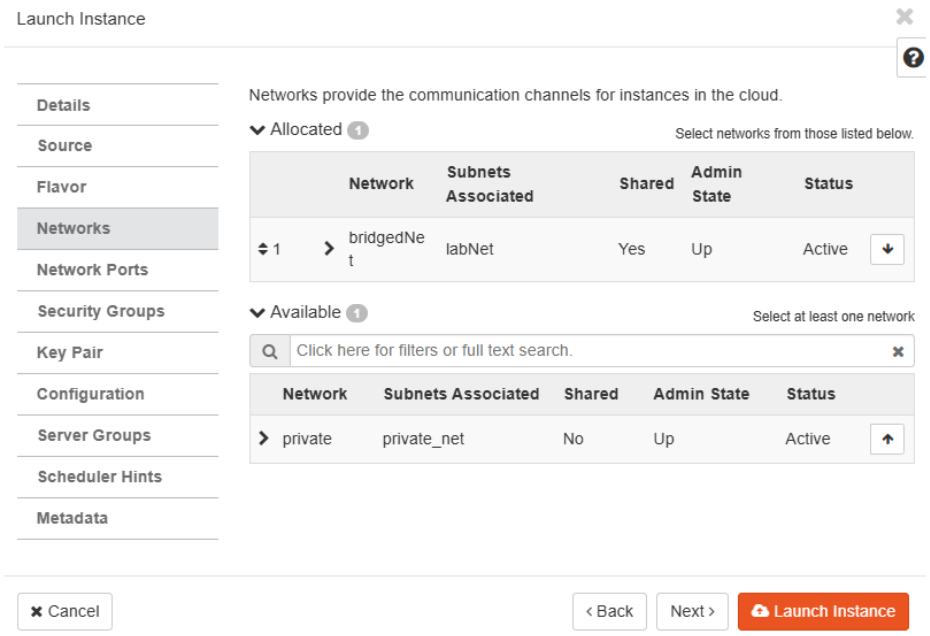
Nota. La imagen muestra como asignar qué imagen o volumen se usará para la instancia.

Figura 20. Asignación de *flavors* para lanzar una instancia



Nota. La imagen muestra como asignar qué *flavor* se usará para la instancia.

Figura 21. Asignación de red para lanzar una instancia



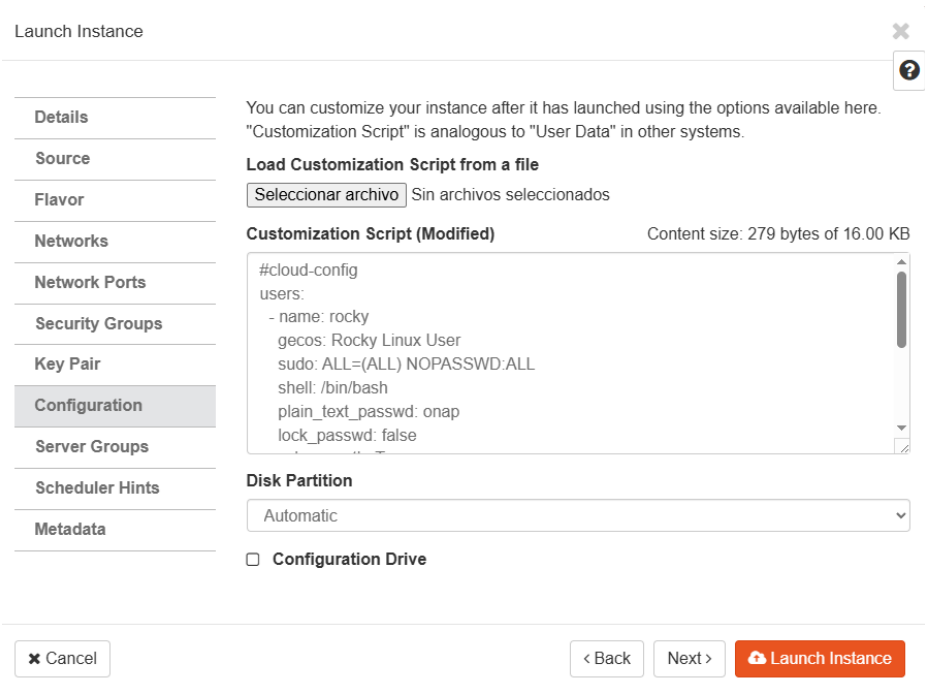
Nota. La imagen muestra como asignar qué red se usará para la instancia.

Cuadro 8. Configuración para la instancia de Rocky

```
#cloud-config
users:
  - name: rocky
    gecos: Rocky Linux User
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash
    plain_text_passwd: onap
    lock_passwd: false
    ssh_pwauth: True
chpasswd:
  list: |
    rocky:onap
  expire: False
ssh_pwauth: True
disable_root: false
```

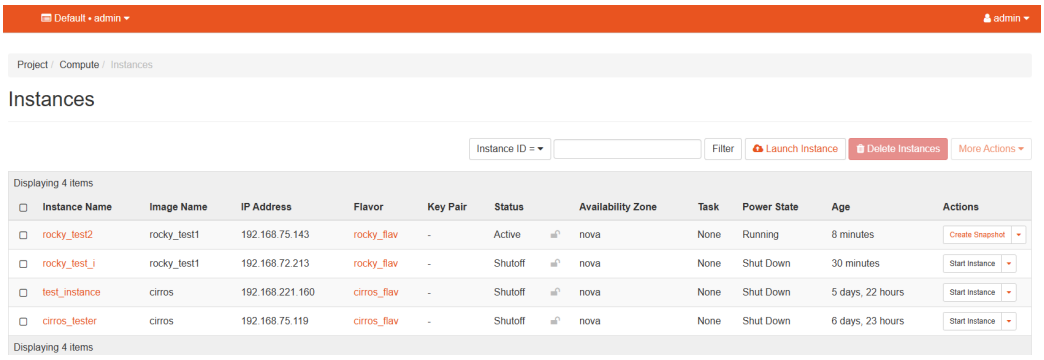
Nota. Esta configuración indica principalmente las credenciales a usar en la instancia, ya que las imágenes .qcow2 están pensadas para no tener que instalar manualmente.

Figura 22. *Script* de configuración para Rocky



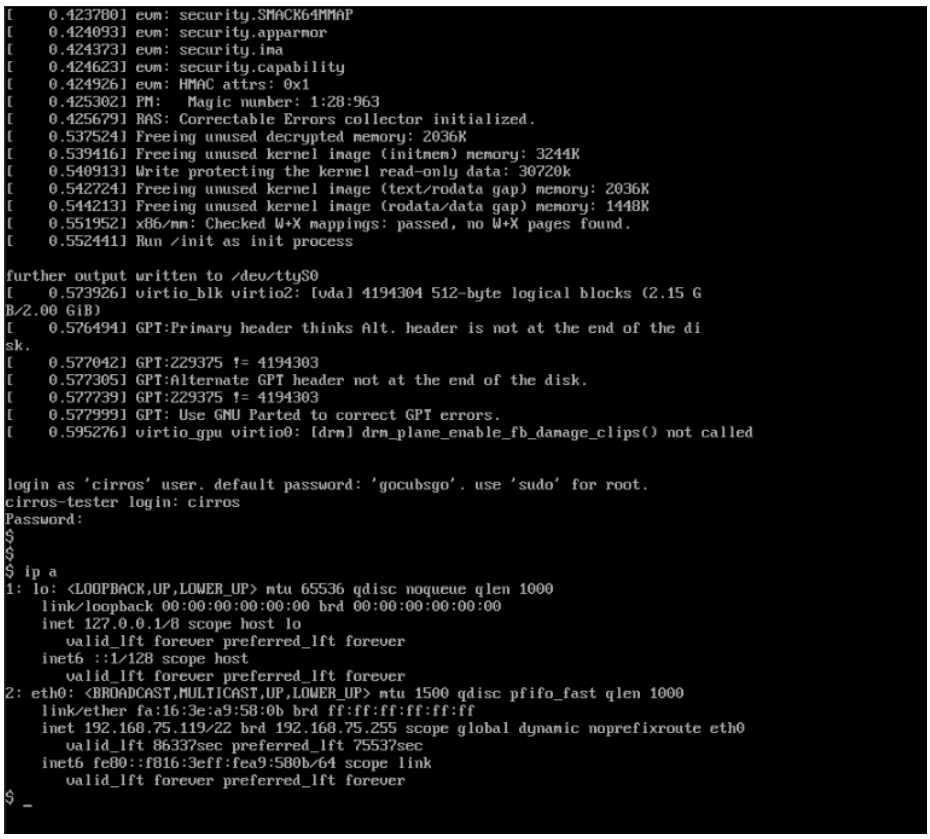
Nota. La imagen muestra como escribir directamente en Horizon el *script* para imágenes de formato .qcow2.

Figura 23. Instancias disponibles en Horizon



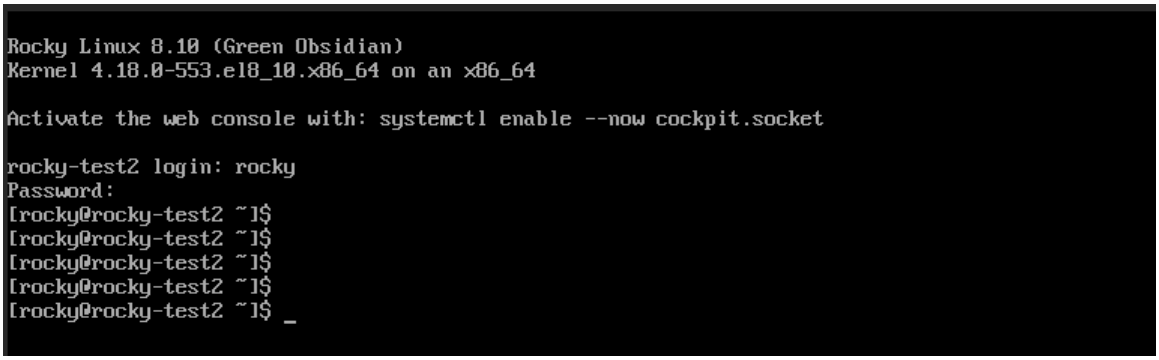
Nota. La imagen muestra las instancias disponibles a utilizar.

Figura 24. Instancia de Cirros funcionando



Nota. La imagen muestra la consola de cirros funcionando.

Figura 25. Instancia de Rocky funcionando

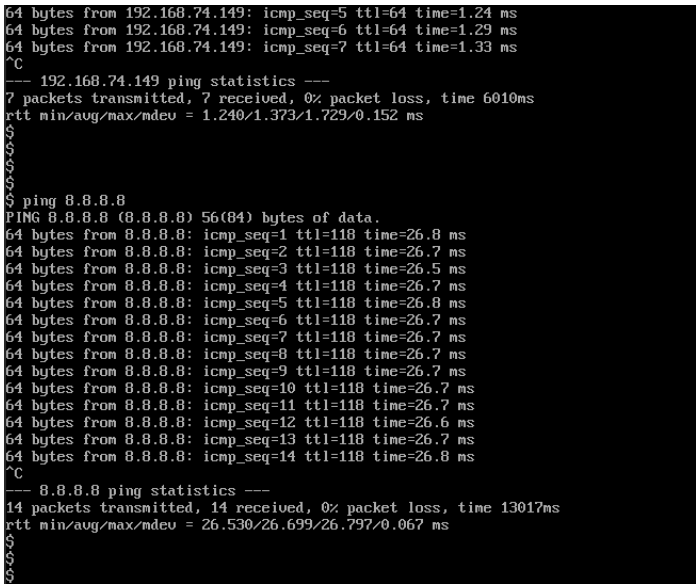


Nota. La imagen muestra la consola de Rocky 8.10 funcionando.

Las pruebas para el funcionamiento de las instancia y su conectividad fue el realizar *pings* hacía internet (8.8.8.8) así como hacía los nodos de cómputo (192.168.74.9 y 192.168.74.10), el nodo controlador (192.168.74.149) y hacía una computadora del laboratorio de la univer-

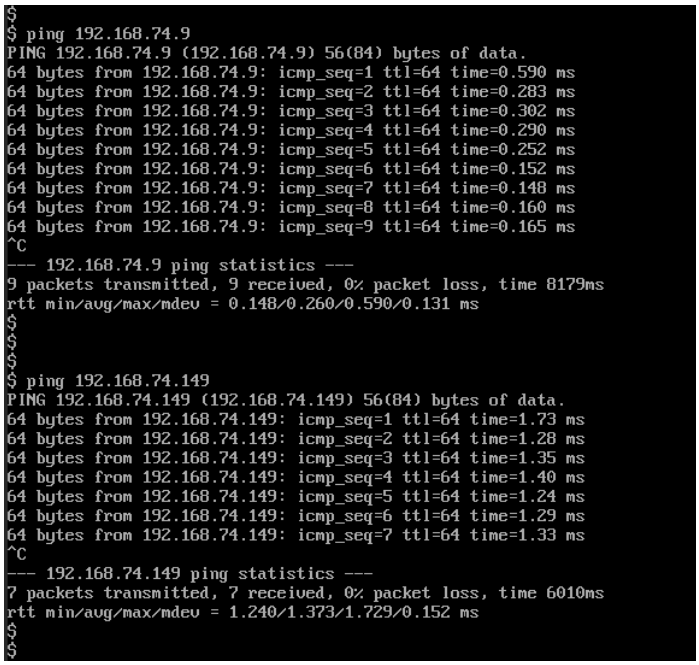
sidad (192.168.72.83).

Figura 26. Instancia Cirros realizando un *ping* hacía internet



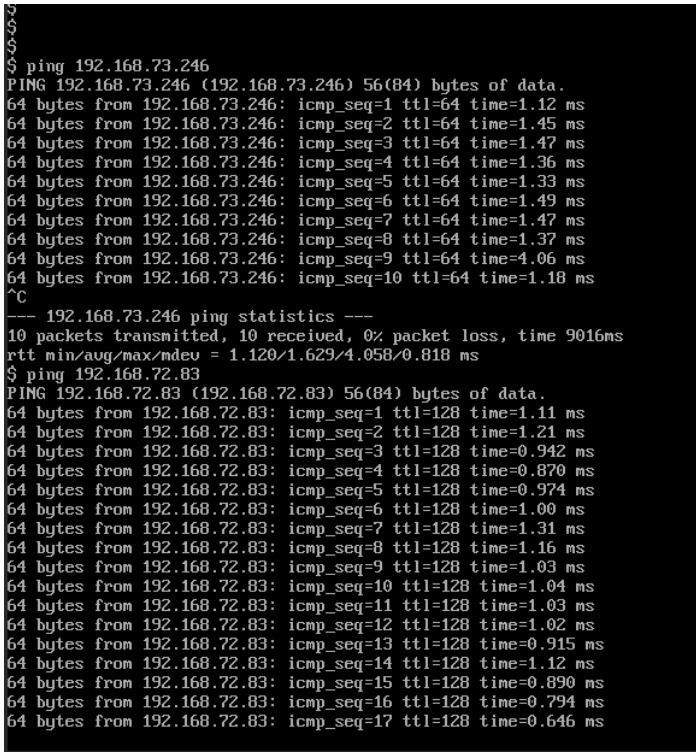
Nota. La imagen muestra a la instancia de cirros realizando un *ping* hacía internet.

Figura 27. Instancia Cirros realizando un *ping* hacía los nodos de OpenStack



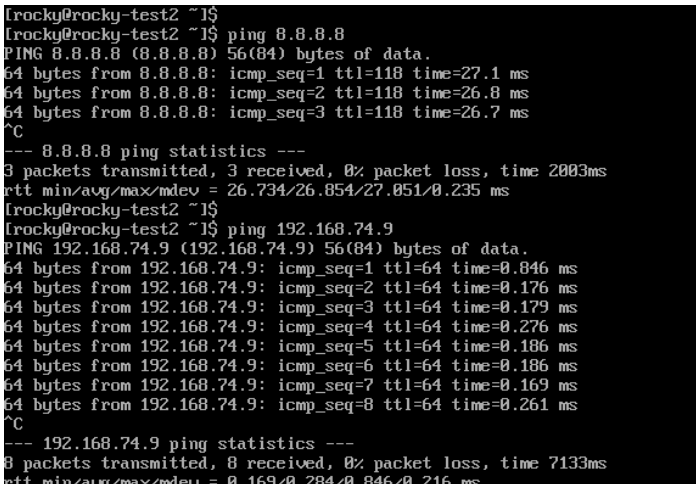
Nota. La imagen muestra a la instancia de cirros realizando un *ping* hacía el nodo controlador y el nodo de cómputo.

Figura 28. Instancia Cirros realizando un *ping* hacía el laboratorio de la universidad



Nota. La imagen muestra a la instancia de cirros realizando un *ping* hacía una computadora del laboratorio de la Universidad del Valle de Guatemala.

Figura 29. Instancia Rocky realizando un *ping* hacía internet y hacía un nodo de cómputo



Nota. La imagen muestra a la instancia de Rocky realizando un *ping* hacía el nodo de cómputo 2 y hacía internet.

7.3. Instalación de Barbican

Barbican es el servicio de gestión de secretos de OpenStack. Su función principal es proporcionar una API segura para el almacenamiento, la recuperación y la gestión de información sensible, como contraseñas, claves de cifrado, certificados o *tokens*. Este servicio es utilizado por otros componentes de OpenStack, como Cinder o Glance, para proteger sus datos confidenciales mediante el cifrado y el control de acceso centralizado.

Barbican permite manejar secretos a través de diferentes *backends* de almacenamiento, que pueden ser *software* o *hardware*. En entornos de desarrollo o pruebas, el almacenamiento por defecto se realiza en el sistema de archivos, mientras que en producción se recomienda utilizar un *backend* seguro o un *plugin* de *hardware*.

- Barbican-API: es el componente principal del servicio. Expone la interfaz RESTful de Barbican, permitiendo a los clientes y a otros servicios de OpenStack interactuar con la API de gestión de secretos. Este componente se ejecuta normalmente bajo el servidor *web* Apache mediante WSGI.
- Barbican-Keystone-Listener: escucha los eventos generados por Keystone (como la creación o eliminación de proyectos y usuarios) y sincroniza esos cambios con Barbican. Gracias a este servicio, la información de control de acceso entre Keystone y Barbican permanece coherente.
- Barbican-Worker: es el proceso encargado de ejecutar tareas asíncronas, como la creación y eliminación de secretos o la comunicación con los *plugins* de almacenamiento. Funciona en segundo plano y permite que la API responda rápidamente, mientras las operaciones complejas se procesan de manera paralela.

7.3.1. Instalación en el *controller node*

Primero, se crea la base de datos barbican en el gestor de base de datos MariaDB, así como asignación de permisos. El usuario de Barbican puede acceder desde el localhost o desde cualquier IP identificandose con su propia `BARBICAN_DBPASS`, la cual se mantuvo como `onap` para mantener la consistencia con el despliegue anterior.

Cuadro 9. Creación de base de datos para Barbican y otorgación de permisos

```
1 $ sudo mysql
2 MariaDB [(none)]>  CREATE DATABASE barbican;
3 MariaDB [(none)]>  GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@
    'localhost' IDENTIFIED BY 'BARBICAN_DBPASS';
4 MariaDB [(none)]>  GRANT ALL PRIVILEGES ON barbican.* TO 'barbican'@
    '%' IDENTIFIED BY 'BARBICAN_DBPASS';
5 MariaDB [(none)]>  exit;
```

Se registra el servicio Barbican en Keystone para que pueda autenticarse y ser descubierto

por los demás componentes. Se deben crear el usuario y contraseña del servicio, así como enlazarlo al dominio por defecto. Con ese usuario se crean roles y el servicio de *key-manager*.

Cuadro 10. Registro de de Barbican con Keystone

```
$ . openstack_files/admin-openrc
$ openstack user create --domain default --password BARBCAN_DBPASS
   barbican
$ openstack role add --project service --user barbican admin
$ openstack service create --name barbican --description "Key
   Manager" key-manager
```

Crear los *endpoints* para este servicio así como la definición de su región.

Cuadro 11. Creación de *endpoints*

```
$ openstack endpoint create --region RegionOne key-manager public
   http://controller:9311/v1/%\(tenant_id\)s
$ openstack endpoint create --region RegionOne key-manager internal
   http://controller:9311/v1/%\(tenant_id\)s
$ openstack endpoint create --region RegionOne key-manager admin
   http://controller:9311/v1/%\(tenant_id\)s
```

Cuadro 12. Salida esperada al crear cada *endpoint*

```
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 58f1cd21551c4f4aa8f3c50b87e880bb |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 19a5ea4e6ca44e599d9e06decc1b6ea0 |
| service_name | barbican                         |
| service_type | key-manager                      |
| url          | http://controller:9311           |
+--------------+----------------------------------+
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | d0129a80794d4ab7be69cae1ab4dafdc |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 19a5ea4e6ca44e599d9e06decc1b6ea0 |
| service_name | barbican                         |
| service_type | key-manager                      |
| url          | http://controller:9311           |
+--------------+----------------------------------+
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 9388470082f04d859d9d48ef0e408f09 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 19a5ea4e6ca44e599d9e06decc1b6ea0 |
| service_name | barbican                         |
| service_type | key-manager                      |
| url          | http://controller:9311           |
+--------------+----------------------------------+
```

Instalación de los paquetes Barbican-API, Barbican-Keystone-Listener y Barbican-Worker. Son necesarias configuraciones adicionales al archivo `/etc/barbican/barbican.conf` para poder realizar las conexiones con la base de datos SQL y la mensajería, y con el autenticador de Keystone.

Cuadro 13. Instalación de Barbican y apertura de su archivo de configuración

```
$ sudo apt install barbican-api barbican-keystone-listener barbican-
    worker
$ sudo nano /etc/barbican/barbican.conf
```

En la sección de `[DEFAULT]` se define la conexión a la base de datos y la mensajería, donde `tansport_url` requiere de una `RABBIT_PASS` la cuál se definió en el despliegue anterior como `onap`, por lo que para mantener la consistencia en este caso también se definió así. Para la `BARBICAN_PASS` esto último también aplica.

Cuadro 14. Cambios al archivo de configuración de Barbican

```
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
sql_connection = mysql+pymysql://barbican:BARBICAN_DBPASS@controller
    /barbican

[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = barbican
password = BARBICAN_PASS
```

Popular la base de datos del *key manager*. Este comando cambia al usuario barbican para ejecutar el comando `barbican-manage db upgrade` desde la terminal, el cual crea el *schema* de la base de datos de Barbican. Esto puede hacerse de manera automática al cambiar `db_auto_create` a *true* en la sección `[DEFAULT]` del archivo de configuración; esta vez se omitió y se hizo manualmente.

Cuadro 15. Popular la base de datos del servicio *key manager*.

```
$ sudo su
$ su -s /bin/sh -c "barbican-manage db upgrade" barbican
```

Barbican tiene una arquitectura de *plugins* que permite al implementador almacenar secretos en un gran número de *backends* diferentes. Por defecto, Barbican tiene una configuración de almacenar secretos en una *keystore* básica basada en archivos. Este método de almacenamiento no se recomienda en un contexto de producción, pues no es seguro.

Cuadro 16. Finalización de la instalación

```
$ sudo service barbican-keystone-listener restart
$ sudo service barbican-worker restart
$ sudo service apache2 restart
```

Nota. Reinicio de servicios.

7.3.2. Verificación de funcionalidad

Cargar variables de entorno de OpenStack para ejecutar los comandos de creación de un secreto, obtención del secreto y verificación de su *payload*. La creación del secreto es solamente de ejemplo y se crea bajo el nombre de `mysecret`. El comando `get` consulta los metadatos del secreto identificado por un UUID (en este caso, `23086clb-9882-4424-845f-dfc914621b6d`), pero no muestra su contenido (es decir, no revela el valor del secreto), para revelarlo al descargar y descifrar (si aplica) el contenido del secreto usando la API REST se agrega el parámetro `-payload` al final del comando.

Cuadro 17. Creación de secreto ejemplo

```
$ . openstack_files/admin-open
$ openstack secret store --name mysecret --payload j4=jd21
+------------------+----------------------------------------+
| Field            | Value                                  |
+------------------+----------------------------------------+
| Secret href      | http://localhost:9311/v1/secrets/      |
|                  | 23086clb-9882-4424-845f-df6914621b6d   |
| Name             | mysecret                               |
| Created          | None                                   |
| Status           | None                                   |
| Content types    | None                                   |
| Algorithm        | aes                                    |
| Bit length       | 256                                    |
| Secret type      | opaque                                 |
| Mode             | cbc                                    |
| Expiration       | None                                   |
+------------------+----------------------------------------+
```

Nota. Comando de creación de secretos y salida esperada.

Cuadro 18. Metadatos del secreto de ejemplo

```
$ openstack secret get http://localhost:9311/v1/secrets/23086clb
   -9882-4424-845f-dfc914621b6d
+------------------+----------------------------------------+
| Field            | Value                                  |
+------------------+----------------------------------------+
| Secret href      | http://localhost:9311/v1/secrets/      |
|                  |  23086clb-9882-4424-845f-dfc914621b6d  |
| Name             | mysecret                               |
| Created          | 2025-11-05T01:39:30+00:00              |
| Status           | ACTIVE                                 |
| Content types    | {'default': 'application/octet-stream'}|
| Algorithm        | aes                                    |
| Bit length       | 256                                    |
| Secret type      | opaque                                 |
| Mode             | cbc                                    |
| Expiration       | None                                   |
+------------------+----------------------------------------+
```

Nota. Comando de consulta a metadatos de secretos y salida esperada.

Cuadro 19. Contenido del secreto de ejemplo

```
$ openstack secret get http://localhost:9311/v1/secrets/23086clb
   -9882-4424-845f-df6914621b6d --payload
+---------+---------------------------------+
| Field   | Value                           |
+---------+---------------------------------+
| Payload | j4=jd21                         |
+---------+---------------------------------+
```

Nota. Comando que revela el contenido de un secreto.

Cuadro 20. Listar secretos

```
$ openstack secret list
+---------------------------------------+----------+--------+
| Secret href                           | Name     | Status |
+---------------------------------------+----------+--------+
| http://localhost:9311/v1/secrets/...| mysecret | ACTIVE |
+---------------------------------------+----------+--------+
```

CAPÍTULO 8

Conclusiones

- Se logró replicar completamente el *compute node 1* del despliegue original en un nuevo nodo denominado *compute node 2*, garantizando que la configuración, los servicios y los componentes instalados fueran idénticos al nodo inicial. Esto permitió disponer de los recursos necesarios para conformar un clúster de cómputo funcional dentro del entorno OpenStack, incrementando la capacidad de procesamiento disponible y habilitando la distribución de cargas de trabajo entre ambos nodos.
- Se documentó el proceso de configuración del clúster, tanto en el *compute node* 1 como en el *compute node 2*, abarcando desde la réplica del nodo original hasta la integración final con el *controller node*. La documentación incluye los pasos de réplica y la validación del correcto reconocimiento de los recursos de cómputo por parte del controlador.
- Se configuró un *controller node* único y centralizado encargado de la administración completa de los recursos de los nodos de cómputo, permitiendo un control unificado del despliegue. Esta arquitectura garantiza una gestión más eficiente de las instancias, los volúmenes, las redes y demás servicios.
- Se evaluó el funcionamiento del clúster mediante la creación y ejecución de instancias de prueba, utilizando imágenes ligeras del sistema Cirros e imágenes para entornos *cloud* de Rocky Linux. Estas instancias se desplegaron correctamente, demostrando la comunicación efectiva entre los componentes del clúster y su conectividad con la red institucional de la Universidad del Valle de Guatemala. Esto confirmó que las instancias generadas en el entorno OpenStack son accesibles desde la red real y plenamente funcionales en términos de red y rendimiento básico.
- Se instaló y configuró el servicio de *key manager* Barbican, integrándolo dentro del despliegue general de OpenStack. Este servicio aporta una capa adicional de seguridad al permitir el almacenamiento y la gestión de información sensible, como contraseñas.

CAPÍTULO 9

Recomendaciones

- Implementar un sistema de almacenamiento compartido o distribuido, por ejemplo, mediante NFS, Ceph o un *backend* dedicado de Cinder, para permitir una gestión más eficiente de los volúmenes y respaldos, así como para darle al clúster una mayor tolerancia a fallos. La ausencia de un almacenamiento externo redundante limitó la alta disponibilidad del entorno.
- Dado que la PowerEdge T560 no cuenta con un arreglo de discos redundante, sería recomendable integrar un sistema RAID o un volumen LVM replicado en futuras expansiones.
- Evitar el uso de Windows a la hora de implementar alguno de los nodos. También debe evitarse la implementación de los nodos de cómputo en máquinas virtuales, pues esto presenta problemas a la hora del lanzamiento de cualquier instancia.
- Migrar el *backend* de almacenamiento de secretos hacia una solución más segura que la basada en archivos, utilizando un módulo de seguridad *hardware* (HSM) o un *backend* criptográfico certificado. Esto alinearía el entorno con las buenas prácticas de seguridad de OpenStack y permitiría el uso de Barbican en contextos de producción.
- Evaluar la integración de servicios adicionales de OpenStack como Heat o Swift, que podrían ampliar las capacidades del clúster, especialmente en la automatización de despliegues y la gestión de almacenamiento de objetos. Sin embargo, su implementación debe planificarse considerando los recursos físicos disponibles.
- Dado que las pruebas con imágenes de Ubuntu presentaron errores durante la instalación, se recomienda realizar un análisis más profundo de compatibilidad de imágenes y *flavors*, verificando la asignación de recursos mínimos y los *drivers* de virtualización utilizados por KVM. Asimismo, podrían probarse versiones más ligeras o imágenes *cloud* oficiales optimizadas para OpenStack.

CAPÍTULO 10

Referencias

[1] F. López, «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,» Tesis doct., Universidad del Valle de Guatemala, 2024.

[2] Y. Jghef y S. Zeebaree, «State of Art Survey for Significant Relations between Cloud Computing and Distributed Computing,» *The International Journal of Science and Business*, 2020.

[3] P. C, *GRID AND CLUSTER COMPUTING*. PHI Learning Pvt. Ltd., 2008.

[4] M. Baker y R. Buyya, *Cluster Computing at a Glance*. Division of Computer Science University of Portsmouth Southsea, Hants, UK, 1999.

[5] T. Heath, B. Diniz, E. Carrera, W. Meira Jr. y B. Ricardo, *Self-Configuring Heterogeneous Server Clusters*. Department of Computer Science Rutgers University, 2003.

[6] L. Xiao, X. Zhang y S. Kubricht, «Incorporating job migration and network RAM to share cluster memory resources,» en *Proceedings the Ninth International Symposium on High-Performance Distributed Computing*, 2000.

[7] J. Hsieh, C. Stanton y R. Ali, «Ninth International Conference on Parallel and Distributed Systems, 2002. Proceedings.,» en *Proceedings the Ninth International Symposium on High-Performance Distributed Computing*, 2002.

[8] A. S. Tanenbaum y H. Bos, *Modern Operating Systems*. Pearson Education, 2015.

[9] The GNU Project, *dd - convert and copy a file*, Available at: `https://man7.org/linux/man-pages/man1/dd.1.html`, 2024.

[10] M. Lamourine, «OpenStack,» *The magazine of USENIX and SAGE*, 2014.

[11] R. Kumar, N. Gupta, S. Charu, K. Jain y S. Kumal Jangir, «Open source solution for cloud computing platform using OpenStack.,» *International Journal of Computer Science and Mobile Computing.*, 2014.

[12] O. Foundation, «Neutron Documentation, Release 26.1.0.dev175,» Neutron development team, inf. téc., 2025.

[13] O. Foundation, «Barbican Documentation, Release 20.1.0.dev26,» OpenStack Foundation, inf. téc., 2025.

[14] L. Girish y H. Guruprasad, «Building private cloud using openstack.,» *International Journal of Emerging Trends and Technology in Computer Science.*, 2014.

[15] O. Foundation, «Glance Documentation, glance 30.1.0.dev45,» OpenStack Foundation, inf. téc., 2024.

[16] O. Foundation, «Horizon Documentation, Release 25.4.0.dev100,» OpenStack Foundation, inf. téc., 2025.

[17] A. Lingayat, A. Singh, V. Naik, R. Badre y A. Gupta, «Horizon, a Web-Based User Interface for Managing Services in OpenStack: An Introspection.,» en *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT).*, 2018.

[18] K. Jackson y C. Bunch, *OpenStack Cloud Computing Cookbook*. Packt Publishing Ltd., 2015.

[19] O. Foundation, «Nova Documentation, Release 31.1.0.dev86,» OpenStack Foundation, inf. téc., 2025.

[20] O. Foundation, «Placement Documentation, openstack-placement 2.0.1.dev2,» OpenStack Foundation, inf. téc., 2019.

[21] O. Foundation, «Placement Documentation, openstack-placement 13.1.0.dev5,» OpenStack Foundation, inf. téc., 2019.

[22] O. Foundation. «OpenStack Software. »dirección: `https://www.openstack.org/software/`.

[23] T. Fifield et al., *OpenStack Operations Guide: Set Up and Manage your OpenStack Cloud.* O'Reilly Media, Inc., 2014.

[24] O. contributors, «Install Guide,» OpenStack contributors, inf. téc., 2025.

[25] O. Foundation. «OpenStack Liberty Installation Guide - Overview. »dirección: `https://docs.openstack.org/liberty/install-guide-rdo/overview.html`.

[26] O. Documentation. «Compute service — Host aggregates and availability zones. »dirección: `https://docs.openstack.org/nova/latest/admin/aggregates.html`.

[27] O. Documentation. «Scheduling — nova. »dirección: `https://docs.openstack.org/nova/latest/reference/scheduling.html`.

[28] O. Documentation. «Cinder Scheduler. »dirección: `https://docs.openstack.org/cinder/latest/configuration/scheduler.html`.

CAPÍTULO 11

Anexos

11.1. Archivos de configuración

Todos los archivos de configuración de esta sección fueron extraídos de la guía de instalación del despliegue modular original [1]. Se presentan con algunos cambios menores que aplican para este despliegue, los cuales están indicados como comentarios.

11.1.1. Configuraciones de red

Cuadro 21. Configuración de IP estática en el archivo de Netplan

```
$ sudo nano /etc/netplan/00-installer-config.yaml

network:
    ethernets:
        ens33: #el nombre de esta interfaz varía en cada dispositivo
            dhcp4: no
            addresses: [192.168.74.149/22] #192.168.74.10/22 para
               Compute 1 y 192.168.74.9/22 para Compute 2
            nameservers:
                addresses: [192.168.8.205,8.8.8.8]
            routes:
            - to: default
              via: 192.168.72.1
    version: 2
```

Nota. Esta configuración de IP se hace en todos los nodos.

Cuadro 22. Modificación de *hostnames*

```
$ sudo nano /etc/hosts

# Add definitions
127.0.0.1 localhost
192.168.74.149 controller
192.168.74.10 compute1
192.168.74.9 compute2
```

Nota. Esta configuración de *hostnames* se hace en todos los nodos.

11.1.2. Configuraciones importantes de Keystone

Cuadro 23. Creación y modificación de archivo de variables del sistema

```
$ mkdir openstack_files
$ sudo nano openstack_files/admin-openrc #apertra del archivo

export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=onap #o la contraseña que se defina para el
    despliegue
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

11.1.3. Configuraciones importantes de Glance

Cuadro 24. Configuración del archivo principal de Glance-Api

```
$ sudo nano /etc/glance/glance-api.conf

[database]
connection = mysql+pymysql://glance:onap@controller/glance

[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = GLANCE_PASS

[paste_deploy]
# ...
flavor = keystone

[glance_store]
# ...
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/

[oslo_limit]
# Add whole section if it doesn't exist
auth_url = http://controller:5000
auth_type = password
user_domain_id = default
username = glance
system_scope = all
password = onap
region_name = RegionOne

[DEFAULT]
# ...
# use_keystone_quotas = True
```

11.1.4. Configuraciones importantes de Placement

Cuadro 25. Configuración del archivo principal de Placement

```
$ sudo nano /etc/placement/placement.conf

[placement_database]
connection = mysql+pymysql://placement:onap@controller/placement

[api]
# ...
auth_strategy = keystone

[keystone_authtoken]
# Remove everything else from this section
auth_url = http://controller:5000/v3
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = onap
```

11.1.5. Configuraciones importantes de Nova en *controller node*

Cuadro 26. Configuración del archivo principal de Nova en *controller node*

```
$ sudo nano /etc/nova/nova.conf

[api_database]
connection = mysql+pymysql://nova:onap@controller/nova_api

[database]
connection = mysql+pymysql://nova:onap@controller/nova

[DEFAULT]
# Due to a packaging bug, remove the log_dir option from the section
transport_url = rabbit://openstack:onap@controller:5672/
my_ip = 192.168.74.149

[api]
# ...
auth_strategy = keystone

[keystone_authtoken]
# Remove all other options
www_authenticate_uri = http://controller:5000/
auth_url = http://controller:5000/
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
```

```
user_domain_name = Default
project_name = service
username = nova
password = onap

[service_user]
# ...
send_service_user_token = true
auth_url = http://controller:5000/identity
auth_strategy = keystone
auth_type = password
project_domain_name = Default
project_name = service
user_domain_name = Default
username = nova
password = onap

[neutron]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = Default
user_domain_name = Default
region_name = RegionOne
project_name = service
username = neutron
password = onap
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET

[vnc]
# ...
enabled = true
server_listen = 192.168.74.149
server_proxyclient_address = 192.168.74.149

[glance]
# ...
api_servers = http://controller:9292

[oslo_concurrency]
# ...
lock_path = /var/lib/nova/tmp

[placement]
# ...
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
username = placement
password = onap
```

```
79
80 [scheduler]
81 discover_hosts_in_cells_interval = 300
```

11.1.6. Configuraciones importantes de Nova en el *compute node*

Cuadro 27. Configuración del archivo principal de Nova en el *compute node*

```
1  $ sudo nano /etc/nova/nova.conf
2
3  [DEFAULT]
4  transport_url = rabbit://openstack:onap@controller
5  my_ip = 192.168.74.10 #192.168.74.9 para Compute Node 2
6
7  [api]
8  # ...
9  auth_strategy = keystone
10
11 [keystone_authtoken]
12 # remove all other options from this section
13 www_authenticate_uri = http://controller:5000/
14 auth_url = http://controller:5000/
15 memcached_servers = controller:11211
16 auth_type = password
17 project_domain_name = Default
18 user_domain_name = Default
19 project_name = service
20 username = nova
21 password = onap
22
23 [service_user]
24 send_service_user_token = true
25 auth_url = http://controller:5000/identity
26 auth_strategy = keystone
27 auth_type = password
28 project_domain_name = Default
29 project_name = service
30 user_domain_name = Default
31 username = nova
32 password = onap
33
34 [neutron]
35 # ...
36 auth_url = http://controller:5000
37 auth_type = password
38 project_domain_name = Default
39 user_domain_name = Default
40 region_name = RegionOne
41 project_name = service
42 username = neutron
43 password = onap
44
```

```
[vnc]
# ...
enabled = true
server_listen = 0.0.0.0
server_proxyclient_address = 192.168.74.10 #192.168.74.9 para
    Compute Node 2
novncproxy_base_url = http://192.168.74.149:6080/vnc_auto.html

[glance]
# ...
api_servers = http://controller:9292

[oslo_concurrency]
# ...
lock_path = /var/lib/nova/tmp

[placement]
# ...
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
username = placement
password = onap

[cinder]
os_region_name = RegionOne
```

Cuadro 28. Configuraciones importantes de Nova-Compute en el *compute node*

```
$ sudo nano /etc/nova/nova-compute.conf

[libvirt]
virt_type = qemu
```

Nota. Esta configuración es solo en caso que no haya compatibilidad con la aceleración de hardware para máquinas virtuales.

11.1.7. Configuraciones importantes de Neutron en el *controller node*.

Cuadro 29. Configuración del archivo principal de Neutron en el *controller node*

```
$ sudo nano /etc/neutron/neutron.conf

[database]
# remove all other connections
connection = mysql+pymysql://neutron:onap@controller/neutron

[DEFAULT]
# ...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = true
transport_url = rabbit://openstack:onap@controller
auth_strategy = keystone
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[keystone_authtoken]
# remove all other options in this section
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = neutron
password = onap

[nova]
# ...
auth_url = http://controller:5000
auth_type = password
project_domain_name = Default
user_domain_name = Default
region_name = RegionOne
project_name = service
username = nova
password = onap

[oslo_concurrency]
# ...
lock_path = /var/lib/neutron/tmp
```

Cuadro 30. Configuraciones importantes de ML2

```
$ sudo nano /etc/neutron/plugins/ml2/ml2_conf.ini

[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = linuxbridge,l2population
extension_drivers = port_security

[ml2_type_flat]
flat_networks = provider

[ml2_type_vxlan]
vni_ranges = 1:1000

[securitygroup]
enable_ipset = true
```

Cuadro 31. Configuraciones importantes de Linux Bridge

```
$ sudo nano /etc/neutron/plugins/ml2/linuxbridge_agent.ini

[linux_bridge]
physical_interface_mappings = provider:ens33 #esta interfaz cambia
    en cada dispositivo.

[vxlan]
enable_vxlan = true
local_ip = 192.168.74.149
l2_population = true

[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
    IptablesFirewallDriver
```

Cuadro 32. Configuraciones del agente de capa 3 l3

```
$ sudo nano /etc/neutron/l3_agent.ini

[DEFAULT]
interface_driver = linuxbridge
```

Cuadro 33. Configuraciones del agente DHCP

```
$ sudo nano /etc/neutron/dhcp_agent.ini

[DEFAULT]
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Cuadro 34. Configuración del agente de metadata

```
$ sudo nano /etc/neutron/metadata_agent.ini

[DEFAULT]
nova_metadata_host = controller
metadata_proxy_shared_secret = onap
```

11.1.8. Configuraciones importantes de Neutron en el *compute node*

Cuadro 35. Configuración del archivo principal de Neutron en el *compute node*

```
$ sudo nano /etc/neutron/neutron.conf

[database]
# comment out any connection options

[DEFAULT]
transport_url = rabbit://openstack:onap@controller
auth_strategy = keystone

[keystone_authtoken]
# Comment out or remove any other options
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = onap

[oslo_concurrency]
lock_path = /var/lib/neutron/tmp
```

Cuadro 36. Configuraciones importantes de Linux Bridge en el *compute node*

```
$ sudo nano /etc/neutron/plugins/ml2/linuxbridge_agent.ini

[linux_bridge]
physical_interface_mappings = provider:enp2s0 #la interfaz varía su
    nombre en cada dispositivo

[vxlan]
enable_vxlan = true
local_ip = 192.168.74.10 #192.168.74.9 en el Compute Node 2
l2_population = true

[securitygroup]
# ...
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
    IptablesFirewallDriver
```

11.1.9. Configuraciones importantes de Cinder en el *controller node*

Cuadro 37. Configuración del archivo principal de Cinder en el *controller node*

```
$ sudo nano /etc/cinder/cinder.conf

[database]
connection = mysql+pymysql://cinder:onap@controller/cinder

[DEFAULT]
transport_url = rabbit://openstack:onap@controller
auth_strategy = keystone
my_ip = 192.168.74.149

[keystone_authtoken]
# Remove everything else from this section
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = onap

[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

11.1.10. Configuraciones importantes de Cinder en el *storage node*

Cuadro 38. Configuración del archivo principal de Cinder en el *storage node*

```
$ sudo nano /etc/cinder/cinder.conf

[database]
connection = mysql+pymysql://cinder:onap@controller/cinder

[DEFAULT]
# ...
transport_url = rabbit://openstack:onap@controller
auth_strategy = keystone
my_ip = 192.168.74.10 #192.168.74.9 para el Compute Node 2
enabled_backends = lvm
glance_api_servers = http://controller:9292

[keystone_authtoken]
# Remove everything else from this section
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = cinder
password = onap

[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = tgtadm

[oslo_concurrency]
lock_path = /var/lib/cinder/tmp
```

Nota. Recordar que el *storage node* es el *compute node* en este despliegue.

Cuadro 39. Configuración del TGT

```
$ sudo nano /etc/tgt/targets.conf

# ...
include /var/lib/cinder/volumes/*
```

11.1.11. Configuraciones importantes de Horizon

Cuadro 40. Configuración del agente de metadata

```
$ sudo nano /etc/openstack-dashboard/local_settings.py

ALLOWED_HOSTS = ['*']

CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.
           MemcachedCache',
        'LOCATION': 'controller:11211',
    }
}

# Comment out any other session storage configuration
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'

OPENSTACK_HOST = "controller"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/identity/v3" %
   OPENSTACK_HOST
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
OPENSTACK_API_VERSIONS = {
    "identity": 3,
    "image": 2,
    "placement": 1,
    "compute": 2,
    "network": 2,
    "volume": 3,
}
TIME_ZONE = "America/Guatemala"
```

CAPÍTULO 12

Glosario

API Interfaz de programación de aplicaciones (*application programming interface*). Conjunto de protocolos y definiciones que permiten la comunicación entre diferentes componentes de software. En OpenStack, cada servicio expone una API RESTful que facilita la interacción con el sistema desde otros servicios o clientes externos.

Barbican Servicio de gestión de claves (*key manager*) de OpenStack que ofrece almacenamiento seguro y administración de secretos, tales como claves criptográficas, certificados o datos binarios sensibles. Mejora la seguridad general del sistema al evitar el manejo directo de información confidencial en archivos de configuración.

Block storage Sistema de almacenamiento basado en bloques gestionado por el servicio Cinder en OpenStack. Permite crear volúmenes persistentes que pueden adjuntarse a instancias como si fueran discos duros físicos, facilitando la gestión de datos y copias de seguridad.

Cinder Servicio de OpenStack encargado de gestionar el almacenamiento en bloques. Proporciona volúmenes persistentes que las instancias pueden montar, formatear y utilizar para almacenar información. Soporta distintos tipos de *backends* como LVM, Ceph o NFS.

Clúster Conjunto de nodos interconectados que trabajan de forma cooperativa para ofrecer una infraestructura de cómputo unificada y de alto rendimiento. En este proyecto, el clúster se compone de dos nodos de cómputo y un controlador centralizado que administra todos los recursos.

Compute node Nodo encargado de ejecutar las instancias de máquinas virtuales dentro de una infraestructura OpenStack. Utiliza hipervisores como KVM o QEMU para la virtualización y mantiene comunicación constante con el nodo controlador para recibir tareas y reportar estados.

Controller node Nodo principal dentro de una arquitectura OpenStack. Aloja los servicios centrales de control, como Keystone (autenticación), Glance (imágenes), Neutron (re-

des), Nova (cómputo), Placement (asignación de recursos) y Horizon (interfaz gráfica). Es el encargado de coordinar las operaciones entre los diferentes nodos.

Disk dump Procedimiento utilizado para generar una copia exacta de un disco duro o partición, incluyendo tanto los datos como la estructura del sistema de archivos. Se emplea en este proyecto para replicar el despliegue original de OpenStack desde una HPC a otra de manera precisa, garantizando la compatibilidad de configuraciones y servicios.

Endpoints Puntos de acceso de red que permiten la comunicación entre los distintos servicios de OpenStack a través de sus APIs. Cada servicio tiene tres tipos de endpoints: público, interno y administrativo, los cuales definen cómo y desde dónde puede accederse al servicio.

Glance Servicio de OpenStack encargado del manejo de imágenes de disco utilizadas para la creación de instancias. Permite almacenar, registrar, consultar y recuperar imágenes de sistemas operativos y plantillas de máquinas virtuales, apoyándose en diversos tipos de almacenamiento *backend*.

Hipervisor Software de virtualización que permite ejecutar múltiples máquinas virtuales en un mismo hardware físico. Asigna y gestiona recursos como CPU, memoria y almacenamiento. En OpenStack, el hipervisor más común es KVM, aunque también se soportan Xen, VMware y Hyper-V.

Horizon Panel de control web de OpenStack que ofrece una interfaz gráfica de usuario para la administración de los diferentes servicios del sistema. Facilita la gestión de instancias, volúmenes, redes y proyectos sin necesidad de usar comandos CLI.

HPC Acrónimo de *high performance computing*. Se refiere a sistemas informáticos diseñados para realizar cálculos complejos a gran velocidad mediante el uso de múltiples procesadores trabajando en paralelo. En este proyecto, las HPC Precision 7920 y PowerEdge T560 constituyen los nodos principales del clúster.

Instancia Máquina virtual creada y gestionada por el servicio Nova dentro del entorno OpenStack. Cada instancia opera como un sistema operativo independiente y puede conectarse a redes virtuales, volúmenes de almacenamiento y servicios de seguridad.

Keystone Servicio de identidad de OpenStack encargado de la autenticación de usuarios, autorización y gestión de roles. Proporciona *tokens* de acceso temporales que permiten a los usuarios y servicios interactuar de manera segura con la nube.

LAN Red de área local (*local area network*) que interconecta dispositivos dentro de un mismo entorno físico o campus. En la arquitectura de OpenStack, se emplea para la comunicación entre nodos, transferencia de datos y sincronización de servicios.

Neutron Servicio de OpenStack que gestiona la conectividad de red como servicio. Permite crear y administrar redes virtuales, subredes, *routers* y reglas de *firewall*. Facilita la comunicación entre instancias y con el exterior, soportando tecnologías como VLANs, VXLAN y SDN.

Nova Módulo de cómputo de OpenStack que proporciona la infraestructura para aprovisionar y gestionar instancias virtuales. Nova se comunica con Glance, Keystone, Neutron y Placement para crear entornos virtualizados escalables y distribuidos.

Placement Servicio de OpenStack que rastrea, inventaría y asigna recursos computacionales a los distintos servicios y cargas de trabajo. Permite optimizar el uso de CPU, RAM y almacenamiento disponibles en los nodos de cómputo.

RegionOne Nombre predeterminado asignado a la región principal dentro de un despliegue de OpenStack. Agrupa los servicios y nodos de una infraestructura en una misma ubicación lógica.

Snapshot Instantánea de una instancia o volumen en un momento determinado. Permite restaurar sistemas a estados anteriores o clonar configuraciones para generar nuevas instancias de manera rápida.

Token Credencial temporal de autenticación generada por Keystone que valida el acceso de usuarios y servicios dentro del entorno OpenStack. Es utilizada en cada interacción entre servicios como prueba de identidad y autorización.

VM Máquina virtual (*virtual machine*). Entorno aislado de ejecución que emula un sistema operativo completo sobre un hipervisor. Las VMs en OpenStack son gestionadas principalmente por Nova.

WSGI Interfaz de comunicación estándar entre aplicaciones web escritas en Python y servidores web. En OpenStack, es utilizada por el servicio Horizon y otros componentes para desplegar sus interfaces mediante Apache.

YAML Lenguaje de serialización de datos utilizado en configuraciones de OpenStack, especialmente en plantillas de orquestación o automatización. Su sintaxis legible y jerárquica lo hace ideal para describir infraestructuras como código.