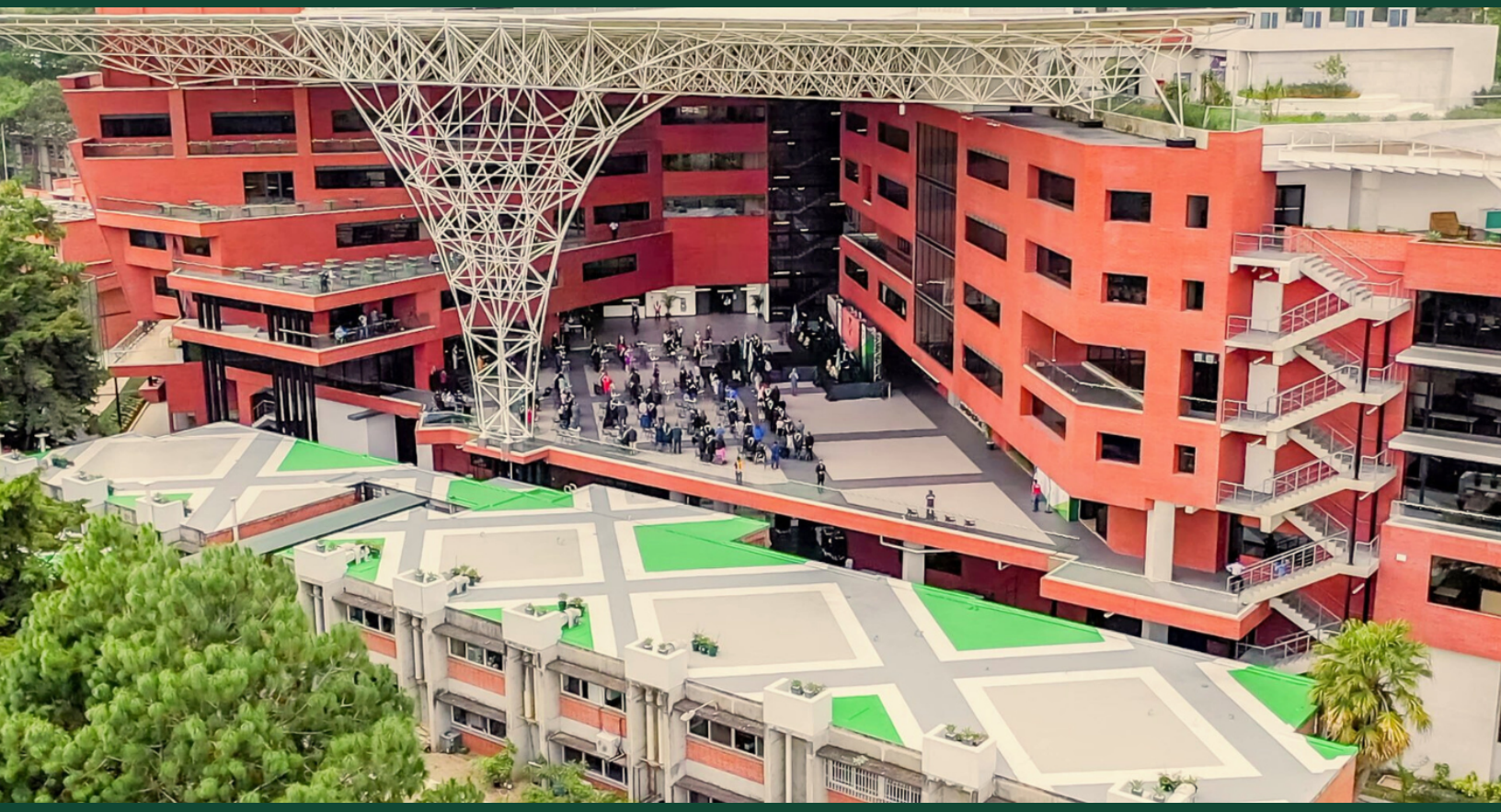
Implementación y configuración del servicio de provisión y gestión automatizada de clústeres de contenedores de OpenStack Magnum

Oscar Fernando Donis Martínez



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación y configuración del servicio de provisión y gestión automatizada de clústeres de contenedores de OpenStack Magnum

Trabajo de graduación presentado por Oscar Fernando Donis Martínez
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación y configuración del servicio de provisión y gestión automatizada de clústeres de contenedores de OpenStack Magnum

Trabajo de graduación presentado por Oscar Fernando Donis Martínez
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 Hernández

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

Prefacio

Antes de iniciar la redacción del contenido de este trabajo quiero expresar mi más grande agradecimiento a Dios, fuente de vida, sabiduría y mi fortaleza, quien me ha guiado a través de este camino y me ha sostenido en todo momento. Extiendo un especial reconocimiento al Departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala, a mis docentes y asesor, quienes con su orientación, exigencia y apoyo me guiaron en el proceso de construcción de este conocimiento. A mis compañeros y amigos, gracias por compartir este recorrido académico con solidaridad y entusiasmo. Finalmente a mi familia cuyo amor incondicional, comprensión y apoyo constante han sido el pilar fundamental para alcanzar este logro.

Índice general

Índice de figuras

Índice de cuadros

Resumen

El presente proyecto tuvo como propósito implementar el componente de OpenStack Magnum e integrarlo en la interfaz de Horizon, se logró ampliar las capacidades de la infraestructura de OpenStack del departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala. Logrando así la integración de la funcionalidad de provisionar y gestionar de forma automatizada clústeres de contenedores utilizando Kubernetes como motor de orquestación. La iniciativa buscó añadir valor a la infraestructura de nube privada de la universidad, que estaba limitada a la provisión de máquinas virtuales, por lo que se logró adaptarla a las tendencias actuales de arquitecturas centradas en contenedores, tomando provecho de su eficiencia para la utilización de recursos y su agilidad.

El proyecto se desarrolló sobre el hardware ya existente, el cual consiste en las dos computadoras de alto rendimiento (HPC), sin ninguna expansión del equipo físico. Durante el proceso de implementación se verificó el entorno de OpenStack existente y se replicó en el segundo nodo *compute*. Luego se encontró el proyecto Kolla-Ansible, el cual simplificó y automatizó la implementación de OpenStack utilizando contenedores Docker. Posteriormente se procedió con la instalación del componente para la orquestación de contenedores Magnum y su plugin para la interfaz de Horizon, así como la preparación de imágenes para nodos Kubernetes y la configuración de plantillas para la creación de clústeres, luego de varios clústeres de prueba fallidos, se buscó en guías no oficiales y foros de la comunidad, encontrando varias soluciones. Para evitar que la infraestructura de OpenStack se viera afectada por las pruebas, se crearon máquinas virtuales en las que se implementó un entorno de OpenStack con Magnum, en las cuales se realizaron las pruebas necesarias hasta lograr la configuración adecuada. Se desplegó exitosamente un clúster de Kubernetes utilizando Magnum y se verificó su funcionamiento mediante la implementación de una instancia del servidor web Nginx. Luego se implementaron y configuraron los componentes para el monitoreo Prometheus y Grafana, logrando recopilar métricas del clúster y visualizarlas en tableros de control. Se documentó todo el proceso de implementación, configuración y solución de problemas encontrados durante el desarrollo del proyecto. Validando un despliegue de clústeres de contenedores de Magnum en OpenStack predecible y replicable, junto al monitoreo del consumo de recursos y desempeño de los clústeres mediante Prometheus y Grafana. Se recomienda a futuros trabajos integrar CAPI (Cluster API) para la gestión de clústeres de contenedores, así como explorar la implementación de otros motores de or-

questación soportados por Magnum, como Docker Swarm y Apache Mesos, para ampliar las opciones disponibles para los usuarios, también se sugiere experimentar con las opciones de *Autohealer*, *Autoscaler* y el componente de balanceo de carga Octavia para mejorar la disponibilidad y escalabilidad de los clústeres de contenedores. Se exhorta a realizar toda la configuración y prueba en un entorno aislado (máquinas virtuales) para evitar afectar la infraestructura de OpenStack en las HPCs.

De esta manera se expandieron las capacidades de la infraestructura de OpenStack para aprovechar mejor los recursos de las computadoras de alto rendimiento. Esto ofrece a los estudiantes y docentes del departamento de Ingeniería Electrónica la oportunidad de experimentar con tecnologías de contenedores y orquestación, para diferentes propósitos académicos y de investigación.

Palabras clave: OpenStack, Magnum, Kubernetes, contenedores, orquestación, Prometheus, Grafana, infraestructura de nube privada.

Abstract

The purpose of this project was to implement the OpenStack Magnum component and integrate it into the Horizon interface. This successfully expanded the capabilities of the OpenStack infrastructure of the Electronic Engineering Department at the Univeridad del Valle de Guatemala. The project achieved the integration of the functionality to provision and manage container clusters automatically using Kubernetes as the orchestration engine. The initiative sought to add value to the university's private cloud infrastructure, which was previously limited to the provisioning of virtual machines. The project successfully adapted the infrastructure to current trends in container-centric architectures, leveraging their resource efficiency and agility.

The project was developed on existing hardware, consisting of two high-performance computing (HPC) systems, without any physical expansion. During the implementation process, the existing OpenStack environment was verified and replicated on the second node. The Kolla-Ansible project was then identified, which simplified and automated the deployment of OpenStack using Docker containers. Subsequently, the Magnum container orchestration component and its Horizon interface plugin were installed, along with the preparation of Kubernetes node images and the configuration of cluster creation templates. After several failed clusters tests, unofficial guides and community forums were consulted, yielding various solutions. To prevent the OpenStack infrastructure from being affected by the tests, virtual machines were created in which an OpenStack environment with Magnum was implemented. The necessary tests were performed on these virtual machines until the correct configuration was achieved. A Kubernetes cluster was successfully deployed using Magnum, and its functionality was verified by deploying an instance of the Nginx web server. The Prometheus and Grafana monitoring components were then implemented and configured, allowing for the collection of cluster metrics and their visualization on dashboards. The entire process of implementation, configuration, and troubleshooting encountered during the project's development was documented. Validating a predictable and replicable deployment of Magnum container clusters on OpenStack, along with monitoring resource consumption and cluster performance using Prometheus and Grafana. Future work is recommended to integrate CAPI (Cluster API) for container cluster management, as well as explore the implementation of other orchestration engines supported by Magnum, such as Docker Swarm and Apache Mesos, to expand the options available to users. It is also suggested to experiment with the

Autohealer, Autoscaler, and Octavia load balancing components to improve the availability and scalability of the container clusters. All configuration and testing should be performed in an isolated environment (virtual machines) to avoid impacting the OpenStack infrastructure on the HPCs.

This expanded the capabilities of the OpenStack infrastructure to better leverage the resources of high-performance computing (HPCs). This offers students and faculty in the Department of Electronic Engineering the opportunity to experiment with container and orchestration technologies for various academic and research purposes.

Keywords: OpenStack, Magnum, Kubernetes, Containers Clusters, Orchestration, Prometheus, Grafana.

CAPÍTULO 1

Introducción

El creciente avance de la tecnología y la digitalización de recursos y procesos han impulsado la casi obligatoria virtualización del desarrollo y despliegue de aplicaciones por medio de máquinas virtuales. Las máquinas virtuales han permitido optimizar el uso de recursos, ofreciendo escalabilidad y flexibilidad para adaptarse a la demanda exponencial de servicios digitales. Sin embargo, a pesar de la optimización que ofrecen las máquinas virtuales, nunca se logra un uso completo de los recursos físicos para el propósito de ejecutar aplicaciones, ya que cada una necesita de su propio sistema operativo, consumiendo más recursos de los necesarios. Debido a estas limitaciones, se crea la tecnología de contenedores, los cuales permiten empaquetar varias aplicaciones y sus dependencias cada uno independiente de los otros, pero compartiendo el mismo sistema operativo, esto permite utilizar los recursos más eficientemente y reduce el tiempo de despliegue de las aplicaciones. Herramientas como Docker y Kubernetes han cambiado la forma en que las aplicaciones son distribuidas y gestionadas, mejorando exponencialmente la flexibilidad y escalabilidad de estas. Sin embargo, al ser una tecnología en constante evolución, su implementación y gestión presenta desafíos significativos para las empresas sin mucha experiencia que intentan adoptarlas.

El objetivo de este trabajo es implementar el componente de orquestación de contenedores Magnum en la infraestructura de nube privada OpenStack en las computadoras de alto rendimiento (HPC) del departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala, como parte del proyecto se debía replicar la infraestructura existente en el segundo nodo *compute*. Su implementación permitirá la creación y gestión de clústeres de contenedores de forma nativa utilizando orquestadores populares como Kubernetes, Docker Swarm y Apache Mesos. Adicionalmente, se implementará su monitoreo utilizando las herramientas de Prometheus y Grafana. Este trabajo permitirá la experimentación y monitoreo de contenedores en el entorno controlado de la universidad para investigadores y estudiantes del departamento, facilitando el aprendizaje y adopción de esta tecnología. Aún cuando se completó la réplica de la infraestructura, durante ese periodo se encontró más simple y automatizada para levantar la infraestructura de OpenStack utilizando Kolla-Ansible [1] por lo que todo lo realizado y documentado en este trabajo se basará en esta herramienta.

Este proyecto se divide en dos partes principales: la implementación y prueba del servicio de orquestación de contenedores Magnum en OpenStack, y la configuración del monitoreo de los clústeres de contenedores utilizando Prometheus y Grafana. En la implementación de Magnum, se instalará y configurará el componente en la infraestructura de OpenStack, se mostrará el proceso de creación de plantillas de clústeres desde la selección y descarga de imágenes base hasta la verificación del correcto funcionamiento de los clústeres creados. En la segunda parte, se instalará y configurará Prometheus y Grafana para la recolección de datos y visualización del rendimiento básico de los clústeres, incluyendo métricas como uso de CPU, memoria y red.

En la primera parte del trabajo, se mostrará cómo instalar Magnum y los componentes necesarios en OpenStack, junto con configuraciones adicionales, luego se mostrará cómo descarga la imagen base para los clústeres y la creación de los recursos necesarios para el clúster, seguido de la creación de la plantilla del clúster y la verificación de los recursos disponibles de *Placement*, para finalizar se creará el clúster, se verificará de su estado, se accederá a los nodos del clúster para comprobar su correcto funcionamiento y se hará una prueba simple de despliegue del servidor web Nginx en el clúster. En la segunda parte del trabajo, se mostrará la instalación y configuración de Prometheus y Grafana, se habilitará el monitoreo de los clústeres y se crearán dashboards con gráficas básicas para visualizar el rendimiento de los clústeres.

CAPÍTULO 2

Antecedentes

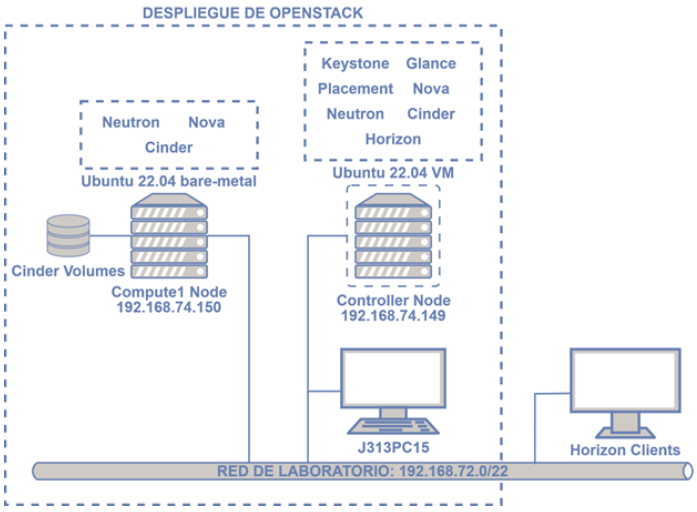
Con el crecimiento exponencial que ha experimentado la orquestación de contenedores y el amplio uso de OpenStack a nivel mundial, se creó un componente que hace que los motores de orquestación de contenedores como Docker Swarm, Kubernetes y Apache Mesos estén disponibles como recursos de primera clase en OpenStack [2].

El Laboratorio Europeo de Física de Partículas (CERN) ha sido de las organizaciones vanguardistas en la innovación de su infraestructura de TI para apoyar en su misión de expandir el conocimiento en física, generando una cantidad masiva de datos llegando a los 30 petabytes anuales solo del Gran Colisionador de Hadrones (LHC). Iniciaron a trabajar con OpenStack desde finales de 2011, comenzando su utilización en producción a mediados de los 2013 y migraron la mayor parte de su infraestructura a OpenStack por medio de máquinas virtuales en 2014. Siempre en busca de adoptar nuevas tecnologías y proporcionar una orquestación de contenedores para sus usuarios, CERN empezó a trabajar en la implementación de Magnum para aplicaciones nativas de contenedores a finales de 2015. La implementación de Magnum en una nube de la magnitud de CERN se administra por medio de Puppet el cual garantiza la consistencia y facilidad de hacer cambios de configuración y actualizaciones de software en miles de servidores. CERN está enfocado en el uso de módulos Puppet upstream y ha colaborado s con la comunidad Puppet para desarrollar configuraciones específicas como puppet-magnum. A pesar de los desafíos para comprender los contenedores y la cantidad de opciones como Kubernetes, Docker Swarm y Apache Meso, CERN sigue explorando y validando el enfoque para una implementación completa a todos sus usuarios en un futuro [3].

El proyecto realizado en la Universidad del Valle de Guatemala en el año 2024 estableció la base para la infraestructura de nube privada en el laboratorio del Departamento de Electrónica 1. El proyecto se enfocó en el despliegue y la configuración de los servicios principales de OpenStack, incluyendo componentes como Keystone el cual se encarga de la gestión de identidad, Glance para administrar las imágenes de las máquinas virtuales, Nova encargado de cómputo y la creación de instancias, Neutron para la creación y gestión de redes virtuales, Cinder para el almacenamiento en volúmenes y Horizon para gestionar la

red de OpenStack a través de una interfaz de gráfica. La implementación de estos componentes ofrece la capacidad de Infraestructura como Servicio (IaaS) centrada en máquinas virtuales. La implementación de esta plataforma operativa de nube sobre las computadoras de alto rendimiento en la universidad, constituyó un primer paso para futuros trabajos como el presente, pues este se fundamenta en esta infraestructura OpenStack previamente establecida, siendo el integrar el módulo OpenStack Magnum permitiendo la creación y gestión eficiente de la orquestación de contenedores, expandiendo así las capacidades del sistema de OpenStack de la universidad [4].

Figura 1. Infraestructura actual de OpenStack en la Universidad del Valle de Guatemala



CAPÍTULO 3

Justificación

Hoy en día, las aplicaciones e infraestructuras distribuidas están pasando de estar centradas en Máquinas Virtuales (VM) a estar centradas en contenedores. Desde 2014, se ha dedicado un considerable esfuerzo de investigación a mejorar las tecnologías de contenedores [5] [6]. La razón principal del cambio es la eficiencia de los contenedores en la utilización de recursos, ya que comparten el kernel del sistema anfitrión y únicamente empacan las dependencias de la aplicación, ofrecen un menor consumo de recursos de CPU, memoria y almacenamiento, derivado a esto los tiempos de arranque disminuyen significativamente y se puede aumentar la densidad de despliegue en comparación con las VMs que requieren de un sistema operativo completo [6] [7].

La optimización de los recursos cada día toma más relevancia, debido a la demanda actual de los servicios por Internet, exige una infraestructura capaz de responder con agilidad y eficiencia ofreciendo dinamismo, alta disponibilidad y escalabilidad. Reducir el consumo de recursos computacionales representa menores costos operativos, aumentando la capacidad de escalar servicios con mayor velocidad y menores costos para satisfacer picos de demanda, debido a la sobrecarga de las infraestructuras que trabajan con VMs, estas tienen limitaciones para alcanzar estos niveles de optimización y agilidad [7].

Con este contexto, la Universidad del Valle de Guatemala cuenta con la implementación de la plataforma desplegada de OpenStack en dos computadoras de alto rendimiento (HPC) con las capacidades necesarias para desplegar un servicio de cómputo en la nube mediante VMs. Por medio del componente Magnum el cual permite la integración de orquestación de contenedores (como Docker Swarm, Kubernetes y Apache Mesos) en OpenStack, existe la oportunidad de optimizar y aumentar su funcionalidad significativamente.

Esta herramienta proveerá a estudiantes como docentes la capacidad de experimentar con tecnologías de contenedores y orquestación, lo que les permitirá adquirir habilidades prácticas en un entorno de nube real. Con la implementación de una nube privada con la capacidad de orquestación de contenedores, se busca fomentar la investigación y el desarrollo de nuevas tecnologías, sin la necesidad de depender de recursos externos, reduciendo costos y aumentando la autonomía. De igual manera, la utilización de contenedores en lugar de

máquinas virtuales permitirá una mayor eficiencia en el uso de los recursos, lo que se traduce a la reducción del consumo energético y de la huella de carbono asociada a la infraestructura de TI.

Este proyecto busca superar la limitación actual de la infraestructura de nube actual de la Universidad del Valle de Guatemala, con la integración de Magnum, permitirá una mayor flexibilidad y eficiencia en la gestión de recursos, siendo capaz de desplegar y gestionar clústeres de contenedores de manera automatizada para casos de alta demanda y variabilidad de cargas de trabajo donde las máquinas virtuales sean insuficientes.

La implementación se llevará a cabo en 4 etapas. Primero, se instalará y configurará los servicios de Heat y Magnum en OpenStack. Luego, se integrará el UI de Magnum en Horizon y se prepararán las imágenes necesarias para nodos Kubernetes. Después, se crearán las plantillas y se desplegarán los clústeres verificando su buen funcionamiento. Finalmente, se instalará un sistema de monitoreo básico con Prometheus y Grafana.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar y probar a profundidad el servicio OpenStack Magnum dentro de la infraestructura OpenStack actualizada con el fin de proveer un servicio de orquestación de contenedores (CaaS).

4.2. Objetivos específicos

- Replicar y probar el entorno OpenStack previamente desplegado para un segundo nodo *compute*.
- Implementar y configurar el servicio de provisión y gestión automatizada de clústeres de contenedores de OpenStack Magnum, así como el *plugin* de Magnum en la interfaz gráfica Horizon.
- Crear exitosamente y verificar el funcionamiento de un clúster de contenedores utilizando OpenStack Magnum.
- Monitoreo básico del estado y rendimiento de los clústeres de contenedores desplegados utilizando OpenStack Magnum.
- Documentar todo el proceso de instalación para facilitar la replicación y gestionar errores.

CAPÍTULO 5

Alcance

El proyecto busca implementar y configurar el servicio de provisión y gestión automatizada de clústeres de contenedores Magnum en la infraestructura de OpenStack existente del departamento de Ingeniería Electrónica de la Universidad del Valle de Guatemala. El objetivo es expandir las capacidades de la infraestructura actual, la cual cuenta solamente con el servicio de provisión de máquinas virtuales. Para llevar a cabo este proyecto, se dividió en 4 etapas. La primera etapa tomó 5 meses, terminando en junio, y se enfocó en la verificación y replicación de la infraestructura existente, luego se realizó la instalación de OpenStack usando Kolla-Ansible [1], así como la realización de las configuraciones necesarias para la instalación y configuración del servicio de administración de claves Barbican y el servicio de provisión de clústeres de contenedores Magnum. Posteriormente, la segunda etapa tomó 1 mes y medio terminando a mediados del mes de agosto, se procedió a implementar y configurar el plugin de Magnum en el panel de control de Horizon, así como la preparación de las imágenes para nodos Kubernetes y la creación de plantillas para crear clústeres. Luego, la tercera etapa tomó 2 meses y medio, terminando a mediados del mes de octubre, se llevaron a cabo pruebas de despliegue y gestión de clústeres de contenedores para comprobar su correcto funcionamiento solucionando los problemas que se presentaron durante las pruebas. Finalmente, la cuarta etapa tomó medio mes, terminando a finales del mes de noviembre, por medio de los componentes Prometheus y Grafana, se implementó un sistema de monitoreo básico del estado y rendimiento de los clústeres de contenedores desplegados por Magnum. El proyecto se desarrolló sobre el hardware existente en la Universidad del Valle de Guatemala, específicamente dos computadoras de alto rendimiento (HPC). Se limitó a una implementación básica, sin características avanzadas de escalabilidad, alta disponibilidad ni monitoreo completo con alarmas personalizadas. Se usó solamente Kubernetes como motor de orquestación, no se implementaron políticas de seguridad avanzadas ni integración con otros servicios externos. Esta expansión de la infraestructura permitirá que tanto estudiantes como docentes puedan experimentar con tecnologías de contenedores y orquestación por medio de la creación, despliegue y gestión de Kubernetes de manera automatizada, ya sea, por medio de la interfaz gráfica de usuario (GUI) o de la línea de comandos (CLI), para diferentes propósitos académicos y de investigación. Se documentó todo el proceso de implementación y configuración, así como los resultados obtenidos, para que sirva como guía

para facilitar su replicación.

CAPÍTULO 6

Marco teórico

6.1. Computación en la nube

La computación en la nube es un modelo que permite la entrega de servicios informáticos que utilizamos diariamente, como lo son bases de datos, almacenamiento, servidores, software todo por medio del Internet. Este modelo elimina la necesidad que las empresas deban invertir en la compra y mantenimiento de una infraestructura física. Su foco central es la virtualización de recursos, permitiendo un servicio más flexible, eficiente y escalable [8]. Este concepto evolucionó de la computación en *grid* y la *utility computing*, cambiando la industria reduciendo costos e incluso permitiendo el acceso a tecnologías avanzadas a empresas en países con economías en desarrollo, ofreciendo una oportunidad para competir en el mercado global [9].

6.1.1. Modelos de servicio

Los modelos de servicio en la nube son diferentes enfoques para ofrecer recursos y servicios a los usuarios. Estos modelos definen el nivel de control y responsabilidad que tienen los usuarios sobre la infraestructura y las aplicaciones. Los tres modelos principales son:

- **SaaS (Software como servicio)**: servicio de aplicaciones (Microsoft 365) o páginas web (Google Workspace) que corren en una infraestructura de la nube. El consumidor no gestiona ni controla la infraestructura de la nube, red, sistema operativo, almacenamiento, capacidades de aplicaciones, ni otros recursos.
- **PaaS (Plataforma como servicio)**: servicio para desarrollar, ejecutar y gestionar aplicaciones y desplegarlas en una infraestructura de la nube adecuada con las herramientas y servicios necesarios (Google App Engine, Microsoft Azure).
- **IaaS (Infraestructura como servicio)**: servicio para provisionar recursos de computación, almacenamiento y redes en la nube, donde los usuarios despliegan y controlan

sistemas operativos o aplicaciones (Amazon EC2, Google Compute Engine). El consumidor tiene control total sobre el sistema operativo, las aplicaciones y los recursos utilizados, pero no sobre la infraestructura física [10].

6.1.2. Modelos de despliegue

Los modelos de despliegue en la nube definen cómo se implementan y gestionan los recursos y servicios en la nube. Existen tres modelos principales:

- **Nube pública**: establecida y gestionada por una empresa conocida como proveedor de servicios en la nube, ofrece recursos disponibles para el público general. Permite que cualquiera tenga acceso a recursos de computación y almacenamiento en la nube, sin necesidad de invertir en infraestructura física.
- **Nube privada**: operada y propiedad exclusiva de una organización, ya sea gestionada internamente o por un tercero. La organización invierte en la infraestructura y los recursos necesarios para ofrecer servicios en la nube para su propio uso.
- **Nube híbrida**: combina las mejores características de las nubes públicas, mejorando aspectos como desempeño, sostenibilidad, flexibilidad, y más, integrándola con la seguridad y el control de una nube privada [10].

6.1.3. Virtualización

Las máquinas virtuales (VM) son uno de los pilares de la computación en la nube. Las máquinas virtuales permiten implementar un sistema operativo completo aislado del hardware pero que se comporta como un sistema físico. Esto permite que un único servidor con múltiples procesadores pueda ejecutar varias VM. Esto permite aprovechar al máximo la capacidad de un servidor, reduciendo los ciclos de CPU no utilizados y reduciendo el desperdicio de energía. Al virtualizar un sistema operativo se reduce su carga de trabajo y permite su movilidad entre hosts físicos, así como su rápida instancia o finalización. Una VM se monta sobre un hipervisor, este presenta una plataforma de hardware virtualizado a las VM para gestionar su ejecución [9].

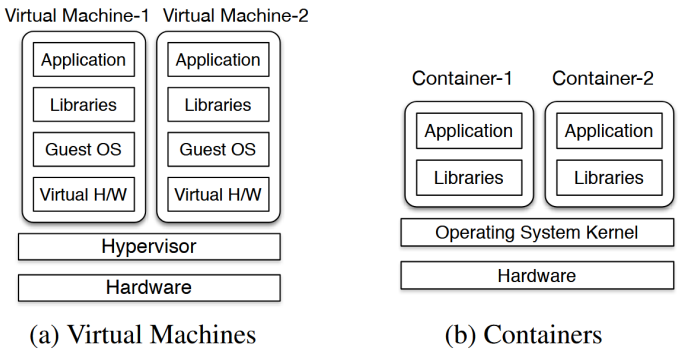
A finales de la década de 1960 y principios de la década de 1970, IBM [11] introdujo el concepto de virtualización en sus sistemas, evolucionando los ecosistemas de virtualización considerablemente. Existen claras diferencias entre los tipos de virtualización relevantes que han surgido en el tiempo, como lo son la paravirtualización y la virtualización completa. En 2005, VMware introdujo la paravirtualización como mecanismo de comunicación entre el hipervisor y el sistema operativo invitado. Este mecanismo ha dado paso a una nueva generación de paravirtualización transparente, en la que solo una versión binaria del sistema operativo puede ser ejecutado en el hardware, hasta que se conceptualizó la virtualización completa como un módulo de hardware. Este concepto permite implementar máquinas virtuales totalmente independientes del sistema operativo del host, lo que puede realizarse directamente sobre el hardware [12].

Históricamente, la virtualización ha sido una plataforma con soporte de hardware específico y permaneció en la computación mainframe hasta finales de la década de 1990. El desarrollo de Xen en 2003 y, posteriormente, el desarrollo de Intel VT-x y AMD-V, en 2005 y 2006 respectivamente, hicieron posible la virtualización de servidores x86 de alto rendimiento. Esto permitió una mayor utilización de servidores estableciendo un nuevo estándar y redujo considerablemente el tiempo necesario para aprovisionar nuevos servidores. La implementación interna a gran escala de la virtualización en varias grandes empresas impactó directamente el desarrollo de la computación en la nube [9].

6.1.4. Contenerización

La contenerización es una alternativa ligera a la virtualización. Como se muestra en la Figura 2, la contenerización se enfoca en extraer solo el sistema operativo, en lugar de virtualizar el hardware completo. Esta alternativa es una tendencia que permite ejecutar aplicaciones. Esencialmente, se basa en la capacidad de desarrollar, probar e implementar aplicaciones dentro de contenedores, garantizando la eficiencia de la comunicación entre dichos contenedores por su ligereza y portabilidad. Además, teniendo un tiempo de inicio más corto que las máquinas virtuales, por su capacidad para compartir recursos con los equipos host, lo que les permite optimizar la utilización de recursos y reducir su uso de CPU, memoria y almacenamiento. Sumado a esto, cada instancia de contenedor en ejecución es independiente, aislando sus procesos, los sistemas de archivos, espacio y recursos de hardware [7].

Figura 2. Diferencias entre máquinas virtuales y contenedores



Nota. Esta imagen muestra la infraestructura necesaria para el funcionamiento de una máquina virtual y un contenedor.

6.2. De la virtualización a la contenerización

La transición de la virtualización a la contenerización es el nuevo paso evolutivo de la computación en la nube. Debido a su capacidad para implementar aplicaciones rápidamente, los contenedores han impulsado un cambio en la forma en que se realizan las operaciones computacionales para la programación científica. Pueden alcanzar un rendimiento casi nativo

al probarse en aplicaciones con alto consumo de recursos de CPU y RAM. Han llegado a tener más relevancia en las aplicaciones de computación de alto rendimiento. Además, la virtualización basada en hipervisor presenta numerosas desventajas, las cuales se han eliminado mediante tecnologías de contenedores. Por ejemplo, se elimina la dependencia del hipervisor en la compilación, la degradación del rendimiento y los tiempos de arranque lentos de las máquinas virtuales [7].

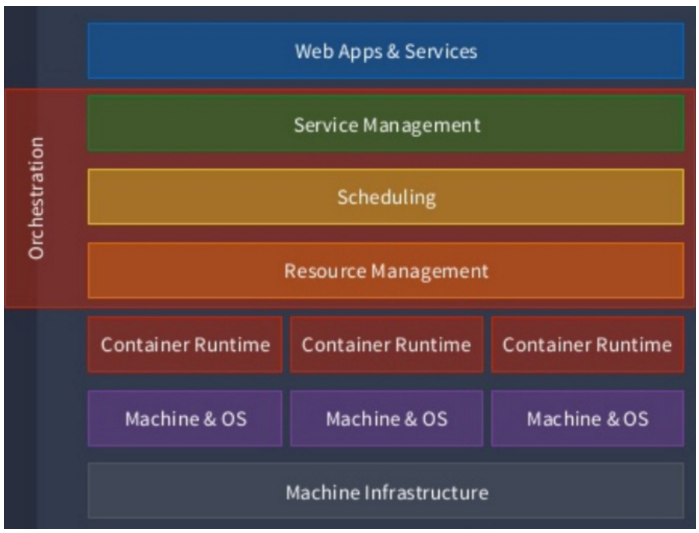
Numerosos estudios [11], [12], [13], [14] y [15] han comparado el hipervisor de máquinas virtuales con contenedores. Por ejemplo, el estudio [16] comparó KVM con contenedores Docker, evaluando velocidad de CPU, ancho de banda de memoria, espacio en disco y latencia de red. Esta evaluación se realizó usando diversos entornos, como hardware, máquinas virtuales, contenedores Linux computación en la nube e internet de las cosas. Según los resultados encontrados, los contenedores se pueden aprovechar mejor que las máquinas virtuales. Además, el hipervisor KVM tiene un alto consumo de la memoria. Docker solo virtualiza el nivel de aplicación, mientras que el hipervisor virtualiza todo el sistema operativo. Las aplicaciones contenedorizadas en hardware llegan a mejorar un 50 % comparado a las de las máquinas virtuales.

6.3. Orquestación de contenedores

La orquestación de contenedores permite definir flujos de trabajo automatizados de aprovisionamiento y gestión de cambios para garantizar siempre las políticas y los niveles de servicio acordados. La Figura 3 muestra un ejemplo de arquitectura de orquestación en capas, donde un conjunto de máquinas, a través de su kernel y el entorno de ejecución del contenedor, constituyen el sustrato de soporte. La estructura del motor de orquestación se encuentra en la parte superior y consta de tres capas: gestión de recursos, programación y gestión de servicios [17].

La capa de gestión de recursos gestiona los recursos de bajo nivel; en este caso, los elementos funcionales son los recursos que se pueden gestionar/componer, e incluyen: memoria, CPU/GPU, espacio en disco, volúmenes (es decir, la posibilidad de interactuar con el sistema de archivos del equipo local), volúmenes persistentes (es decir, la posibilidad de interactuar también con un sistema de archivos remoto en la nube), puerto e IP (es decir, configuración de puertos UDP/TCP e IP dentro de la red virtual de contenedores). Su objetivo es maximizar la utilización y minimizar la interferencia entre los contenedores que compiten por los recursos [17].

Figura 3. Capas de orquestación de contenedores



Nota. Esta imagen muestra las diferentes capas que componen un motor de orquestación de contenedores.

La capa de programación busca utilizar los recursos del clúster de forma eficiente. Normalmente recibe indicaciones proporcionadas por el usuario (por ejemplo, restricciones de ubicación, grado de replicación, etc.) como entrada y, a continuación, decide cómo ubicar todos los contenedores que componen las aplicaciones. Las capacidades más importantes incluyen:

- *Ubicación*: para controlar directamente las decisiones de programación.
- *Replicación/escalado*: para expresar el número de réplicas de microservicios.
- *Comprobación de la disponibilidad*: para incluir contenedor solo cuando esté listo para responder.
- *Resurrección*: para recrear procesos rápidos de larga duración cuyo trabajo requiere estar siempre en funcionamiento.
- *Reprogramación*: para reiniciar y programar automáticamente los contenedores bloqueados que se ejecutan en un nodo fallido.
- *Implementación continua*: para actualizar o degradar automáticamente la versión de la aplicación.
- *Coubicación*: para imponer restricciones de implementación, como coubicar contenedores para aprovechar la comunicación local entre procesos [17].

Finalmente, la capa de administración de servicios proporciona capacidades funcionales para crear e implementar aplicaciones complejas. Administra aspectos de alto nivel que incluyen:

- *etiquetas*: para adjuntar metadatos a objetos de contenedor
- *grupos/espacios de nombres*: para aislar contenedores y admitir multiinquilino
- *dependencias*: para expresar dependencias entre microservicios
- *balanceo de carga*: para dividir la carga entrante
- *verificación de preparación*: para que la aplicación esté disponible en línea solo cuando esté lista para aceptar tráfico entrante [17]

6.4. Plataformas de orquestación de contenedores

La tecnología de virtualización basada en contenedores ha sido investigada a fondo desde hace varios años atrás [18]. Debido a que los contenedores se implementan de forma muy densa, esto dificulta enormemente su gestión debido a la cantidad tan grande de contenedores, incluso cuando varios contenedores se ejecutan en diferentes hosts. Para gestionar eficientemente los contenedores, se utilizan herramientas de orquestación. Las aplicaciones con una alta complejidad y con múltiples contenedores que son implementadas en un clúster se pueden ejecutar mediante la expansión de las capacidades del motor de orquestación de contenedores [19]. Todas las tareas de gestión de contenedores, desde su desarrollo, implementación y programación, hasta su supervisión, se automatizan por medio de herramientas de orquestación de contenedores [12].

6.4.1. Kubernetes

Kubernetes es una plataforma de código abierto introducida por Google en 2014 para la gestión de aplicaciones en contenedores en un clúster de máquinas. Kubernetes se basa en una arquitectura maestro/esclavo, en el que un desarrollador envía una lista de aplicaciones a un nodo maestro y, posteriormente, la plataforma las despliega en los nodos esclavo y maestro. El nodo maestro representa un plano de control del clúster y puede replicarse para garantizar alta disponibilidad y tolerancia a fallos aprovechando la capa de programación. Los nodos esclavos (conocidos como *minions*) son aquellos nodos donde se ejecutan los contenedores de aplicaciones. Kubernetes proporciona una aplicación en contenedores como un conjunto de contenedores, cada uno de los cuales es específico para un único microservicio. Un pod es una unidad básica en Kubernetes y representa un grupo de contenedores coprogramados. Todos los contenedores dentro de un pod se controlan como una única aplicación y, por lo tanto, comparten el mismo entorno. Puede haber uno o más contenedores dentro de un pod. Dado que los pods se coprograman y se ejecutan en un contexto compartido, los contenedores dentro del pod se pueden escalar como una sola aplicación. La replicación de pods la gestiona el componente de Kubernetes llamado Controlador de Replicación, responsable de garantizar que un número determinado de pods preste un servicio específico. Si el estado actual se desvía de lo esperado, como en el caso de una interrupción en un nodo, el controlador de replicación inicia automáticamente la programación de una nueva instancia en un nodo esclavo diferente. El mecanismo del controlador de latidos no es demasiado agresivo y está diseñado con un sistema de notificación a suscriptores [17] [20].

6.4.2. Docker Swarm

Docker Swarm es una herramienta de agrupación y programación de contenedores de Docker. Permite a los operadores de TI gestionar un clúster de nodos Docker como un único sistema. Esto es importante porque crea un grupo cooperativo de máquinas que proporciona redundancia y habilita el mecanismo de conmutación por error si uno o más nodos experimentan una interrupción. El orquestador se basa en el modelo maestro/esclavo, donde el maestro domina. El maestro (conocido como Administrador) es el nodo responsable de programar los contenedores, mientras que el esclavo (comúnmente conocido como Agente) se encarga de lanzar los contenedores recibidos. La capa de programación gestiona tanto la redundancia como la ubicación. Para conectar contenedores alojados en diferentes nodos bajo la misma red, Docker Swarm ofrece una red superpuesta que aprovecha el túnel VXLAN (Red de Área Local Extensible, permite que una sola red física sea compartida por varias organizaciones diferentes) para crear una red virtual entre hosts. El nodo Administrador monitorea el estado de todos los nodos de un clúster (en Docker Swarm, este servicio se denomina descubrimiento y se basa en el mecanismo de latido que el módulo de red superpuesto utiliza para determinar si un demonio de Docker en un host remoto del clúster sigue funcionando). En entornos de nube, la elasticidad es una característica importante. Docker Swarm permite la escalabilidad detallada de parte del servicio, escalando uno o más servicios replicados, ya sea hacia arriba o hacia abajo, hasta el número deseado de réplicas [17] [7].

6.4.3. Apache Mesos

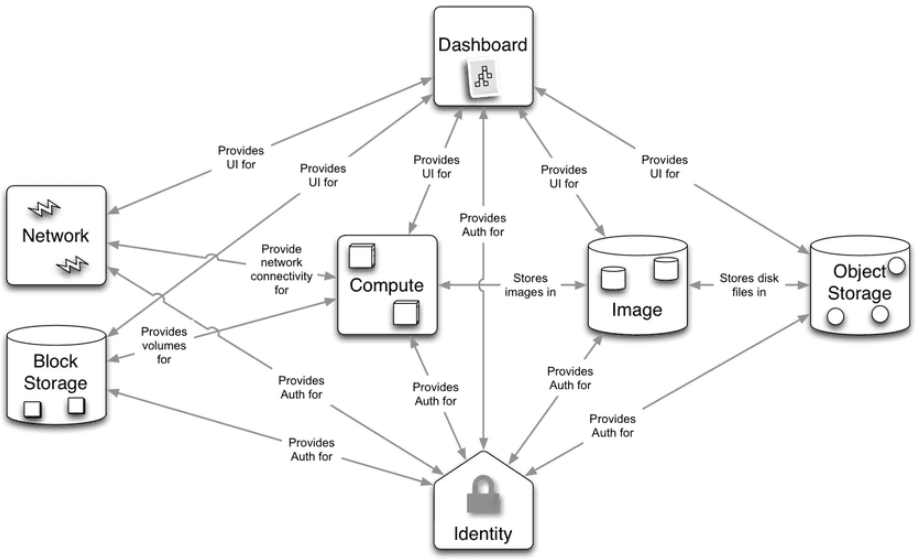
Apache Mesos es un proyecto pionero de código abierto desarrollado por la Universidad de California, Berkeley. Su arquitectura consiste en un patrón de diseño maestro/esclavo donde la ejecución de tareas se delega a los nodos esclavos. El proceso maestro, que se ejecuta en un nodo administrador del clúster, es responsable de la gestión y monitorización de toda la arquitectura del clúster. Por lo tanto, se comunica con los *frameworks* que programan tareas en los nodos esclavos. Las soluciones Mesos se desarrollan a menudo instalando, sobre un clúster Mesos, un sistema de gestión a nivel de aplicación llamado Marathon. Marathon interactúa con el componente maestro, proporcionando funcionalidades de orquestación a todo el clúster Mesos. De esta forma, en caso de fallos en los nodos esclavos, Marathon inicia una nueva instancia para garantizar la tolerancia a fallos. Mesos ofrece nodos maestros replicantes de alta disponibilidad para proporcionar mecanismos de conmutación por error en caso de fallos del maestro. Para ello, utiliza Apache Zookeeper, que consiste en un algoritmo de elección que selecciona un nuevo nodo para que desempeñe el rol de maestro [17] [7].

6.5. OpenStack

OpenStack es un software de código abierto para computación en la nube de cualquier tipo, ya sea pública, privada o híbrida. Su enfoque es proporcionar una plataforma sencilla de implementar, con alta escalabilidad y una gran variedad de funcionalidades [2]. OpenStack ofrece una amplia gama de servicios, como almacenamiento, redes, computación y orquestación de contenedores. Su arquitectura se basa en un diseño modular, donde cada componente se encarga de una función específica. Esto permite a los usuarios personalizar

su implementación de OpenStack dependiendo de lo que necesiten. Al ser modular, todos los servicios pueden ser instalados y configurados de forma independiente, pero trabajan juntos para ofrecer una solución de nube versátil y adaptable.

Figura 4. Diagrama de bloques de la arquitectura de OpenStack



Nota. Esta imagen muestra los componentes que conforman una arquitectura básica de OpenStack.

En la Figura 4 cada uno de los componentes se encarga de una función específica [21]. Estos componentes son:

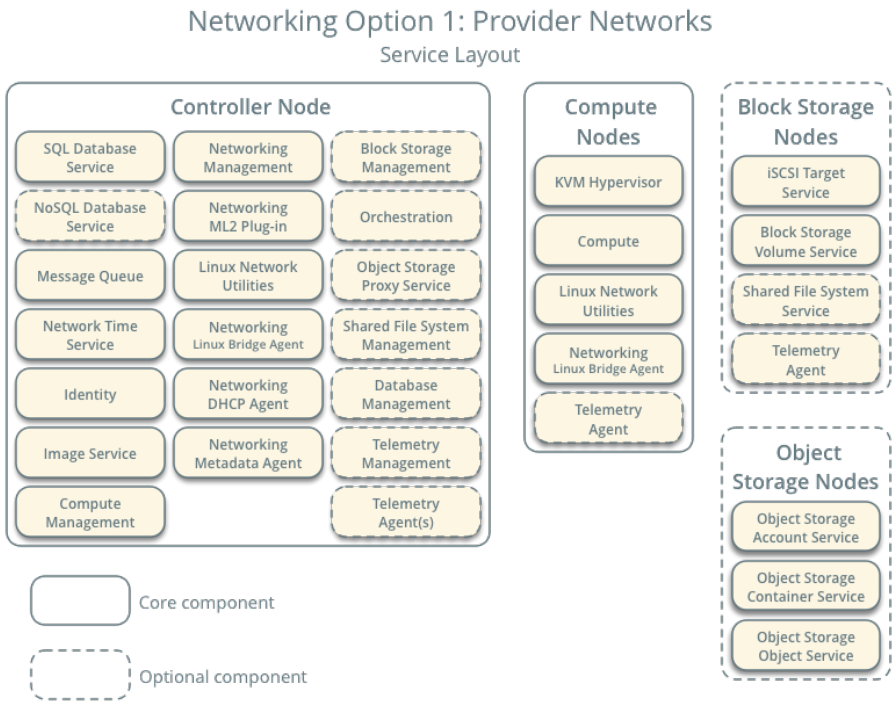
- **Keystone:** se encarga de la asignación de los usuarios y sus permisos, así como de la autenticación y autorización de los mismos.
- **Glance:** ofrece un servicio para el descubrimiento, registro y distribución de discos e imágenes de servidor. Las imágenes almacenadas se pueden utilizar como modelo para desplegar nuevas instancias permitiendo tener un numero limitado, solamente por el espacio de almacenamiento disponible, de copias de seguridad.
- **Nova:** se encarga de controlar la estructura *Cloud Computing*, este gestiona y automatiza los recursos de computación. Trabaja con tecnologías de virtualización como KVM, Xen y LXM.
- **Neutron:** es el sistema que gestiona redes y direcciones IP. Permitiendo crear redes virtuales, asignar direcciones IP y gestionar la conectividad entre instancias.
- **Cinder:** ofrece dispositivos de almacenamiento por medio de bloques para que sean utilizados con instancias de OpenStack Compute, permitiendo gestionar sus almacenamiento dependiendo sus necesidades.

- **Swift:** es un sistema de almacenamiento escalable y redundante que distribuye objetos y archivos en múltiples discos ubicados en distintos servidores. Garantizando la replicación y la integridad de los datos en todo el clúster.
- **Horizon:** proporciona una interfaz gráfica para que administradores y usuarios puedan acceder y gestionar los recursos de OpenStack. Permitiendo incluso la integración de productos de terceros [2].

6.5.1. Arquitectura de OpenStack

La arquitectura de OpenStack se basa en un diseño modular, permitiendo la existencia de diversas implementaciones dependiendo de las necesidades del usuario. Un ejemplo de estas comprende nodos de control, nodos de cómputo, nodos de almacenamiento de bloques y nodos de almacenamiento de objetos. Cada uno de estos nodos se encarga de una función específica [2].

Figura 5. Ejemplo de diagrama de nodos de la arquitectura de OpenStack



Nota. En Esta imagen se muestra un ejemplo de la distribución de nodos de una infraestructura de OpenStack. Sin embargo, la distribución y cantidad de nodos puede variar dependiendo de las necesidades y disponibilidad del usuario.

- **Nodos de control:** Se encargan de gestionar los recursos del clúster, así como la comunicación entre los diferentes componentes. Estos nodos son responsables de la gestión de la infraestructura y la orquestación de los servicios.

- **Nodos de cómputo:** Se encargan de ejecutar las instancias de OpenStack, proporcionando los recursos de computación necesarios para el funcionamiento del clúster. Estos nodos son responsables de la ejecución de las instancias y la gestión de los recursos de computación.
- **Nodos de almacenamiento de bloques:** Se encargan de gestionar el almacenamiento de bloques, proporcionando un servicio de almacenamiento persistente para las instancias de OpenStack. Estos nodos son responsables de la gestión del almacenamiento de bloques y la replicación de los datos.
- **Nodos de almacenamiento de objetos:** Se encargan de gestionar el almacenamiento de objetos, proporcionando un servicio de almacenamiento escalable y redundante para los datos. Estos nodos son responsables de la gestión del almacenamiento de objetos y la replicación de los datos [22].

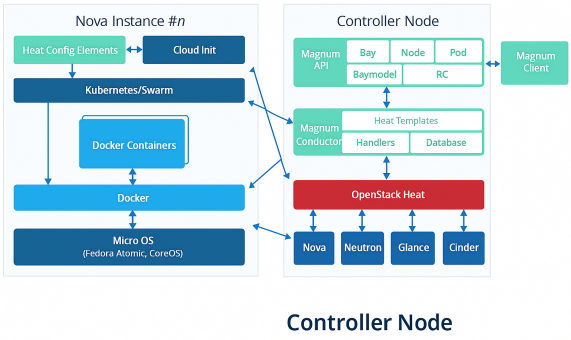
6.5.2. Heat

Heat es un servicio para orquestar aplicaciones compuestas en la nube. Heat utiliza plantillas para describir la infraestructura y los servicios necesarios para ejecutar una aplicación. Estas plantillas se definen en formato YAML o JSON y permiten a los usuarios crear, actualizar y eliminar recursos de forma automatizada. Heat proporciona una API RESTful (Interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de Internet [23]) para interactuar con las plantillas y realizar operaciones de orquestación. Además, Heat permite la integración con otros servicios de OpenStack, como Nova, Neutron y Cinder, para gestionar los recursos de computación, redes y almacenamiento de forma coordinada [24].

6.5.3. Magnum

Magnum es un servicio de OpenStack que permite la creación y gestión de clústeres de contenedores de forma sencilla y eficiente. Magnum proporciona una API RESTful para interactuar con los clústeres de contenedores y permite a los usuarios crear, actualizar y eliminar clústeres de forma automatizada. Magnum es compatible con diferentes tecnologías de contenedores, como Docker Swarm, Kubernetes y Mesos, lo que permite a los usuarios elegir la tecnología que mejor se adapte a sus necesidades. Magnum también proporciona una interfaz gráfica por medio de un plugin de Horizon para la gestión de clústeres de contenedores, lo que facilita la administración y supervisión de los clústeres. Además, Magnum permite la integración con otros servicios de OpenStack, como Nova, Neutron y Cinder [25].

Figura 6. Diagrama de funcionamiento de OpenStack Magnum



Nota. Esta imagen muestra la interacción entre los diferentes componentes de OpenStack Magnum para la gestión de clústeres de contenedores.

Como se puede ver en la Figura 6, Magnum comienza la creación de clústeres cuando el usuario le pide un clúster (un *Bay*) por medio de *Magnum Client* al contenedor *Magnum API*. El contenedor *Magnum Conductor* toma esta solicitud, la traduce usando una plantilla de Heat y lo manda al componente Heat. Heat se encarga de crear el clúster, le pide a Nova que cree las máquinas virtuales (las instancias de Nova), a Neutron que configure las redes y a Glance que provea la imagen del sistema operativo (Fedora CoreOS). Finalmente, en cada máquina virtual creada, se ejecutan *scripts* de *Cloud Init* que instalan automáticamente Docker y Kubernetes, formando así el clúster funcional listo para desplegar contenedores.

6.5.4. Barbican

Barbican es un servicio de OpenStack que proporciona una solución segura para la gestión de secretos, como claves de cifrado, certificados y contraseñas. Barbican utiliza una API RESTful para interactuar con los secretos y permite a los usuarios crear, almacenar y recuperar secretos de forma segura. Barbican utiliza un enfoque de seguridad basado en roles, lo que permite a los administradores definir políticas de acceso para los secretos. Además, Barbican permite la integración con otros servicios de OpenStack, como Nova, Neutron y Cinder, para gestionar los secretos de forma coordinada [26].

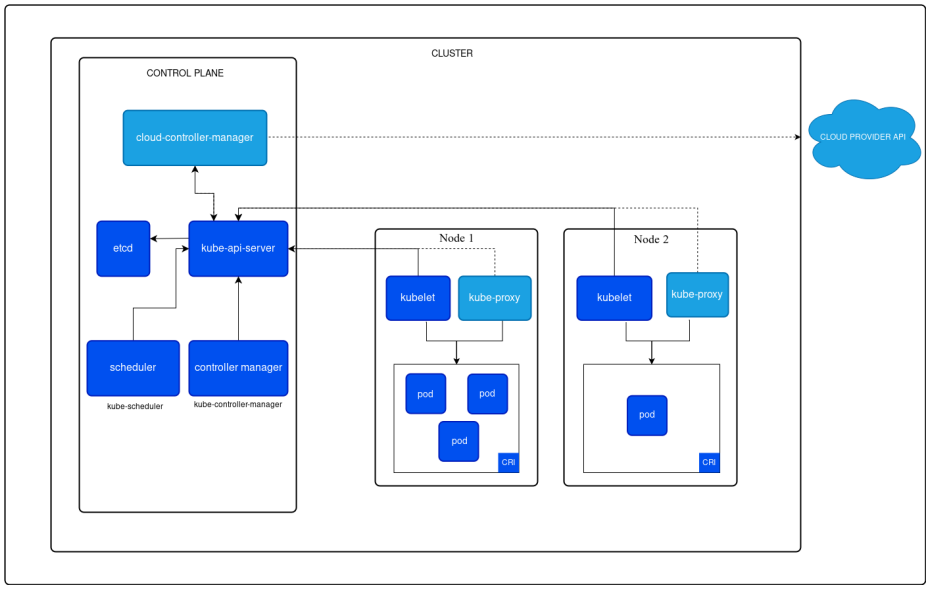
6.6. Clúster Kubernetes

Un clúster Kubernetes se define como una colección de almacenamiento y red de recursos de computación que son utilizados por Kubernetes para correr varias cargas de trabajo o como una sola unidad de computadoras que están conectadas para trabajar juntas y las cuales son provisionadas con componentes de Kubernetes. Un clúster consiste en dos tipos de instancias: maestro y nodos. Una instancia puede ser una máquina física o virtual, dependiendo de la implementación [27].

Los componentes del plano de control de clúster Kubernetes hacen decisiones globales sobre el clúster (por ejemplo, programación) al igual que detecta y responde a eventos en el clúster (por ejemplo, iniciar un nuevo pod cuando la carga de trabajo aumenta).

- **Kube-apiserver:** es el componente del plano de control que expone la API de Kubernetes.
- **Kube-scheduler:** es el componente del plano de control que asigna los pods a los nodos.
- **Kube-controller-manager:** es el componente del plano de control que ejecuta los controladores de Kubernetes.
- **etcd:** es el almacén de claves-valor distribuido que almacena la configuración del clúster y el estado de los recursos.
- **Cloud-controller-manager:** es el componente del plano de control que interactúa con el proveedor de servicios en la nube [28].

Figura 7. Componentes de un clúster Kubernetes



Nota. Esta imagen muestra los componentes principales de un clúster Kubernetes y sus interacciones.

6.7. Monitoreo y gestión de clústeres Kubernetes

6.7.1. Monitoreo básico de clústeres Kubernetes

El monitoreo básico de clústeres Kubernetes en OpenStack por medio de Magnum se realiza con la implemnetación de componentes clave como *metrics-server*, el cual proporciona

las funcionalidades esenciales para controlar solicitudes de API, incluyendo *kubectl top* que es un comando que utiliza métricas básicas. Con el componente Prometheus, se puede habilitar el monitoreo para recolectar métricas como CPU, memoria, disco, etc. Dependiendo de *monitoring enabled* se puede integrar métricas personalizadas con la API *custom.metrics.k8s.io*. Utilizando etiquetas como *prometheus operator chart tag* para gestionar la versiones. Con el componente Grafana, se puede visualizar las métricas recolectadas en un panel de control. Esta implementación permite a los usuarios monitorear el estado y el rendimiento de sus clústeres Kubernetes de manera efectiva, dando un enfoque básico pero funcional y escalable para la supervisión de recursos y el rendimiento de las aplicaciones desplegadas en el clúster [29].

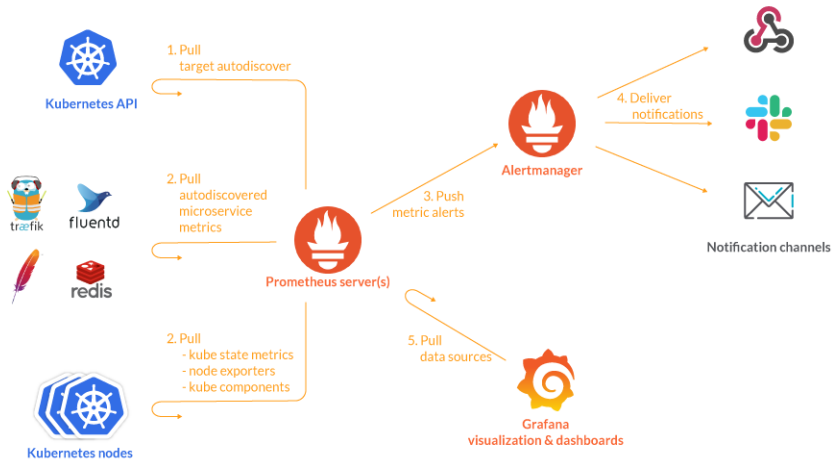
6.7.2. Monitoreo completo de clúster Kubernetes

La instalación de Prometheus proporcionado con la etiqueta *monitoring enabled* es un servicio multicomponente. Esta instalación es gestionada con el gráfico Helm de *prometheus operator* y sus componentes son:

- **Prometheus** colección de datos para el nodo kubelet
  - **nodo-exporter:** fuente de datos para el nodo kubelet.
  - **kube-state-metrics:** exporta métricas del estado de los recursos de Kubernetes.
- **alertmanager:** gestiona las alertas generadas por Prometheus.
- **Grafana:** proporciona una interfaz gráfica para visualizar las métricas y crear paneles de control personalizados [29] [30].

El monitoreo completo de clústeres Kubernetes en OpenStack por medio de Magnum ofrece una infraestructura de métricas altamante personalizables por medio de etiquetas, que permiten ajustar los parámetros según dependiendo de lo requerido. una de estar es *grafana admin passwd* el cual permite establecer una contraseña para el usuario administrador de Grafana. otra etiqueta es *monitoring retention days* que permite establecer el número de días que se retendrán las métricas recolectadas, junto con *monitoring interval seconds* y *monitoring retention size* que permiten ajustar el intervalo de muestreo y limitar el tamaño de retención de las métricas, respectivamente. Pero entre las etiquetas una de las más importantes es *prometheus adapter configmap* que permite configurar el adaptador de Prometheus, ofreciendo un control exhaustivo sobre un monitoreo avanzado y robusto de los clústeres Kubernetes [29].

Figura 8. Arquitectura de monitoreo completo de Kubernetes



Nota. Esta imagen ilustra la arquitectura de monitoreo completo de un clúster Kubernetes, destacando los componentes clave y su interconexión.

CAPÍTULO 7

Guía de instalación de Magnum

En este caso se utiliza Kolla-Ansible para la instalación y configuración del entorno OpenStack, un proyecto que simplifica el despliegue de OpenStack mediante el uso de contenedores Docker y Ansible [1], por lo que la instalación de Magnum se realiza como parte de este despliegue de OpenStack. Magnum requiere que el servicio de orquestación Heat sea instalado y configurado, ya que Magnum utiliza Heat para la provisión y gestión de los clústeres de contenedores, como componente adicional instalar el componente Barbican para la gestión de certificados TLS, aunque no es requerido, después de varias pruebas y retroalimentación de otros usuarios, se recomienda instalarlo para evitar problemas con la creación de clústeres que utilizan TLS.

7.1. Instalación y configuración de Magnum

Antes de iniciar necesitamos ingresar el comando para entrar en el modo super usuario, necesario para realizar ciertas operaciones administrativas en el sistema:

Cuadro 1. Comando para entrar en modo super usuario

```
$ sudo su
```

Te pedirá la contraseña del usuario, ingresas la contraseña y presionas enter.

Cuadro 2. Solicitud de contraseña para entrar en modo super usuario

```
[sudo] password for <usuario>:
```

Activar y entrar al entorno virtual en donde está desplegado Kolla-Ansible OpenStack:

Cuadro 3. Comando para activar el entorno virtual

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

El camino */root/openstack/os-venv/bin/activate* es el que creó el ambiente virtual en este proyecto.

Para instalar y configurar Magnum y los componentes adicionales por medio de Kolla-Ansible se debe modificar el archivo *globals.yml* que se encuentra en la ruta */etc/kolla/*. Este archivo contiene la configuración global de Kolla-Ansible en el cual se define que servicios serán instalados, se debe habilitar Barbican, Heat, Magnum y la interfaz de usuario de Horizon para Magnum. (Heat por defecto ya viene habilitado en Kolla-Ansible, ya que es parte de los componentes esenciales).

Cuadro 4. Configuraciones necesarias en globals.yml

```
enable_barbican: true
enable_magnum: true
enable_horizon_magnum: true
```

También es necesario que el servicio de Magnum tenga habilitado y configurado las siguientes opciones en el archivo *magnum.conf*, se debe crear este archivo en una de las rutas de configuración de Kolla-Ansible

Para saber que rutas de configuración utiliza Kolla-Ansible, se puede ejecutar el comando *kolla-ansible deploy*. Antes de desplegar OpenStack con Magnum, es recomendable ejecutar los prechecks para verificar que la configuración sea correcta:

Cuadro 5. Comando para ejecutar los prechecks de Kolla-Ansible

```
# kolla-ansible prechecks -i <inventario>
```

inventario: archivo de inventario que describe la configuración de los nodos a utilizar *all-in-one* o *multinode*, si no se encuentra en el directorio donde esta el archivo debe escribirse la ruta y el archivo. Por ejemplo, *../ansible/inventory/all-in-one*.

Luego, para desplegar únicamente el servicio de Magnum y obtener las rutas de configuración, se debe ejecutar el siguiente comando:

Cuadro 6. Comando para desplegar el servicio de Magnum

```
# kolla-ansible deploy -i <inventario> -t magnum,barbican -vvv
```

- **-t magnum,barbican**: indica que solo se desplegará el servicio de Magnum y Barbi-

can.

- **-vvv**: modo verboso para obtener más detalles del despliegue.

En la salida del comando, se debe buscar la sección:

Cuadro 7. Sección del despliegue donde se encuentran las rutas de configuración de Magnum

```
TASK [magnum : Copying over magnum.conf] ******************************
```

Donde se encuentra una parte que indica las rutas de configuración que utiliza Kolla-Ansible para Magnum:

Cuadro 8. Rutas de configuración que utiliza Kolla-Ansible para Magnum

```
"source": [
    "/root/openstack/share/kolla-ansible/ansible/roles/magnum/templates/
        magnum.conf.j2",
    "/etc/kolla/config/global.conf",
    "/etc/kolla/config/magnum.conf",
    "/etc/kolla/config/magnum/magnum-conductor.conf",
    "/etc/kolla/config/magnum/localhost/magnum.conf"
]
```

Dependiendo de la ruta que se elija, el archivo de configuración tendrá prioridad sobre las demás rutas, al igual que los nodos donde se aplicará la configuración. En este caso se utilizará la ruta */etc/kolla/config/magnum.conf*.

Por lo general en los despliegues de Kolla no se crea la carpeta *config*, por lo que si no existe, debemos crearla en la ruta */etc/kolla/* y crear un archivo llamado *magnum.conf* dentro de esta archivo escribimos esta configuración:

Cuadro 9. Configuraciones necesarias en magnum.conf

```
[trust]
cluster_user_trust = True

[cluster_template]
kubernetes_allowed_network_drivers = calico
kubernetes_default_network_driver = calico
```

cluster_user_trust: permite que los nodos del clúster confíen en el usuario del clúster para realizar operaciones administrativas. Sin esta opción habilitada, los nodos del clúster no podrán realizar ciertas operaciones que requieren privilegios elevados, causando errores durante la creación y gestión de los clústeres.

kubernetes_allowed_network_drivers: especifica los controladores de red permitidos para los clústeres de Kubernetes.

kubernetes_default_network_driver: define el controlador de red predeterminado para los clústeres de Kubernetes.

Se utiliza **Calico** como controlador de red, ya que es el controlador de red que mejor trabaja con Kubernetes y es compatible con Magnum.

Si se desea agregar Magnum a un entorno de OpenStack ya desplegado por Kolla-Ansible, debes volver a desplegarlo con la nueva configuración.

Finalmente, desplegamos OpenStack con las configuraciones necesarias para Magnum, se debe ejecutar el siguiente comando:

Cuadro 10. Comando para desplegar el servicio de Magnum

```
# kolla-ansible reconfigure -i <inventario> -t magnum
```

Verificar que los contenedores de Magnum se hayan desplegado correctamente:

Cuadro 11. Comando para verificar los contenedores de Magnum

```
# docker ps | grep magnum
```

Deberían verse dos contenedores en ejecución, uno para el *magnum-api* y otro para el *magnum-conductor*:

Cuadro 12. Contenedores de Magnum en ejecución

```
quay.io/openstack.kolla/magnum-conductor:master-ubuntu-noble ... up 2
    hours (healthy)
quay.io/openstack.kolla/magnum-api:master-ubuntu-noble       ... up 2
    hours (healthy)
```

Asegurar que Magnum tenga las configuraciones necesarias revisando los archivos configuración de los contenedores de Magnum en */etc/kolla/magnum-conductor/magnum.conf* y */etc/kolla/magnum-api/magnum.conf* en el nodo donde se desplegó el servicio de Magnum.

Deberían verse las configuraciones de las siguiente manera:

Cuadro 13. Configuraciones vistas en magnum.conf

```
[trust]
...
cluster_user_trust = True

[cluster_template]
kubernetes_allowed_network_drivers = calico
kubernetes_default_network_driver = calico
```

Conectar la línea de comandos al entorno OpenStack con las credenciales de administrador:

Cuadro 14. Comando para conectar la línea de comandos al entorno OpenStack

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

La ruta donde se encuentra este archivo por lo general es la misma, pues se crea al desplegar kolla-ansible.

Ya completado el despliegue, se debe instalar el cliente de Python de Magnum. Es necesario para poder interactuar con Magnum desde la línea de comandos:

Cuadro 15. Comando para instalar el cliente de Python de Magnum

```
# apt install python-magnumclient
```

Ya terminada la instalación y configuración de Magnum, se debe verificar que el servicio esté activo:

Cuadro 16. Comando para verificar el estado del servicio de Magnum

```
# openstack coe service list
```

Este comando lista los servicios de *Container Orchestration Engine (COE)* disponibles en OpenStack.

Si el servicio está activo, se debe obtener una salida similar a la siguiente:

Cuadro 17. Salida con el estado del servicio de Magnum

```
+----+----------------+------------------+-------+----------+ ...
| id | host           | binary           | state | disabled | ...
+----+----------------+------------------+-------+----------+ ...
|  1 | 192.168.74.8   | magnum-conductor | up    | False    | ...
+----+----------------+------------------+-------+----------+ ...
```

7.2. Descarga y creación de imagen Fedora CoreOS

Magnum requiere una imagen de sistema operativo para poder crear los nodos de trabajo. En este caso se utilizará la imagen de Fedora CoreOS, ya que, es una distribución ligera y optimizada para contenedores y la única compatible con Kubernetes.

Identificar la arquitectura de nuestro equipo:

Cuadro 18. Comando para identificar la arquitectura del equipo

```
# uname -m
```

En este caso por ser x86_64, debe ingresar a la url de la arquitectura x86_64 de Fedora CoreOS:

Podemos obtener la imagen de Fedora CoreOS estable para Openstack con dos métodos:

7.2.1. Primer método

En este método utilizamos la API de Fedora CoreOS para obtener la URL de la última imagen estable. No es muy recomendable, ya que si la versión es muy reciente se pueden presentar problemas que no se tienen en versiones anteriores ya probadas. Pero es útil para obtener siempre la última versión disponible.

El siguiente comando redirige a la API de Fedora CoreOS y extrae la URL de la imagen estable más reciente en formato *.qcow2.xz* para OpenStack. Establecer la variable *URL* con la ubicación de la imagen de Fedora CoreOS, esta variable solo sirve para no escribir todo de corrido, se puede evitar si se desea.

Cuadro 19. Comando para establecer la variable URL con la ubicación de la imagen de Fedora CoreOS

```
# URL=$(curl -fsSL https://builds.coreos.fedoraproject.org/streams/
    stable.json \
| jq -r '.architectures.x86_64.artifacts.openstack.formats["qcow2.xz"].
    disk.location')
```

- curl: herramienta de línea de comandos para transferir datos desde o hacia un servidor.
- jq: herramienta de línea de comandos para procesar datos JSON.
- https://builds.coreos.fedoraproject.org/streams/stable.json: URL de la API de Fedora CoreOS que proporciona información sobre las imágenes disponibles.
- .architectures.x86_64.artifacts.openstack.formats["qcow2.xz"].disk.location: Ruta en el JSON que contiene la URL de la imagen de Fedora CoreOS en formato *.qcow2.xz* para OpenStack.

Para verificar que la variable *URL* tenga la ubicación correcta:

Cuadro 20. Comando para verificar la variable URL

```
# echo "$URL"
```

Descargar la última imagen disponible de Fedora CoreOS para OpenStack:

Cuadro 21. Comando para descargar la imagen de Fedora CoreOS

```
# curl -fL -o fedora-coreos-openstack-latest.qcow2.xz "$URL"
```

Este comando descargará la imagen de Fedora CoreOS en formato *.qcow2.xz*.

7.2.2. Segundo método

En este método se accede directamente a la página de descargas de Fedora CoreOS `https://fedoraproject.org/es/coreos/download/` y buscar la imagen estable deseada para OpenStack. Después de varias pruebas y retroalimentación de otros usuarios, se recomienda usar la versión *40.20240616.3.0*.

Descargar la imagen correspondiente a esta versión.

Cuadro 22. Comando para descargar la imagen de Fedora CoreOS 40.20240616.3.0

```
# wget https://builds.coreos.fedoraproject.org/prod/streams/stable/
    builds/40.20240616.3.0/x86_64/fedora-coreos-40.20240616.3.0-
    openstack.x86_64.qcow2.xz
```

7.2.3. Creación de imagen en Glance

Ya con la imagen descargada, descomprimir el archivo *.xz* para obtener la imagen en formato *.qcow2*:

Cuadro 23. Comando para descomprimir la imagen de Fedora CoreOS

```
# unxz fedora-coreos-...-openstack.x86_64.qcow2.xz
```

...: Dependiendo del método de descarga, el nombre del archivo puede variar, por ejemplo: *fedora-coreos-openstack-latest.qcow2.xz* o *fedora-coreos-40.20240616.3.0-openstack.x86_64.qcow2.xz*.

El archivo descomprimido se llama *fedora-coreos-...-openstack.x86_64.qcow2* y se eliminará el archivo comprimido.

Mover el archivo a la carpeta deseada:

Cuadro 24. Comando para mover la imagen de Fedora CoreOS

```
# mv fedora-coreos-...-openstack.x86_64.qcow2 /<Carpeta_deseada>
```

<Carpeta_deseada>: Dirección donde se desea mover la imagen, por ejemplo: */home/usuario/fedora-coreos/*.

Finalmente, crear la imagen en Glance con la siguiente estructura:

Cuadro 25. Comando para crear la imagen de Fedora CoreOS en Glance

```
# openstack image create fedora-coreos-magnum \
  --public \
  --disk-format qcow2 \
  --container-format bare \
  --property os_distro='fedora-coreos' \
  --file fedora-coreos-...-openstack.x86_64.qcow2
```

Donde:

- **fedora-coreos-magnum**: es el nombre de la imagen que se creará en Glance (Se recomienda usar este nombre).
- –**public**: indica que la imagen será pública y podrá ser utilizada por cualquier usuario de OpenStack (Eliminar esta opción si solo el administrador usará esta imagen).
- –**disk-format**: indica el formato del disco de la imagen.
- –**container-format**: indica el formato del contenedor de la imagen. Este formato es adecuado para imágenes de sistema operativo que no requieren un sistema de archivos específico.
- –**property os_distro**: indica que la imagen es de la distribución Fedora CoreOS (Necesario para que magnum la reconozca).
- –**file**: indica el archivo de la imagen que se creará en Glance.

Se debe obtener una salida similar a la siguiente:

Cuadro 26. Salida con los detalles de la imagen creada en Glance

```
+------------------+-----------------------------------------------+
| Field            | Value                                         |
+------------------+-----------------------------------------------+
| container_format | bare                                          |
| created_at       | 2025-08-09T19:43:54Z                          |
| disk_format      | qcow2                                         |
| file             | /v2/images/e6c143cc-2e88-454b-ba90-...        |
| id               | e6c143cc-2e88-454b-ba90-079d1ff45def          |
| min_disk         | 0                                             |
| min_ram          | 0                                             |
| name             | fedora-coreos-magnum                          |
| owner            | 431b4b0e9d594207a4ef1df4bbe8463a              |
| properties       | os_distro='fedora-coreos', os_hidden='False',|
|                  | owner_specified.openstack.md5='',             |
|                  | owner_specified.openstack.object=...,         |
|                  | owner_specified.openstack.sha256=''           |
| protected        | False                                         |
| schema           | /v2/schemas/image                             |
| status           | queued                                        |
| tags             |                                               |
| updated_at       | 2025-08-09T19:43:54Z                          |
| visibility       | public                                        |
+------------------+-----------------------------------------------+
```

7.3. Creación de redes

Para poder crear una plantilla de clúster, se debe crear una red externa con salida a Internet que permita la comunicación entre el clúster y servicios externos así como la descarga de paquetes. También se puede crear una red privada para los nodos del clúster.

7.3.1. Creación de red externa

Antes de crear la red externa, se debe verificar que en el archivo de *globals.yml* de Kolla-Ansible neutron tenga un interfaz externa asignada. Es recomendable tener dos NICs (*Network Interface Cards*) en el equipo, una para la red del dispositivo y otra para la red externa que se usará para conectar las máquinas virtuales a Internet. Para esto, se debe verificar que la siguiente opción esté habilitada:

Cuadro 27. Opción de interfaz externa en globals.yml

```
neutron_external_interface: "ethX"
```

En donde *ethX* es el nombre de la interfaz externa que se utilizó para las máquinas virtuales.

Se puede verificar las interfaces disponibles con el siguiente comando:

Cuadro 28. Comando para verificar las interfaces de red

```
# ip addr
```

Se mostrarán de la siguiente manera:

Cuadro 29. Interfaces de red disponibles

```
2:  eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    link/ether f2:3c:93:60:50:30 brd ff:ff:ff:ff:ff:ff
    inet 178.79.148.108/24 brd 178.79.148.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2a01:7e00::f03c:93ff:fe60:5030/64 ...
       valid_lft 5155sec preferred_lft 1555sec
    inet6 fe80::f03c:93ff:fe60:5030/64 scope link
       valid_lft forever preferred_lft forever
```

En este caso, la interfaz externa es *eth0*. Dependiendo de la cantidad de interfaces de red que tenga el equipo puede que la interfaz externa sea *ensp516*, *ensp865*, etc. Asegurar que la interfaz no tenga una dirección IP asignada, ya que esta interfaz se utilizará solo para conectar la red de Neutron al router. Siendo capaz de asignar direcciones IP del router a las máquinas virtuales que se creen en OpenStack.

Luego de verificar la interfaz externa debemos ver que en el archivo *ml2_ conf.ini* generado en la ruta */etc/kolla/neutron-server/*, tenga asignada una red física:

Cuadro 30. Opción de red física en ml2_conf.ini

```
[ml2_type_flat]
flat_networks = physnet1
```

Finalmente, después de verificar que la interfaz externa y la red física están correctamente configuradas, se debe crear la red externa:

Cuadro 31. Comando para crear la red externa

```
# openstack network create public \
  --provider-network-type flat \
  --external \
  --share \
  --provider-physical-network physnet1 \
  --project service
```

Donde:

- **public**: nombre de la red externa

- –**provider-network-type**: el tipo de red que será utilizada (en esta red, **flat**)
- –**external**: indica que es una red externa (Para que los nodos se identifiquen la red como salida a Internet)
- –**provider-physical-network**: la red física que utilizará la red externa (por lo verificado previamente, *physnet1*)
- –**project**: el proyecto al que pertenece la red externa, ya que es un servicio, *service*

Cuadro 32. Salida con los detalles de la red externa creada

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2025-08-10T10:09:04Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 372170ca-7d2e-48a2-8449-670e4ab66c23 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| mtu                       | 1450                                 |
| name                      | public                               |
| port_security_enabled     | True                                 |
| project_id                | 224c32c0dd2e49cbaadfd1cda069f149     |
| provider:network_type     | flat                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 3                                    |
| qos_policy_id             | None                                 |
| revision_number           | 4                                    |
| router:external           | External                             |
| segments                  | None                                 |
| shared                    | TRUE                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| updated_at                | 2025-08-10T10:09:04Z                 |
+---------------------------+--------------------------------------+
```

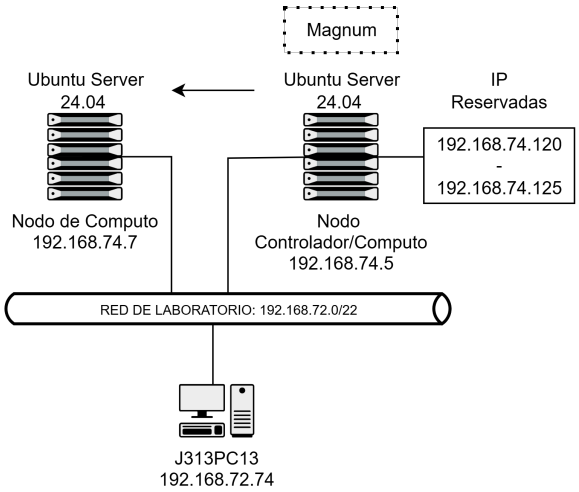
Ya creada la red externa, se debe crear una subred externa:

Cuadro 33. Comando para crear la subred externa

```
# openstack subnet create public_subnet \
  --network public \
  --gateway 192.168.72.1 \
  --allocation-pool start=192.168.74.120,end=192.168.74.125 \
  --subnet-range 192.168.72.0/22 \
  --ip-version 4
```

La red externa de Neutron debe configurarse con la misma mascara de subred y puerta de enlace que la red real que se usó para conectar OpenStack a Internet. La puerta de enlace de la red de la Universidad del Valle de Guatemala es 192.168.72.1 y un rango de direcciones *IP* disponibles en el momento en que se creó la subred externa fueron 192.168.74.120-192.168.74.125, verificar si siguen disponibles.

Figura 9. Topología de red de laboratorio con red externa



Nota. Figura de la topología de red del laboratorio, donde se muestra la conexión de la red externa con la red física a través de la interfaz externa configurada en Neutron.

Se debe obtener un mensaje de confirmación parecido a este:

Cuadro 34. Salida con los detalles de la subred externa creada

```
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.74.120-192.168.74.125        |
| cidr              | 192.168.72.0/22                      |
| created_at        | 2025-08-10T10:46:15Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.72.1                         |
| host_routes       |                                      |
| id                | 04185f6c-ea31-4109-b20b-fd7f935b3828 |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | public_subnet                        |
| network_id        | 372170ca-7d2e-48a2-8449-670e4ab66c23 |
| project_id        | d9e40a0aff30441083d9f279a0ff50de     |
| revision_number   | 2                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| updated_at        | 2025-08-10T10:46:15Z                 |
+-------------------+--------------------------------------+
```

7.3.2. Creación de red privada y router

Este paso es opcional, pues al crear la plantilla del clúster se puede crear una red privada automáticamente. Pero si se desea crear una red privada, se debe crear la red privada y su subred, al igual que un router para permitir la comunicación entre la red privada y la red externa, permitiendo la salida a Internet.

Para crear la red privada:

Cuadro 35. Comando para crear la red privada y su subred

```
# openstack network create red_privada
# openstack subnet create subnet_privada \
  --network red_privada \
  --subnet-range 192.168.2.0/24 \
  --gateway 192.168.2.1 \
  --allocation-pool start=192.168.2.2,end=192.168.2.254 \
  --ip-version 4
```

Donde:

- **red_privada**: nombre de la red privada.
- **subnet_privada**: nombre de la subred privada.

- –**subnet-range**: rango de direcciones IP de la subred privada.
- –**gateway**: dirección IP de la puerta de enlace de la subred privada.
- –**allocation-pool**: rango de direcciones IP que se asignarán a las máquinas virtuales en la subred privada.

El rango de direcciones IP de la subred privada está libre a elección del administrador.

Se debe obtener un mensaje de confirmación parecido a este:

Cuadro 36. Salida con detalles de la subred privada creada

```
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.2.2-192.168.2.254            |
| cidr              | 192.168.2.0/24                       |
| created_at        | 2025-08-10T21:55:10Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.2.1                          |
| host_routes       |                                      |
| id                | b1f4e8f3-5d3e-4c6a-8f4e-2e5f9c3b6a1d |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | subnet_privada                       |
| network_id        | 9c8f4e2b-3d4e-4a5b-9f4e-1e2f3c4b5a6d |
| project_id        | 431b4b0e9d594207a4ef1df4bbe8463a     |
| revision_number   | 1                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| updated_at        | 2025-08-10T21:55:10Z                 |
+-------------------+--------------------------------------+
```

Para crear el router:

Cuadro 37. Comando para crear el router privado

```
# openstack router create router_privado
```

Donde:

router_privado: nombre del router.

Se debería obtener una respuesta similar a la siguiente:

Cuadro 38. Salida con los detalles del router creado

```
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| admin_state_up          | UP                                   |
| availability_zone_hints |                                      |
| availability_zones      |                                      |
| created_at              | 2025-08-10T21:59:25Z                 |
| description             |                                      |
| distributed             | False                                |
| enable_ndp_proxy        | None                                 |
| external_gateway_info   | null                                 |
| flavor_id               | None                                 |
| ha                      | False                                |
| id                      | 02c52689-717c-42f2-a141-be7490f73dbb |
| name                    | router_privado                       |
| project_id              | 431b4b0e9d594207a4ef1df4bbe8463a     |
| revision_number         | 1                                    |
| routes                  |                                      |
| status                  | ACTIVE                               |
| tags                    |                                      |
| tenant_id               | 431b4b0e9d594207a4ef1df4bbe8463a     |
| updated_at              | 2025-08-10T21:59:25Z                 |
+-------------------------+--------------------------------------+
```

Luego se debe agregar la subred privada al router creado:

Cuadro 39. Comando para agregar la subred privada al router

```
# openstack router add subnet router_privado subnet_privada
```

Y se debe establecer la puerta de enlace externa del router.

Cuadro 40. Comando para establecer la puerta de enlace externa del router

```
# openstack router set router_privado --external-gateway public
```

Verificar que la red externa esté correctamente asignada al router:

Cuadro 41. Comando para verificar la red externa asignada al router

```
# openstack network show public -f value -c router:external
```

7.4. Creación de recursos necesarios

Para la creación de las plantillas de los clústeres Kubernetes, se deben crear los sabores y llaves SSH.

Los sabores o *flavors* en OpenStack definen las características de hardware virtual que tendrán las máquinas virtuales (VMs) que se crean en el entorno OpenStack. Estas características incluyen la cantidad de CPU, memoria RAM, y espacio en disco asignados a cada VM.

Se pueden crear diferentes sabores para los nodos maestros y nodos de trabajo dependiendo de la carga de trabajo de cada nodo, pero en este caso se creará un sabor para ambos tipos de nodos. En este caso se creará con las características mínimas recomendadas para un clúster de Kubernetes.

Cuadro 42. Comando para crear el sabor de los nodos

```
# openstack flavor create flavor-kubernetes\
  --vcpus 2 --ram 4096 --disk 20
```

Donde:

- <**flavor_kubernetes**>: nombre del sabor del nodo maestro.
- –**vcpus**: número de vCPUs asignadas al nodo maestro.
- –**ram**: cantidad de memoria RAM en MB asignada al nodo maestro.
- –**disk**: tamaño del disco en GB asignado al nodo maestro.

Cuadro 43. Salida con los detalles del sabor creado

```
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| description                | None                                 |
| disk                       | 20                                   |
| id                         | 5bfcf56c-6f1e-454a-8c9d-ccdc6ac1faa8 |
| name                       | flavor-kubernetes                    |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 4096                                 |
| rxtx_factor                | 1.0                                  |
| swap                       | 0                                    |
| vcpus                      | 2                                    |
+----------------------------+--------------------------------------+
```

Verificar que el sabor se haya creado correctamente:

Cuadro 44. Comando para verificar los sabores creados

```
# openstack flavor list
```

Debe mostrar un mensaje similar a este:

Cuadro 45. Salida con los sabores creados

```
+--------------------------------------+--------------------+------+------+
| ID                                   | Name               | RAM  | Disk |
+--------------------------------------+--------------------+------+------+
| 3e30f6ee-962f-4663-bee7-cbdb278779 | flavor-kubernetes | 4096 |  20  |
+--------------------------------------+--------------------+------+------+
```

Es posible conectarse a los nodos del clúster por medio de SSH para realizar tareas de administración y monitoreo, por lo que se deben crear las llaves SSH.

Cuadro 46. Comando para crear la llave SSH

```
# openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
```

Donde:

- –**public-key**: ruta donde se encuentra la llave pública que se utilizará para el acceso a los nodos (aquí se registran los usuarios que pueden acceden a los nodos).
- **mykey**: nombre de la llave que se creará en OpenStack.

Verificamos que la llave se haya creado correctamente:

Cuadro 47. Comando para verificar la llave SSH creadas

```
# openstack keypair list
```

Cuadro 48. Salida con las llaves SSH creadas

```
+---------+-------------------------------------------------+------+
| Name    | Fingerprint                                     | Type |
+---------+-------------------------------------------------+------+
| mykey   | 7a:15:94:01:2b:aa:94:01:5a:ba:97:f4:4d:b2:ee:66 | ssh  |
+---------+-------------------------------------------------+------+
```

7.5. Configuraciones finales

Según la documentación oficial de Magnum, estas eran todas las configuraciones necesarias para crear la plantilla y el clúster en Magnum. Sin embargo, en la práctica, se encontraron problemas para crear un clúster exitosamente, por lo que después de consultar guías no oficiales y foros, se encontraron varias opciones para solucionar estos problemas. Debido al temor de fallar una implementación y hacer que se caiga la infraestructura que estaba siendo usada también por otros compañeros, se decidió crear máquinas virtuales con diferentes sistemas operativos con la infraestructura de OpenStack y Magnum para implementar y comprobar cual de las diferentes soluciones era la más efectiva. Después de varias pruebas, se logró crear un clúster exitosamente en Magnum siguiendo los pasos descritos a continuación.

Antes de crear la plantilla del clúster y el clúster en Magnum, se deben realizar algunas configuraciones finales para asegurarse de que todo funcione correctamente. Primero debemos crear un archivo de configuración para Kind, Kind es una herramienta que permite ejecutar clústeres de Kubernetes localmente utilizando contenedores Docker como nodos del clúster. Es útil para pruebas y desarrollo local de aplicaciones en Kubernetes. Debemos crear un archivo de configuración, en este caso lo llamaremos *kind-config.yaml* con la siguiente configuración:

Cuadro 49. Configuración para Kind

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
```

Pueden haber más versiones de la API, pero esta es la que se utilizó en las pruebas y funcionó correctamente.

Ahora instalamos kubectl, la herramienta de línea de comandos para interactuar con clústeres de Kubernetes y Kind utilizando la herramienta arkade:

Cuadro 50. Comandos para instalar kubectl y helm con arkade

```
# curl -sLS https://get.arkade.dev | sh
# arkade get kubectl@v1.27.8 helm
# mv /root/.arkade/bin/kubectl /usr/local/bin/
# mv /root/.arkade/bin/helm /usr/local/bin/
# export KUBECONFIG=/root/config
# arkade get kind
```

Ya instalados kubectl y kind, debemos crear un clúster local de Kubernetes con Kind para probar que todo funcione correctamente:

Cuadro 51. Comando para crear el clúster local de Kubernetes con Kind

```
# kind create cluster --config kind-config.yaml
```

Si todo funciona correctamente, se debería obtener una respuesta similar a esta:

Cuadro 52. Salida de la creación del clúster local de Kubernetes

```
Creating cluster "kind" ...
 Ensuring node image (kindest/node:v1.27.3)
 Preparing nodes
 Writing configuration
 Starting control-plane
 Installing CNI
 Installing StorageClass
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next?  Check out https://kind.sigs.k8s.io/docs/user/
    quick-start/
```

La creación del cluster puede tardar entre 1 a 3 minutos.

Para verificar la ip y puerto del clúster local:

Cuadro 53. Comando para ver la información del clúster local

```
# kubectl cluster-info
```

Debe mostrar un mensaje similar a este:

Cuadro 54. Salida con la información del clúster local

```
Kubernetes control plane is running at https://<ip>:6443
CoreDNS is running at https://<ip>:6443/api/v1/namespaces/kube-system/
    services/kube-dns:dns/proxy
```

ip: es la ip del clúster local.

Con la configuración finalizada, se puede eliminar el clúster local de Kubernetes:

Cuadro 55. Comando para eliminar el clúster local de Kubernetes

```
# kind delete cluster
```

La creación del clúster local es solo para verificar que kubectl y kind estén funcionando correctamente antes de crear el clúster en Magnum. Ya con todo listo, se puede proceder a la creación de la plantilla del clúster y el clúster en Magnum.

CAPÍTULO 8

Creación de clústeres Kubernetes

8.1. Creación de plantilla de clúster

Para automatizar la creación de clústeres de Kubernetes, se debe crear una plantilla de clúster que defina las características del clúster. Hay dos maneras de crear una plantilla de clúster: utilizando la línea de comandos de OpenStack o utilizando el panel de Horizon.

8.1.1. Creación de plantilla de clúster con Horizon

Para poder ingresar a la interfaz gráfica de OpenStack Horizon, debemos contar con la contraseña del administrador o de un usuario con permisos para ingresar.

En este caso se utilizó el administrador. Para ello debemos obtener la contraseña del administrador, podemos correr el comando:

Cuadro 56. Comando para obtener la contraseña del administrador de OpenStack

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

Este comando mostrará la contraseña del usuario administrador de OpenStack:

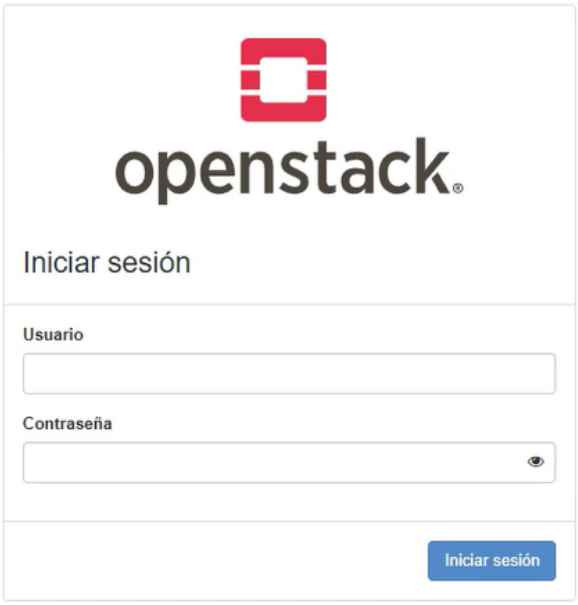
Cuadro 57. Salida con la contraseña del administrador de OpenStack

```
keystone_admin_password: 1d5awfs64ghj84g1svaete
```

(La contraseña puede variar dependiendo de la instalación). Con la contraseña del administrador, ingresar la dirección IP del controlador de OpenStack o la ip interna de kolla-ansible en un navegador web para dirigirse al panel de Horizon.

Al ingresar la dirección IP (192.168.74.5), se mostrará la página de inicio de sesión de Horizon (también se puede ingresar con la dirección VIP interna de kolla-ansible, en este caso *http://192.168.74.69*):

Figura 10. Página de inicio de sesión de Horizon



Nota. Esta es la página de inicio de sesión de la interfaz gráfica de OpenStack Horizon. Aquí es donde los usuarios pueden ingresar sus credenciales para acceder al panel de administración de OpenStack.

Ingresar las credenciales del administrador:

- **Username**: admin
- **Password**: 1d5awfs64ghj84g1svaete (contraseña obtenida previamente)

Después de iniciar sesión, se muestra el panel principal de Horizon, dirigirse a la sección de *Container Infra* y seleccionar *Cluster Template*, ya en la interfaz seleccionar *Create Cluster Template* (Aún si OpenStack esta en español, la sección está en inglés).

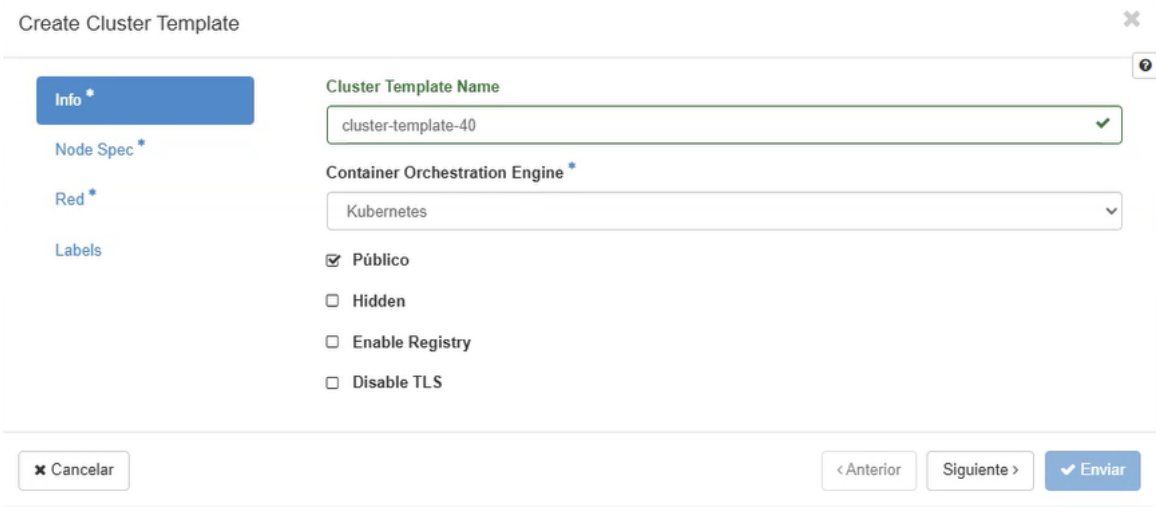
Figura 11. Sección de *Cluster Template* en Horizon

Nota. Sección de *Cluster Template* en Horizon, donde se pueden ver las plantillas de clústeres existentes y la opción para crear una nueva plantilla.

Nos mostrará un formulario para crear la plantilla del clúster. Llenar los campos de la sección de *Info*:

- **Name**: nombre de la plantilla del clúster (por ejemplo *cluster-template-40*).
- **Container Orchestration Engine**: seleccionar *Kubernetes*.
- **Público**: hace que la plantilla del clúster sea accesible para todos los usuarios (Marcar esta opción).

Figura 12. Formulario de creación de plantilla de clúster en Horizon (Info)



Nota. Sección *Info* del formulario de creación de plantilla de clúster en Horizon, donde se ingresan los detalles básicos de la plantilla.

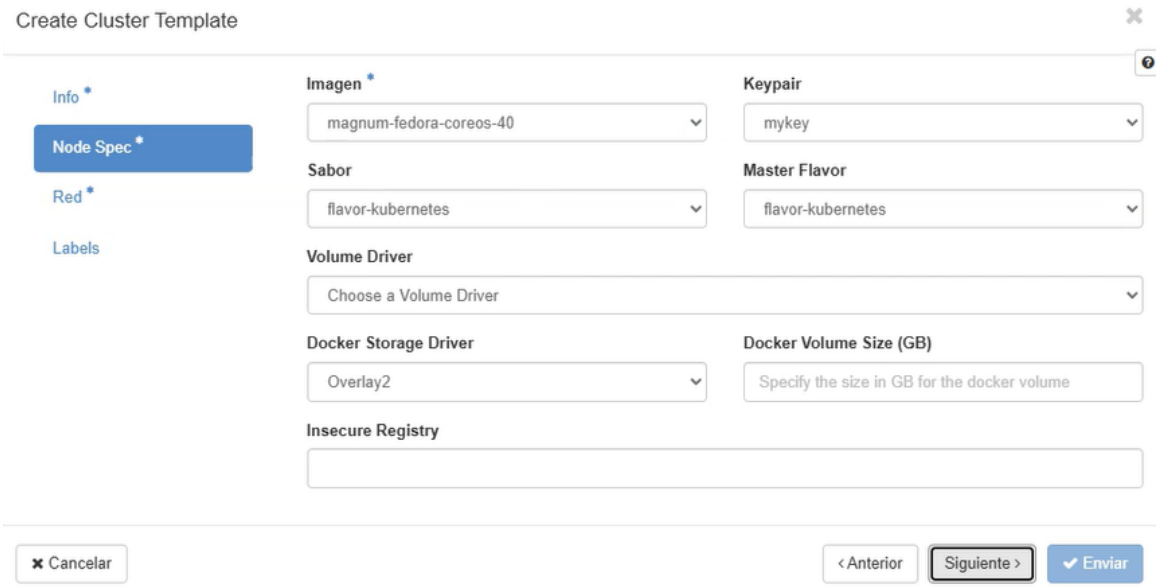
Presionamos siguiente para ir a la sección de *Node Specs* y llenamos los campos:

- **Imagen**: seleccionar la imagen de Fedora CoreOS creada previamente (*fedora-coreos-magnum*).

- **Keypair**: seleccionar la llave SSH creada previamente (*mykey*).
- **Sabor**: seleccionar el sabor creado previamente (*flavor_kubernetes*).
- **Master Flavor**: seleccionar el sabor creado previamente (*flavor_kubernetes*).
- **Docker Storage Driver**: driver de almacenamiento de Docker (Seleccionar *overlay2*).

Aunque no se utilice Docker para almacenamiento de contenedores, por foros y pruebas se ha comprobado que seleccionar *overlay2* en este campo ayuda a evitar problemas al crear el clúster.

Figura 13. Formulario de creación de plantilla de clúster en Horizon (Node Specs)

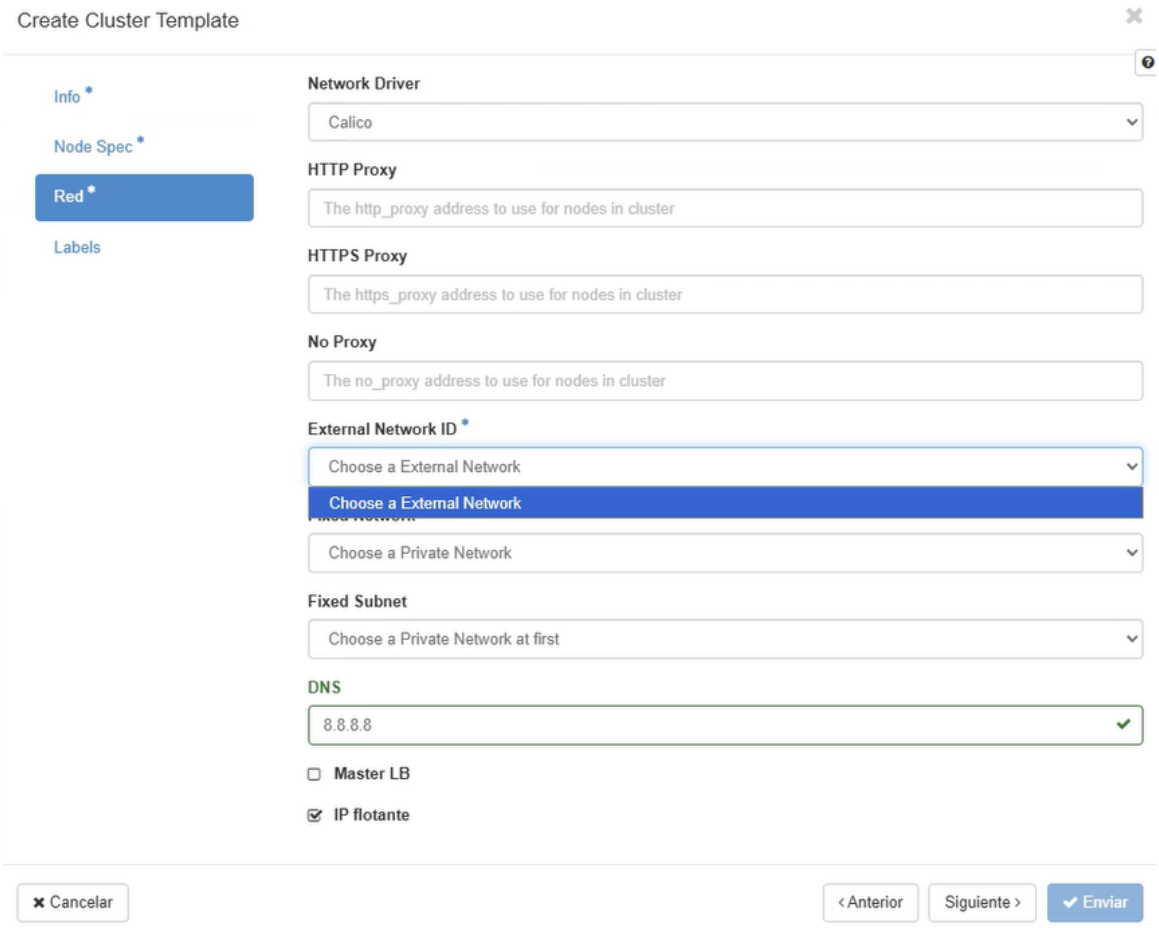


Nota. Sección *Node Specs* del formulario de creación de plantilla de clúster en Horizon, donde se ingresan los detalles específicos de los nodos.

Presionamos siguiente para ir a la sección de *Red* y llenamos los campos:

- **Network Driver**: seleccionar *Calico*.
- **External Network**: seleccionar la red externa creada previamente (*public*).
- **DNS Nameserver**: ingresar *8.8.8.8*.
- **IP flotante**: marcar esta opción.

Figura 14. Formulario de creación de plantilla de clúster en Horizon (Red)



Nota. Sección *Red* del formulario de creación de plantilla de clúster en Horizon, donde se ingresan los detalles de red para el clúster.

Dependiendo de la versión de OpenStack se puede encontrar con un problema al crear la plantilla del clúster desde Horizon, incluso al intentar en diferentes computadoras y sistemas operativos, la versión de OpenStack 2025 *bleeding edge* no permite visualizar ni seleccionar ninguna red externa al momento de crear la plantilla del clúster. Ya que este paso es obligatorio, no es posible crear la plantilla del clúster desde Horizon en esta versión, sin embargo, si se puede modificar una plantilla ya existente. Se comprobó que si se utiliza una versión anterior de OpenStack como la 2024.2 (Caracal) este error no se presenta y es posible crear la plantilla del clúster desde Horizon sin problemas.

Presionamos siguiente para ir a la sección de *Labels*. Después de varías pruebas, se comprobó la creación exitosa y el buen funcionamiento del clúster utilizando las siguientes etiquetas para la creación de la plantilla del clúster:

- **kube_tag**: v1.27.8-rancher2

- **container_runtime**: containerd
- **containerd_version**: 1.6.28
- **containerd_tarball_sha256**: f70736e52d61e5ad225f4fd21643b5ca1220013ab8b6c380434caeefb572da9b
- **cloud_provider_tag**: v1.27.3

Ingresarlas de la siguiente manera:

Cuadro 58. Etiquetas para la creación de la plantilla del clúster

```
kube_tag=v1.27.8-rancher2,container_runtime=containerd,
    containerd_version=1.6.28,containerd_tarball_sha256=
    f70736e52d61e5ad225f4fd21643b5ca1220013ab8b6c380434caeefb572da9b,
    cloud_provider_tag=v1.27.3
```

Es importante que no tengan espacios entre las comas y los nombres de las etiquetas.

Figura 15. Formulario de creación de plantilla de clúster en Horizon (Labels)

Nota. Sección *Labels* del formulario de creación de plantilla de clúster en Horizon, donde se ingresan las etiquetas necesarias para la configuración del clúster.

Por último, presionamos Enviar para crear la plantilla del clúster.

8.1.2. Creación de plantilla de clúster con CLI

Utilizando la línea de comandos de OpenStack, se puede crear una plantilla de clúster:

Cuadro 59. Comando para crear la plantilla del clúster

```
# openstack coe cluster template create cluster-template-40 \
  --image fedora-coreos-magnum  \
  --external-network public \
  --dns-nameserver 8.8.8.8 \
  --master-flavor flavor-kubernetes \
  --flavor flavor-kubernetes \
  --network-driver calico \
  --coe kubernetes \
  --labels kube_tag=v1.27.8-rancher2,container_runtime=containerd,
      containerd_version=1.6.28,containerd_tarball_sha256=
      f70736e52d61e5ad225f4fd21643b5ca1220013ab8b6c380434caeefb572da9b,
      cloud_provider_tag=v1.27.3
```

Donde:

- **cluster-template-40**: nombre del template a crear.
- **image**: nombre de la imagen a utilizar para los nodos.
- **external-network**: nombre de la red externa a utilizar (red creada previamente).
- **dns-nameserver**: nombre del servidor DNS a utilizar (por lo general 8.8.8.8).
- **flavor**: nombre del sabor a utilizar para los nodos.
- **master-flavor**: nombre del sabor a utilizar para el nodo maestro.
- **keypair**: nombre de la llave SSH a utilizar para el acceso a los nodos.
- **coe**: tipo de orquestador de contenedores a utilizar (en este caso Kubernetes).
- **labels**: etiquetas adicionales para la configuración del clúster.

Debería mostrar un mensaje similar a este:

Cuadro 60. Salida con los detalles de la plantilla del clúster creada

```
Request to create cluster template cluster-template-40 accepted
+-----------------------+----------------------------------------+
| Field                 | Value                                  |
+-----------------------+----------------------------------------+
| insecure_registry     | -                                      |
| labels                | {'kube_tag': 'v1.27.8-rancher2',       |
|                       | 'container_runtime': 'containerd',     |
|                       | 'containerd_version': '1.6.28',        |
|                       | 'containerd_tarball_sha256': 'f70...', |
|                       | 'cloud_provider_tag': 'v1.27.3'}       |
| updated_at            | -                                      |
| floating_ip_enabled   | True                                   |
| fixed_subnet          | -                                      |
| master_flavor_id      | flavor-kubernetes                      |
| uuid                  | 23c922fc-8cdc-48fa-a99a-3e3beaf7e351   |
| no_proxy              | -                                      |
| https_proxy           | -                                      |
| tls_disabled          | False                                  |
| keypair_id            |                                        |
| public                | False                                  |
| http_proxy            | -                                      |
| docker_volume_size    | -                                      |
| server_type           | vm                                     |
| external_network_id   | public                                 |
| cluster_distro        | fedora-coreos                          |
| image_id              | fedora-coreos-magnum                   |
| volume_driver         | -                                      |
| registry_enabled      | False                                  |
| docker_storage_driver | overlay2                               |
| apiserver_port        | -                                      |
| name                  | cluster-template-40                    |
| created_at            | 2025-10-17T21:37:28+00:00              |
| network_driver        | calico                                 |
| fixed_network         | -                                      |
| coe                   | kubernetes                             |
| flavor_id             | flavor-kubernetes                      |
| master_lb_enabled     | False                                  |
| dns_nameserver        | 8.8.8.8                                |
| hidden                | False                                  |
| tags                  | -                                      |
+-----------------------+----------------------------------------+
```

8.2. Verificación de recursos de Placement

Antes de crear el clúster revisar la disponibilidad de recursos que puede asignar el componente Placement. Primero se debe instalar el cliente de Placement:

Cuadro 61. Comando para instalar el cliente de Placement

```
# pip install osc-placement
```

Luego verificar los provedores de recursos creados:

Cuadro 62. Comando para listar los proveedores de recursos

```
# openstack resource provider list
```

Se mostrará un mensaje similar a este:

Cuadro 63. Salida con los proveedores de recursos

```
+--------------------------------------+--------+------------+...
| uuid                                 | name   | generation |...
+--------------------------------------+--------+------------+...
| ab80d639-5aa7-41cc-bb88-110d49980811 | deploy |         98 |...
+--------------------------------------+--------+------------+...
```

Con el *UUID* del proveedor de recursos, se puede verificar los recursos disponibles:

Cuadro 64. Comando para listar los recursos del proveedor

```
# openstack resource provider inventory list <UUID_proveedor>
```

Donde:

UUID_proveedor: es el *UUID* del proveedor de recursos obtenido en el paso anterior.

Se debería obtener un mensaje similar a este:

Cuadro 65. Salida con los recursos del proveedor

```
+----------------+...+----------+----------+...+--------+-------+
| resource_class |...| max_unit | reserved |...|  total |  used |
+----------------+...+----------+----------+...+--------+-------+
| VCPU           |...|      112 |        0 |...|    112 |    11 |
| MEMORY_MB      |...|   257410 |     8192 |...| 257410 | 16512 |
| DISK_GB        |...|     1757 |        0 |...|   1757 |   121 |
+----------------+...+----------+----------+...+--------+-------+
```

Lo importante es verificar que haya suficientes recursos disponibles para crear el clúster de Kubernetes, si no hay suficientes recursos disponibles, se debe ajustar la configuración de los nodos o agregar más recursos al proveedor.

Cuadro 66. Comando para asignar recursos al proveedor

```
# openstack resource provider inventory set <RP_UUID> \
  --resource VCPU=<cantidad_vcpu> \
  --resource MEMORY_MB=<cantidad_memoria> \
  --resource DISK_GB=<cantidad_disco>
```

Donde:

- <**RP_UUID**>: es el *UUID* del proveedor de recursos.
- <**cantidad_vcpu**>: es la cantidad de vCPUs a asignar.
- <**cantidad_memoria**>: es la cantidad de memoria en MB a asignar.
- <**cantidad_disco**>: es la cantidad de disco en GB a asignar.

Los recursos mínimos de cada nodo para crear un clúster de Kubernetes son:

- **vCPU**: 2
- **MEMORY_MB**: 4096
- **DISK_GB**: 20

Asignarle recursos menores a estos valores causa que los nodos del clúster se encuentren en estado *unhealthy*.

8.3. Creación de clúster

Así como en la creación de la plantilla del clúster, se puede crear el clúster de Kubernetes utilizando la línea de comandos de OpenStack o utilizando el panel de Horizon.

8.3.1. Creación de clúster con Horizon

Para crear el clúster de Kubernetes desde Horizon, dirigirse a la sección de *Container Infra* y seleccionar *Clusters*, ya en la interfaz seleccionar *Create Cluster*.

Figura 16. Sección de *Clusters* en Horizon

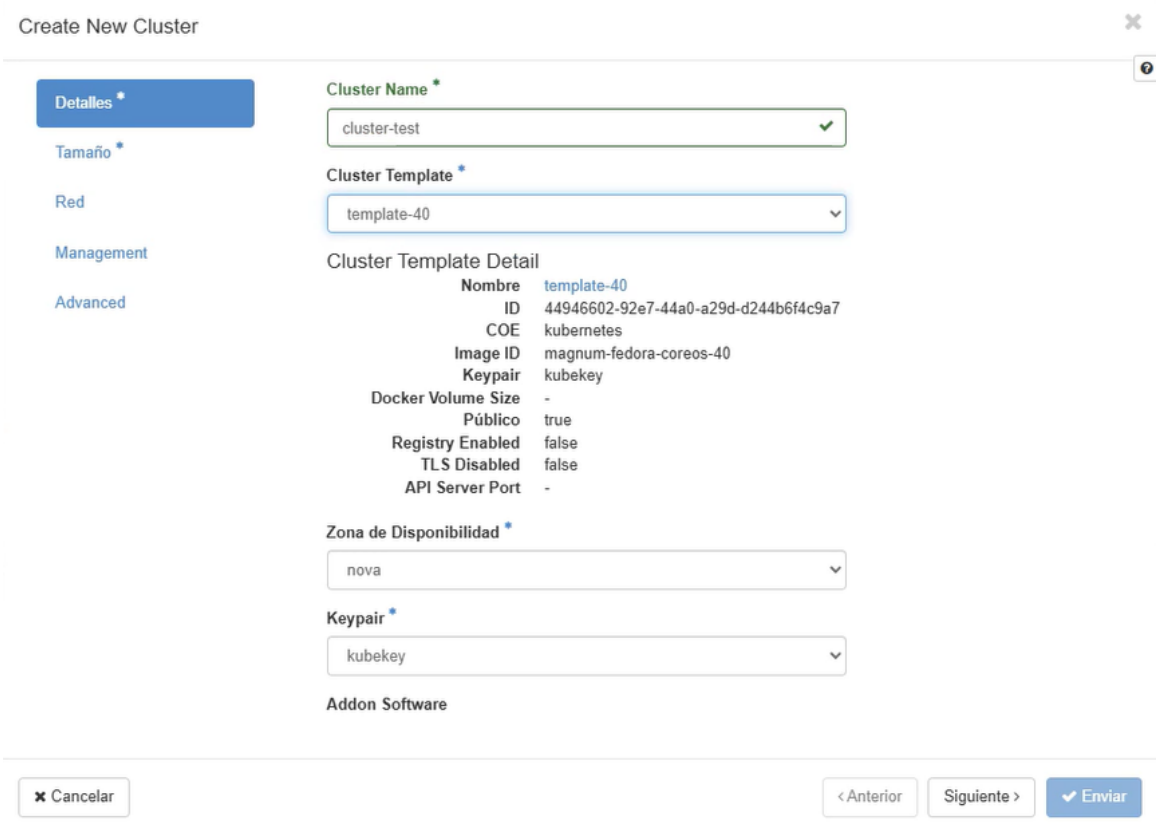
Nota. Sección de *Clusters* en Horizon, donde se pueden ver los clústeres existentes y la opción para crear un nuevo clúster.

Se mostrará un formulario para crear el clúster. Llenar los campos de la sección de *Info*:

- **Name**: nombre del clúster (por ejemplo *cluster-test*).
- **Cluster Template**: seleccionar la plantilla de clúster creada previamente (*cluster-template-40*).

Al seleccionar la plantilla del clúster, el campo de keypair se llenará automáticamente.

Figura 17. Formulario de creación de clúster en Horizon (Info)



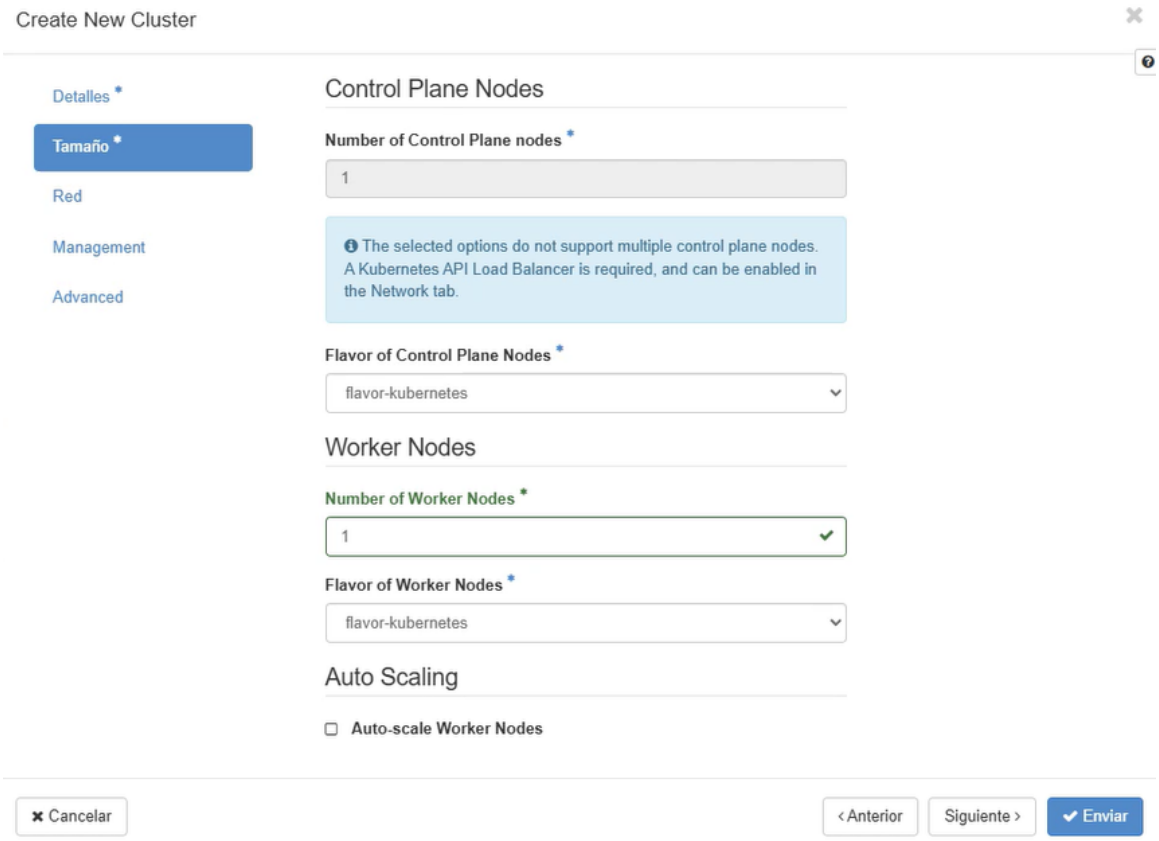
Nota. Sección de detalles de formulario de creación de clúster en Horizon, donde se deben ingresar los detalles del clúster.

Presionamos siguiente para ir a la sección de *Tamaño* y solamente llenamos:

- **Node Count**: número de nodos de trabajo a crear (en este caso 1).

(Los demás campos ya están llenados automáticamente con la información de la plantilla del clúster).

Figura 18. Formulario de creación de clúster en Horizon (Tamaño)



Nota. Sección de tamaño del formulario de creación de clúster en Horizon, donde se debe especificar el número de nodos y sabores para el clúster.

Por último, presionamos Enviar para crear el clúster de Kubernetes.

8.3.2. Creación de clúster con CLI

Con la plantilla de clúster creada y recursos revisados, se puede proceder a la creación del clúster de Kubernetes. Para la comprobación se creará un nodo maestro y un nodo de trabajo. Se puede ajustar el número de nodos según las necesidades.

Cuadro 67. Comando para crear el clúster de Kubernetes

```
# openstack coe cluster create cluster-test \
  --cluster-template cluster-template-40 \
  --master-count 1 \
  --node-count 1 \
  --keypair mykey
```

Donde:

- **cluster-test**: nombre del clúster a crear.
- **cluster-template**: nombre de la plantilla de clúster a utilizar.
- **master-count**: número de nodos maestros a crear.
- **node-count**: número de nodos de trabajo a crear.
- **keypair**: nombre de la llave SSH a utilizar para el acceso a los nodos.

Si muestra este error al intentar crear el clúster:

Cuadro 68. Salida con error al crear el clúster

```
ClusterTemplate cluster-template-40 could not be found (HTTP 404) ...
```

Verificar que la plantilla de clúster se haya creado correctamente:

Cuadro 69. Comando para listar las plantillas de clúster

```
# openstack coe cluster template list
```

Deberá aparece un mensaje similar a este:

Cuadro 70. Salida con las plantillas de clúster disponibles

```
+--------------------------------------+---------------------+------+
| uuid                                 | name                | tags |
+--------------------------------------+---------------------+------+
| 23c922fc-8cdc-48fa-a99a-3e3beaf7e351 | cluster-template-40 | None |
+--------------------------------------+---------------------+------+
```

Si la plantilla de clúster no aparece en la lista, deberá crearla nuevamente. Pero si la plantilla aparece, al crear nuevamente el clúster, puede ser por la versión de OpenStack, en la versión más actualizada (master) de OpenStack, este es un problema común. Si no se ha arreglado en la versión que se esté utilizando, se puede utilizar al *UUID* de la plantilla en lugar del nombre:

Cuadro 71. Comando para crear el clúster de Kubernetes utilizando el UUID de la plantilla

```
# openstack coe cluster create k8s-cluster \
  --cluster-template 23c922fc-8cdc-48fa-a99a-3e3beaf7e351 \
  --master-count 1 \
  --node-count 2 \
  --keypair mykey
```

Se deberá obtener un mensaje similar a este:

Cuadro 72. Salida con los detalles del clúster creado

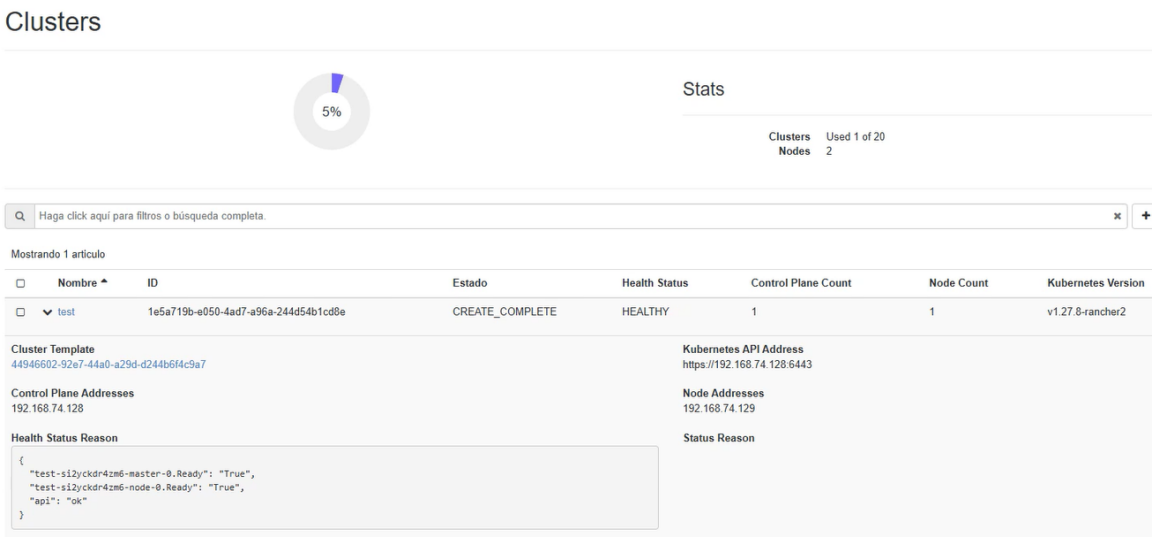
```
Request to create cluster 4ff1762d-d4a9-4a42-99b1-de7210904a4b accepted
```

8.4. Verificación del estado del clúster

8.4.1. Verificación del estado del clúster por Horizon

En la sección de *Container Infra* seleccionar *Clusters*, se mostrará una lista de los clústeres creados, también mostrará cuantos nodos están activos. Seleccionar el clúster creado (*cluster-test*) para ver los detalles del clúster.

Figura 19. Detalles del clúster en Horizon



Nota. Detalles del clúster en Horizon, donde se pueden ver los estados de los nodos y otra información relevante del clúster.

8.4.2. Verificación del estado del clúster por CLI

Para poder verificar el estado del clúster creado, se puede utilizar el siguiente comando:

Cuadro 73. Comando para mostrar los detalles del clúster

```
# openstack coe cluster show cluster-test
```

Donde:

cluster-test: Nombre del clúster creado.

Debería mostrar un mensaje similar a este:

Cuadro 74. Salida con los detalles del clúster

```
+----------------------+------------------------------------------+
| Field                | Value                                    |
+----------------------+------------------------------------------+
| status               | CREATE_IN_PROGRESS                       |
| health_status        | None                                     |
| cluster_template_id  | 44946602-92e7-44a0-a29d-d244b6f4c9a7     |
| node_addresses       | []                                       |
| uuid                 | c160c0de-f652-488e-8492-198775efe2c4     |
| stack_id             | b50bcef6-961e-4512-bf51-0be8a3b253fd     |
| status_reason        | None                                     |
| created_at           | 2025-10-17T22:39:19+00:00                |
| updated_at           | 2025-10-17T22:39:28+00:00                |
| coe_version          | None                                     |
| labels               | {'kube_tag': 'v1.27.8-rancher2',         |
|                      | 'container_runtime': 'containerd',       |
|                      | 'containerd_version': '1.6.28',          |
|                      | 'containerd', 'containerd_version':      |
|                      | '1.6.28', 'containerd_tarball_sha256':   |
|                      | 'f70736e52d61e5ad225f4fd21643b5ca1220013' |
|                      | 'ab8b6c380434caeefb572da9b',             |
|                      | 'cloud_provider_tag': 'v1.27.3',         |
|                      | 'availability_zone': 'nova',             |
|                      | 'auto_scaling_enabled': 'False',         |
|                      | 'auto_healing_enabled': 'False',         |
|                      | 'master_lb_floating_ip_enabled': 'False'} |
| labels_overridden    | {}                                       |
| labels_skipped       | {}                                       |
| labels_added         | {'availability_zone': 'nova',            |
|                      | 'auto_scaling_enabled': 'False',         |
|                      | 'auto_healing_enabled': 'False',         |
|                      | 'master_lb_floating_ip_enabled': 'False'} |
| fixed_network        | None                                     |
| fixed_subnet         | None                                     |
| floating_ip_enabled  | True                                     |
| faults               |                                          |
| keypair              | kubekey                                  |
| api_address          | None                                     |
| master_addresses     | []                                       |
| master_lb_enabled    | False                                    |
| create_timeout       | 60                                       |
| node_count           | 1                                        |
| discovery_url        | https://discovery.etcd.io/a5861d67f05048 |
|                      | 58cee2044c1d6                            |
|                      | 621af                                    |
| docker_volume_size   | None                                     |
| master_count         | 1                                        |
| container_version    | None                                     |
| name                 | test                                     |
| master_flavor_id     | flavor-kubernetes                        |
| flavor_id            | flavor-kubernetes                        |
| health_status_reason | {}                                       |
```

```
| project_id           | 5260c7767b584c58a1da728373b2d8f3          |
+----------------------+-------------------------------------------+
```

El proceso de creación del clúster con esta cantidad de nodos puede tardar 8 a 12 minutos, dependiendo de la cantidad de nodos y los recursos disponibles en el entorno OpenStack puede variar el tiempo.

Para mayor detalle del estado del clúster, se puede revisar que eventos de la pila de Heat se estan creando y si llegará a fallar que recurso fue. Debemos descargar el cliente de Heat:

Cuadro 75. Comando para instalar el cliente de Heat

```
# pip install python-heatclient
```

Luego verificar los eventos de la pila de Heat creada para esto se debe obtener el *stack_ id* del clúster con el comando anterior y luego utilizar el siguiente comando:

Cuadro 76. Comando para verificar eventos de la pila de Heat

```
# openstack stack event list <stack_id>
```

Se mostrará un mensaje similar a este:

Cuadro 77. Salida con los eventos de la pila de Heat

```
2025-10-17 22:39:27Z [test-44mknmkulfty]: CREATE_IN_PROGRESS  Stack
    CREATE started
2025-10-17 22:39:28Z [test-44mknmkulfty.master_nodes_server_group]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:28Z [test-44mknmkulfty.master_nodes_server_group]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:29Z [test-44mknmkulfty.secgroup_kube_master]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:30Z [test-44mknmkulfty.worker_nodes_server_group]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:30Z [test-44mknmkulfty.worker_nodes_server_group]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:31Z [test-44mknmkulfty.network]: CREATE_IN_PROGRESS
    state changed
2025-10-17 22:39:31Z [test-44mknmkulfty.secgroup_kube_master]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:31Z [test-44mknmkulfty.secgroup_kube_minion]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:32Z [test-44mknmkulfty.secgroup_kube_minion]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:32Z [test-44mknmkulfty.secgroup_rule_udp_kube_minion]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:32Z [test-44mknmkulfty.
    secgroup_rule_udp_kube_minion_pods_cidr]: CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:39:32Z [test-44mknmkulfty.
    secgroup_rule_tcp_kube_minion_pods_cidr]: CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:39:32Z [test-44mknmkulfty.secgroup_rule_tcp_kube_minion]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:39:33Z [test-44mknmkulfty.
    secgroup_rule_udp_kube_minion_pods_cidr]: CREATE_COMPLETE  state
    changed
2025-10-17 22:39:33Z [test-44mknmkulfty.
    secgroup_rule_tcp_kube_minion_pods_cidr]: CREATE_COMPLETE  state
    changed
2025-10-17 22:39:33Z [test-44mknmkulfty.secgroup_rule_udp_kube_minion]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:33Z [test-44mknmkulfty.secgroup_rule_tcp_kube_minion]:
    CREATE_COMPLETE  state changed
2025-10-17 22:39:34Z [test-44mknmkulfty.network]: CREATE_COMPLETE  state
     changed
2025-10-17 22:39:35Z [test-44mknmkulfty.api_lb]: CREATE_IN_PROGRESS
    state changed
2025-10-17 22:39:35Z [test-44mknmkulfty.etcd_lb]: CREATE_IN_PROGRESS
    state changed
2025-10-17 22:39:36Z [test-44mknmkulfty.api_lb]: CREATE_COMPLETE  state
    changed
2025-10-17 22:39:36Z [test-44mknmkulfty.etcd_lb]: CREATE_COMPLETE  state
     changed
2025-10-17 22:39:37Z [test-44mknmkulfty.kube_masters]:
    CREATE_IN_PROGRESS  state changed
```

Aqui se puede observar el progreso de la creación del clúster y si algún recurso falla,

se puede identificar cuál fue el recurso que causó el fallo. Si todo se crea correctamente, el estado final del clúster debería verse así:

Cuadro 78. Salida con los eventos de la pila de Heat al finalizar la creación del clúster

```
2025-10-17 22:56:34Z [test-4ucrap3zysl4]: CREATE_IN_PROGRESS  Stack CREATE
    started
2025-10-17 22:56:35Z [test-4ucrap3zysl4.secgroup_kube_master]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:56:36Z [test-4ucrap3zysl4.worker_nodes_server_group]:
    CREATE_IN_PROGRESS  statechanged
2025-10-17 22:56:36Z [test-4ucrap3zysl4.worker_nodes_server_group]:
    CREATE_COMPLETE  statechanged
2025-10-17 22:56:37Z [test-4ucrap3zysl4.network]: CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:56:37Z [test-4ucrap3zysl4.secgroup_kube_master]:
    CREATE_COMPLETE  state changed
2025-10-17 22:56:37Z [test-4ucrap3zysl4.secgroup_kube_minion]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.master_nodes_server_group]:
    CREATE_IN_PROGRESS  statechanged
2025-10-17 22:56:38Z [test-4ucrap3zysl4.secgroup_kube_minion]:
    CREATE_COMPLETE  state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.
    secgroup_rule_tcp_kube_minion_pods_cidr]:CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.secgroup_rule_udp_kube_minion]:
    CREATE_IN_PROGRESS state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.master_nodes_server_group]:
    CREATE_COMPLETE  statechanged
2025-10-17 22:56:38Z [test-4ucrap3zysl4.secgroup_rule_tcp_kube_minion]:
    CREATE_IN_PROGRESS state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.
    secgroup_rule_udp_kube_minion_pods_cidr]:CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.
    secgroup_rule_tcp_kube_minion_pods_cidr]:CREATE_COMPLETE  state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.secgroup_rule_udp_kube_minion]:
    CREATE_COMPLETE  statechanged
2025-10-17 22:56:38Z [test-4ucrap3zysl4.
    secgroup_rule_udp_kube_minion_pods_cidr]:CREATE_COMPLETE  state changed
2025-10-17 22:56:38Z [test-4ucrap3zysl4.secgroup_rule_tcp_kube_minion]:
    CREATE_COMPLETE  statechanged
2025-10-17 22:56:41Z [test-4ucrap3zysl4.network]: CREATE_COMPLETE  state
    changed
2025-10-17 22:56:42Z [test-4ucrap3zysl4.api_lb]: CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:56:42Z [test-4ucrap3zysl4.etcd_lb]: CREATE_IN_PROGRESS  state
    changed
2025-10-17 22:56:43Z [test-4ucrap3zysl4.etcd_lb]: CREATE_COMPLETE  state
    changed
2025-10-17 22:56:43Z [test-4ucrap3zysl4.api_lb]: CREATE_COMPLETE  state
    changed
2025-10-17 22:56:44Z [test-4ucrap3zysl4.kube_masters]: CREATE_IN_PROGRESS
    state changed
2025-10-17 23:01:18Z [test-4ucrap3zysl4.kube_masters]: CREATE_COMPLETE
    state changed
2025-10-17 23:01:19Z [test-4ucrap3zysl4.api_address_lb_switch]:
```

```
    CREATE_IN_PROGRESS  statechanged
2025-10-17 23:01:19Z [test-4ucrap3zysl4.etcd_address_lb_switch]:
    CREATE_IN_PROGRESS  statechanged
2025-10-17 23:01:19Z [test-4ucrap3zysl4.kube_cluster_config]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 23:01:20Z [test-4ucrap3zysl4.api_address_lb_switch]:
    CREATE_COMPLETE  state changed
2025-10-17 23:01:20Z [test-4ucrap3zysl4.etcd_address_lb_switch]:
    CREATE_COMPLETE  state changed
2025-10-17 23:01:20Z [test-4ucrap3zysl4.kube_cluster_config]:
    CREATE_COMPLETE  state changed
2025-10-17 23:01:20Z [test-4ucrap3zysl4.kube_cluster_deploy]:
    CREATE_IN_PROGRESS  state changed
2025-10-17 23:01:21Z [test-4ucrap3zysl4.api_address_floating_switch]:
    CREATE_IN_PROGRESS  statechanged
2025-10-17 23:01:21Z [test-4ucrap3zysl4.kube_minions]: CREATE_IN_PROGRESS
    state changed
2025-10-17 23:01:21Z [test-4ucrap3zysl4.api_address_floating_switch]:
    CREATE_COMPLETE  statechanged
2025-10-17 23:01:43Z [test-4ucrap3zysl4.kube_cluster_deploy]:
    SIGNAL_IN_PROGRESS  Signal:deployment 57667e14-2dd8-49d4-8c6a-55
    bbe51fce1d succeeded
2025-10-17 23:01:44Z [test-4ucrap3zysl4.kube_cluster_deploy]:
    CREATE_COMPLETE  state changed
2025-10-17 23:04:32Z [test-4ucrap3zysl4.kube_minions]: CREATE_COMPLETE
    state changed
2025-10-17 23:04:32Z [test-4ucrap3zysl4]: CREATE_COMPLETE  Stack CREATE
    completed successfully
```

Revisar por último la salud del clúster:

Cuadro 79. Comando para revisar la salud del clúster

```
    # openstack coe cluster show cluster-test --format value -c
        health_status
```

Debe mostrar:

Cuadro 80. Salida con el estado de salud del clúster

```
    HEALTHY
```

Para añadir las credenciales del clúster al archivo *KUBECONFIG* local, se puede utilizar la siguiente serie de comandos:

Cuadro 81. Comandos para descargar las credenciales del clúster

```
    # mkdir -p /root/openstack/kubeconfigs
    # cd /root/openstack/kubeconfigs
    # openstack coe cluster config cluster-test
    # export KUBECONFIG=/root/openstack/kubeconfigs/config
```

Creamos un directorio para guardar las credenciales del clúster, descargamos las credenciales y luego exportamos la variable de entorno *KUBECONFIG* para que apunte al archivo de credenciales descargado. Con las credenciales del clúster ya podemos utilizar la herramienta *kubectl* para administrar el clúster de Kubernetes.

Ahora podemos obtener los componentes del controlador de clúster Kubernetes y verificar si están funcionando correctamente:

Cuadro 82. Comando para obtener los pods del clúster Kubernetes

```
# kubectl -n kube-system get pods
```

Debería mostrar un mensaje similar a este:

Cuadro 83. Salida con los pods del clúster Kubernetes

```
NAME                                            READY   STATUS
    RESTARTS   AGE
calico-kube-controllers-5b8c748c67-sq2fk        1/1     Running
    0          4h52m
calico-node-5jm95                               0/1     Running
    0          4h52m
calico-node-cj7l9                               0/1     Running
    0          4h49m
coredns-5ff97bd88-54ffj                         1/1     Running
    0          4h52m
coredns-5ff97bd88-mp8vn                         1/1     Running
    0          4h52m
dashboard-metrics-scraper-84fdcc4457-t9lbc      1/1     Running
    0          4h52m
k8s-keystone-auth-2scp9                         1/1     Running
    0          4h52m
kube-dns-autoscaler-6c8ccf9bb4-z2qzf            1/1     Running
    0          4h52m
kubernetes-dashboard-6c54687645-5b6c9           1/1     Running
    0          4h52m
magnum-metrics-server-8457b5867c-ldm74          1/1     Running
    0          4h52m
npd-bgzfk                                       1/1     Running
    0          4h48m
openstack-cloud-controller-manager-f5h7s        1/1     Running
    0          4h52m
```

8.5. Acceso al clúster Kubernetes

En ocasiones los errores para crear el clúster se deben a problemas en los nodos, por lo que poder acceder a los nodos para revisar los *logs* es necesario. Para poder acceder a los nodos del clúster, los nodos deben tener una ip pública asignada. Por defecto la opción de ip pública está habilitada en la plantilla del clúster, por lo que los nodos deberían tener una ip pública asignada. Para obtener las ip públicas de los nodos maestros, se puede utilizar el

siguiente comando:

Cuadro 84. Comando para obtener la ip pública del nodo maestro

```
# openstack coe cluster show cluster-test --format value -c
    master_addresses
```

Debería de mostrar la ip pública del nodo maestro de esta manera:

Cuadro 85. Salida con la ip pública del nodo maestro

```
['192.168.74.120']
```

Para obtener las ip públicas de los nodos de trabajo, se hace de la siguiente manera:

Cuadro 86. Comando para obtener la ip pública del nodo de trabajo

```
# openstack coe cluster show cluster-test --format value -c
    node_addresses
```

Deberia de mostrar las ip públicas de los nodos de trabajo de esta manera:

Cuadro 87. Salida con la ip pública del nodo de trabajo

```
['192.168.74.121']
```

Si hay varios nodos de trabajo, se mostrarán todas las ip públicas en una lista así:

Cuadro 88. Salida con las ip públicas de varios nodos de trabajo

```
['192.168.74.121', '192.168.74.122']
```

Ya con la ip pública, se puede acceder a los nodos por medio de SSH:

Cuadro 89. Comando para acceder al nodo por SSH

```
# ssh core@<ip_publica_nodo>
```

ip_publica_nodo: es la ip pública del nodo maestro o del nodo de trabajo al que se desea acceder.

Al acceder por primera vez, se pedirá confirmar la huella digital del servidor, se debe escribir *yes* y presionar *Enter*. Se mostrará un mensaje similar a este:

Cuadro 90. Terminal del nodo por SSH

```
The authenticity of host 192.168.1.10 (192.168.1.10) cannot be
    established.
ED25519 key fingerprint is SHA256:YmzbU43RDf4rZtBEXCpEFoWdof1WAMrGmtqF/
    uyEMvI.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 192.168.1.10 (ED25519) to the list of known
    hosts.
Fedora CoreOS 40.20240616.3.0
Tracker: https://github.com/coreos/fedora-coreos-tracker
Discuss: https://discussion.fedoraproject.org/tag/coreos

[core@test-4ucrap3zysl4-node-0 ~]$
```

Ya en la terminal del nodo, se pueden revisar los logs del sistema y de los servicios de Kubernetes para identificar posibles problemas.

Antes debemos ingresar como usuario root:

Cuadro 91. Comando para entrar en modo super usuario en el nodo

```
$ sudo su
```

Cuadro 92. Comando en el *root* del nodo

```
# sudo -i
```

Para revisar fallos en el nodo, se puede utilizar el siguiente comando:

Cuadro 93. Comando para revisar servicios fallidos en el nodo

```
# systemctl --failed
```

Debería mostrar un mensaje similar a este:

Cuadro 94. Salida de servicios fallidos en el nodo

```
UNIT                            LOAD   ACTIVE SUB    DESCRIPTION
0 loaded units listed.
```

Se puede revisar que todos los contenedores estén corriendo correctamente con el siguiente comando:

Cuadro 95. Comando para revisar los contenedores en el nodo

```
# podman ps -a
```

Debería mostrar un mensaje similar a este:

Cuadro 96. Estado de los contenedores en el nodo

```
CONTAINER ID  IMAGE                                COMMAND
    CREATED       STATUS       NAMES
4fc8ae691333  docker.io/openstackmagnum/heat-...  /usr/bin/start-he...
    1 hours ago  Up 1 hours  heat-container-agent
faafa3d56252  docker.io/rancher/hyperkube...      kubelet --v=3 --p...
    1 hours ago  Up 1 hours  kubelet
f04fde93afb8  docker.io/rancher/hyperkube...      kube-proxy --v=3 ...
    1 hours ago  Up 1 hours  kube-proxy
123456789abc  docker.io/rancher/hyperkube...      kube-scheduler --v=3
    1 hours ago  Up 1 hours  kube-scheduler
987654321def  docker.io/rancher/hyperkube...      kube-controller-m...
    1 hours ago  Up 1 hours  kube-controller-manager
456789abcdef  docker.io/rancher/hyperkube...      kube-api-server --v=3
    1 hours ago  Up 1 hours  kube-apiserver
0123456789ab  quay.io/rancher/hyperkube...        /usr/local/bin/et...
    1 hours ago  Up 1 hours  etcd
```

Si todo está funcionando correctamente, el clúster está operativo y listo para usarse.

8.6. Prueba de despliegue de nginx en el clúster

Para comprobar que el clúster de Kubernetes está funcionando correctamente, se puede desplegar una aplicación sencilla como Nginx.

Cuadro 97. Comando para desplegar Nginx en el clúster

```
# kubectl run nginx --image=nginx
```

Se debería mostrar un mensaje similar a este:

Cuadro 98. Salida del comando para desplegar Nginx en el clúster

```
pod/nginx created
```

Para verificar que el pod de Nginx se haya creado correctamente, se puede utilizar el siguiente comando:

Cuadro 99. Comando para obtener los pods creados en clúster

```
# kubectl get pods
```

Debería mostrar un mensaje similar a este:

Cuadro 100. Salida con los pods creados en el clúster

```
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          30s
```

CAPÍTULO 9

Monitoreo y gestión de clústeres Kubernetes

El monitoreo es una parte crucial en la gestión de clústeres Kubernetes para asegurar su rendimiento y disponibilidad. En este capítulo, se explicará cómo instalar y configurar Prometheus y Grafana para el monitoreo de los clústeres Kubernetes creados con OpenStack Magnum.

9.1. Instalación y configuración de Prometheus y Grafana

Para instalar y configurar Prometheus y Grafana en nuestra infraestructura de OpenStack actual debemos modificar el archivo de *globals.yaml* en el directorio */etc/kolla/* para habiliar la instalación de estos componentes.

Cuadro 101. Configuración para habilitar Prometheus y Grafana en *globals.yaml*

```
enable_prometheus: "yes"
enable_grafana: "yes"

enable_prometheus_alertmanager: "yes"
enable_prometheus_cadvisor: "yes"
enable_prometheus_node_exporter: "yes"
enable_prometheus_openstack_exporter: "yes"
```

Donde:

- **enable_prometheus_alertmanager**: habilita el componente Alertmanager de Prometheus para gestionar alertas.

- **enable_prometheus_cadvisor**: habilita el exportador cAdvisor para monitorear contenedores.
- **enable_prometheus_node_exporter**: habilita el exportador Node Exporter para monitorear nodos.
- **enable_prometheus_openstack_exporter**: habilita el exportador OpenStack Exporter para monitorear servicios de OpenStack.

Después de modificar el archivo *globals.yaml*, se debe desplegar estos componentes utilizando Kolla-Ansible:

Cuadro 102. Comando para desplegar Prometheus y Grafana con Kolla-Ansible

```
# kolla-ansible deploy -i <inventory> -t prometheus,grafana
```

Donde:

inventory es el archivo de inventario utilizado para la instalación de Kolla-Ansible (all-in-one o multinode).

Verificar que los contenedores de Prometheus y Grafana estén corriendo correctamente:

Cuadro 103. Comando para verificar los contenedores de Prometheus

```
# docker ps | grep prometheus
```

La cantidad de contenedores y nombres pueden variar dependiendo de los componentes habilitados y la versión de Kolla-Ansible, pero debería mostrar algo similar a esto:

Cuadro 104. Salida con los contenedores de Prometheus

```
quay.io/openstack-kolla/prometheus-libvirt-exporter:master-ubuntu-noble
     ... Up 2 hours
quay.io/openstack-kolla/prometheus-blackbox-exporter:master-ubuntu-noble
      ... Up 2 hours
quay.io/openstack-kolla/prometheus-openstack-exporter:master-ubuntu-
    noble  ... Up 2 hours
quay.io/openstack-kolla/prometheus-cadvisor:master-ubuntu-noble
    ... Up 2 hours
quay.io/openstack-kolla/prometheus-memcached-exporter:master-ubuntu-
    noble  ... Up 2 hours
quay.io/openstack-kolla/prometheus-mysqld-exporter:master-ubuntu-noble
    ... Up 2 hours
quay.io/openstack-kolla/prometheus-node-exporter:master-ubuntu-noble
    ... Up 2 hours
quay.io/openstack-kolla/prometheus-server:master-ubuntu-noble
    ... Up 2 hours
```

Verificamos ahora los contenedores de Grafana:

Cuadro 105. Comando para verificar los contenedores de Grafana

```
# docker ps | grep grafana
```

Debería mostrar algo similar a esto:

Cuadro 106. Salida con los contenedores de Grafana

```
quay.io/openstack-kolla/grafana:master-ubuntu-noble  ... Up 2 hours
```

Verificar que los contenedores de Prometheus y Grafana estén estén *up*, buscar si hay contenedores en estado *exited*

Usando el siguiente comando:

Cuadro 107. Comando para verificar todos los contenedores

```
# docker ps -a
```

Este comando mostrará todos los contenedores, incluyendo los que están en estado *exited*. Si alguno de los contenedores de Prometheus o Grafana está en estado *exited*, reiniciarlos utilizando el siguiente comando:

Cuadro 108. Comando para reiniciar un contenedor

```
# docker restart <contenedor>
```

9.2. Verificación de Helm

Para que prometheus puede monitorear los nodos del clúster Kubernetes, *helm* debe ser instalado en los contenedores de Magnum. En versiones recientes de Kolla-Ansible, *helm* ya viene preinstalado en los contenedores de Magnum, pero si no es así, es necesario asegurarse de que *helm* esté instalado.

Para verificar si *helm* está instalado en los contenedores del servicio de Magnum (magnum-api y magnum-conductor), se puede acceder al contenedor y ejecutar los siguientes comandos:

Cuadro 109. Comando para acceder al contenedor de Magnum y verificar la versión de Helm

```
# docker exec -it <contenedor_magnum> bash
# helm version
```

Debería desplegarse la versión de Helm instalada, parecida a esta versión:

Cuadro 110. Salida con la versión de Helm instalada

```
version.BuildInfo{Version:"v3.16.3", GitCommit:"cdifne...", GitTreeState
    :"clean", GoVersion:"go1.22.7"}
```

9.3. Habilitar monitoreo en la plantilla de clúster

Con Prometheus y Grafana instalados, y helm verificado, se debe habilitar el monitoreo en la plantilla de clúster utilizada. Para esto, se debe actualizar la plantilla de clúster para habilitar el monitoreo de Prometheus. Antes de añadir las etiquetas, se debe de eliminar todo clúster creado con esa plantilla, ya que no se pueden modificar plantillas de clúster con clústeres asociados. Agregar las siguientes etiquetas a la plantilla de clúster:

Figura 20. Habilitar el monitoreo en la plantilla de clúster en Horizon

Nota. Esta imagen muestra la sección de etiquetas (labels) en la plantilla de clúster donde se habilita el monitoreo.

Donde:

- **monitoring_enabled**: habilita el monitoreo en el clúster.
- **prometheus_adapter_enabled**: habilita el adaptador de Prometheus para Kubernetes para métricas personalizadas de clúster.

Con esta modificación, cualquier nuevo clúster creado con esta plantilla tendrá el monitoreo habilitado. Prometheus recopilará métricas de los nodos y servicios del clúster, y Grafana podrá visualizar estas métricas a través de dashboards.

9.4. Acceso a Grafana

Antes de acceder a Grafana se necesita obtener la contraseña del usuario administrador. Para esto, se debe ejecutar el siguiente comando en el nodo donde está corriendo el contenedor de Grafana:

Cuadro 111. Comando para obtener la contraseña del usuario administrador de Grafana

```
# grep grafana_admin_password /etc/kolla/passwords.yml
```

Se mostrará un mensaje similar a este:

Cuadro 112. Salida con la contraseña del usuario administrador de Grafana

```
grafana_admin_password: aoiwnoennsvej35j3i5o3k3nsjai34 (Ejemplo)
```

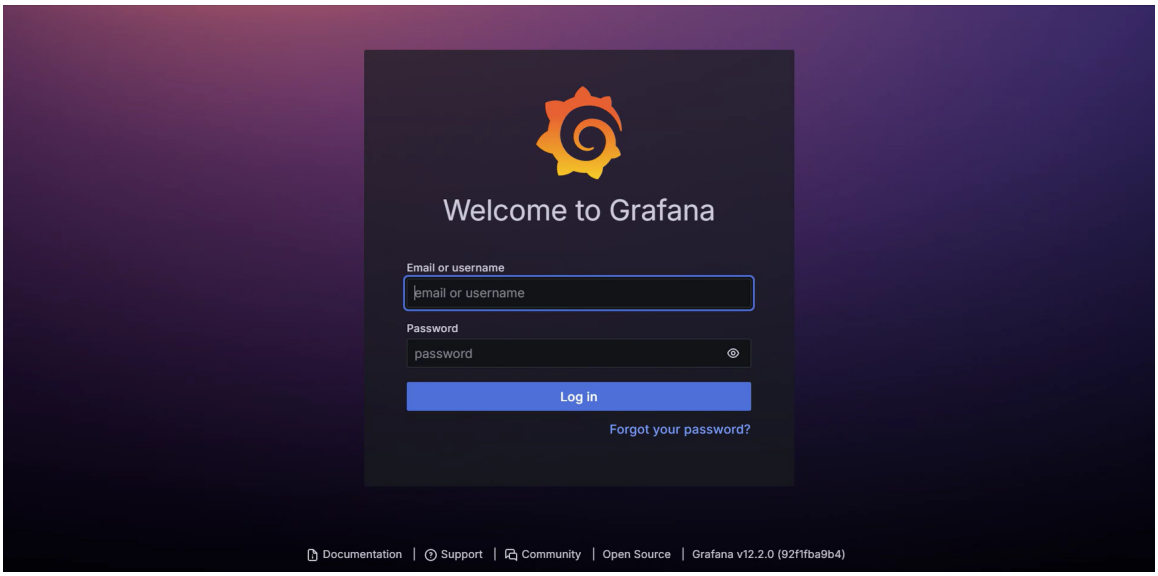
Con la contraseña obtenida, se puede acceder a la interfaz web de Grafana. En tu navegador preferido ingresar la URL de acceso la cual es:

Cuadro 113. URL de acceso a Grafana

```
http://192.168.74.5:3000
```

Se mostrará la pantalla de inicio de sesión de Grafana:

Figura 21. Pantalla de inicio de sesión de Grafana



Nota. Pantalla de inicio de sesión de Grafana donde se ingresan las credenciales de acceso.

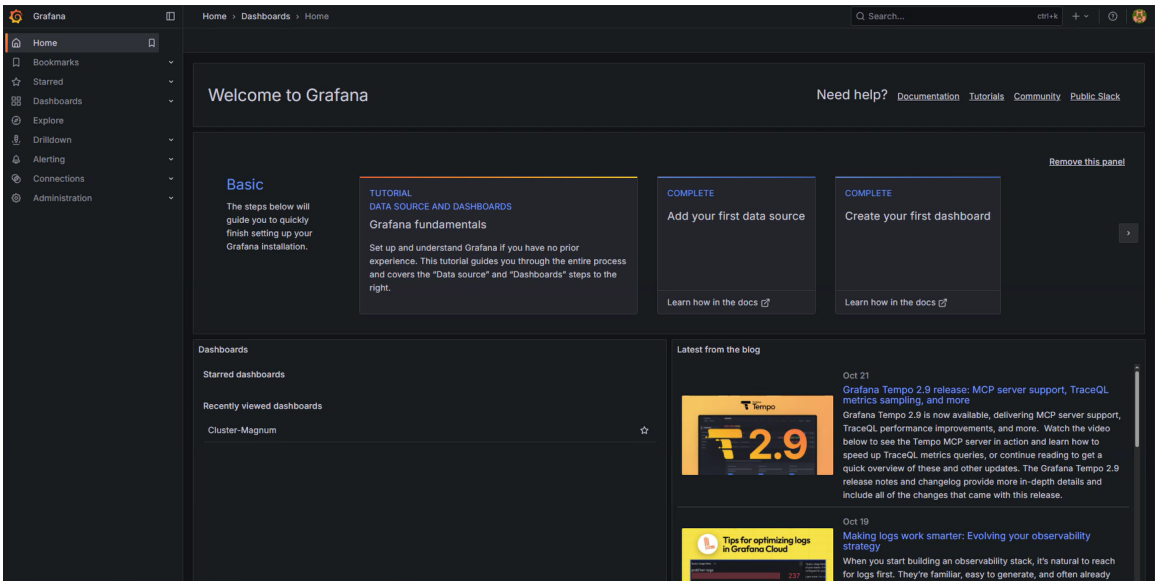
Ingresar las credenciales de acceso:

- Usuario: *admin*

- Contraseña: *aoiwnoennsvej35j3i5o3k3nsjai34*

Después de iniciar sesión, ingresaremos al panel principal de Grafana

Figura 22. Panel principal de Grafana

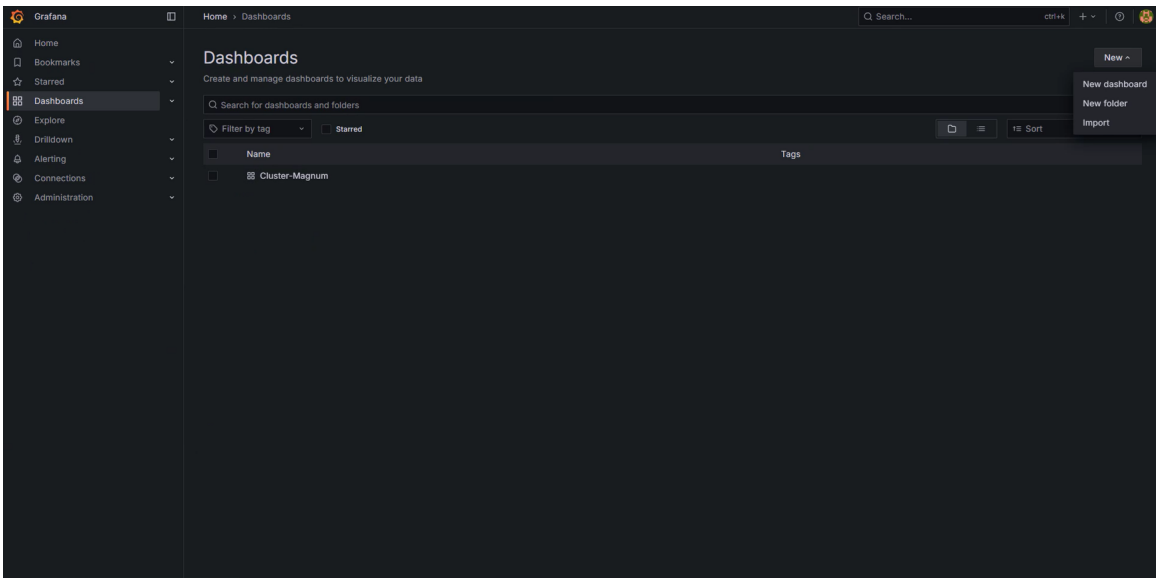


Nota. Panel principal de Grafana don de se puede acceder a las diferentes funcionalidades.

9.5. Creación de Dashboards en Grafana

Nos dirigimos a *dashboards*, le damos click en *New* y presionamos *New dashboard*.

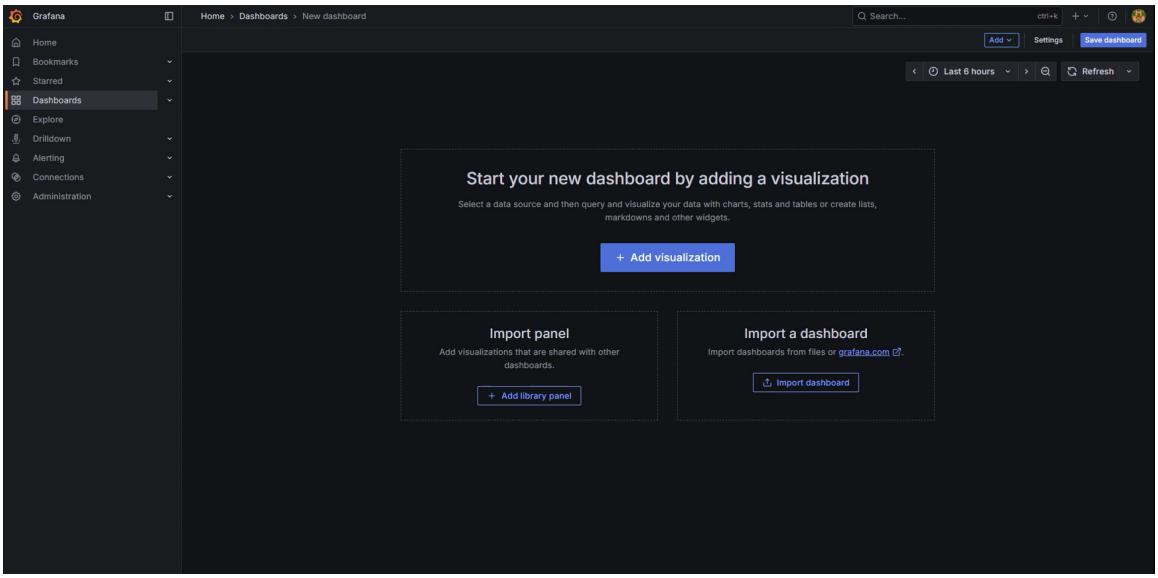
Figura 23. Creación de nuevo Dashboard en Grafana



Nota. Pantalla para la selección y creación de dashboards en Grafana.

Nos mostrará 3 opciones, crear una visualización, importar un *dashboard* o *panel* existente. Seleccionamos la opción *Add visualization*.

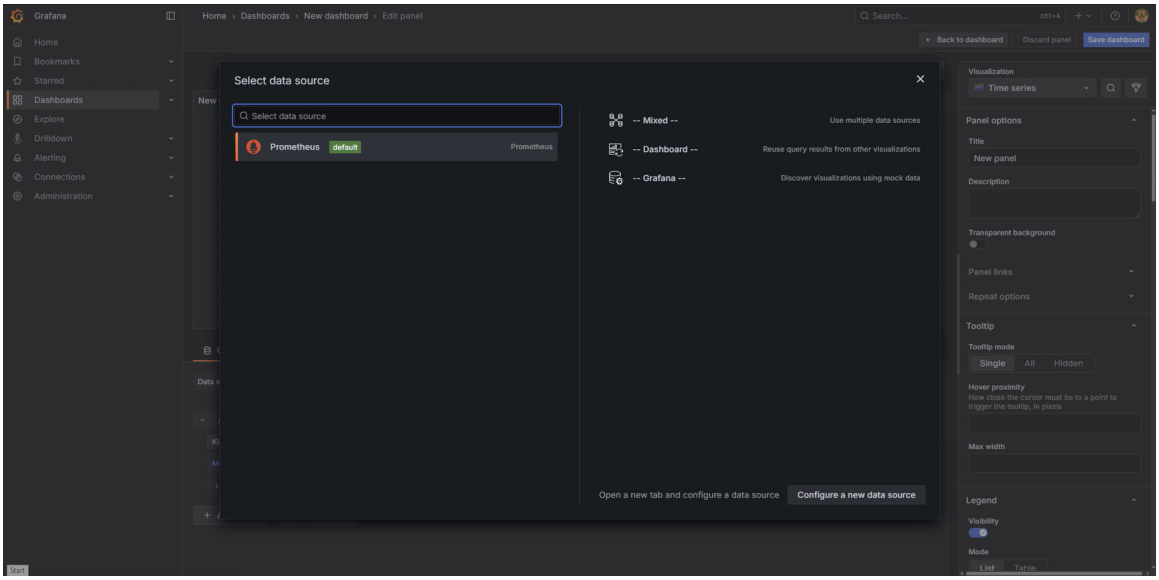
Figura 24. Agregar visualización en Grafana



Nota. Pantalla para agregar visualizaciones, paneles e importar dashboards en Grafana.

Al seleccionar esta opción, nos pedirá seleccionar la fuente de datos, seleccionamos *Prometheus*.

Figura 25. Selección de fuente de datos Prometheus en Grafana



Nota. Pantalla para la selección de la fuente de datos en Grafana.

9.6. Creación de visualizaciones

Prometheus recolecta diferentes tipos de métricas, entre las más comunes para el monitoreo de clústeres están las métricas de *node* y *container*.

Las métricas de *node* son las que recolectan información de los nodos del clúster, como uso de CPU, memoria, disco, red, etc.

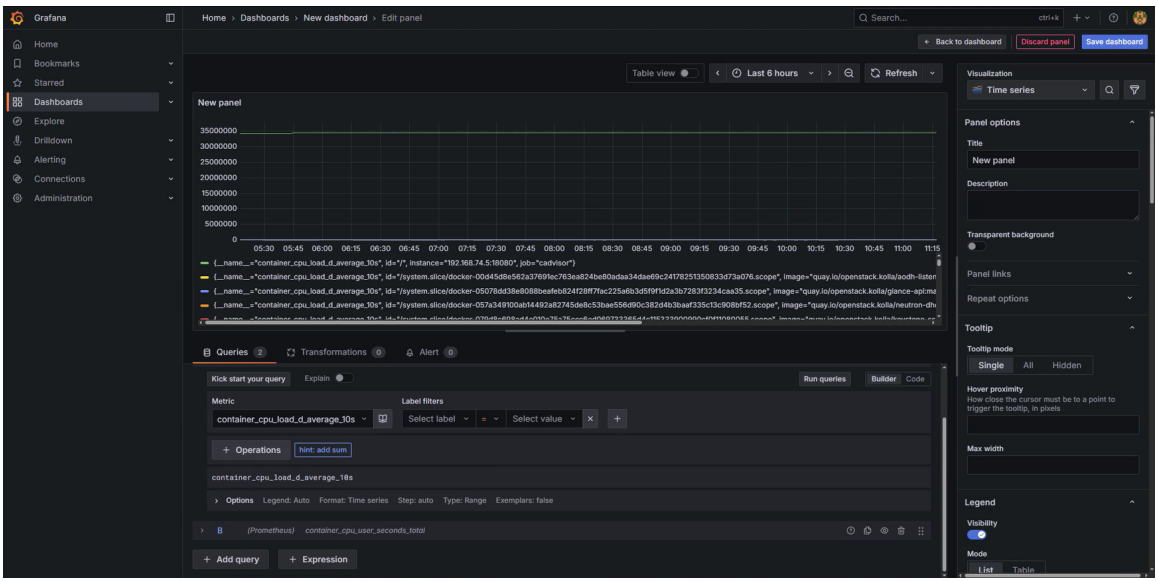
Las métricas de *container* son las que recolectan información de los contenedores que se están ejecutando de los componentes de OpenStack como de los nodos del clúster, como uso de CPU, memoria, disco, red, etc.

Para crear visualizaciones en Grafana, existen dos métodos principales para obtener y mostrar las métricas recolectadas por Prometheus.

9.6.1. Método 1

Se utiliza la opción *builder* para obtener métricas específicas de Prometheus en la sección de métricas donde dice *Metric*. Podemos añadir más de una métrica en la misma visualización dando click en *Add query*. Cuando hayamos seleccionado las métricas, damos click en *Run queries*. y nos mostrará la gráfica con las métricas seleccionadas.

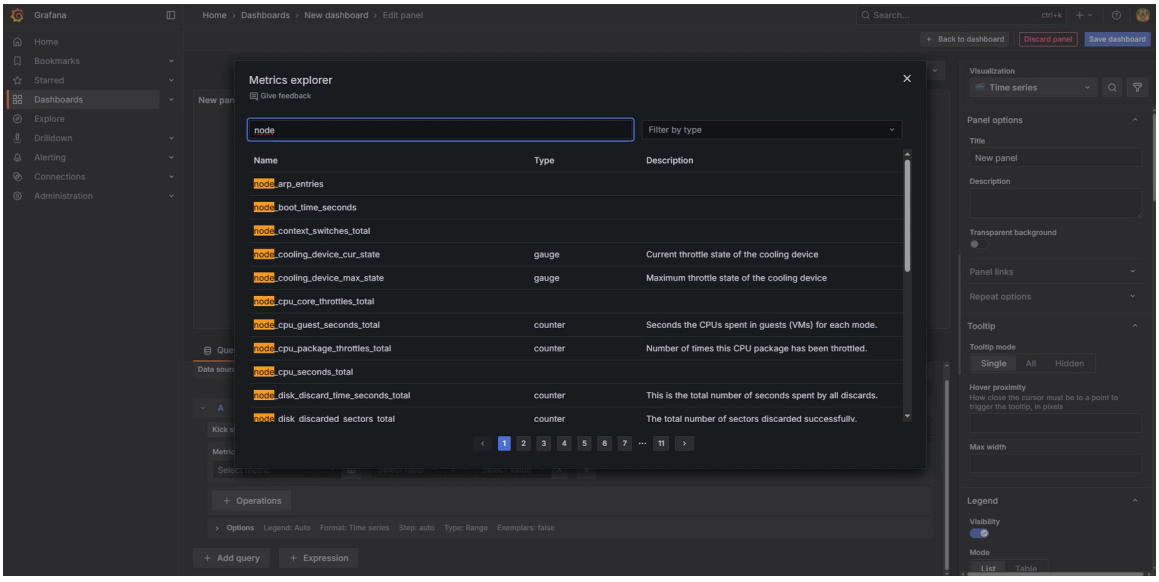
Figura 26. Constructor de consultas en Grafana



Nota. Pantalla para la construcción de consultas y previsualización en Grafana.

Se puede buscar la métrica deseada en el menú *Metric explorer* que se encuentra a la par de *Select metric*, escribimos el nombre de la métrica y seleccionamos la métrica deseada.

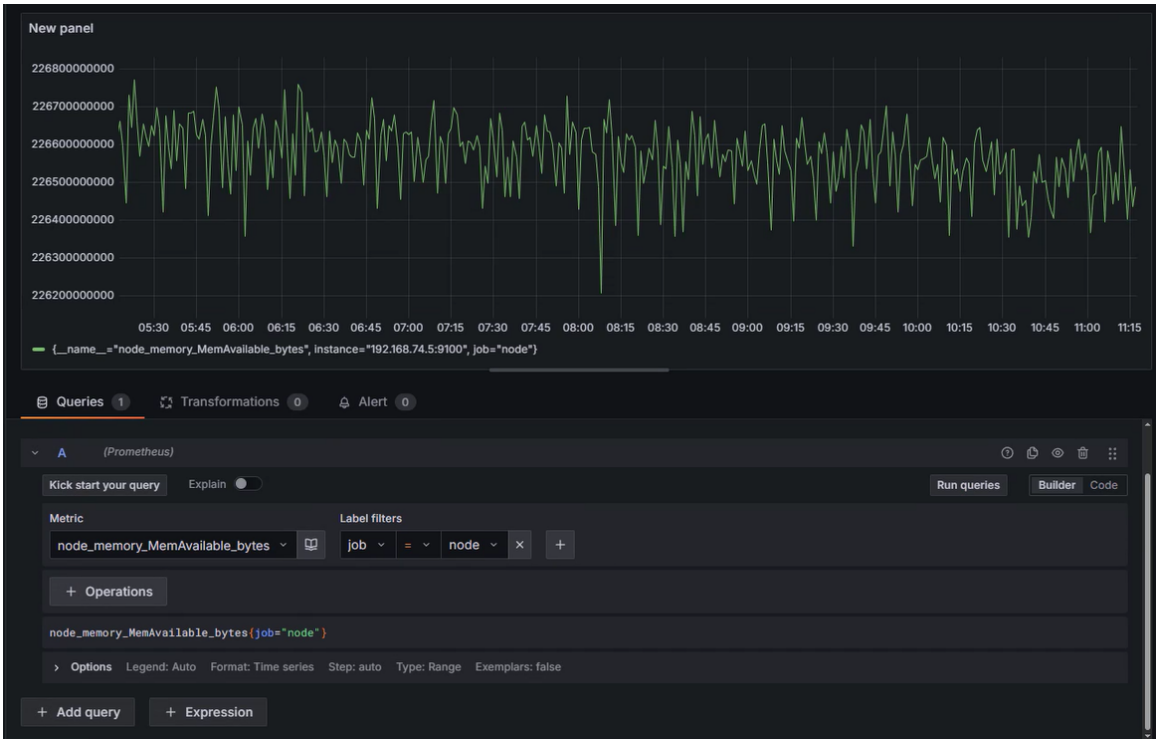
Figura 27. Explorador de métricas en Grafana



Nota. Pantalla del explorador de métricas en Grafana para facilitar la selección de métricas.

Se puede monitorear un clúster con varios nodos, filtrando por el nombre del nodo en la sección de *Label filters* seleccionando la etiqueta *job* y el nombre del nodo deseado, o se puede monitorear una instancia específica de contenedor, filtrando por la etiqueta *instance* y la ip y puerto del contenedor. Si se desea monitorear todos los nodos, se puede dejar sin filtro.

Figura 28. Filtros de métricas en Grafana

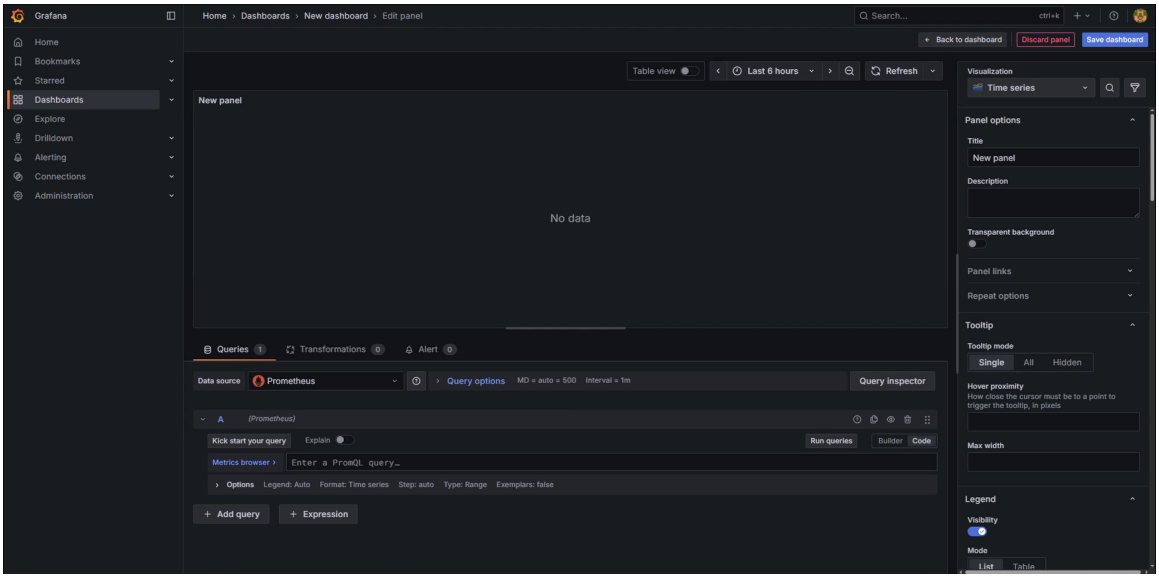


Nota. Aplicación de filtros en las métricas seleccionadas en Grafana.

9.6.2. Método 2

Se utiliza la opción *code* para escribir consultas en lenguaje PromQL.

Figura 29. Editor de código en Grafana



Nota. Pantalla del editor de código en Grafana para escribir consultas PromQL.

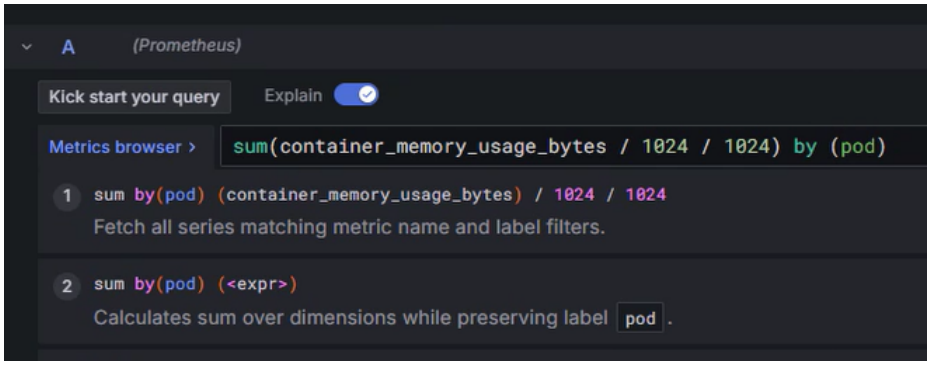
Se pueden utilizar consultas PromQL para obtener métricas específicas de Prometheus. A continuación, se presentan algunos ejemplos de consultas PromQL que se pueden utilizar para monitorear diferentes aspectos del clúster Kubernetes.

Cuadro 114. Ejemplos de consultas PromQL para Grafana

Panel	**Comando de PromQL**
% CPU Usado	`100-(avgby(instance)(rate(node_cpu_seconds_total{mode="idle"}[5m]))*100)`
% Memoria Usada	`100*(1-(node_memory_MemAvailable_bytes/node_memory_MemTotal_bytes))`
% Disco Usado	`100*(1-(node_filesystem_avail_bytes/node_filesystem_size_bytes))`
Bytes Recibidos y Transmitidos por Segundo	`rate(node_network_receive_bytes_total[5m])/1024/1024` y `rate(node_network_transmit_bytes_total[5m])/1024/1024`

Dependiendo de lo deseado se puede sumar, restar, promediar o agrupar las métricas utilizando las funciones de PromQL. Si no se sabe que hace una función u operador, se puede consultar la documentación oficial de PromQL en `https://prometheus.io/docs/prometheus/latest/querying/basics/` o en algunos casos se puede activar la opción *Explain* que se encuentra en la parte inferior del editor de código para obtener una explicación de la consulta escrita.

Figura 30. Explicación de consultas PromQL en Grafana

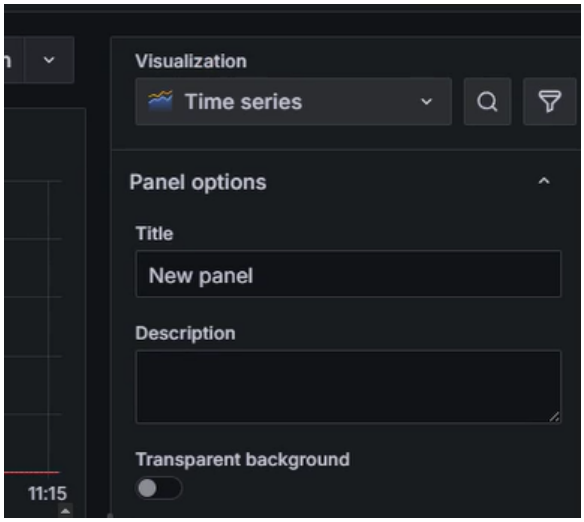


Nota. Explicación de una consulta PromQL en Grafana.

9.7. Guardar cambios y Dashboard

Al finalizar la configuración de la visualización, modificar el título del panel en la sección *Panel options* y en la parte de *Title* escribir el nombre deseado para la visualización.

Figura 31. Título del panel en Grafana



Nota. Sección para la modificación del título del panel en Grafana.

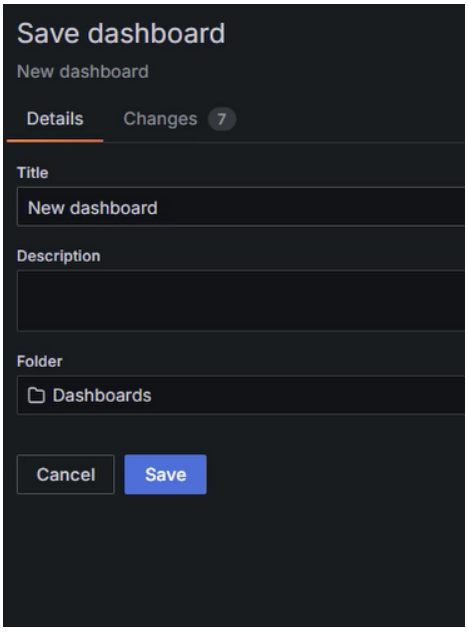
Para ver la visualización creada, presionar en *Back to dashboard* para regresar al dashboard principal y ver la visualización. Para guardar el dashboard o cualquier cambio realizado, dar click en *Save dashboard* en la parte superior derecha.

Figura 32. Guardar Dashboard en Grafana

Nota. Opción para guardar el dashboard en Grafana.

Escribirle un nombre para el dashboard y presionar en *Save*.

Figura 33. Nombre del Dashboard en Grafana



Nota. Sección para asignar un nombre al dashboard y guardarlo en una carpeta en Grafana.

9.8. Dashboard ejemplo

A medida que se agreguen más visualizaciones al dashboard, se van a ir acomodando automáticamente. Se pueden mover las visualizaciones arrastrándolas y soltándolas en la posición deseada dentro del dashboard. Este es un ejemplo de un dashboard con varias visualizaciones:

Figura 34. Ejemplo de Dashboard en Grafana

Nota. Ejemplo de un dashboard en Grafana con varias visualizaciones de métricas.

Se puede ver en la esquina superior derecha del dashboard, se puede modificar el rango de tiempo para las visualizaciones, así como el tiempo de actualización automática. Esto es útil para monitorear el clúster en tiempo real o para analizar datos históricos.

CAPÍTULO 10

Conclusiones

- Se logró la réplica del entorno OpenStack en el segundo nodo *compute*, sin embargo, por recomendación del trabajo anterior, se encontró una mejor manera de realizar la instalación y configuración utilizando el proyecto *Kolla-Ansible* simplificando y automatizando el proceso de despliegue de OpenStack.
- Se implementó y configuró el servicio de orquestación de contenedores Magnum para la creación y gestión de clústeres de contenedores de manera nativa en OpenStack, junto con su plugin en la interfaz gráfica Horizon.
- Se creó con éxito un clúster de contenedores utilizando Kubernetes como orquestador, y se verificó su correcto funcionamiento desplegando una instancia de prueba en el clúster.
- Se implementaron y configuraron los servicios de Prometheus y Grafana para el monitoreo básico del estado y rendimiento de los clústeres de contenedores desplegados por Magnum en OpenStack.
- Se documentó todo el proceso de instalación, configuración, despliegue y monitoreo de los clústeres de contenedores de Magnum en OpenStack, creando una guía que facilite su replicación para futuros trabajos.
- El proceso de implementación validó el despliegue de clústeres de contenedores utilizando Kubernetes en la infraestructura de OpenStack con resultados predecibles y replicables.
- La integración de Prometheus y Grafana permite un entendimiento claro de las tendencias del consumo de recursos y desempeño de los clústeres a través de los nodos y contenedores.

CAPÍTULO 11

Recomendaciones

1. Debido a errores en la instalación y complicaciones con el despliegue de los clústeres, no se pudo realizar pruebas con el balanceador de carga Octavia. Se recomienda que en futuros trabajos se realicen pruebas con Octavia para evaluar el desempeño de los clústeres bajo diferentes cargas de trabajo.

2. Se recomienda aprender e implementar CAPI (Cluster API) para la creación y gestión de clústeres, ya que es una herramienta que permite que K8s clústeres puedan desplegar K8s clústeres adicionales, eliminando la necesidad de usar scripts basados en Heat.

3. Profundizar en la configuración de Prometheus y Grafana para crear visualizaciones más avanzadas y crear alertas basadas en métricas específicas de los clústeres de contenedores.

4. Experimentar con las opciones de *Autohealer* y *Autoscaler* de Magnum para mejorar la escalabilidad y adaptabilidad de los clústeres en un ambiente de producción. También aumentar el espacio de almacenamiento usando Cinder para los nodos del clúster.

5. Replicar lo realizado en este trabajo utilizando los otros orquestadores soportados, como Docker Swarm y Apache Mesos, para comparar su desempeño y facilidad de uso en comparación con Kubernetes.

6. En futuros trabajos, utilizar máquinas virtuales como ambiente de pruebas antes de implementarlo en hardware físico, para aprovechar características como hacer *snapshots* que permiten revertir cambios en caso de errores o resultados no deseados durante la configuración y evitar dañar la infraestructura existente funcional.

CAPÍTULO 12

Referencias

[1] E. Castillo, «Automatización de despliegue de plataforma de Cloud Computing Openstack: Una estrategia escalable empleando Infraestructura como Código,» nov. de 2025.

[2] T. O. Foundation, *OpenStack Installation Guide — Installation Guide documentation.* dirección: https://docs.openstack.org/install-guide/#

[3] N. Martinelli, «OpenStack Magnum on the CERN production cloud - Superuser,» *Superuser*, abr. de 2016. dirección: https://superuser.openinfra.org/articles/openstack-magnum-on-the-cern-production-cloud/

[4] F. J. L. Turcios, «Despliegue modular de la plataforma de cloud computing OpenStack en la red de laboratorios del Departamento de Electrónica de la Universidad del Valle de Guatemala,» *Universidad del Valle de Guatemala (UVG)*, págs. 1-135, ene. de 2024.

[5] E. Casalicchio y S. Iannucci, «The state-of-the-art in container technologies: Application, orchestration and security,» *Concurrency and Computation: Practice and Experience*, vol. 32, e5668, sep. de 2020. dirección: https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.5668%20https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5668%20https://onlinelibrary.wiley.com/doi/10.1002/cpe.5668

[6] A. Randal, «The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers,» *ACM Computing Surveys (CSUR)*, vol. 53, 1 feb. de 2020, ISSN: 15577341. DOI: 10.1145/3365199 dirección: https://dl.acm.org/doi/10.1145/3365199

[7] O. Bentaleb, A. S. Belloum, A. Sebaa y A. El-Maouhab, «Containerization technologies: taxonomies, applications and challenges,» *Journal of Supercomputing*, vol. 78, págs. 1144-1181, 1 ene. de 2022, ISSN: 15730484. DOI: 10.1007/S11227-021-03914-1/METRICS dirección: https://link.springer.com/article/10.1007/s11227-021-03914-1

[8] A. Ghani, A. Badshah, S. Jan, A. A. Alshdadi y A. Daud, «Issues and challenges in Cloud Storage Architecture: A Survey,» *SSRN Electronic Journal*, vol. 1, págs. 50-65, 1 abr. de 2020. DOI: 10.2139/ssrn.3630761 dirección: https://arxiv.org/abs/2004.06809v2

[9] J. S. Ward y A. Barker, «A Cloud Computing Survey: Developments and Future Trends in Infrastructure as a Service Computing,» jun. de 2013. dirección: https://arxiv.org/abs/1306.1394v1

[10] P. Mell y T. Grance, «The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,» sep. de 2011. DOI: 10.6028/NIST.SP.800-145

[11] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu y W. Zhou, «A Comparative Study of Containers and Virtual Machines in Big Data Environment,» *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2018-July, págs. 178-185, sep. de 2018, ISSN: 21596190. DOI: 10.1109/CLOUD.2018.00030

[12] R. Badre, A. K. Gupta, A. Lingayat, R. R. Badre y A. K. Gupta, «Integration of Linux Containers in OpenStack: An Introspection,» *Article in TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 12, págs. 1094-1105, 3 2018, ISSN: 2502-4752. DOI: 10.11591/ijeecs.v12.i3.pp1094-1105

[13] B. Bermejo, C. Juiz y C. Guerrero, «Virtualization and consolidation: a systematic review of the past 10 years of research on energy and performance,» *Journal of Supercomputing*, vol. 75, págs. 808-836, 2 feb. de 2019, ISSN: 15730484. DOI: 10.1007/S11227-018-2613-1/METRICS dirección: https://link.springer.com/article/10.1007/s11227-018-2613-1

[14] A. Kovacs, «Comparison of different linux containers,» *2017 40th International Conference on Telecommunications and Signal Processing, TSP 2017*, vol. 2017-January, págs. 47-51, oct. de 2017. DOI: 10.1109/TSP.2017.8075934

[15] Marcel, «Performance Evaluation of MikroTik-based Virtual Machine for Small-Scale Network Virtualization on VMware Platform,» *Proceedings - 2018 International Conference on Control, Electronics, Renewable Energy and Communications, ICCEREC 2018*, págs. 154-158, jul. de 2018. DOI: 10.1109/ICCEREC.2018.8712000

[16] A. Lingayat, R. R. Badre y A. K. Gupta, «Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine,» *Proceedings of the 3rd International Conference on Communication and Electronics Systems, ICCES 2018*, págs. 1019-1023, oct. de 2018. DOI: 10.1109/CESYS.2018.8723998

[17] I. M. A. Jawarneh et al., «Container Orchestration Engines: A Thorough Functional and Performance Comparison,» *IEEE International Conference on Communications*, vol. 2019-May, mayo de 2019.

[18] S. JE, *Virtual Machines: Versatile Platforms for Systems and Processes*, T. M. K. S. in Computer Architecture y D. Series, eds. Morgan Kaufmann Publishers, 2005. dirección: https://books.google.com.gt/books?hl=es&lr=&id=JPhQw41vD2MC&oi=fnd&pg=PP1&dq=Virtual+machines:+versatile+platforms+for+systems+and+processes.+The+Mor-+gan+Kaufmann+Series+in+Computer+Architecture+and+Design+Series&ots=TSdOd6vvhN&sig=k24RtLsmh6H8bJpc867cuQh7VVg&redir_esc=y#v=onepage&q&f=false

[19] C. Pahl, «Containerization and the PaaS Cloud,» *IEEE Cloud Computing*, vol. 2, págs. 24-31, 3 mayo de 2015, ISSN: 23256095. DOI: 10.1109/MCC.2015.51

[20] B. Burns, J. Beda, K. Hightower y L. Evenson, *Kubernetes: Up and Running: Dive into the Future of Infrastructure - Brendan Burns, Joe Beda, Kelsey Hightower, Lachlan Evenson - Google Libros*, Third Edition, B. Burns, J. Beda, K. Hightower y L. Evenson, eds. O'Reilly Media, Inc., ago. de 2022. dirección: https://books.google.com.gt/books?hl=es&lr=&id=KeB-EAAAQBAJ&oi=fnd&pg=PT150&dq=Kubernete&ots=VaQWIQsnU9&sig=5otpJSmOxJdHcoiMb1zRIXrNQYU&redir_esc=y#v=onepage&q&f=false

[21] Davidochobits, *¿Qué es OpenStack? - ochobitshacenunbyte*, mar. de 2015. dirección: https://www.ochobitshacenunbyte.com/2015/03/27/openstack/

[22] A. A. Siddiqui, *OpenStack Orchestration*, A. A. Siddiqui, ed. Packt Publishing, oct. de 2015. dirección: https://books.google.com.gt/books?hl=es&lr=&id=RP9_CwAAQBAJ&oi=fnd&pg=PP1&dq=Openstack+container+orchestration&ots=QnnGXcPniJ&sig=TZOjf8GsxYqBpQRPvOEWSl_vPBY&redir_esc=y#v=onepage&q&f=false

[23] A. W. Services, *¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS*, 2024. dirección: https://aws.amazon.com/es/what-is/restful-api/

[24] T. O. Foundation, *Welcome to the Heat documentation! — openstack-heat 24.0.1.dev3 documentation.* dirección: https://docs.openstack.org/heat/2025.1/index.html

[25] T. O. Foundation, *Welcome to Magnum's Developer Documentation! — magnum 20.0 rc2.dev3 documentation.* dirección: https://docs.openstack.org/magnum/2025.1/index.html

[26] T. O. Foundation, *OpenStack Key Manager (barbican) — Barbican 21.1.0.dev10 documentation.* dirección: https://docs.openstack.org/barbican/latest/

[27] A. Poniszewska-Marańda y E. Czechowska, «Kubernetes Cluster for Automating Software Production Environment,» *Sensors 2021, Vol. 21, Page 1910*, vol. 21, pág. 1910, 5 mar. de 2021, ISSN: 1424-8220. DOI: 10.3390/S21051910 dirección: https://www.mdpi.com/1424-8220/21/5/1910/htm%20https://www.mdpi.com/1424-8220/21/5/1910

[28] Kubernetes, *Cluster Architecture | Kubernetes.* dirección: https://kubernetes.io/docs/concepts/architecture/

[29] T. O. Foundation, *Container Monitoring in Kubernetes — magnum 20.1.0.dev16 documentation.* dirección: https://docs.openstack.org/magnum/latest/user/monitoring.html

[30] M. Burillo, *Kubernetes monitoring with Prometheus, the ultimate guide*, 2021. dirección: https://sysdig.com/blog/kubernetes-monitoring-prometheus/

CAPÍTULO 13

Glosario

Alertmanager: componente de Prometheus que gestiona, agrupa y enruta alertas hacia canales de notificación.

API RESTful: interfaz basada en principios REST para interacción cliente-servidor mediante operaciones HTTP sobre recursos.

CaaS (*Containers as a Service*): modelo centrado en la provisión y gestión de clústeres de contenedores como servicio.

CIDR (*Classless Inter-Domain Routing*): notación para definir rangos de direcciones IP y máscaras de subred sin clases.

Cinder: servicio de almacenamiento en bloques de OpenStack que ofrece volúmenes persistentes para instancias y aplicaciones.

Clúster: conjunto de nodos (plano de control y trabajadores) que operan de forma coordinada para ejecutar cargas de trabajo y servicios.

Deployment: objeto de Kubernetes que declara el estado deseado de pods y gestiona actualizaciones, réplicas y retrocesos.

Docker: plataforma de contenedores que facilita empaquetado, distribución y ejecución de aplicaciones en entornos aislados.

Entorno de Ejecución de Contenedores: software que ejecuta contenedores, gestiona imágenes y aislamiento (por ejemplo, containerd).

etcd: almacén distribuido clave-valor que guarda el estado del clúster de Kubernetes y su configuración.

Exporter: agente o servicio que expone métricas en formato compatible con Prometheus (por ejemplo, node_exporter, kube-state-metrics).

Fedora CoreOS: distribución minimalista orientada a contenedores y Kubernetes.

Glance: servicio de catálogo y distribución de imágenes de sistemas operativos para su uso en instancias o nodos.

Grafana: plataforma de visualización para construir tableros (*dashboards*) e interpretar métricas de Prometheus y otras fuentes.

Heat: servicio de orquestación de OpenStack que despliega recursos mediante plantillas declarativas (YAML/JSON) para crear, actualizar y eliminar infraestructura.

Helm: gestor de paquetes para Kubernetes que define, versiona e instala aplicaciones mediante *charts*.

Hipervisor: capa de virtualización que permite ejecutar múltiples máquinas virtuales en un host físico con aislamiento de recursos.

Horizon: panel web de OpenStack que permite a usuarios y administradores gestionar proyectos y recursos, incluido el plugin de Magnum.

IaaS (*Infrastructure as a Service*): modelo de nube que ofrece recursos de cómputo, red y almacenamiento bajo demanda.

IP Flotante (*Floating IP*): dirección IP pública enrutable asignable dinámicamente a instancias o servicios para exponerlos al exterior.

Keystone: servicio de identidad de OpenStack que gestiona autenticación, autorización, usuarios, proyectos y emisión de tokens para acceder a APIs.

Kolla-Ansible: proyecto que despliega OpenStack con contenedores y playbooks de Ansible, simplificando instalación, actualización y operación.

kube-apiserver: componente del plano de control que expone la API de Kubernetes y valida/gestiona solicitudes.

kube-controller-manager: proceso que ejecuta los controladores de Kubernetes para mantener el estado deseado del clúster.

kube-scheduler: componente del plano de control que asigna pods a nodos según recursos, afinidades y restricciones.

Kubernetes (K8s): plataforma de código abierto para orquestación de contenedores que automatiza despliegue, escalamiento y operación de aplicaciones.

Magnum: servicio de OpenStack para la provisión y gestión automatizada de clústeres de contenedores (por ejemplo, Kubernetes) mediante API y panel en Horizon.

Metrics Server: componente ligero que agrega métricas de uso de CPU y memoria en Kubernetes para funciones como *kubectl top* y autoescalado.

Neutron: servicio de redes de OpenStack que provee conectividad L2/L3, direccionamiento IP, ruteo, NAT, balanceo y seguridad para instancias y servicios.

Nodo de Trabajo (*Worker Node*): equipo físico o virtual que ejecuta contenedores y pods asignados por el plano de control.

Nova: servicio de cómputo de OpenStack encargado del ciclo de vida de las instancias, su programación en hosts y la integración con hipervisores.

OpenStack: plataforma de software libre para computación en la nube (pública, privada o híbrida) con arquitectura modular que ofrece servicios de cómputo, redes, almacenamiento y gestión.

PaaS (*Platform as a Service*): modelo de nube que proporciona plataforma y herramientas para desarrollar, desplegar y gestionar aplicaciones.

Par de Claves (*Key Pair*): credencial basada en criptografía asimétrica usada para acceso seguro (por ejemplo, SSH) a instancias o nodos.

Plano de Control (*Control Plane*): conjunto de componentes que gestionan el estado global del clúster (programación, controladores y API).

Pod: unidad mínima de ejecución en Kubernetes que agrupa uno o más contenedores con red y almacenamiento compartidos.

Prometheus: sistema de monitoreo y alerta basado en extracción de métricas (pull) y consultas con PromQL; ampliamente usado en Kubernetes.

QCOW2 (*QEMU Copy-On-Write v2*): formato de imagen de disco con soporte de compresión y *snapshots*, usado comúnmente en OpenStack.

Red Externa (*External Network*) red en Neutron con salida a Internet que permite asignar IPs públicas, ruteo y NAT hacia recursos internos.

Red Proveída (*Provider Network*): red en Neutron mapeada a una red física del centro de datos (por ejemplo, tipo *flat*) para exponer conectividad externa.

ReplicaSet: controlador de Kubernetes que asegura un número definido de réplicas de pods en ejecución.

Router de Neutron: dispositivo lógico de capa 3 que interconecta redes y aplica ruteo y NAT entre redes privadas y la red externa.

SaaS (*Software as a Service*): modelo de nube que entrega aplicaciones listas para usar a través de Internet sin administrar la infraestructura subyacente.

Sabor (*Flavor*): plantilla de recursos de cómputo (vCPU, RAM, disco) para instancias o nodos en OpenStack.

Service: recurso de Kubernetes que expone un conjunto de pods como un servicio estable mediante una IP virtual y políticas de balanceo.

Subred (*Subnet*): segmento de red con rango CIDR, puerta de enlace y, opcionalmente, DHCP, usado para direccionamiento interno.

Swift: servicio de almacenamiento de objetos de OpenStack, distribuido y tolerante a fallos para archivos y datos no estructurados.

Tablero (*Dashboard*): conjunto de visualizaciones e indicadores en Grafana u otra herramienta para monitorear estado y rendimiento.

VXLAN (*Virtual eXtensible LAN*): tecnología de red superpuesta que encapsula tráfico de capa 2 sobre capa 3 para crear redes virtuales a gran escala.