Despliegue e implementación de *Open Network Automation Platform* para la automatización de redes en entornos virtuales y físicos

José Carlo Santizo Olivet

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Despliegue e implementación de *Open Network Automation Platform* para la automatización de redes en entornos virtuales y físicos

Trabajo de graduación presentado por José Carlo Santizo Olivet para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

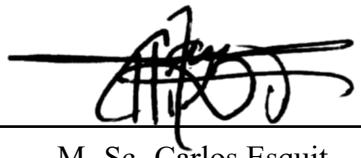
Despliegue e implementación de *Open Network Automation Platform* para la automatización de redes en entornos virtuales y físicos

Trabajo de graduación presentado por José Carlo Santizo Olivet para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2024

Vo.Bo.:

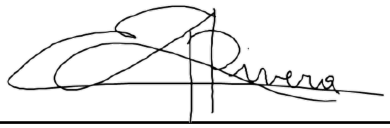
(f) 

M. Sc. Carlos Esquit

Tribunal Examinador:

(f) 

M.Sc. Carlos Esquit

(f) 

Dr. Luis Alberto Rivera Estrada

(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025.

Prefacio

A lo largo de mis estudios universitarios, he desarrollado un gran interés por las telecomunicaciones y la automatización. Mi acercamiento a estas áreas, especialmente al trabajar con tecnologías nuevas como lo es *Kubernetes*, me ha permitido combinar mi interés por las nuevas tecnologías con la posibilidad de automatizar y facilitar procesos en campos existentes como las telecomunicaciones. Este proyecto representa para mí una oportunidad invaluable para aplicar y expandir mis conocimientos en un ámbito que realmente me entusiasma.

Quiero expresar mi más sincero agradecimiento a mis papás, ya que gracias a ellos estoy donde estoy hoy en día. Han sido un soporte inmenso para mí, brindándome su amor y apoyo incondicional en cada paso de este camino. Les debo todo a ellos, y este logro es tanto mío como suyo. También, quiero agradecer a mi hermano, quien ha sido mi mejor amigo y una persona muy importante a lo largo de mis estudios y mi vida. Específicamente durante la universidad, su apoyo ha sido invaluable; de él he aprendido muchísimas cosas que han enriquecido tanto mi vida personal como académica.

Agradezco a Dios, ya que a través de Él he podido sobrellevar muchos problemas y he recibido innumerables bendiciones en mi vida. Su guía y fortaleza me han ayudado a superar obstáculos y a seguir adelante en momentos difíciles.

Finalmente, este proyecto es un reflejo de mi pasión por las telecomunicaciones y mi deseo de contribuir al avance de esta área mediante el uso de nuevas tecnologías para automatizar y mejorar procesos existentes. Estoy entusiasmado por lo que el futuro depara y ansioso por seguir aprendiendo y creciendo en este campo.

Índice

Lista de figuras

Lista de cuadros

Resumen

En este trabajo se implementó un clúster de Kubernetes de alta disponibilidad, compuesto por seis nodos: tres nodos de trabajo y tres nodos de control. Además, se incluyeron dos balanceadores de carga utilizando HAProxy y un servidor de almacenamiento NFS (Network File System). Este diseño permitía una distribución óptima de los recursos y garantizaba la disponibilidad de las aplicaciones dentro del clúster, asegurando un entorno robusto para pruebas y desarrollo en automatización de redes.

Para facilitar la administración y el despliegue eficiente de aplicaciones en el clúster, se instaló HELM como gestor de paquetes. Este sistema proporcionó una herramienta confiable para gestionar las aplicaciones, permitiendo que se desplieguen y actualicen de manera efectiva en el entorno de Kubernetes. Como prueba de concepto inicial, se intentó desplegar ONAP en el clúster para evaluar su capacidad de desplegar aplicaciones que permitan la automatización de servicios en redes; sin embargo, esto no fue posible debido a limitaciones de recursos y al entorno de implementación local por parte del clúster.

Para comprobar la funcionalidad del clúster, se desarrolló una API para la automatización de procesos en una red simulada en GNS3, utilizando Python y las tecnologías FastAPI, HTML y Bootstrap. La API integró funcionalidades de automatización y monitoreo de red mediante conexiones SSH con paquetes de Python como Netmiko, presentando una plataforma visual y controlable de configuración de redes, al combinarlo con HTML y Bootstrap. Finalmente, esta API fue desplegada exitosamente en el clúster, demostrando la capacidad del entorno para manejar aplicaciones críticas en redes, y validando así la eficiencia y disponibilidad del clúster para este tipo de aplicaciones.

Abstract

In this work, a high-availability Kubernetes cluster was implemented, consisting of six nodes: three worker nodes and three control nodes. Additionally, two load balancers using HAProxy and a Network File System (NFS) storage server were included. This design enabled optimal resource distribution and ensured application availability within the cluster, providing a robust environment for network automation testing and development.

To facilitate efficient application management and deployment in the cluster, HELM was installed as a package manager. This system provided a reliable tool for managing applications, allowing them to be deployed and updated effectively in the Kubernetes environment. As an initial proof of concept, an attempt was made to deploy ONAP in the cluster to evaluate its capacity to deploy applications that enable service automation in networks; however, this was not possible due to resource limitations and the local deployment environment of the cluster.

To verify the cluster's functionality, an API was developed to automate processes in a network simulated in GNS3, using Python and technologies such as FastAPI, HTML, and Bootstrap. The API integrated network automation and monitoring functionalities via SSH connections with Python packages like Netmiko, presenting a visual and controllable network configuration platform by combining it with HTML and Bootstrap. Finally, this API was successfully deployed in the cluster, demonstrating the environment's capability to handle critical network applications and thereby validating the cluster's efficiency and availability for such applications.

CAPÍTULO 1

Introducción

En los últimos años, el desarrollo de tecnologías de contenedorización y virtualización ha revolucionado la administración y despliegue de aplicaciones en diversos entornos. Kubernetes, al ser una plataforma de código abierto ampliamente utilizada, ha facilitado la orquestación y el manejo de aplicaciones en contenedores, permitiendo a las organizaciones optimizar sus recursos y reducir los tiempos de implementación. Este tipo de tecnologías es especialmente relevante en el ámbito de las telecomunicaciones, donde la necesidad de infraestructura escalable y de alta disponibilidad es crítica para la gestión eficiente de redes y servicios.

La virtualización y la orquestación automatizada son herramientas clave en la evolución hacia redes de nueva generación. Estas tecnologías permiten optimizar todos aquellos recursos físicos y virtuales disponibles, proporcionando un entorno flexible y adaptable para la implementación de funciones de red. Además, la capacidad de escalar aplicaciones y servicios de manera automatizada contribuye a la reducción de costos y a la mejora de la eficiencia operativa, facilitando el desarrollo de soluciones innovadoras en el contexto de redes definidas por software (SDN) y la virtualización de funciones de red (NFV).

Este trabajo explora la implementación de un clúster de Kubernetes de alta disponibilidad, diseñado específicamente para el despliegue de aplicaciones orientadas a la automatización de redes en entornos virtualizados o físicos. La adopción de estas tecnologías no solo promueve la eficiencia y la automatización, sino que también establece una base sólida para futuros avances en telecomunicaciones, ofreciendo una plataforma robusta y escalable para el manejo de diversas aplicaciones.

CAPÍTULO 2

Antecedentes

2.1. Implementación de funciones de red virtualizadas en una nube privada utilizando ONAP

En la tesis titulada "*Developing and Deploying NFV solutions with OpenStack, Kubernetes and Docker*"[1], se aborda la implementación de funciones de red virtualizadas (NFV) en una nube privada utilizando ONAP. El documento [1] detalla cómo se configuró la infraestructura de nube desde cero, incluyendo la instalación de *OpenStack* y otros componentes esenciales, siguiendo las mejores prácticas del mercado.

En esta tesis [1], *Kubernetes* juega un papel crucial en el diseño, la orquestación y la gestión de todos los elementos relacionados con las funciones de red virtualizadas. Durante el desarrollo del proyecto, se encontraron numerosos desafíos y obstáculos al desplegar ONAP, reflejando que aún está en desarrollo esta solución. La tesis no solo documenta estos desafíos, sino que también ofrece soluciones y alternativas para superarlos. La configuración adecuada de los servidores, dispositivos de red y las NICs, así como la instalación y configuración de ONAP y utilización de *Kubernetes*, son aspectos clave que se tratan en detalle para asegurar una implementación exitosa.

El objetivo final de este proyecto es crear un entorno en una nube privada con funciones de red virtualizadas completamente operativas, sirviendo como una guía práctica para otros interesados en desplegar ONAP junto con tecnologías nuevas como *Kubernetes*. La experiencia adquirida y las soluciones propuestas en este trabajo son aplicables a escenarios de mayor escalabilidad, incluyendo entornos de producción utilizados por operadores de telecomunicaciones.

2.2. Orquestación de servicios en redes 5G utilizando ONAP y MEC

En la tesis titulada "*Service Modelling and End-to-End Orchestration in 5G Networks*"[2], explora cómo las redes 5G aprovechan los avances en técnicas de virtualización como NFV (*Network Function Virtualization*), SDN (*Software Defined Networks*) y MEC (*Multi Access Edge Computing*) para ofrecer mayor ancho de banda y menor latencia. Pero, también identifica la necesidad de automatización y capacidades avanzadas de gestión y orquestación como desafíos críticos para las nuevas redes 5G. En este contexto, la tesis enfatiza el desarrollo de soluciones innovadoras que aprovechan las ventajas de estas tecnologías emergentes. ONAP se presenta como una plataforma de código abierto relevante y orientada a estándares, diseñada para soportar la gestión y orquestación de servicios e infraestructuras de extremo a extremo y multi-dominio.

La tesis [2] detalla cómo el proyecto Mobilizer 5G, una iniciativa portuguesa, desarrolla soluciones innovadoras para el ecosistema 5G, enfocándose en casos de uso como la entrega de contenido multimedia a pasajeros de vehículos. Estos servicios se benefician de las estrategias de caché mejoradas y de la tecnología MEC para reducir la latencia, destacando la necesidad de capacidades de orquestación para mantener la calidad del servicio y optimizar los recursos. Para abordar estos desafíos, la tesis se enfoca en estudiar, comprender e implementar una solución de orquestación alineada con la producción, utilizando ONAP, y validar esta plataforma mediante la implementación de una solución de orquestación para un servicio virtual de *Content Delivery Network* (vCDN). Esta investigación [2] proporciona una guía práctica para la implementación de ONAP en un entorno 5G, destacando su potencial para mejorar la eficiencia y automatización de las redes de próxima generación.

2.3. Automatización de servicios de extremo a extremo en redes de telecomunicaciones

En el trabajo titulado "*Experimentation of End-to-End Telco Services through a service portal operating on an ONAP orchestrator*"[3], se describe cómo la infraestructura utilizada por los operadores de red para entregar servicios a los usuarios en el borde de la red está evolucionando hacia una arquitectura nativa de la nube. Como resultado, los servicios de extremo a extremo se implementan como cadenas de Funciones de Red Físicas (PNFs) y Funciones de Red Virtualizadas (VNFs), requiriendo un proceso automatizado para gestionar su ciclo de vida. El ONAP se destaca como una de las alternativas disponibles para gestionar estos servicios, proporcionando capacidades de diseño, creación, monitoreo de KPI y control de bucle cerrado. La tesis se centra en ayudar a los Proveedores de Servicios de Comunicación a gestionar esta infraestructura y todas las operaciones comerciales relacionadas, siguiendo las especificaciones estándar de API REST del programa Open API del TeleManagement Forum.

ONAP se sitúa en la capa de Recursos , la cual es la más baja dentro de la jerarquía definida por el TM Forum. Específicamente, ONAP se encarga de gestionar los recursos ubicados en diferentes Oficinas Centrales en la Nube (Cloud-COs) y se comunica con apli-

caciones en la capa de Servicios, a través de su componente de APIs Externas (ONAP NBI). La implementación de las APIs Externas de ONAP está basada en las especificaciones REST del TM Forum, permitiendo a las aplicaciones en la capa de Servicios desencadenar acciones sobre los recursos de manera estandarizada. Este componente ha sido estudiado, probado y evaluado en la tesis. En la capa de Servicios, se encuentran aplicaciones de Gestión de Órdenes de Servicio, como el *Service Resolver* desarrollado por *Orange*, un proyecto de código abierto que ha sido probado y mejorado. *Service Resolver* expone un conjunto de APIs REST basadas en el estándar del TM Forum y acepta solicitudes de aplicaciones BSS, mapeándolas en solicitudes para ONAP NBI para iniciar la instanciación de los recursos necesarios.

Finalmente, en la capa de Producto, las aplicaciones BSS son responsables de mapear las solicitudes de los clientes en solicitudes para las aplicaciones SOM. El trabajo realizado en esta capa incluye la implementación de una aplicación BSS llamada Service Portal, que consta de una interfaz gráfica desde donde los clientes pueden solicitar productos y un componente *backend* que envía solicitudes a Service Resolver, permitiendo la instanciación del servicio correspondiente al producto adquirido. La cadena de funcionalidades implementadas en las tres capas descritas realiza los pasos necesarios para que los proveedores de servicios instancien servicios de extremo a extremo de acuerdo con las solicitudes de los clientes.

2.4. Implementación de la automatización de redes en entornos de telecomunicaciones utilizando Kubernetes

En la tesis titulada "*Cloud Native Intent Automation*" [4], se habla sobre la implementación de la automatización en redes mediante el uso de *Kubernetes* como plataforma principal para la gestión de aplicaciones en contenedores. La tesis explora cómo los proveedores de infraestructuras, especialmente aquellos en el sector de telecomunicaciones, enfrentan el desafío de gestionar múltiples clústeres de *Kubernetes* en ambientes distribuidos y descentralizados, con el objetivo de asegurar conexiones privadas y estables entre ubicaciones remotas [4].

Kubernetes se utiliza en este contexto como un orquestador de contenedores que permite despliegues automatizados, escalabilidad y una administración optimizada de aplicaciones. Además, la tesis incorpora el uso de *plugins* como Cilium para la gestión avanzada de redes, así como Nephio para la orquestación de servicios, facilitando la conectividad y automatización de configuraciones en un entorno multi-clúster. Este enfoque reduce la necesidad de configuraciones manuales y asegura una infraestructura más uniforme y adaptable [4].

El objetivo principal de este trabajo es demostrar un sistema de despliegue automatizado que integra soluciones de red avanzadas, validando la posibilidad de que infraestructuras complejas en telecomunicaciones puedan ser desplegadas y gestionadas de manera eficiente mediante herramientas de automatización basadas en *Kubernetes*. Esta investigación provee una guía práctica para la implementación de tecnologías *cloud-native* en redes de telecomunicaciones, destacando su potencial para mejorar la eficiencia y reducir el tiempo de despliegue en redes de próxima generación.

2.5. Automatización de servicios en redes 5G mediante Kubernetes y OpenShift

En la tesis "*5G Cloud-Native: Network Management & Automation*" [5], se explora la automatización de redes en un entorno *cloud-native*, particularmente enfocado en la implementación y gestión de redes 5G. Este trabajo destaca el papel de *Kubernetes* y OpenShift Operator como herramientas clave para la orquestación de servicios complejos en redes 5G, proporcionando capacidades de escalabilidad, auto-recuperación y despliegue rápido de servicios [5].

Kubernetes permite una flexibilidad crucial en redes 5G al soportar arquitecturas de RAN (*Radio Access Network*) disgregadas y configuraciones dinámicas. Esto facilita cambios de configuración automáticos y la adaptación a diferentes niveles de servicio según las necesidades de la red. La tesis también aborda el uso de *OpenShift* Operator para administrar y actualizar servicios de red automáticamente, garantizando una infraestructura adaptable que puede manejar demandas de redes virtualizadas en 5G [5].

El objetivo de esta investigación es demostrar cómo una arquitectura basada en *Kubernetes* permite la automatización y escalabilidad requeridas para soportar servicios 5G, optimizando el uso de recursos y garantizando un rendimiento de red que cumple con las especificaciones de la industria.

2.6. Automatización de clústeres de Kubernetes para la orquestación de funciones de red virtualizadas

En la tesis "*Automation and Provisioning of Kubernetes on Bare-Metal Telco Edge Infrastructures*"[6], se analiza el despliegue de clústeres de *Kubernetes* en entornos de telecomunicaciones con infraestructura bare-metal, orientado a la orquestación de funciones de red virtualizadas (VNFs). Este trabajo implementa un marco de referencia modular y extensible, adaptado a las necesidades de los proveedores de telecomunicaciones europeos y enfocado en facilitar la administración de nodos distribuidos [6].

Kubernetes desempeña un papel crucial en la estandarización del proceso de despliegue, asegurando que todos los componentes de la infraestructura de red sean gestionados de manera uniforme. La arquitectura propuesta incluye operadores y definiciones de recursos personalizados (CRDs) que simplifican la configuración de nodos bare-metal, permitiendo una colaboración más efectiva entre administradores de infraestructura y desarrolladores de aplicaciones [6].

El objetivo principal de este estudio es optimizar la agilidad del despliegue en entornos de telecomunicaciones, al tiempo que se proporciona una guía práctica para implementar *Kubernetes* en infraestructuras de borde, mejorando así la eficiencia operativa en redes de telecomunicaciones.

2.7. Casos de éxito de automatización de redes con Kubernetes en la industria de telecomunicaciones

A nivel global, empresas en telecomunicaciones han comenzado a adoptar *Kubernetes* para la gestión y automatización de sus redes, beneficiándose de sus capacidades de escalabilidad y orquestación en redes de próxima generación [7]. Empresas como AT&T, Vodafone y China Mobile han integrado *Kubernetes* en sus infraestructuras, aprovechando su capacidad para gestionar servicios de red tanto físicos como virtuales, así como la monitorización de indicadores clave de rendimiento (KPIs) [7].

La adopción de *Kubernetes* no solo mejora la eficiencia operativa, sino que también reduce el tiempo de despliegue de nuevos servicios. La plataforma ha demostrado ser una herramienta valiosa para las redes *cloud-native*, especialmente en el contexto de la creciente demanda de servicios automatizados y escalables en la industria de telecomunicaciones.

CAPÍTULO 3

Justificación

El despliegue de un clúster de *Kubernetes* de alta disponibilidad en redes simuladas o físicas es un tema de gran relevancia en el campo de las redes de telecomunicaciones. *Kubernetes* es una plataforma de código abierto que proporciona un conjunto de herramientas y servicios para la orquestación y gestión de aplicaciones en contenedores, permitiendo a los proveedores de servicios de telecomunicaciones mejorar la eficiencia operativa y acelerar la implementación de nuevos servicios.

Kubernetes presenta varios beneficios. En primer lugar, permite familiarizarse con una plataforma de orquestación de contenedores ampliamente utilizada en la industria. Actualmente, las telecomunicaciones del mundo se están desplazando hacia procesos de nuevas generaciones, para lo cual se vuelven esenciales los procesos de orquestación y automatización en tiempo real, dirigidos por políticas de red, para redes físicas, simuladas y en arquitectura *cloud native*. Con estas funcionalidades, los nuevos servicios de red y el mantenimiento completo del ciclo de vida de las aplicaciones nativas en la nube se pueden automatizar rápidamente, lo cual es crítico para 5G y las redes de próxima generación. Con esto, el despliegue de *Kubernetes* permite a futuras generaciones ampliar sus conocimientos en el campo de las redes de telecomunicaciones y adquirir habilidades en tecnologías emergentes.

Aparte de lo anteriormente mencionado, el despliegue de un clúster de *Kubernetes* de alta disponibilidad en redes simuladas o físicas ofrece un entorno de prueba seguro y controlado para experimentar con diferentes configuraciones y escenarios de red. Dicho entorno de prueba nos permite utilizar las diferentes funciones de orquestación y automatización de aplicaciones, lo cual facilita la investigación y el desarrollo de nuevas soluciones y servicios de telecomunicaciones, así como la evaluación de su rendimiento y escalabilidad. Con todo esto, se tiene la oportunidad de utilizar herramientas didácticas para realizar todas las pruebas necesarias para comprender toda la arquitectura de *Kubernetes* y sus componentes, manejando de esta forma entornos con tecnologías de telecomunicaciones de última generación.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Despliegue de un clúster de *Kubernetes* de alta disponibilidad, para el despligue eficiente de aplicaciones que permitan la automatización y gestión de redes en entornos virtualizados o físicos.

4.2. Objetivos específicos

- Instalar y configurar herramientas como *Docker*, *Kubelet*, *Kubeadm*, *Kubectl* y *Kubernetes* en entornos de pruebas virtuales.
- Implementar y configurar un clúster de *Kubernetes* de alta disponibilidad, incluyendo la configuración de balanceadores de carga para distribuir el tráfico entre los nodos maestros y trabajadores.
- Configurar un servidor *NFS* para proporcionar almacenamiento compartido entre los nodos del clúster de *Kubernetes*, facilitando el acceso a volúmenes persistentes.
- Implementar una solución de red para los pods utilizando *Calico*, permitiendo la comunicación eficiente y segura entre los componentes del clúster.
- Desarrollar una API en *Python* para automatizar procesos de configuración y monitoreo de dispositivos de red, utilizando tecnologías como *FastAPI*, *HTML*, *Netmiko*, *PySNMP* y *Bootstrap*.
- Desplegar una aplicación desarrollada en *Python* en el clúster de *Kubernetes*, utilizando *Helm* para gestionar el despliegue, configuración y mantenimiento de la aplicación.
- Diseñar y configurar una topología de red simulada en *GNS3* para integrar y probar la comunicación entre la aplicación desplegada y dispositivos de red simulados.

- Elaborar un manual detallado de instalación y configuración del clúster de *Kubernetes*, incluyendo las configuraciones de balanceadores de carga, *Calico* y el servidor *NFS*.

CAPÍTULO 5

Alcance

El proyecto busca implementar un clúster de alta disponibilidad de *Kubernetes* para el despliegue de aplicaciones de automatización de redes en entornos virtualizados o físicos. Se pretende crear un entorno local con funciones de red virtualizadas, utilizando un clúster de *Kubernetes* operativamente completo y altamente disponible, como base para el despliegue de aplicaciones que automatizan funciones de red. Este clúster estará compuesto por 9 máquinas virtuales distribuidas en 5 PCs físicas, cada una con *Ubuntu Server 22.04 LTS*, de las cuales 6 serán nodos de *Kubernetes* (3 nodos de control y 3 nodos de trabajo), 2 actuarán como balanceadores de carga (*HAProxy*) y 1 como servidor *NFS* (*Network File System*).

Cada máquina virtual contará con 4 CPUs, 8 GB de RAM y al menos 100 GB de almacenamiento, con excepciones para el nodo de control principal y el servidor *NFS*, los cuales tendrán 15 CPUs, 24 GB de RAM y al menos 1 TB de almacenamiento para asegurar la alta disponibilidad y capacidad necesaria para el clúster de *Kubernetes*. Con todo esto funcionando, se desplegará una aplicación desarrollada en *Python* para la automatización y monitoreo de redes, a fin de validar la funcionalidad del clúster en la gestión y mantenimiento de aplicaciones desplegadas en el mismo.

El proyecto se enfocará en la implementación de un clúster de *Kubernetes* de alta disponibilidad en un entorno de laboratorio, con el objetivo de proporcionar una guía práctica para futuros despliegues de *Kubernetes* en escenarios de mayor escalabilidad. Además, se elaborará una guía de instalación paso a paso del clúster, incluyendo las configuraciones de balanceadores de carga, *Calico* para la red de pods, y el servidor *NFS*, que servirá como referencia clara y detallada para futuros proyectos o trabajos que utilicen *Kubernetes*, facilitando su uso e instalación desde cero. Todo esto busca promover la adopción de *Kubernetes* en la industria de las telecomunicaciones y mejorar la eficiencia en la automatización de redes.

CAPÍTULO 6

Marco teórico

6.1. ONAP

ONAP, que significa *Open Network Automation Platform*, es un proyecto de código abierto liderado por *The Linux Foundation*. Su objetivo principal es ofrecer una plataforma comprensiva para la automatización, orquestación y gestión de servicios de red en tiempo real y bajo políticas, tanto en redes de comunicaciones tradicionales como en redes definidas por *software* (*SDN*) y en entornos de virtualización de funciones de red (*NFV*). *ONAP* se ha convertido en una iniciativa clave en la transformación de las telecomunicaciones hacia arquitecturas más flexibles, automatizadas y basadas en estándares abiertos [8].

Estas características garantizan que *ONAP* provea una configuración fácil y óptima de servicios de redes físicas y virtuales. Asimismo, provee una gestión del ciclo de vida de dichas redes y, en conjunto con objetos de código abierto, permite la creación de un ecosistema para una automatización mucho más rápida que un producto individual [8].

La función principal de *ONAP*, que es diseñar, desplegar y operar servicios, se puede dividir en dos *frameworks* principales: *Design-time framework* y *Run-time framework*.

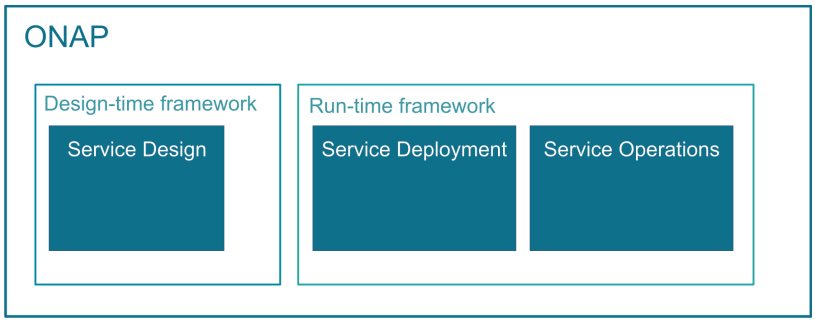


Figura 1: *Frameworks* disponibles en *ONAP* [8].

En el caso del *Design-time framework*, se encuentra el *Service Design*:

- ***Service Design***: Esta función permite especificar los diversos aspectos que se llevan a cabo para la gestión elástica de los servicios a desplegar. Modelado de recursos y relaciones, políticas, aplicaciones, análisis y todos aquellos eventos de control que se llevarán a cabo, son algunos de los aspectos que se pueden especificar dentro del *Service Design*. Este marco de diseño utiliza *TOSCA* como su enfoque principal [8].

En el caso del *Run-time framework*, se encuentran las siguientes funciones:

- ***Service Deployment***: Esta función se encarga de la instanciación automática de servicios, así como de la aplicación de todos los diferentes aspectos especificados con anterioridad sobre el mismo [8].
- ***Service Operations***: Marco analítico que permite un monitoreo del servicio desplegado durante todo su ciclo de vida. En este marco, se garantiza el ajuste al servicio, ya sea por la necesidad de ajustar los recursos del mismo de forma elástica o por la necesidad de cumplir las especificaciones de diseño, análisis y políticas establecidas [8].

6.1.1. Ambientes operativos de ONAP

Al ser una plataforma flexible, modular y altamente configurable, *ONAP* puede operar en diferentes entornos de red [8]. Los siguientes son entornos en los que se ha garantizado la funcionalidad y eficiencia de *ONAP*:

- *Voice Over LTE (VoLTE) (VoLTE)*
- *Customer Premise Equipment* (*CPE*)
- Redes 5G
- *Cross Domain and Cross Layer VPN* (*CCVPN*)
- *Broadband Service* (*BBS*)

6.1.2. Arquitectura de ONAP

ONAP, al ser una plataforma con naturaleza modular y por capas, garantiza la interoperabilidad y simplifica la integración de la plataforma a diferentes entornos. De hecho, *ONAP*, para garantizar una facilidad en el despliegue de servicios y en su propia operabilidad, cuenta con un gran número de modelos de datos (representaciones estructuradas de todos los componentes y servicios de red que definen cómo deben configurarse y gestionarse los mismos) y de APIs REST abiertas e interoperables.

Por ello, *ONAP* ofrece diversas herramientas que trabajan juntas para ofrecer capacidades de automatización y orquestación de red. La arquitectura de *ONAP* se basa en dos principales componentes [9]:

- ***Workforce***: Este componente se refiere a los recursos de la plataforma *ONAP*. Estos incluyen los modelos de información y de diseño, así como las políticas y las herramientas de orquestación necesarias para implementar y gestionar servicios de red.
- ***Middleware***: El *middleware* de *ONAP* proporciona la infraestructura necesaria para conectar y coordinar los diversos componentes de la plataforma. Esto incluye una variedad de herramientas para la gestión de eventos, la comunicación entre componentes, el almacenamiento de datos y la gestión de la configuración.

De hecho, como dice [8]:

> *The ONAP enables service/resource independent capabilities for design, creation and lifecycle management, in accordance with the following foundational principles:*
>
> - *Ability to dynamically introduce full service lifecycle orchestration (design, provisioning and operation) and service API for new services and technologies without the need for new software releases or without affecting operations for the existing services*
> - *Scalability and distribution to support a large number of services and large networks*
> - *Metadata-driven and policy-driven architecture to ensure flexible and automated ways in which capabilities are used and delivered*
> - *The architecture shall enable sourcing best-in-class components*
> - *Common capabilities are 'developed' once and 'used' many times*
> - *Core capabilities shall support many diverse services and infrastructures*

Para cumplir con todos los principios mencionados anteriormente, *ONAP* utiliza una arquitectura funcional que contiene una descripción de los diferentes componentes, así como interfaces que permitan una sencilla integración. Todos estos componentes son modulares y en capas para que se puedan integrar y operar de forma sencilla [8].

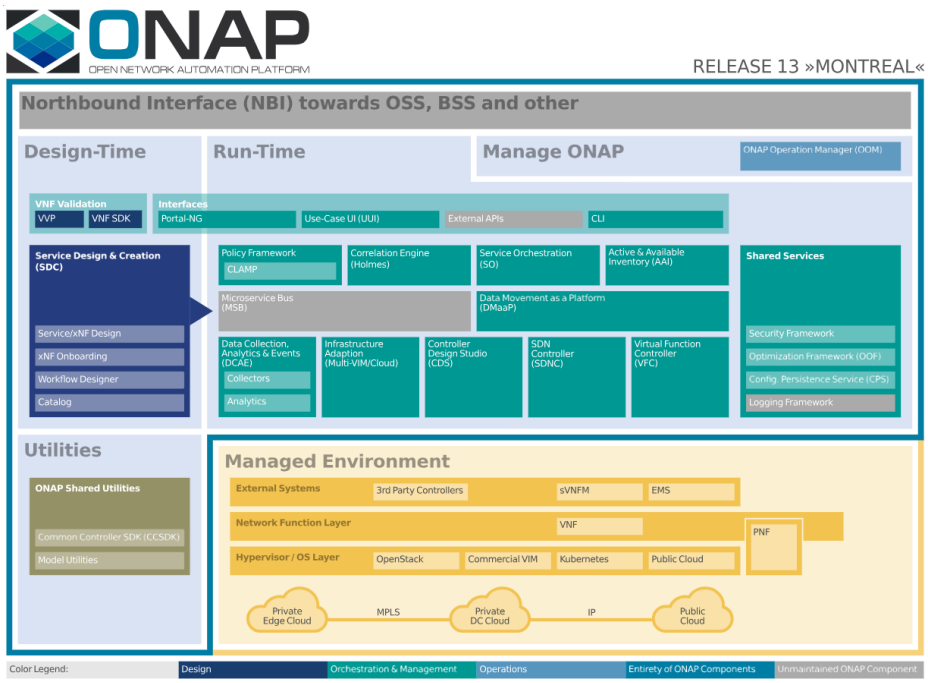


Figura 2: Arquitectura de *ONAP* [8].

Como se puede observar en la Figura 2, la arquitectura de *ONAP* se divide principalmente en 3 ambientes de trabajo: *Design-Time*, *Run-Time* y *Manage ONAP*.

- **Entorno de diseño (*Design-Time Environment*):**
  - ***Service Design and Creation (SDC)*:** Herramientas para definir, simular y certificar servicios y recursos.
  - ***VNF SDK*:** Creación y validación de paquetes *VNF/PNF*.
  - ***VVP (VNF Validation Program)*:** Validación de paquetes *VNF*.
- **Entorno de ejecución (*Run-Time Environment*):**
  - ***Service Orchestrator (SO)*:** Orquestación de servicios y recursos en la red.
  - ***DCAE (Data Collection, Analytics, and Events)*:** Recolección y análisis de datos.
  - ***A&AI (Active and Available Inventory)*:** Gestión de inventario en tiempo real.
  - ***Policy Framework*:** Gestión de políticas para automatizar decisiones de red.
- **Controladores y gestión:**
  - ***SDN-C (SDN Controller)*:** Control de redes definidas por *software*.
  - ***VNFM (VNF Manager)*:** Gestión de funciones de red virtualizadas.
  - ***Multi-Cloud*:** Adaptación e interoperabilidad con múltiples entornos de nube.
- **Herramientas compartidas:**

- ***ONAP Operations Manager (OOM)*:** Gestión del ciclo de vida de los componentes de *ONAP*.
- ***Security Framework*:** Comunicación segura y autenticación.
- ***Logging Framework*:** Generación y manejo de *logs*.

- **Interfaces y *APIs*:**
  - ***External API*:** Proporciona interoperabilidad hacia el norte.
  - ***Microservices Bus* (reemplazado por *Istio*)**: Gestión de microservicios.

6.1.3. Componentes de ONAP

ONAP está compuesto por varios proyectos y componentes interrelacionados que permiten adaptar la plataforma a las necesidades específicas de cada entorno de red que tenga la empresa o el individuo que desea instalar *ONAP*. Todos los componentes disponibles para utilizar son [8]:

- ***AAI***: *Active and Available Inventory*
- ***CCSDK***: *Common Controller Software Development Kit*
- ***CPS***: *Configuration Persistence Service*
- ***DCAE Gen 2***: *Data Collection, Analysis and Events*
- ***DMAAP***: *Data Movement As A Platform*
- ***HOLMES***: *Alarm Correlation and Analysis*
- ***INT***: *Integration*
- ***MOD***: *Modeling*
- ***MSB***: *Microservices Bus*
- ***MULTICLOUD***: *Multicloud Framework*
- ***OOM***: *ONAP Operations Manager*
- ***OOF***: *Optimization Framework*
- ***PF***: *Policy Framework*
- ***SDC***: *Service Design and Creation*
- ***SDN-C***: *SDN Controller*
- ***SO***: *Service Orchestration*
- ***UUI***: *User User Interface*
- ***VFC***: *Virtual Function Controller*

- ***VNFSDK***: *VNF Software Development Kit*

Algunos de los más destacados son [10]:

- ***A&AI***: *Active and Available Inventory* (*A&AI*) proporciona una vista unificada y en tiempo real de todos los recursos de red y de infraestructura disponibles en la red, lo que facilita la gestión y la asignación de recursos.
- ***Policy***: El componente de política en *ONAP* permite definir y aplicar políticas de red para controlar el comportamiento de los servicios y recursos de red de manera dinámica.
- ***SDN Controller***: Este componente proporciona capacidades de control de red definidas por *software* (*SDN*), permitiendo la programación y gestión centralizada de dispositivos de red.
- ***VNF Manager***: El *VNF Manager* es responsable de gestionar las funciones de red virtualizadas (*VNF*) en un entorno de virtualización de funciones de red (*NFV*). Esto incluye la gestión del ciclo de vida de las *VNF*, la configuración y la monitorización.
- ***SO***: *Service Orchestration* (*SO*) es responsable de orquestar y gestionar servicios de red complejos, incluyendo la coordinación de múltiples *VNF* y recursos de red.
- ***Multi-VIM***: *Multi-VIM* proporciona una interfaz unificada para gestionar múltiples infraestructuras de virtualización (*VIM*), permitiendo a *ONAP* gestionar recursos de red en entornos de virtualización heterogéneos.
- ***Common Controller Software Development Kit (CCSDK)***: *CCSDK* proporciona una variedad de herramientas y bibliotecas para el desarrollo de controladores de red y aplicaciones de red [11].
    - ***CDS***: El *Controlador de Datos Común* (*CDS*) es un componente central dentro del *CCSDK* que proporciona una capa de abstracción para acceder y manipular datos en los sistemas de almacenamiento subyacentes. Sus principales características incluyen:
        - ***Abstracción de datos***: *CDS* permite a las aplicaciones de *ONAP* acceder a datos de manera uniforme, independientemente de la fuente de datos subyacente.
        - ***Gestión de transacciones***: *CDS* ofrece capacidades de gestión de transacciones para garantizar la integridad y la consistencia de los datos manipulados por las aplicaciones.
        - ***Escalabilidad y rendimiento***: El diseño modular y escalable de *CDS* permite manejar grandes volúmenes de datos y proporcionar un rendimiento óptimo incluso en entornos de alta carga.
    - ***Aplicaciones (apps)***: Las aplicaciones en *ONAP* se refieren a los módulos de *software* que implementan funcionalidades específicas dentro de la plataforma. Estas aplicaciones pueden utilizar los servicios proporcionados por el *CCSDK*, como el *CDS*, para acceder a datos y servicios comunes en *ONAP*. Algunas de las aplicaciones más importantes dentro del *CCSDK* incluyen:

- ***Controladores de servicio***: Estos son componentes responsables de la orquestación y la gestión de servicios de red, como el *Controlador de Servicios de Red* (*SDC*) y el *Controlador de Funciones Virtuales* (*VF-C*).
- ***Controladores de infraestructura***: Estos controladores gestionan recursos de infraestructura física y virtual, como el *Controlador de Recursos* (*RES*) y el *Controlador de Recursos Virtuales* (*VFC*).
- ***Aplicaciones de gestión***: Estas aplicaciones proporcionan capacidades de gestión y operación para *ONAP*, como el *Portal de Usuario* (*UUI*) y el *Controlador de Políticas* (*Policy*).

- ***Open Radio Access Network (ORAN)***: *ORAN* es una iniciativa dentro de *ONAP* que se centra en la automatización y la orquestación de redes de acceso radioeléctrico (*RAN*) en entornos de redes móviles. *ORAN* utiliza componentes y servicios proporcionados por el *CCSDK* para implementar funciones de control y gestión de red en la infraestructura *RAN* [12].

  Algunos de los componentes de *ORAN* que pueden utilizar el *CCSDK* incluyen:
  - ***Controladores de RAN***: Estos son componentes responsables de la gestión y la optimización de la infraestructura *RAN*, como el *Controlador de RAN Abierto* (*O-CU*) y el *Controlador de Distribución Abierta* (*O-DU*).
  - ***Aplicaciones de automatización***: *ORAN* utiliza aplicaciones de automatización basadas en el *CCSDK* para implementar políticas y procedimientos de automatización en la infraestructura *RAN*, como la *Gestión Dinámica de Recursos* (*DRM*) y la *Optimización Automática de Redes* (*ANOP*).

6.2. Microservicios

En la actualidad, no todas las aplicaciones se encuentran funcionando en la nube. Las aplicaciones monolíticas llegan a ser un problema para las tecnologías futuras, pues al ser piezas de *software* que funcionan sobre una sola máquina o servidor, obligan a estos mismos a tener buenas capacidades de memoria, de cómputo, almacenamiento y capacidades de red. A medida que se sigue trabajando con aplicaciones monolíticas, estas continúan incrementando en tamaño y se vuelven poco eficientes. Para poder solucionar estos problemas y que nuestras aplicaciones se puedan adaptar más a las tecnologías futuras, se comenzó a crear una nueva arquitectura de *software*: *Microservicios*.

Si pensamos en las aplicaciones monolíticas como una roca gigante, movilizarlas sería un trabajo muy complicado. Una forma de poder transportarlas fácilmente de un lugar a otro sería romper dicha roca en pequeños trozos. Esto es lo que hacen los *microservicios*: dividen el *software* en pequeños componentes, cada uno responsable de una tarea específica. Estos componentes, al trabajar juntos, logran cumplir con la función total que la aplicación monolítica ofrecía [13].

Como menciona [13]:

> "*The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an* ***HTTP*** *resource* ***API****.*"

De hecho, los *microservicios*, a pesar de ser una arquitectura nueva de *software*, comparten principios con arquitecturas como la *Event-driven Architecture* y *Service-Oriented Architecture (SOA)*. Que las aplicaciones complejas se encuentren compuestas por procesos pequeños e independientes, los cuales se comunican entre sí a través de *Application Programming Interfaces (API)*, es el principal principio que dichas tres arquitecturas de *software* comparten.

6.3. Virtualización

A medida que las nuevas tecnologías buscan orientarse hacia la computación en la nube o *Cloud Computing*, el concepto de virtualización adquiere una importancia significativa. El concepto principal de la virtualización se basa en correr una instancia virtual de un sistema de cómputo sobre una capa abstraída del *hardware* físico [14].

La virtualización, al realizar la abstracción de los recursos físicos del *hardware*, crea versiones virtuales de estos, conocidas como máquinas virtuales (*VMs*). Este proceso se logra mediante un hipervisor, que es un *software* que permite ejecutar múltiples sistemas operativos sobre un solo *hardware* físico, gestionando los recursos de manera eficiente y segura. La virtualización es fundamental en la arquitectura de *TI* moderna, ya que permite una mayor utilización de los recursos, mejor aislamiento de las aplicaciones y mayor flexibilidad en la gestión de la infraestructura [15].

6.3.1. Tipos de virtualización

- **Virtualización de hardware**: Permite ejecutar múltiples sistemas operativos en un solo servidor físico mediante hipervisores como *VMware ESXi*, *Microsoft Hyper-V* o *KVM*. Esto optimiza la utilización del *hardware* y facilita la gestión centralizada [16].
- **Virtualización de red**: Abstrae los servicios de red de la infraestructura de red física, facilitando la creación y gestión de redes virtuales. Tecnologías como *VMware NSX* y *Cisco ACI* son ejemplos prominentes [17].
- **Virtualización de almacenamiento**: Agrupa el almacenamiento físico en un único recurso de almacenamiento virtualizado, mejorando la gestión y la flexibilidad del almacenamiento [18].
- **Virtualización de escritorios**: Proporciona escritorios virtuales a los usuarios finales, permitiendo el acceso remoto y centralizado. Soluciones como *VMware Horizon* y *Citrix Virtual Apps and Desktops* son ejemplos de esta tecnología [19].

6.3.2. Contenedores

Un contenedor es un método de virtualización que está centrado principalmente en aplicaciones, lo cual permite entregar las mismas con rendimiento alto y eficiente. De hecho, los contenedores son especialmente buenos para la entrega de *microservicios*. El modelo de realizar pequeños ambientes aislados, en los cuales se encuentran corriendo aplicaciones,

es especialmente bueno para los *microservicios*, ya que cada una de esas aplicaciones no interfiere con la otra.

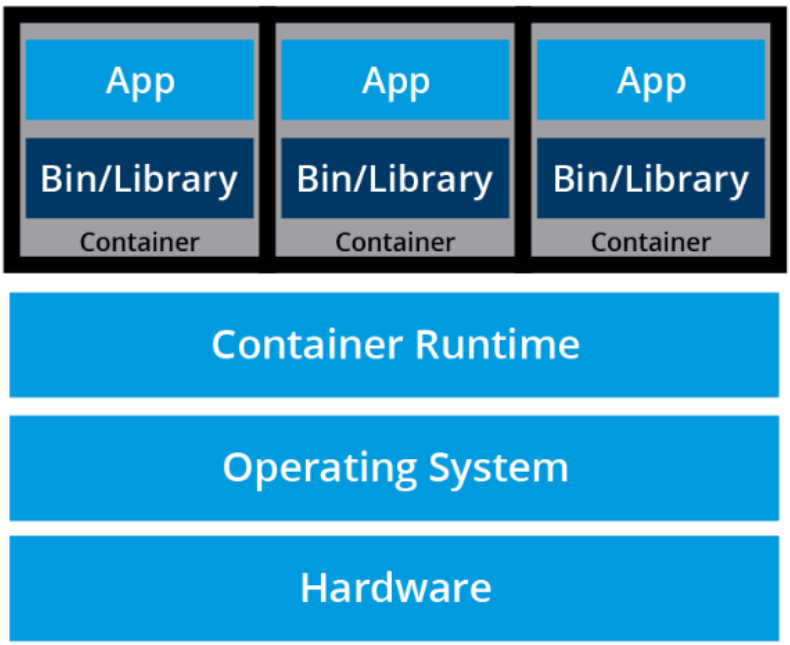


Figura 3: Esquema de contenedores [14] [20].

A pesar de que un contenedor encapsula *microservicios* y todas las dependencias del mismo, estos no corren el *microservicio* de forma directa. Los contenedores corren imágenes de contenedor o *Container Images*.

6.3.2.1. Container images

Una imagen de contenedor agrupa la aplicación junto con su entorno de ejecución, bibliotecas y dependencias. Esto representa la fuente de un contenedor desplegado y ayuda a tener un entorno ejecutable completamente aislado para la aplicación. Estas imágenes se pueden desplegar a través de diversas plataformas como estaciones de trabajo, máquinas virtuales, nubes públicas, entre otros.

6.3.2.2. Docker

Docker es una plataforma de contenedorización que permite a los desarrolladores empaquetar aplicaciones y todas sus dependencias en contenedores ligeros y portátiles. Los contenedores de *Docker* proporcionan un entorno aislado y consistente, lo que asegura que las aplicaciones se ejecuten de manera uniforme en cualquier entorno, desde la máquina del desarrollador hasta los entornos de producción en la nube [14].

6.3.2.2.1. Características de Docker

- **Portabilidad**: Los contenedores de *Docker* son portátiles y se pueden ejecutar en cualquier entorno que tenga *Docker* instalado, lo que facilita la implementación y la migración de aplicaciones [21].
- **Aislamiento**: *Docker* utiliza tecnologías de virtualización a nivel de sistema operativo para proporcionar un entorno aislado para las aplicaciones, lo que garantiza que no haya conflictos entre las dependencias de las aplicaciones [22].
- **Eficiencia**: Los contenedores de *Docker* son ligeros y comparten el *Kernel* del sistema operativo subyacente, lo que reduce el consumo de recursos y mejora la eficiencia.
- **Escalabilidad**: *Docker* facilita la escalabilidad de las aplicaciones mediante la creación y el despliegue de múltiples contenedores de manera rápida y sencilla [23].

6.3.2.2.2. Componentes de Docker

- ***Docker Engine***: Es el componente principal de *Docker* que permite la creación, ejecución y gestión de contenedores. Incluye el demonio de *Docker* (*dockerd*) y la interfaz de línea de comandos de *Docker* (*docker*).
- ***Docker Compose***: Es una herramienta que permite definir y gestionar aplicaciones multi-contenedor con un archivo *YAML File*. Facilita la configuración y la ejecución de aplicaciones complejas con múltiples servicios.
- ***Docker Swarm***: Es una herramienta de orquestación de contenedores que permite gestionar y escalar múltiples contenedores de *Docker* en un clúster. Proporciona capacidades de alta disponibilidad y balanceo de carga.
- ***Docker Registry***: Es un repositorio de imágenes de *Docker* que permite almacenar y distribuir imágenes de contenedores. *Docker Hub* es el registro público de *Docker*, pero también se pueden implementar registros privados.

6.4. Orquestación de contenedores

Al momento en que buscamos lanzar una aplicación a través de contenedores, esta misma debe pasar por diversos entornos de desarrollo para comprobar su funcionalidad y así, finalmente, lanzarla hacia el usuario final. Los tres principales entornos de desarrollo por los que debe pasar una aplicación son:

1. **Entorno de desarrollo**: Entorno donde los desarrolladores crean y realizan pruebas sobre nuevas funcionalidades y algunos cambios realizados en el *software*. Dentro de este se busca tener herramientas de desarrollo, múltiples versiones de prueba de la aplicación y configuraciones que garanticen realizar pruebas y verificaciones rápidas para dicha aplicación.

2. **Entorno de garantía de calidad (*Quality Assurance, QA*)**: Entorno donde las aplicaciones son sometidas a pruebas exhaustivas para comprobar y asegurarse de que la aplicación cumpla con los estándares de calidad previo a ser lanzada al entorno de producción. Este entorno implica una simulación del entorno de producción, en el cual se pueden realizar pruebas de carga, integración, regresión, etc. Todo esto se realiza para comprobar una correcta funcionalidad dentro de un entorno altamente similar al final.
3. **Entorno de producción**: Entorno donde la aplicación se encuentra disponible para todos los usuarios finales. Este entorno, al ser el más importante, es necesario que sea altamente estable, seguro, óptimo y con configuraciones redundantes para garantizar la alta disponibilidad y una buena recuperación ante posibles problemas.

Al momento de utilizar el entorno de desarrollo, la aplicación puede mantenerse corriendo sobre un mismo *Host*, lo cual no causa problemas a futuro. Ahora, cuando se busca pasar a entornos de Garantía de Calidad y de Producción, no se vuelve una buena opción correr todo dentro de un mismo *host*, ya que nuestra aplicación debe cumplir con especificaciones como:

- Tolerancia a fallos
- Escalabilidad bajo demanda (Ajuste automático de recursos computacionales bajo necesidades de la carga de trabajo).
- Uso óptimo de recursos
- Descubrimiento automático para descubrirse y comunicarse entre sí
- Accesibilidad desde el mundo exterior
- Actualizaciones/Reversiones sin interrupciones

Para cumplir con todas estas especificaciones y evitar el problema de realizarlo todo de forma manual, es que entran los orquestadores de contenedores.

Los orquestadores de contenedores son herramientas que agrupan sistemas en *Cluster*, donde se automatiza el despliegue y manejo de los contenedores; todo para cumplir con las especificaciones mencionadas anteriormente. La ventaja de tener estos *Clusters* es que los sistemas distribuidos en ellos tienen un mejor rendimiento, alta eficiencia, distribución de carga optimizada y una latencia reducida. En sistemas donde se tiene un número reducido de contenedores, tal vez no implique una ventaja el utilizar los orquestadores; sin embargo, donde se denota una alta ventaja de los mismos es al momento de aplicarlos en sistemas donde corren cientos o miles de contenedores.

Otra de las grandes ventajas que ofrecen los orquestadores de contenedores es que pueden ser desplegados en la infraestructura que más nos beneficie. Esto implica que se puede instalar en nubes públicas o híbridas, máquinas virtuales e infraestructura física.

6.4.1. Kubernetes

Kubernetes (*K8s*) es un sistema *open-source* para la automatización del despliegue, escalabilidad y manejo de aplicaciones en contenedores. Este fue desarrollado por *Google* y actualmente, forma parte de la *Cloud Native Computing Foundation* (*CNCF*). Este es uno de los orquestadores de contenedores con más demanda actualmente [24].

6.4.1.1. Componentes de Kubernetes

- **Nodo**: Un nodo es una máquina física o virtual que ejecuta los contenedores de la aplicación. Cada nodo tiene un agente llamado *kubelet* que se comunica con el clúster de *Kubernetes* y gestiona los contenedores en el nodo [25].
- **Pod**: Un *pod* es la unidad básica de despliegue en *Kubernetes* y puede contener uno o más contenedores. Los contenedores en un *pod* comparten recursos y se ejecutan en el mismo espacio de red y almacenamiento [26].
- ***Control Plane***: El plano de control de *Kubernetes* es el componente central que gestiona el clúster y coordina las operaciones de los nodos y los *pods*. Incluye varios componentes como el *API Server*, el *Controller Manager*, el *Scheduler* y *etcd*.
- ***API Server***: El servidor de *API* de *Kubernetes* es el punto de entrada para las operaciones de administración del clúster y proporciona una interfaz *RESTful* para interactuar con el clúster.
- ***Controller Manager***: El administrador de controladores de *Kubernetes* es responsable de la ejecución de controladores que regulan el estado del clúster, como el controlador de replicación y el controlador de estado.
- ***Scheduler***: El programador de *Kubernetes* es responsable de asignar *pods* a nodos en función de los recursos disponibles y los requisitos de la aplicación.
- ***etcd***: *etcd* es un almacén de datos distribuido que se utiliza para almacenar la configuración del clúster y el estado del clúster de *Kubernetes*.
- **Nodos worker**: Los nodos *worker* son los nodos que ejecutan los *pods* de la aplicación y proporcionan los recursos computacionales y de almacenamiento necesarios para ejecutar los contenedores [27].
  - ***Kubelet***: El *kubelet* es el agente que se ejecuta en cada nodo y es responsable de gestionar los *pods* y los contenedores en el nodo.
  - ***Kube-proxy***: El *kube-proxy* es un proxy de red que se ejecuta en cada nodo y es responsable de la gestión del tráfico de red para los *pods* en el nodo.
  - ***Container Runtime***: El tiempo de ejecución de contenedores es el *software* que se utiliza para ejecutar los contenedores en los nodos. *Docker* y *containerd* son ejemplos de tiempos de ejecución de contenedores compatibles con *Kubernetes*.

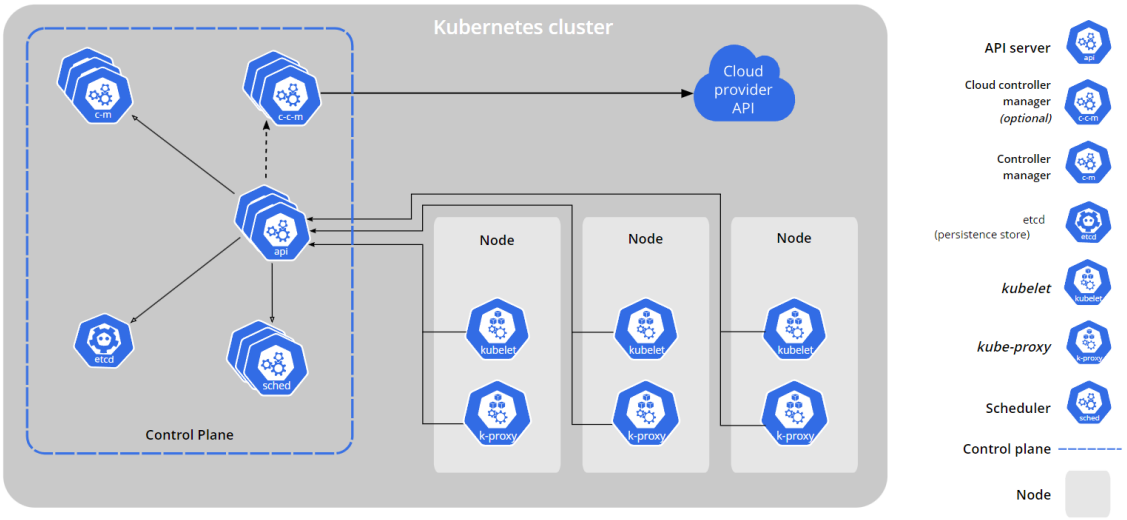


Figura 4: Componentes de Kubernetes y su rol dentro de un clúster [28].

6.4.1.2. Cluster de Kubernetes

Kubernetes, al ser un orquestador de contenedores, tiene la capacidad de generar *Clusters* que permiten agrupar diversos sistemas o nodos y, de esa forma, optimizar el rendimiento de una aplicación. A muy alto nivel, *Kubernetes* ofrece dos servidores al momento de levantar un *cluster*: *Master Node* y *Worker Nodes*.

El *Control Plane* es responsable de las decisiones globales sobre el *cluster*, como la programación de *pods* y la respuesta a eventos. Este plano incluye varios componentes críticos, como el *kube-apiserver*, que actúa como la interfaz principal de la *API* de *Kubernetes*, *etcd*, un almacén de valores clave consistente y altamente disponible, y el *kube-scheduler*, encargado de asignar los *pods* a los nodos [28], [29].

Al configurar un *cluster* de *Kubernetes*, es posible optar por una configuración de alta disponibilidad (*HA*), distribuyendo los nodos del *Control Plane* en múltiples máquinas. Esto asegura que el *cluster* continúe funcionando incluso si algunos nodos fallan. Hay dos enfoques principales para *HA*: con nodos del plano de control apilados y con un *etcd* externo, siendo este último más robusto y a gran escala [30]. De hecho, para despliegues a gran escala, es crucial considerar aspectos como la escalabilidad, la capacidad de los nodos y la distribución de la carga. *Kubernetes* está diseñado para soportar hasta 5000 nodos y 150,000 *pods*, lo que requiere una planificación cuidadosa para mantener un rendimiento óptimo [31].

A nivel general, existen muchas maneras de implementar un *cluster* de *Kubernetes*, pero existe una herramienta que permite una fácil implementación del mismo llamada *Kubeadm.*Además de configurar un *cluster*, *kubeadm* maneja funciones adicionales como la actualización y el manejo del ciclo de vida del *cluster*. Esta herramienta es compatible tanto con despliegues en la nube como en entornos locales [32].

6.4.1.3. Automatización y escalabilidad

Kubernetes permite automatizar y escalar aplicaciones en contenedores mediante un enfoque declarativo. Los usuarios definen el estado deseado de las aplicaciones, y Kubernetes ajusta el despliegue para alcanzar ese estado [33]. Esto incluye la capacidad de realizar actualizaciones continuas y revertir cambios automáticamente, lo que reduce el tiempo de inactividad.

Además, Kubernetes soporta escalado automático (*autoscaling*) basado en métricas de rendimiento, permitiendo que las aplicaciones aumenten o disminuyan los recursos según la demanda del tráfico [34].

6.4.1.4. Integración de APIs y automatización

Las APIs en Kubernetes facilitan la integración con herramientas de integración y entrega continua (*CI/CD*), permitiendo automatizar todo el ciclo de vida de las aplicaciones. A través de la API de Kubernetes, se pueden configurar y gestionar aplicaciones a gran escala, desde la configuración inicial hasta la monitorización y escalado. Esta API permite la implementación de estrategias de automatización que garantizan que las aplicaciones se mantengan consistentes, eficientes y escalables, optimizando recursos y facilitando la adaptabilidad a los cambios en la demanda de servicios [35].

6.5. Helm

Helm es un gestor de paquetes para *Kubernetes* que facilita la definición, instalación y actualización de aplicaciones sobre *Kubernetes*. *Helm* utiliza un formato de paquetes llamado *charts*, que son colecciones de archivos que describen los recursos de *Kubernetes* necesarios para desplegar una aplicación [36].

6.5.1. Componentes de Helm

- ***Helm CLI***: La interfaz de línea de comandos de *Helm* permite a los usuarios buscar, instalar y gestionar *charts* de *Kubernetes*. Los comandos de *Helm* incluyen *helm install*, *helm upgrade*, *helm rollback*, *helm list*, entre otros.
- ***Helm Charts***: Los *charts* de *Helm* son paquetes que contienen los recursos de *Kubernetes* necesarios para desplegar una aplicación. Cada *chart* contiene un archivo *Chart.yaml* que describe el *chart* y una carpeta *templates* que contiene los recursos de *Kubernetes*.
- ***Helm Repositories***: Los repositorios de *Helm* son almacenes de *charts* que permiten a los usuarios buscar y descargar *charts* de *Kubernetes*. *Helm* proporciona un repositorio público llamado *Helm Hub*, pero también se pueden configurar repositorios privados.

6.6. Protocolo VRRP

El *Virtual Router Redundancy Protocol* (VRRP) es un protocolo de red que es utilizado para aumentar la disponibilidad de las rutas en redes, permitiendo que múltiples routers en una red local (LAN) trabajen juntos para presentarse como un único router virtual a los dispositivos finales. Esto se logra mediante la asignación dinámica de la responsabilidad de un router virtual a uno de los routers físicos en la LAN, permitiendo la continuidad del servicio en caso de fallos de hardware o software en uno de los routers [37].

6.6.1. Funcionamiento de VRRP

VRRP opera mediante la creación de un router virtual compuesto por un identificador de router virtual (VRID) y una o más direcciones IP asociadas. Los routers físicos que participan en VRRP se comunican entre sí para determinar cuál de ellos tomará el rol de *Master* y cuáles actuarán como *Backup*. El router *Master* reenvía todo el tráfico que es enviado a la o las direcciones IP del router virtual hacia, mientras que los routers *Backup* monitorean el estado del *Master* y están preparados para asumir su rol en caso de falla [38].

El proceso de elección del *Master* se basa en una prioridad configurada en cada router que participa del protocolo. El router con la prioridad más alta se convierte en el *Master*. Si dos routers tienen la misma prioridad, se utiliza la dirección IP más alta como criterio de desempate. Los routers se comunican entre sí para enviar mensajes periódicos que indican el estado de los mismos; si un router *Backup* no recibe anuncios del *Master* dentro de un intervalo de tiempo específico, asume que el *Master* ha fallado y toma su lugar [39].

6.6.2. Aspectos clave de VRRP

- **Redundancia y alta disponibilidad:** VRRP proporciona una ruta de red altamente disponible sin necesidad de configurar protocolos de enrutamiento dinámico o descubrimiento de routers en cada dispositivo final [37].
- **Compatibilidad con IPv4 e IPv6:** Las versiones más recientes de VRRP, como la versión 3, ofrecen soporte tanto para IPv4 como para IPv6, permitiendo su implementación en redes modernas que utilizan ambos protocolos [39].
- **Operación eficiente:** VRRP está diseñado para operar de manera eficiente en redes de área local, minimizando el tráfico de control y proporcionando una conmutación por error rápida en caso de fallos del router [38].
- **Configuración de prioridades:** La capacidad de asignar prioridades a los routers permite a los administradores de red controlar cuál router debe actuar como *Master*, facilitando la gestión y el equilibrio de carga en la red [37].

CAPÍTULO 7

Guía de instalación para un clúster de *Kubernetes* de alta disponibilidad

Un clúster de *Kubernetes* de alta disponibilidad es esencial para garantizar la estabilidad y confiabilidad de las aplicaciones desplegadas en un entorno de producción. La alta disponibilidad se logra a través de la redundancia de los nodos del clúster, lo que permite que las aplicaciones continúen funcionando incluso si uno o más nodos fallan. En esta guía, se proporciona un paso a paso detallado para instalar y configurar un clúster de *Kubernetes* de alta disponibilidad en un entorno local.

7.1. Instalación del clúster de *Kubernetes*

Previo a realizar toda la instalación del clúster de *Kubernetes*, es necesario tener los siguientes componentes de *software* instalados en el sistema. Nuestro clúster depende de todos ellos, por lo que es necesario garantizar que estén instalados y configurados correctamente. Los componentes necesarios son:

- Máquinas virtuales con Ubuntu Server 22.04 LTS
- kubectl
- kubeadm
- kubelet
- Docker

7.1.1. Estructura del clúster de *Kubernetes*

El cluster de *Kubernetes*, se realizará utilizando 5 PCs físicas que, cada una de ellas, contendrá 2 máquinas virtuales con *Ubuntu Server* 22.04 LTS como sistema operativo, exceptuando la máquina 9 que solo contendrá 1 máquina virtual. Se realizará de esta manera para que se tenga un cluster de *Kubernetes* de 6 nodos, 2 balanceadores de carga y un *Network File System* (NFS) para todos los nodos del cluster. Son 9 máquinas en total para tener:

- 2 máquinas funcionando como balanceadores de carga para los nodos de control
- 3 nodos de control
- 3 nodos de trabajo o worker nodes
- 1 nodo NFS (*Network File System*)

7.2. Creación del clúster de *Kubernetes*

Para elaborar el clúster de *Kubernetes*, se seguirán los siguientes pasos:

7.2.1. Paso 1: Instalación de *Ubuntu server* 22.04 LTS en máquinas virtuales

Para comenzar a desarrollar el cluster de , va a ser necesario configurar el entorno o las máquinas virtuales que sirvan como los nodos de nuestros cluster. En este caso, se utilizarán 6 de las 9 máquinas virtuales disponibles, para que creemos un clúster de 6 nodos. Cada máquinas virtual será realizada con *Ubuntu Server* 22.04 LTS.

Para cada máquina virtual realizada (o en este caso, para cada nodo), será necesario realizar particiones de la RAM y del disco duro de la computadora física. Esta partición, para cada VM, se realizará de la siguiente forma:

- 4 CPUs
- 8 GB RAM
- Por lo menos 100 GB de almacenamiento

Esto nos garantiza realizar un clúster de *kubernetes* con alta disponibilidad y suficiente capacidad para realizar pruebas sencillas con las aplicaciones a desplegar.

Ahora, para las máquinas que contendrán al nodo de control 1 y al servidor NFS, se les ampliarán los recursos, ya que, en el caso del nodo de control 1, se utilizará para desplegar cualquier aplicación necesaria, y, el NFS, se utilizará como fuente de almacenamiento extra para cada nodo. Para estas dos máquinas, se utilizarán recursos de la siguiente manera:

- 15 CPUs
- 24 GB RAM
- Por lo menos 1 TB de disco duro de almacenamiento.

7.2.1.1. Máquinas virtuales

- Particiones de la RAM

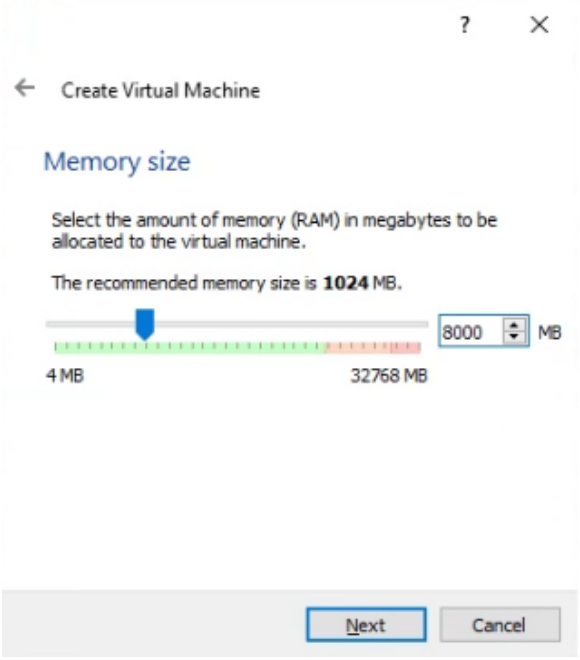


Figura 5: Configuración de *Ubuntu Server* 22.04 LTS

- Disco Duro

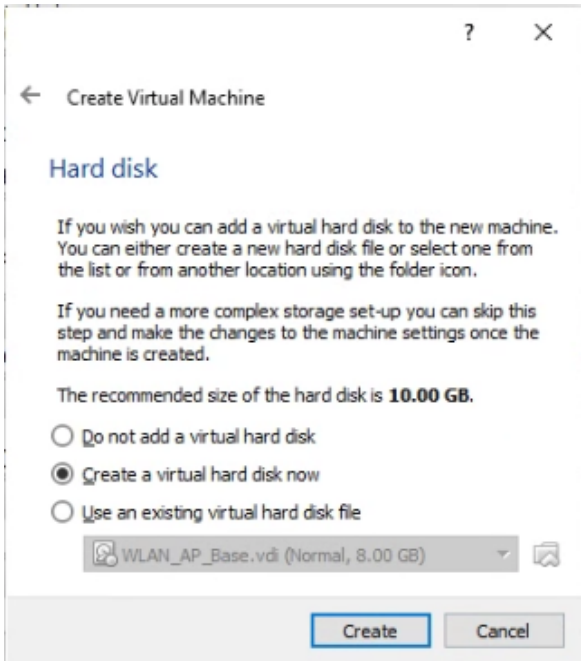


Figura 6: Configuración de *Ubuntu Server* 22.04 LTS

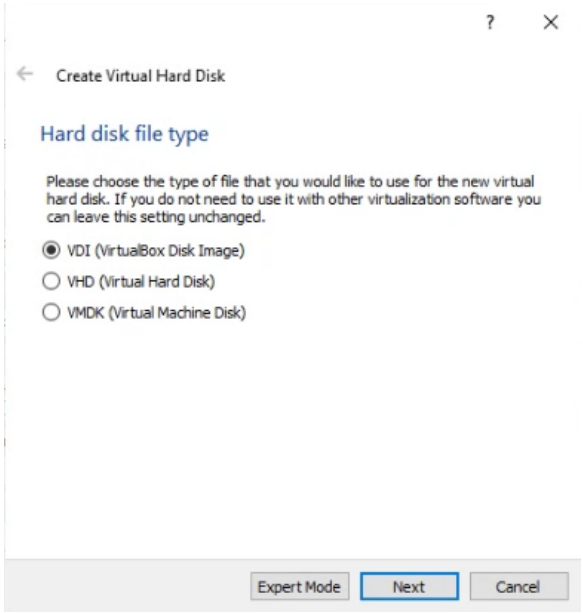


Figura 7: Configuración de *Ubuntu Server* 22.04 LTS

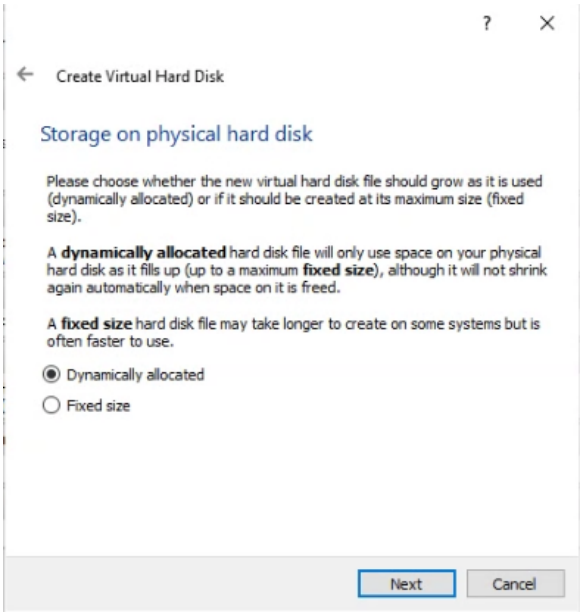


Figura 8: Configuración de *Ubuntu Server* 22.04 LTS

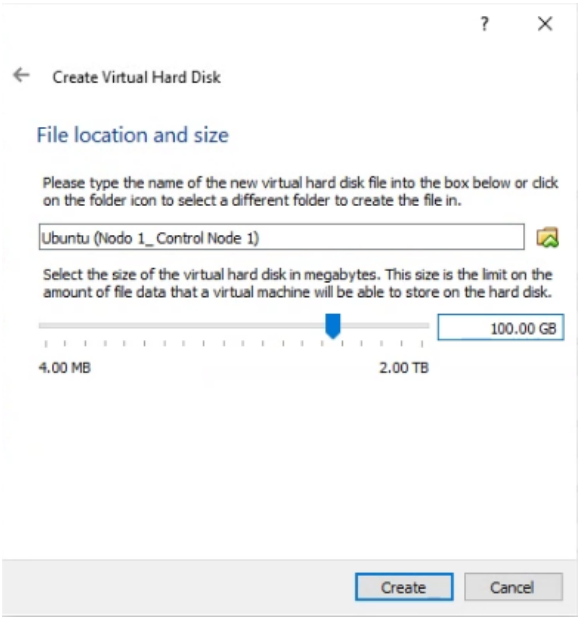


Figura 9: Configuración de *Ubuntu Server* 22.04 LTS

- CPUs en cada máquina virtual

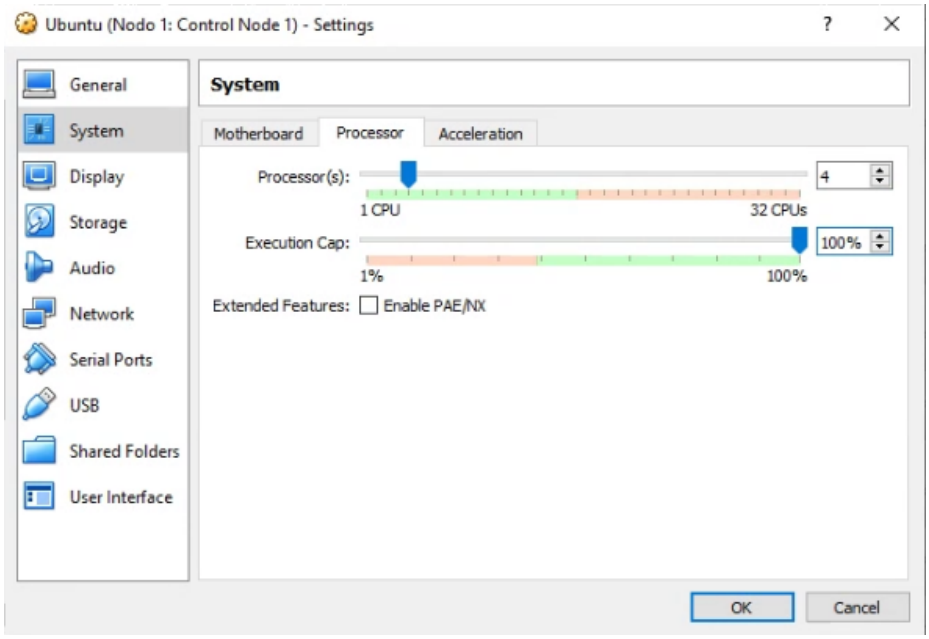


Figura 10: CPUs para la máquina virtual

- Una vez inicializada la máquina virtual, seleccionar la imagen .iso para la instalación

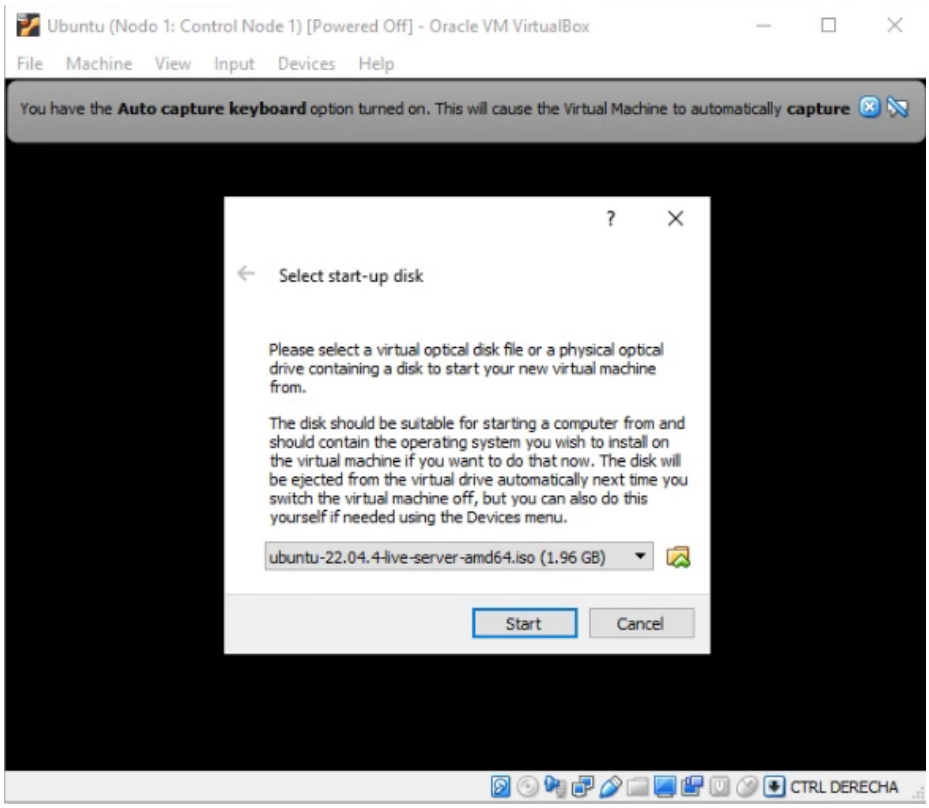


Figura 11: Selección de la imagen .iso para la instalación

- Iniciar la instalación de *Ubuntu Server* 22.04 LTS

- Seleccionar lenguaje: Inglés
- Inicializar sin actualizar

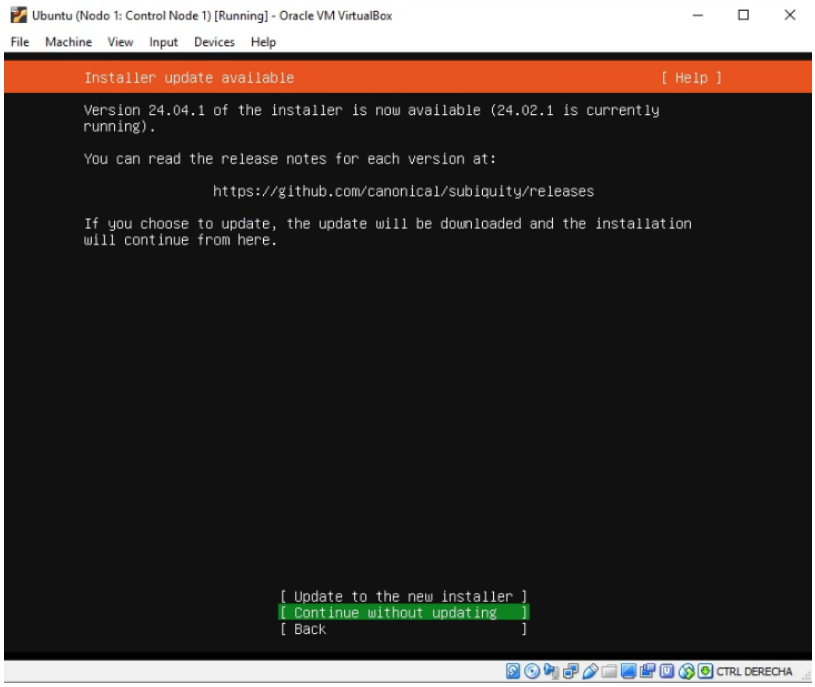


Figura 12: Inicialización sin actualizar

- Lenguaje del teclado

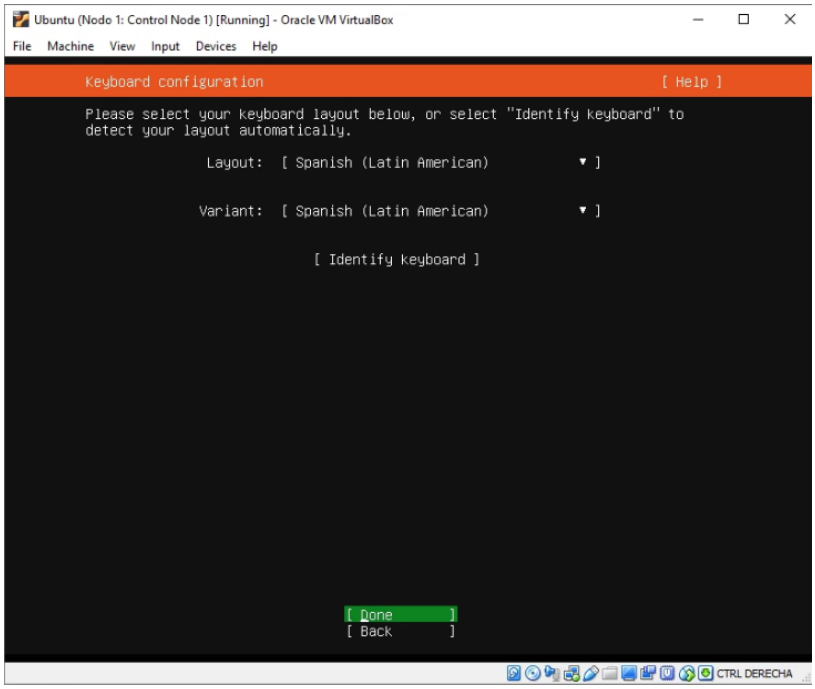


Figura 13: Configurar el lenguaje del teclado

- Seleccionar *Ubuntu Server*

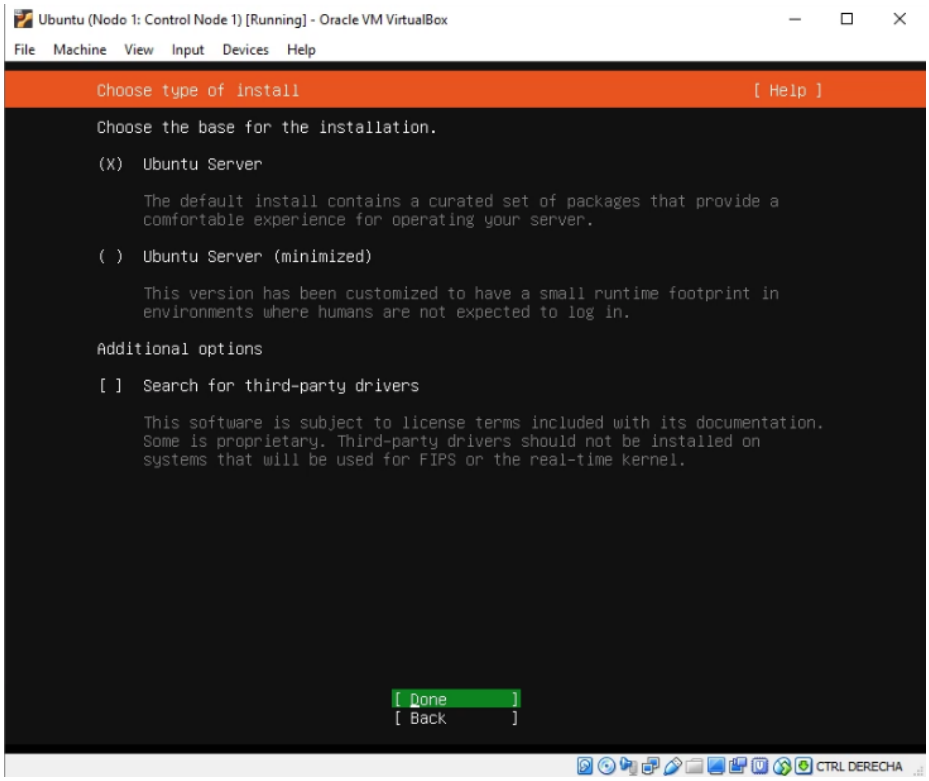


Figura 14: Selección de *Ubuntu Server*

- Previo a ver la sección de red, es necesario añadir el adaptador 1 de la máquina virtual, en *bridged adapter*. Luego, se puede proseguir con la instalación.

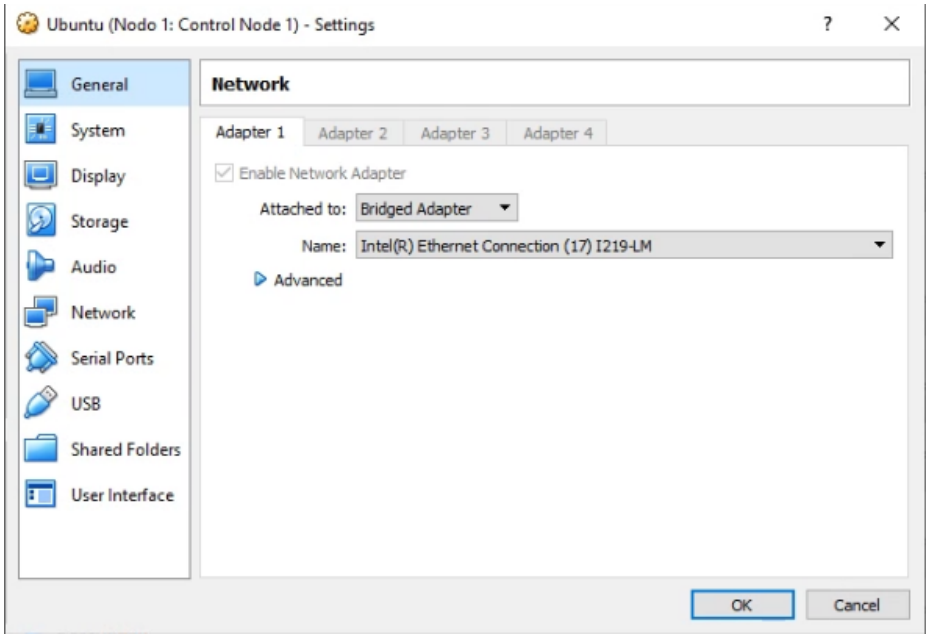


Figura 15: Configuración de adaptador de red

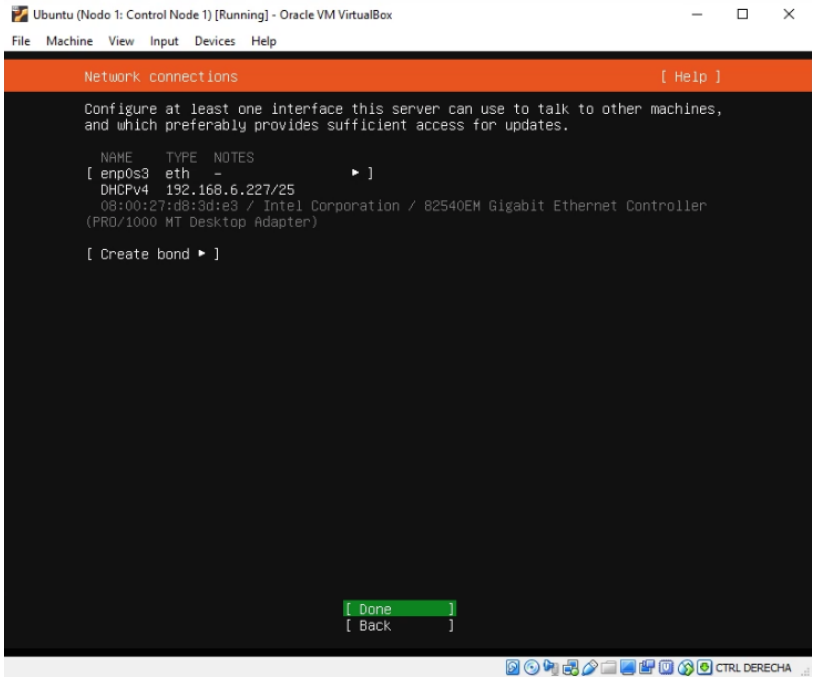


Figura 16: Configuración de adaptador de red

- Configurar *proxy*

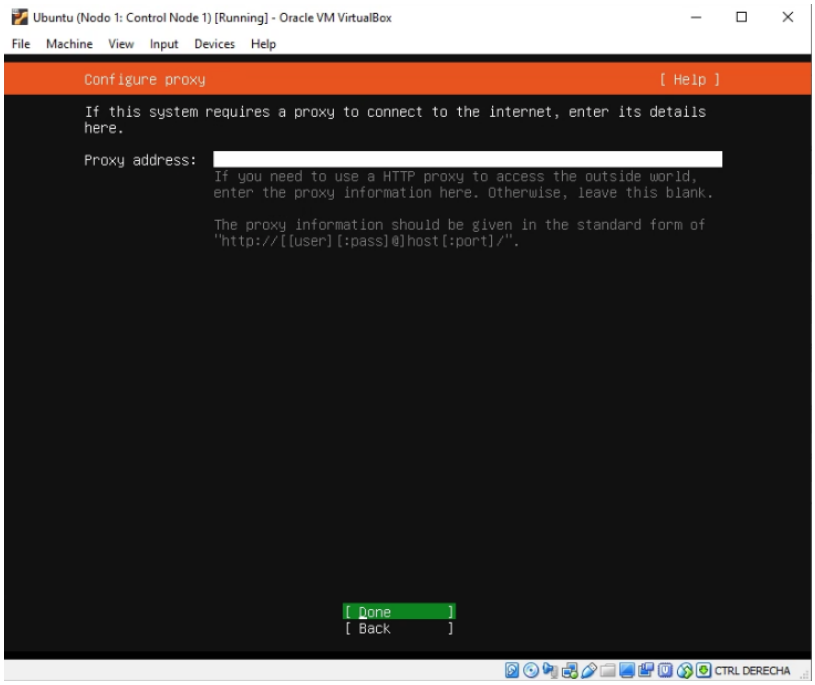


Figura 17: Configuración de *proxy*

- *Ubuntu Archive Mirror*

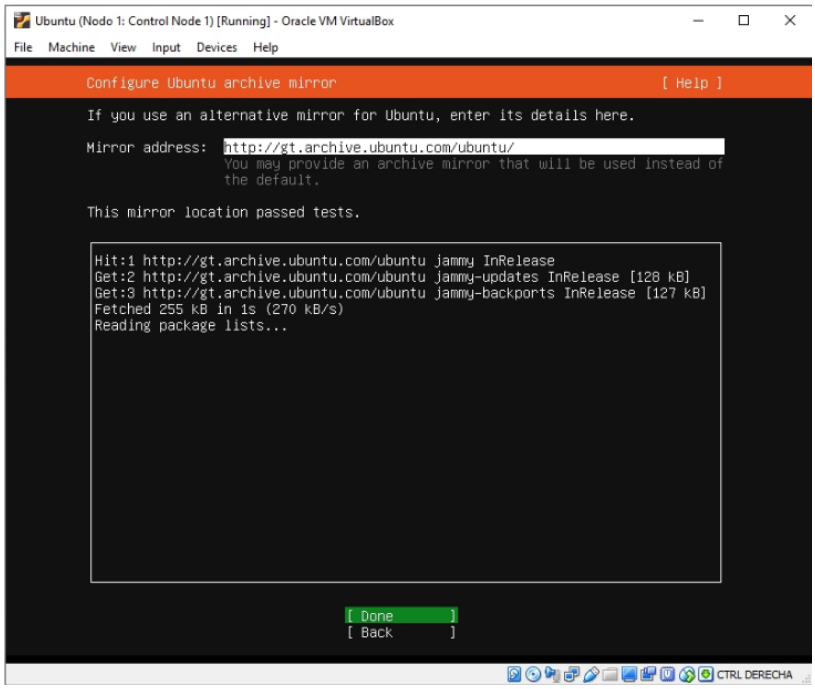


Figura 18: Configuración de *Ubuntu Archive Mirror*

- Uso del disco virtual

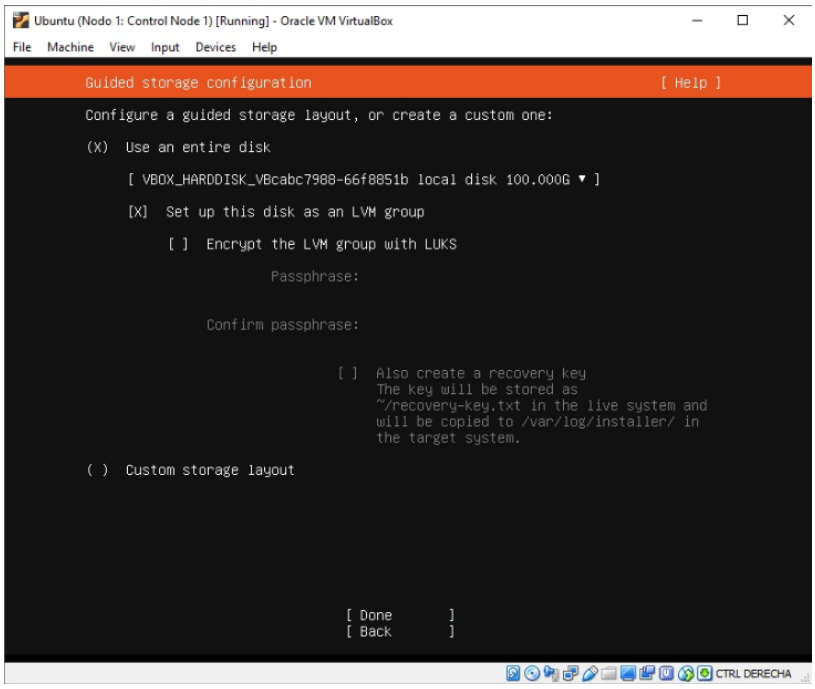


Figura 19: Configuración del disco de la máquina virtual

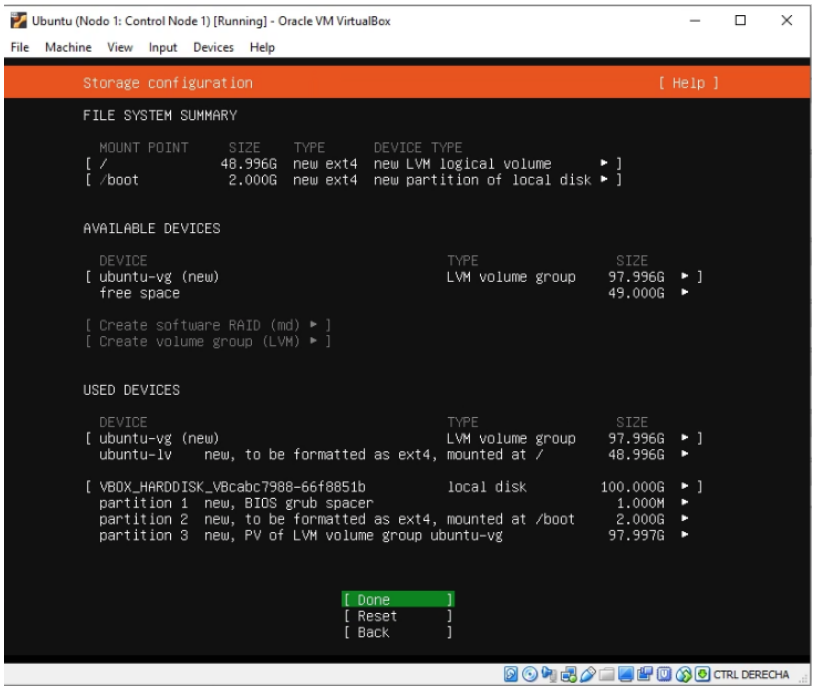


Figura 20: Configuración del disco de la máquina virtual

- Seleccionar un usuario, contraseña y hostname
- Setear *SSH*

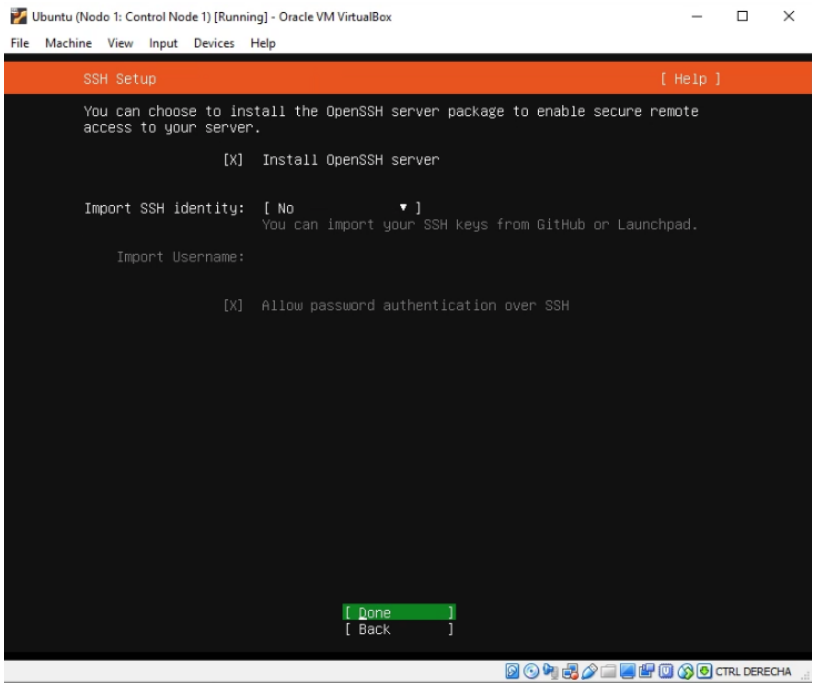


Figura 21: Configuración de *SSH*

Una vez se terminan estas configuraciones, reiniciamos la máquina virtual para tener

nuestra máquina virtual lista.

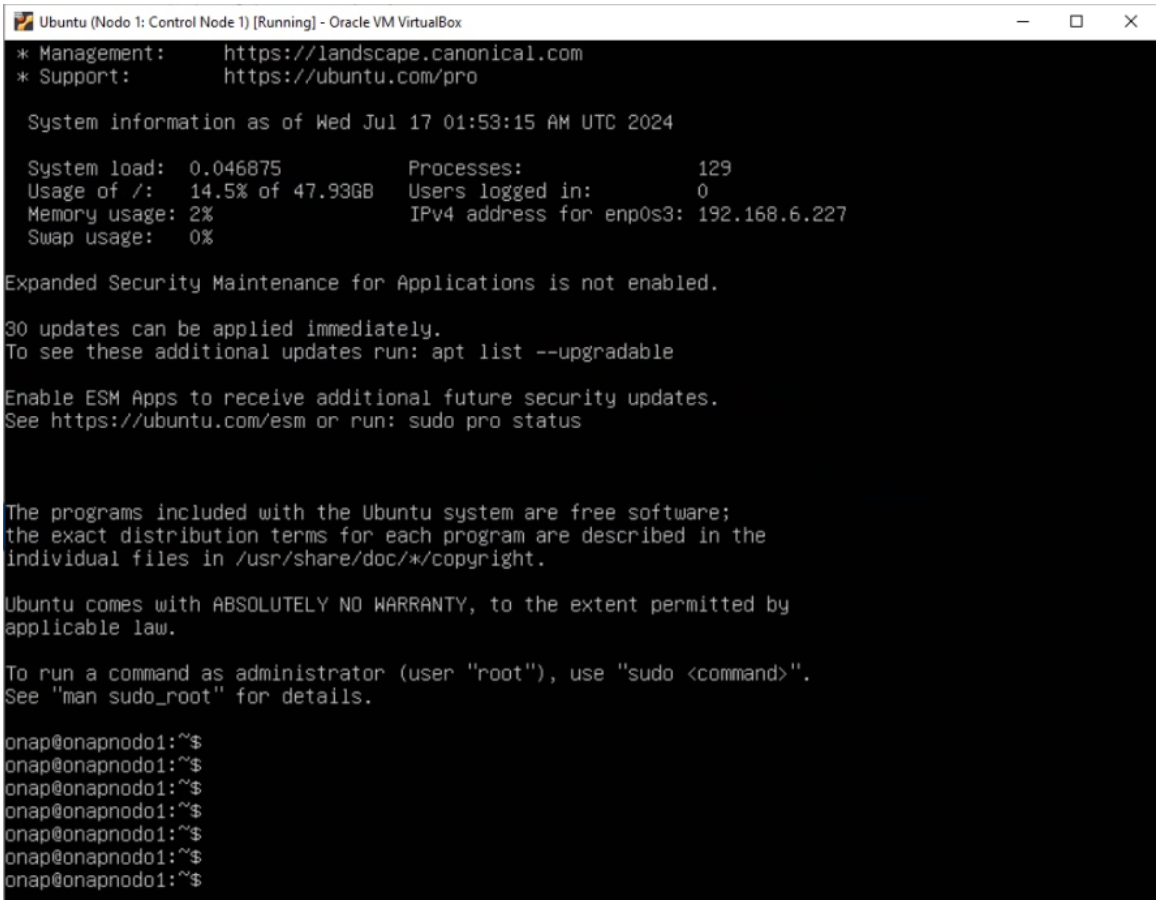


Figura 22: *Ubuntu Server* 22.04 LTS instalado en la máquina virtual

7.2.2. Paso 2: Instalación de *Container Runtime* en cada máquina virtual (*Docker Engine*)

El siguiente paso a seguir en la instalación del clúster de es la instalación del *Container Runtime*. En este caso, instalaremos *Docker Engine* como *Container Runtime*. Por ello, nos basaremos en la documentación oficial de *Docker* para realizar la instalación en *Ubuntu* [40]

Para dicha instalación, utilizaremos la opción de instalar utilizando un repositorio *apt* para garantizar que se instale la última versión de *Docker Engine*. Para ello, se deben seguir los siguientes pasos:

- Eliminar posibles paquetes anteriores de *Docker*

```
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-
    ↪ docker containerd runc; do sudo apt-get remove \$pkg; done
```

Cuadro 1: Código para eliminar posibles paquetes anteriores de *Docker*

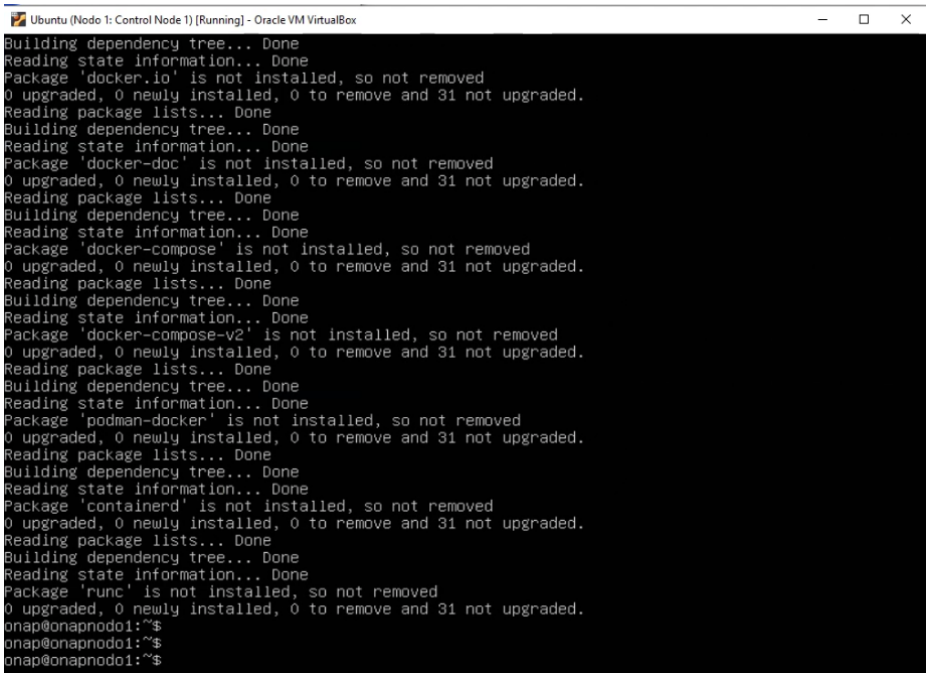


Figura 23: Paquetes de *Docker* eliminados

- Configurar el repositorio *apt* de *Docker*

```
# Add official GPG key
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt
    ↪ /keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Cuadro 2: Código para configurar el repositorio *apt* de *Docker*

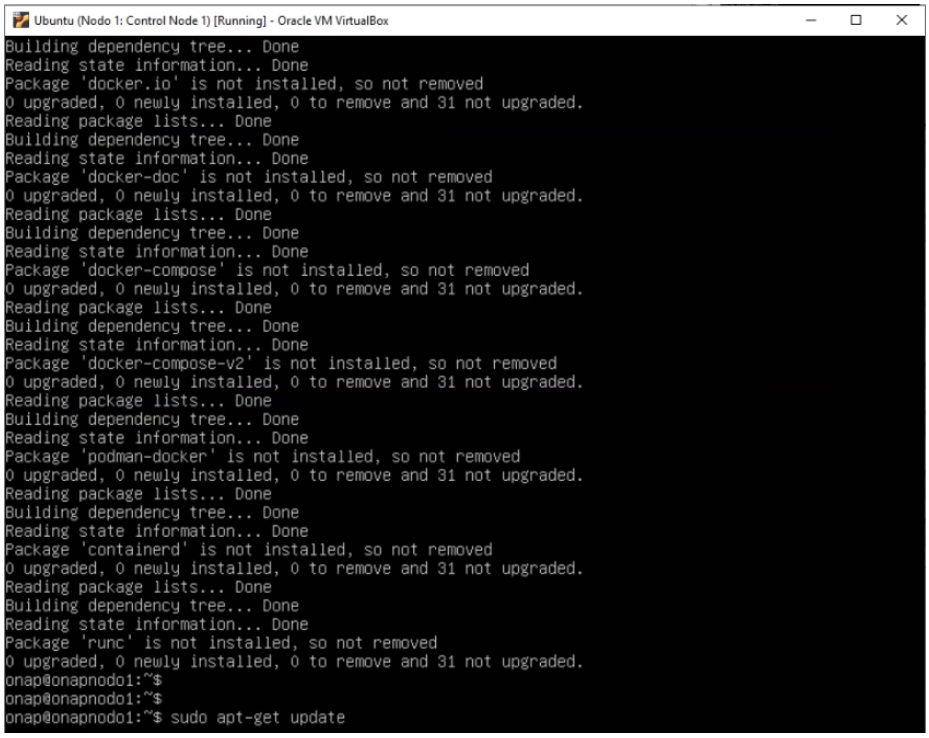


Figura 24: Configurar el repositorio `apt` de *Docker*

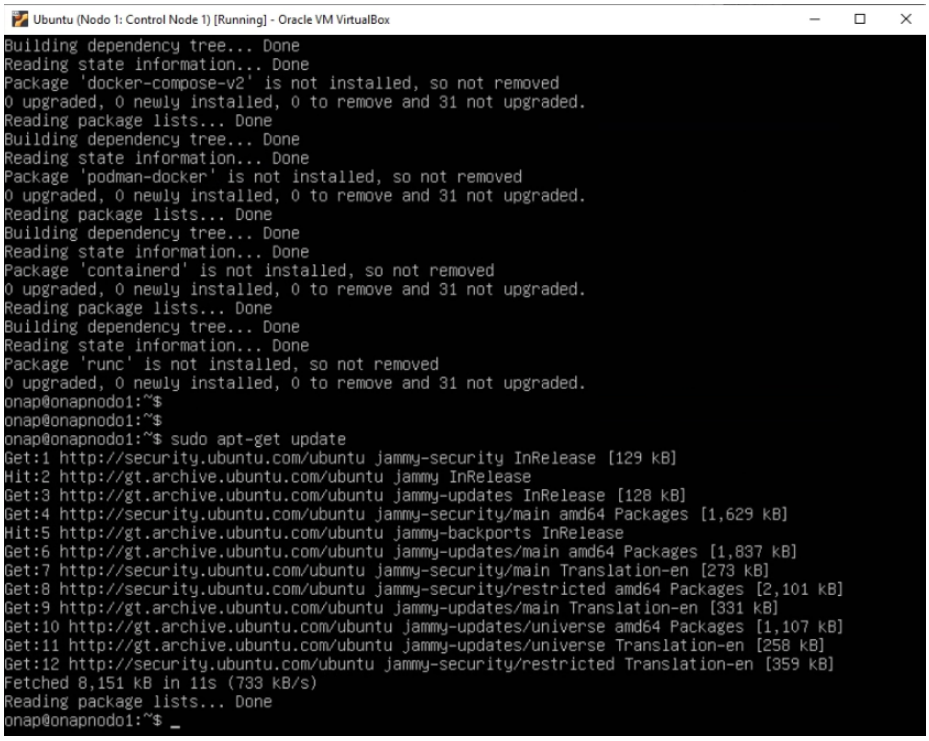


Figura 25: Configurar el repositorio `apt` de *Docker*

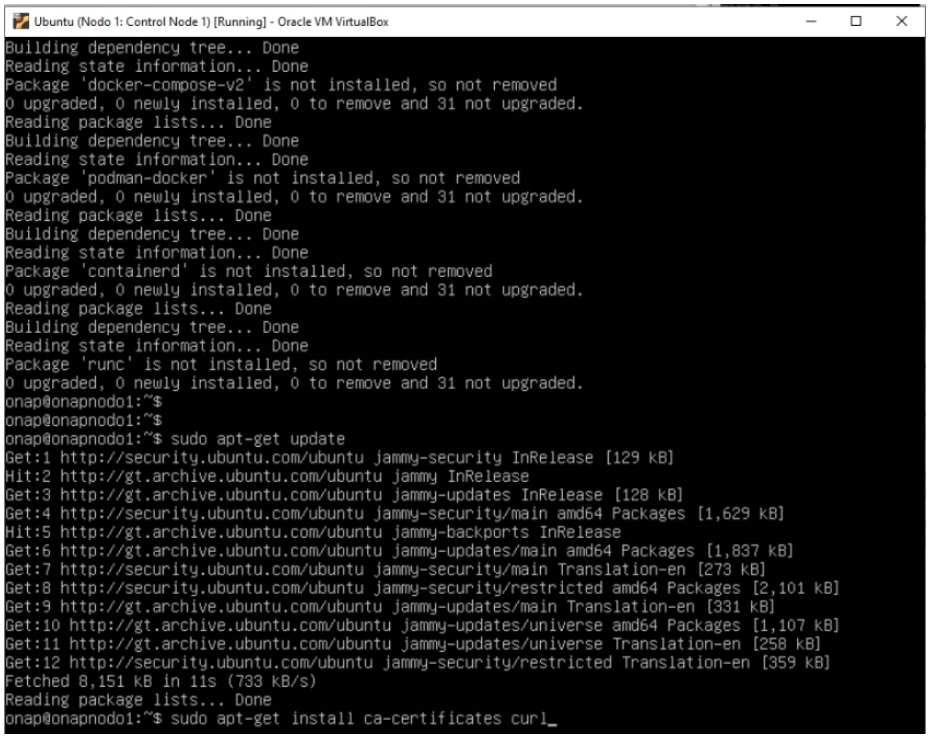


Figura 26: Configurar el repositorio `apt` de *Docker*

```
onap@onapnodo1:~$ sudo apt-get install ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.16).
curl set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
onap@onapnodo1:~$ _
```

Figura 27: Configurar el repositorio `apt` de *Docker*

```
onap@onapnodo1:~$ sudo install -m 0755 -d /etc/apt/keyrings
onap@onapnodo1:~$
```

Figura 28: Configurar el repositorio `apt` de *Docker*

```
onap@onapnodo1:~$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
docker.asc
onap@onapnodo1:~$ _
```

Figura 29: Configurar el repositorio `apt` de *Docker*

```
onap@onapnodo1:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
onap@onapnodo1:~$ _
```

Figura 30: Configurar el repositorio `apt` de *Docker*

```
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings
    ↪ /docker.asc] https://download.docker.com/linux/ubuntu $(. /etc/os-
    ↪ release && echo "$VERSION_CODENAME") stable" | sudo tee /etc/apt/
    ↪ sources.list.d/docker.list > /dev/null
sudo apt-get update
```

Cuadro 3: Añadir el repositorio a una fuente *apt*

Si se nota, este comando se implementó de forma distinta a como la documentación nos lo indica, pues la forma original daba problemas en la sintaxis. Al cambiar el comando a la forma documentada arriba, dio resultados exitosos.

```
onap@onapnodo1:~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.as
c] https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo "$VERSION_CODENAME") stable"
 | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
[sudo] password for onap:
onap@onapnodo1:~$
```

Figura 31: Añadir el repositorio a una fuente *apt*

```
onap@onapnodo1:~$ sudo apt-get update
Get:1 https://download.docker.com/linux/ubuntu jammy InRelease [48.8 kB]
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Get:3 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [35.7 kB]
Hit:4 http://gt.archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://gt.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://gt.archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 84.5 kB in 1s (138 kB/s)
Reading package lists... Done
onap@onapnodo1:~$ _
```

Figura 32: Añadir el repositorio a una fuente *apt*

- Instalar los paquetes de *Docker*

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx
    ↪ -plugin docker-compose-plugin
```

Cuadro 4: Instalación de *Docker*

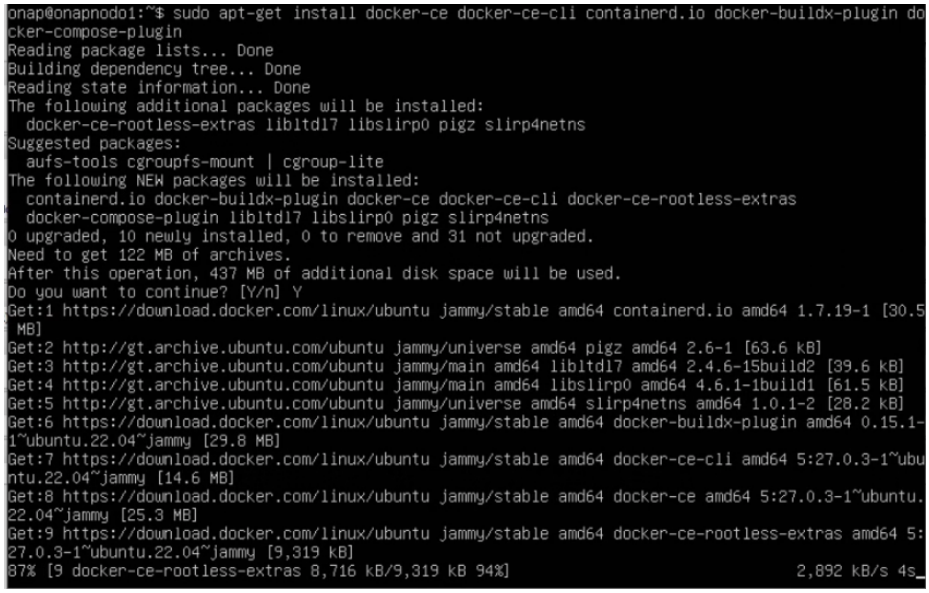


Figura 33: Instalación de *Docker*

- Verificar que la instalación sea exitosa al utilizar una imagen de prueba llamada *"hello-world"*. Si nuestra instalación está correcta, en la terminal aparecerá un mensaje indicando que la instalación fue exitosa.

```
1 sudo docker run hello-world
```

Cuadro 5: Prueba *"Hello world"*

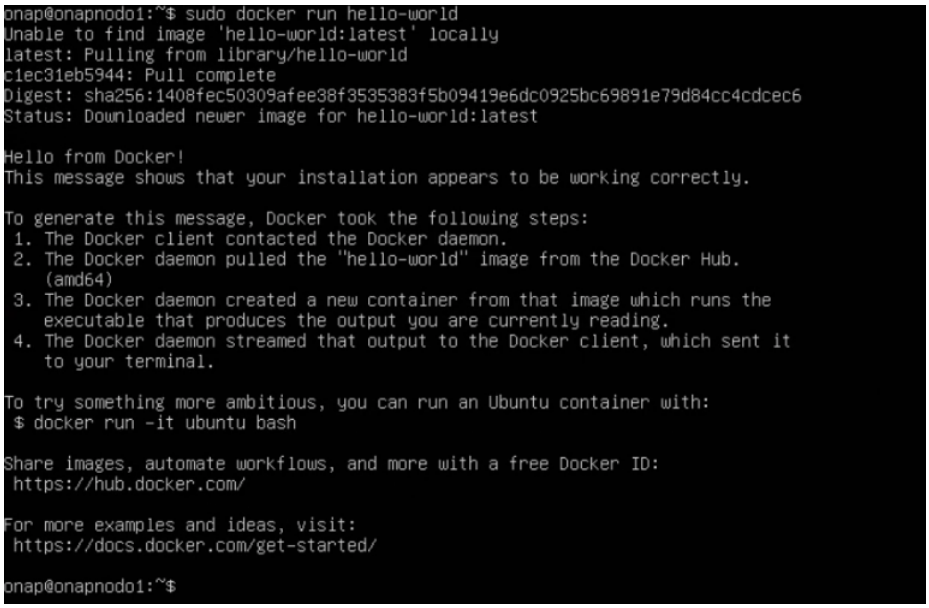


Figura 34: Prueba de *Docker* con *"Hello world"*

En la instalación de *Docker* existen otras posibilidades que pueden funcionar, tal como instalar desde un paquete tipo *deb* o a través de un *convenience script*. En este caso se decidió instalar a través de repositorios de *Docker*, pero si existe algún problema con este tipo de instalación, se puede utilizar cualquiera de las otras dos opciones.

Docker es una de las opciones más utilizadas actualmente. A pesar de ello, desde la versión 1.20 de *Kubernetes*, *Docker* ya no es soportado directamente como *runtime* de contenedores. Por esto es que es necesario realizar la instalación de *cri-dockerd*, el cual es un componente que permite que *Docker* sea compatible como *Container Runtime* para interactuar con *Kubernetes*, a través de la API CRI (*Container Runtime Interface*).

Esta instalación se realizó de forma manual a través de la documentación de *cri-dockerd* [41]. A pesar de que *Mirantis* nos da las instrucciones para la instalación manual, antes debemos de clonar el repositorio de *Github* de *cri-dockerd* para que la instalación funcione. Estos son los pasos a seguir para la instalación de *cri-dockerd*:

- Clonar repositorio de *cri-dockerd*:

```
git clone https://github.com/Mirantis/cri-dockerd.git
cd cri-dockerd
mkdir bin
```

Cuadro 6: Clonar repositorio de *cri-dockerd*

```
onap@onapnodo1:~$ git clone https://github.com/Mirantis/cri-dockerd.git
Cloning into 'cri-dockerd'...
remote: Enumerating objects: 23144, done.
remote: Counting objects: 100% (3816/3816), done.
remote: Compressing objects: 100% (1787/1787), done.
remote: Total 23144 (delta 2314), reused 2657 (delta 1967), pack-reused 19328
Receiving objects: 100% (23144/23144), 48.37 MiB | 2.93 MiB/s, done.
Resolving deltas: 100% (11861/11861), done.
onap@onapnodo1:~$
```

Figura 35: Repositorio de *cri-dockerd* clonado

- Instalar *GO* 1.22.5 en root (no en el directorio de *cri-dockerd*)

```
sudo apt install golang-go
```

Cuadro 7: Instalar *GO* 1.22.5

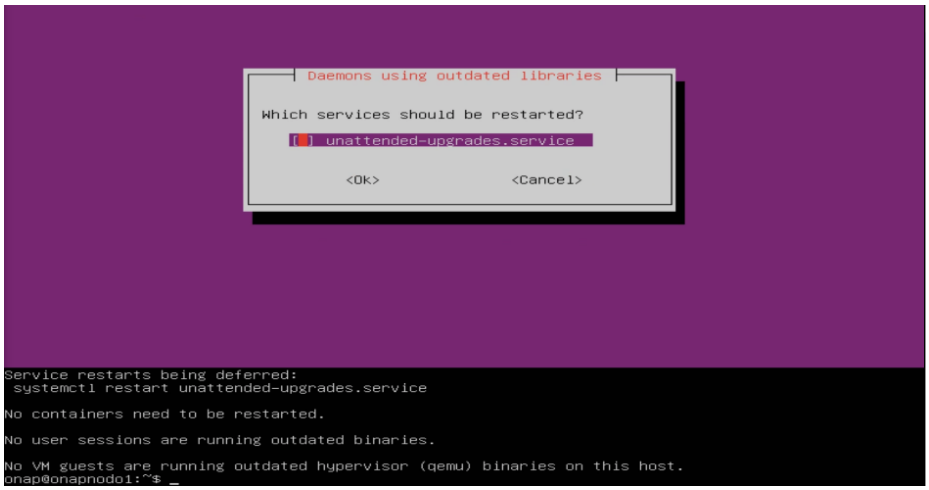


Figura 36: Instalación de *GO* 1.22.5

```
wget https://go.dev/dl/go1.22.5.linux-amd64.tar.gz -O go.tar.gz
```

Cuadro 8: Instalar *GO* 1.22.5

Figura 37: Instalación de *GO* 1.22.5

```
sudo tar -xzvf go.tar.gz -C /usr/local
```

Cuadro 9: Instalar *GO* 1.22.5

Figura 38: Instalación de *GO* 1.22.5

```
echo export PATH=$HOME/go/bin:/usr/local/go/bin:$PATH >> ~/.profile
source ~/.profile
```

Cuadro 10: Añadir ejecutables de *GO* a la variable *PATH* del sistema

Figura 39: Ejecutables de *GO* añadidos a la variable *PATH* del sistema

- Verificar versión de *GO*

```
go version
```

Cuadro 11: Ver versión de *GO* instalada

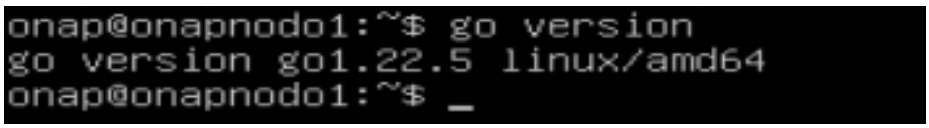


Figura 40: *GO* 1.22.5 instalado

- Movernos al directorio en el que se compiló *cri-dockerd* y contruir el binario

```
cd cri-dockerd
go build -o bin/cri-dockerd
```

Cuadro 12: Construir binario para *cri-dockerd*

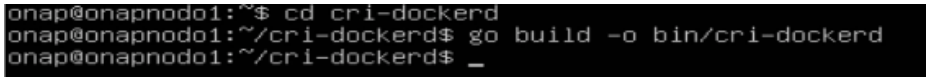


Figura 41: Construcción de binario para *cri-dockerd*

- Instalar el binario desde el directorio en el que nos encontramos, instalamos los archivos de servicio *Systemd* y modificamos el archivo de servicio

```
sudo install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-
    ↪ dockerd
sudo install packaging/systemd/* /etc/systemd/system
sudo sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc
    ↪ /systemd/system/cri-docker.service
```

Cuadro 13: Instalación de binario contruido

Figura 42: Binario anteriormente construido, instalado

- Recargar *Systemd* y habilitar el servicio

```
sudo systemctl daemon-reload
sudo systemctl enable --now cri-docker.socket
sudo systemctl enable cri-docker.service
```

Cuadro 14: Habilitar *cri-docker*

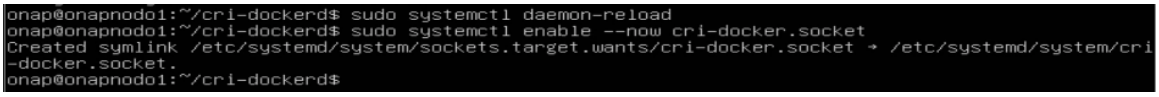


Figura 43: Servicio *cri-docker* habilitado

- Inicializar manualmente el servicio para asegurarnos que el servicio se encuentre arriba

```
sudo systemctl start cri-docker.service
```

Cuadro 15: Habilitar *cri-docker*

```
onap@onapnodo1:~/cri-dockerd$ cd
onap@onapnodo1:~$ sudo systemctl start cri-docker.service
onap@onapnodo1:~$
```

Figura 44: Habilitación manual de *cri-docker*

- Verificar el estado de la instalación

```
sudo systemctl status cri-docker.service
sudo systemctl status cri-docker.socket
```

Cuadro 16: Comprobación de la instalación realizada

```
onap@onapnodo1:~$ sudo systemctl status cri-docker.service
● cri-docker.service - CRI Interface for Docker Application Container Engine
     Loaded: loaded (/etc/systemd/system/cri-docker.service; disabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-07-18 23:59:12 UTC; 1min 12s ago
TriggeredBy: ● cri-docker.socket
       Docs: https://docs.mirantis.com
   Main PID: 10907 (cri-dockerd)
      Tasks: 8
     Memory: 9.4M
        CPU: 32ms
     CGroup: /system.slice/cri-docker.service
             └─10907 /usr/local/bin/cri-dockerd --container-runtime-endpoint fd://

Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Hairpin m>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="The binar>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="The binar>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Loaded ne>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Docker cr>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Setting c>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Docker cr>
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Starting >
Jul 18 23:59:12 onapnodo1 cri-dockerd[10907]: time="2024-07-18T23:59:12Z" level=info msg="Start cri>
Jul 18 23:59:12 onapnodo1 systemd[1]: Started CRI Interface for Docker Application Container Engine.
lines 1-22/22 (END)
```

Figura 45: Estado de la instalación de *cri-docker*

```
onap@onapnodo1:~$ sudo systemctl status cri-docker.socket
● cri-docker.socket - CRI Docker Socket for the API
     Loaded: loaded (/etc/systemd/system/cri-docker.socket; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-07-18 23:57:14 UTC; 4min 9s ago
   Triggers: ● cri-docker.service
     Listen: /run/cri-dockerd.sock (Stream)
      Tasks: 0 (limit: 9159)
     Memory: 0B
        CPU: 488us
     CGroup: /system.slice/cri-docker.socket

Jul 18 23:57:14 onapnodo1 systemd[1]: Starting CRI Docker Socket for the API...
Jul 18 23:57:14 onapnodo1 systemd[1]: Listening on CRI Docker Socket for the API.
onap@onapnodo1:~$
```

Figura 46: Estado de la instalación de *cri-docker*

Ahora que nuestro servicio y el socket de *cri-dockerd* se encuentran activos, quiere decir que la instalación de la plataforma se logró exitosamente.

7.2.3. Paso 3: Instalar *kubeadm*, *kubelet* y *kubectl*

Según la documentación de *Kubernetes*, la última versión disponible para instalar es la 1.30 para julio de 2024. A pesar de ello, ya que se intentó levantar una instancia de ONAP en la versión *Montreal*, será necesario instalar otra versión de *kubeadm*, *kubelet* y *kubectl*, que sean compatibles con la versión *Montreal* de ONAP. Nos podemos basar en lo que nos menciona la documentación oficial de ONAP [8]:

Release	Kubernetes	Helm	kubectl	Docker	Cert-Manager	Strimzi
Kohn	1.23.8	3.8.2	1.23.8	20.10.x	1.8.0	0.32.0
London	1.23.8	3.8.2	1.23.x	20.10.x	1.12.2	0.35.0
Montreal	1.27.5	3.12.3	1.27.x	20.10.x	1.13.2	0.36.1

Figura 47: Versiones de *Kubernetes* compatibles con diferentes versiones de ONAP

Por ello, instalaremos la versión 1.27.5 de *Kubernetes*:

- Actualizar el paquete *apt* e instalar paquetes necesarios para utilizar el repositorio *apt* de *Kubernetes*

```
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

Cuadro 17: Actualización e instalación de paquetes para uso del repositorio de *Kubernetes*

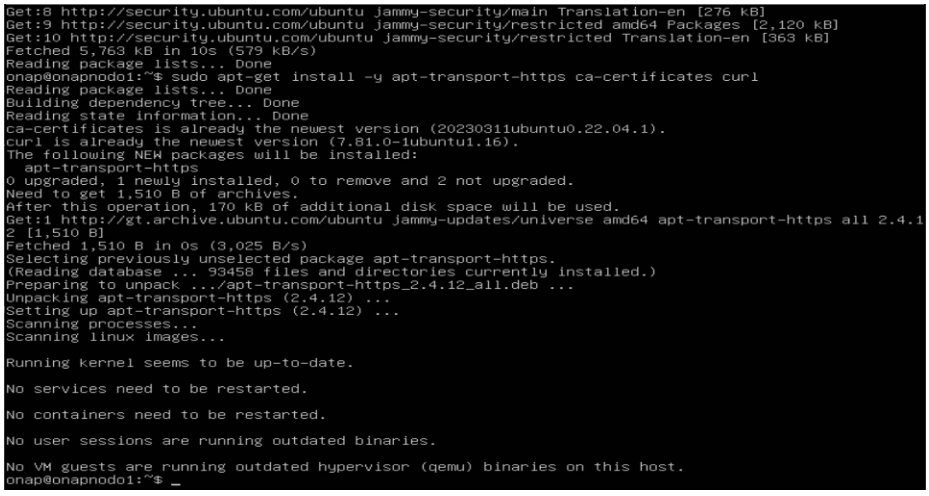


Figura 48: Actualización e instalación de paquetes para uso del repositorio de *Kubernetes*

- Necesitaremos una clave de firma pública que se utiliza en todos los repositorios de paquetes de *Kubernetes*, por lo que debemos descargarla. Es bueno tomar en cuenta que para versiones más antiguas que *Debian* 12 y *Ubuntu* 22.04, el directorio */etc/apt/keyrings* no existe por defecto, por lo que hay que crearlo:

```
sudo mkdir -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.27/deb/Release.key |
    ↪ sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

Cuadro 18: Descarga de firma pública

```
onap@onapnodo1:~$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.27/deb/Release.key | sudo gpg --de
armor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
onap@onapnodo1:~$ _
```

Figura 49: Firma pública descargada para el repositorio de paquetes de *Kubernetes*

- Agregar el repositorio apt de *Kubernetes*. En este caso se utilizará el repositorio para la versión 1.27 de *Kubernetes* pero si en caso se quiere utilizar otra versión, será necesario cambiar el URL de la versión apropiada.

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https
    ↪ ://pkgs.k8s.io/core:/stable:/v1.27/deb/ /' | sudo tee /etc/apt/
    ↪ sources.list.d/kubernetes.list
```

Cuadro 19: Agregar repositorio para versión 1.27 de *Kubernetes*

```
onap@onapnodo1:~$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyrings.gpg] https://pkgs.k
8s.io/core:/stable:/v1.27/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyrings.gpg] https://pkgs.k8s.io/core:/stable:/v1.2
7/deb/ /
onap@onapnodo1:~$
```

Figura 50: Repositorio de *Kubernetes* agregado

- Una vez se tienen los repositorios correspondientes, ya podemos instalar *kubeadm*, *kubelet* y *kubectl* en la versión correspondiente. Es importante hacer los comandos en el código 22, ya que este evita que los paquetes instalados sean susceptibles a actualizaciones automáticas y permanezcan con la misma versión siempre. Ahora, puede ser que al intentar instalar alguna versión del paquete, la versión no se encuentre. Para ello, es bueno realizar los siguientes comandos:

```
apt-cache madison kubelet
apt-cache madison kubeadm
apt-cache madison kubectl
```

Cuadro 20: Versiones de paquetes de *Kubernetes*

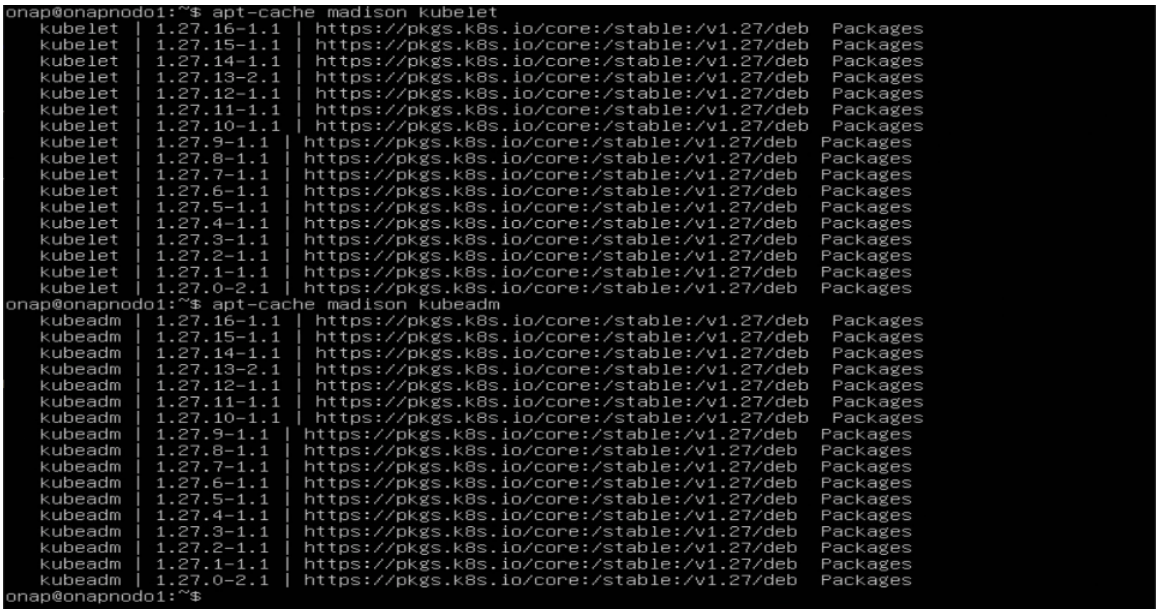


Figura 51: Versiones de paquetes de *Kubernetes*

A través de esos comandos, podemos observar todas las versiones disponibles para instalar. Una vez encontremos la versión que queremos instalar, podemos proseguir con los siguientes comandos:

```
sudo apt-get update
sudo apt-get install -y kubelet=1.27.5-1.1 kubeadm=1.27.5-1.1 kubectl
    ↪ =1.27.5-1.1
sudo apt-mark hold kubelet kubeadm kubectl
```

Cuadro 21: Instalación de *Kubelet*, *Kubeadm* y *Kubectl*

Figura 52: Actualizar *apt-get*

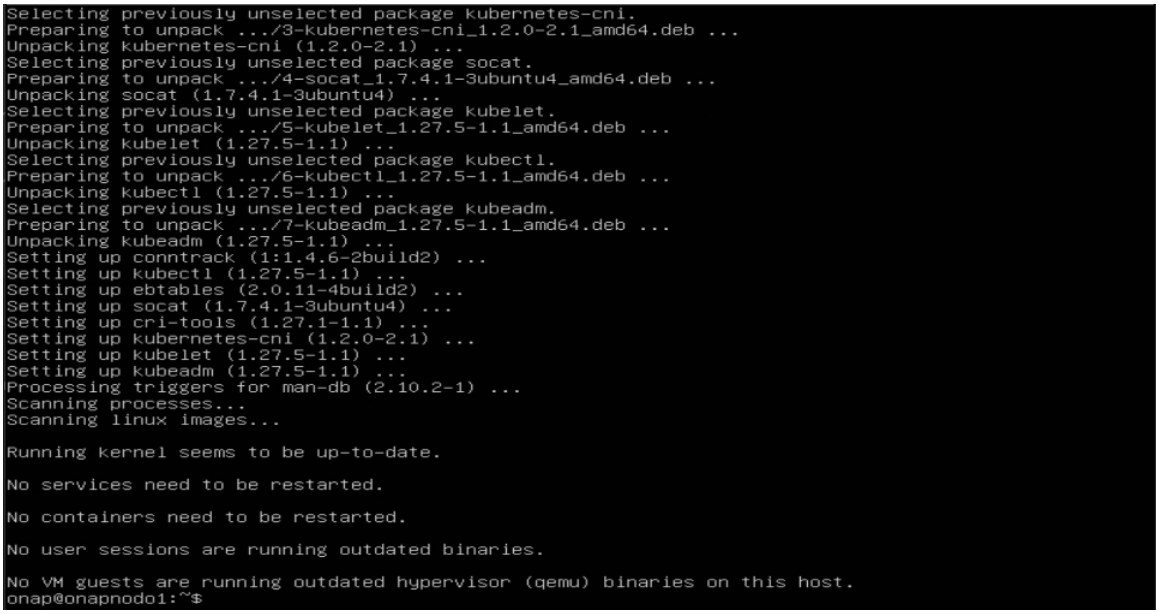


Figura 53: Instalación de *kubeadm*, *kubelet* y *kubectl*

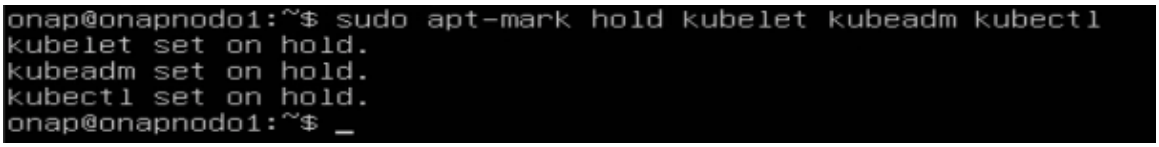


Figura 54: Restringir actualizaciones automáticas de otras versiones de *kubeadm*, *kubelet* y *kubectl*

Con esto, ya tendríamos listos nuestros paquetes de *kubeadm*, *kubelet* y *kubectl*, con los cuales podremos levantar nuestro clúster de *Kubernetes* de forma más sencilla.

- Verificar la instalación de los componentes de *Kubernetes* (opcional)

```
1 kubeadm version
2 kubelet --version
3 kubectl version --client
```

Cuadro 22: Verificación de instalación de *Kubernetes*

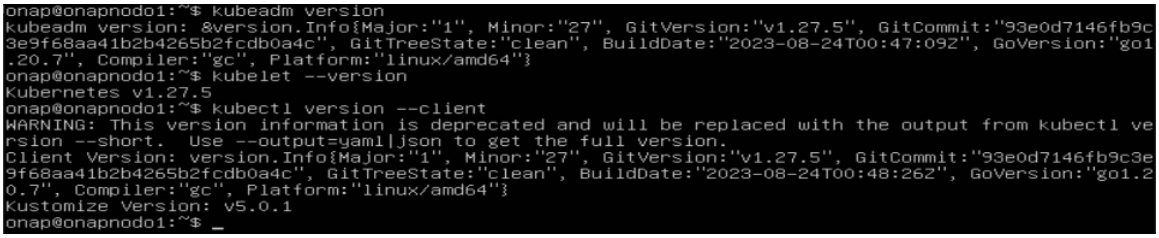


Figura 55: Verificación de instalación de *Kubernetes*

Esto nos indica que todos los componentes se instalaron de la manera apropiada y en la versión correcta.

7.2.4. Paso 4: Creación del clúster de Kubernetes

Al momento de crear el clúster de *Kubernetes*, necesitaremos realizar los pasos anteriores para cada máquina virtual que funcionen como nodos de nuestro clúster. Una forma sencilla de realizarlo es configurando e instalando todo en una sola máquina virtual y luego hacer clones de la misma para tener las máquinas virtuales necesarias listas. Una buena práctica es guardar una copia del archivo .ova de nuestra máquina virtual con todos los prerrequisitos instalados.

Una vez se tienen todas las máquinas virtuales listas, necesitaremos asignarle una *IP* estática a cada máquina virtual. A través de la *IP* es que podremos construir nuestro clúster. Es bueno aclarar que para que nuestro clúster soporte el despliegue de cualquier tipo de aplicación, será necesario que este mismo tenga alta disponibilidad. Alta disponiblidad en un clúster se refiere a cuando tenemos múltiples nodos de control que permitan manejar los recursos de los nodos de trabajo. Otro aspecto importante a tomar en cuenta es que para que nuestro clúster posea alta disponibilidad, instalaremos dos máquinas virtuales adicionales que contengan *load balancers* que añadan redundancias al crear una *IP* virtual que se mantenga todo el tiempo disponible utilizando el protocolo de redundancia de enrutador virtual (VRRP). Con estas máquinas, estaremos listos para desplegar nuestro clúster y así, evitar que el clúster sea inestable. A continuación se proporcionará una serie de pasos a seguir para levantar el clúster de *kubernetes* de forma exitosa:

- Asignar *IP* estática a cada máquina virtual. Para esto, se utilizará la siguiente tabla, para saber que *IPs* tendrán cada una de nuestras máquinas virtuales:

Nodo	**PC**	**Rol**	***IP***
1	07	*Control-plane* node 1	192.168.74.32/22
2	07	*Control-plane node* 2	192.168.74.33/22
3	08	*Control-plane node* 3	192.168.74.34/22
4	08	*Worker node* 1	192.168.74.35/22
5	12	*Worker node* 2	192.168.74.36/22
6	12	*Worker node* 3	192.168.74.37/22
7	02	*HAProxy* 1	192.168.74.30/22
8	02	*HAProxy* 2	192.168.74.31/22
-	-	*IP* Virtual	192.168.74.38/22
9	16	*Server NFS*	192.168.74.39/22

Cuadro 23: Descripción de los nodos del clúster

Para asignar la *IP* estática, primero debemos de verificar el nombre del archivo con la configuración de red y luego ejecutar el siguiente comando:

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

Cuadro 24: Acceso a configuración de red de la máquina virtual

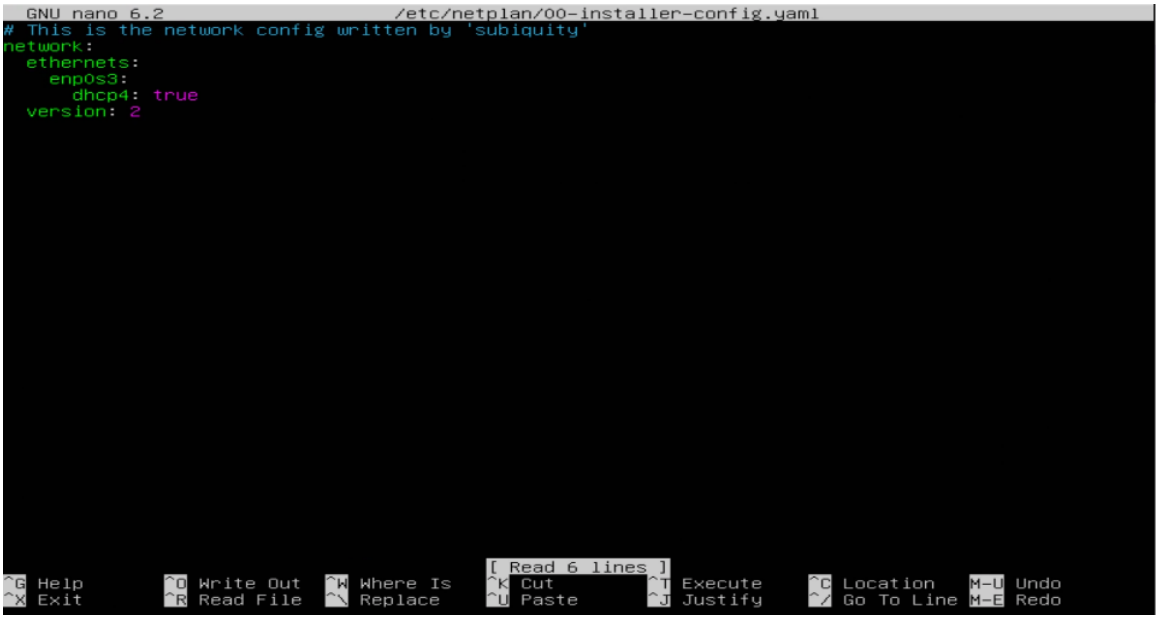


Figura 56: Configuración de red de la máquina virtual

Cambiar los parámetros para configurar una *IP* estática:

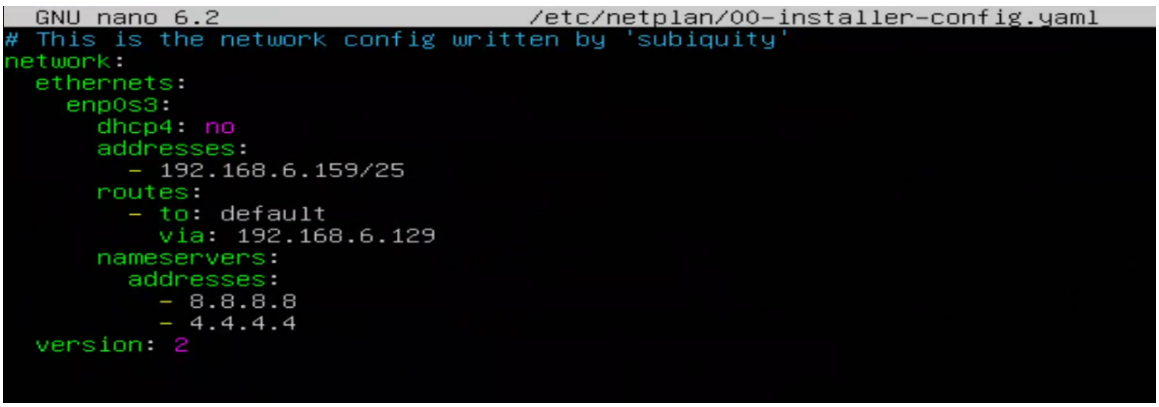


Figura 57: *IP* estática asignada a la máquina virtual

Luego, podemos verificar si la configuración funciona. Si este comando no funciona, revertirá la configuración luego de un *timeout*, pero, si funciona, tendremos verificación que nuestra configuración funciona.

```
sudo netplan try
```

Cuadro 25: Prueba de configuración de red

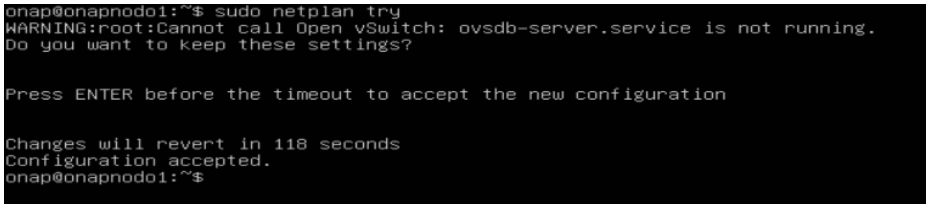


Figura 58: Prueba de configuración de red

En este caso solo tuvimos un *warning* que no afecta con la configuración de red que intentamos aplicar.

Una vez vemos que todo funciona de manera correcta, corremos los comandos:

```
sudo netplan apply
sudo systemctl restart systemd-networkd
sudo systemctl restart systemd-resolved
```

Cuadro 26: Comandos para aplicar configuración de red en la máquina virtual

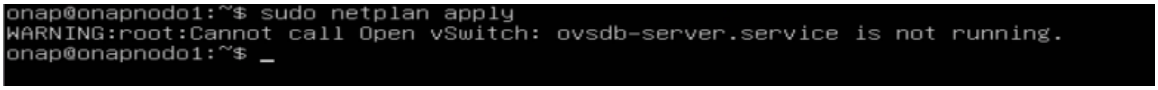


Figura 59: Aplicación de configuración de red en la máquina virtual

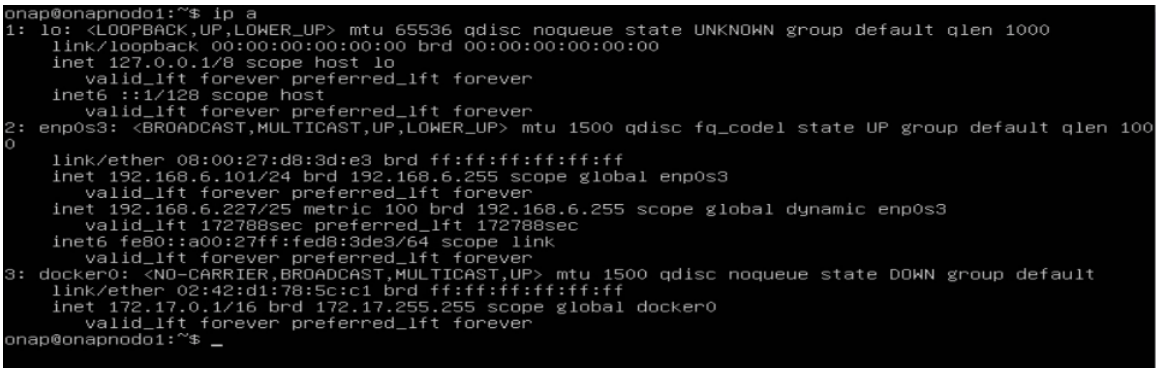


Figura 60: Comprobación de nueva configuración de red en la máquina virtual

- Aplicar la configuración para todas las máquinas virtuales que van a conformar nuestro clúster.

- Cambiar *Hostname* e ip para cada una de las máquinas virtuales (Cambiar el nombre correspondiente en las líneas donde aparece onap_nodox):

```
1 sudo nano /etc/hostname
```

Cuadro 27: Cambio de *hostname*

Figura 61: *Hostname* nuevo asignado a la máquina virtual

```
1 sudo nano /etc/hosts
```

Cuadro 28: Ajuste de nombre de *host*

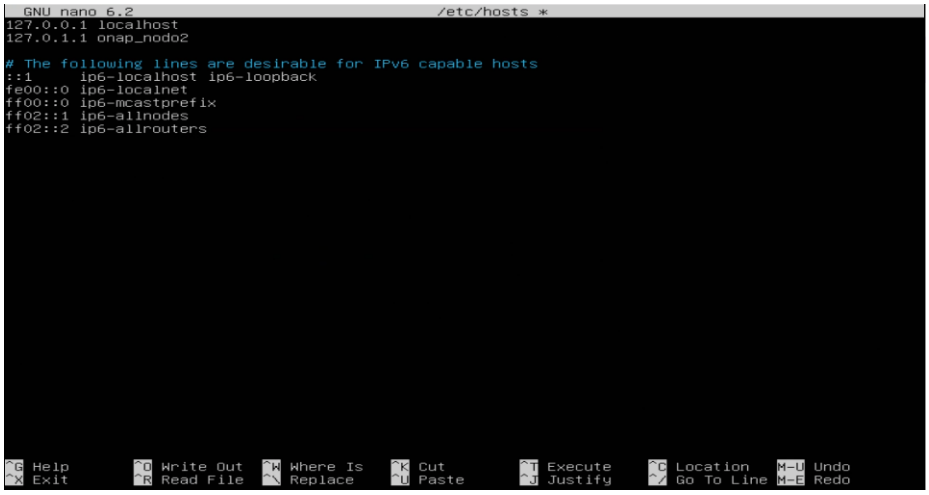


Figura 62: Ajuste de nombre de *host*

```
sudo hostnamectl set-hostname onapnodo2 --transient
sudo hostnamectl set-hostname onapnodo2 --static
hostnamectl
sudo reboot
```

Cuadro 29: Aplicación de la configuración nueva de *hostname*

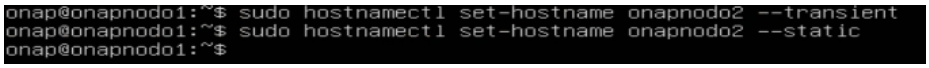


Figura 63: Aplicación de la configuración nueva de *hostname*

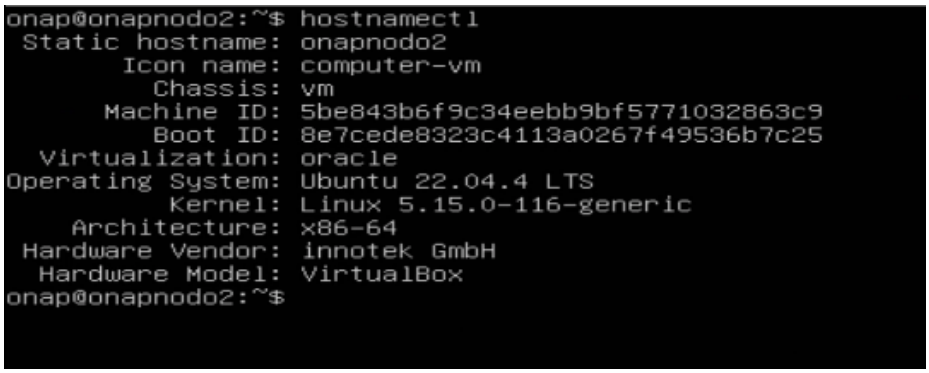


Figura 64: Comprobación de la configuración nueva de *hostname*

- Configurar dos *Load Balancers* (en máquinas virtuales diferentes) para los nodos de control (Se utilizará *HAProxy*) junto con *Keepalived*. Este *Load Balancer* junto con *Keepalived* no será parte del clúster, pero ayudará a generar un *endpoint* del plano de control y a darle redundancia al mismo.

```
sudo apt-get update
sudo apt-get install -y haproxy keepalived
```

Cuadro 30: Instalar *HAProxy* y *Keepalived*

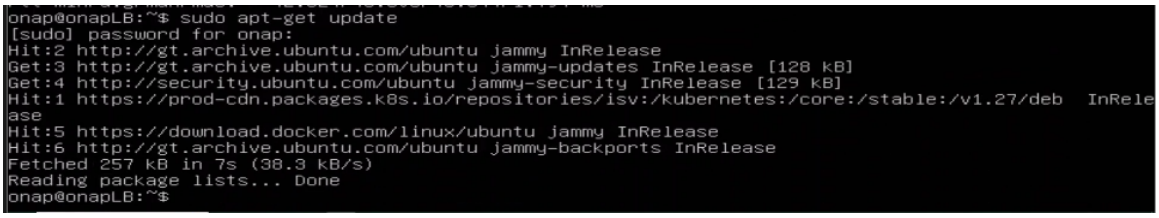


Figura 65: Instalación de *HAProxy* y *Keepalived*

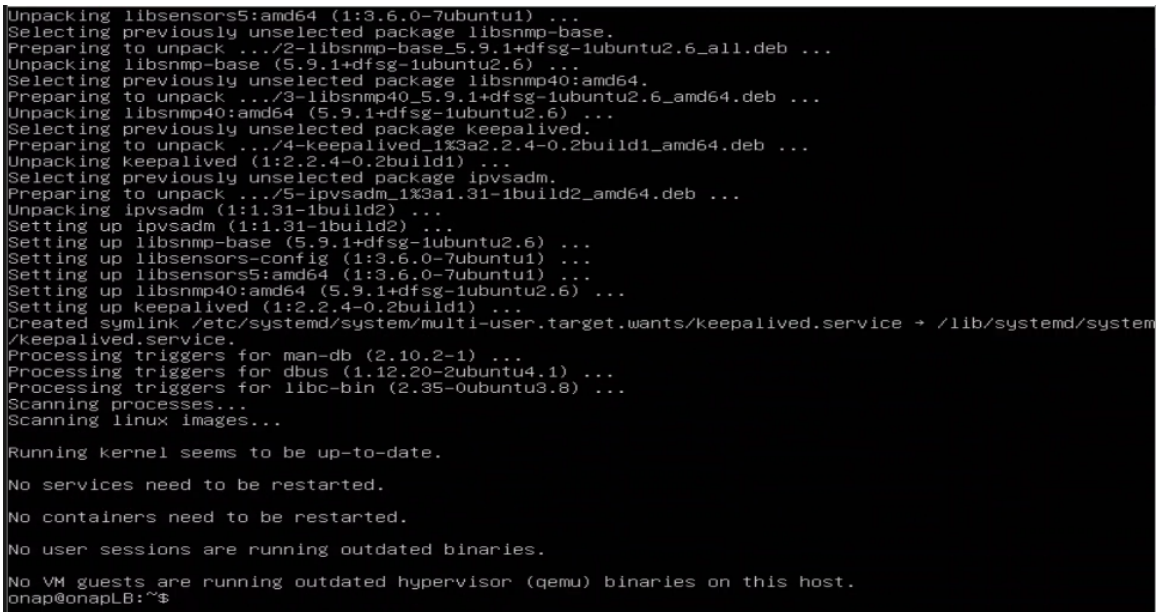


Figura 66: Instalación de *HAProxy* y *Keepalived*

Previo a escribir nuestra configuración para el *HAProxy*, debemos realizar los siguientes comandos:

```
sudo nano /etc/selinux/config
getenforce
```

Cuadro 31: Deshabilitar *getenforce*

Figura 67: *SELinux* deshabilitado

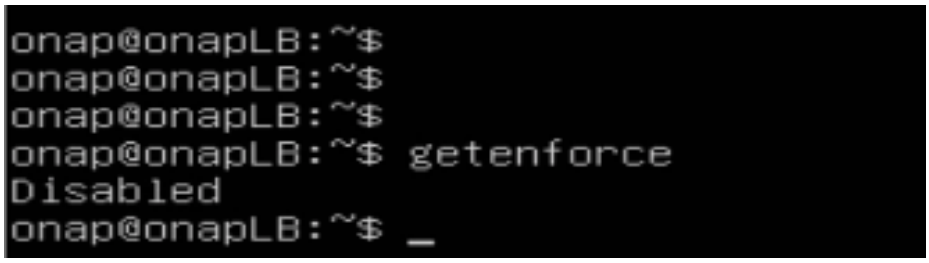


Figura 68: *Getenforce* deshabilitado

Si no funciona el comando *getenforce*, realizar comando: *sudo apt install selinux-utils* previo a realizar el comando *getenforce*.

Una vez modificado esto, debemos configurar el archivo en la ruta */etc/haproxy/haproxy.cfg* para poder agregar toda la configuración necesaria para que el *Load Balancer* (*HAProxy*) funcione sobre las máquinas que funcionarán como los nodos de control:

```
1 sudo nano /etc/haproxy/haproxy.cfg
```

Cuadro 32: Acceso a configuración de *HAProxy*

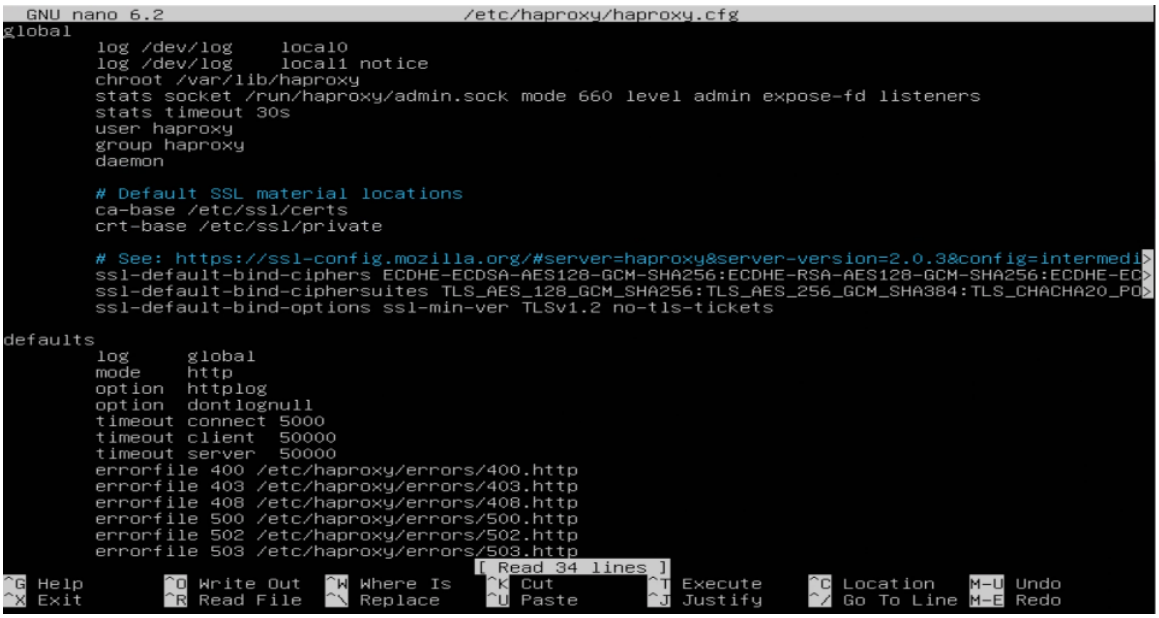


Figura 69: Configuración de *HAProxy*

En este archivo, adicional a la configuración encontrada, colocar la siguiente configuración:

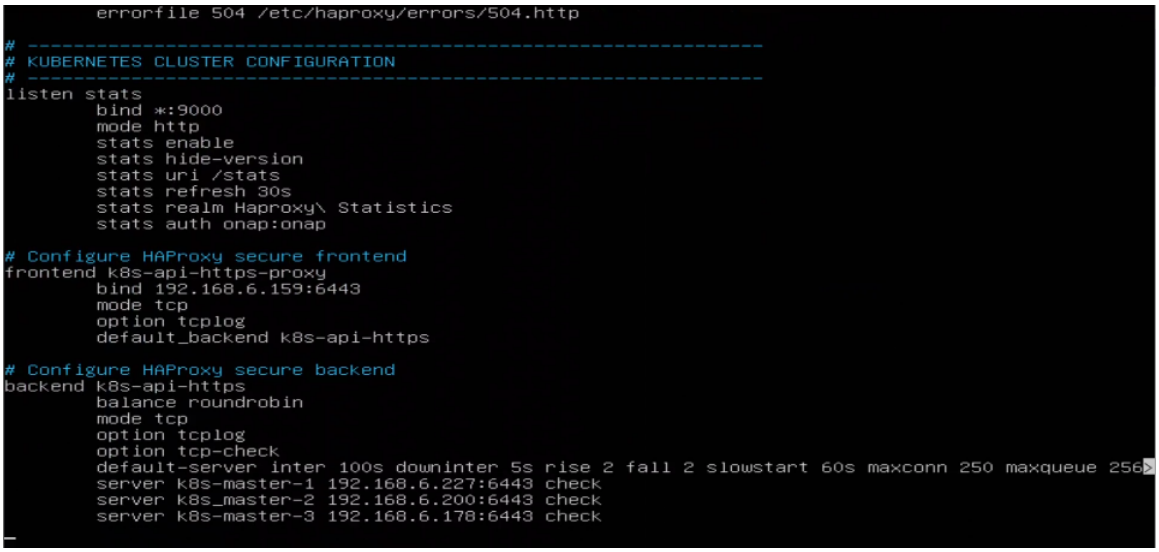


Figura 70: Configuración añadida a *HAProxy*

Toda la configuración colocada se debe ver de la siguiente manera:

```
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd
            ↪ listeners
        stats timeout 30s
        maxconn 4000
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-version
            ↪ =2.0.3&config=intermediate
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
            ↪ AES128-GCM-SHA256:ECDHE-ECDSA-AES256->        ssl-default-bind
            ↪ -ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
            ↪ TLS_CHACHA20_POLY1305_SHA2>        ssl-default-bind-options
            ↪ ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
```

```
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

# ---------------------------------------------------------------
# KUBERNETES CONFIGURATION
# ---------------------------------------------------------------
listen stats
        bind *:9000
        mode http
        stats enable
        stats hide-version
        stats uri /stats
        stats refresh 30s
        stats realm Haproxy\ Statistics
        stats auth onap:onap

frontend k8s-api-https-proxy
        bind *:6443
        mode tcp
        option tcplog
        default_backend k8s-api-https

backend k8s-api-https
        balance roundrobin
        mode tcp
        option tcplog
        option tcp-check
        default-server inter 100s downinter 5s rise 2 fall 2 slowstart 60s
            ↪ maxconn 250 maxqueue 256 weight 100
        server k8s-api-1 192.168.6.137:6443 check
        server k8s-api-2 192.168.6.138:6443 check
        server k8s-api-3 192.168.6.139:6443 check
```

Cuadro 33: Configuración de *HAProxy* completa

Luego, reiniciar los servicios de *haproxy* y verificar el *status* del mismo:

```
sudo systemctl restart haproxy
sudo systemctl status haproxy
```

Cuadro 34: Reinicio y comprobación de *HAProxy*

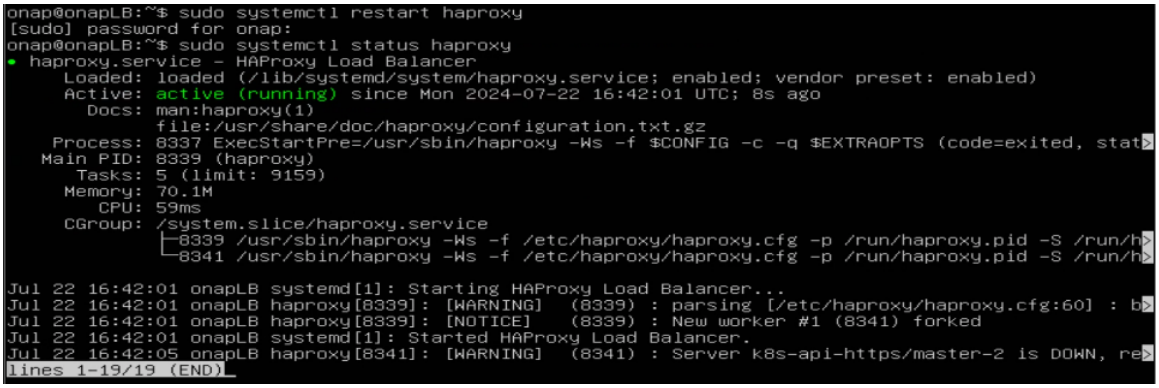


Figura 71: Comprobación del estado de *HAProxy*

Luego, modificar el archivo de configuración para *Keepalived*:

```
sudo nano /etc/keepalived/keepalived.conf
```

Cuadro 35: Acceso a configuración de *Keepalived*

Dentro de este archivo (el cual es muy probable que se encuentre vacío), colocar la siguiente configuración (Tomar en cuenta los comentarios dentro de la configuración ya que usualmente estos son los parámetros más importantes por cambiar):

```
global_defs {
    notification_email {
    }
    router_id onapLB1                    # Hostname de la maquina actual
    vrrp_skip_check_adv_addr
    vrrp_garp_interval 0
    vrrp_gna_interval 0
}

vrrp_script chk_haproxy {
    script "killall -0 haproxy"
    interval 2
    weight 2
}

vrrp_instance 50 {
    state MASTER                         # En VM master, state = MASTER
    priority 50                          # En VM backup, state = BACKUP
    interface enp0s3                     # Network card de nuestra VM
    virtual_router_id 60
    advert_int 1
    unicast_src_ip 192.168.6.143         # Direccion IP de maquina host
    unicast_peer {
        192.168.6.144                    # Direccion IP de maquina peer
    }

    virtual_ipaddress {
        192.168.6.145/25                 # Direccion IP virtual
```

```
    }

    track_script {
        chk_haproxy
    }
}
```

Cuadro 36: Configuración completa para *Keepalived*

Una vez guardamos este archivo, debemos realizar los siguientes comandos para habilitar el *Keepalived* y reiniciarlo. De esta forma, se ejecuta la configuración asignada:

```
sudo systemctl enable keepalived
sudo systemctl restart haproxy
sudo systemctl restart keepalived
```

Cuadro 37: Habilitar servicio de *Keepalived*

Una vez tenemos esto, logramos observar que en una, de nuestras 2 máquinas virtuales que funcionan como *Load Balancers*, nuestra interfaz de red va a tener 2 *IPs*, una dirección *IP* estática que es la asignada anteriormente, y otra *IP* que va a funcionar como la *IP* virtual que nos ayudará a mantener una redundancia en caso se caiga nuestro balanceador de carga principal.

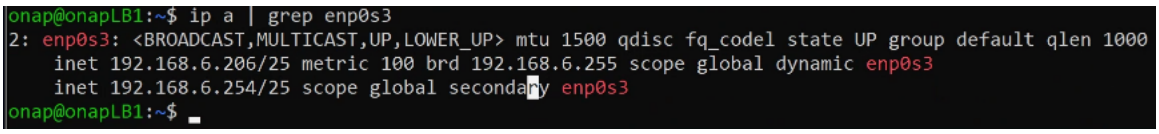


Figura 72: *IP* virtual asignada a la máquina virtual a través de *Keepalived*

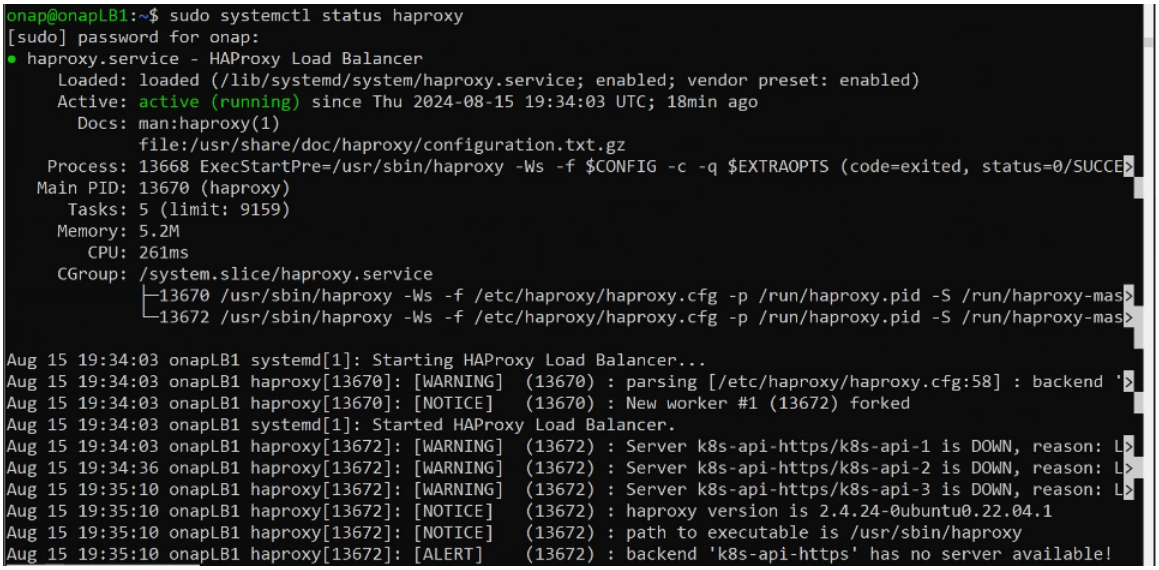


Figura 73: *Keepalived* funcionando correctamente

- Con esto terminado, ya podemos proceder a levantar y unir cada uno de nuestros nodos al cluster. Antes de hacer esto, es necesario deshabilitar swap en cada uno de los nodos del clúster. *Swap* es espacio de almacenamiento en el disco, el cual se utiliza como respaldo cuando la memoria RAM de un sistema se llena. *Kubernetes* no soporta *swap*, por lo que es necesario deshabilitarlo. Para deshabilitar *swap*, se debe correr el siguiente comando en cada uno de los nodos del clúster:

```
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Cuadro 38: Deshabilitar memoria *swap*

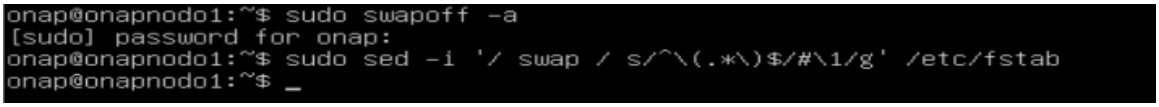


Figura 74: Deshabilitar memoria *swap*

- Ir a la máquina virtual que corresponde al primer nodo de control y levantar este primer nodo para unirlo al *control-plane endpoint*. A través de este primer nodo es que inicializaremos el cluster completo:

```
sudo kubeadm init --control-plane-endpoint "192.168.74.38:6443" --upload
    ↪ -certs --pod-network-cidr=192.168.0.0/16 --kubernetes-version v1
    ↪ .27.5 --cri-socket=unix:///var/run/cri-dockerd.sock

# Si existe configuracion de kubernetes previa, realizar los siguientes
    ↪ comandos:
sudo kubeadm reset --cri-socket=unix:///var/run/cri-dockerd.sock
sudo rm -rf /etc/cni/net.d
sudo rm -rf $HOME/.kube/config
```

Cuadro 39: Inicialización del clúster de *kubernetes*

- Si existe configuracion de *kubernetes* previa, realizar los siguientes comandos:

```
sudo kubeadm reset --cri-socket=unix:///var/run/cri-dockerd.sock
sudo rm -rf /etc/cni/net.d
sudo rm -rf $HOME/.kube/config
```

Cuadro 40: Reinicio de configuración de *Kubernetes* (por si existe configuración previa)

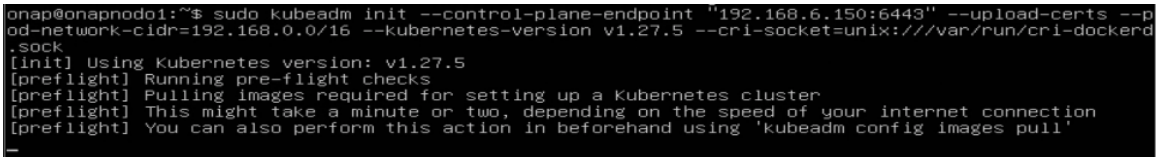


Figura 75: Inicialización del clúster de *Kubernetes*

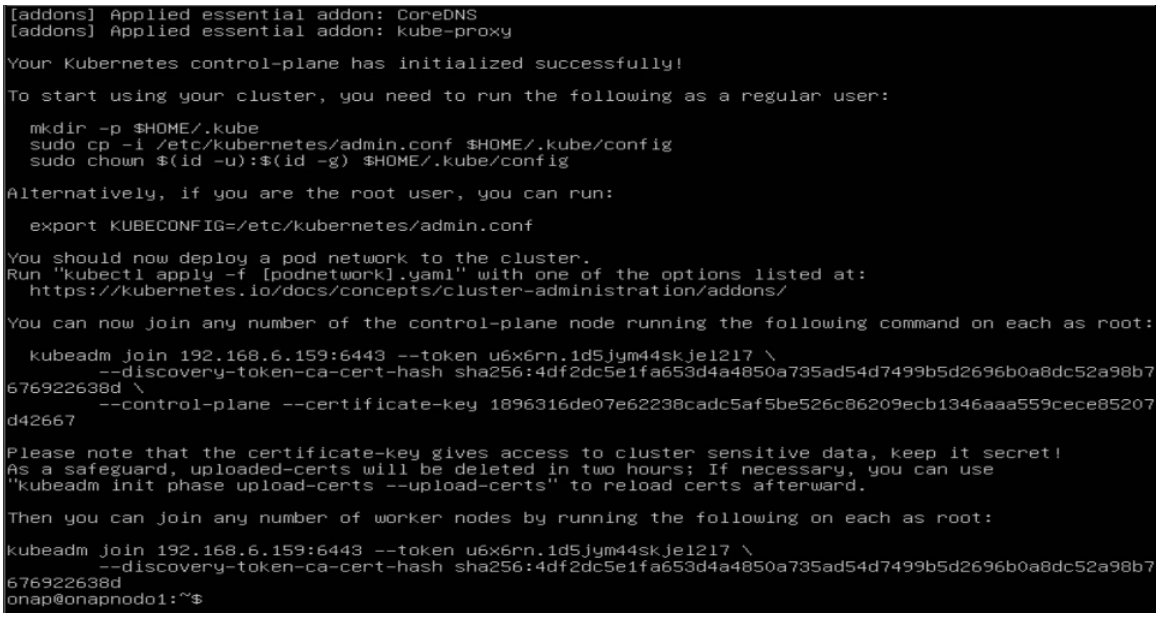


Figura 76: Clúster de *Kubernetes* inicializado

Esta configuración que nos aparece, la tenemos que guardar ya que con estas llaves y *tokens*, podremos unir los demás nodos a nuestro clúster.

- Para comenzar a utilizar nuestro clúster, debemos correr los siguientes comandos solamente en los nodos que formarán parte de nuestro *control plane*. Con estos comandos podremos utilizar nuestro clúster como usuario regular. Esto nos permite hacer comandos como *kubectl get nodes* y así, podremos visualizar los nodos del clúster y su estado actual:

```
1 mkdir -p $HOME/.kube
2 sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
3 sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Cuadro 41: Comandos para utilizar nuestro clúster

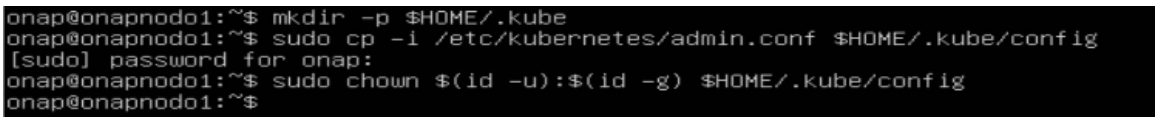


Figura 77: Comandos para utilizar nuestro clúster

- Luego, necesitaremos instalar un *network add-on* basado en un *Container Network Interface* (CNI) ya que de esta forma nuestros *pods* se podrán comunicar, entre sí, sin problema alguno. Este es un paso esencial, ya que queremos que nuestros *pods* (los cuales correrán todos los contenedores de las aplicaciones a desplegar) se comuniquen entre sí y así poder balancear su carga de trabajo en cada uno. En este caso utilizaremos *calico*, el cual tiene un *Classles Inter-Domain Routing* (CIDR) de 192.168.0.0/16:

```
curl https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/
    ↪ manifests/calico.yaml -O

kubectl apply -f calico.yaml
```

Cuadro 42: Instalar *calico*

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  247k  100  247k    0     0   357k      0 --:--:-- --:--:-- --:--:--  358k
onap@onapnodo1:~$ kubectl apply -f calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
serviceaccount/calico-cni-plugin created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgpfilters.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org cr
eated
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrole.rbac.authorization.k8s.io/calico-cni-plugin created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-cni-plugin created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
onap@onapnodo1:~$ _
```

Figura 78: Instalación de *calico*

- Una vez configuramos esto, debemos esperar un tiempo para que todos los *pods* de *calico* y los del clúster comiencen a correr sin problemas. Una vez esperamos un tiempo, podemos utilizar el siguiente comando para comprobar que todo está corriendo correctamente:

```
kubectl get pods --all-namespaces
```

Cuadro 43: Comprobación del funcionamiento de todos los *pods* de *calico* en nuestro clúster

```
onap@onapnodo1:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-8498bff86b-mx9ns   1/1     Running   0          14m
kube-system   calico-node-mkhjw                          1/1     Running   0          14m
kube-system   coredns-5d78c9869d-6n7jn                   1/1     Running   0          56m
kube-system   coredns-5d78c9869d-p8w6c                   1/1     Running   0          56m
kube-system   etcd-onapnodo1                             1/1     Running   0          56m
kube-system   kube-apiserver-onapnodo1                   1/1     Running   0          56m
kube-system   kube-controller-manager-onapnodo1          1/1     Running   0          56m
kube-system   kube-proxy-5hjmc                           1/1     Running   0          56m
kube-system   kube-scheduler-onapnodo1                   1/1     Running   0          56m
onap@onapnodo1:~$
```

Figura 79: *Pods* de *calico* corriendo en nuestro clúster

- Ahora es necesario unir los demás nodos al clúster. Siempre recordar que para cada nodo es necesario aplicar la configuración para no tener que correr todo como

root. También es importante quitar la opción de memoria *swap*. Los comandos a utilizar son los que logramos observar que aparecen al momento de inicializar nuestro clúster. Algo importante a remarcar es que siempre es necesario colocar la línea *–cri-socket=unix:///var/run/cri-dockerd.sock*, ya que esto es lo que nos permite utilizar *Docker* como nuestro *Container Runtime* y así evitar cualquier problema al unir los nodos al clúster. A continuación se presenta un resumen de diversos comandos útiles para unir nodos al clúster y para realizar otras configuraciones en el mismo:

```
# Previo a cualquier join (ya sea que se haga join de un control node o de un Worker node)
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab

# Para hacerle reset a cualquier configuracion previa
sudo kubeadm reset --cri-socket=unix:///var/run/cri-dockerd.sock
sudo rm -rf /etc/cni/net.d
sudo rm -rf $HOME/.kube/config

# Para unir nodos al Control-plane
sudo kubeadm join 192.168.74.38:6443 --token vkpr4p.3cqfqbvrmn8gaxco --discovery-token-ca-cert-hash sha256:b7769864cac2fd5e87cc9903e8c0c1e7962afc874cc647aef2bc2a030659c113 --control-plane --certificate-key 677a091e07e29b08cc19b1cb2ecf1f379bfa6762adfb48afd7c173a21498687e --cri-socket=unix:///var/run/cri-dockerd.sock

# Para unir nodos que funcionaran como Worker nodes
sudo kubeadm join 192.168.74.38:6443 --token vkpr4p.3cqfqbvrmn8gaxco --discovery-token-ca-cert-hash sha256:b7769864cac2fd5e87cc9903e8c0c1e7962afc874cc647aef2bc2a030659c113 --cri-socket=unix:///var/run/cri-dockerd.sock

# Configuracion para luego de unir el nodo al cluster (esto solo hacerlo en el nodo que querramos utilizar como controlador, siempre y cuando sea parte del control plane)
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

# En caso sean necesarias nuevas credenciales, realizar esto (Las credenciales vencen luego de cada cierto tiempo, por lo que es bueno realizar esto si mucho tiempo ha pasado)
sudo kubeadm init phase upload-certs --upload-certs

# En caso sea necesario renovar el token, realizar este comando:
sudo kubeadm token create --print-join-command
```

Cuadro 44: Comandos útiles al momento de unir nodos a nuestro clúster de *Kubernetes*

Al unir nodos al *control plane*, se debe de visualizar esto:

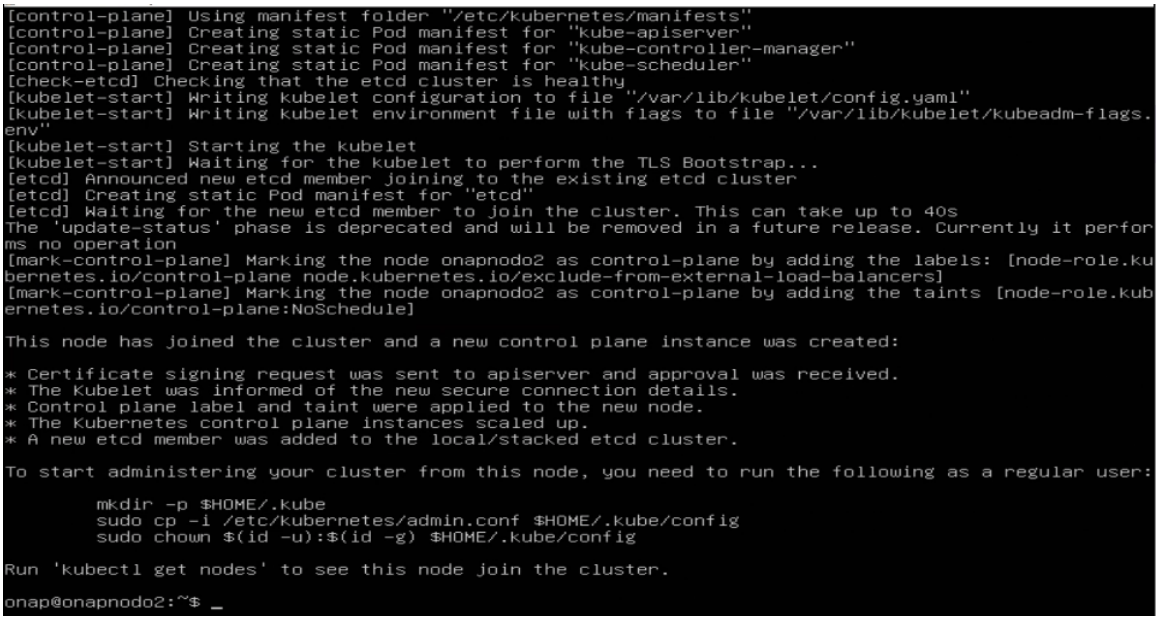


Figura 80: Unión de nodos al *control plane*

Por otro lado, al unir nodos que funcionarán como *Worker Nodes*, se debe visualizar esto:

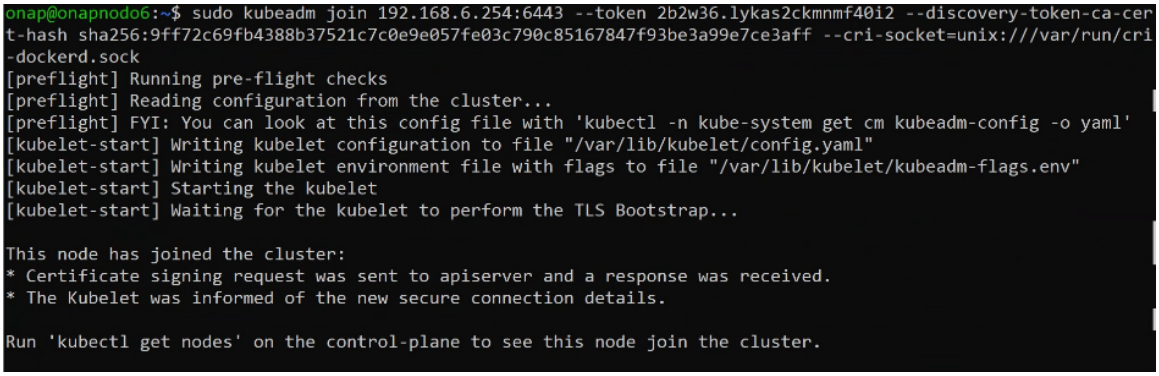


Figura 81: Unión de nodos como *Worker Nodes*

- Una vez unimos todos los nodos a nuestro clúster, podemos verificar que el clúster esté corriendo y funcionando de forma correcta al utilizar estos comandos:

```
kubectl get nodes
kubectl get pods -n kube-system
kubectl get pods --all-namespaces
```

Cuadro 45: Comprobación del correcto funcionamiento del clúster de *Kubernetes*

```
onap@onapnodo1:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE   VERSION
onapnodo1    Ready    control-plane   31m   v1.27.5
onapnodo2    Ready    control-plane   24m   v1.27.5
onapnodo3    Ready    control-plane   20m   v1.27.5
onapnodo4    Ready    <none>          17m   v1.27.5
onapnodo5    Ready    <none>          15m   v1.27.5
onapnodo6    Ready    <none>          14m   v1.27.5
```

Figura 82: Nodos del clúster de *Kubernetes*

```
onap@onapnodo1:~$ kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-8498bff86b-ql4jw   1/1     Running   0             4m12s
calico-node-5qmqq                          1/1     Running   0             4m12s
calico-node-cmj8f                          1/1     Running   0             4m12s
calico-node-js9zb                          1/1     Running   0             4m12s
calico-node-nnvkv                          1/1     Running   0             4m12s
calico-node-v49fd                          1/1     Running   0             4m12s
calico-node-w7kv4                          1/1     Running   0             4m12s
coredns-5d78c9869d-68dfc                   1/1     Running   0             31m
coredns-5d78c9869d-lj4ff                   1/1     Running   0             31m
etcd-onapnodo1                             1/1     Running   0             31m
etcd-onapnodo2                             1/1     Running   0             25m
etcd-onapnodo3                             1/1     Running   0             20m
kube-apiserver-onapnodo1                   1/1     Running   0             31m
kube-apiserver-onapnodo2                   1/1     Running   0             25m
kube-apiserver-onapnodo3                   1/1     Running   0             20m
kube-controller-manager-onapnodo1          1/1     Running   1 (24m ago)   31m
kube-controller-manager-onapnodo2          1/1     Running   0             24m
kube-controller-manager-onapnodo3          1/1     Running   0             20m
kube-proxy-bntjd                           1/1     Running   0             16m
kube-proxy-ds84z                           1/1     Running   0             25m
kube-proxy-mdgzv                           1/1     Running   0             17m
kube-proxy-nmkbp                           1/1     Running   0             31m
```

Figura 83: Pods corriendo en el clúster de *Kubernetes*

Con esto, logramos instalar todos los componentes y tener un clúster de *Kubernetes* funcional con alta disponibilidad.

7.3. Prueba de concepto 1: Intento fallido de despliegue de ONAP

Al momento en el que logramos tener nuestro clúster de *Kubernetes* listo y funcional, podemos proseguir con los siguientes pasos que nos llevarán a levantar una instancia de ONAP. Estos son los pasos que se siguieron, basándonos en la documentación oficial de ONAP. A pesar que el despliegue de ONAP no fue exitoso, diversos pre requisitos y configuraciones fueron realizadas, las cuales se describen en los pasos presentados a continuación.

ONAP es desplegado utilizando un componente del mismo, el cual se llama *ONAP Operations Manager* (OOM). El OOM nos permite desplegar una instancia de ONAP al utilizar el clúster de *Kubernetes* ya realizado, junto con *Docker* y una instalación de *Helm*.

Por esto mismo es que, previo a realizar cualquier tipo de despliegue de ONAP, es necesario revisar que ya tengamos listos los siguientes componentes de *software*:

- Clúster de *Kubernetes* con alta disponibilidad
- *Helm* instalado
- *Kubectl* instalado
- *Docker* instalado

Si tenemos todos los componentes de software necesarios, podemos proseguir a crear una plataforma base para instalar el OOM. Para esto utilizaremos la documentación oficial de ONAP [8].

Se irán escribiendo todos los comandos que se fueron realizando, ya que existen algunas cosas que se tuvieron que cambiar para que la instalación de la plataforma base de OOM fuera exitosa. El primer paso para ello es instalar y configurar *Helm*, lo cual se realizará a través de los siguientes pasos:

7.3.1. *Helm*

Primero será necesario ejecutar los siguientes comandos para la instalación de *Helm*. Se coloca la version 3.12.3 ya que se lanzara una instancia de ONAP *Montreal*, el cual solicita utilizar *v*ersión 3.12.3:

```
wget https://get.helm.sh/helm-v3.12.3-linux-amd64.tar.gz
tar -zxvf helm-v3.12.3-linux-amd64.tar.gz
sudo mv linux-amd64/helm /usr/local/bin/helm
```

Cuadro 46: Descargar paquetes para la instalación de *Helm*

Figura 84: Descarga de paquetes para la instalación de *Helm*

```
onap@onapnodo1:~$ tar -zxvf helm-v3.12.3-linux-amd64.tar.gz
linux-amd64/
linux-amd64/LICENSE
linux-amd64/README.md
linux-amd64/helm
onap@onapnodo1:~$ sudo mv linux-amd64/helm /usr/local/bin/helm
[sudo] password for onap:
onap@onapnodo1:~$
```

Figura 85: Instalación de *Helm*

Una vez terminamos de hacer estos comandos, podemos utilizar el siguiente comando para comprobar una correcta instalación de *Helm* y su versión correspondiente:

```
helm version
```

Cuadro 47: Chequear versión de *Helm* instalada

```
onap@onapnodo1:~$ helm version
version.BuildInfo{Version:"v3.12.3", GitCommit:"3a31588ad33fe3b89af5a2a54ee1d25bfe6eaa5e", GitTreeState:"clean", GoVersi
on:"go1.20.7"}
onap@onapnodo1:~$
```

Figura 86: Chequeo de versión de *Helm*

Adicional a esto, es bueno realizar el siguiente comando, ya que como nos indica la documentación de ONAP [8]:

> *"Helm's default CNCF provided Curated applications for Kubernetes repository called stable can be removed to avoid confusion."*

```
helm repo remove stable
```

Cuadro 48: Eliminar repositorio *"stable"*

Con esto, ya tenemos helm listo para desplegar ONAP a través a través del OOM. Antes de ello, será necesario instalar todos los *plugins* adicionales para desplegar o desinstalar los *Helm charts* del OOM:

```
git clone https://gerrit.onap.org/r/oom
helm plugin install ~/oom/kubernetes/helm/plugins/deploy
helm plugin install ~/oom/kubernetes/helm/plugins/undeploy
helm plugin ls
```

Cuadro 49: Instalación de repositorio de OOM y de *plugins deploy* y *undeploy*

```
onap@onapnodo1:~$ git clone https://gerrit.onap.org/r/oom
Cloning into 'oom'...
remote: Counting objects: 16, done
remote: Total 94985 (delta 0), reused 94985 (delta 0)
Receiving objects: 100% (94985/94985), 40.07 MiB | 2.95 MiB/s, done.
Resolving deltas: 100% (72016/72016), done.
```

Figura 87: Clonar repositorio de OOM

```
onap@onapnodo1:~$ helm plugin install ~/oom/kubernetes/helm/plugins/deploy
Installed plugin: deploy
onap@onapnodo1:~$ helm plugin install ~/oom/kubernetes/helm/plugins/undeploy
Installed plugin: undeploy
```

Figura 88: Instalación de *plugins deploy* y *undeploy*

```
onap@onapnodo1:~$ helm plugin ls
NAME            VERSION DESCRIPTION

deploy          1.0.0   install (upgrade if release exists) parent charty and all subcharts as separate but related rele
ases
undeploy        1.0.0   delete parent chart and subcharts that were deployed as separate releases
```

Figura 89: Chequeo de *plugins* instalados

Una vez tenemos esto instalado, procedemos a seguir instalando los demás componentes para la plataforma base del OOM.

7.3.2. Creación de NFS para funcionar como un *StorageClass*

Para la instancia de ONAP que se desplegará, se busca tener componentes de mucho análisis de datos o de realización de bases de datos sobre una red en específico. Por esto mismo, es necesario tener un *StorageClass* para nuestro clúster de *Kubernetes*. Los *StorageClass* se usan para tener un almacenamiento centralizado para todos los nodos del clúster. Estos son volúmenes persistentes que, a pesar que algún contenedor o *pod* quede inutilizado, este almacenamiento permanece ahí sin problema alguno, para que todos los demás nodos puedan seguir utilizando el mismo. Al ser volúmenes persistentes, trabajan más allá del tiempo de vida de algún contenedor o *pod*.

En nuestro caso, configuraremos y emplearemos un NFS (*Network File System*) server y así, de esa manera, tener nuestros volúmenes persistentes. Para esto, utilizaremos una máquina virtual dedicada para dicho servidor. A contiuación se presentan todos los pasos para habilitar este servidor NFS luego de haber levantado una máquina virtual dedicada para esto:

- Hacer update a *apt* y luego instalar el *nfs-server* en la máquina dedicada para el servidor NFS:

```
1 sudo apt-get update
2 sudo apt-get install nfs-common nfs-kernel-server -y
```

Cuadro 50: Instalar *nfs-kernel-server*

- Crear un directorio para poder exportar el servidor NFS

```
sudo mkdir -p /data/nfs
sudo chown nobody:nogroup /data/nfs
sudo chmod 2770 /data/nfs
```

Cuadro 51: Crear directorio a exportar

```
onap@onapNFS:~$ sudo apt-get install nfs-common nfs-kernel-server -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nfs-common is already the newest version (1:2.6.1-1ubuntu1.2).
nfs-common set to manually installed.
nfs-kernel-server is already the newest version (1:2.6.1-1ubuntu1.2).
0 upgraded, 0 newly installed, 0 to remove and 5 not upgraded.
onap@onapNFS:~$ sudo mkdir -p /data/nfs
onap@onapNFS:~$ sudo chown nobody:nogroup /data/nfs
onap@onapNFS:~$ sudo chmod 2770 /data/nfs
```

Figura 90: Creación de directorio a exportar

- Exportar el directorio

```
echo -e "/data/nfs\t192.168.72.0/22(rw,sync,no_subtree_check,
    ↪ no_root_squash)" | sudo tee -a /etc/exports

sudo exportfs -av
```

Cuadro 52: Exportar directorio creado

```
onap@onapNFS:~$ echo -e "/data/nfs\t192.168.6.128/25(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -a /etc/export
s
/data/nfs       192.168.6.128/25(rw,sync,no_subtree_check,no_root_squash)
onap@onapNFS:~$ sudo exportfs -av
exporting 192.168.6.128/25:/data/nfs
```

Figura 91: Exportar directorio creado

- Reiniciar y verificar el estado del servidor NFS

```
onap@onapNFS:~$ sudo systemctl restart nfs-kernel-server
onap@onapNFS:~$ sudo systemctl status nfs-kernel-server
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
     Active: active (exited) since Sun 2024-08-18 20:24:57 UTC; 7s ago
    Process: 37854 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
    Process: 37855 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
   Main PID: 37855 (code=exited, status=0/SUCCESS)
        CPU: 3ms

Aug 18 20:24:57 onapNFS systemd[1]: Starting NFS server and services...
Aug 18 20:24:57 onapNFS systemd[1]: Finished NFS server and services.
```

Figura 92: Reinicio y verificación del estado del servidor NFS

- Comprobar los detalles de la exportación del directorio:

```
/sbin/showmount -e 192.168.74.39
```

Cuadro 53: Chequeo de detalles de exportación

```
onap@onapNFS:~$ /sbin/showmount -e 192.168.6.251
Export list for 192.168.6.251:
/data/nfs 192.168.6.128/25
```

Figura 93: Detalles de exportación

- Una vez se tenga todo esto realizado, podemos avanzar a instalar los siguientes paquetes en cada uno de los nodos de nuestro clúster (es esencial que antes de avanzar, este paso se haga en todos los nodos del clúster para evitar problemas posteriores):

```
sudo apt update
sudo apt install nfs-common -y
```

Cuadro 54: Instalar paquete *nsf-common*

```
onap@onapnodo1:~$ sudo apt install nfs-common -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  keyutils libnfsidmap1 rpcbind
Suggested packages:
  watchdog
The following NEW packages will be installed:
  keyutils libnfsidmap1 nfs-common rpcbind
0 upgraded, 4 newly installed, 0 to remove and 20 not upgraded.
Need to get 381 kB of archives.
After this operation, 1,447 kB of additional disk space will be used.
Ign:1 http://gt.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libnfsidmap1 amd64 1:2.6.1-1ubuntu1.2
Get:2 http://gt.archive.ubuntu.com/ubuntu jammy/main amd64 rpcbind amd64 1.2.6-2build1 [46.6 kB]
Get:3 http://gt.archive.ubuntu.com/ubuntu jammy/main amd64 keyutils amd64 1.6.1-2ubuntu3 [50.4 kB]
Get:4 http://gt.archive.ubuntu.com/ubuntu jammy-updates/main amd64 nfs-common amd64 1:2.6.1-1ubuntu1.2 [241 kB]
Get:1 http://gt.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libnfsidmap1 amd64 1:2.6.1-1ubuntu1.2 [42.9 kB]
Fetched 381 kB in 41s (9,242 B/s)
Selecting previously unselected package libnfsidmap1:amd64.
(Reading database ... 129065 files and directories currently installed.)
Preparing to unpack .../libnfsidmap1_1%3a2.6.1-1ubuntu1.2_amd64.deb ...
Unpacking libnfsidmap1:amd64 (1:2.6.1-1ubuntu1.2) ...
Selecting previously unselected package rpcbind.
Preparing to unpack .../rpcbind_1.2.6-2build1_amd64.deb ...
```

Figura 94: Instalación de paquete *nfs-common*

- Luego, en uno de los nodos de control (en nuestro caso el nodo 1), se deberá de añadir el repositorio de *Helm* para el *nfs-subdir-external-provisioner*:

```
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.
    ↪ github.io/nfs-subdir-external-provisioner/
```

Cuadro 55: Añadir repositorio *nfs-subdir-external-provisioner*

```
onap@onapnodo1:~$ helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-pr
ovisioner/
"nfs-subdir-external-provisioner" has been added to your repositories
```

Figura 95: Añadir repositorio *nfs-subdir-external-provisioner*

- Una vez se tiene esto, procederemos a instalar el *Helm Chart* para NFS, teniendo cuidado de indicar la *IP* correcta en la línea *–set nfs.server="IP del servidor NFS"*

```
helm install nfs-subdir-external-provisioner nfs-subdir-external-
    ↪ provisioner/nfs-subdir-external-provisioner --create-namespace --
    ↪ namespace nfs-provisioner --set nfs.server=192.168.74.39 --set nfs
    ↪ .path=/data/nfs --set storageClass.onDelete=true
```

Cuadro 56: Instalación de *Helm Chart* para el servidor NFS

```
onap@onapnodo1:~$ helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provi
sioner --create-namespace --namespace nfs-provisioner --set nfs.server=192.168.6.251 --set nfs.path=/data/nfs --set stor
ageClass.onDelete=true
NAME: nfs-subdir-external-provisioner
LAST DEPLOYED: Sun Aug 18 20:44:40 2024
NAMESPACE: nfs-provisioner
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

Figura 96: Instalación de *Helm Chart* para el servidor NFS

- Con esto, podemos comprobar que el *StorageClass* configurado, se encuentra levantado y corriendo de forma correcta a través de los siguientes comandos:

```
kubectl get sc
```

Cuadro 57: Obtener *storageClasses* dentro del clúster

```
onap@onapnodo1:~$ kubectl get sc
NAME         PROVISIONER                                    RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION
AGE
nfs-client   cluster.local/nfs-subdir-external-provisioner   Delete          Immediate           true
17s
```

Figura 97: *StorageClasses* dentro del clúster

- Otro punto importante es que, para que nuestros clúster de *kubernetes* pueda solicitar almacenamiento al momento de requerirlo, vamos a necesitar hacer 2 archivos .yaml, uno para *Persistent Volumes* (PV) y otro para *Persistent Volume Claim* (PVC). A través del archivo PV, el servidor NFS provee almacenamiento a los nodos del clúster para cuando lo requieran. Ahora, mediante el archivo PVC es como los *pods* pueden hacer solicitudes de almacenamiento extra cuando lo necesiten. Por ello, es necesario crear los archivos .yaml:

```
sudo nano pv.yaml
sudo nano pvc.yaml
```

Cuadro 58: Archivos para *Persistent Volumes* y *Persistent Volume Claims*

- En el archivo pv.yaml es necesario colocar la siguiente información:

```
apiVersion: v1
kind: PersistentVolume
metadata:
    name: k8s-pv
spec:
    accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
    - ReadWriteMany
    capacity:
    storage: 20Gi
    storageClassName: nfs-client
    persistentVolumeReclaimPolicy: Recycle
    volumeMode: Filesystem
    nfs:
    server: 192.168.74.39
    path: /data/nfs
    readOnly: no
```

Cuadro 59: Archivo pv.yaml

- Y en el archivo pvc.yaml es necesario colocar la siguiente información:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
    name: k8s-pvc
spec:
    accessModes:
    - ReadWriteMany
    resources:
    requests:
        storage: 2Gi
    storageClassName: nfs-client
    volumeName: k8s-pv
```

Cuadro 60: Archivo pvc.yaml

- Una vez se tiene esto, se aplican ambos archivos

```
kubectl apply -f pv.yaml
kubectl apply -f pvc.yaml
```

Cuadro 61: Aplicar archivos pv.yaml y pvc.yaml al clúster de *Kubernetes*

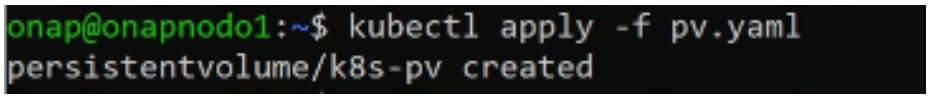


Figura 98: Aplicación de archivo pv.yaml

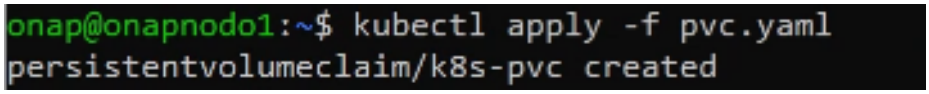


Figura 99: Aplicación de archivo pvc.yaml

- Ahora podemos comprobar que nuestro PV está corriendo de forma correcta mediante el siguiente comando. Es bueno aclarar que la pestaña que dice *"STATUS"*, debería de decir *"Bound"*, ya que este PV está unido al archivo PVC, lo cual nos indica que ambos archivos funcionaron de forma correcta:

```
kubectl get pv
```

Cuadro 62: Obtener *Persistent Volumes* en el clúster

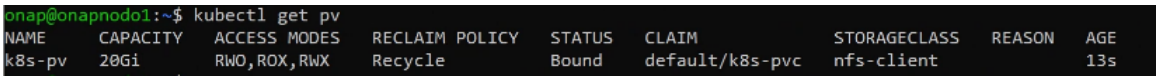


Figura 100: *Persistent Volumes* en el clúster

- Por último paso, es buena idea asginar el *StorageClass* creado, por defecto:

```
kubectl patch storageclass nfs-client -p '{"metadata": {"annotations":{"
    ↪ storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Cuadro 63: Configurar *Storage Class nfs-client* como *"default"*

```
onap@onapnodo1:~$ kubectl patch storageclass nfs-client -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-
default-class":"true"}}}'
storageclass.storage.k8s.io/nfs-client patched
```

Figura 101: Configuración de *Storage Class* como *"default"*

Con esto, tendríamos un *StorageClass* creado junto con un servidor NFS para almacenamiento centralizado para todos los nodos de nuestro clúster.

7.3.3. *Strimzi Kafka Operator*

- Añadir repositorio de *Helm*

```
helm repo add strimzi https://strimzi.io/charts/
```

Cuadro 64: Añadir el repo de *Helm* de *Strimzi*

```
onap@onapnodo1:~$ helm repo add strimzi https://strimzi.io/charts/
"strimzi" has been added to your repositories
```

Figura 102: Añadir repositorio de *Helm* de *Strimzi*

- Instalar *Strimzi*

```
helm install strimzi-kafka-operator strimzi/strimzi-kafka-operator --
    ↪ namespace strimzi-system --version 0.36.1 --set watchAnyNamespace=
    ↪ true --create-namespace
```

Cuadro 65: Instalar *Strimzi*

```
onap@onapnodo1:~$ helm install strimzi-kafka-operator strimzi/strimzi-kafka-operator --namespace strimzi-system --versio
n 0.36.1 --set watchAnyNamespace=true --create-namespace
NAME: strimzi-kafka-operator
LAST DEPLOYED: Mon Aug 19 00:44:09 2024
NAMESPACE: strimzi-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing strimzi-kafka-operator-0.36.1

To create a Kafka cluster refer to the following documentation.

https://strimzi.io/docs/operators/latest/deploying.html#deploying-cluster-operator-helm-chart-str
```

Figura 103: Instalación de *Strimzi*

- Chequear la instalación:

```
kubectl get po -n strimzi-system
```

Cuadro 66: Chequear *pods* de *Strimzi* corriendo en el clúster

```
onap@onapnodo1:~$ kubectl get po -n strimzi-system
NAME                                        READY   STATUS    RESTARTS   AGE
strimzi-cluster-operator-64f9bf48c9-gbrl9   1/1     Running   0          3m26s
```

Figura 104: *Pods* de *Strimzi* corriendo en el clúster

Con esto, podemos confirmar que se instaló de forma correcta *Strimzi Kafka*

7.3.4. *Cert Manager*

- Instalar el *cert manager*:

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/
    ↪ download/v1.12.2/cert-manager.yaml
```

Cuadro 67: Aplicar archivo *cert-manager.yaml* al clúster

```
onap@onapnodo1:~$ kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.13.2/cert-manager.yaml
namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
configmap/cert-manager created
configmap/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
```

Figura 105: Aplicación de archivo *cert-manager.yaml*

- Verificar que todo esté corriendo de forma correcta:

```
kubectl get po -n cert-manager
```

Cuadro 68: Obtener *pods* del *cert-manager* en el clúster

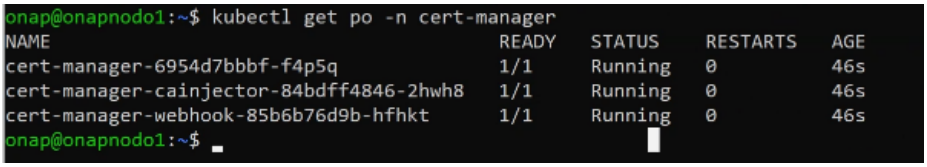


Figura 106: *Pods* de *Cert Manager* corriendo en el clúster

Con esto, podemos confirmar que se instaló de forma correcta *Cert Manager*

7.3.5. Instalar *Isitio service Mesh*

```
helm repo add istio https://istio-release.storage.googleapis.com/charts
helm repo update
```

Cuadro 69: Añadir repositorio de *Helm* de *Istio*

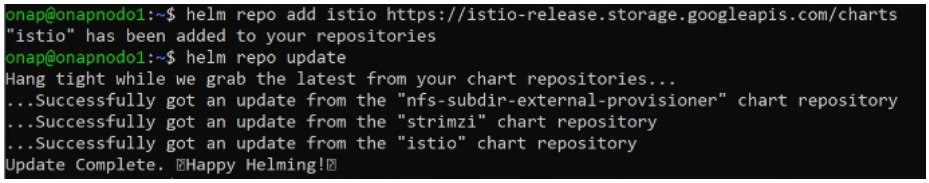


Figura 107: Añadir repositorio de *Helm* de *Istio*

- Crear *namespaces*

```
kubectl create namespace istio-config
kubectl create namespace istio-system
```

Cuadro 70: Creación de *namespaces "istio-config"* e *"istio-system"*

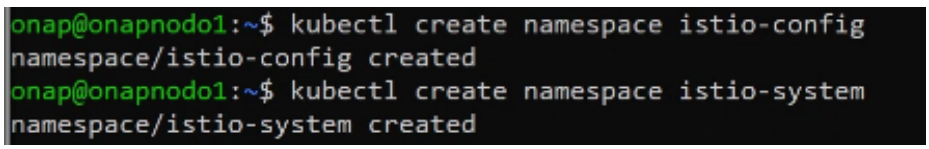


Figura 108: *Namespaces* creados

- Instalar el *Istio Base chart*

```
helm upgrade -i istio-base istio/base -n istio-system --version 1.17.2
```

Cuadro 71: Instalar *Istio Base Chart*

```
onap@onapnodo1:~$ helm upgrade -i istio-base istio/base -n istio-system --version 1.19.3
Release "istio-base" does not exist. Installing it now.
NAME: istio-base
LAST DEPLOYED: Mon Aug 19 01:02:23 2024
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Istio base successfully installed!

To learn more about the release, try:
  $ helm status istio-base
  $ helm get all istio-base
```

Figura 109: Instalación de *Istio Base Chart*

- Crear el archivo *istiod.yaml* e ingresar los siguientes parámetros:

```yaml
global:
proxy:
    # Controls if sidecar is injected at the front of the container list and
        ↪ blocks the start of the other containers until the proxy is ready
    holdApplicationUntilProxyStarts: true
#logging:
#  level: "default:debug"
meshConfig:
rootNamespace: istio-config
extensionProviders:
- name: oauth2-proxy
    envoyExtAuthzHttp:
    service: oauth2-proxy.default.svc.cluster.local
    port: 80
    timeout: 1.5s
    includeHeadersInCheck: ["authorization", "cookie"]
    headersToUpstreamOnAllow: ["x-forwarded-access-token", "authorization", "
        ↪ path", "x-auth-request-user", "x-auth-request-email", "x-auth-
        ↪ request-access-token"]
    headersToDownstreamOnDeny: ["content-type", "set-cookie"]
pilot:
env:
    PILOT_HTTP10: true
```

Cuadro 72: Valores del archivo *istiod.yaml*

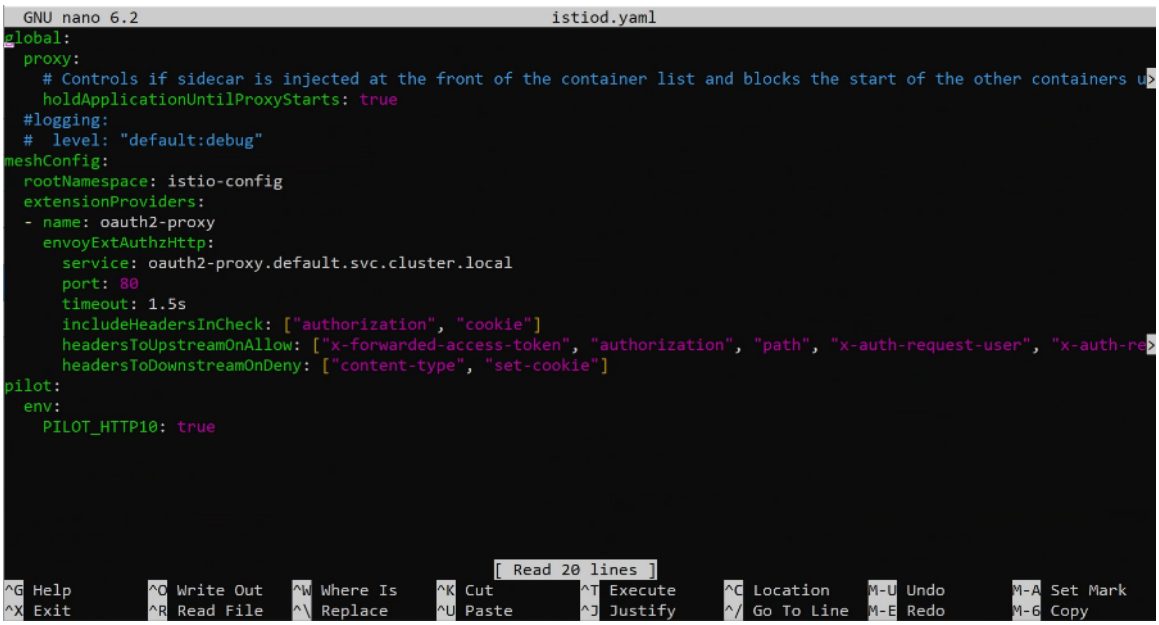


Figura 110: Archivo *istiod.yaml*

- Instalar el archivo yaml para instalar el *Istio Base Discovery chart*

```
helm upgrade -i istiod istio/istiod -n istio-system --version 1.17.2 --wait -
    ↪ f ./istiod.yaml
```

Cuadro 73: Instalación de archivo *istiod.yaml*

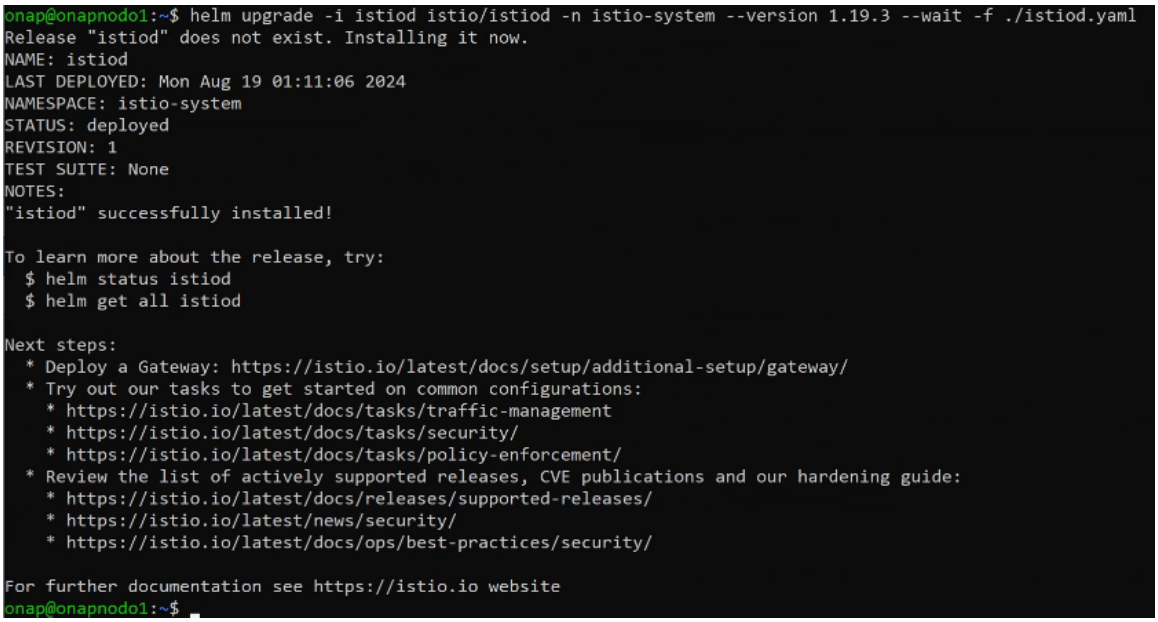


Figura 111: Archivo *istiod.yaml* instalado

- Crear un *EnvoyFilter file*

```
sudo nano envoyfilter-case.yaml
```

Cuadro 74: Creación de archivo *envoyfilter-case.yaml*

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
    name: header-casing
    namespace: istio-config
spec:
    configPatches:
    - applyTo: CLUSTER
    match:
        context: SIDECAR_INBOUND
    patch:
        operation: MERGE
        value:
        typed_extension_protocol_options:
            envoy.extensions.upstreams.http.v3.HttpProtocolOptions:
            '@type': type.googleapis.com/envoy.extensions.upstreams.http.v3.
                ↪ HttpProtocolOptions
            use_downstream_protocol_config:
                http_protocol_options:
                header_key_format:
                    stateful_formatter:
                    name: preserve_case
                    typed_config:
                        '@type': type.googleapis.com/envoy.extensions.http.
                            ↪ header_formatters.preserve_case.v3.
                            ↪ PreserveCaseFormatterConfig
    - applyTo: NETWORK_FILTER
    match:
        listener:
        filterChain:
            filter:
            name: envoy.filters.network.http_connection_manager
    patch:
        operation: MERGE
        value:
        typed_config:
            '@type': type.googleapis.com/envoy.extensions.filters.network.
                ↪ http_connection_manager.v3.HttpConnectionManager
            http_protocol_options:
            header_key_format:
                stateful_formatter:
                name: preserve_case
                typed_config:
                    '@type': type.googleapis.com/envoy.extensions.http.
                        ↪ header_formatters.preserve_case.v3.
                        ↪ PreserveCaseFormatterConfig
---
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
    name: header-casing-outbound
```

```
    namespace: istio-config
    #annotations:
    #  argocd.argoproj.io/hook: PostSync
spec:
    configPatches:
    - applyTo: CLUSTER
    match:
        context: SIDECAR_OUTBOUND
    patch:
        operation: MERGE
        value:
        typed_extension_protocol_options:
            envoy.extensions.upstreams.http.v3.HttpProtocolOptions:
            '@type': type.googleapis.com/envoy.extensions.upstreams.http.v3.
                ↪ HttpProtocolOptions
            use_downstream_protocol_config:
                http_protocol_options:
                header_key_format:
                    stateful_formatter:
                    name: preserve_case
                    typed_config:
                        '@type': type.googleapis.com/envoy.extensions.http.
                            ↪ header_formatters.preserve_case.v3.
                            ↪ PreserveCaseFormatterConfig
    - applyTo: NETWORK_FILTER
    match:
        listener:
        filterChain:
            filter:
            name: envoy.filters.network.http_connection_manager
    patch:
        operation: MERGE
        value:
        typed_config:
            '@type': type.googleapis.com/envoy.extensions.filters.network.
                ↪ http_connection_manager.v3.HttpConnectionManager
            http_protocol_options:
            header_key_format:
                stateful_formatter:
                name: preserve_case
                typed_config:
                    '@type': type.googleapis.com/envoy.extensions.http.
                        ↪ header_formatters.preserve_case.v3.
                        ↪ PreserveCaseFormatterConfig
```

Cuadro 75: Datos del archivo *envoyfilter-case.yaml*

Luego de añadir esto al archivo, hay que aplicarlo con *kubectl*

```
kubectl apply -f envoyfilter-case.yaml
```

Cuadro 76: Aplicar archivo *envoyfilter-case.yaml* al clúster

```
onap@onapnodo1:~$ onap@onapnodo1:~$ kubectl apply -f envoyfilter-case.yaml
envoyfilter.networking.istio.io/header-casing created
envoyfilter.networking.istio.io/header-casing-outbound created
```

Figura 112: Aplicación de archivo *envoyfilter-case.yaml*

7.3.6. Instalar el *Gateway-API*

- Instalar el repo de *Gateway-API*

```
kubectl apply -f https://github.com/kubernetes-sigs/gateway-api/releases
    ↪ /download/v0.6.2/experimental-install.yaml
```

Cuadro 77: Aplicar archivo de *gateway-api* al clúster

```
onap@onapnodo1:~$ kubectl apply -f https://github.com/kubernetes-sigs/gateway-api/releases/download/v1.0.0/experimental-
install.yaml
customresourcedefinition.apiextensions.k8s.io/backendtlspolicies.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/gatewayclasses.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/gateways.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/grpcroutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/httproutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/referencegrants.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/tcproutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/tlsroutes.gateway.networking.k8s.io configured
customresourcedefinition.apiextensions.k8s.io/udproutes.gateway.networking.k8s.io configured
```

Figura 113: Aplicación de archivo de *gateway-api*

- Crear el archivo *common-gateway.yaml* e instalar el mismo

```
sudo nano common-gateway.yaml
kubectl create namespace istio-ingress
kubectl apply -f common-gateway.yaml
```

Cuadro 78: Creación e instalación de archivo *"common-gateway.yaml"*

- Dentro del archivo .yaml colocar lo siguiente:

```
apiVersion: gateway.networking.k8s.io/v1beta1
kind: Gateway
metadata:
    name: common-gateway
    namespace: istio-ingress
spec:
    gatewayClassName: istio
    listeners:
    - name: http-80
        hostname: "*.simpledemo.onap.org"
        port: 80
        protocol: HTTP
        allowedRoutes:
        namespaces:
```

```
            from: All
    - name: https-443
        hostname: "*.simpledemo.onap.org"
        port: 443
        protocol: HTTPS
        allowedRoutes:
        namespaces:
            from: All
        tls:
        mode: Terminate
        certificateRefs:
            - kind: Secret
            group: ""
            name: ingress-tls-secret
    - name: udp-162
        protocol: UDP
        port: 162
        allowedRoutes:
        kinds:
            - kind: UDPRoute
        namespaces:
            from: All
    - name: tcp-4334
        protocol: TCP
        port: 4334
        allowedRoutes:
        kinds:
            - kind: TCPRoute
        namespaces:
            from: All
    - name: tcp-9000
        allowedRoutes:
        namespaces:
            from: All
        hostname: "kafka-api.simpledemo.onap.org"
        port: 9000
        protocol: TLS
        tls:
        certificateRefs:
            - group: ""
            kind: Secret
            name: ingress-tls-secret
        mode: Terminate
    - name: tcp-9001
        allowedRoutes:
        namespaces:
            from: All
        hostname: "kafka-api.simpledemo.onap.org"
        port: 9001
        protocol: TLS
        tls:
        certificateRefs:
            - group: ""
            kind: Secret
            name: ingress-tls-secret
        mode: Terminate
    - name: tcp-9002
        allowedRoutes:
        namespaces:
```

```
            from: All
        hostname: "kafka-api.simpledemo.onap.org"
        port: 9002
        protocol: TLS
        tls:
        certificateRefs:
            - group: ""
            kind: Secret
            name: ingress-tls-secret
        mode: Terminate
    - name: tcp-9010
        allowedRoutes:
        namespaces:
            from: All
        hostname: "kafka-bootstrap-api.simpledemo.onap.org"
        port: 9010
        protocol: TLS
        tls:
        certificateRefs:
            - group: ""
            kind: Secret
            name: ingress-tls-secret
        mode: Terminate
```

Cuadro 79: Valores del archivo *common-gateway.yaml*

```
onap@onapnodo1:~$ kubectl apply -f common-gateway.yaml
gateway.gateway.networking.k8s.io/common-gateway created
```

Figura 114: Archivo *common-gateway.yaml* aplicado al clúster

7.3.7. Instalar *Keycloak*

- Instalar los repositorios de *Helm*

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add codecentric https://codecentric.github.io/helm-charts
helm repo update
```

Cuadro 80: Añadir repositorios de *Helm* para *"Bitnami"* y *"Codecentric"*

```
onap@onapnodo1:~$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
```

Figura 115: Añadir repositorios de *Helm*

```
onap@onapnodo1:~$ helm repo add codecentric https://codecentric.github.io/helm-charts
"codecentric" has been added to your repositories
```

Figura 116: Añadir repositorios de *Helm*

```
onap@onapnodo1:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "nfs-subdir-external-provisioner" chart repository
...Successfully got an update from the "codecentric" chart repository
...Successfully got an update from the "istio" chart repository
...Successfully got an update from the "strimzi" chart repository
...Successfully got an update from the "bitnami" chart repository
Update Complete. ⎈Happy Helming!⎈
```

Figura 117: Actualización de repositorios de *Helm*

- Crear el *namespace* para *Keycloak*

```
kubectl create namespace keycloak
kubectl label namespace keycloak istio-injection=disabled
```

Cuadro 81: Crear *namespace "keycloak"*

```
onap@onapnodo1:~$ kubectl create namespace keycloak
namespace/keycloak created
onap@onapnodo1:~$ kubectl label namespace keycloak istio-injection=disabled
namespace/keycloak labeled
```

Figura 118: Creación de *namespace "keycloak"*

- Instalar la base de datos de *Keycloak*

```
sudo nano keycloak-db-values.yaml
```

Cuadro 82: Creación de archivo *"keycloak-db-values.yaml"*

- Dentro de este archivo colocar los siguientes valores:

```
global:
    postgresql:
    auth:
        username: dbusername
        password: dbpassword
        database: keycloak
```

Cuadro 83: Datos en el archivo *"keycloak-db-values.yaml"*

- Luego de esto, instalar la base de datos de *Keycloak*

```
helm -n keycloak upgrade -i keycloak-db bitnami/postgresql --values ./
    ↪ keycloak-db-values.yaml
```

Cuadro 84: Instalar archivo para instalar base de datos de *Keycloak*

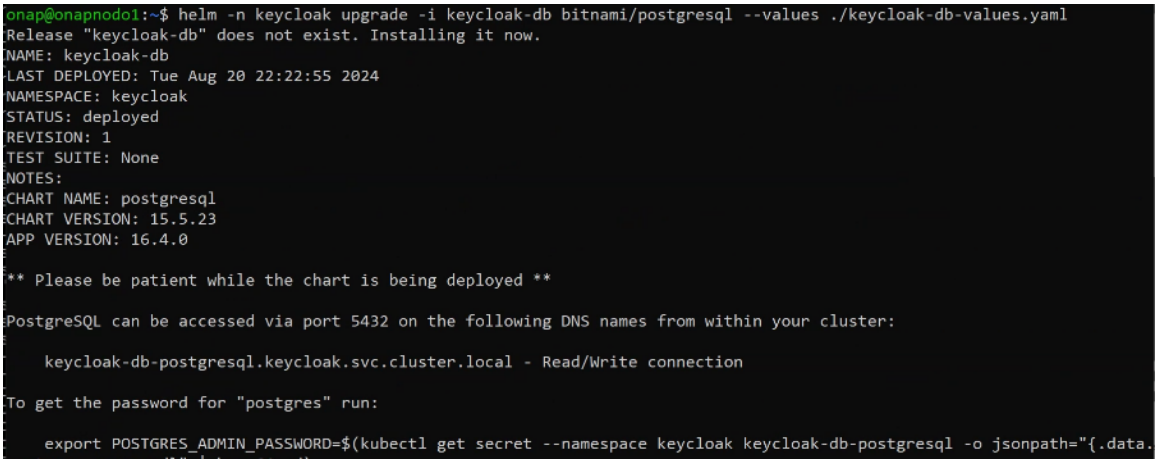


Figura 119: Instalación de base de datos de *Keycloak*

- Esto nos devuelve los siguientes datos que nos pueden ser útiles luego:

```
** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

    keycloak-db-postgresql.keycloak.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

    export POSTGRES_ADMIN_PASSWORD=$(kubectl get secret --namespace keycloak keycloak-db-postgresql -o jsonpath="{.data.postgres-password}" | base64 -d)

To get the password for "dbusername" run:

    export POSTGRES_PASSWORD=$(kubectl get secret --namespace keycloak keycloak-db-postgresql -o jsonpath="{.data.password}" | base64 -d)

To connect to your database run the following command:

    kubectl run keycloak-db-postgresql-client --rm --tty -i --restart='Never' --namespace keycloak --image docker.io/bitnami/postgresql:16.4.0-debian-12-r2 --env="PGPASSWORD=$POSTGRES_PASSWORD" \
      --command -- psql --host keycloak-db-postgresql -U dbusername -d keycloak -p 5432

    > NOTE: If you access the container using code_structure, make sure that you execute "/opt/bitnami/scripts/postgresql/entrypoint.sh /bin/bash" in order to avoid the error "psql: local user with ID 1001} does not exist"

To connect to your database from outside the cluster execute the following commands:

```

```
    kubectl port-forward --namespace keycloak svc/keycloak-db-postgresql
        ↪  5432:5432 &
    PGPASSWORD="$POSTGRES_PASSWORD" psql --host 127.0.0.1 -U dbusername
        ↪ -d keycloak -p 5432

WARNING: The configured password will be ignored on new installation in
    ↪ case when previous PostgreSQL release was deleted through the helm
    ↪  command. In that case, old PVC will have an old password, and
    ↪ setting it through helm will not take effect. Deleting persistent
    ↪ volumes (PVs) will solve the issue.

WARNING: There are "resources" sections in the chart not set. Using "
    ↪ resourcesPreset" is not recommended for production. For production
    ↪  installations, please set the following values according to your
    ↪ workload needs:
    - primary.resources
    - readReplicas.resources
+info https://kubernetes.io/docs/concepts/configuration/manage-resources
    ↪ -containers/
```

Cuadro 85: Output al momento de instalar la base de datos de *Keycloak*

- Configurar *Keycloak*

```
sudo nano keycloak-server-values.yaml
```

Cuadro 86: Configurar *Keycloak* a través de un archivo .yaml

- Dentro de este archivo colocar lo siguiente

```
---
command:
    - "/opt/keycloak/bin/kc.sh"
    - "--verbose"
    - "start"
    - "--http-enabled=true"
    - "--http-port=8080"
    - "--hostname-strict=false"
    - "--spi-events-listener-jboss-logging-success-level=info"
    - "--spi-events-listener-jboss-logging-error-level=warn"

extraEnv: |
    - name: KEYCLOAK_ADMIN
    valueFrom:
        secretKeyRef:
        name: {{ include "keycloak.fullname" . }}-admin-creds
        key: user
    - name: KEYCLOAK_ADMIN_PASSWORD
    valueFrom:
        secretKeyRef:
        name: {{ include "keycloak.fullname" . }}-admin-creds
        key: password
    - name: JAVA_OPTS_APPEND
    value: >-
```

```
        -XX:+UseContainerSupport
        -XX:MaxRAMPercentage=50.0
        -Djava.awt.headless=true
        -Djgroups.dns.query={{ include "keycloak.fullname" . }}-headless
    - name: PROXY_ADDRESS_FORWARDING
    value: "true"

dbchecker:
    enabled: true

database:
    vendor: postgres
    hostname: keycloak-db-postgresql
    port: 5432
    username: dbusername
    password: dbpassword
    database: keycloak

secrets:
    admin-creds:
    stringData:
        user: admin
        password: secret

http:
    # For backwards compatibility reasons we set this to the value used
      ↪ by previous Keycloak versions.
    relativePath: "/" # "/auth"
```

Cuadro 87: Datos del archivo *"keycloak-server-values.yaml"*

- Instalar *keycloak*

```
helm -n keycloak upgrade -i keycloak codecentric/keycloakx --values ./
  ↪ keycloak-server-values.yaml
```

Cuadro 88: Instalación de *keycloak* a través de *Helm*

```
onap@onapnodo1:~$ helm -n keycloak upgrade -i keycloak codecentric/keycloakx --values ./keycloak-server-values.yaml
Release "keycloak" does not exist. Installing it now.
NAME: keycloak
LAST DEPLOYED: Tue Aug 20 22:27:36 2024
NAMESPACE: keycloak
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
**************************************************************************
*                                                                        *
*              Keycloak.X Helm Chart by codecentric AG                   *
*                                                                        *
**************************************************************************

Keycloak was installed with a Service of type ClusterIP

Create a port-forwarding with the following commands:

export POD_NAME=$(kubectl get pods --namespace keycloak -l "app.kubernetes.io/name=keycloakx,app.kubernetes.io/instance=
keycloak" -o name)
echo "Visit http://127.0.0.1:8080 to use your application"
kubectl --namespace keycloak port-forward "$POD_NAME" 8080
```

Figura 120: Instalación de *Keycloak*

- Crear un *Ingress gateway* entry para el *keycloak web interface*

```
sudo nano keycloak-ingress.yaml
```

Cuadro 89: Crear archivo *"keycloak-ingress.yaml"*

- En este archivo colocar la siguiente información:

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
    labels:
    app.kubernetes.io/managed-by: Helm
    name: keycloak-ui-http-route
    namespace: keycloak
spec:
    hostnames:
    - keycloak-ui.simpledemo.onap.org
    parentRefs:
    - group: gateway.networking.k8s.io
    kind: Gateway
    name: common-gateway
    namespace: istio-ingress
    sectionName: https-80
    rules:
    - filters:
    - type: RequestRedirect
        requestRedirect:
        scheme: https
        port: 443
        statusCode: 301
    matches:
    - path:
        type: PathPrefix
        value: /auth

---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
    labels:
    app.kubernetes.io/managed-by: Helm
    name: keycloak-ui-http-route
    namespace: keycloak
spec:
    hostnames:
    - keycloak-ui.simpledemo.onap.org
    parentRefs:
    - group: gateway.networking.k8s.io
    kind: Gateway
    name: common-gateway
    namespace: istio-ingress
    sectionName: https-443
    rules:
    - backendRefs:
    - group: ""
```

```
        kind: Service
        name: keycloak-keycloakx-http
        port: 80
        weight: 1
    matches:
    - path:
        type: PathPrefix
        value: /auth
```

Cuadro 90: Datos del archivo *"keycloak-ingress.yaml"*

- Luego de esto, aplicar dicho archivo

```
kubectl -n keycloak apply -f keycloak-ingress.yaml
```

Cuadro 91: Aplicación de archivo *"keycloak-ingress.yaml"*

```
onap@onapnodo1:~$ kubectl -n keycloak apply -f keycloak-ingress.yaml
httproute.gateway.networking.k8s.io/keycloak-ui-http-route configured
httproute.gateway.networking.k8s.io/keycloak-ui-http-route configured
```

Figura 121: Aplicación de archivo *keycloak-ingress.yaml*

- Con ello, podemos comprobar que nuestros *pods* de *keycloak* están corriendo perfectamente:

```
kubectl get pods -n keycloak
```

Cuadro 92: Obtener *pods* de *keycloak* corriendo en nuestro clúster

```
onap@onapnodo1:~$ kubectl get pods -n keycloak
NAME                        READY   STATUS    RESTARTS       AGE
keycloak-db-postgresql-0    1/1     Running   78 (15h ago)   4d20h
keycloak-keycloakx-0        1/1     Running   0              109s
```

Figura 122: *Pods* de *Keycloak* corriendo en el clúster

7.4. Instalación de ONAP

Anteriormente, al momento de instalar *Helm*, tuvimos que instalar un *plugin* del mismo, llamado *"deploy"*. ONAP utiliza este *plugin* para instalar una instancia del mismo. Previo a esto, es necesario que a través de el archivo *"values.yaml"* en la ruta *oom/kubernetes/onap*, se modifiquen valores para indicar cuales son los submódulos de ONAP que se van a instalar al momento de realizar el despliegue de ONAP. Para ello realizaremos los siguientes pasos:

- Debemos movernos al directorio *oom/kubernetes/onap* y abrir el archivo *"values.yaml"*

```
cd oom/kubernetes/onap
sudo nano values.yaml
```

Cuadro 93: Movernos a directorio *oom/kubernetes/onap* para trabajar en archivo *"values.yaml"*

- Dentro de este archivo buscaremos la sección que diga "*Enable/disable and configure helm charts (ie. applications) to customize the ONAP deployment*". En esta misma, buscaremos colocar *"true"* en los *Helm charts* o submódulos que deseamos instalar al levantar nuestra instancia de ONAP, tal como se puede observar en la línea siguiente:

```
aai:
    enabled: true
```

Cuadro 94: Ejemplo de como modificar valores en archivo *"values.yaml"*

Esto lo tenemos que hacer también en el archivo *"onap-all.yaml"* dentro del directorio *oom/kubernetes/onap/resources/overrides*

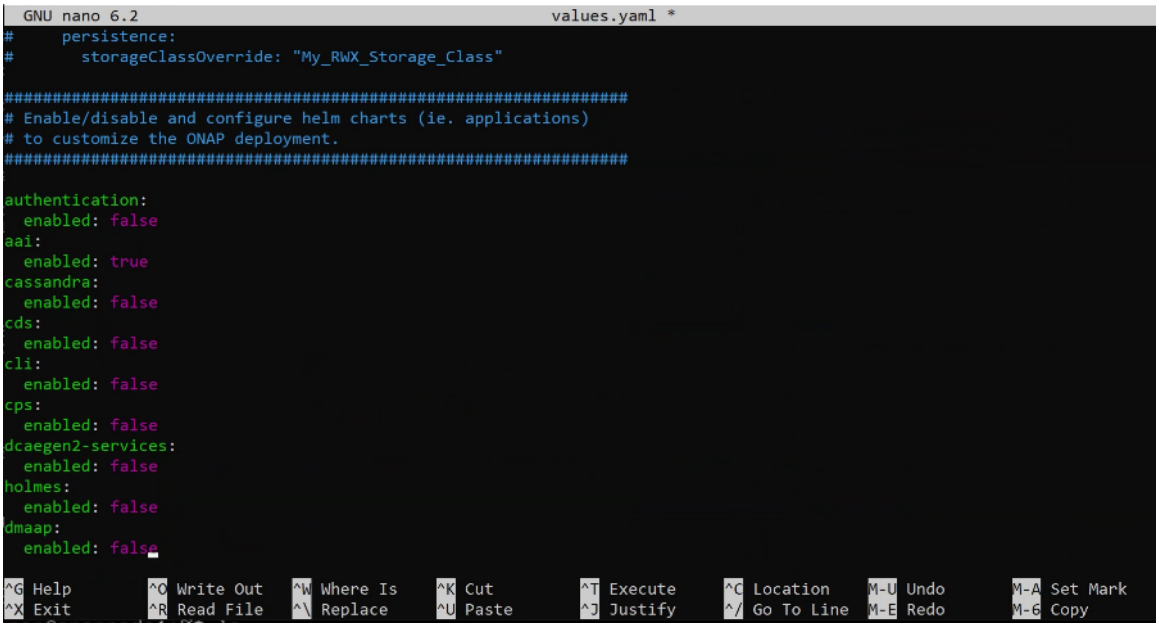


Figura 123: Archivo *"values.yaml"*

En nuestro caso lanzaremos 4 componentes:

1. *AAF*
2. *Cassandra*

3. *Portal-NG*
4. *SDC*

Con estos 4 componentes de ONAP podremos generar pruebas sencillas para demostrar la utilidad y eficiencia de la plataforma.

- Añadir el repo de *Helm* de *Release & Deploy* del OOM

```
helm repo add onap-release https://nexus3.onap.org/repository/onap-helm-release/
helm repo update
```

Cuadro 95: Añadir repositorio *onap release* para la instalación de ONAP

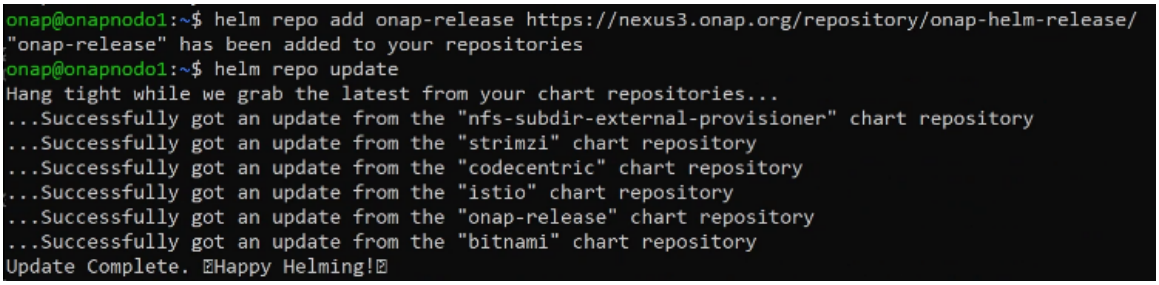


Figura 124: Añadir repositorio de ONAP

- Luego, verificaremos la versión que queremos desplegar dentro de los diferentes releases incluidos en el repo anteriormente agregado. Para nuestro caso queremos lanzar una instancia de ONAP *Montreal*. Para ello realizaremos los siguientes comandos para verificar la versión del release que queremos:

```
helm search repo onap --versions
```

Cuadro 96: Buscar versiones disponibles a desplegar de ONAP

```
onap@onapnodo1:~$ helm search repo onap --versions
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION

onap-release/onap                       13.0.0          Montreal        Open Network Automation Platform (ONAP)

onap-release/onap                       12.0.0          London          Open Network Automation Platform (ONAP)
(
onap-release/onap                       11.0.0          Kohn            Open Network Automation Platform (ONAP)
(
(onap-release/onap                      10.0.0          Jakarta         Open Network Automation Platform (ONAP)
(
(
onap-release/onap                       9.0.0           Istanbul        Open Network Automation Platform (ONAP)

onap-release/a1policymanagement         13.0.0          1.0.0           A Helm chart for A1 Policy Management Se
rvice
onap-release/a1policymanagement         12.0.0          1.0.0           A Helm chart for A1 Policy Management Se
(rvice
onap-release/a1policymanagement         11.0.0          1.0.0           A Helm chart for A1 Policy Management Se
rvice
onap-release/a1policymanagement         10.0.0          1.0.0           A Helm chart for A1 Policy Management Se
rvice
onap-release/a1policymanagement         9.0.0           1.0.0           A Helm chart for A1 Policy Management Se
rvice
onap-release/aaf                        11.0.0                          ONAP Application Authorization Framework

(onap-release/aaf                       10.0.0                          ONAP Application Authorization Framework
```

Figura 125: Versiones disponibles de ONAP

- Una vez tenemos la versión que queremos lanzar, procederemos a instalar ONAP reemplazando el parámetro *"version"* por la versión que queremos instalar. Como estamos intentando lanzar una instancia de ONAP *Montreal*, seleccionaremos la versión 13.0.0:

```
helm deploy dev onap-release/onap --namespace onap --create-namespace --
    ↪ set global.masterPassword=onap --version 13.0.0 -f oom/kubernetes/
    ↪ onap/resources/overrides/onap-all.yaml
```

Cuadro 97: Desplegar instancia de ONAP en el clúster de *Kubernetes*

```
(onap@onapnodo1:~$ helm deploy dev onap-release/onap --namespace onap --create-namespace --set global.masterPassword=onap
 --version 13.0.0 -f oom/kubernetes/onap/resources/overrides/onap-all.yaml
(v3.12.3
(Use cache dir: /home/onap/.local/share/helm/plugins/deploy/cache
(0
0
0
0
fetching onap-release/onap
(history.go:56: [debug] getting history for release dev
(install.go:200: [debug] Original chart version: "13.0.0"
install.go:217: [debug] CHART PATH: /home/onap/.local/share/helm/plugins/deploy/cache/onap

release "dev" deployed
release "dev-roles-wrapper" deployed
(release "dev-repository-wrapper" deployed
(release "dev-strimzi" deployed
waiting for dev-strimzi-entity-operator to be deployed
```

Figura 126: Despliegue de ONAP

Una vez la terminal nos indique que todos los módulos fueron instalados de manera exitosa, quiere decir que nuestra instancia de ONAP fue levantada de manera apropiada. A pesar de observar que todos los módulos están en estado *"deployed"*, debido a recursos y problemas en el lanzamiento de los mismos, existían errores de autentificación y manejo de recursos, por lo que fue imposible probar y utilizar ONAP.

Debido a esto se buscó desarrollar nuestra propia aplicación para automatizar redes. De esta forma, se demostraría lo fácil que es desplegar aplicaciones dentro de nuestro cluster de *kubernetes*.

7.5. Prueba de concepto 2: Despliegue de aplicaciones para automatización de redes en un clúster de *Kubernetes*

Al momento de desarrollar o instalar una API o aplicación que permita la automatización de redes, se busca implementar la misma dentro del clúster de *kubernetes* para que se pueda comprobar el funcionamiento de nuestro clúster. Esto es bastante sencillo al realizarlo a través de *Helm*, ya que *Helm* nos permite administrar los paquetes dentro de *kubernetes*.

7.5.1. Creación de un *chart* de *Helm*

Primero debemos crear el *chart* de *Helm*:

```
1 helm create network-automation-app
```

Cuadro 98: Crear *Helm Chart* para la aplicación desarrollada

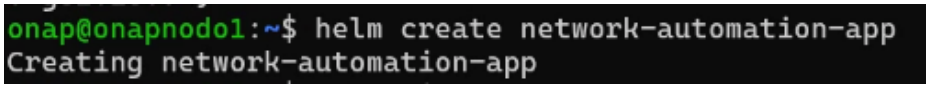


Figura 127: Creación de *Helm Chart*

Luego debemos de navegar a ese mismo directorio y modificar los archivos de *"Chart.yaml"* y *"values.yaml"*. *"Chart.yaml"* nos permitirá ajustar valores como el nombre de la aplicación, la versión de la aplicación, la versión del *chart*, entre otros. Mientras qué, *"values.yaml"* nos permitirá ajustar valores como el número de replicas, el puerto de la aplicación, el tipo de servicio, repositorio del cual extraeremos la aplicación, entre otros.

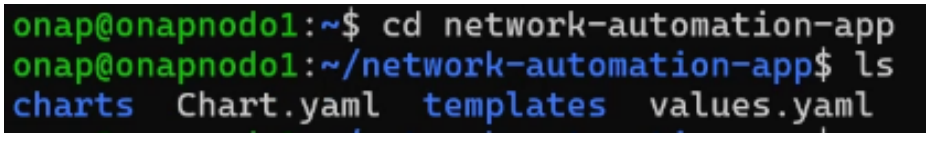


Figura 128: Archivos de *"Chart.yaml"* y *"values.yaml"* dentro del directorio de nuestro *chart* de *Helm* creado

Estos archivos los modificaremos de la siguiente forma:

- *Chart.yaml*:

```yaml
apiVersion: v2
name: network-automation-app
description: Helm chart to deploy a network automation app via a Kubernetes cluster

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into versioned archives
# to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer. They are included as
# a dependency of application charts to inject those utilities and functions into the rendering
# pipeline. Library charts do not define any templates and therefore cannot be deployed.
type: application

# This is the chart version. This version number should be incremented each time you make changes
# to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.1.0

# This is the version number of the application being deployed. This version number should be
# incremented each time you make changes to the application. Versions are not expected to
# follow Semantic Versioning. They should reflect the version the application is using.
# It is recommended to use it with quotes.
appVersion: "1.0.0"
```

Cuadro 99: Datos del archivo *"Chart.yaml"*

- *values.yaml*:

```yaml
# Default values for network-automation-app.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

replicaCount: 1

image:
    ## Use the name of the Docker Hub repository where we uploaded our image containing our application
    repository: jocarsoli2001/pruebadeconcepto-app
    pullPolicy: IfNotPresent
    # Overrides the image tag whose default is the chart appVersion.
    tag: "v1"

imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""
```

```yaml

serviceAccount:
    # Specifies whether a service account should be created
    create: false
    # Annotations to add to the service account
    annotations: {}
    # The name of the service account to use.
    # If not set and create is true, a name is generated using the fullname
        ↪ template
    name: ""

podAnnotations: {}

podSecurityContext: {}
    # fsGroup: 2000

securityContext: {}
    # capabilities:
    #   drop:
    #   - ALL
    # readOnlyRootFilesystem: true
    # runAsNonRoot: true
    # runAsUser: 1000

service:
    type: NodePort
    port: 8000
    nodePort: 30080

ingress:
    enabled: false
    className: ""
    annotations: {}
    # kubernetes.io/ingress.class: nginx
    # kubernetes.io/tls-acme: "true"
    hosts:
    - host: chart-example.local
        paths:
        - path: /
            pathType: ImplementationSpecific
    tls: []
    #  - secretName: chart-example-tls
    #    hosts:
    #      - chart-example.local

resources: {}
    # We usually recommend not to specify default resources and to leave this
        ↪  as a conscious
    # choice for the user. This also increases chances charts run on
        ↪ environments with little
    # resources, such as Minikube. If you do want to specify resources,
        ↪ uncomment the following
    # lines, adjust them as necessary, and remove the curly braces after '
        ↪ resources:'.
    # limits:
    #   cpu: 100m
    #   memory: 128Mi
    # requests:
    #   cpu: 100m
```

```
    #   memory: 128Mi

autoscaling:
    enabled: false
    minReplicas: 1
    maxReplicas: 100
    targetCPUUtilizationPercentage: 80
    # targetMemoryUtilizationPercentage: 80

nodeSelector: {}

tolerations: []

affinity: {}
```

Cuadro 100: Datos para el archivo *"values.yaml"*

Una vez con esto modificado, ya podemos implementar nuestra aplicación dentro de nuestro clúster.

```
# Desde el directorio de nuestro chart de helm
helm install network-app ./
```

Cuadro 101: Desplegar la aplicación en el clúster de *Kubernetes* mediante *Helm*

```
onap@onapnodo1:~/network-automation-app$ helm install network-app ./
NAME: network-app
LAST DEPLOYED: Thu Nov  7 02:48:18 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services network-app-network-
automation-app)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
```

Figura 129: Despliegue de la aplicación en el clúster de *Kubernetes*

Ahora podemos comprobar si nuestro servicio está corriendo:

```
kubectl get svc
kubectl get pods
```

Cuadro 102: Obtener servicios y *pods* de la aplicación desplegada

```
onap@onapnodo1:~$ kubectl get svc
NAME                                TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes                          ClusterIP   10.96.0.1      <none>        443/TCP          5h24m
network-app-network-automation-app  NodePort    10.98.180.10   <none>        8000:31736/TCP   5m3s
onap@onapnodo1:~$ kubectl get pods
NAME                                                READY   STATUS    RESTARTS   AGE
network-app-network-automation-app-564fdb657c-sv9xk   1/1     Running   0          5m3s
```

Figura 130: Obtención de servicios y *pods* de la aplicación desplegada

Con esto, ya tenemos nuestra aplicación corriendo dentro de nuestro clúster de *kubernetes*.

7.5.2. Visualización de nuestro servicio desplegado

Para poder visualizar nuestra aplicación desde un navegador, necesitamos obtener la URL de nuestra aplicación. Esto se hace a través de los siguientes comandos:

```
export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports
    ↪ [0].nodePort}" services network-app-network-automation-app)

export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items
    ↪ [0].status.addresses[0].address}")

echo http://$NODE_IP:$NODE_PORT
```

Cuadro 103: Exportar credenciales de acceso para visualizar, en la web, la interfaz gráfica de nuestra *API*

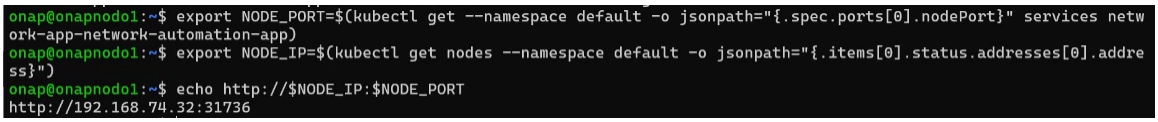


Figura 131: Exportar credenciales de acceso para visualizar la interfaz gráfica de nuestra *API*

Ahora, como podemos observar, ya tenemos una URL a la que podemos accesar y así, visualizar nuestra aplicación sin problema alguno:

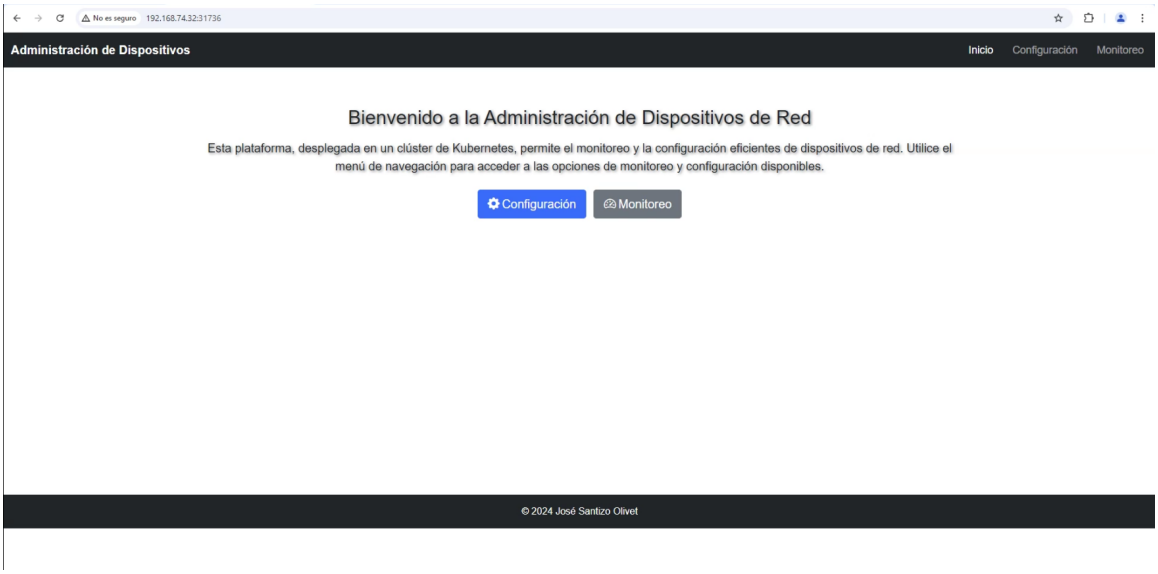


Figura 132: *API*, desplegada en el clúster de *Kubernetes*, vista desde un navegador

Con esto, podemos observar lo sencillo que es desplegar aplicaciones dentro del clúster de *Kubernetes*. Al momento de hacer esto, garantizamos que el clúster funcione apropiadamente y que las aplicaciones desplegadas dentro del mismo también lo hagan.

CAPÍTULO 8

Resultados

8.1. Clúster de *Kubernetes* de alta disponibilidad

Para implementar un clúster de *Kubernetes* de alta disponibilidad, se logró configurar y desplegar un clúster que incorporaba un servidor *NFS* para proporcionar almacenamiento compartido entre los nodos, así como dos balanceadores de carga, lo que facilitó la gestión de datos y la persistencia de información.

8.1.1. Balanceadores de carga utilizando *HAProxy*

Se implementaron dos balanceadores de carga utilizando *HAProxy* para distribuir el tráfico de red entre los nodos de control y los nodos de trabajo del clúster. Estos balanceadores permitieron una distribución equitativa de las solicitudes de red y garantizaron la alta disponibilidad de las aplicaciones desplegadas en el clúster.

Los dos balanceadores de carga se configuraron junto a *KeepAlived* para contar con una dirección IP virtual (vIP) mediante el protocolo VRRP, proporcionando un punto de acceso único al clúster, lo que permitió una comunicación eficiente y segura con las aplicaciones desplegadas. Además, se establecieron reglas de balanceo de carga para optimizar el rendimiento y la disponibilidad de los servicios en el clúster. La configuración de *HAProxy* junto con *KeepAlived* se muestra en el Código 104 y 105.

```
1 global
2         log /dev/log    local0
3         log /dev/log    local1 notice
4         chroot /var/lib/haproxy
5         stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd
              ↪ listeners
```

```
        stats timeout 30s
        maxconn 4000
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # See: https://ssl-config.mozilla.org/#server=haproxy&server-version
            ↪ =2.0.3&config=intermediate
        ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
            ↪ AES128-GCM-SHA256:ECDHE-ECDSA-AES256->        ssl-default-bind
            ↪ -ciphersuites TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
            ↪ TLS_CHACHA20_POLY1305_SHA2>        ssl-default-bind-options
            ↪ ssl-min-ver TLSv1.2 no-tls-tickets

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

# ---------------------------------------------------------------
# KUBERNETES CONFIGURATION
# ---------------------------------------------------------------
listen stats
        bind *:9000
        mode http
        stats enable
        stats hide-version
        stats uri /stats
        stats refresh 30s
        stats realm Haproxy\ Statistics
        stats auth onap:onap

frontend k8s-api-https-proxy
        bind *:6443
        mode tcp
        option tcplog
        default_backend k8s-api-https

backend k8s-api-https
        balance roundrobin
        mode tcp
        option tcplog
        option tcp-check
        default-server inter 100s downinter 5s rise 2 fall 2 slowstart 60s
```

```
            ↪ maxconn 250 maxqueue 256 weight 100
        server k8s-api-1 192.168.6.137:6443 check
        server k8s-api-2 192.168.6.138:6443 check
        server k8s-api-3 192.168.6.139:6443 check
```

Cuadro 104: Configuración de *HAProxy* para balanceo de carga

```
global_defs {
  notification_email {
  }
  router_id onapLB1                       # Hostname de la máquina actual
  vrrp_skip_check_adv_addr
  vrrp_garp_interval 0
  vrrp_gna_interval 0
}

vrrp_script chk_haproxy {
  script "killall -0 haproxy"
  interval 2
  weight 2
}

vrrp_instance 50 {
  state MASTER                            # En VM master, state = MASTER
  priority 50                             # En VM backup, state = BACKUP
  interface enp0s3                        # Network card de nuestra VM
  virtual_router_id 60
  advert_int 1
  unicast_src_ip 192.168.6.143            # Dirección IP de máquina host
  unicast_peer {
    192.168.6.144                         # Dirección IP de máquina peer
  }

  virtual_ipaddress {
    192.168.6.145/25                      # Dirección IP virtual
  }

  track_script {
    chk_haproxy
  }
}
```

Cuadro 105: Configuración de *KeepAlived* para alta disponibilidad

Con ambos componentes configurados, se garantizó la alta disponibilidad y la tolerancia a fallos en el clúster. El correcto funcionamiento de los balanceadores de carga se puede visualizar en la Figura 133, mientras que la IP virtual asignada a los balanceadores de carga (mediante el buen funcionamiento de *KeepAlived*, tal como se muestra en la Figura 135) se muestra en la Figura 134.

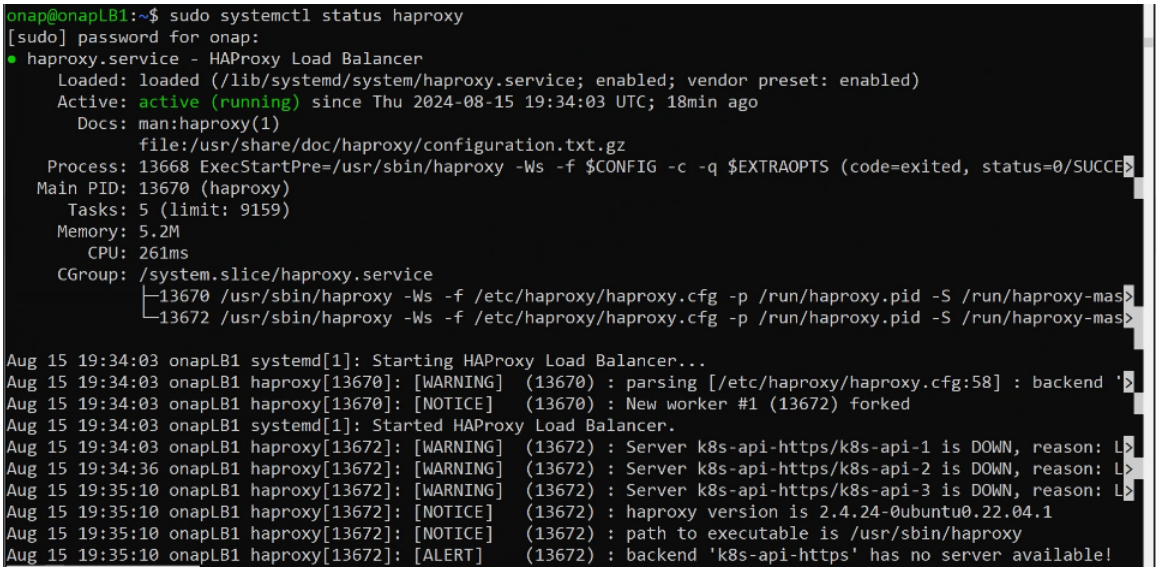


Figura 133: Balanceadores de carga funcionales para el clúster de *Kubernetes*

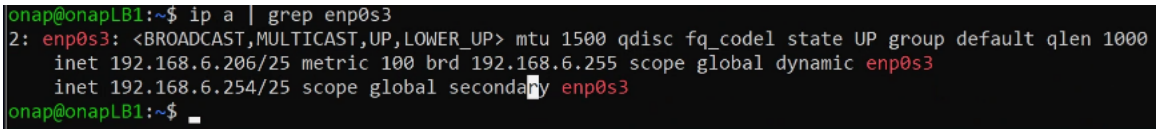


Figura 134: Dirección IP virtual funcional, asignada a router *Master* mediante *KeepAlived*

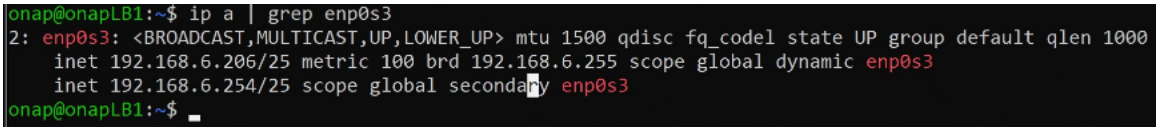


Figura 135: *KeepAlived* configurado y funcionando para alta disponibilidad en el clúster de *Kubernetes*

8.1.2. Servidor de almacenamiento *NFS*

Se implementó un servidor de almacenamiento *NFS* para proporcionar almacenamiento compartido entre los nodos del clúster. El servidor *NFS* permitió a los nodos acceder y compartir archivos de manera eficiente, facilitando la persistencia de datos y la comunicación entre los nodos del clúster.

Se configuró la máquina que serviría como servidor *NFS* para exportar un directorio compartido a los nodos del clúster a través de una red. La configuración del servidor *NFS* se muestra en el Código 106.

```
# -- Creación de directorio a exportar mediante una red --
sudo mkdir -p /data/nfs
sudo chown nobody:nogroup /data/nfs
sudo chmod 2770 /data/nfs

# -- Exportación de directorio a través de la red 192.168.72.0/22 --
echo -e "/data/nfs\t192.168.72.0/22(rw,sync,no_subtree_check,no_root_squash)"
    ↪  | sudo tee -a /etc/exports

sudo exportfs -av
```

Cuadro 106: Configuración del servidor *NFS* para almacenamiento compartido

Con esto configurado, se pudo comprobar que el servidor *NFS* estaba funcionando correctamente y que los nodos del clúster podían acceder y compartir archivos a través de la red. Las figuras 136 y 137 muestran el correcto funcionamiento del servidor *NFS* y la exportación de un directorio compartido a los nodos del clúster. Además, se puede observar que ya existe un cliente *NFS* corriendo en nuestro clúster en la Figura 138.

```
onap@onapNFS:~$ sudo systemctl restart nfs-kernel-server
onap@onapNFS:~$ sudo systemctl status nfs-kernel-server
● nfs-server.service - NFS server and services
     Loaded: loaded (/lib/systemd/system/nfs-server.service; enabled; vendor preset: enabled)
     Active: active (exited) since Sun 2024-08-18 20:24:57 UTC; 7s ago
    Process: 37854 ExecStartPre=/usr/sbin/exportfs -r (code=exited, status=0/SUCCESS)
    Process: 37855 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
   Main PID: 37855 (code=exited, status=0/SUCCESS)
        CPU: 3ms

Aug 18 20:24:57 onapNFS systemd[1]: Starting NFS server and services...
Aug 18 20:24:57 onapNFS systemd[1]: Finished NFS server and services.
```

Figura 136: Servidor *NFS* funcionando correctamente para el clúster de *Kubernetes*

```
onap@onapNFS:~$ echo -e "/data/nfs\t192.168.6.128/25(rw,sync,no_subtree_check,no_root_squash)" | sudo tee -a /etc/export
s
/data/nfs       192.168.6.128/25(rw,sync,no_subtree_check,no_root_squash)
onap@onapNFS:~$ sudo exportfs -av
exporting 192.168.6.128/25:/data/nfs
```

Figura 137: Exportación de directorio compartido a los nodos del clúster de *Kubernetes*

```
onap@onapnodo1:~$ kubectl get pods --all-namespaces
NAMESPACE         NAME                                          READY   STATUS    RESTARTS        AGE
kube-system       calico-kube-controllers-8498bff86b-kkkk8      1/1     Running   0               5m29s
kube-system       calico-node-8d8zs                             1/1     Running   0               5m29s
kube-system       calico-node-d2pxv                             1/1     Running   0               5m29s
kube-system       calico-node-m482s                             1/1     Running   0               5m29s
kube-system       calico-node-nlhzg                             1/1     Running   0               5m29s
kube-system       calico-node-nmzgj                             1/1     Running   0               5m29s
kube-system       calico-node-vlgsg                             1/1     Running   0               5m29s
kube-system       coredns-5d78c9869d-42v2k                      1/1     Running   0               11m
kube-system       coredns-5d78c9869d-sfw9j                      1/1     Running   0               11m
kube-system       etcd-onapnodo1                                1/1     Running   0               11m
kube-system       etcd-onapnodo2                                1/1     Running   0               9m59s
kube-system       etcd-onapnodo3                                1/1     Running   0               8m59s
kube-system       kube-apiserver-onapnodo1                      1/1     Running   0               11m
kube-system       kube-apiserver-onapnodo2                      1/1     Running   0               9m59s
kube-system       kube-apiserver-onapnodo3                      1/1     Running   1 (9m14s ago)   8m58s
kube-system       kube-controller-manager-onapnodo1             1/1     Running   1 (9m49s ago)   11m
kube-system       kube-controller-manager-onapnodo2             1/1     Running   0               9m43s
kube-system       kube-controller-manager-onapnodo3             1/1     Running   0               8m9s
kube-system       kube-proxy-47kr7                              1/1     Running   0               8m43s
kube-system       kube-proxy-5cjgf                              1/1     Running   0               9m59s
kube-system       kube-proxy-75j5l                              1/1     Running   0               9m9s
kube-system       kube-proxy-8lmvk                              1/1     Running   0               11m
kube-system       kube-proxy-bnstt                              1/1     Running   0               8m54s
kube-system       kube-proxy-fzkkd                              1/1     Running   0               8m49s
kube-system       kube-scheduler-onapnodo1                      1/1     Running   1 (9m45s ago)   11m
kube-system       kube-scheduler-onapnodo2                      1/1     Running   0               9m43s
kube-system       kube-scheduler-onapnodo3                      1/1     Running   0               9m5s
nfs-provisioner   nfs-subdir-external-provisioner-6d97f6d7f-kd959   1/1     Running   0               74s
```

Figura 138: Pod corriendo cliente *NFS* en el clúster de *Kubernetes*

8.1.3. Funcionalidad del clúster de *Kubernetes*

El clúster de *Kubernetes* fue implementado con éxito y garantizaba estabilidad en todos los pods que se ejecutaban en el mismo. Como se puede visualizar en la Figura 139, el clúster de *Kubernetes* contaba con múltiples nodos que corrían servicios como *Calico* para la red de comunicación de todos los pods; *APIserver*, *Controller Manager* y *Scheduler* para la administración de los nodos; *CoreDNS* para la resolución de nombres de dominio; *Etcd*, *Kube-proxy*, servidor *NFS* y la aplicación para la automatización de redes que se desarrolló como prueba para el clúster de *Kubernetes*. Todos estos servicios se encontraban funcionando correctamente y probaban la alta disponibilidad y tolerancia a fallos del clúster.

```
onap@onapnodo1:~$ kubectl get pods --all-namespaces
NAMESPACE          NAME                                                READY   STATUS    RESTARTS       AGE
default            network-app-network-automation-app-7d4566b579-bzsr4   1/1     Running   0              108m
kube-system        calico-kube-controllers-8498bff86b-kkkk8            1/1     Running   0              115m
kube-system        calico-node-8d8zs                                   1/1     Running   0              115m
kube-system        calico-node-d2pxv                                   1/1     Running   0              115m
kube-system        calico-node-m482s                                   1/1     Running   0              115m
kube-system        calico-node-nlhzg                                   1/1     Running   0              115m
kube-system        calico-node-nmzgj                                   1/1     Running   0              115m
kube-system        calico-node-vlgsg                                   1/1     Running   0              115m
kube-system        coredns-5d78c9869d-42v2k                            1/1     Running   0              121m
kube-system        coredns-5d78c9869d-sfw9j                            1/1     Running   0              121m
kube-system        etcd-onapnodo1                                      1/1     Running   0              121m
kube-system        etcd-onapnodo2                                      1/1     Running   0              119m
kube-system        etcd-onapnodo3                                      1/1     Running   0              118m
kube-system        kube-apiserver-onapnodo1                            1/1     Running   0              121m
kube-system        kube-apiserver-onapnodo2                            1/1     Running   0              119m
kube-system        kube-apiserver-onapnodo3                            1/1     Running   1 (119m ago)   118m
kube-system        kube-controller-manager-onapnodo1                   1/1     Running   1 (119m ago)   121m
kube-system        kube-controller-manager-onapnodo2                   1/1     Running   0              119m
kube-system        kube-controller-manager-onapnodo3                   1/1     Running   0              118m
kube-system        kube-proxy-47kr7                                    1/1     Running   0              118m
kube-system        kube-proxy-5cjgf                                    1/1     Running   0              119m
kube-system        kube-proxy-75j5l                                    1/1     Running   0              119m
kube-system        kube-proxy-8lmvk                                    1/1     Running   0              121m
kube-system        kube-proxy-bnstt                                    1/1     Running   0              118m
kube-system        kube-proxy-fzkkd                                    1/1     Running   0              118m
kube-system        kube-scheduler-onapnodo1                            1/1     Running   1 (119m ago)   121m
kube-system        kube-scheduler-onapnodo2                            1/1     Running   0              119m
kube-system        kube-scheduler-onapnodo3                            1/1     Running   0              118m
kube-system        metrics-server-664c989b66-znb6t                     1/1     Running   0              81m
nfs-provisioner    nfs-subdir-external-provisioner-6d97f6d7f-kd959     1/1     Running   0              111m
```

Figura 139: Pods dentro del clúster de *Kubernetes* funcionando correctamente, probando la funcionalidad del mismo

Por otro lado, podemos visualizar el correcto funcionamiento de los nodos del clúster en la Figura 140, donde se muestra la cantidad de nodos que se encuentran en el clúster, así como su estado. Cada uno de estos nodos implica una máquina virtual corriendo dentro de las diferentes computadoras físicas que se utilizaron para el proyecto.

Asimismo, cada uno de estos nodos ejecuta una cantidad de *pods* específica que consume recursos de los mismos. Tal como se observa en la Figura 141, se pueden ver todos los recursos que los diferentes nodos ocupan.

```
onap@onapnodo1:~$ kubectl get nodes
NAME        STATUS   ROLES           AGE   VERSION
onapnodo1   Ready    control-plane   8h    v1.27.5
onapnodo2   Ready    control-plane   8h    v1.27.5
onapnodo3   Ready    control-plane   8h    v1.27.5
onapnodo4   Ready    <none>          8h    v1.27.5
onapnodo5   Ready    <none>          8h    v1.27.5
onapnodo6   Ready    <none>          8h    v1.27.5
```

Figura 140: Nodos del clúster de *Kubernetes* funcionando correctamente

```
onap@onapnodo1:~$ kubectl top nodes
NAME        CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
onapnodo1   85m          0%     2237Mi          9%
onapnodo2   82m          2%     1732Mi          22%
onapnodo3   69m          0%     2992Mi          12%
onapnodo4   32m          0%     2603Mi          12%
onapnodo5   30m          0%     1258Mi          8%
onapnodo6   41m          0%     1275Mi          8%
```

Figura 141: Recursos empleados por los nodos del clúster de *Kubernetes*

8.2. Limitaciones y aprendizajes en el despliegue de ONAP

A pesar de los esfuerzos realizados para desplegar ONAP (*Open Network Automation Platform*) en el clúster de *Kubernetes* de alta disponibilidad, no fue posible completar el despliegue de manera exitosa. Sin embargo, el proceso proporcionó valiosos aprendizajes y permitió identificar una serie de limitaciones que influyeron en el resultado final.

8.2.1. Plataforma base de ONAP con OOM

Se intentó utilizar el *ONAP Operations Manager* (OOM) para el despliegue de ONAP en el clúster. Para ello, fue necesario preparar y configurar una serie de componentes críticos en el clúster de *Kubernetes*, los cuales se detallan en secciones próximas.

Previo a realizar la instalación de los componentes críticos, se clonó el repositorio oficial de OOM y se prepararon los archivos de configuración necesarios para el despliegue de ONAP. Esto se puede visualizar en la Figura 142.

```
onap@onapnodo1:~$ git clone https://gerrit.onap.org/r/oom
Cloning into 'oom'...
remote: Counting objects: 16, done
remote: Total 94985 (delta 0), reused 94985 (delta 0)
Receiving objects: 100% (94985/94985), 40.07 MiB | 2.95 MiB/s, done.
Resolving deltas: 100% (72016/72016), done.
```

Figura 142: Repositorio de OOM clonado para el despliegue de ONAP

8.2.1.1. Helm

Como parte fundamental de la instalación de ONAP, se utilizó *Helm* como gestor de paquetes para *Kubernetes*. *Helm* facilitó la automatización del despliegue y la configuración de los diferentes servicios que conforman la arquitectura de ONAP. Se logró instalar la versión 3 de *Helm* en el clúster, lo cual permitió gestionar y orquestar el despliegue de todos los *charts* necesarios, incluyendo el propio *chart* de ONAP, OOM. La instalación de *Helm* se puede visualizar en la Figura 143

```
onap@onapnodo1:~$ tar -zxvf helm-v3.12.3-linux-amd64.tar.gz
linux-amd64/
linux-amd64/LICENSE
linux-amd64/README.md
linux-amd64/helm
onap@onapnodo1:~$ sudo mv linux-amd64/helm /usr/local/bin/helm
[sudo] password for onap:
onap@onapnodo1:~$
```

Figura 143: Instalación de *Helm* en el clúster de *Kubernetes*

Figura 144: Chequeo de versión instalada de *Helm*

El uso de *Helm* permitió simplificar el manejo de las dependencias y parametrizar los archivos de configuración, lo cual fue esencial para tratar de llevar a cabo el despliegue de una plataforma tan compleja como ONAP.

8.2.1.2. Operador Strimzi Kafka

Otro componente importante que se instaló en el clúster fue el operador *Strimzi Kafka*. Este operador se utilizó para desplegar y gestionar *Kafka*, una tecnología clave para la recolección y distribución de eventos dentro de ONAP. El despliegue de *Strimzi* se realizó con éxito y se logró instalar un clúster de *Kafka* funcional en el clúster de *Kubernetes*, como se muestra en las figuras 145 y 146.

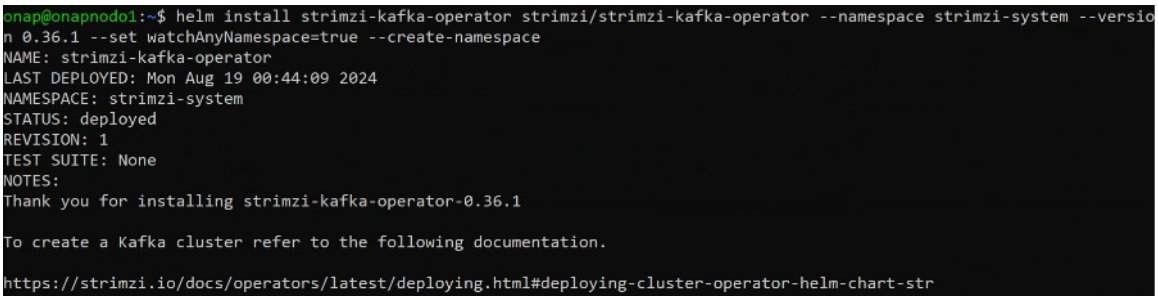


Figura 145: Despliegue exitoso de *Strimzi* en el clúster de *Kubernetes*

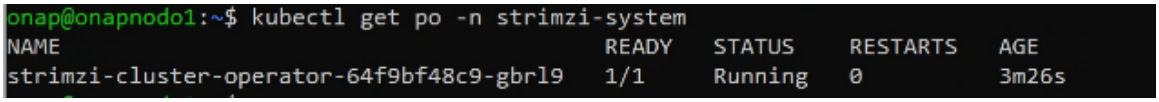


Figura 146: Pods de *Strimzi* corriendo en el clúster de *Kubernetes*

Kafka permite manejar grandes volúmenes de datos, y su uso en ONAP resulta fundamental para la comunicación asíncrona entre sus diferentes módulos. El operador *Strimzi* facilitó la instalación y gestión de los clústeres de *Kafka* y proporcionó un entorno escalable y robusto que es necesario para el funcionamiento de ONAP.

8.2.1.3. Cert Manager

Se instaló *Cert Manager* en el clúster para la gestión automática de certificados *SSL/TLS*. ONAP depende de una infraestructura segura para garantizar la autenticación y la comunicación cifrada entre sus módulos. *Cert Manager* permitió generar y renovar certificados de manera automática, garantizando la seguridad del clúster de *Kubernetes* y facilitando

la configuración de los componentes de ONAP que requieren comunicaciones seguras. La correcta instalación de *Cert Manager*, junto con sus pods corriendo dentro del clúster, se puede visualizar en las figuras 147 y 148.

```
onap@onapnodo1:~$ kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.13.2/cert-manager.yaml
namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.io created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
configmap/cert-manager created
configmap/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-clusterissuers created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-certificates created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-orders created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-challenges created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-ingress-shim created
```

Figura 147: Instalación de *Cert Manager* en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ kubectl get po -n cert-manager
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-6954d7bbbf-f4p5q              1/1     Running   0          46s
cert-manager-cainjector-84bdff4846-2hwh8   1/1     Running   0          46s
cert-manager-webhook-85b6b76d9b-hfhkt      1/1     Running   0          46s
onap@onapnodo1:~$
```

Figura 148: Pods del *Cert Manager* corriendo en el clúster de *Kubernetes*

La correcta instalación de *Cert Manager* ayudó a mantener la integridad y confidencialidad de la información intercambiada entre los microservicios del ecosistema de ONAP.

8.2.1.4. Istio Service Mesh

Para garantizar la conectividad, el control y la seguridad de los microservicios desplegados, se instaló *Istio* como *Service Mesh*. *Istio* proporciona funcionalidades avanzadas de observabilidad, balanceo de carga, y manejo de políticas de seguridad entre los servicios que conforman ONAP. La instalación de *Istio*, su base de datos (*Istiod*) y la verificación de la versión instalada se pueden visualizar en las figuras 149, 150 y 151.

```
onap@onapnodo1:~$ helm upgrade -i istio-base istio/base -n istio-system --version 1.19.3
Release "istio-base" does not exist. Installing it now.
NAME: istio-base
LAST DEPLOYED: Mon Aug 19 01:02:23 2024
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Istio base successfully installed!

To learn more about the release, try:
  $ helm status istio-base
  $ helm get all istio-base
```

Figura 149: Instalación exitosa de *Istio* en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ helm upgrade -i istiod istio/istiod -n istio-system --version 1.19.3 --wait -f ./istiod.yaml
Release "istiod" does not exist. Installing it now.
NAME: istiod
LAST DEPLOYED: Mon Aug 19 01:11:06 2024
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
"istiod" successfully installed!

To learn more about the release, try:
  $ helm status istiod
  $ helm get all istiod

Next steps:
  * Deploy a Gateway: https://istio.io/latest/docs/setup/additional-setup/gateway/
  * Try out our tasks to get started on common configurations:
    * https://istio.io/latest/docs/tasks/traffic-management
    * https://istio.io/latest/docs/tasks/security/
    * https://istio.io/latest/docs/tasks/policy-enforcement/
  * Review the list of actively supported releases, CVE publications and our hardening guide:
    * https://istio.io/latest/docs/releases/supported-releases/
    * https://istio.io/latest/news/security/
    * https://istio.io/latest/docs/ops/best-practices/security/

For further documentation see https://istio.io website
onap@onapnodo1:~$
```

Figura 150: Instalación de la base de datos para *Istio* al correr en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ onap@onapnodo1:~$ kubectl apply -f envoyfilter-case.yaml
envoyfilter.networking.istio.io/header-casing created
envoyfilter.networking.istio.io/header-casing-outbound created
```

Figura 151: Chequeo de versión instalada de *Istio*

Con la instalación exitosa de *Istio*, se logró asegurar que el tráfico entre los diferentes microservicios fuese monitoreado y protegido, agregando una capa adicional de seguridad al clúster. *Istio* también facilitó la identificación de problemas de conectividad entre los módulos de ONAP, ayudando a entender mejor el comportamiento de la plataforma.

8.2.1.5. Gateway API

Se instaló el *Gateway API* en el clúster para gestionar las peticiones de entrada y asegurar que estas fueran dirigidas correctamente a los microservicios correspondientes. *Gateway API* proporcionó una interfaz moderna y flexible para manejar el tráfico de entrada al clúster, algo fundamental para una plataforma distribuida y compleja como ONAP.

```
onap@onapnodo1:~$ kubectl apply -f common-gateway.yaml
gateway.gateway.networking.k8s.io/common-gateway created
```

Figura 152: Configuración aplicada al *Gateway API* para un despliegue exitoso en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ helm upgrade -i istiod istio/istiod -n istio-system --version 1.19.3 --wait -f ./istiod.yaml
Release "istiod" does not exist. Installing it now.
NAME: istiod
LAST DEPLOYED: Mon Aug 19 01:11:06 2024
NAMESPACE: istio-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
"istiod" successfully installed!

To learn more about the release, try:
  $ helm status istiod
  $ helm get all istiod

Next steps:
  * Deploy a Gateway: https://istio.io/latest/docs/setup/additional-setup/gateway/
  * Try out our tasks to get started on common configurations:
    * https://istio.io/latest/docs/tasks/traffic-management
    * https://istio.io/latest/docs/tasks/security/
    * https://istio.io/latest/docs/tasks/policy-enforcement/
  * Review the list of actively supported releases, CVE publications and our hardening guide:
    * https://istio.io/latest/docs/releases/supported-releases/
    * https://istio.io/latest/news/security/
    * https://istio.io/latest/docs/ops/best-practices/security/

For further documentation see https://istio.io website
onap@onapnodo1:~$
```

Figura 153: Instalación de *Gateway API* en el clúster de *Kubernetes*

El despliegue del *Gateway API* permitió definir rutas para las solicitudes y asegurarse de que cada petición llegara al servicio adecuado. Esto resultó especialmente útil para manejar las conexiones entrantes a ONAP y las interacciones con otros servicios externos.

8.2.1.6. Keycloak

Para gestionar la autenticación y autorización de los usuarios y servicios, se instaló *Keycloak* como sistema de identidad y gestión de accesos. *Keycloak* facilitó la configuración de permisos y usuarios dentro del clúster, garantizando un acceso seguro a los diferentes módulos de ONAP. En las figuras 154, 155 y 156 se puede visualizar la instalación de la base de datos para *Keycloak*, la instalación de *Keycloak* en el clúster y los *pods* de *Keycloak* corriendo en el clúster, respectivamente.

```
onap@onapnodo1:~$ helm -n keycloak upgrade -i keycloak-db bitnami/postgresql --values ./keycloak-db-values.yaml
Release "keycloak-db" does not exist. Installing it now.
NAME: keycloak-db
LAST DEPLOYED: Tue Aug 20 22:22:55 2024
NAMESPACE: keycloak
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: postgresql
CHART VERSION: 15.5.23
APP VERSION: 16.4.0

** Please be patient while the chart is being deployed **

PostgreSQL can be accessed via port 5432 on the following DNS names from within your cluster:

    keycloak-db-postgresql.keycloak.svc.cluster.local - Read/Write connection

To get the password for "postgres" run:

    export POSTGRES_ADMIN_PASSWORD=$(kubectl get secret --namespace keycloak keycloak-db-postgresql -o jsonpath="{.data.
```

Figura 154: Instalación de la base de datos para *Keycloak* en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ helm -n keycloak upgrade -i keycloak codecentric/keycloakx --values ./keycloak-server-values.yaml
Release "keycloak" does not exist. Installing it now.
NAME: keycloak
LAST DEPLOYED: Tue Aug 20 22:27:36 2024
NAMESPACE: keycloak
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
******************************************************************
*                                                                *
*               Keycloak.X Helm Chart by codecentric AG          *
*                                                                *
******************************************************************

Keycloak was installed with a Service of type ClusterIP

Create a port-forwarding with the following commands:

export POD_NAME=$(kubectl get pods --namespace keycloak -l "app.kubernetes.io/name=keycloakx,app.kubernetes.io/instance=
keycloak" -o name)
echo "Visit http://127.0.0.1:8080 to use your application"
kubectl --namespace keycloak port-forward "$POD_NAME" 8080
```

Figura 155: Instalación de *Keycloak*, a través de Helm, en el clúster de *Kubernetes*

```
onap@onapnodo1:~$ kubectl get pods -n keycloak
NAME                        READY   STATUS    RESTARTS       AGE
keycloak-db-postgresql-0    1/1     Running   78 (15h ago)   4d20h
keycloak-keycloakx-0        1/1     Running   0              109s
```

Figura 156: Pods de *Keycloak* corriendo en el clúster de *Kubernetes*

Con la correcta configuración de *Keycloak*, se establecieron mecanismos de autenticación basados en *tokens*, lo que aseguró que solo los usuarios autorizados pudieran interactuar con los servicios desplegados. Esta autenticación fue clave para mantener la seguridad del entorno y proteger los recursos críticos de ONAP.

8.2.2. Despliegue de ONAP

Luego de configurar la plataforma base utilizando todos los componentes mencionados, se procedió al despliegue de ONAP. Para esto, se clonó el repositorio oficial de OOM (*ONAP Operations Manager*) (tal como se mira en la Figura 142) y se prepararon los archivos de configuración necesarios, incluyendo el ajuste de los valores en el archivo *values.yaml* de *Helm*, que permite personalizar los parámetros de despliegue de ONAP, tal como se visualiza en la Figura 157.

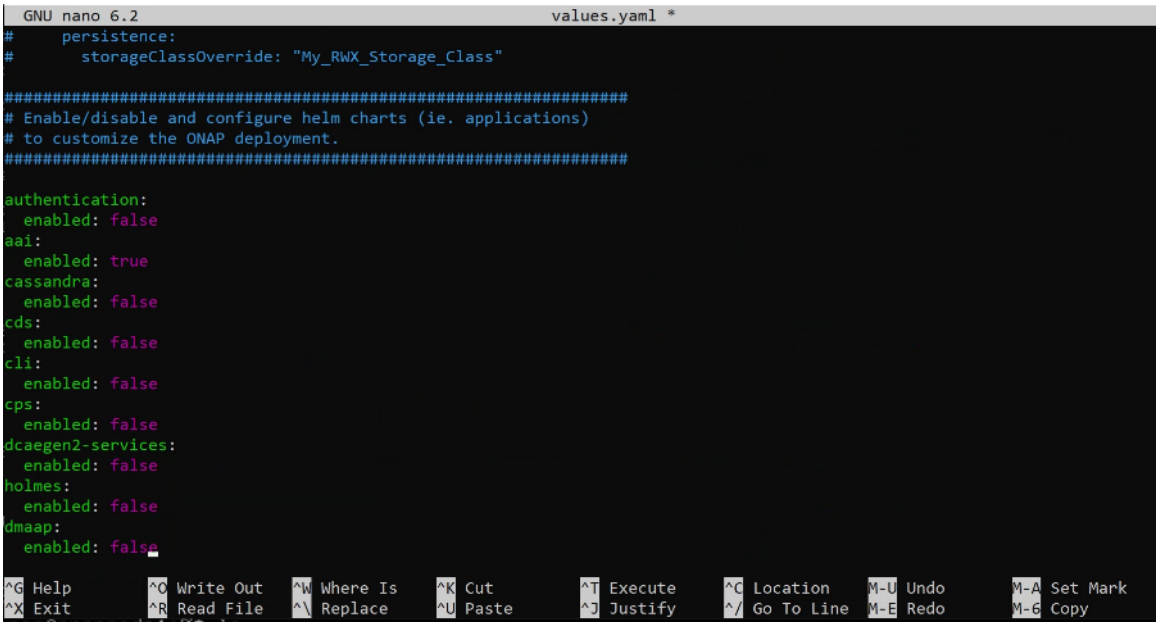


Figura 157: Edición de valores en el archivo *values.yaml* para editar los módulos a desplegar en el despliegue de ONAP

Todos los parámetros colocados dentro del archivos de *"values.yaml"* se puede visualizar en el bloque de código 107.

```
# Copyright 2019 Amdocs, Bell Canada
# Copyright 2020 Nordix Foundation, Modifications
# Modifications Copyright 2020-2021 Nokia
# Modifications Copyright 2023 Nordix Foundation
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#       http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

```

```
################################################################
# Global configuration overrides.
#
# These overrides will affect all helm charts (ie. applications)
# that are listed below and are 'enabled'.
################################################################
global:
  # Change to an unused port prefix range to prevent port conflicts
  # with other instances running within the same k8s cluster
  nodePortPrefix: 302
  nodePortPrefixExt: 304

  # ONAP Repository
  # Four different repositories are used
  # You can change individually these repositories to ones that will serve
    ↪ the
  # right images. If credentials are needed for one of them, see below.
  repository: nexus3.onap.org:10001
  dockerHubRepository: &dockerHubRepository docker.io
  elasticRepository: &elasticRepository docker.elastic.co
  quayRepository: quay.io
  googleK8sRepository: k8s.gcr.io
  githubContainerRegistry: ghcr.io

  # Default credentials
  # they're optional. If the target repository doesn't need them, comment
    ↪ them
  repositoryCred:
    user: docker
    password: docker
  # If you want / need authentication on the repositories, please set
  # Don't set them if the target repo is the same than others
  # so id you've set repository to value `my.private.repo` and same for
  # dockerHubRepository, you'll have to configure only repository (exclusive)
    ↪  OR
  # dockerHubCred.
  # dockerHubCred:
  #   user: myuser
  #   password: mypassord
  # elasticCred:
  #   user: myuser
  #   password: mypassord
  # googleK8sCred:
  #   user: myuser
  #   password: mypassord

  # Default definition of the secret containing the docker image repository
  # credentials. In the default ONAP deployment the secret is created by the
  # repository-wrapper component, which uses the secrets defined above.
  # If this is not wanted or other secrets are created, alternative secret
  # names can be used
  # Overrides for specific images can be done, if the "image" entry is used
    ↪ as
  # a map and the "pullSecrets" is used, e.g.
  # image:
  #   ...
  #   pullSecrets:
  #     - myRegistryKeySecretName
  #
```

```
  imagePullSecrets:
    - '{{ include "common.namespace" . }}-docker-registry-key'

  # common global images
  # Busybox for simple shell manipulation
  busyboxImage: busybox:1.34.1

  # curl image
  curlImage: curlimages/curl:7.80.0

  # env substitution image
  envsubstImage: dibi/envsubst:1

  # generate htpasswd files image
  # there's only latest image for htpasswd
  htpasswdImage: xmartlabs/htpasswd:latest

  # kubenretes client image
  kubectlImage: bitnami/kubectl:1.22.4

  # logging agent
  loggingImage: beats/filebeat:5.5.0

  # mariadb client image
  mariadbImage: bitnami/mariadb:10.5.8

  # mongodb server image
  mongodbImage: percona/percona-server-mongodb:7.0.5-3

  # nginx server image
  nginxImage: bitnami/nginx:1.21.4

  # postgreSQL client and server image
  postgresImage: crunchydata/crunchy-postgres:centos8-13.2-4.6.1

  # readiness check image
  readinessImage: onap/oom/readiness:6.0.3

  # image pull policy
  pullPolicy: Always

  # default java image
  jreImage: onap/integration-java11:10.0.0

  # default clusterName
  # {{ template "common.fullname" . }}.{{ template "common.namespace" . }}.
      ↪ svc.{{ .Values.global.clusterName }}
  clusterName: cluster.local

  # default mount path root directory referenced
  # by persistent volumes and log files
  persistence:
    mountPath: /dockerdata-nfs
    enableDefaultStorageclass: false
    parameters: {}
    storageclassProvisioner: kubernetes.io/no-provisioner
    volumeReclaimPolicy: Retain

  # Global flag to enable the creation of default roles instead of using
```

```yaml
  # common roles-wrapper
  createDefaultRoles: false

  # override default resource limit flavor for all charts
  flavor: unlimited

  # flag to enable debugging - application support required
  debugEnabled: false

  # default password complexity
  # available options: phrase, name, pin, basic, short, medium, long, maximum
    ↪  security
  # More datails: https://www.masterpasswordapp.com/masterpassword-algorithm.
    ↪ pdf
  passwordStrength: long

  # configuration to set log level to all components (the one that are using
  # "common.log.level" to set this)
  # can be overrided per components by setting logConfiguration.
    ↪ logLevelOverride
  # to the desired value

  # Global ingress configuration
  ingress:
    # generally enable ingress for ONAP components
    enabled: false
    # enable all component's Ingress interfaces
    enable_all: false

    # Provider: ingress, istio, gw-api
    provider: istio
    # Ingress class (only for provider "ingress"): e.g. nginx, traefik
    ingressClass:
    # Ingress Selector (only for provider "istio") to match with the
    # ingress pod label "istio=ingress"
    ingressSelector: ingress
    # optional: common used Gateway (for Istio, GW-API) and listener names
    commonGateway:
      name: ""
      httpListener: ""
      httpsListener: ""

    # default Ingress base URL and preAddr- and postAddr settings

    # Ingress URLs result:
    # <preaddr><component.ingress.service.baseaddr><postaddr>.<baseurl>
    virtualhost:
      # Default Ingress base URL
      # can be overwritten in component by setting ingress.baseurlOverride
      baseurl: "simpledemo.onap.org"
      # prefix for baseaddr
      # can be overwritten in component by setting ingress.preaddrOverride
      preaddr: ""
      # postfix for baseaddr
      # can be overwritten in component by setting ingress.postaddrOverride
      postaddr: ""

    # All http (port 80) requests via ingress will be redirected
    # to port 443 on Ingress controller
```

```
    # only valid for Istio Gateway (ServiceMesh enabled)
    config:
      ssl: "redirect"
    # you can set an own Secret containing a certificate
    # only valid for Istio Gateway (ServiceMesh enabled)
    #  tls:
    #    secret: 'my-ingress-cert'

    # optional: Namespace of the Istio IngressGateway or Gateway-API
    # only valid for Istio Gateway (ServiceMesh enabled)
    namespace: istio-ingress

# Global Service Mesh configuration
serviceMesh:
  enabled: false
  tls: true
  # be aware that linkerd is not well tested
  engine: "istio" # valid value: istio or linkerd
  # if nativeSidecars are enabled in Istio, this value can be set to "true"
  # and will disable the deployment of sidecar killer containers in jobs
  nativeSidecars: false

# Global Istio Authorization Policy configuration
authorizationPolicies:
  enabled: false

# metrics part
# If enabled, exporters (for prometheus) will be deployed
# if custom resources set to yes, CRD from prometheus operartor will be
# created
# Not all components have it enabled.
#
metrics:
  enabled: true
  custom_resources: false

# Disabling AAF
# POC Mode, only for use in development environment
# Keep it enabled in production
aafEnabled: true

# Disabling MSB
# POC Mode, only for use in development environment
msbEnabled: true

# default values for certificates
certificate:
  default:
    renewBefore: 720h #30 days
    duration:    8760h #365 days
    subject:
      organization: "Linux-Foundation"
      country: "US"
      locality: "San-Francisco"
      province: "California"
      organizationalUnit: "ONAP"
    issuer:
      group: certmanager.onap.org
      kind: CMPv2Issuer
```

```
246         name: cmpv2-issuer-onap
247
248   # Enabling CMPv2
249   cmpv2Enabled: false
250   platform:
251     certificates:
252       clientSecretName: oom-cert-service-client-tls-secret
253       keystoreKeyRef: keystore.jks
254       truststoreKeyRef: truststore.jks
255       keystorePasswordSecretName: oom-cert-service-certificates-password
256       keystorePasswordSecretKey: password
257       truststorePasswordSecretName: oom-cert-service-certificates-password
258       truststorePasswordSecretKey: password
259
260   # Indicates offline deployment build
261   # Set to true if you are rendering helm charts for offline deployment
262   # Otherwise keep it disabled
263   offlineDeploymentBuild: false
264
265   # TLS
266   # Set to false if you want to disable TLS for NodePorts. Be aware that this
267   # will loosen your security.
268   # if set this element will force or not tls even if serviceMesh.tls is set.
269   tlsEnabled: false
270
271   # Logging
272   # Currently, centralized logging is not in best shape so it's disabled by
273   # default
274   centralizedLoggingEnabled: &centralizedLogging false
275
276   # Example of specific for the components where you want to disable TLS only
        ↪ for
277   # it:
278   # if set this element will force or not tls even if global.serviceMesh.tls
        ↪ and
279   # global.tlsEnabled is set otherwise.
280   # robot:
281   #   tlsOverride: false
282
283   # Global storage configuration
284   #    Set to "-" for default, or with the name of the storage class
285   #    Please note that if you use AAF, CDS, SDC, Netbox or Robot, you need a
286   #    storageclass with RWX capabilities (or set specific configuration for
        ↪ these
287   #    components).
288   # persistence:
289   #   storageClass: "-"
290
291 # Example of specific for the components which requires RWX:
292 # cds:
293 #   cds-blueprints-processor:
294 #     persistence:
295 #       storageClassOverride: "My_RWX_Storage_Class"
296 # sdc:
297 #   sdc-onboarding-be:
298 #     persistence:
299 #       storageClassOverride: "My_RWX_Storage_Class"
300
301 ################################################################
```

```
# Enable/disable and configure helm charts (ie. applications)
# to customize the ONAP deployment.
################################################################
aaf:
  enabled: true
  aaf-service:
    readiness:
      initialDelaySeconds: 150
authentication:
  enabled: false
aai:
  enabled: false
cassandra:
  enabled: true
  replicaCount: 3
  config:
    cluster_domain: cluster.local
    heap:
      max: 1G
      min: 256M
  liveness:
    initialDelaySeconds: 60
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
    enabled: true
  readiness:
    initialDelaySeconds: 120
    periodSeconds: 20
    timeoutSeconds: 10
    successThreshold: 1
    failureThreshold: 3
cds:
  enabled: false
cli:
  enabled: false
cps:
  enabled: false
dcaegen2-services:
  enabled: false
holmes:
  enabled: false
dmaap:
  enabled: false
  message-router:
    enabled: false
  dmaap-dr-prov:
    enabled: false
  dmaap-dr-node:
    enabled: false
oof:
  enabled: false
mariadb-galera:
  enabled: false
msb:
  enabled: false
multicloud:
  enabled: false
```

```
nbi:
  enabled: false
  config:
    # openstack configuration
    openStackRegion: "Yolo"
    openStackVNFTenantId: "1234"
policy:
  enabled: false
portal-ng:
  enabled: true
robot:
  enabled: false
  config:
    # openStackEncryptedPasswordHere should match the encrypted string used
      ↪ in SO and overridden per environment
    openStackEncryptedPasswordHere: "c124921a3a0efbe579782cde8227681e"
sdc:
  enabled: true
  config:
    environment:
      vnfRepoPort: 8703
  sdc-be:
    config:
      javaOptions: "-Xmx1g -Xns512m"
    liveness:
      periodSeconds: 300
      timeoutSeconds: 180
    readiness:
      periodSeconds: 300
      timeoutSeconds: 240
  sdc-fe:
    resources:
      small:
        limits:
          cpu: 1
          memory: 2Gi
        requests:
          cpu: 100m
          memory: 500Mi

sdnc:
  enabled: false

  replicaCount: 1

  mysql:
    replicaCount: 1
so:
  enabled: false

  replicaCount: 1

  liveness:
    # necessary to disable liveness probe when setting breakpoints
    # in debugger so K8s doesn't restart unresponsive container
    enabled: false

  # so server configuration
  config:
```

```
    # message router configuration
    dmaapTopic: "AUTO"
    # openstack configuration
    openStackUserName: "vnf_user"
    openStackRegion: "RegionOne"
    openStackKeyStoneUrl: "http://1.2.3.4:5000"
    openStackServiceTenantName: "service"
    openStackEncryptedPasswordHere: "c124921a3a0efbe579782cde8227681e"

  # in order to enable static password for so-monitoring uncomment:
  # so-monitoring:
  #   server:
  #     monitoring:
  #       password: demo123456!

strimzi:
  enabled: false
  # Kafka replication & disk storage should be dimensioned
  # according to each given system use case.
  replicaCount: 3
  persistence:
    kafka:
      size: 10Gi
    zookeeper:
      size: 1Gi
  # Strimzi kafka bridge is an optional http api towards
  # kafka provided by https://strimzi.io/docs/bridge/latest/
  strimzi-kafka-bridge:
    enabled: false

uui:
  enabled: false
vfc:
  enabled: false
vnfsdk:
  enabled: false
modeling:
  enabled: false
platform:
  enabled: false
a1policymanagement:
  enabled: false
repository-wrapper:
  enabled: false
roles-wrapper:
  enabled: false
```

Cuadro 107: Valores editados en el archivo *values.yaml* para el despliegue personalizado de ONAP

Se generaron todas las configuraciones para ajustar los valores de configuración de ONAP, lo cual, logró desplegar los diferentes módulos de ONAP en el clúster de *Kubernetes*, tal como se muestra en la Figura 158.

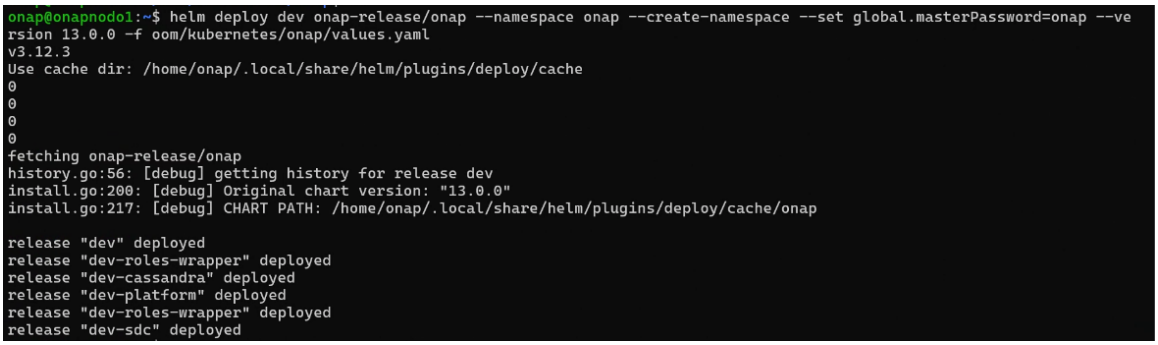


Figura 158: Confirmación de despligue de los diferentes módulos configurados de ONAP en el clúster de *Kubernetes*

8.3. Problemas encontrados durante el despliegue de ONAP

A pesar de la preparación, configuración y despliegue correcto de los módulos de ONAP, no fue posible utilizar instalar y utilizar la misma. Debido a que no funcionó el despliegue de la instancia de ONAP, se identificaron varios problemas críticos que impidieron que ONAP funcionara correctamente.

8.3.1. Ausencia de componentes críticos (Base de datos *Cassandra*)

Uno de los problemas principales observados fue la ausencia de componentes esenciales dentro del *namespace* de ONAP. Específicamente, los *pods* de la base de datos *Cassandra* no estaban presentes, como se confirmó al ejecutar el comando mostrado en el código 108.

```
kubectl get pods -n onap | grep cassandra
```

Cuadro 108: Verificación de la presencia de pods de *Cassandra* en el *namespace* de ONAP

Este comando no devolvió resultados, indicando que *Cassandra* no se desplegó o falló al inicializarse. *Cassandra* es un componente importante para ONAP, ya que funciona como un almacén de datos principal para varios servicios. Sin él, diferentes servicios que dependen de *Cassandra* no pueden funcionar correctamente.

La ausencia de *Cassandra* condujo a fallos en cascada en diversos pods que dependen de ella. Por ejemplo, el *pod* `dev-sdc-cs`, responsable de ciertas funcionalidades del SDC (*Service Design and Creation*), se quedó atascado en la fase de inicialización, indicando repetidamente lo visto en el código 109.

```
2024-11-16 20:51:47,362 - INFO - Checking if container cassandra is ready
```

Cuadro 109: *Logs* en pod dev-sdc-cs indicando que el mismo espera la inicialización de *Cassandra*

Este mensaje indica que el *pod* espera indefinidamente a que *Cassandra* esté disponible.

8.3.2. Falta de recursos en el clúster de *Kubernetes*

Una de las principales razones por las que diferentes componentes esenciales (como el caso de *Cassandra*) no pudieron desplegarse correctamente fue la falta de recursos en el clúster de *Kubernetes*. La falta de recursos, como memoria y CPU, impidió que se pudieran desplegar y ejecutar componentes críticos de ONAP, lo que dió como resultado múltiples fallos en cascada.

Como se mencionó antes, el clúster fue desplegado en un entorno local con recursos limitados. Principalmente, la falta de memoría RAM fue un factor crítico, ya que en algunos nodos solo se podía utilizar un máximo de 15 GB de memoría RAM, lo cual no es suficiente cuando se busca desplegar múltiples componentes de ONAP.

8.3.3. Configuraciones incorrectas y recursos RBAC faltantes

Análisis previos también revelaron problemas con el Control de Acceso Basado en Roles (RBAC). Ciertos roles y vinculaciones de roles estaban ausentes, lo que causó errores de permisos cuando los *pods* intentaron realizar acciones como listar otros *pods*. Esto fue evidente en errores como los mostrados en el código 110.

```
User "system:serviceaccount:onap:dev-sdc-be-read" cannot list resource "pods"
    ↪  in API group "" in the namespace "onap": RBAC: role.rbac.
    ↪ authorization.k8s.io "dev-read" not found
```

Cuadro 110: Problemas con el Control de Acceso Basado en Roles (RBAC)

Si bien este problema buscó resolverse al ajustar configuraciones en el archivo *"values.yaml"* para habilitar la creación de roles predeterminados o habilitar el módulo de *roles-wrapper*, la ausencia de otros componentes críticos como *Cassandra* debido a falta de recursos, implicaba un mayor problema dentro del despliegue.

8.3.4. Aprendizajes adquiridos

Como resultado del intento de desplegar ONAP, se logró ver que la complejidad de la arquitectura de ONAP, combinada con la necesidad de una alta coordinación entre sus múltiples componentes, representó un gran desafío que, finalmente, no permitió la correcta operación de la plataforma.

Entre los aprendizajes más importantes, se puede hablar sobre la importancia de una correcta planificación y preparación del entorno de despliegue. El clúster de *Kubernetes* se desplegó con alta disponibilidad y contaba con los recursos necesarios, pero la integración de los diversos componentes de ONAP, junto con la falta de documentación clara en algunos aspectos fundamentales, complicaron el proceso. Además, la configuración de módulos como *SDN-C*, *SDC* y *SO* indicaba la gran necesidad de tener una alta compatibilidad entre versiones de software y mejores herramientas de monitoreo para identificar y resolver problemas en tiempo real.

A pesar de no haber logrado el despliegue completo de ONAP, el proceso permitió obtener conocimientos sobre la arquitectura de la plataforma, dependencias entre sus módulos, y los diversos requisitos previos para desplegar un entorno de automatización de redes a gran escala. Estos aprendizajes son fundamentales para futuros intentos de implementar plataformas de orquestación similares y para mejorar la preparación de la infraestructura necesaria.

8.4. API desarrollada para la automatización de redes

Para demostrar la correcta funcionalidad del clúster de *Kubernetes* y de los diferentes componentes desplegados, se desarrolló una API que permite la automatización de procesos como la configuración y el monitoreo en redes. Esta API, desarrollada en *Python*, se encarga de realizar operaciones básicas de configuración, tales como la asignación de IPs a interfaces de routers, el apagado y encendido de interfaces, el cambio del *ID* de router en OSPF, y la adición o eliminación de redes en un proceso OSPF. También realiza tareas de monitoreo, como la visualización de la tabla de enrutamiento de un router, el estado de las interfaces, los vecinos OSPF, y otros datos extraídos a través de SNMP.

8.4.1. Estructura de la API

La API fue desarrollada utilizando el framework *FastAPI*, junto con bibliotecas como *Netmiko* y *PySNMP* para facilitar la automatización de procesos y el monitoreo de redes. La estructura del proyecto, como se muestra en el código 111, sigue una organización modular, donde cada componente está separado en diferentes archivos y carpetas, permitiendo un mayor control y mantenibilidad del código. Esta división facilita el desarrollo de múltiples *endpoints* para realizar diversas acciones, tales como la configuración de dispositivos de red y la obtención de datos mediante SNMP, asegurando una arquitectura clara y escalable para la API.

```
PRUEBA_DE_CONCEPTO/
+-- app/
|   +-- routers/
|   |   +-- config.py
|   |   +-- monitor.py
|   +-- static/
|   +-- templates/
|   |   +-- config/
|   |   |   +-- config.html
|   |   |   +-- interface_ip_config.html
|   |   |   +-- interface.html
|   |   |   +-- ospf_network_remove.html
|   |   |   +-- ospf_network.html
|   |   |   +-- ospf_router_id.html
|   |   +-- monitor/
|   |   |   +-- snmp/
|   |   |   |   +-- diagram_form.html
|   |   |   |   +-- diagram.html
|   |   |   |   +-- execute_command.html
```

```
|   |   |   |   +-- interfaces.html
|   |   |   |   +-- monitor.html
|   |   |   |   +-- ospf_neighbors.html
|   |   |   |   +-- routing_table.html
|   |   |   |   +-- uptime.html
|   |   |   +-- base.html
|   |   |   +-- index.html
|   +-- utils/
|       +-- cdp_helpers.py
|       +-- netmiko_helpers_prueba.py
|       +-- netmiko_snmp_helpers.py
+-- main.py
+-- helm-chart/
+-- .dockerignore
+-- docker-compose.yml
+-- Dockerfile
+-- netmiko_session.log
+-- poetry.lock
+-- pyproject.toml
+-- requirements.txt
```

Cuadro 111: Estructura de la API realizada

En este caso, la API cuenta con dos módulos principales: *routers/config.py* y *routers/monitor.py*. El primero se encarga de la configuración de dispositivos de red, mientras que el segundo se encarga del monitoreo de los mismos. Cada uno de estos módulos contiene diferentes *endpoints* que utilizan funciones específicas extraídas del archivo *utils/netmiko_snmp_helpers.py* para realizar las conexiones a los dispositivos de red y ejecutar comandos de configuración y monitoreo. En los códigos 118, 114 y 115 se pueden visualizar los códigos de *routers/config.py*, *routers/monitor.py* y *utils/netmiko_snmp_helpers.py*, respectivamente.

```python
# ------------------------------------------------------------------------------
# Importación de librerías
# ------------------------------------------------------------------------------
from fastapi import FastAPI, Request
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates

# ------------------------------------------------------------------------------
# Crear instancia de FastAPI
# ------------------------------------------------------------------------------
app = FastAPI(title="Network Automation App", version="0.1")

# ------------------------------------------------------------------------------
# Montar directorio de archivos estáticos
# ------------------------------------------------------------------------------
app.mount("/static", StaticFiles(directory="app/static"), name="static")

# ------------------------------------------------------------------------------
# Crear instancia de Jinja2Templates
# ------------------------------------------------------------------------------
templates = Jinja2Templates(directory="app/templates")

# ------------------------------------------------------------------------------
# Importar rutas
# ------------------------------------------------------------------------------
from app.routers import config, monitor
```

```python

# ------------------------------------------------------------------------------
# Registrar rutas en la instancia de FastAPI
# ------------------------------------------------------------------------------
app.include_router(config.router)
app.include_router(monitor.router)

# ------------------------------------------------------------------------------
# Ruta principal
# ------------------------------------------------------------------------------
@app.get("/")
def read_root(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})
```

Cuadro 112: Código que contiene la página principal de nuestra API

```python
# Descripción: Rutas para cuando se quiera configurar un dispositivo de red.

# ------------------------------------------------------------------------------
# Importaciones de librerías y módulos
# ------------------------------------------------------------------------------
# Librerías
from fastapi import APIRouter, Request, Form
from fastapi.responses import HTMLResponse

# Módulos
from app.main import templates
from app.utils.netmiko_snmp_helpers import (
    shutdown_interface,
    configure_interface_ip,
    change_ospf_router_id,
    add_ospf_network,
    remove_ospf_network,
)

# ------------------------------------------------------------------------------
# Crear instancia de APIRouter
# ------------------------------------------------------------------------------
router = APIRouter()

# ------------------------------------------------------------------------------
# Rutas
# ------------------------------------------------------------------------------
# Ruta principal de configuración
@router.get("/config", response_class=HTMLResponse)
async def config(request: Request):
    return templates.TemplateResponse("/config/config.html", {"request": request})

# Ruta para Apagar/Encender interfaz
@router.get("/config/interface", response_class=HTMLResponse)
async def interface(request: Request):
    return templates.TemplateResponse("/config/interface.html", {"request": request})

@router.post("/config/interface")
async def interface(
    request: Request,
    ip: str = Form(...),
    interface: str = Form(...),
    action: str = Form(...),
):
    if action == "shutdown":
        output = shutdown_interface(ip, interface, shutdown=True)
    elif action == "no_shutdown":
        output = shutdown_interface(ip, interface, shutdown=False)
    else:
        output = "Acción no válida"
```

```
    return templates.TemplateResponse("/config/interface.html", {"request": request,
        ↪ "output": output})

# Ruta para Configurar IP en interfaz
@router.get("/config/interface/ip", response_class=HTMLResponse)
async def interface_ip_config_page(request: Request):
    return templates.TemplateResponse("/config/interface_ip_config.html",
        ↪ {"request": request})

@router.post("/config/interface/ip")
async def interface_ip_config(
    request: Request,
    ip: str = Form(...),
    interface: str = Form(...),
    ip_address: str = Form(...),
    subnet_mask: str = Form(...),
):
    output = configure_interface_ip(ip, interface, ip_address, subnet_mask)
    return templates.TemplateResponse(
        "/config/interface_ip_config.html", {"request": request, "output": output}
    )

# Ruta para Cambiar Router ID de OSPF
@router.get("/config/ospf/router_id", response_class=HTMLResponse)
async def ospf_router_id_page(request: Request):
    return templates.TemplateResponse("/config/ospf_router_id.html", {"request":
        ↪ request})

@router.post("/config/ospf/router_id")
async def ospf_router_id(
    request: Request,
    ip: str = Form(...),
    router_id: str = Form(...),
):
    output = change_ospf_router_id(ip, router_id)
    return templates.TemplateResponse("/config/ospf_router_id.html", {"request":
        ↪ request, "output": output})

# Ruta para Agregar Red OSPF
@router.get("/config/ospf/network", response_class=HTMLResponse)
async def ospf_network_page(request: Request):
    return templates.TemplateResponse("/config/ospf_network.html", {"request":
        ↪ request})

@router.post("/config/ospf/network")
async def ospf_network(
    request: Request,
    ip: str = Form(...),
    network: str = Form(...),
    wildcard_mask: str = Form(...),
    process_id: str = Form(...),
    area: str = Form(...),
):
    output = add_ospf_network(ip, network, wildcard_mask, area, process_id)
    return templates.TemplateResponse("/config/ospf_network.html", {"request":
        ↪ request, "output": output})

# Ruta para Eliminar Red OSPF
@router.get("/config/ospf/network/remove", response_class=HTMLResponse)
async def ospf_network_remove_page(request: Request):
    return templates.TemplateResponse("/config/ospf_network_remove.html",
        ↪ {"request": request})

@router.post("/config/ospf/network/remove")
async def ospf_network_remove(
    request: Request,
    ip: str = Form(...),
    network: str = Form(...),
```

```
    wildcard_mask: str = Form(...),
    process_id: str = Form(...),
    area: str = Form(...),
):
    output = remove_ospf_network(ip, network, wildcard_mask, area, process_id)
    return templates.TemplateResponse("/config/ospf_network_remove.html",
        ↪ {"request": request, "output": output})
```

Cuadro 113: Código que contiene los *endpoints* para la sección de configuración

```
# Descripción: Rutas para monitorear un dispositivo de red.

# ------------------------------------------------------------------------------
# Importaciones de librerías y módulos
# ------------------------------------------------------------------------------
# Librerías
from fastapi import APIRouter, Request, Form, Query, HTTPException
from fastapi.responses import HTMLResponse
from itertools import zip_longest
import ipaddress
import struct
from typing import List, Dict

# Módulos
from app.main import templates
from app.utils.netmiko_snmp_helpers import (
    get_interface_status,
    get_routing_table,
    get_ospf_neighbors,
    get_device_uptime,
    execute_custom_command,
    snmp_get,
    snmp_walk,
    snmp_oids,
)

# ------------------------------------------------------------------------------
# Crear instancia de APIRouter
# ------------------------------------------------------------------------------
router = APIRouter()

# ------------------------------------------------------------------------------
# Rutas
# ------------------------------------------------------------------------------
# Ruta principal de monitoreo
@router.get("/monitor", response_class=HTMLResponse)
async def monitor(request: Request):
    return templates.TemplateResponse("/monitor/monitor.html", {"request": request})

# Ruta para obtener información de interfaces
@router.get("/monitor/interfaces", response_class=HTMLResponse)
async def interfaces_status_page(request: Request):
    return templates.TemplateResponse("/monitor/interfaces.html", {"request":
        ↪ request})

@router.post("/monitor/interfaces")
async def interfaces_status(
    request: Request,
    ip: str = Form(...),
    community: str = Form("public"),
):
```

```
    oid_if_descr = '1.3.6.1.2.1.2.2.1.2'  # OID para la descripción de las interfaces
    oid_if_oper_status = '1.3.6.1.2.1.2.2.1.8'  # OID para el estado operativo de
        ↪ las interfaces

    # Realizar SNMP walk para obtener las descripciones de las interfaces
    interfaces = snmp_walk(ip, community, oid_if_descr)

    # Realizar SNMP walk para obtener el estado operativo de las interfaces
    statuses = snmp_walk(ip, community, oid_if_oper_status)

    if not interfaces or not statuses:
        error_message = "No se pudo obtener la información de las interfaces"
        return templates.TemplateResponse("/monitor/interfaces.html", {"request":
            ↪ request, "error": error_message})

    # Procesar los resultados y crear una lista de interfaces con su estado
    interfaces_status_list = []
    for (oid_descr, if_descr), (oid_status, if_status) in zip(interfaces, statuses):
        status = "up" if if_status == '1' else "down"
        interfaces_status_list.append({"interface": if_descr, "status": status})

    # Renderizar la plantilla con la lista de interfaces y su estado
    return templates.TemplateResponse("/monitor/interfaces.html", {"request":
        ↪ request, "output": interfaces_status_list, "ip": ip})

# Ruta para obtener tabla de enrutamiento
@router.get("/monitor/routing-table", response_class=HTMLResponse)
async def routing_table_page(request: Request):
    return templates.TemplateResponse("/monitor/routing_table.html", {"request":
        ↪ request})

@router.post("/monitor/routing-table")
async def routing_table(
    request: Request,
    ip: str = Form(...),
):
    try:
        # Utilizar la función de Netmiko o SNMP para obtener la tabla de enrutamiento
        routes = get_routing_table(ip)  # Debería devolver una lista de tuplas con
            ↪ el tipo, IP, e interfaz
    except ValueError as e:
        return templates.TemplateResponse("/monitor/routing_table.html", {"request":
            ↪ request, "error": str(e)})

    # Formatear los datos de la tabla de enrutamiento para que sean utilizados en la
        ↪ plantilla
    routing_table = [
        {"route_type": route[0], "ip": route[1], "interface": route[2]}
        for route in routes
    ]

    # Renderizar la tabla de enrutamiento
    return templates.TemplateResponse("/monitor/routing_table.html", {"request":
        ↪ request, "routing_table": routing_table, "ip": ip})

# Ruta para obtener vecinos OSPF
@router.get("/monitor/ospf-neighbors", response_class=HTMLResponse)
async def ospf_neighbors_page(request: Request):
    return templates.TemplateResponse("/monitor/ospf_neighbors.html", {"request":
        ↪ request})

@router.post("/monitor/ospf-neighbors")
async def ospf_neighbors(
    request: Request,
    ip: str = Form(...),
):
    ospf_neighbors = get_ospf_neighbors(ip)
    return templates.TemplateResponse("/monitor/ospf_neighbors.html", {"request":
```

```
        ↪ request, "output": ospf_neighbors})

# Ruta para obtener tiempo de actividad del dispositivo
@router.get("/monitor/uptime", response_class=HTMLResponse)
async def device_uptime_page(request: Request):
    return templates.TemplateResponse("/monitor/uptime.html", {"request": request})

@router.post("/monitor/uptime")
async def device_uptime(
    request: Request,
    ip: str = Form(...),
):
    uptime = get_device_uptime(ip)
    return templates.TemplateResponse("/monitor/uptime.html", {"request": request,
        ↪ "output": uptime})

# Ruta para ejecutar comando personalizado
@router.get("/monitor/custom-command", response_class=HTMLResponse)
async def custom_command_page(request: Request):
    return templates.TemplateResponse("/monitor/execute_command.html", {"request":
        ↪ request})

@router.post("/monitor/custom-command")
async def custom_command(
    request: Request,
    ip: str = Form(...),
    command: str = Form(...),
):
    output = execute_custom_command(ip, command)
    return templates.TemplateResponse("/monitor/execute_command.html", {"request":
        ↪ request, "output": output})

# ------------------------------------------------------------------------------
# SNMP
# ------------------------------------------------------------------------------
# Ruta principal de SNMP
@router.get("/monitor/snmp/snmp", response_class=HTMLResponse)
async def snmp_page(request: Request):
    return templates.TemplateResponse("/monitor/snmp/snmp.html", {"request":
        ↪ request})

# Ruta para obtener uso de CPU
@router.get("/monitor/snmp/cpu-usage", response_class=HTMLResponse)
async def cpu_usage_page(request: Request):
    return templates.TemplateResponse("/monitor/snmp/cpu_usage.html", {"request":
        ↪ request})

@router.post("/monitor/snmp/cpu-usage")
async def cpu_usage(
    request: Request,
    ip: str = Form(...),
    community: str = Form("public"),
):
    oid = snmp_oids["cpu-usage"]
    value = snmp_get(ip, community, oid)
    if value is None:
        error_message = "No se pudo obtener el uso de CPU"
        return templates.TemplateResponse("/monitor/snmp/cpu_usage.html",
            ↪ {"request": request, "error": error_message})
    return templates.TemplateResponse("/monitor/snmp/cpu_usage.html", {"request":
        ↪ request, "output": value, "ip": ip})

# Ruta para obtener uso de memoria
@router.get("/monitor/snmp/memory-usage", response_class=HTMLResponse)
async def memory_usage_page(request: Request):
    return templates.TemplateResponse("/monitor/snmp/memory_usage.html", {"request":
        ↪ request})

```

```python
@router.post("/monitor/snmp/memory-usage")
async def memory_usage(
    request: Request,
    ip: str = Form(...),
    community: str = Form("public"),
):
    oid = snmp_oids["memory-usage"]
    value = snmp_get(ip, community, oid)
    if value is None:
        error_message = "No se pudo obtener el uso de memoria"
        return templates.TemplateResponse("/monitor/snmp/memory_usage.html",
            ↪ {"request": request, "error": error_message})
    return templates.TemplateResponse("/monitor/snmp/memory_usage.html", {"request":
        ↪ request, "output": value, "ip": ip})

# Ruta para información del sistema
@router.get("/monitor/snmp/system-info", response_class=HTMLResponse)
async def system_info_page(request: Request):
    return templates.TemplateResponse("/monitor/snmp/system_info.html", {"request":
        ↪ request})

@router.post("/monitor/snmp/system-info")
async def system_info(
    request: Request,
    ip: str = Form(...),
    community: str = Form("public"),
):
    oid = snmp_oids["system-info"]
    value = snmp_get(ip, community, oid)
    if value is None:
        error_message = "No se pudo obtener la información del sistema"
        return templates.TemplateResponse("/monitor/snmp/system_info.html",
            ↪ {"request": request, "error": error_message})
    return templates.TemplateResponse("/monitor/snmp/system_info.html", {"request":
        ↪ request, "output": value, "ip": ip})

# Ruta para paquetes con errores
@router.get("/monitor/snmp/error-packets", response_class=HTMLResponse)
async def error_packets_page(request: Request):
    return templates.TemplateResponse("/monitor/snmp/error_packets.html",
        ↪ {"request": request})

@router.post("/monitor/snmp/error-packets")
async def error_packets(
    request: Request,
    ip: str = Form(...),
    community: str = Form("public"),
):
    oid_in_errors = '1.3.6.1.2.1.2.2.1.14'   # Errores de entrada
    oid_out_errors = '1.3.6.1.2.1.2.2.1.20'  # Errores de salida
    oid_if_descr = '1.3.6.1.2.1.2.2.1.2'     # Descripción de la interfaz

    in_errors = snmp_walk(ip, community, oid_in_errors)
    out_errors = snmp_walk(ip, community, oid_out_errors)
    if_descr = snmp_walk(ip, community, oid_if_descr)

    if in_errors is None or out_errors is None or if_descr is None:
        error_message = "No se pudo obtener la información de paquetes con errores"
        return templates.TemplateResponse("/monitor/snmp/error_packets.html",
            ↪ {"request": request, "error": error_message})

    errors = []
    for ((_, if_name), (_, in_error_count), (_, out_error_count)) in
        ↪ zip_longest(if_descr, in_errors, out_errors, fillvalue=(None, 'N/A')):
        errors.append({
            "interface": if_name,
            "in_errors": int(in_error_count) if in_error_count.isdigit() else 'N/A',
            "out_errors": int(out_error_count) if out_error_count.isdigit() else
```

```
                ↪ 'N/A'
        })

    return templates.TemplateResponse("/monitor/snmp/error_packets.html",
        ↪ {"request": request, "output": errors, "ip": ip})
```

Cuadro 114: Código que contiene los *endpoints* para la sección de monitoreo

```
# Descripción: Código con funciones auxiliares para interactuar con dispositivos de
    ↪ red utilizando Netmiko.

# ****************************************************************************
# FUNCIONES PRINCIPALES
# ****************************************************************************
# ----------------------------------------------------------------------------
# Importaciones de librerías
# ----------------------------------------------------------------------------
from netmiko import ConnectHandler
from netmiko import NetmikoAuthenticationException, NetmikoTimeoutException
from pysnmp.hlapi import getCmd, nextCmd, SnmpEngine, CommunityData,
    ↪ UdpTransportTarget, ContextData, ObjectType, ObjectIdentity
import re

# ----------------------------------------------------------------------------
# Creación de diccionario de credenciales
# ----------------------------------------------------------------------------
def get_device_connection(ip):
    device = {
        "device_type": "cisco_ios_telnet",
        "host": ip,
        "username": "cisco",
        "password": "cisco",
        "secret": "cisco",
        "port": 23,
        "session_log": "netmiko_session.log",
    }

    try:
        net_connect = ConnectHandler(**device)
        net_connect.enable()
    except Exception as e:
        print(f"Error: {e}")
    return net_connect

# ****************************************************************************
# DICCIONARIO PARA FUNCIONES AUXILIARES
# ****************************************************************************
# ----------------------------------------------------------------------------
# OID de la MIB SNMP
# ----------------------------------------------------------------------------
snmp_oids = {
    "cpu-usage": "1.3.6.1.4.1.9.9.109.1.1.1.1.8.1",   # cpmCPUTotal5minRev
    "memory-usage": "1.3.6.1.4.1.9.9.48.1.1.1.6.1", # ciscoMemoryPoolFree
    "system-info": "1.3.6.1.2.1.1.1.0",
}

# ****************************************************************************
# FUNCIONES AUXILIARES: SNMP
# ****************************************************************************
# ----------------------------------------------------------------------------
# Consulta SNMP GET
# ----------------------------------------------------------------------------
def snmp_get(ip, community, oid):
    iterator = getCmd(
        SnmpEngine(),
        CommunityData(community, mpModel=0),
```

```
        UdpTransportTarget((ip, 161)),
        ContextData(),
        ObjectType(ObjectIdentity(oid))
    )

    errorIndication, errorStatus, errorIndex, varBinds = next(iterator)

    if errorIndication:
        print(errorIndication)
        return None
    elif errorStatus:
        print(f'{errorStatus.prettyPrint()} at {errorIndex}')
        return None
    else:
        for varBind in varBinds:
            print(varBind)
            return varBind.prettyPrint().split('=')[1].strip()

# ------------------------------------------------------------------------
# Consulta SNMP WALK
# ------------------------------------------------------------------------
def snmp_walk(ip, community, oid):
    iterator = nextCmd(
        SnmpEngine(),
        CommunityData(community),
        UdpTransportTarget((ip, 161)),
        ContextData(),
        ObjectType(ObjectIdentity(oid)),
        lexicographicMode=False
    )

    results = []
    for errorIndication, errorStatus, errorIndex, varBinds in iterator:
        if errorIndication:
            print(errorIndication)
            return None
        elif errorStatus:
            print(f'{errorStatus.prettyPrint()} at {errorIndex}')
            return None
        else:
            for varBind in varBinds:
                results.append((str(varBind[0]), str(varBind[1])))
    return results


# ************************************************************************
# FUNCIONES AUXILIARES: CONFIGURACIÓN
# ************************************************************************
# ------------------------------------------------------------------------
# Apagar/Encender interfaz
# ------------------------------------------------------------------------
def shutdown_interface(ip, interface, shutdown=False):
    net_connect = get_device_connection(ip)

    # Command depends on the no_shutdown parameter
    command = "no shutdown" if not shutdown else "shutdown"

    # List of commands to send
    command_list = [
        f"interface {interface}",
        command,
        "end",
    ]

    # Send commands to device
    output = net_connect.send_config_set(command_list)
    net_connect.save_config()

```

```
    # Disconnect from device
    net_connect.disconnect()
    return output

# ------------------------------------------------------------------------------
# Configurar IP en interfaz
# ------------------------------------------------------------------------------
def configure_interface_ip(ip, interface, ip_address, subnet_mask):
    net_connect = get_device_connection(ip)

    # List of commands to send
    command_list = [
        f"interface {interface}",
        f"ip address {ip_address} {subnet_mask}",
        "no shutdown",
    ]

    # Send commands to device
    output = net_connect.send_config_set(command_list)
    net_connect.save_config()

    # Disconnect from device
    net_connect.disconnect()
    return output

# ------------------------------------------------------------------------------
# Cambiar Router ID de OSPF
# ------------------------------------------------------------------------------
def change_ospf_router_id(ip, router_id):
    net_connect = get_device_connection(ip)

    # List of commands to send
    command_list = [
        "router ospf 1",
        f"router-id {router_id}",
        "end",
    ]

    # Send commands to device
    output = net_connect.send_config_set(command_list)

    # Clear OSPF process by sending command by command
    output += net_connect.send_command("clear ip ospf process",
        ↪ expect_string=r"Reset ALL OSPF processes\? \[no\]:")

    # Answer "yes" to the prompt
    output += net_connect.send_command("yes", expect_string=r"#")

    net_connect.save_config()

    # Disconnect from device
    net_connect.disconnect()
    return output

# ------------------------------------------------------------------------------
# Agregar Red OSPF
# ------------------------------------------------------------------------------
def add_ospf_network(ip, network_address, wildcard_mask, area, process_id):
    net_connect = get_device_connection(ip)

    if not process_id:
        process_id = 10

    # List of commands to send
    command_list = [
        f"router ospf {process_id}",
        f"network {network_address} {wildcard_mask} area {area}",
    ]
```

```
192
193     # Send commands to device
194     output = net_connect.send_config_set(command_list)
195     net_connect.save_config()
196
197     # Disconnect from device
198     net_connect.disconnect()
199     return output
200
201 # ------------------------------------------------------------------------------
202 # Eliminar Red OSPF
203 # ------------------------------------------------------------------------------
204 def remove_ospf_network(ip, network_address, wildcard_mask, area, process_id):
205     net_connect = get_device_connection(ip)
206
207     # List of commands to send
208     command_list = [
209         f"router ospf {process_id}",
210         f"no network {network_address} {wildcard_mask} area {area}",
211     ]
212
213     # Send commands to device
214     output = net_connect.send_config_set(command_list)
215     net_connect.save_config()
216
217     # Disconnect from device
218     net_connect.disconnect()
219     return output
220
221 # ******************************************************************************
222 # FUNCIONES AUXILIARES: MONITOREO
223 # ******************************************************************************
224 # ------------------------------------------------------------------------------
225 # Obtener estado de interfaces
226 # ------------------------------------------------------------------------------
227 def get_interface_status(ip):
228     net_connect = get_device_connection(ip)
229
230     # Send command to device
231     output = net_connect.send_command("show ip interface brief")
232
233     # Disconnect from device
234     net_connect.disconnect()
235     return output
236
237 # ------------------------------------------------------------------------------
238 # Obtener tabla de enrutamiento
239 # ------------------------------------------------------------------------------
240 def get_routing_table(ip):
241     net_connect = get_device_connection(ip)
242
243     # Send command to device
244     output = net_connect.send_command("show ip route")
245
246     # Regex pattern
247     pattern = r'^\s*([LCSRMBDEX0])[ ]+([\d.]+\/\d+).+?(\b\w+\d+\/\d+|\bLoopback\d+)$'
248
249     # Find all matches
250     routing_table = re.findall(pattern, output, re.MULTILINE)
251     routes = []
252
253
254     for match in routing_table:
255         # Change name of route type
256         if match[0] == "C":
257             route_type = "Conected"
258         elif match[0] == "L":
259             route_type = "Local"
```

```
260             elif match[0] == "S":
261                 route_type = "Static"
262             elif match[0] == "R":
263                 route_type = "RIP"
264             elif match[0] == "M":
265                 route_type = "Mobile"
266             elif match[0] == "B":
267                 route_type = "BGP"
268             elif match[0] == "D":
269                 route_type = "EIGRP"
270             elif match[0] == "E":
271                 route_type = "EGP"
272             elif match[0] == "EX":
273                 route_type = "EIGRP external"
274             elif match[0] == "O":
275                 route_type = "OSPF"
276             else:
277                 route_type = "Unknown"
278             ip_a = match[1]
279             interface = match[2]
280             routes.append((route_type, ip_a, interface))
281 
282     # Disconnect from device
283     net_connect.disconnect()
284     return routes
285 
286 # ------------------------------------------------------------------------------
287 # Obtener vecinos OSPF
288 # ------------------------------------------------------------------------------
289 def get_ospf_neighbors(ip):
290     net_connect = get_device_connection(ip)
291 
292     # Send command to device
293     output = net_connect.send_command("show ip ospf neighbor")
294 
295     # Disconnect from device
296     net_connect.disconnect()
297     return output
298 
299 # ------------------------------------------------------------------------------
300 # Obtener tiempo de actividad del dispositivo
301 # ------------------------------------------------------------------------------
302 def get_device_uptime(ip):
303     net_connect = get_device_connection(ip)
304 
305     # Send command to device
306     output = net_connect.send_command("show version | include uptime")
307 
308     # Disconnect from device
309     net_connect.disconnect()
310     return output
311 
312 # ------------------------------------------------------------------------------
313 # Ejecutar comando personalizado
314 # ------------------------------------------------------------------------------
315 def execute_custom_command(ip, command):
316     net_connect = get_device_connection(ip)
317 
318     # Send command to device
319     output = net_connect.send_command(command)
320 
321     # Disconnect from device
322     net_connect.disconnect()
323     return output
```

Cuadro 115: Código que contiene todas las funciones para conexiones a dispositivos de red

8.4.2. Opciones de configuración dentro de la API

Como se mencionó anteriormente, la API cuenta con diferentes *endpoints* que permiten realizar acciones de configuración y monitoreo en dispositivos de red. Algunas de las opciones de configuración disponibles incluyen:

- Asignación de IPs a interfaces de routers
- Apagado y encendido de interfaces
- Cambio del ID de router en OSPF
- Adición y eliminación de redes en un proceso OSPF

Todas estas opciones se realizan utilizando una conexión *Telnet* a los dispositivos de red, lo cual permite ejecutar comandos de configuración y obtener información de los mismos. Se evitó utilizar conexiones SSH debido a que, al ser un protocolo seguro con cifrado, los dispositivos antiguos simulados en *GNS3* presentaban problemas de intercambio de llaves. Además, el firewall de la universidad impedía las conexiones SSH hacia estos dispositivos en la red simulada de prueba.

En el código 118 se pueden ver todos los *endpoints* disponibles para configurar dispositivos. Pero, ya que queremos tener una forma visual de ver las opciones de configuración, se crearon diferentes plantillas HTML (en conjunto con *Bootstrap*) que permiten al usuario tener una interfaz gráfica para seleccionar la acción que desea realizar. En las siguientes figuras y códigos se presentan todas las plantillas HTML de configuración, junto con su visualización final en la API funcional.

```
<!-- templates/base.html -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Mi Aplicación{% endblock %}</title>
    <!-- Meta etiquetas para responsive design -->
    <meta name="viewport" content="width=device-width, initial-scale=1,
        ↪ shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap
            ↪ .min.css"
        rel="stylesheet"
        integrity="sha384-EVSTQN3/
            ↪ azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
        crossorigin="anonymous"
    >

    <!-- Tu archivo CSS personalizado -->
    <link rel="stylesheet" href="/static/css/styles.css">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
        ↪ icons@1.10.5/font/bootstrap-icons.css">
```

```
</head>
<body>
    <!-- Barra de navegación -->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
          <a class="navbar-brand" href="/">Administración de Dispositivos</a>
          <button class="navbar-toggler" type="button" data-bs-toggle="
              ↪ collapse" data-bs-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false" aria-label="
                ↪ Alternar navegación">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ms-auto">
              <li class="nav-item">
                <a class="nav-link active" aria-current="page" href="/">
                    ↪ Inicio</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/config">Configuración</a>
              </li>
              <li class="nav-item">
                <a class="nav-link" href="/monitor">Monitoreo</a>
              </li>
            </ul>
          </div>
        </div>
    </nav>

    <!-- Contenido principal -->
    <main class="container my-4">
        {% block content %}
        {% endblock %}
    </main>

    <!-- Pie de página -->
    <footer class="bg-dark text-white text-center py-3">
    <div class="container">
        <p class="mb-0">&copy; 2024 José Santizo Olivet</p>
    </div>
    </footer>

    <!-- Bootstrap JS -->
    <script
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.
            ↪ bundle.min.js"
        integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+
            ↪ JcXn/tWtIaxVXM"
        crossorigin="anonymous">
    </script>

    <!-- Tu archivo JS personalizado -->
    <script src="/static/js/scripts.js"></script>
</body>
</html>
```

Cuadro 116: Plantilla HTML para página principal

```
<!-- templates/index.html -->
{% extends "base.html" %}

{% block title %}Inicio{% endblock %}

{% block content %}
<div class="text-center">
    <h2 class="mt-4">Bienvenido a la Administración de Dispositivos de Red</
        ↪ h2>
    <!-- Imagen o icono opcional -->
    <!-- <img src="/static/images/network.png" alt="Red de Dispositivos"
        ↪ class="img-fluid my-4"> -->
    <p class="lead mt-3">
        Esta plataforma, desplegada en un clúster de Kubernetes, permite el
            ↪ monitoreo y la configuración eficientes de dispositivos de red
            ↪ . Utilice el menú de navegación para acceder a las opciones de
            ↪  monitoreo y configuración disponibles.
    </p>
    <div class="mt-4">
        <a href="/config" class="btn btn-primary btn-lg me-2">
            <i class="bi bi-gear-fill me-1"></i>Configuración
        </a>
        <a href="/monitor" class="btn btn-secondary btn-lg">
            <i class="bi bi-speedometer2 me-1"></i>Monitoreo
        </a>
    </div>
</div>
{% endblock %}
```

Cuadro 117: Contenido que se añade a la plantilla *"base.html"* para mostrarse en la API

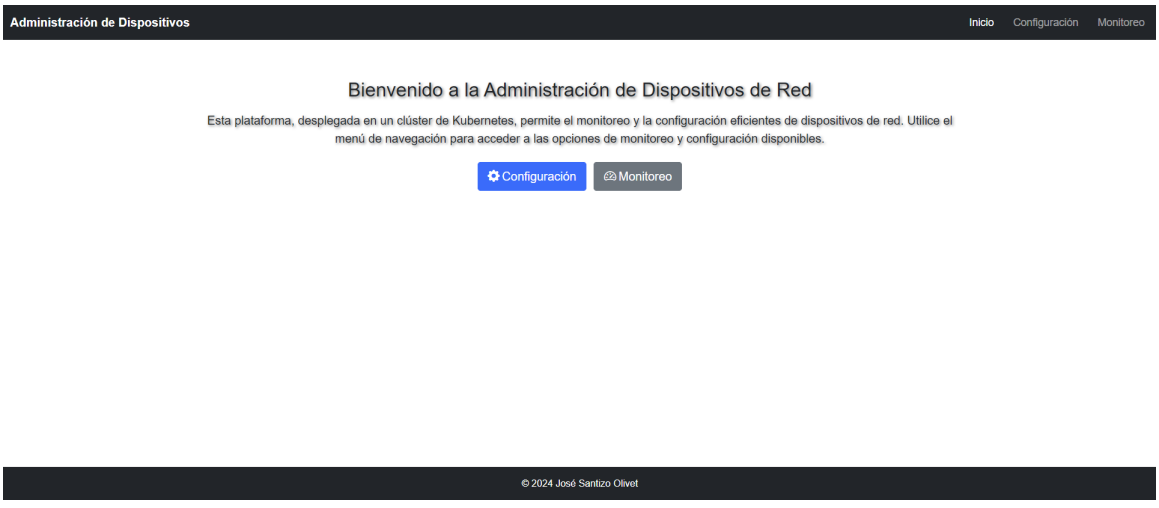


Figura 159: Página principal de la API (visualización gráfica del código 116 y 117)

```
<!-- templates/config/config.html -->
{% extends "base.html" %}

{% block title %}Configuración{% endblock %}
```

```
5
6  {% block content %}
7  <div class="container">
8      <h2 class="mt-4">Opciones de Configuración</h2>
9      <p class="lead">Seleccione una de las siguientes opciones para configurar
       ↪  su dispositivo de red:</p>
10     <div class="list-group mt-4">
11         <a href="/config/interface" class="list-group-item list-group-item-
           ↪ action">
12             <i class="bi bi-toggle-off me-2"></i> Apagar/Encender Interfaz
13         </a>
14         <a href="/config/interface/ip" class="list-group-item list-group-item
           ↪ -action">
15             <i class="bi bi-ethernet me-2"></i> Configurar IP en Interfaz
16         </a>
17         <a href="/config/ospf/router_id" class="list-group-item list-group-
           ↪ item-action">
18             <i class="bi bi-router-fill me-2"></i> Cambiar Router ID de OSPF
19         </a>
20         <a href="/config/ospf/network" class="list-group-item list-group-item
           ↪ -action">
21             <i class="bi bi-plus-circle me-2"></i> Agregar Red OSPF
22         </a>
23         <a href="/config/ospf/network/remove" class="list-group-item list-
           ↪ group-item-action">
24             <i class="bi bi-dash-circle me-2"></i> Eliminar Red OSPF
25         </a>
26     </div>
27 </div>
28 {% endblock %}
```

Cuadro 118: Plantilla HTML para la página de configuración

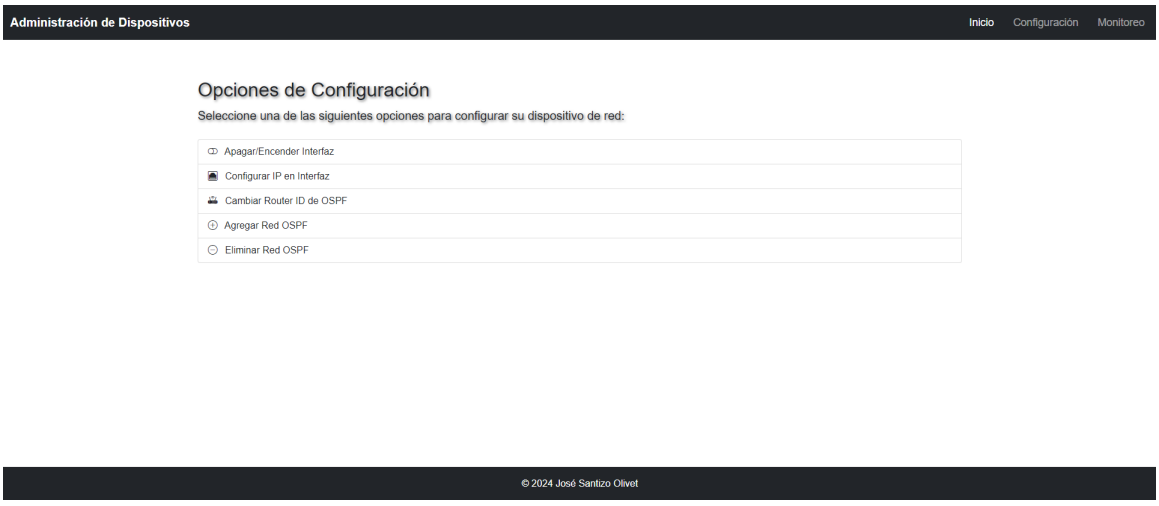


Figura 160: Página principal con opciones de configuración (visualización gráfica del código 118)

```
1  <!-- templates/config/interface.html -->
2  {% extends "base.html" %}
3
```

```
{% block title %}Apagar/Encender Interfaz{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Apagar/Encender Interfaz</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</label>
            <input type="text" class="form-control" id="ip" name="ip" placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="interface" class="form-label">Interfaz</label>
            <input type="text" class="form-control" id="interface" name="interface" placeholder="Ejemplo: ethernet 0/1" required>
        </div>
        <div class="mb-3">
            <label for="action" class="form-label">Acción</label>
            <select class="form-select" id="action" name="action">
                <option value="shutdown">Apagar</option>
                <option value="no_shutdown">Encender</option>
            </select>
        </div>
        <button type="submit" class="btn btn-primary">Enviar</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 119: Plantilla HTML para la página que puede encender/apagar una interfaz

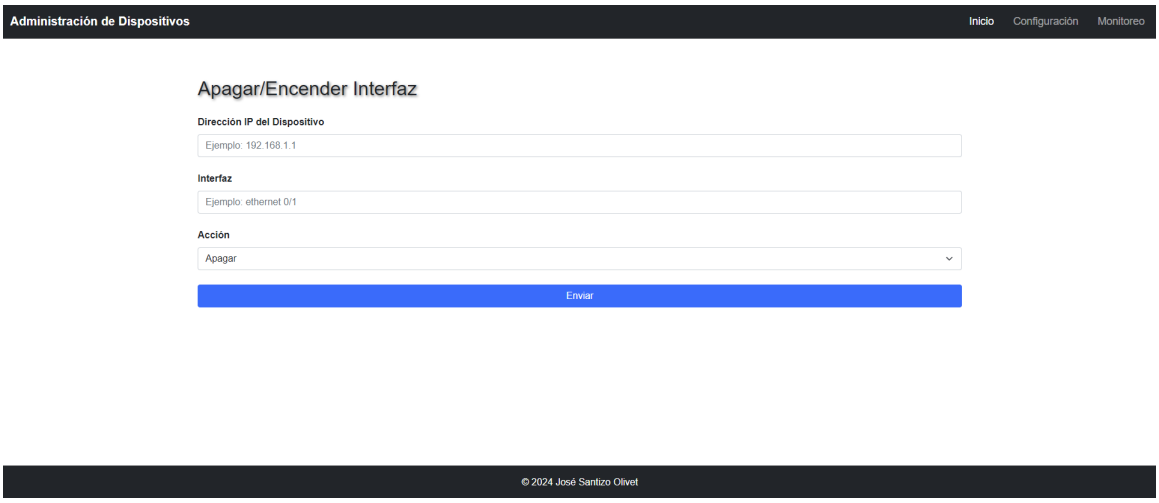


Figura 161: Página para encender o apagar una interfaz (visualización gráfica del código 125)

```
<!-- templates/config/interface_ip_config.html -->
{% extends "base.html" %}

{% block title %}Configurar IP en Interfaz{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Configurar IP en Interfaz</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="interface" class="form-label">Interfaz</label>
            <input type="text" class="form-control" id="interface" name="
                ↪ interface" placeholder="Ejemplo: GigabitEthernet0/1"
                ↪ required>
        </div>
        <div class="mb-3">
            <label for="ip_address" class="form-label">Dirección IP a
                ↪ Configurar</label>
            <input type="text" class="form-control" id="ip_address" name="
                ↪ ip_address" placeholder="Ejemplo: 10.0.0.1" required>
        </div>
        <div class="mb-3">
            <label for="subnet_mask" class="form-label">Máscara de Subred</
                ↪ label>
            <input type="text" class="form-control" id="subnet_mask" name="
                ↪ subnet_mask" placeholder="Ejemplo: 255.255.255.0" required
                ↪ >
        </div>
        <button type="submit" class="btn btn-primary">Configurar</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 120: Plantilla HTML para la página que puede configurar una IP en una interfaz de red

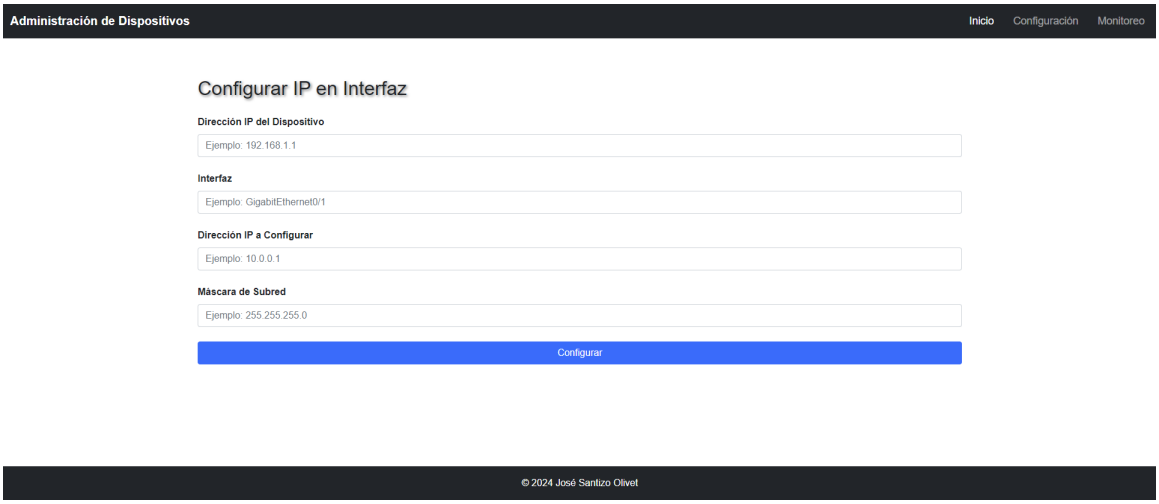


Figura 162: Página para configurar una IP en una interfaz de red (visualización gráfica del código 120)

```html
<!-- templates/config/ospf_router_id.html -->
{% extends "base.html" %}

{% block title %}Cambiar Router ID de OSPF{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Cambiar Router ID de un router en la red OSPF</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</label>
            <input type="text" class="form-control" id="ip" name="ip" placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="router_id" class="form-label">Nuevo Router ID</label>
            <input type="text" class="form-control" id="router_id" name="router_id" placeholder="Ejemplo: 1.1.1.1" required>
        </div>
        <button type="submit" class="btn btn-primary">Cambiar Router ID</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 121: Plantilla HTML para la página que puede configurar un nuevo router ID en un proceso OSPF

Figura 163: Página para configurar un nuevo router ID en un proceso OSPF (visualización gráfica del código 121)

```
<!-- templates/config/ospf_network.html -->
{% extends "base.html" %}

{% block title %}Agregar Red OSPF{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Agregar Red al Proceso OSPF</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="network" class="form-label">Red</label>
            <input type="text" class="form-control" id="network" name="
                ↪ network" placeholder="Ejemplo: 10.0.0.0" required>
        </div>
        <div class="mb-3">
            <label for="wildcard_mask" class="form-label">Wildcard Mask</
                ↪ label>
            <input type="text" class="form-control" id="wildcard_mask" name="
                ↪ wildcard_mask" placeholder="Ejemplo: 0.0.0.255" required>
        </div>
        <div class="mb-3">
            <label for="wildcard_mask" class="form-label">Process ID</label>
            <input type="text" class="form-control" id="process_id" name="
                ↪ process_id" placeholder="Ejemplo: 10" required>
        </div>
        <div class="mb-3">
            <label for="area" class="form-label">Área</label>
            <input type="text" class="form-control" id="area" name="area"
                ↪ placeholder="Ejemplo: 0" required>
        </div>
        <button type="submit" class="btn btn-primary">Agregar Red</button>
```

```
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 122: Plantilla HTML para la página que puede añadir una red a un proceso OSPF

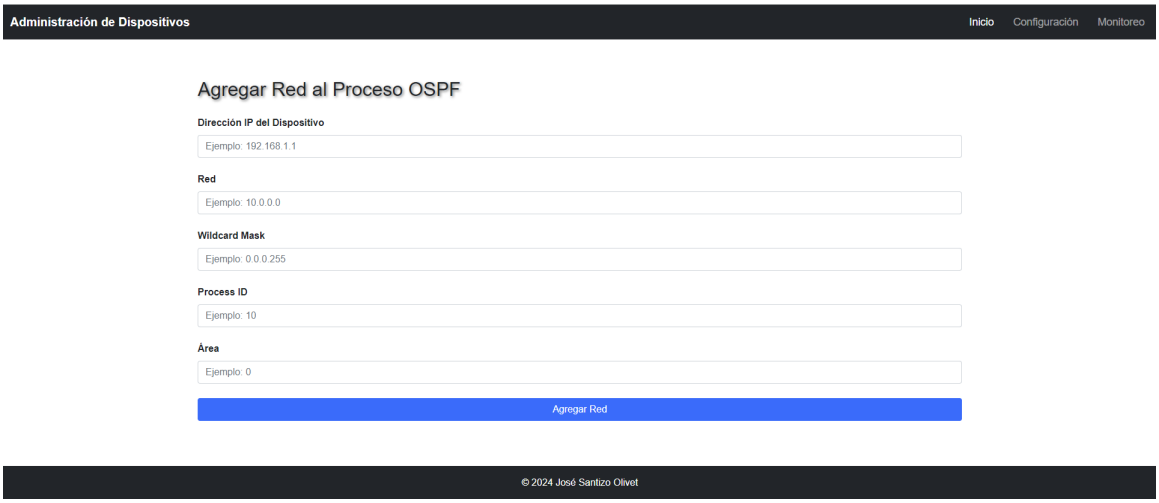


Figura 164: Página para añadir una red a un proceso OSPF (visualización gráfica del código 122)

```
<!-- templates/config/ospf_network_remove.html -->
{% extends "base.html" %}

{% block title %}Eliminar Red OSPF{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Eliminar Red del Proceso OSPF</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="network" class="form-label">Red</label>
            <input type="text" class="form-control" id="network" name="
                ↪ network" placeholder="Ejemplo: 10.0.0.0" required>
        </div>
        <div class="mb-3">
            <label for="wildcard_mask" class="form-label">Wildcard Mask</
                ↪ label>
```

```
            <input type="text" class="form-control" id="wildcard_mask" name="
                ↪ wildcard_mask" placeholder="Ejemplo: 0.0.0.255" required>
        </div>
        <div class="mb-3">
            <label for="wildcard_mask" class="form-label">Process ID</label>
            <input type="text" class="form-control" id="process_id" name="
                ↪ process_id" placeholder="Ejemplo: 10" required>
        </div>
        <div class="mb-3">
            <label for="area" class="form-label">Área</label>
            <input type="text" class="form-control" id="area" name="area"
                ↪ placeholder="Ejemplo: 0" required>
        </div>
        <button type="submit" class="btn btn-danger">Eliminar Red</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 123: Plantilla HTML para la página que puede eliminar una red de un proceso OSPF

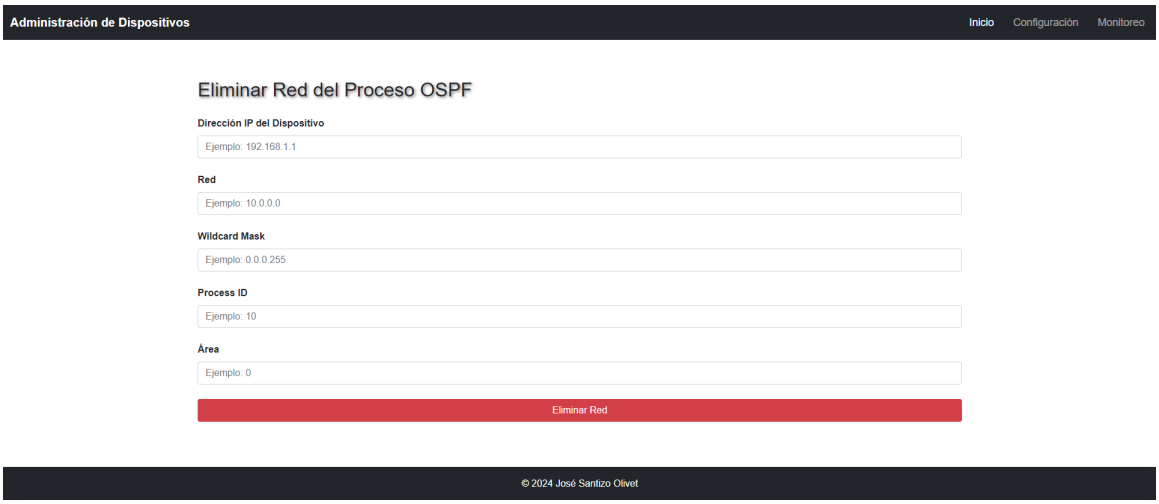


Figura 165: Página para eliminar una red de un proceso OSPF (visualización gráfica del código 123)

8.4.3. Opciones de monitoreo dentro de la API

Además de las opciones de configuración, la API también cuenta con diferentes *end-points* para realizar tareas de monitoreo en dispositivos de red. Algunas de las opciones de monitoreo disponibles incluyen:

- Visualización de la tabla de enrutamiento de un router
- Estado de las interfaces
- Vecinos OSPF
- Tiempo de actividad del dispositivo
- Ejecutar un comando personalizado en el dispositivo
- Datos extraídos a través de SNMP como:
  - Uso de CPU
  - Paquetes de entrada y salida con error en interfaces
  - Uso de memoria
  - Información del sistema

Todas estas opciones se realizan al utilizar un conexión *Telnet* a los dispositivos de red, lo cual permite ejecutar comandos de configuración y obtener información de los mismos. En el código 114 se pueden ver todos los *endpoints* disponibles para monitorear dispositivos. Pero, ya que queremos tener una forma visual de ver las opciones de monitoreo, se crearon diferentes plantillas HTML (En conjunto con *Bootstrap*) que permiten al usuario tener una interfaz gráfica para seleccionar la acción que desea realizar. En las siguientes figuras y códigos, se presentan todas las plantillas HTML de monitoreo, junto con su visualización final en la API funcional.

```
<!-- templates/monitor/monitor.html -->
{% extends "base.html" %}

{% block title %}Monitoreo{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Opciones de Monitoreo</h2>
    <p class="lead">Seleccione una de las siguientes opciones para monitorear
        ↪ su dispositivo de red:</p>
    <div class="list-group mt-4">
        <a href="/monitor/interfaces" class="list-group-item list-group-item-
            ↪ action">
            <i class="bi bi-hdd-network me-2"></i> Estado de las Interfaces
        </a>
        <a href="/monitor/snmp/snmp" class="list-group-item list-group-item-
            ↪ action">
            <i class="bi bi-speedometer me-2"></i> Datos del dispositivo
        </a>
        <a href="/monitor/routing-table" class="list-group-item list-group-
            ↪ item-action">
            <i class="bi bi-diagram-3 me-2"></i> Tabla de Enrutamiento
        </a>
        <a href="/monitor/ospf-neighbors" class="list-group-item list-group-
            ↪ item-action">
            <i class="bi bi-people-fill me-2"></i> Vecinos OSPF
        </a>
```

```
        <a href="/monitor/uptime" class="list-group-item list-group-item-
            ↪ action">
            <i class="bi bi-clock-fill me-2"></i> Tiempo de Actividad del
                ↪ Dispositivo
        </a>
        <a href="/monitor/custom-command" class="list-group-item list-group-
            ↪ item-action">
            <i class="bi bi-terminal-fill me-2"></i> Ejecutar Comando
                ↪ Personalizado
        </a>
    </div>
</div>
{% endblock %}
```

Cuadro 124: Plantilla HTML para la página principal de monitoreo

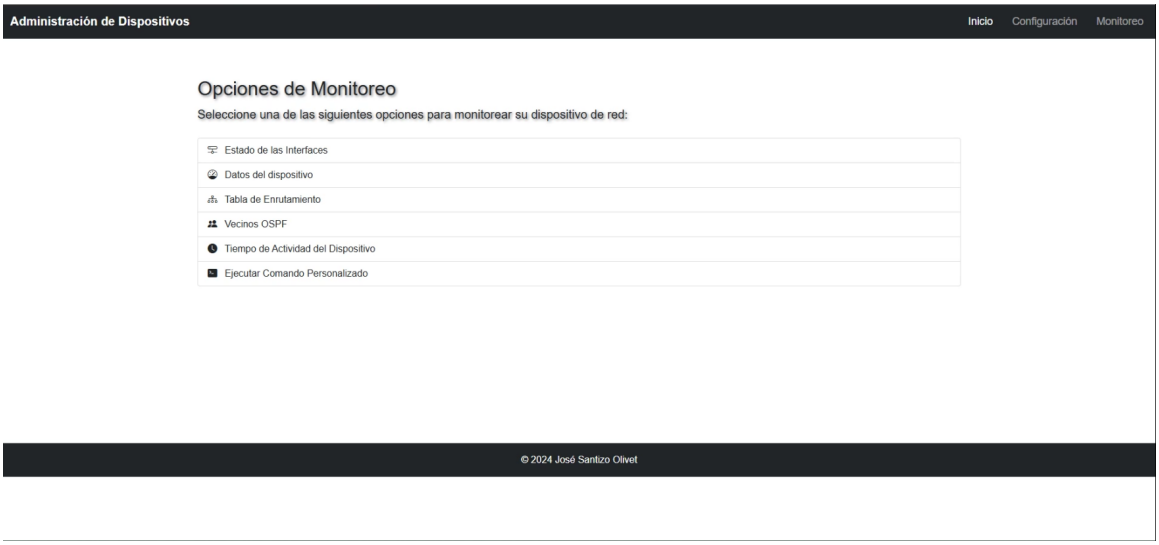


Figura 166: Página principal con todas las opciones de monitoreo (visualización gráfica del código 124)

```
<!-- templates/monitor/snmp/interfaces_status.html -->
{% extends "base.html" %}

{% block title %}Estado de Interfaces{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Estado de Interfaces del Dispositivo</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="community" class="form-label">Comunidad SNMP</label>
```

```
            <input type="text" class="form-control" id="community" name="
                ↪ community" value="public">
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if error %}
    <div class="alert alert-danger mt-4" role="alert">
        {{ error }}
    </div>
    {% endif %}

    {% if output %}
    <div class="mt-4">
        <h4>Estado de Interfaces para {{ ip }}</h4>
        <table class="table table-striped mt-3">
            <thead>
                <tr>
                    <th>Interfaz</th>
                    <th>Estado</th>
                </tr>
            </thead>
            <tbody>
                {% for item in output %}
                <tr>
                    <td>{{ item.interface }}</td>
                    <td>
                        {% if item.status == "up" %}
                            <span class="badge bg-success">Up</span>
                        {% elif item.status == "down" %}
                            <span class="badge bg-danger">Down</span>
                        {% else %}
                            <span class="badge bg-secondary">{{ item.status
                                ↪ }}</span>
                        {% endif %}
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 125: Plantilla HTML para la página de estado de las interfaces

Figura 167: Página para visualizar el estado de las interfaces de un dispositivo (visualización gráfica del código 125)

```
<!-- templates/monitor/routing_table.html -->
{% extends "base.html" %}

{% block title %}Tabla de Enrutamiento{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Tabla de Enrutamiento</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if error %}
    <div class="alert alert-danger mt-4" role="alert">
        {{ error }}
    </div>
    {% endif %}

    {% if routing_table %}
    <div class="mt-4">
        <h4>Tabla de Enrutamiento para {{ ip }}</h4>
        <table class="table table-striped mt-3">
            <thead>
                <tr>
                    <th>Tipo de Ruta</th>
                    <th>IP/Prefijo</th>
                    <th>Interfaz</th>
                </tr>
            </thead>
            <tbody>
                {% for route in routing_table %}
                <tr>
```

```
                    <td>{{ route.route_type }}</td>
                    <td>{{ route.ip }}</td>
                    <td>{{ route.interface }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 126: Plantilla HTML para la página de visualización de la tabla de enrutamiento

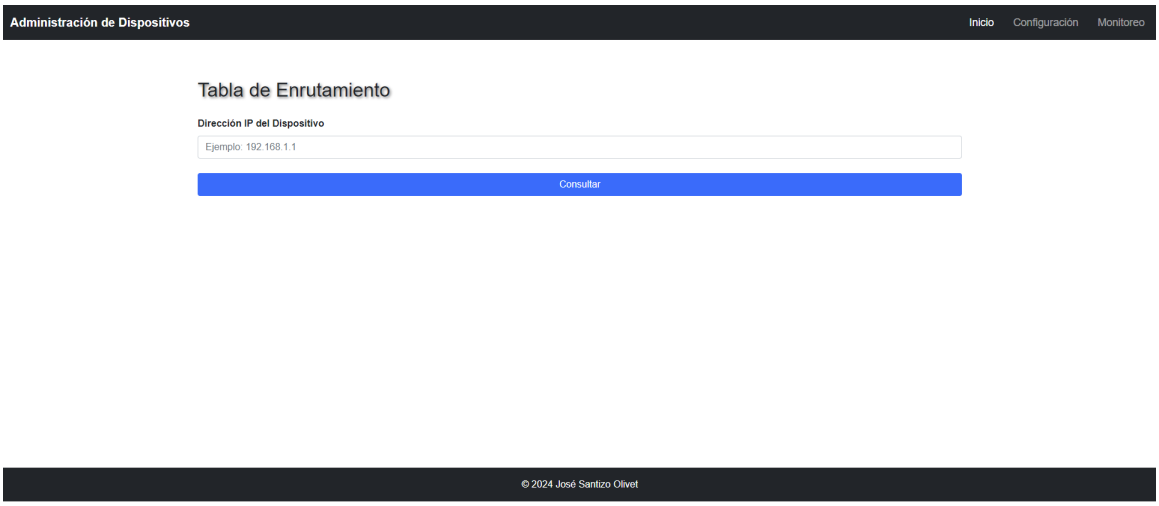


Figura 168: Página para visualizar la tabla de enrutamiento de un dispositivo (visualización gráfica del código 126)

```
<!-- templates/monitor/ospf_neighbors.html -->
{% extends "base.html" %}

{% block title %}Vecinos OSPF{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Vecinos OSPF</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
```

```
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 127: Plantilla HTML para la página de visualización de los vecinos OSPF

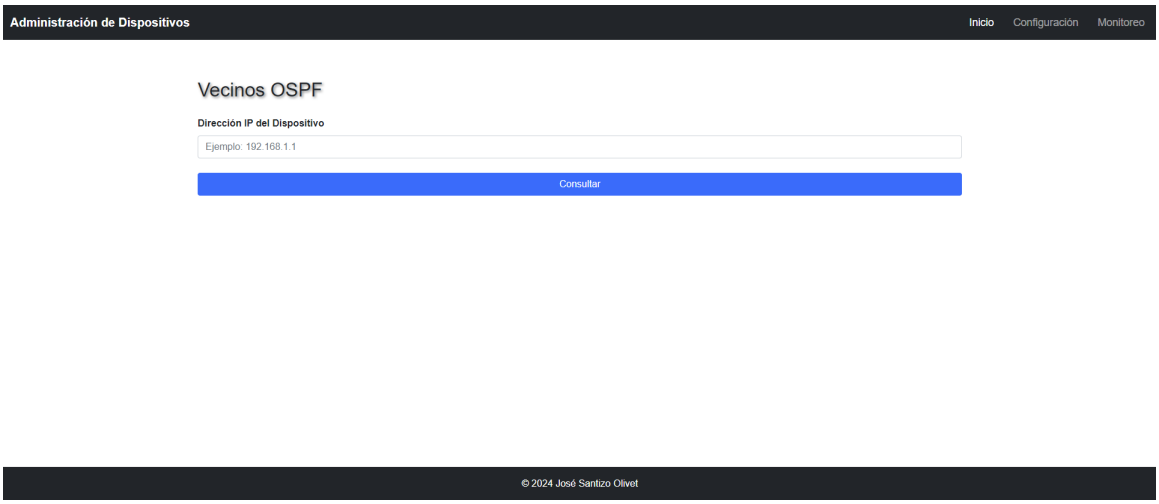


Figura 169: Página para visualizar los vecinos OSPF de un dispositivo (visualización gráfica del código 127)

```
<!-- templates/monitor/uptime.html -->
{% extends "base.html" %}

{% block title %}Tiempo de Actividad del Dispositivo{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Tiempo de Actividad del Dispositivo</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado:</h3>
```

```
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 128: Plantilla HTML para la página de visualización del tiempo de actividad del dispositivo

Figura 170: Página para visualizar el tiempo de actividad de un dispositivo (visualización gráfica del código 128)

```
<!-- templates/monitor/execute_command.html -->
{% extends "base.html" %}

{% block title %}Ejecutar Comando Personalizado{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Ejecutar Comando Personalizado</h2>
    <form method="post" action="/monitor/custom-command" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="command" class="form-label">Comando a Ejecutar</label
                ↪ >
            <input type="text" class="form-control" id="command" name="
                ↪ command" placeholder="Ejemplo: show version" required>
        </div>
        <button type="submit" class="btn btn-primary">Ejecutar</button>
    </form>

    {% if output %}
    <div class="mt-4">
        <h3>Resultado del Comando:</h3>
```

```
        <pre>{{ output }}</pre>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 129: Plantilla HTML para la página de ejecución de un comando personalizado

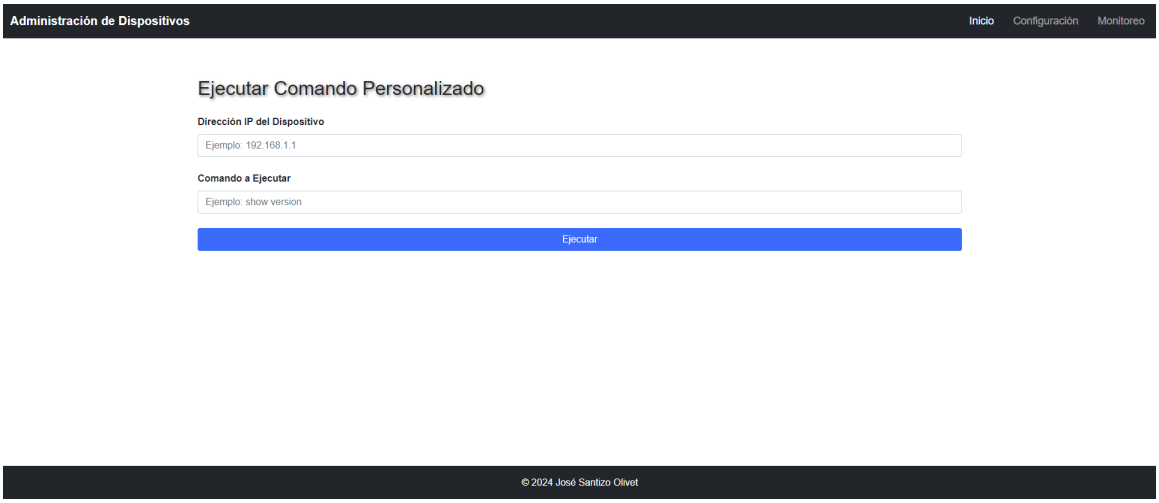


Figura 171: Página para ejecutar un comando personalizado en un dispositivo (visualización gráfica del código 129)

Como se mencionó anteriormente, también se añadieron opciones para obtener datos a través de SNMP. Las siguientes figuras y códigos muestran las plantillas HTML y la visualización final de las mismas en la API.

```
<!-- templates/monitor/snmp/snmp.html -->
{% extends "base.html" %}

{% block title %}Monitoreo SNMP{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Monitoreo SNMP</h2>
    <p class="lead">Seleccione una de las siguientes opciones para monitorear
        ↪ su dispositivo de red utilizando SNMP:</p>
    <div class="list-group mt-4">
        <a href="/monitor/snmp/cpu-usage" class="list-group-item list-group-
            ↪ item-action">
            <i class="bi bi-speedometer2 me-2"></i> Uso de CPU
        </a>
        <a href="/monitor/snmp/memory-usage" class="list-group-item list-
            ↪ group-item-action">
            <i class="bi bi-hdd me-2"></i> Uso de Memoria
        </a>
        <a href="/monitor/snmp/system-info" class="list-group-item list-group
            ↪ -item-action">
```

```
            <i class="bi bi-info-circle me-2"></i> Información del Sistema
        </a>
        <a href="/monitor/snmp/error-packets" class="list-group-item list-
            ↪ group-item-action">
            <i class="bi bi-exclamation-triangle me-2"></i> Paquetes con
                ↪ Errores
        </a>
    </div>
</div>
{% endblock %}
```

Cuadro 130: Plantilla HTML para la página principal de monitoreo a través de SNMP

Figura 172: Página principal con todas las opciones de monitoreo a través de SNMP (visualización gráfica del código 130)

```
<!-- templates/monitor/snmp/cpu_usage.html -->
{% extends "base.html" %}

{% block title %}Uso de CPU{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Uso de CPU</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="community" class="form-label">Comunidad SNMP</label>
            <input type="text" class="form-control" id="community" name="
                ↪ community" value="public">
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>
```

```
20
21      {% if error %}
22      <div class="alert alert-danger mt-4" role="alert">
23          {{ error }}
24      </div>
25      {% endif %}
26
27      {% if output %}
28      <div class="alert alert-success mt-4" role="alert">
29          <h4 class="alert-heading">Resultado para {{ ip }}</h4>
30          <p>Uso de CPU: <strong>{{ output }}%</strong></p>
31      </div>
32      {% endif %}
33  </div>
34  {% endblock %}
```

Cuadro 131: Plantilla HTML para la página de obtención del uso de CPU en el dispositivo

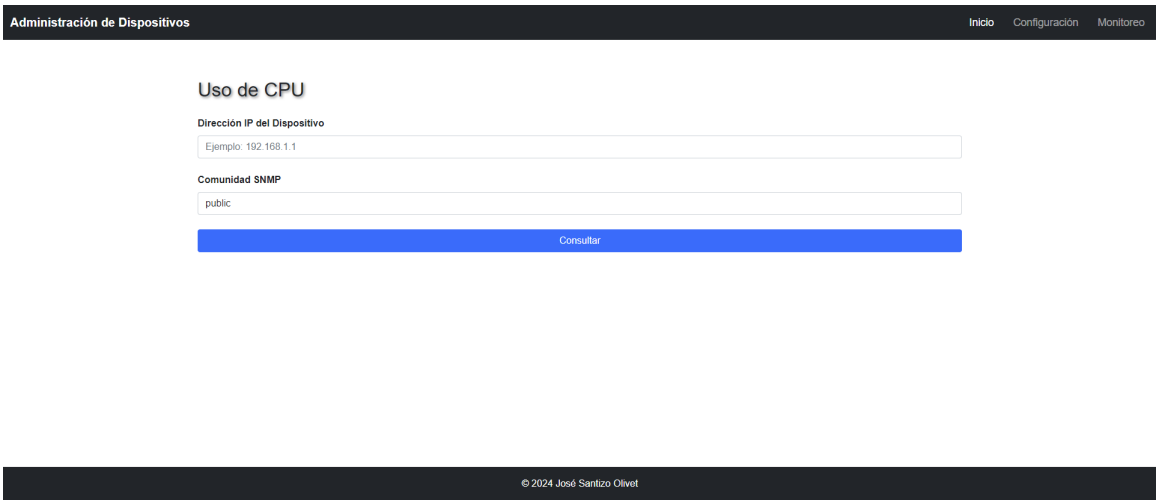


Figura 173: Página para obtener el uso de CPU en un dispositivo a través de SNMP (visualización gráfica del código 131)

```
1   <!-- templates/monitor/snmp/error_packets.html -->
2   {% extends "base.html" %}
3
4   {% block title %}Paquetes con Errores{% endblock %}
5
6   {% block content %}
7   <div class="container">
8       <h2 class="mt-4">Paquetes con Errores por Interfaz</h2>
9       <form method="post" class="mt-4">
10          <div class="mb-3">
11              <label for="ip" class="form-label">Dirección IP del Dispositivo</
                    ↪ label>
12              <input type="text" class="form-control" id="ip" name="ip"
                    ↪ placeholder="Ejemplo: 192.168.1.1" required>
13          </div>
14          <div class="mb-3">
15              <label for="community" class="form-label">Comunidad SNMP</label>
```

```
            <input type="text" class="form-control" id="community" name="
                ↪ community" value="public">
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if error %}
    <div class="alert alert-danger mt-4" role="alert">
        {{ error }}
    </div>
    {% endif %}

    {% if output %}
    <div class="mt-4">
        <h4>Paquetes con errores para {{ ip }}</h4>
        <table class="table table-striped mt-3">
            <thead>
                <tr>
                    <th>Interfaz</th>
                    <th>Errores de Entrada</th>
                    <th>Errores de Salida</th>
                </tr>
            </thead>
            <tbody>
                {% for item in output %}
                <tr>
                    <td>{{ item.interface }}</td>
                    <td>{{ item.in_errors }}</td>
                    <td>{{ item.out_errors }}</td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 132: Plantilla HTML para la página de obtención de paquetes de entrada/salida en una interfaz y que contienen errores

Figura 174: Página para obtener paquetes de entrada/salida en una interfaz y que contienen errores a través de SNMP (visualización gráfica del código 132)

```
<!-- templates/monitor/snmp/memory_usage.html -->
{% extends "base.html" %}

{% block title %}Uso de Memoria{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Uso de Memoria</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="community" class="form-label">Comunidad SNMP</label>
            <input type="text" class="form-control" id="community" name="
                ↪ community" value="public">
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if error %}
    <div class="alert alert-danger mt-4" role="alert">
        {{ error }}
    </div>
    {% endif %}

    {% if output %}
    <div class="alert alert-success mt-4" role="alert">
        <h4 class="alert-heading">Resultado para {{ ip }}</h4>
        <p>Uso de Memoria: <strong>{{ output }} bytes de memoria del
            ↪ procesador libre</strong></p>
```

```
    </div>
    {% endif %}
</div>
{% endblock %}
```

Cuadro 133: Plantilla HTML para la página que obtiene la cantidad de memoria del procesador utilizada por el dispositivo

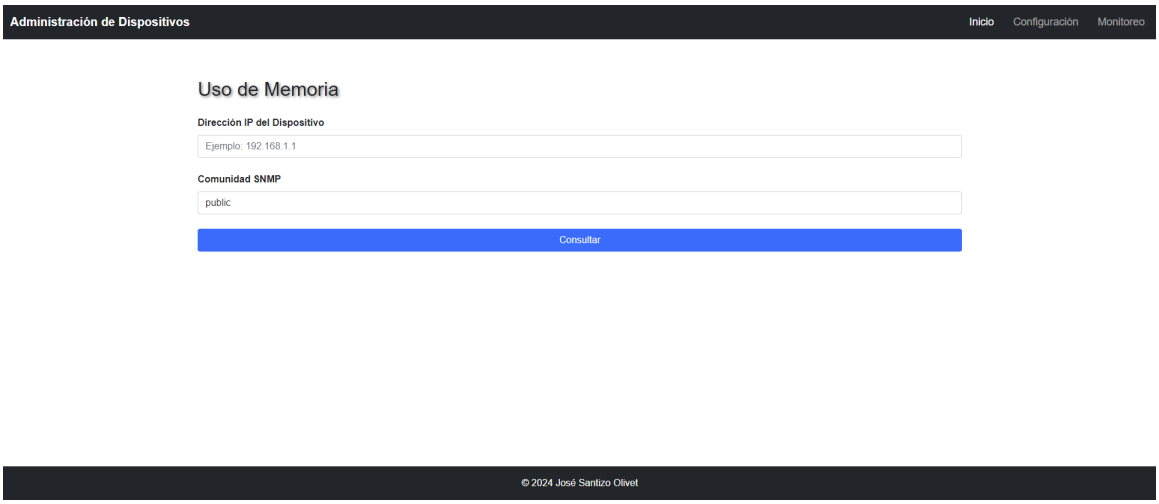


Figura 175: Página para obtener la cantidad de memoria del procesador utilizada por el dispositivo a través de SNMP (visualización gráfica del código 133)

```
<!-- templates/monitor/snmp/system_info.html -->
{% extends "base.html" %}

{% block title %}Información del Sistema{% endblock %}

{% block content %}
<div class="container">
    <h2 class="mt-4">Información del Sistema</h2>
    <form method="post" class="mt-4">
        <div class="mb-3">
            <label for="ip" class="form-label">Dirección IP del Dispositivo</
                ↪ label>
            <input type="text" class="form-control" id="ip" name="ip"
                ↪ placeholder="Ejemplo: 192.168.1.1" required>
        </div>
        <div class="mb-3">
            <label for="community" class="form-label">Comunidad SNMP</label>
            <input type="text" class="form-control" id="community" name="
                ↪ community" value="public">
        </div>
        <button type="submit" class="btn btn-primary">Consultar</button>
    </form>

    {% if error %}
    <div class="alert alert-danger mt-4" role="alert">
        {{ error }}
    </div>
```

```
25      {% endif %}
26
27      {% if output %}
28      <div class="alert alert-success mt-4" role="alert">
29          <h4 class="alert-heading">Resultado para {{ ip }}</h4>
30          <p>{{ output }}</p>
31      </div>
32      {% endif %}
33  </div>
34  {% endblock %}
```

Cuadro 134: Plantilla HTML para la página que obtiene información del sistema

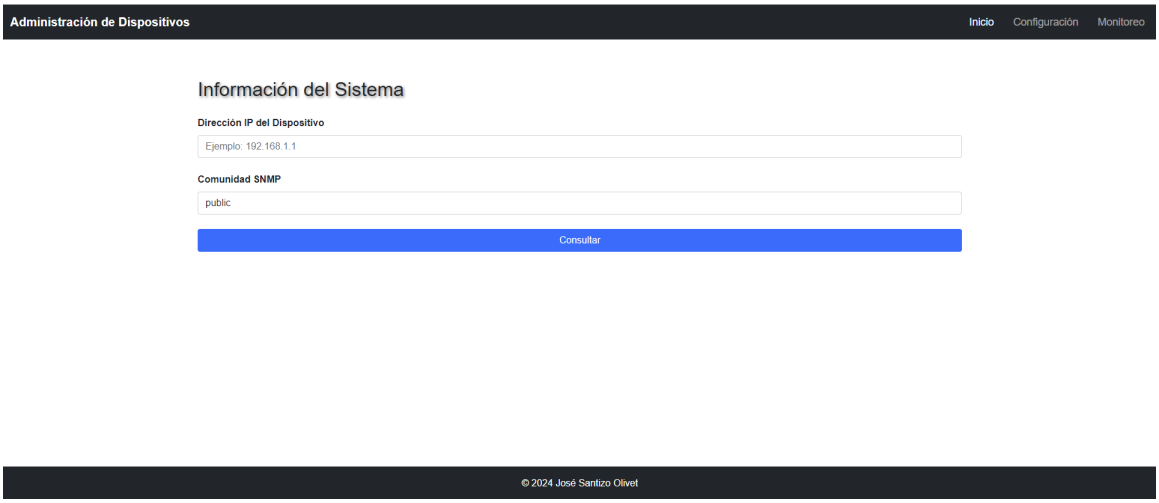


Figura 176: Página para obtener información del sistema a través de SNMP (visualización gráfica del código 134)

8.4.4. Despliegue y funcionamiento correcto de la API

Para desplegar la API, se utilizó un contenedor de *Docker* que contenía todos los archivos necesarios para ejecutar la API. El contenedor se construyó a partir de un archivo *Dockerfile*, el cual se subió a un repositorio en *Docker Hub* para su posterior descarga y ejecución en el clúster de *Kubernetes*. Toda la descarga y ejecución en el clúster, se realizó a través de Helm, el cual permitió generar un Helm Chart que contenía toda la información necesaria para desplegar la API en el clúster, tal como se puede visualizar en la Figura 177.

```
onap@onapnodo1:~/network-automation-app$ helm install network-app ./
NAME: network-app
LAST DEPLOYED: Thu Nov  7 02:48:18 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services network-app-network-
automation-app)
  export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
```

Figura 177: API desplegada en el clúster de *Kubernetes*

Una vez se desplegó en el clúster, se pudo acceder a la API a través de un navegador web, utilizando la dirección IP del servicio que contenia a la misma. La API tuvo un comportamiento correcto, permitiendo realizar diferentes acciones de configuración y monitoreo en dispositivos de red, tal como se puede visualizar de la figura 178 a la 181, las cuales muestran los logs obtenidos a través del clúster de *kubernetes*.

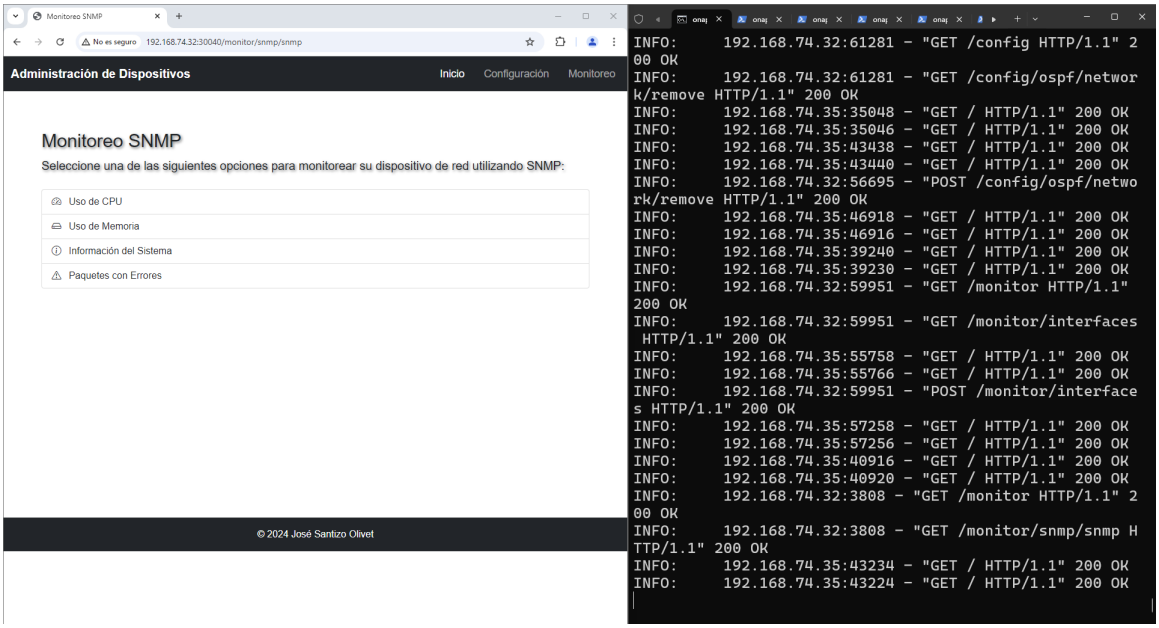


Figura 178: Prueba 1 de la API: Visualización del menú de monitoreo a través de SNMP

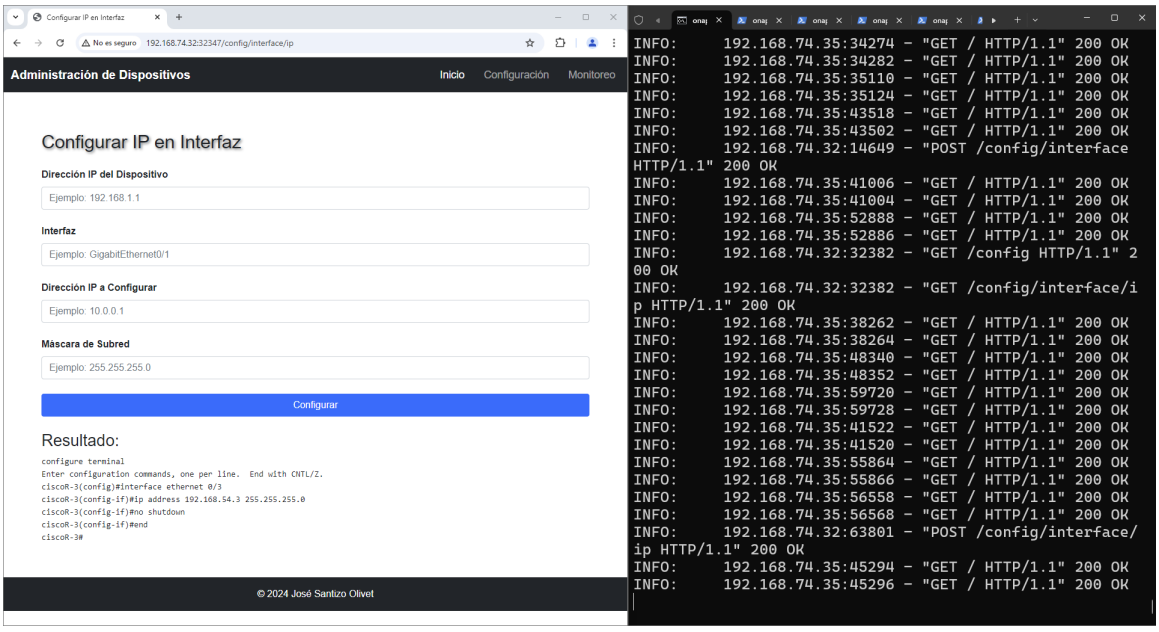


Figura 179: Prueba 2 de la API: Configuración de una IP en una interfaz de red

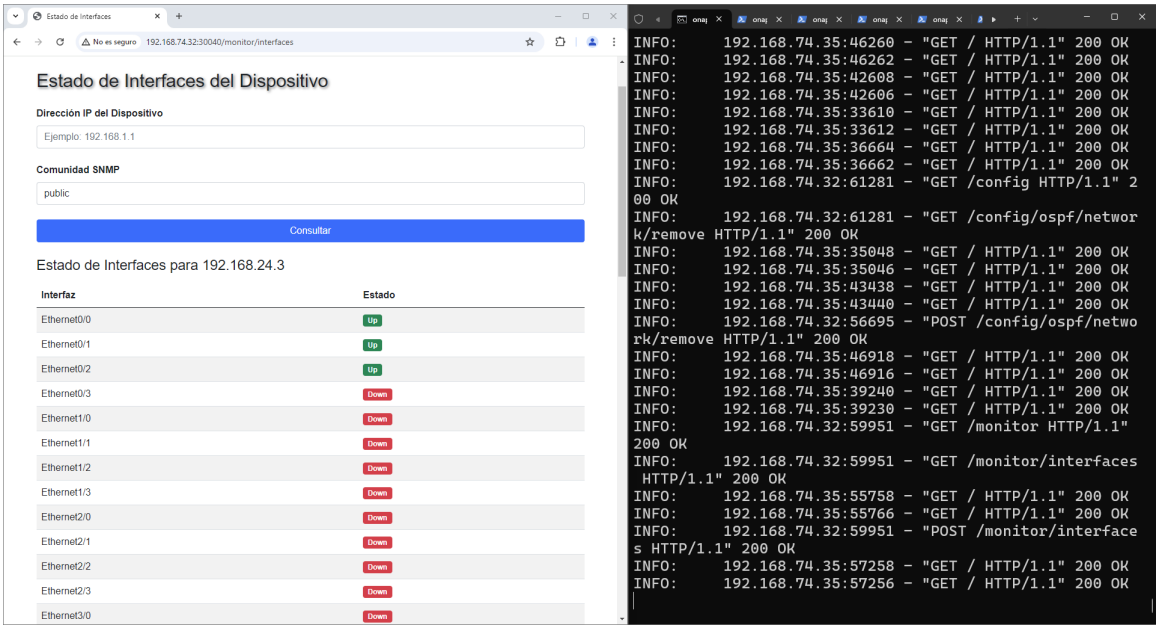


Figura 180: Prueba 3 de la API: Visualización del estado de las interfaces de un dispositivo

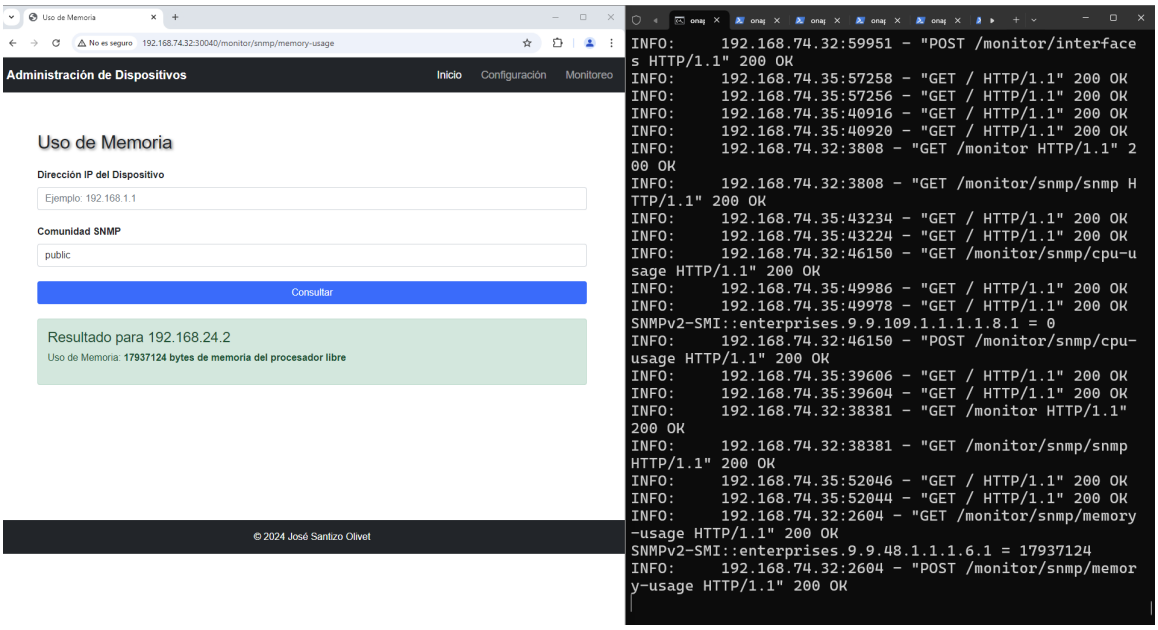


Figura 181: Prueba 4 de la API: Visualización de la cantidad de memoria del procesador utilizada por un dispositivo

A través de esto, pudimos comprobar que nuestra aplicación fue desplegada correctamente en el clúster de *Kubernetes*.

8.4.5. Pruebas de la API

Una forma de lograr probar la API, en un entorno local y controlado, fue a través del desarrollo de una pequeña red simulada en *GNS3*. Esta red consistía en 3 routers conectados entre sí, los cuales tenían configurado OSPF para el intercambio de rutas. Dicha red se configuró para recibir tráfico de la red de al Universidad, permitiendo realizar pruebas de configuración y monitoreo a través de la API (para ver red, ver figura 182).

A través de todo esto, pudimos comprobar que la API fue desplegada correctamente en el clúster de *Kubernetes*, y que permitió realizar diferentes acciones de configuración y monitoreo en dispositivos de red.

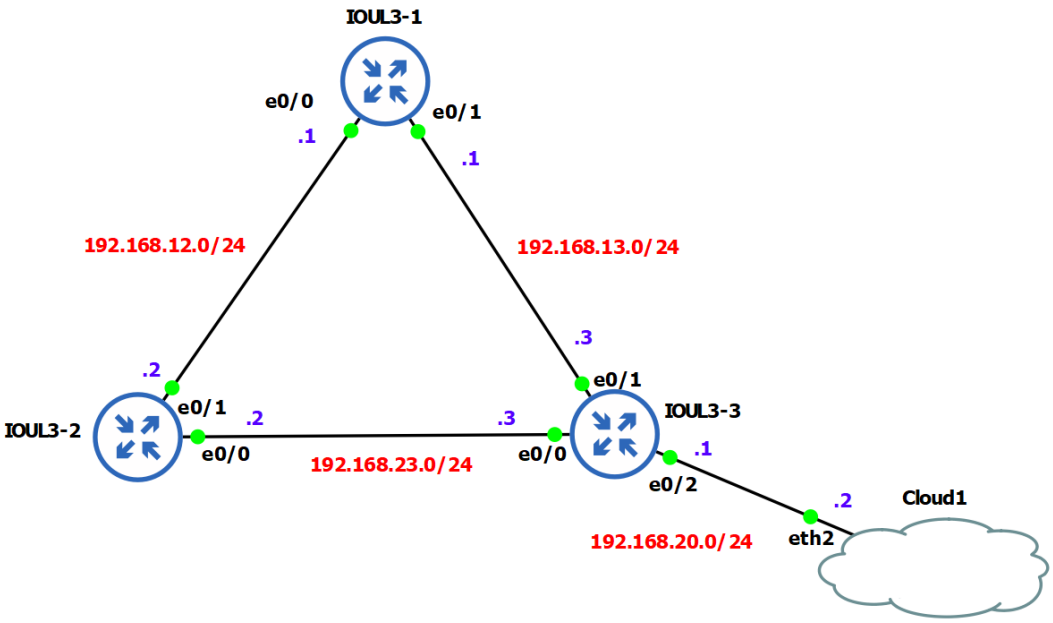


Figura 182: Red simulada en *GNS3* para integrar y probar la comunicación entre la aplicación desplegada y dispositivos de red simulados.

CAPÍTULO 9

Conclusiones

1. **Implementación exitosa de un clúster de *Kubernetes* de alta disponibilidad con un nodo de almacenamiento *NFS*:** Se logró configurar y desplegar un clúster de *Kubernetes* altamente disponible, incorporando un servidor *NFS* para proporcionar almacenamiento compartido entre los nodos y dos balanceadores de carga, lo que facilitó la gestión de datos y la persistencia de información.
2. **Utilización de *Helm* para el despliegue eficiente de aplicaciones:** Se implementó *Helm* como herramienta de gestión de paquetes para *Kubernetes*, permitiendo el despliegue automatizado y simplificado de aplicaciones, mejorando la eficiencia y reduciendo la complejidad del proceso de implementación.
3. **Integración del clúster de *Kubernetes* con una red virtual en *GNS3*:** Se logró conectar el clúster de *Kubernetes* a una red virtual simulada en *GNS3*, lo que permitió realizar pruebas de aplicaciones que automatizan redes en entornos controlados y replicar escenarios de red realistas.
4. **Desarrollo y despliegue de una aplicación en *Python* para la automatización y monitoreo de redes:** Se diseñó una aplicación utilizando *Python*, *FastAPI* y *Netmiko*, la cual fue desplegada en el clúster de *Kubernetes*. Esta aplicación demostró la capacidad del clúster para soportar aplicaciones de automatización de redes y validó su funcionalidad en entornos reales.
5. **Implementación de *Calico* como solución de red para los pods:** Se integró *Calico* como la solución de red dentro del clúster de *Kubernetes*, permitiendo una comunicación eficiente, segura y escalable entre los pods, mejorando el rendimiento general de las aplicaciones desplegadas.
6. **Elaboración de una guía detallada de instalación y configuración del clúster de *Kubernetes*:** Se creó un manual exhaustivo que documenta todos los pasos y configuraciones realizadas, incluyendo la instalación de *Kubernetes*, configuración de los balanceadores de carga, implementación de *Calico* y *NFS*, y el uso de *Helm*. Este manual proporciona una valiosa referencia para futuros proyectos y facilita la replicación del entorno.

7. **Análisis de las limitaciones del despliegue de *ONAP* en entornos locales:** Durante el proyecto, se intentó desplegar *ONAP* en el clúster de *Kubernetes*, pero se enfrentaron múltiples errores y desafíos debido a la complejidad y requisitos de recursos de *ONAP*. Este análisis permitió identificar las limitaciones de los entornos locales para aplicaciones de gran escala y resaltó la necesidad de utilizar servicios en la nube para este tipo de implementaciones.

CAPÍTULO 10

Recomendaciones

- **Utilizar entornos de prueba con mayores recursos:** Se recomienda emplear PCs con mayor capacidad de RAM y almacenamiento para evitar problemas al desplegar tecnologías como *Kubernetes* o plataformas como *ONAP*. Un hardware más potente reducirá las limitaciones y mejorará el rendimiento durante las implementaciones.
- **Capacitación del personal técnico y de TI:** Es esencial realizar capacitaciones para técnicos y personal de TI, para que se encuentren familiarizados con estas tecnologías y puedan gestionarlas adecuadamente. Esto también ayudará a evitar interrupciones en el funcionamiento, como apagar las computadoras de forma inesperada, lo que puedan causar problemas en los despliegues.
- **Elaboración de documentación detallada y accesible:** Se sugiere crear documentación clara y sencilla de entender para facilitar la instalación y configuración de tecnologías como *Kubernetes* y plataformas como *ONAP*. Dado que fue necesario realizar mucha investigación y experimentación para cumplir con la instalación de estas tecnologías, una documentación bien estructurada ayudará a futuros usuarios a ahorrar tiempo y reducir errores.
- **Mejora de la infraestructura del aula y equipos de trabajo:** Se recomienda mejorar la infraestructura del aula donde se encuentran las computadoras utilizadas por los alumnos para trabajos de graduación. Esto asegura un entorno estable y adecuado que minimice problemas técnicos y facilite el desarrollo de proyectos complejos.
- **Mejorar la documentación y soporte de *ONAP*:** Dado que el intento de desplegar *ONAP* no fue exitoso debido a múltiples errores y falta de claridad en la documentación, se recomienda mejorar la documentación oficial de la plataforma. Es importante proporcionar guías más detalladas y actualizadas, así como establecer canales de soporte y foros activos donde los usuarios puedan plantear preguntas y compartir soluciones a problemas comunes.

CAPÍTULO 11

Bibliografía

[1] A. M. Luque, «Developing and Deploying NFV solutions with OpenStack, Kubernetes and Docker,» Tesis de mtría., Universitat Politècnica de Catalunya, 2019.

[2] J. P. A. da Silva, «Service Modelling and End-to-End Orchestration in 5G Networks,» Tesis de mtría., Faculdade de Engenharia da Universidade do Porto, 2019.

[3] S. Flocco, «Experimentation of End-to-End Telco Services through a Service Portal Operating on an ONAP Orchestrator,» Tesis de mtría., Polytechnic University of Turin, 2020.

[4] M. Eberhard, L. Dina y L. Schlunegger, «Cloud Native Intent Automation,» Bachelor Thesis, Tesis de mtría., OST - University of Applied Sciences, 2023. dirección: `https://nephio.org/`.

[5] O. Arouk y N. Nikaein, «5G Cloud-Native: Network Management & Automation,» en *Proceedings of the International Conference on Network Operations and Management Symposium (NOMS)*, IEEE, 2020. dirección: `https://doi.org/10.1109/NOMS47738.2020.9110392`.

[6] F. Bruno, «Automation and Provisioning of Kubernetes on Bare-Metal Telco Edge Infrastructures,» Master's Degree Thesis, Tesis de mtría., Politecnico di Torino, 2024. dirección: `https://www.polito.it/`.

[7] A. T. Marin, «Containerizing ONAP using Kubernetes and Docker,» Master's Thesis in Applied Telecommunications and Engineering Management (MASTEAM), Tesis de mtría., Universitat Politècnica de Catalunya, 2019. dirección: `https://upc.edu/`.

[8] T. L. Foundation, *ONAP Documentation*, ONAP Community, 2024. dirección: `https://docs.onap.org/`.

[9] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini y H. Flinck, «Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions,» *IEEE Communications Surveys & Tutorials*, vol. 20, n.º 3, págs. 2429-2453, 2018.

[10] R. Vilalta y et al., «End-to-End Network Service Orchestration: An Ongoing Journey from INSPIRE to 5G CROSSHAUL,» *Journal of Optical Communications and Networking*, vol. 10, n.º 4, págs. 375-384, 2018.

[11] L. Foundation, *CCSDK Documentation*, https://onap.readthedocs.io/en/latest/submodules/ccsdk/, 2023.

[12] L. Foundation, *ORAN Documentation*, https://onap.readthedocs.io/en/latest/submodules/oran/, 2023.

[13] O. Zimmermann, «Microservices tenets,» *Computer Science-Research and Development*, vol. 32, n.º 3, págs. 301-310, 2016.

[14] C. Pahl, «Containerization and the PaaS Cloud,» *IEEE Cloud Computing*, vol. 2, n.º 3, págs. 24-31, 2015.

[15] Q. Zhu y et al., «The Application of Virtualization Technology in Cloud Computing,» *Journal of Cloud Computing*, vol. 7, n.º 1, págs. 1-13, 2018.

[16] P. Barham y et al., «Xen and the Art of Virtualization,» *ACM SIGOPS Operating Systems Review*, vol. 37, n.º 5, págs. 164-177, 2003.

[17] M. Casado y et al., «Software-defined Networking: A Comprehensive Survey,» *Proceedings of the IEEE*, vol. 102, n.º 1, págs. 14-76, 2014.

[18] S. Venkatraman y D. Soni, «A Survey on Storage Virtualization,» *International Journal of Computer Applications*, vol. 162, n.º 1, págs. 34-39, 2017.

[19] S. A. A. Mujtaba y et al., «Virtual Desktop Infrastructure: A Survey,» *International Journal of Computer Science and Network Security*, vol. 14, n.º 7, págs. 11-17, 2014.

[20] T. L. Foundation, *Introduction to Kubernetes*, edX, Curso en línea, 2024. dirección: https://www.edx.org/learn/kubernetes/the-linux-foundation-introduction-to-kubernetes (visitado 01-09-2024).

[21] C. Boettiger, «An Introduction to Docker for Reproducible Research,» *ACM SIGOPS Operating Systems Review*, vol. 49, n.º 1, págs. 71-79, 2015.

[22] D. Merkel, «Docker: Lightweight Linux Containers for Consistent Development and Deployment,» *Linux Journal*, vol. 2014, n.º 239, pág. 2, 2014.

[23] R. Varun y S. R, «Scaling Applications with Docker Swarm,» *International Journal of Advanced Research in Computer Science*, vol. 11, n.º 2, págs. 33-38, 2020.

[24] T. K. Authors, *Kubernetes Documentation*, https://kubernetes.io/docs/, 2023.

[25] B. Burns, J. Beda y K. Hightower, «Kubernetes: Up and Running,» *O'Reilly Media*, vol. 1, págs. 1-200, 2016.

[26] H. Kelsey y et al., «Kubernetes: The Future of Cloud Infrastructure,» *Journal of Cloud Computing*, vol. 6, n.º 1, págs. 12-15, 2017.

[27] K. Hightower y et al., «Kubernetes: Up and Running: Dive into the Future of Infrastructure,» *O'Reilly Media*, vol. 2, págs. 20-25, 2017.

[28] Kubernetes Documentation Team, *Cluster Architecture | Kubernetes*, Accessed: 2024-08-30, 2024. dirección: https://kubernetes.io/docs/concepts/architecture/.

[29] Kubernetes Documentation Team, *Kubernetes Components | Kubernetes*, Accessed: 2024-08-30, 2024. dirección: https://kubernetes.io/docs/concepts/overview/components/.

[30] Kubernetes Documentation Team, *Creating Highly Available Clusters with kubeadm | Kubernetes*, Accessed: 2024-08-30, 2024. dirección: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/.

[31] Kubernetes Documentation Team, *Considerations for large clusters | Kubernetes*, Accessed: 2024-08-30, 2024. dirección: https://kubernetes.io/docs/setup/best-practices/cluster-large/.

[32] Kubernetes Documentation Team, *Creating a cluster with kubeadm | Kubernetes*, Accessed: 2024-08-30, 2024. dirección: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.

[33] L. M. F. Alan Hohn, *Getting Started with Kubernetes*, 2018. dirección: https://dzone.com/refcardz.

[34] M. Mannambeth, *Kubernetes for Beginners*, 2021. dirección: https://dzone.com/refcardz.

[35] DZone, *Kubernetes eBook Bundle*, 2019. dirección: https://dzone.com/refcardz.

[36] T. H. Authors, *Helm Documentation*, https://helm.sh/docs/, 2023.

[37] S. Knight, D. Weaver, D. Whipple et al., *Virtual Router Redundancy Protocol*, RFC 2338, 1998. dirección: https://www.rfc-editor.org/rfc/rfc2338.

[38] R. Hinden, *Virtual Router Redundancy Protocol (VRRP)*, RFC 3768, 2004. dirección: https://www.rfc-editor.org/rfc/rfc3768.

[39] S. Nadas, *Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6*, RFC 5798, 2010. dirección: https://www.rfc-editor.org/rfc/rfc5798.

[40] Docker, *Docker documentation*, 2024. dirección: https://docs.docker.com.

[41] Mirantis, *Install cri-dockerd Manually*, https://mirantis.github.io/cri-dockerd/usage/install-manually/, Accessed: 2024-11-16, 2024.

CAPÍTULO 12

Glosario

API : Application Programming Interface, conjunto de reglas que permiten que diferentes aplicaciones interactúen entre sí mediante solicitudes y respuestas. 13

APIs REST : Interfaz de programación de aplicaciones que sigue los principios de la arquitectura REST, facilitando la comunicación entre sistemas a través de HTTP. 12

Cloud Computing : Modelo de prestación de servicios de TI en el que los recursos (como almacenamiento, procesamiento y aplicaciones) se ofrecen a través de internet. 18

Cluster : Conjunto de máquinas o nodos que trabajan juntos para ejecutar aplicaciones distribuidas, proporcionando alta disponibilidad y escalabilidad. 21

Host : Máquina física o virtual en la que se ejecutan aplicaciones o servicios dentro de un clúster o red. 21

Kernel : Parte central de un sistema operativo que gestiona los recursos del hardware y permite que las aplicaciones interactúen con estos recursos. 20

Kubeadm : Herramienta que facilita la configuración de un clúster de Kubernetes, permitiendo la instalación y gestión de los nodos del clúster. 23

NFV : Network Function Virtualization, tecnología que permite la virtualización de funciones de red que tradicionalmente se ejecutaban en hardware dedicado. 11

Persistent Volume Claim : Un *Persistent Volume Claim* es una solicitud de almacenamiento persistente realizada por un usuario en un clúster de *Kubernetes*. Es similar a un *Pod* en que los usuarios pueden solicitar recursos, en este caso, almacenamiento persistente, y *Kubernetes* se encarga de vincularla con un *Persistent Volume* adecuado. 75

Persistent Volumes : Un *Persistent Volume* es un recurso de almacenamiento en un clúster de *Kubernetes* que ha sido aprovisionado de forma estática o dinámica y que es independiente del ciclo de vida de los *Pods*. Proporciona una abstracción para el almacenamiento persistente. 75

SDN : Software Defined Networking, arquitectura que permite gestionar redes de manera centralizada mediante software en lugar de configuraciones manuales de hardware. 11

TI : Tecnologías de la Información, un término que se refiere al uso de sistemas computacionales, redes y almacenamiento para crear, procesar y gestionar información. 18

TOSCA : Topology and Orchestration Specification for Cloud Applications, un estándar que permite la definición de topologías de servicios en la nube. 12

Voice Over LTE (VoLTE) : Tecnología que permite realizar llamadas de voz sobre redes LTE (Long-Term Evolution), optimizando el uso del espectro y mejorando la calidad de las llamadas. 12

YAML File : Formato de archivo de texto sencillo utilizado para escribir configuraciones en Kubernetes y otras herramientas. Es legible para humanos y utiliza indentación. 20