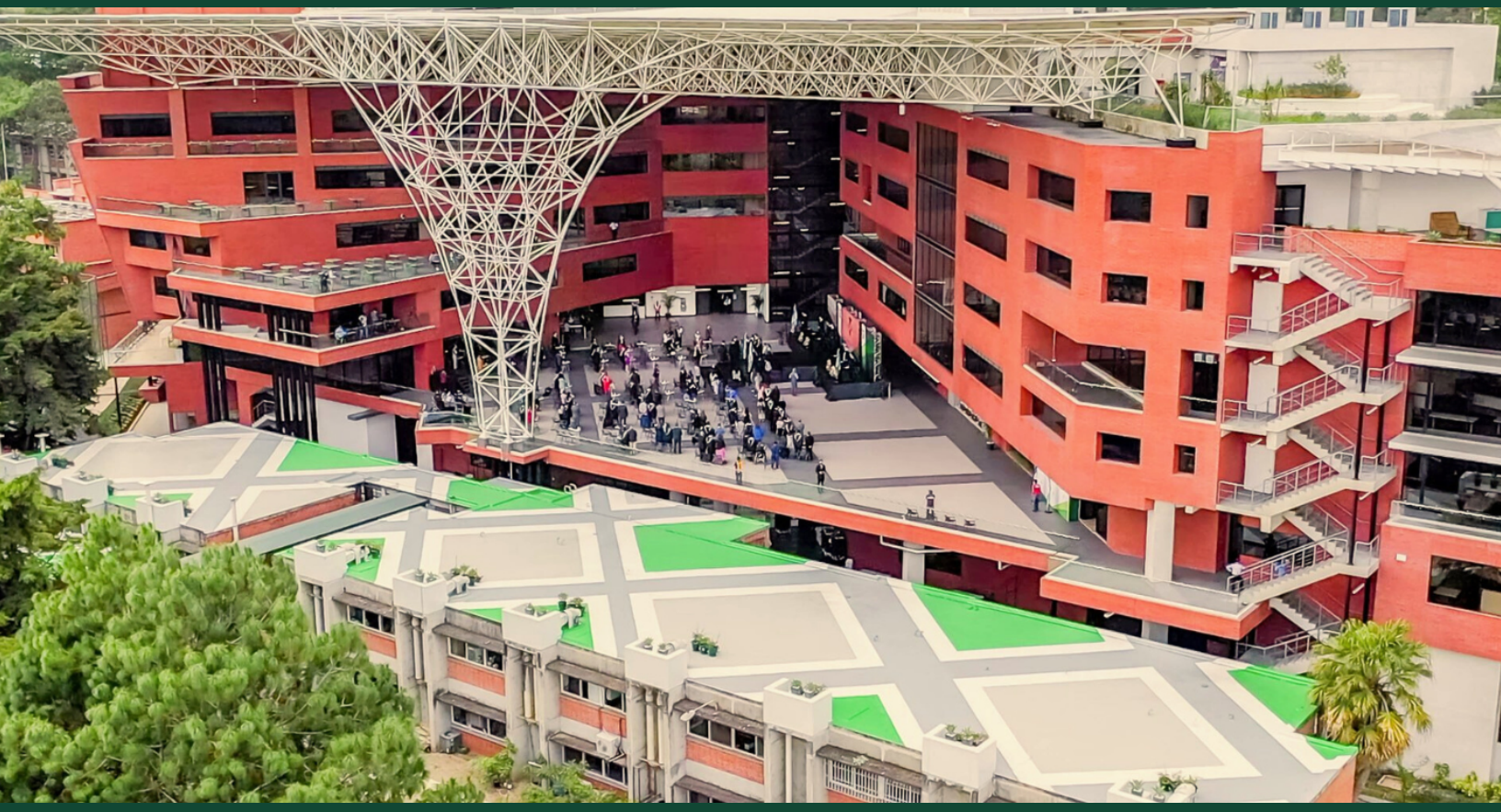
Despliegue de VPN L2 y L3 sobre una red MPLS con monitoreo y optimización basados en inteligencia artificial y aprendizaje de máquina

Michelle Andrea Serrano Monzón



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Despliegue de VPN L2 y L3 sobre una red MPLS con monitoreo y optimización basados en inteligencia artificial y aprendizaje de máquina

Trabajo de graduación presentado por Michelle Andrea Serrano Monzón
para optar al grado académico de Licenciada en Ingeniería Electrónica

Guatemala,

2025

Vo.Bo.:

(f) 

M.Sc. Jonathan de los Santos

(f) 

M.Sc. Carlos Esquit Hernández

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

Prefacio

Para mi hermana, Melissa. En memoria de la última noche que compartimos; aunque hoy no puedas estar aquí, esto es para ti.

Índice general

Índice de figuras

Índice de códigos

Resumen

El presente proyecto tuvo como propósito el diseño y la implementación de redes privadas virtuales (VPN) de capa 2 y 3 sobre una infraestructura MPLS virtualizada en el emulador GNS3, utilizando routers Cisco con capacidades de segmentación basadas en VRF y BGP. Su objetivo principal consistió en garantizar la escalabilidad y el aislamiento del tráfico, emulando entornos equivalentes a los de proveedores de servicios.

La topología MPLS fue completamente virtualizada y configurada con el protocolo de enrutamiento OSPF para el establecimiento de label switched paths (LSP), sobre los cuales se implementaron VPNv4 mediante BGP. El sistema fue complementado con la integración del software de monitoreo Zabbix, a través del cual se recolectaron métricas de latencia, pérdida de paquetes, disponibilidad e interrupciones de interfaz, almacenadas para su análisis posterior. A partir de estos datos, se desarrolló un modelo predictivo de aprendizaje automático, tipo árbol de decisión, entrenado sobre el histórico de caídas de interfaz para estimar la probabilidad de falla de cada enlace. Dicho modelo generó predicciones con su probabilidad asociada en un horizonte de doce horas, información utilizada por un módulo de optimización para ajustar dinámicamente los costos OSPF, lo que favoreció rutas más estables. Además, se integró un asistente inteligente basado en Google Gemini, encargado de analizar problemas registrados por Zabbix y proponer causas y recomendaciones automáticas ante incidentes detectados.

Los resultados demostraron una operación estable de las VPNs, una reducción en los tiempos de convergencia y una mejora en la resiliencia del servicio. Asimismo, la aplicación combinada de aprendizaje automático e inteligencia artificial fortaleció la capacidad predictiva y de diagnóstico del sistema, permitiendo una gestión más eficiente y proactiva de la red.

Palabras clave: MPLS, VPN L2/L3, inteligencia artificial, aprendizaje de máquina.

Abstract

This project aimed to design and implement Layer 2 and Layer 3 Virtual Private Networks (VPNs) over a virtualized MPLS infrastructure within the GNS3 emulator, using Cisco routers with segmentation capabilities based on VRF and BGP. Its main objective was to ensure scalability and traffic isolation, emulating environments equivalent to those of service providers.

The MPLS topology was fully virtualized and configured with the OSPF routing protocol for the establishment of Label Switched Paths (LSP), over which VPNv4 were deployed through BGP. The system was complemented by the integration of the Zabbix monitoring software, through which metrics such as latency, packet loss, availability, and interface interruptions were collected and stored for further analysis.

Based on this data, a predictive machine learning model, using a decision-tree algorithm, was trained on the historical record of interface failures to estimate the likelihood of future link outages. The model generated failure probability predictions within a 12-hour horizon, which were used by an optimization module to dynamically adjust OSPF costs, prioritizing more stable routes. In addition, an intelligent assistant based on Google Gemini was integrated to analyze problems reported by Zabbix and provide automated root causes and recommendations for detected incidents.

The results demonstrated stable VPN operation, reduced convergence times, and improved network resilience. Furthermore, the combined application of machine learning and artificial intelligence strengthened the system's predictive and diagnostic capabilities, enabling more efficient and proactive network management.

Keywords: MPLS, VPN L2/L3, artificial intelligence, machine learning.

CAPÍTULO 1

Introducción

En el campo de las telecomunicaciones y la ingeniería de redes, la creciente demanda de conectividad segura y escalable de alto rendimiento ha impulsado el desarrollo de tecnologías que permitan segmentar y aislar el tráfico de múltiples clientes sobre una misma infraestructura física. Dentro de este contexto, el Protocolo de Conmutación de Etiquetas Multiprotocolo (MPLS) se ha consolidado como una solución fundamental para los proveedores de servicio, al permitir el transporte eficiente de tráfico mediante el uso de etiquetas en lugar de rutas tradicionales. Sobre esta base, las redes privadas virtuales (VPN) de capa 2 y 3 constituyen una herramienta clave para ofrecer servicios de conectividad punto a punto y multipunto, manteniendo independencia lógica entre usuarios.

El presente trabajo de graduación tuvo como finalidad diseñar e implementar redes VPN L2 y L3 sobre una red MPLS virtualizada, empleando el emulador de red GNS3 y routers Cisco configurados con VRF y BGP para la propagación de rutas VPNv4. Además, se incorporaron mecanismos de monitoreo y optimización basados en inteligencia artificial (IA) y aprendizaje automático (ML, por sus siglas en inglés), orientados a mejorar la gestión, estabilidad y capacidad predictiva de la red.

La integración de estos componentes permitió ampliar las capacidades de observabilidad del entorno virtual. Por un lado, el sistema de monitoreo Zabbix fue utilizado para recolectar métricas críticas de latencia, pérdida de paquetes, disponibilidad y estado operativo de las interfaces, que posteriormente fueron procesadas y almacenadas para su análisis. A partir de estos datos, se entrenó un modelo predictivo supervisado tipo árbol de decisión, capaz de estimar la probabilidad de falla de cada enlace en un horizonte de doce horas. Estas predicciones fueron empleadas por un módulo de optimización dinámica que ajustó automáticamente los costos OSPF, de manera que los enlaces con mayor riesgo de inestabilidad fueran penalizados, lo que promovió rutas alternativas más confiables dentro del dominio MPLS.

De forma complementaria, se desarrolló un asistente inteligente basado en la inteligencia artificial de Google Gemini, encargado de interpretar eventos y alertas registradas por Zabbix. Este componente analizó el contexto de las fallas detectadas y generó explicaciones automáticas, posibles causas y recomendaciones de solución, fortaleciendo la capacidad de diagnóstico y respuesta del operador.

El tema se delimitó al entorno de simulación mediante GNS3, priorizando la implementación funcional de tecnologías MPLS y su integración con herramientas de monitoreo y automatización. Se adoptó una metodología experimental y analítica que comprendió la virtualización completa de la topología, la configuración de los protocolos de enrutamiento (OSPF, BGP) y el desarrollo de los módulos inteligentes de análisis y optimización.

Entre las principales conclusiones se destacaron la operación estable de las VPN L2 y L3, la correcta propagación de rutas IPv4 y la mejora en la resiliencia del servicio gracias a la aplicación combinada de técnicas de aprendizaje automático y de inteligencia artificial. Este enfoque demostró el potencial de la automatización inteligente para anticipar fallas, optimizar el desempeño del enrutamiento y promover la administración proactiva de redes MPLS en entornos académicos y profesionales.

CAPÍTULO 2

Antecedentes

2.1. Red MPLS

Multiprotocol Label Switching(MPLS) se ha convertido en una alternativa más eficiente al enrutamiento IP tradicional, dando como resultado una solución en áreas como ingeniería de tráfico (TE) y QoS. [1] En el trabajo de titulación "Análisis y diseño de una red MPLS con BGP emulando un escenario típico de empresas multinacionales" se resalta cómo MPLS, en su implementación, permite un transporte de paquetes con mayor ancho de banda y velocidades de transmisión al utilizar etiquetas que guían el encaminamiento de paquetes sin requerir el análisis del encabezado de IP. [2]

En dicho trabajo , se distinguen dos dispositivos principales, los cuales, juegan un papel crucial dentro de la arquitectura MPLS: La unidad básica de la red, *Label Switching Router* (LSR) y la unidad conectada a otras redes, *Label Edge Router.*(LER)

El objetivo final del proyecto es que la red MPLS diseñanda ofrezca soporte *dual stack* (IPv4) con protocolos como BGP y OSPF. Permitiendo mostrar el funcionamiento y características al priorizar el tráfico a través de comparativa de valores de QoS, facilitando una gestión más inteligente y adaptable. [2]

2.2. Virtualización de red

El trabajo de titulación "Ruta óptima de tráfico de una red virtual basada en análisis de datos y algoritmo de *machine learning*" [3] resalta cómo la virtualización de las redes, implementadas de manera adecuada, permite mejorar aproximadamente entre 5 al 15 % de su capacidad operativa. Por su parte, el proyecto "Integración y optimización de redes

MPLS: Un caso práctico" resalta la implementación de VPN MPLS para soportar servicios tanto de capa 2 como de capa 3, detallando las instancias de enrutamiento y reenvio virtual *(virtual routing forwarding)*(VRF), manteniendo aisladas las rutas aprendidas mejorando así la eficiencia y seguridad.

La virtualización no solo habilita la consolidación de recursos, sino que también permite una administración dinámica y adaptable de redes, elemento clave en entornos habilitados para AI y ML. [4]

La tesis final para grado de magister titulada "Diseño de una red privada virtual con tecnología MPLS" evidencia el uso de la herramienta GNS3 como un recurso esencial para la implementación de redes MPLS en contextos virtuales. La utilización de esta simulación permite obtener arquitecturas de red idénticas a las que se observarían en una infraestructura real sin necesidad de hardware físico.

El uso de GNS3 permitió implementar una red completamente funcional que incorpora características MPLS, la creación de VRF, el establecimiento de LSP y el enrutamiento BGP. Esta virtualización no solo facilitó la evaluación del rendimiento de la red en diversos escenarios de tráfico, sino que también permitió aplicar modelos de ingeniería de tráfico y pruebas de QoS sin necesidad de invertir en equipos físicos. [5]

2.3. VPN, MPLS y BGP

Las redes privadas virtuales (VPN) sobre MPLS representan una solución sólida para la interconexión segura. En la tesis "Integración y optimización de redes MPLS: Un caso práctico" , se abordan paquetes con la misma dirección de destino de nivel 3 que pertenecen a un grupo de prefijos BGP, todos con el mismo siguiente salto de BGP. Todos los paquetes que entran por el mismo *Ingress* LSR y salen por el mismo *Egress* LSR tienen la misma etiqueta. El trabajo explica cómo los prefijos IP son transformados en prefijos VPNv4 al asignarles un Route Distinguisher (RD) y cómo estos son distribuidos entre los routers PE mediante BGP.

El diseño estructurado del protocolo BGP mejora rutas de red para la transmisión de datos, facilitando la propagación eficiente de rutas a través de dominios administrativos. [4]

2.4. AI y ML en *network monitoring & optimization*

La inteligencia artificial y el aprendizaje automático pueden analizar datos anteriores así como el estado actual de dispositivos como routers y switches, para predecir posibles fallos. Esto puede ser útil para programar mantenimiento preventivo y reducir el tiempo de inactividad. Además de la monitorización, se pueden analizar patrones de tráfico en tiempo real, optimizando las decisiones de enrutamiento reduciendo la latencia, aumentar el ancho de banda, evitar cuellos de botella y prevenir interrupciones. [6].

En la tesis "Ruta Óptima de Tráfico de una Red" , la investigación se centra en el análisis y predicción del tráfico mediante algoritmos de *machine learning* para la determinación de la

ruta óptima en una red virtual. Este trabajo demuestra que los algoritmos de ML permiten la predicción de tráfico, el análisis de estado de enlaces y la anticipación de fallas, lo que mejora significativamente la confiabilidad del monitoreo.

El objetivo principal de este trabajo es minimizar la congestión y optimizar el uso de los recursos disponibles en la red, incrementando así su rendimiento global. [4]

CAPÍTULO 3

Justificación

La alta demanda y creciente complejidad de las redes de telecomunicaciones, representan retos significativos en la escalabilidad, confiabilidad y gestión eficiente de recursos de la red donde las técnicas de monitoreo y enrutamiento convencionales resultan insuficientes e inceficientes puesto que, se identifica el problema solo después de sus consecuencias, no se concocen patrones ni tendencias y, a medida que crece la red, se elevan los costos operativos y se reduce considerablemente la calidad de la experiencia del usuario [7]. En redes que deben garantizar la continuidad de servicio, esta limitación es aún más crítica.

El presente trabajo busca dar respuesta a la problemática de gestión y optimización en redes mediante la implementación de VPNs sobre una red MPLS virtualizada. La red integra funcionalidades como BGP y VRF, lo que permite simular un entorno realista de proveedores de servicio. Sobre esta infraestructura se desarrolla un sistema de monitoreo inteligente, encargado de recolectar y analizar datos operativos de la red. Dichos datos utilizados como base para la aplicación de algoritmos de Aprendizaje de Máquina, orientados a optimizar dinámicamente el enrutamiento, mejorar el desempeño del tráfico y predecir fallas mediante modelos predictivos.

Dicho proyecto, aporta una estructura que combina virtualización, protocolos de enrutamiento y técnicas de análisis de datos e inteligencia artificial que resultan de alto valor en áreas de alta demanda actual, ofreciendo una referencia práctica para proveedores de servicios y empresas que buscan optimizar sus infraestructuras de comunicación al mejorar los niveles de calidad de servicio. El impacto es significativo al promover redes más estables y seguras, que soporten servicios críticos en sectores como salud, banca y comercio electrónico, impactando de manera directa en la confiabilidad de las interacciones digitales.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Diseñar e implementar redes privadas virtuales L2 y L3 sobre una red MPLS virtualizada mediante GNS3, utilizando routers Cisco configurados con capacidades de segmentación como VRF y BGP para propagación de rutas VPNv4 entre sitios remotos, garantizando escalabilidad y aislamiento del tráfico. Además, procesos de monitoreo y optimización basados en aprendizaje de máquina e inteligencia artificial para mejorar la calidad del servicio.

4.2. Objetivos específicos

- Implementar una red MPLS virtualizada que soporte VPNs de capa 2 y capa 3, integrando funcionalidades de BGP, LSP y VRF mediante IOS en GNS3.
- Desarrollar un sistema de monitoreo que recolecte datos operativos de la red utilizando herramientas estadísticas y visualización.
- Aplicar algoritmos de Inteligencia Artificial y Aprendizaje de Máquina para optimizar dinámicamente el enrutamiento, mejorando el desempeño del tráfico de red.
- Construir un modelo predictivo basado en técnicas de aprendizaje supervisado para anticipar fallas en la red, a partir de registros históricos de tráfico y eventos.

- Integrar un módulo de gestión que combine monitoreo en tiempo real, análisis de desempeño y predicciones de fallos, capaz de ejecutar o sugerir reconfiguraciones automáticas de tráfico.
- Documentar detalladamente el diseño e implementación, incluyendo topologías, pruebas, simulaciones, configuraciones, estrategias de monitoreo y optimización.

CAPÍTULO 5

Alcance

El proyecto está enfocado al diseño y la implementación virtualizada de una red MPLS sobre el emulador de software de red GNS3 empleando *routers* Cisco para simular un entorno de proveedor de servicios por medio de, *Routers* de borde del proveedor (PE), *Router* del proveedor (P), *Customer Router* perimetral (CE) y un *Route Reflector* para simular condiciones reales de red. La configuración incuye la aplicación de protocolos de enrutamiento como OSPF, BGP y mecanismos de VRF. Se desarrollan e integran redes privadas virtuales (VPNs) para lograr la segmentación de tráfico y propagación de rutas y sitios remotos.

De igual forma, se emplea la herramienta de monitoreo *Zabbix* para recolectar y visualizar métricas clave del comportamiento de la red tales como latencia, ancho de banda, disponibilidad de enlaces, pérdida de paquetes y desempeño de rutas. Estas métricas se analizan con técnicas de Aprendizaje de Máquina e Inteligencia Artificial, para detectar patrones y tendencias en el tráfico, predecir posibles fallas a partir de registros históricos y así sugerir ajustes dinámicos de enrutamiento que mejoren la eficiencia de la red. Se evaluará el comportamiento de la red bajo diferentes condiciones de tráfico. Se verificará la conectividad de tráfico entre sitios de clientes a través de pruebas en las rutas VPNv4 para validar la funcionalidad del diseño.

El alcance del proyecto incluye la documentación detallada de la implementación, configuración, simulaciones y resultados obtenidos, así como el análisis del desempeño bajo condiciones de carga simuladas. Se documentarán todos los procesos de implementación, configuración, resultados de pruebas y análisis del desempeño de la red. No se contempla la implementación de una estructura física ni el despliegue en entornos productivos, limitando el alcance a simulaciones controladas. El proyecto se limita a la validación técnica y funcionalidad de los conceptos clave.

CAPÍTULO 6

Marco teórico

6.1. *Multiprotocol Label Switching*(MPLS)

Multiprotocol Label Switching(MPLS) es una tecnología de redes de telecomunicaciones que utiliza la conmutación de paquetes basada en etiquetas, en vez del reenvío tradicional por direcciones de IP. MPLS supera las limitaciones de enrutamiento tradicional al agrupar los paquetes y asignarles etiquetas que determinan su trayectoria a través de la red.

Al ingresar a una red MPLS, los paquetes son clasificados y etiquetados por *routers* de "borde"(*Label Edge routers - LER*), mientras que los *routers* "internos"(*Label Switch routers - LSRs*) utilizan dichas etiquetas para determinar el siguiente salto, evitando así el procesamiento completo de las cabeceras IP, lo que reduce significativamente la complejidad del procesamiento y mejora el rendimiento del reenvío de paquetes. Dicha arquitectura permite la creación de rutas conmutadas denominadas *Label Switched Paths*(LSP), que garantizan la Calidad de Servicio (QoS) [8].

6.1.1. Encaminamiento basado en etiquetas

El encaminamiento basado en etiquetas, central en tecnologías como MPLS, permite acelerar significativamente el reenvío de paquetes en redes conmutadas al eliminar el análisis de encabezados IP en cada salto de red y reenviar el paquete mediante la tabla de etiquetas. Los *routers* utilizan una etiqueta corta y de longitud fija que identifica de forma única el flujo de tráfico al que pertenece el paquete.

Este mecanismo facilita que el reenvío se realice a nivel de *hardware* mediante una simple operación de búsqueda y sustitución de etiquetas. Como consecuencia, se reduce la complejidad computacional y la latencia de procesamiento lo que mejora el rendimiento

general de la red, logrando una arquitectura más escalable y eficiente ante la recuperación de fallos [9].

6.1.2. Separación entre plano de control y plano de datos

Las redes tradicionales están integradas por el plano de control (que decide cómo gestionar el tráfico de red) y el plano de datos (que reenvía el tráfico según las decisiones tomadas por el plano de control), ambos planos, están agrupados dentro del mismo dispositivo de red. Sin embargo, en las redes definidas por *software* (SDN) la separación de ambos planos es fundamental [10]. Esta separación puede ser especialmente ventajosa en la arquitectura de MPLS, logrando que esta sea más simple al mantener el plano de datos MPLS con la eficiencia del reenvío mediante etiquetas, y reemplazar el complejo plano de control tradicional por un plano de control basado en SDN. [11].

Al abstraer y centralizar el plano de control en un controlador lógico, y reducir los dispositivos de red como *routers* y *switches* a elementos con funciones de reenvío. Se permite una gestión más flexible, logrando la adaptación dinámica a cambios en la topología o en la demanda de tráfico.

El reenvío de paquetes, ahora ubicado exclusivamente en el plano de datos, se realiza de manera eficiente, siguiendo las decisiones preestablecidas por el controlador. Esta arquitectura elimina la necesidad de que cada dispositivo mantenga su propia lógica de control, facilita escalabilidad y reduce la complejidad en los nodos intermedios de la red [12].

6.1.3. Uso de clases de equivalencia de reenvío (FEC)

Las Clases de Equivalencia de Reenvío *Forwarding Equivalence Class* (FEC) representan un concepto fundamental en la arquitectura MPLS, permiten asignar una única etiqueta a todos los paquetes de una clase, es decir, recibirán un tratamiento idéntico en su proceso de reenvío por la red. Permiten que el plano de control defina rutas lógicas optimizadas, mientras que el plano de datos se limita a reenviar paquetes según sus etiquetas [13].

Una FEC constituye un grupo de paquetes que comparten requisitos similares para su transporte agrupados según criterios contextuales del tráfico, lo cual simplifica el procesamiento en los nodos intermedios. Permitiendo que todos los paquetes que pertenecen a una misma FEC sigan la misma ruta eliminando la necesidad de realizar búsquedas complejas en las tablas de encaminamiento en cada salto. [14]

6.1.4. Establecimiento de LSP (*label switched paths*)

Label Switched Path es la ruta unidireccional preestablecida en una red MPLS, a través de la cual fluyen los paquetes basados en etiquetas. La LSP está definida por una secuencia coordinada de *routers* que procesan y envían los paquetes, implicando una coordinación entre la LER, donde se aplica la primera etiqueta y la LSR, que gestiona la conmutación a lo largo de la ruta. [1].

Para un establecimiento dinámico se utilizan protocolos como LDP *Label Distribution Protocol* donde se siguen las rutas IP convencionales como OSPF o RSVP-TE *Resource Reservation Protocol - Traffic Engineering* que permite definir trayectorias precisas independientes de las tablas de enrutamiento estándar. Ambos protocolos permiten que los *routers* intercambien información sobre las etiquetas y así establezcan rutas que cumplan los criterios de rendimiento de la red. Una vez establecida, el *router* solo consulta la etiqueta para determinar el siguiente salto, sin necesidad de analizar el encabezado IP completo [8].

6.1.5. *Label switching router*(LSR)

El *Label Switch Router*(LSR) constituye un componente fundamental en la arquitectura MPLS, opera en el núcleo de la red y realiza funciones de conmutación de etiquetas. Estos dispositivos implementan el plano de control para la distribución de etiquetas y el plano de datos para la conmutación de paquetes basada en etiquetas.

Los LSR examinan únicamente la etiqueta de entrada del paquete MPLS para determinar el próximo salto y la operación de etiqueta correspondiente. Este proceso de conmutación se realiza mediante la consulta de la *Label Forwarding Information Base*(LFIB), una tabla que correlaciona las etiquetas de entrada con las etiquetas de salida y las interfaces correspondientes. Los LSR participan activamente en protocolos de señalización como LDP (*Label Distribution Protocol*) o RSVP-TE (*Resource Reservation Protocol - Traffic Engineering*) para establecer y mantener los *Label Switched Paths* (LSP) a través de la red. [8]

6.1.6. Label edge router (LER)

El *Label Edge Router* (LER) representa un elemento crítico en la arquitectura MPLS que opera en la parte externa de la red y actúa como punto de interconexión entre dominios MPLS y no-MPLS.

Realizan la clasificación del tráfico entrante basándose en múltiples criterios (dirección IP, puertos, QoS) para asignarlo a *Forwarding Equivalence Classes* (FEC) específicas, imponen las etiquetas MPLS correspondientes y determinan el *Label Switched Path* (LSP) inicial. Los LER implementan una mayor complejidad computacional que los LSR internos, ya que deben mantener la inteligencia necesaria para analizar los encabezados IP completos, aplicar políticas de servicio diferenciadas y traducir a una conmutación basada en etiquetas y enrutamiento basado en destino. [8]

6.1.7. Integración con mecanismos de calidad de servicio (QoS)

La integración de MPLS con mecanismos de Calidad de Servicio (QoS) busca satisfacer los requerimientos de diversas aplicaciones según las exigencias de rendimiento de la red. MPLS facilita la implementación de políticas operativas de QoS. En este contexto, la integración de MPLS e *IntServ* forma la ingeniería de tráfico de conmutación de etiquetas multiprotocolo (MPLS TE), *MPLS Traffic Engineering (MPLS-TE)*, que utiliza el protocolo RSVP-TE para establecer rutas explícitas (LSP) y así reservar recursos en cada nodo

de la red, evitando puntos de congestión. Y la combinación de MPLS y *DiffServ* genera MPLS *DiffServ* que permite diferenciar servicios mediante la clasificación del tráfico en distintas clases, marcando los bits EXP en la cabecera MPLS y proporcionando tratamiento preferencial a los paquetes de alta prioridad.

La combinación de ambos enfoques resuelve las limitaciones individuales, permitiendo separar la reserva de ancho de banda para diferentes clases de tráfico, rastrear dinámicamente la disponibilidad de recursos por clase y garantizar parámetros como latencia, *jitter* y pérdida de paquetes. Esta arquitectura integrada facilita la implementación de servicios avanzados como VoIP, videoconferencia y aplicaciones empresariales críticas, garantizando el cumplimiento de los Acuerdos de Nivel de Servicio (SLA) y optimizando la utilización de recursos de red [15].

6.2. Virtual routing and forwarding (VRF)

Virtual Routing and Forwarding (VRF) es un componente clave de la arquitectura MPLS que permite la división lógica de recursos de enrutamiento dentro de un único dispositivo físico, para así habilitar la coexistencia de múltiples instancias de enrutamiento. Para redes virtuales privadas MPLS (MPLS VPN); esta tecnología es esencial, cada VRF consta de una o más tablas de enrutamiento específicas, una tabla de enrutamiento derivada e interfaces físicas o lógicas que especifican su funcionamiento.

Los *routers* de borde (LER) hacen uso de las instancias VRF para mantener el aislamiento de tráfico entre diferentes clientes o servicios. La red MPLS transporta los paquetes a través de *label switched paths* (LSP) sin necesidad de conocer la topología interna de cada VPN. Las VRF se interconectan mediante *route targets* (RT), que actuan como identificadores para controlar la importación y exportación de rutas entre distintas VRF. Gracias a esta arquitectura, se permite ofrecer servicios VPN de capa 3 con conectividad punto a multipunto, eliminando la necesidad de redundancia virtual dedicada entre cada par de sitios [14].

6.2.1. Tablas de enrutamiento independientes por cliente

Mediante el uso de la tecnología de enrutamiento y reenvío virtual (VRF), cada instancia de VRF mantiene un conjunto de tablas de enrutamiento y reenvío autónomas, aislando el tráfico entre varios clientes. En un MPLS que utiliza VPN, las tablas independientes del cliente constituyen un mecanismo fundamental que permite la coexistencia de múltiples espacios de dirección dentro de una infraestructura compartida. Cada interfaz de cliente en la arquitectura VPN MPLS recibe una VRF específica de los enrutadores de la placa. El intercambio de información entre instancias de VRF se realiza mediante el protocolo *BGP multiprotocolo* (BGP), que transmite tanto la información de la dirección IP como los identificadores de distinción de ruta (RD) y destino de ruta (RT), lo que permite la diferenciación de rutas con preferencias únicas del cliente. [14]

6.2.2. Asignación lógica de interfaces a instancias de reenvío

Red MPLS es clave para la implementación de VPN de capa 3, donde los *routers* de borde utilizan las etiquetas MPLS para identificar la ruta como la VRF asociada al paquete, permitiendo que los *routers* intermedios reenvíen los paquetes sin necesidad de conocer el contexto del cliente. Este proceso, permite que un único *router* físico pueda soportar múltiples dominios de enrutamiento independientes donde cada interfaz se asocia a una ins de VRF específica, lo que garantiza que el tráfico que ingrese por una interfaz se procese únicamente por medio de la tabla de enrutamiento correspondiente. Dicho proceso es esencial para lograr servicios de red multiusuario con apartamiento completo entre clientes. Al asignar interfaces a VRF los paquetes de diferentes clientes no se mezclan, aún si utilizan el mismo direccionamiento IP. Una separación lógica, como de interfaces a instancias de reenvío, proporciona un nivel de seguridad, ya que cada VRF opera como un entorno independiente con sus respectivas reglas de acceso [14].

6.2.3. Segmentación y aislamiento del tráfico

Uno de los principios durante la implementación de redes MPLS incluye la segmentación y aislamiento de tráfico, especialmente en entornos multiservicio donde la separación lógica entre distintos flujos de comunicación es importante. Con la combinación de tecnologías complementarias como VPN MPLS y *Virtual Routing and Forwarding* (VRF), se logra una infraestructura compartida con garantía de distanciamiento lógico.

La estructura MPLS emplea identificadores únicos (etiquetas) y la ténica de ampliado de etiquetas (*label stacking*) para crear canales independientes que garantizan la separación de tráfico entre servicios, lo que evita así filtraciones de datos entre dominios distintos y facilita políticas de seguridad específicas por segmento, así como la asignación de recursos delicados dentro de cada dominio virtualizado.

La segmentación a nivel del plano de control mediante VRF, examina cómo cada cliente o servicio dispone de un enrutamiento individual permitiendo la coexistencia de esquemas de direccionamiento superpuesto y proporcionando aislamiento estricto entre clientes que comparten recursos comunes. En entornos MPLS VPN, la segmentación se facilita mediante *Route Distinguishers* (RD) y *Route Targets* (RT) al lograr diferenciar rutas con prefijos idénticos pero que pertenecen a diferentes servicios para controlar así la importación/exportación de información de enrutamiento VRF [12].

6.2.4. Uso en redes de capa 3 para servicios VPN

Como se a mencionado anterioremente, MPLS permite encapsular paquetes IP dentro de túneles identificados por etiquetas lo que, por consecuencia, facilita la creación de VPN de capa 3. En las L3VPN sobre MPLS, el protocolo BGP distribuye las rutas entre los respectivos *routers* de borde, de manera que las rutas asociadas a etiquetas MPLS permitan reenviar los paquetes a través sin necesidad de *routers* intermedios, puesto que los LSP permitirán establecer rutas explícitas para cada VPN optimizando así los recursos de la red permitiendo escalar a miles de clientes sin comprometer el rendimiento ni seguridad. [14]

6.3. Virtual switching instance (VSI)

Dentro de las implementaciones de servicios VPLS sobre redes MPLS, las VSI representan la funcionalidad de conmutación necesaria para emular redes Ethernet sobre proveedores de servicios. La VSI opera como una instancia vitual del conmutador Ethernet implementado en un *rourter* de borde, cada una asociada a un servicio VPLS específico. Casa VSI mantiene su tabla MAC, realizando funciones de aprendizaje basado en direcciones MAC de destino. Múltiples VSI dentro del mismo equipo físico permite una segmentación total del tráfico entre diferentes clientes logrando aislamiento no solo a nivel de direccionamiento MAC sino también de otros protocolos de control Ethernet.

Las VSI se interconectan mediante túneles de punto a punto que han sido establecidos a través de la red MPLS formando una malla completa que emula un segmento LAN compartido, donde cada túnel punto a punto transporta tramas de Ethernet encapsuladas en paquetes MPLS entre las VSI. Para esto, se incluyen protocolos como BGP y LDP. BGP permite que los *routers* de borde anuncien su participación en el mismo servicio VPLS específicos y descurbir así autónomamente otros participando. LDP, utiliza servicios para señalar los túneles punto a punto entre pares de *routers* de borde. [16]

6.3.1. Conmutación en capa 2

La conmutación en capa 2 opera a nivel del enlace de datos del modelo OSI, utiliza direcciones MAC para determinar el reenvío de paquetes entre dispositivos de una misma red local. A diferencia del enrutamiento en capa 3, no requiere procesamiento de encabezados IP. La evolución en capa 2 ha permitido su integración con arquitecturas MPLS mediante la correcta separación del plano de datos con el plano de control, implementa las tablas de conmutación distribuidas y técnicas de encapsulación que preserva las cabeceras Ethernet con mecanismos de aprendizaje y propagación de direcciones MAC a través de dominios MPLS. Ha constituido la base de la implementación de instancias virtuales VSI en coexistencia con VRF sobre la mimsa infraestructura física MPLS. [14]

El proceso de conmutación en capa 2 se basa en dos mecanismos que resultan críticos para la operación de VSI en entornos MPLS.

- Aprendizaje de direcciones MAC: los *switches* físicos y virtuales construyen tablas de direcciones MAC dinámicamente, en un entorno MPLS L2VPN, se requiere aprendizaje local en el *router* de borde así como distribución de información MAC mediante protocolo BGP, sincronización de las tablas MAC entre instancias VSI y manejo de la información temporal para así asegurar la consistencia entre sitios.
- Proceso de reenvío de tramas: en entornos VSI sobre MPLS, se involucra la identificación de la VSI correspondiente mediante MPLS, consulta de tablas MAC específicas de cada instancia de VSI determinando así la etiqueta de salida correspondiente al *router* de borde remoto.

Integrar estos mecanismos con tecnologías MPLS extienden dominios de conmutación de capa 2 más allá de los límites usuales de LANs constituyendo la base operativa de las L2VPN. [17]

Las tecnologías avanzadas de conmutación en capa 2 se complementan con las implementaciones de VRF, permitiendo el despliegue de servicios integrados de L2/L3 a través de redes VPLS que extienden la funcionalidad LAN a través de MPLS, logrando una tecnología *full-mesh* de túnel punto a punto entre *routers* borde. En este contexto, la VPLS implementa instancias VSI independientes por cliente, aprendizaje MAC e integración para servicios hibrídos L2/L3. Además, la arquitectura avanzada de Ethernet VPN utiliza BGP para control del plano y MPLS para plano de datos, separando los planos de control y datos, aprendizaje MAC controlado mediante protocolos de señalización y capacidad directa de VRF para servicios integrados de L2/L3. [18]

6.3.2. Implementación de servicios multipunto mediante VPLS

VPLS proporciona conectividad multipunto sobre infraestructuras MPLS. A diferencia de las soluciones punto a punto como PWE3 (*Pseudowire Emulation Edge-to-Edge*), VPLS implementa un dominio de conmutación Ethernet virtualizado y distribuido que emula el comportamiento de una LAN conmutada tradicional.

La arquitectura VPLS se fundamenta en; instancias virtuales (VSI) que reperentan virtualmente *switches* Ethernet en los*routers* de borde donde cada VSI mantiene sus propias direcciones MAC y realiza decisiones de reenvío de forma independiente. Los túneles de transporte entre *routers* de borde, que forman una topología *full-mesh* donde se establecen conexiones punto a punto con todos los demás *routers* de borde participanetes en la misma instancia VPLS.

El modelo operativo de VPLS emula los procesos fundamentales de un *switch* Ethernet:

- Aprendizaje de direcciones MAC de origen en tramas entrantes.
- Toma decisiones de reenvío basadas en direcciones MAC de destino.
- *Flooding* de tramas cuando el destino es desconocido. [19]

Para enfrentar los retos de las implementaciones de VPLS se implementa la arquitectura H-VPLS (*Hierarchical VPLS*), introduce una distinción entre los *routers* de borde de núcleo y los *routers* de borde de usuario, donde los *routers* de borde de núcleo participan en una topología *full-mesh*. Los *routers* de borde de usuario se conectan a uno o más *routers* de borde de núcleo mediante conexiones punto a punto o anillos redundates restringiendo el alcance de las tramas broadcast y multicast, reduciendo así la carga en la red.[16]

6.3.3. Integración con redes físicas y virtuales

La integración entre dominios MPLS de proveedores de servicios y redes físicas requiere una arquitectura que garantice escalabilidad y seguridad, esta interconexión se implementa

mediante mecanimos que aseguran preservar las características operativas de ambos entornos. Incuye componentes clave como; *customer edge* (CE) ubicados en las instalaciones del cliente, establecen conectividad con la red del proveedor y pueden operar en capa 2 para servicios L2VPN o en capa 3 para servicios L3VPN. Los *provider edge* (PE) son los *routers* de borde implementan la separación del tráfico, encapsulación MPLS y mantenimiento de las instancias virtuales (VSI y VRF). Además, enlaces físicos implementandos, *Attachment Circuits* (AC) que conectan con los PE.

Algunas técnicasde demarcación y separación de dominio incluyen:

- *Q-in-Q(802.1ad)*: permite la preservación de VLANs del cliente por medio de doble etiquetado separando las VLANs de servicio de las VLANs de clientes.
- *VLAN mapping*: traduce entre esquemas de VLAN del cliente y del proveedor para mantener la transparencia del servicio.
- *Service delimitation tags*: delimitan el alcance de los servicios y establecen puntos claros entre responsabilidades del cliente y del proveedor. [20]

La convergencia entre MPLS tradicionales y entornos virtualizados permite la creación de servicios hibrídos que combianan recursos locales con capacidades de *cloud*. Los modelos de integración más predominantes incluyen; la extensión de segmentos VXLAN *Virtual Extensible LAN* mediante encapsulación de paquetes en túneles punto a punto o VPN MPLS permitiendo así la interconexión entre los entornos virtualizados. Utilización de BGP EVPN como plano de control para gestionar tanto segmentos de red MPLS tradicionales basados en VXLAN por ejemplo, proporcionando un modelo operativo común. Implementación de soluciones específicas para extensión de dominios de capa 2 entre *datacenters* físicos y virtuales.

Además, se presentan soluciones de interoperabilidad con plataformas *cloud* que permiten la extensión fluida de servicios entre infraestructuras físicas tradicionales y entornos modernos virtualizados proporcionando base para arquitecturas híbridas.

Algunas de estas soluciones comprenden:

- *Cloud interconnect*: establecen conectividad entre infraestructuras MPLS corporativas y proveedores *cloud* públicos como *AWS Direct Connect, Azure ExpressRoute o Google Cloud Interconnect*.
- *Network Function Virtualization*(NFV): virtualización de funciones tradicionalmente implementadas en hardware como *firewalls*, balanceadores o *routers* sobre infraestructuras *cloud*, manteniendo conectividad con servicios MPLS. [21]

Se utilizan técnicas avanzadas de orquestación y automatización que proporcionan visibilidad unificada y control centralizado para gestionar la complejidad de estos entornos integrados. Las plataformas de orquestación multidimensional, las API abiertas, los modelos de datos normalizados y la telemetría integrada permiten la administración coherente de recursos físicos y virtuales. [22]

6.4. VPN, MPLS y BGP

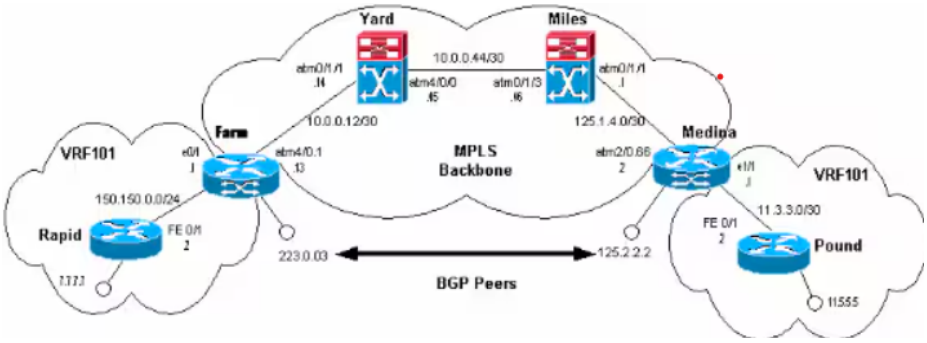
La tecnología de MPLS descrita anterioremente, es fundamental para las VPN L2 y L3, ya que proporciona aislamiento lógico y rutas predecibles para cada servicio o cliente sobre una infraestructura compartida. Su adopción failita la gestión de múltiples clientes y servicios. [1]

Multiprotocol BGP (BGP) extiende BGP para manejar múltiples tipos de direcciones como VPNv4 y VPNv6. Para la implementación de VPN MPLS L3 es fundamental ya que permite la distribución de rutas sin mezclar la información de enrutamiento por cliente garantizando así un aislamiento seguro entre ellos. [20] En una arquitectura MPLS VPN, los *routers* PE conectan la red de cliente con la red de proveedor y BGP se encarga de la distribución de rutas entre los PE sin intervención de los *routers* núcleo, de esta manera, se trabaja más eficientemente, estos *routers* solo mantienen la información de etiquetas MPLS, no las rutas completas de clientes. La separación entre el plano de control (BGP) y el plano de reenvío (MPLS) mejora la escalabilidad de la red. [9] [14]

BGP permite la interoperabilidad entre sistemas autónomos lo cual resulta útil para escenarios donde un proveedor transporte tráfico VPN de otro, por ejemplo, proveedores de cobertura global o multiservicio. [20].

Para entender cómo funciona una VPN MPLS, podemos observar el siguiente ejemplo de configuración:

Figura 1. Ejemplo de arquitectura MPLS VPN con BGP para intercambio de rutas.



Nota. Adaptada de [23]. El esquemamuestra cómo los PE utilizan VRF y BGP para distribuir rutas entre sitios de cliente, manteniendo aislamiento y escalabilidad en la red MPLS.

La imagen muestra un ejemplo de flujo de paquetes MPLS a través de una red basada en ATM, mostrando cómo los *routers* de borde del proveedor (PE) encapsulan los paquetes y los envían a través de una infraestructura modificada. El escenario destaca cómo se asignan las etiquetas mediante LDP, se establecen las LSP y se realiza la conmutación basada en etiquetas dentro de la red del proveedor. [23]

6.4.1. Interconexión de sitios remotos con *routers* PE y CE

La interconexión entre sitios remotos mediante *routers* PE (*provider edge*) y CE (*customer edge*) son un fundamento estructural de las implementaciones VPN sobre MPLS ya que define tanto los componentes físicos como procesos lógicos.

La arquitectura incluye elementos fundamentaes: *router CE (customer edge)*, *router PE (provider edge)*, Red de núcleo MPLS (P *routers*) y Circuito de acceso (AC - Attachment Circuit). Estos elementos implementan una clara separación donde el proveedor gestiona la infraestructura de transporte y los mecanismos de separación de tráfico mientras el cliente mantiene control de sus políticas y seguridad interna. [24]

El intercambio de información de enrutamiento entre *routers* PE y CE representa un componente crítico que determina tanto la operatividad como las capacidades de servicio. Puede realizarse por medio de enrutamiento estático al configurar manualmente rutas en los extremos del enlace, por medio de BGP PE-CE que utiliza BGP para intercambiar información de enrutamiento ente PE y CE y protocolos de enrutamiento dinámico IGP implementando protocolos como OSPF, IS-IS o EIGRP entre PE y CE. [25]

6.4.2. Intercambio de rutas mediante *multiprotocol* BGP

Constituye en una extensión del protocolo BGP-4 que permite el transporte de información de enrutamiento para múltiples direcciones, familias de direcciones y subfamilias de información posterior, fundamental para arquitecturas MPLS VPN para transportar rutas para diferentes protocolos de manera simultánea. [25]

BGP constitye atributos escenciales para anunciar rutas alcanzables y retirar rutas no alcanzables permitiendo así la transmición de información para múltiples protocolos capa 3, incluyendo IPv4, VPN-IPv4. [9] BGP funciona como un protocolo fundamental de enrutamiento VPN entre *routers* PE y PE, cada PE mantiene múltiples tablas de enrutamiento VRF y utiliza BGP para transportar estas rutas con los atributos respectivos entre distintos sitios de la VPN. [24]

6.4.3. Uso de *route distinguisher* (RD) y *route target* (RT)

En redes MPLS VPN, múltiples clientes llegan a usar el mismo espacio de direccionamiento IP privado (por ejemplo, 192.168.0.0/16) y para evitar colisiones se introduce el *Route Distinguisher* (RD) para permitir que los proveedores se distingan entre rutas de diferentes clientes incluso si estas usan direcciones idénticas. [25] Por su parte, el RD es un prefijo de 8 bytes que se añade a la dirección IP de la ruta en la tabla BGP-VPN, creando un VPNv4 o VPNv6 *address*. No se utiliza para el reenvío de paquetes pero es escencial en el plano de control para que cada ruta sea única globalmente. [19]

A diferencia del RD, el *Route Target* (RT) se utiliza dentro del plano de control definiendo qué rutas deben ser importadas o exportadas por cada VPN. Permite construir topologías complejas como la mencionada *full-mesh* facilitando la escalabilidad en la gestión de múltiples VPN sobre una MPLS. [24]

En las VPN L3 sobre MPLS, los RD se asignan por instancias de cliente (VRF) y los RT permiten que las rutas sean compartidas con otros sitios según la política establecida. En las L2VPN no requieren VRF pero los RT son críticos para que las instancias de servicio sepan con qué otras deben intercambiar tráfico. Al permitir múltiples VRF con independencia de direcciones IP, los RD habilitan entornos escalables y los RT brindan mecanismos de control para definir explícitamente rutas en los límites de VPN. [8]

6.5. Entorno de simulación de redes: GNS3

GNS3 (*Graphical Network Simulator-3*) es una plataforma de código abierto que permite crear entornos de red realistas mediante emulación y simulación. Utiliza imágenes reales de sistemas operativos de red como Cisco IOS, JunOs, entre otros. GNS3 emula el comportamiento funcional exacto de *routers* y *switches* permitiendo la réplica del funcionamiento de redes complejas incluyendo configuraciones avanzadas como MPLS, VPN, protocolos de enrutamiento dinámico y políticas de QoS. Una de las ventajas más significativas de GNS3 incluyen la arquitectura cliente-servidor que permite separar la interfaz gráfica del proceso de emulación. El componente gráfico puede ejecutarse en una máquina local mientras que el backend puede ejecutarse en máquinas virtuales o contenedores permitiendo mayor escalabilidad y distribución de carga. [3]

GNS3 ha denostrado ser eficaz en la emulación de escenarios complejos simulando arquitecturas típicas de proveedores de servicios. Además, permite configurar fallos de enlaces y validación de políticas de red lo cual es escencial para analizar rútas óptimas y desempeño bajo diferentes condiciones de tráfico. [2]

6.6. Ingeniería de tráfico (TE)

La Ingeniería de Tráfico permite controlar de manera eficiente el flujo de tráfico en redes modernas, especialmente en arquitecturas basadas en MPLS. Optimiza la utilización de recursos de red y cumple con los requisitos de calidad de servicio (QoS). Una sus capacidades clave incluye la definición de rutas explícitas las cuales permiten seleccionar de forma de mejor manera, el flujo que llevarán los paquetes a través de la red, facilitando así la distribución uniforme de tráfico entre múltiples rutas disponibles. [26] Puede tomar decisiones basadas en métricas más complejas.

Permite reservar y asignar ancho de banda de manera eficiente según las necesidades específicas de cada flujo. En redes MPLS, esto se logra estableciendo LSP inspirados en enjambres o algoritmos genéticos, implementa mecanismos de control de congestión mediante métricas que identifican cuándo un enlace se encuentra saturado. Empleando modelos que permiten la clasificación de tráfico en base a políticas definidas, garantizando un tratamiento preferencial de ciertos servicios sin degradar el rendimiento general de la red. [27] Al definir rutas que evitan enlaces congestionados y optimizar el uso del ancho de banda, se contribuye directamente a reducir la latencia de extremo a extremo y la pérdida de paquetes. [28] En entornos de red con alta demanda o los que soportan VPN, la implementación de TE brinda una mejora significativa. [15]

6.7. Monitoreo de red y análisis de tráfico

Son procesos fundamentales para garantizar el funcionamiento eficiente, seguro y continuo de una red MPLS. Observan el estado de los dispositivos y enlaces en tiempo real para anticipar fallos, detectar cuellos de botella, y tomar decisiones informadas. La captura de tráfico en tiempo real monitorea los paquetes que circulan por una red inspeccionando su contenido, comportamiento, y patrones. Permite el diagnóstico activo de problemas como pérdida de paquetes, *jitter*, o latencias elevadas. Herramientas como Wireshark, tcpdump o tshark permiten la observación directa de flujos de datos, mostrando información detallada a nivel de encabezados de capa 2 a capa 7. [27]

En redes MPLS, estas capturas permiten además visualizar etiquetas asignadas a los paquetes, detectar errores de encapsulación o loops de reenvío, y validar políticas de calidad de servicio (QoS). Cuando se recompila la información del tráfico, el análisis de comportamiento permite evaluar indicadores clave de desempeño realizar este análisis en tiempo real o de forma histórica, ayuda a establecer tendencias, identificar enlaces sobrecargados o inestables y tomar decisiones de ingeniería de tráfico. [3]

Para una gestión eficiente, el monitoreo debe estar vinculado con sistemas de respuesta automatizada. Esto se logra mediante plataformas de gestión de red (NMS) o soluciones de monitoreo como Zabbix, Nagios, Prometheus o PRTG, que generan alertas en base a eventos específicos permitiendo que el monitoreo no solo sea reactivo, sino predictivo y adaptativo. [7]

6.8. *Machine learning* en redes

El uso de *machine learning* (ML) transforma la forma en que se realiza el monitoreo, análisis, predicción y optimización del tráfico en una red, permitiendo desarrollar sistemas proactivos y adaptativos que identifican patrones en el tráfico, anticipando anomalías y tomando decisiones automatizadas en tiempo real. Puede utilizar datos tanto históricos como actuales de la red para entrenar modelos y lograr modelos con una mejor comprensión de la evolución de la red, lo que mejora la precisión de las predicciones y permite decisiones basadas en contexto. En redes, el aprendizaje supervisado es aquel que se aplica a conjuntos de datos etiquetados que relacionan métricas observadas con comportamientos esperados. Por ejemplo, modelos de regresión lineal, regresión logística o categorización de eventos como "normal.º .ªnómalo". Algoritmos como árboles de decisión, redes neuronales multicapa o Naïve Bayes son utilizados para clasificar eventos de red en tiempo. [3]

Cuando los datos no están etiquetados, el aprendizaje no supervisado se convierte en una herramienta poderosa para descubrir estructuras internas y relaciones latentes en el tráfico, por medio de *clustering* o algoritmos como PCA que permiten visualizar y analizar grandes volúmenes de tráfico. [27]

Una de las aplicaciones más fuertes del ML en redes es la identificación de patrones de tráfico recurrentes que detectan comportamientos inusuales que llevarían a fallos, ataques o congestiones inminentes y emitir alertas cuando se detectan desviaciones significativas. El sistema puede disparar el recalculo de rutas o el failover sin intervención humana. [7]

6.9. Optimización basada en IA

La optimización basada en Inteligencia Artificial (IA) mejora la eficiencia operativa de las redes al automatizar procesos de decisión complejos que se adapten dinámicamente a las condiciones del tráfico, en redes MPLS la IA se aplica especialmente a la Ingeniería de Tráfico para optimizar la selección de rutas, distribuir la carga y mejorar el uso de los recursos, sin comprometer la calidad de servicio (QoS).

Algoritmos inspirados en la naturaleza y las redes neuronales tienen un potencial alto en problemas de ruteo y asignación de recursos, por ejemplo, ACO (*Ant Colony Optimization*), algoritmos genéticos y redes neuronales artificiales (ANN). [3]

Una característica de la optimización basada en IA es su capacidad de trabajar con funciones objetivo, una función objetivo minimiza el uso de recursos, la congestión y el costo total de los enlaces en redes MPLS siendo superior frente a modelos estándar. [27] La capacidad de adaptación en tiempo real es una ventaja del uso de IA, los algoritmos pueden incorporar mecanismos de actualización iterativa que evalúan el estado actual de la red y ajustan sus decisiones en consecuencia. [3]

En redes altamente complejas, se permite aprovechar la redundancia de MPLS para canalizar flujos sin afectar el rendimiento, con modelos optimizados por IA existe una mejora significativa de la tasa de pérdida de paquetes y el retardo. [26]

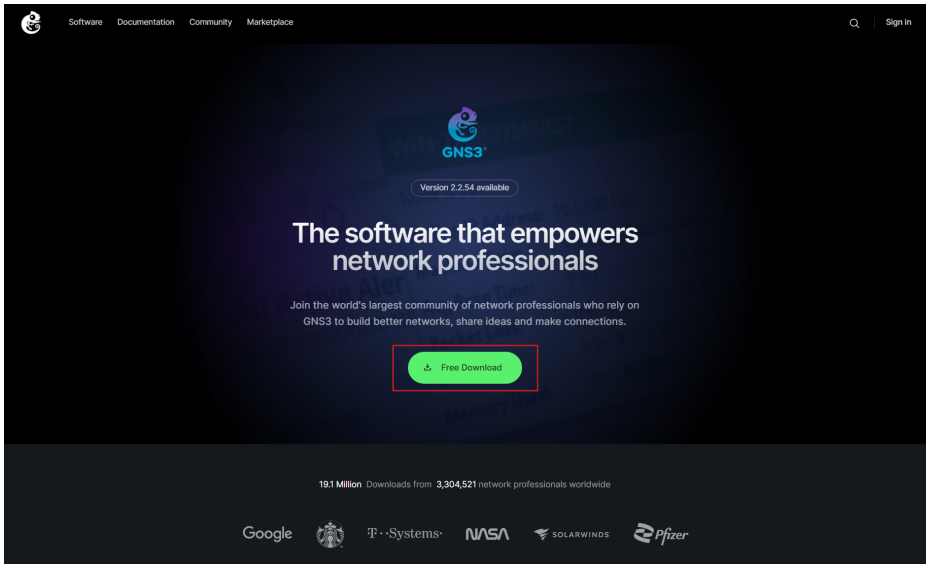
CAPÍTULO 7

Entorno en el emulador GNS3

7.1. Instalación de GNS3

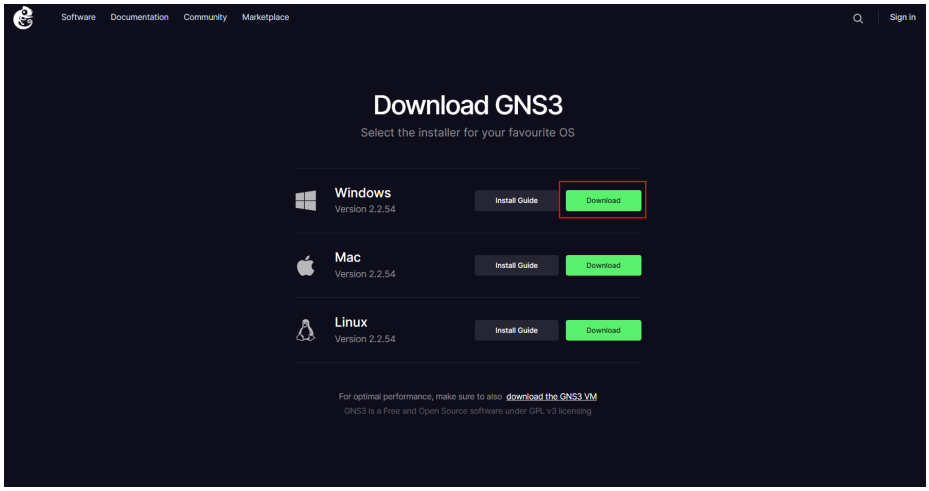
La instalación de GNS3 se realiza a través del sitio oficial del, disponible en: https://gns3.com/software/download. Desde esta página se puede descargar el instalador correspondiente al sistema operativo Windows 10 o 11. Se recomienda crear una cuenta gratuita en la plataforma de GNS3 para acceder a las últimas versiones.

Figura 2. Página principal de GNS3 para descarga del software



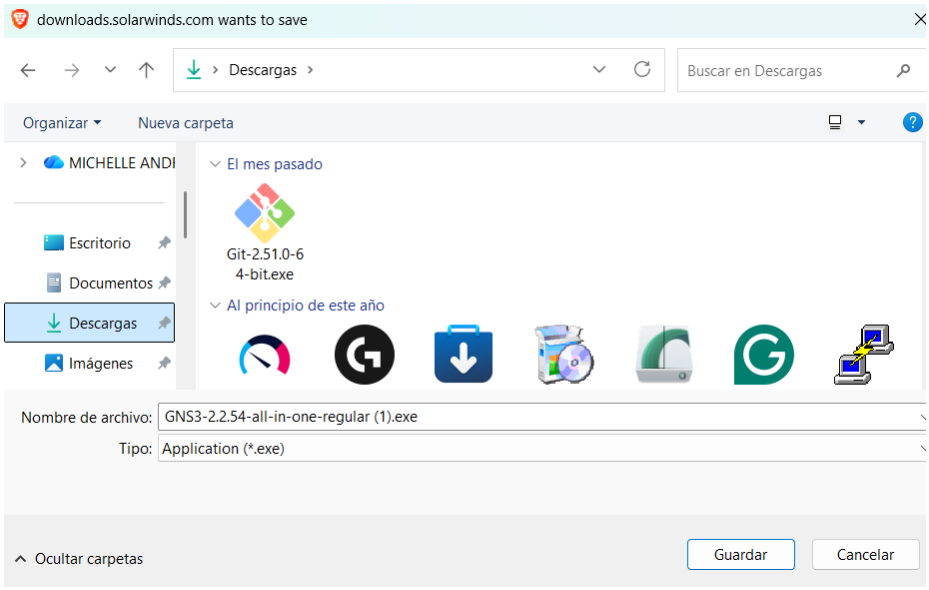
Nota. Elaboración propia. Descarga del software GNS3 desde su sitio oficial.

Figura 3. Descarga de GNS3 para Windows



Nota. Elaboración propia. Selección de la versión de GNS3 para Windows.

Figura 4. Descarga de GNS3 para Windows en un directorio específico

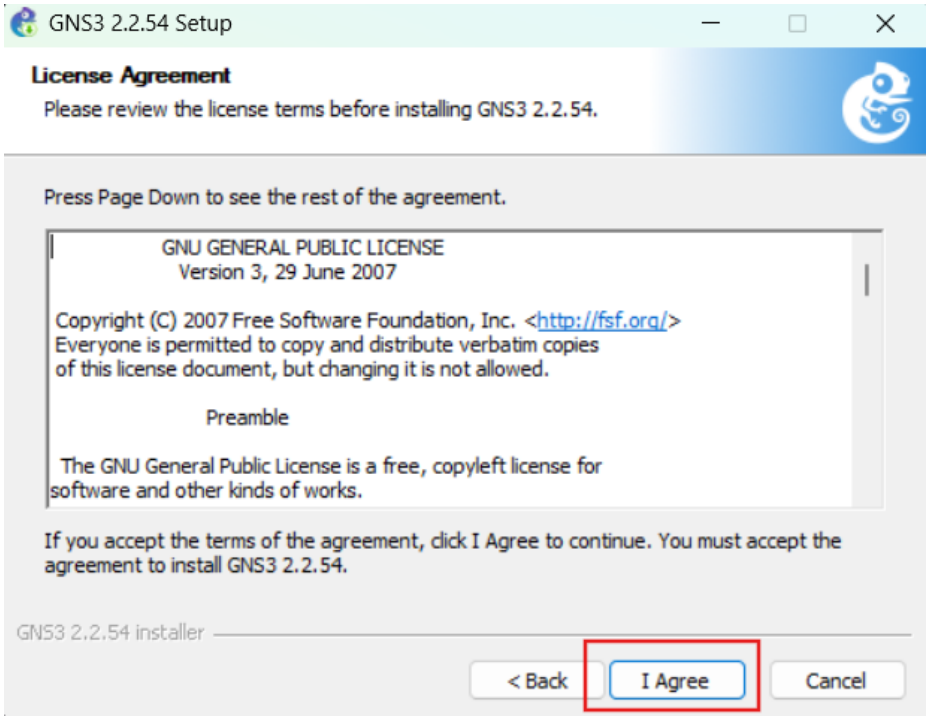


Nota. Elaboración propia. Descarga del instalador de GNS3 en el equipo local.

Figura 5. Pasos de instalación de GNS3 en Windows

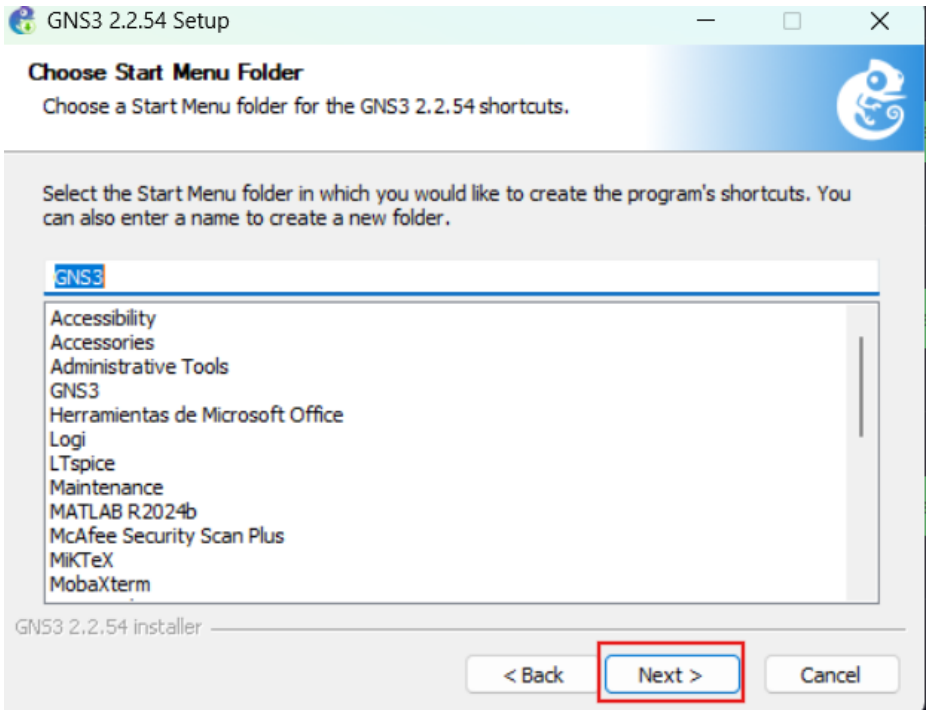
Nota. Elaboración propia. Proceso de instalación del software GNS3 en Windows.

Figura 6. Términos y condiciones de GNS3 en Windows



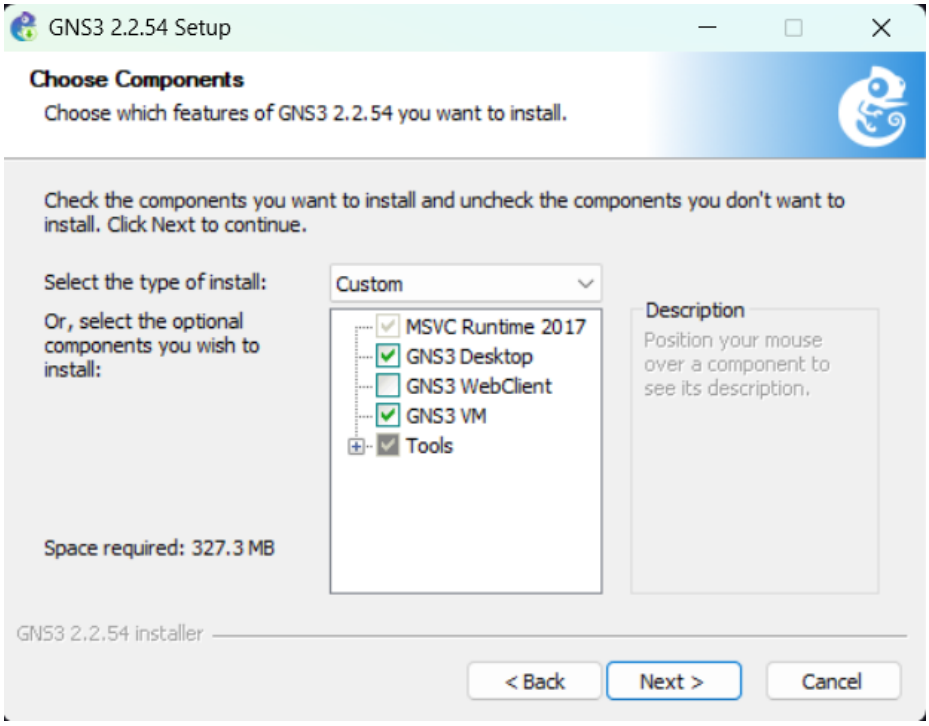
Nota. Elaboración propia. Aceptación de términos y condiciones para la instalación de GNS3.

Figura 7. Selección de fólder de instalación de GNS3 en Windows



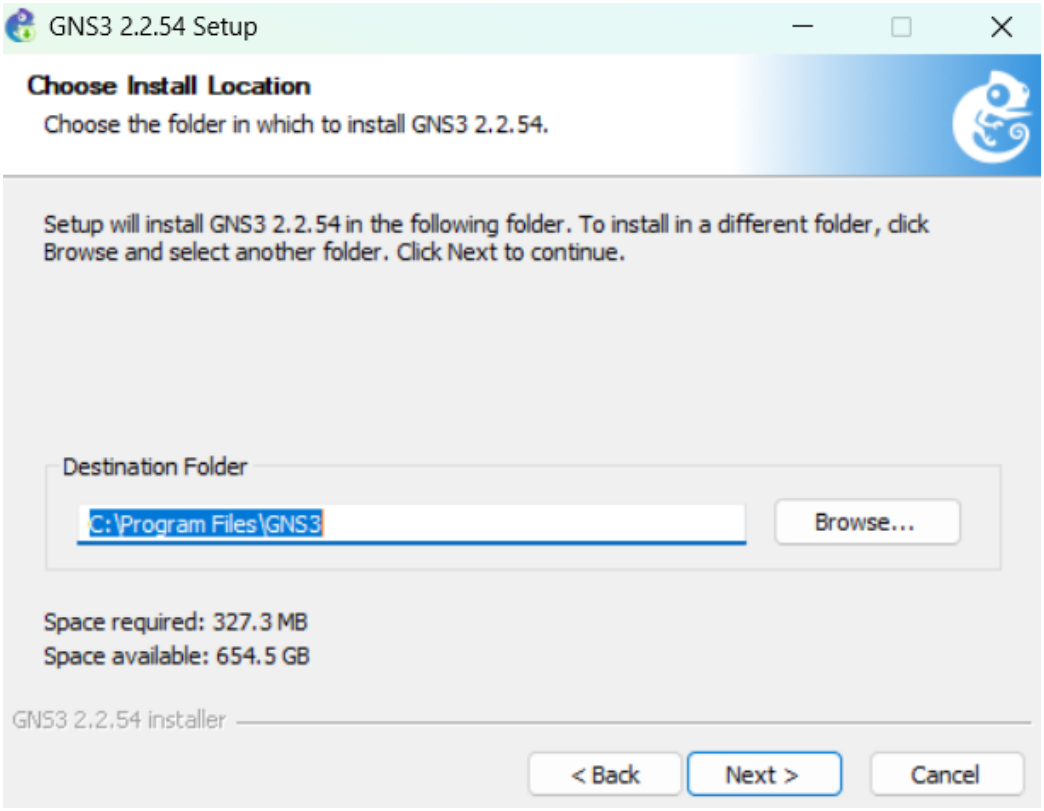
Nota. Elaboración propia. Selección del directorio de instalación de GNS3 en el equipo local.

Figura 8. Selección de componentes a instalar en GNS3 para Windows



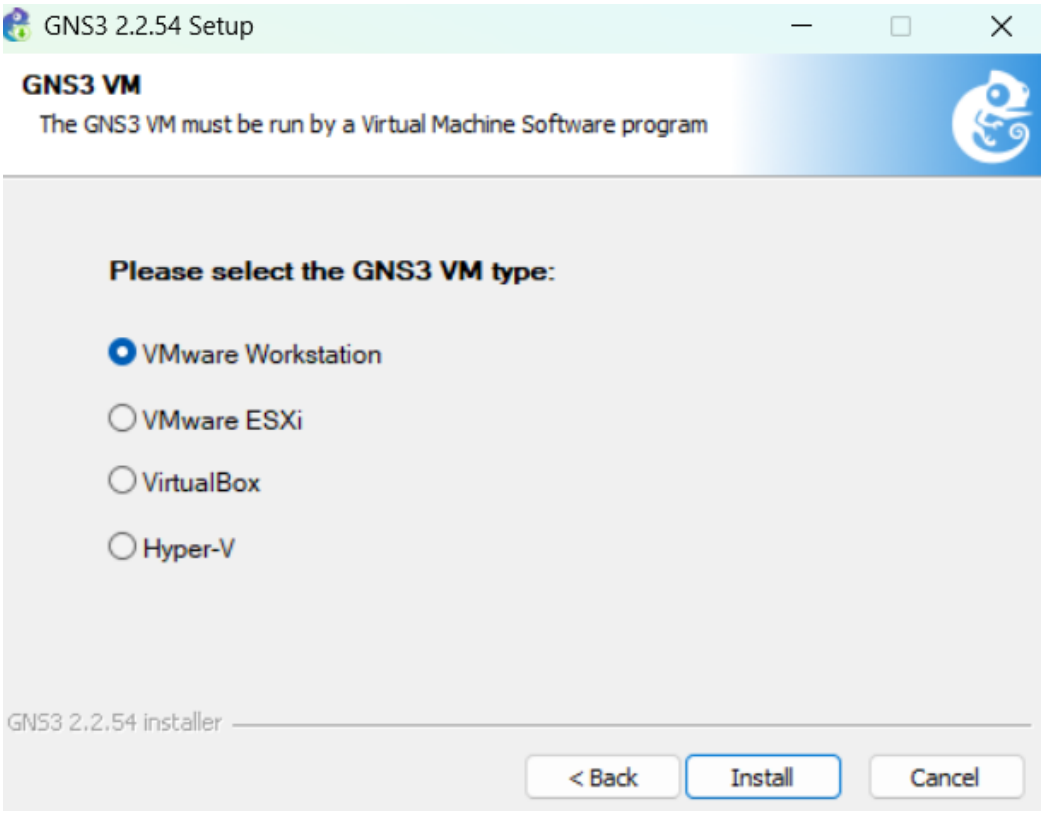
Nota. Elaboración propia. Selección de componentes adicionales para la instalación de GNS3.

Figura 9. Ubicación para la instalación de GNS3 en Windows



Nota. Elaboración propia. Selección del directorio para la instalación de GNS3.

Figura 10. Selección de VMware Workstation para ejecución de GNS3 en Windows

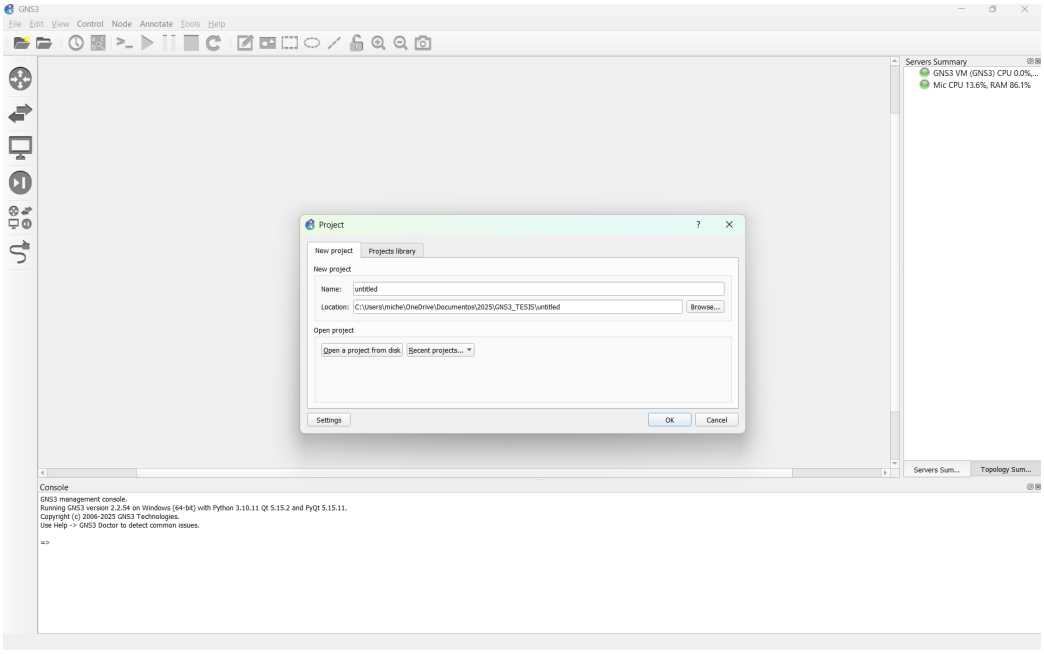


Nota. Elaboración propia. Selección de la opción para utilizar VMware Workstation como motor de virtualización en GNS3.

Figura 11. Selección de VMware Workstation para ejecución de GNS3 en Windows

Nota. Elaboración propia. Selección de la opción para utilizar VMware Workstation como motor de virtualización en GNS3.

Figura 12. GNS3 en Windows sobre VMware Workstation

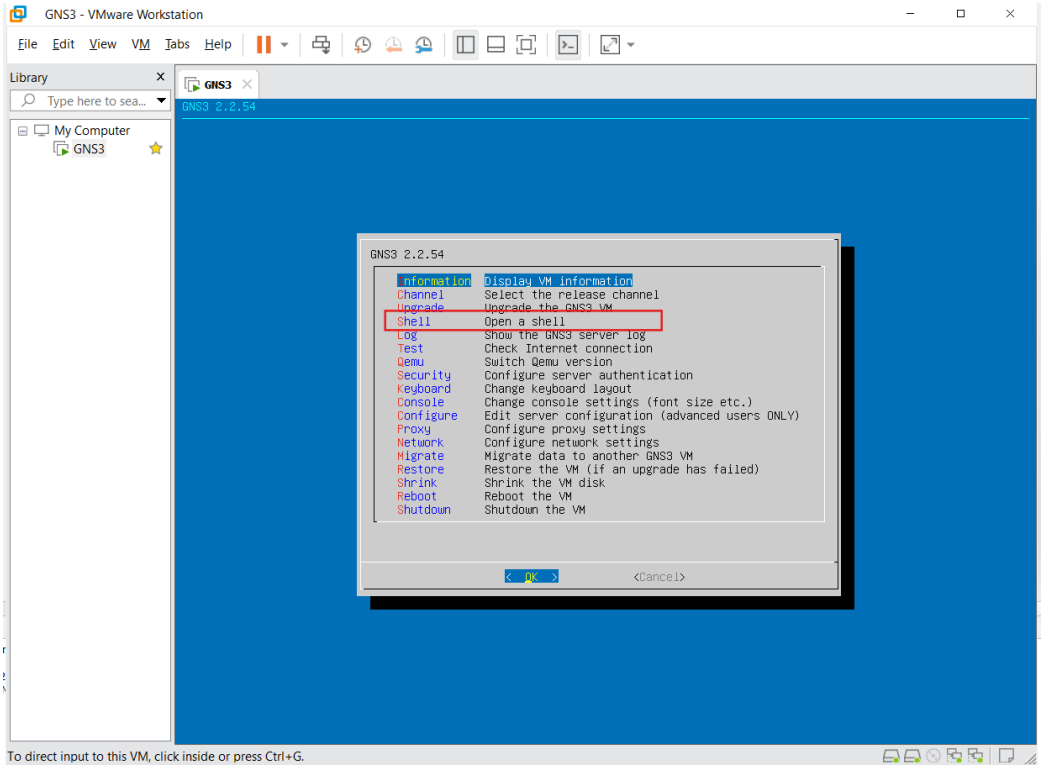


Nota. Elaboración propia. GNS3 en Windows utilizando VMware Workstation como motor de virtualización.

7.2. Instalación de imágenes IOU Cisco

Para agregar imágenes IOU en GNS3, se utiliza la guía adjunta en 33. Accediendo a la terminal de GNS 3 VM, es necesario actualizar los repositorios e instalar las dependencias necesarias para ejecutar imágenes IOU.

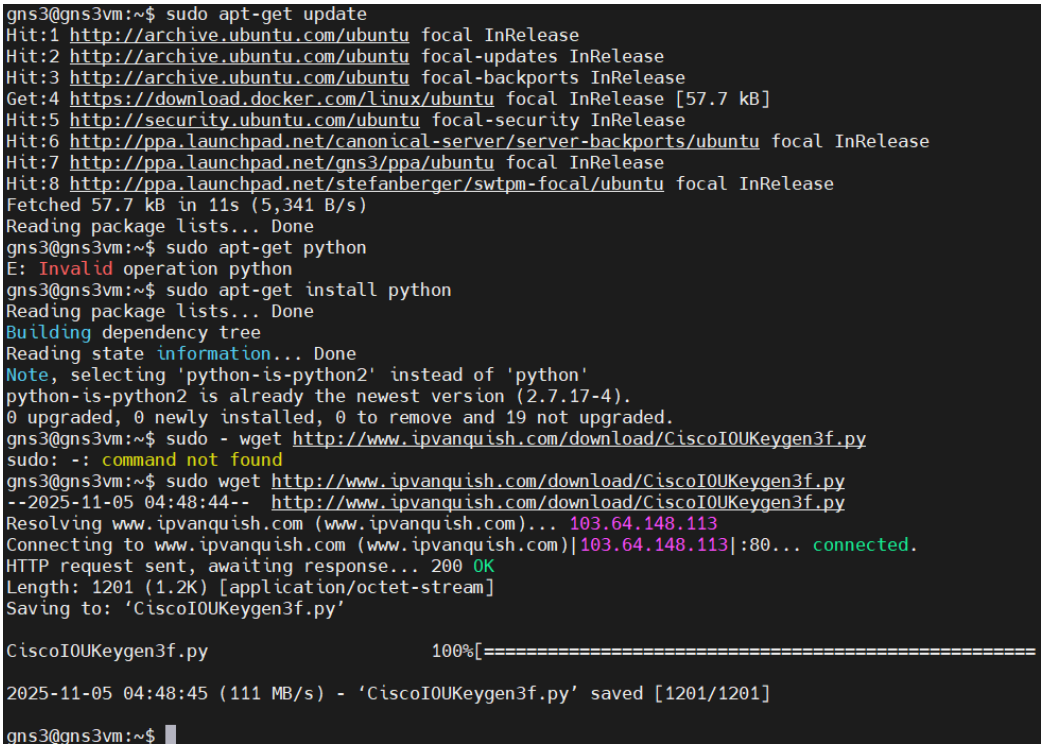
Figura 13. Acceso a la terminal de GNS3 VM



Nota. Elaboración propia. Acceso a la terminal de GNS3 VM para actualización e instalación de repositorios y paquetes.

Se actualizaron los índices de los repositorios del sistema utilizando apt-get update, para una disponibilidad de las versiones más recientes de los paquetes. Se procedió a instalar Python mediante el comando apt-get install python, y la herramienta *keygen* correspondiente empleando wget desde la dirección http://www.ipvanquish.com/download/CiscoIOUKeygen3f.py.

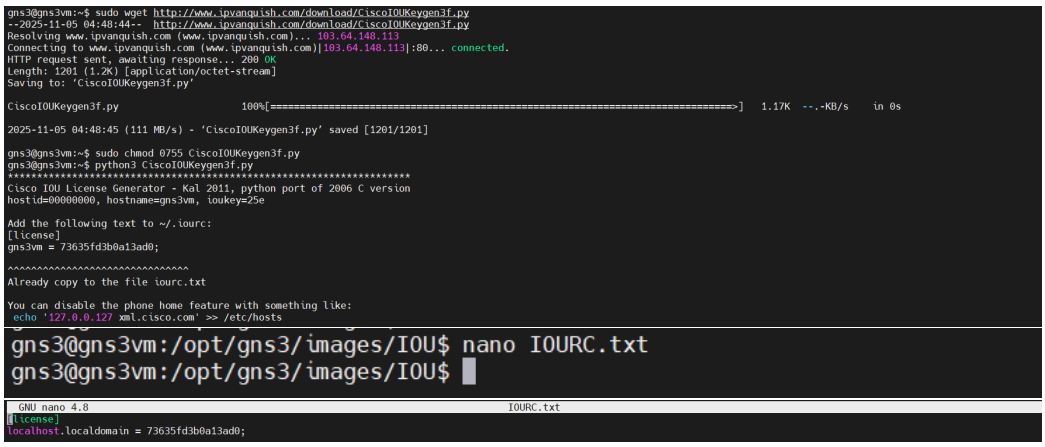
Figura 14. Actualización de repositorios en GNS3 VM



Nota. Elaboración propia. Actualización de los repositorios en GNS3 VM utilizando. Instalación de python y *keygen*.

Se asignaron los permisos de ejecución al archivo correspondiente mediante el comando chmod 0755 CiscoIOUKeygen3f.py para permitir su ejecución, anotando la clave de licencia, se accedió al directorio de imágenes de GNS3 con cd /opt/GNS3/images/IOU, se creó el archivo de configuración IOURC.txt y se pegó la clave de licencia en el siguiente formato: [license] localhost.localdomain = 73635fd3b0a13ad0;

Figura 15. Configuración de la licencia IOU en GNS3 VM

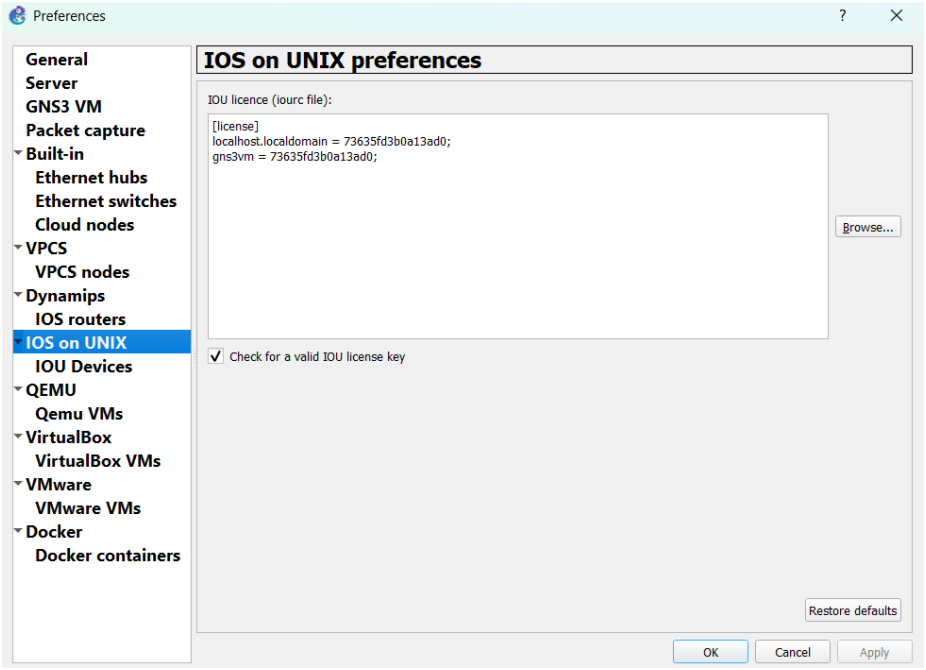


Nota. Elaboración propia. Configuración de la licencia IOU en GNS3 VM mediante la creación del archivo IOURC.txt con la clave generada.

7.3. Configuración de imágenes IOU Cisco

En la interfaz de GNS3 se accedió a Preferencias → IOU UNIX y se especificó la clave proporcionada, respetando el formato definido en el archivo IOURC.txt.

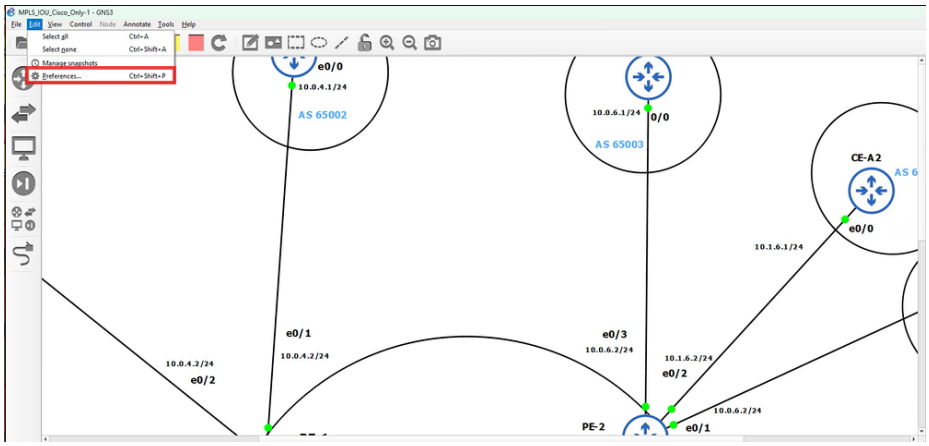
Figura 16. Configuración de la licencia IOU en GNS3



Nota. Elaboración propia. Configuración de la licencia IOU en GNS3 mediante la clave generada.

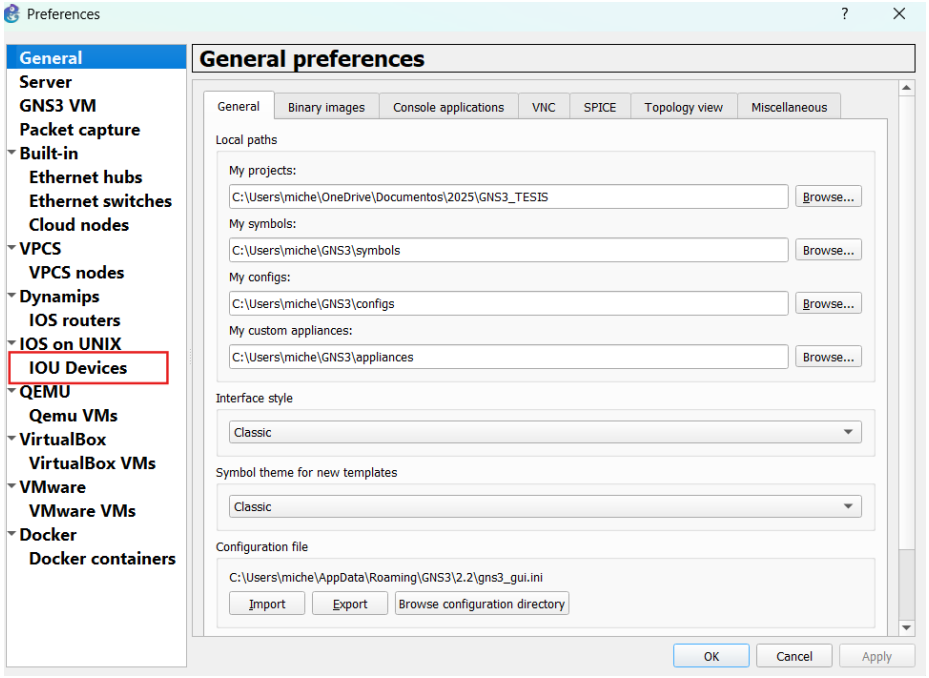
En la interfaz de GNS3, se añadió la imagen IOU L3 para routers Cisco, especificando la configuración de red de la imagen.

Figura 17. Sección de preferencias IOU en GNS3



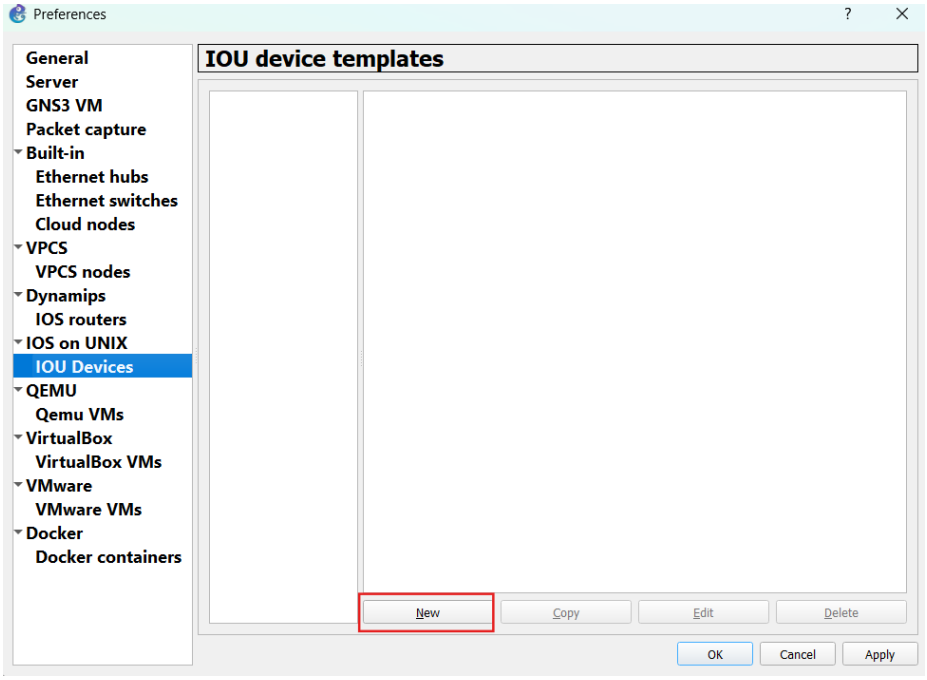
Nota. Elaboración propia. Referencia a las preferencias en GNS3.

Figura 18. Sección de imágenes IOU en GNS3



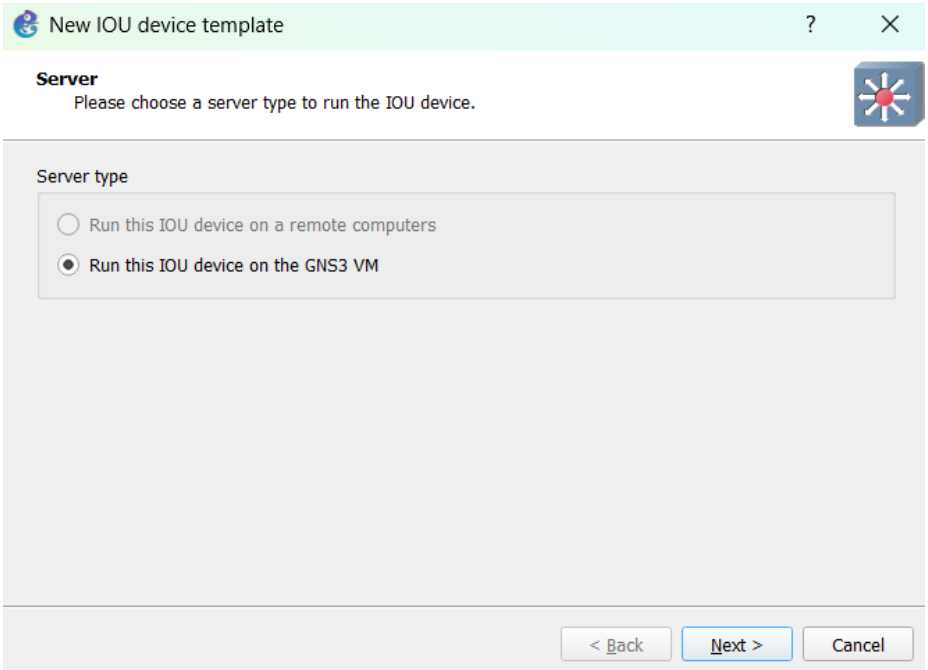
Nota. Elaboración propia. Referencia a la configuración de imágenes IOU en GNS3.

Figura 19. Adición de imagen IOU L3 en GNS3



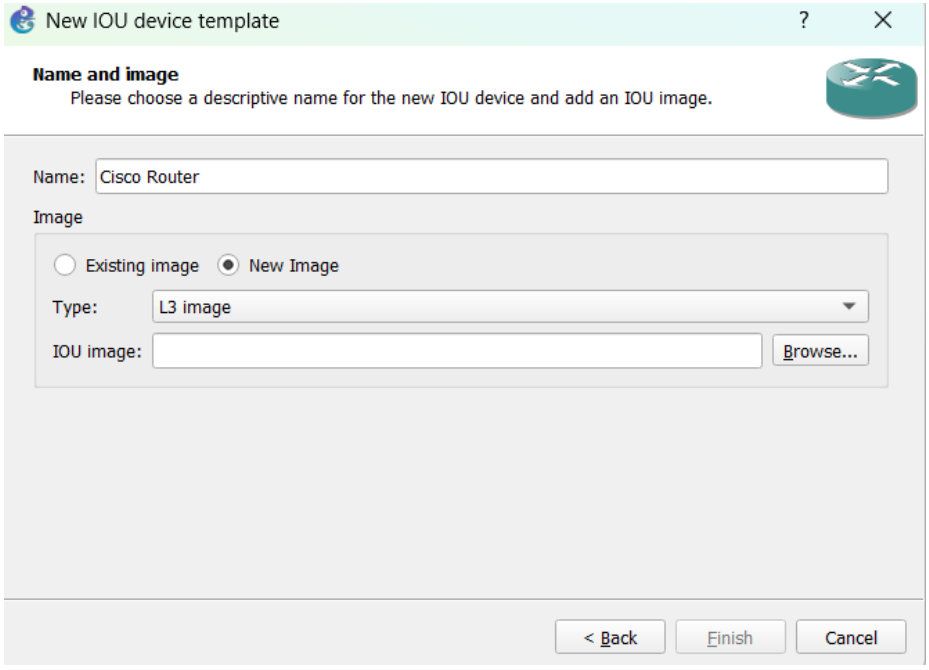
Nota. Elaboración propia. Creación de una nueva imagen en la sección de plantillas de imágenes IOU en GNS3

Figura 20. *Server type* para imagen IOU L3 en GNS3



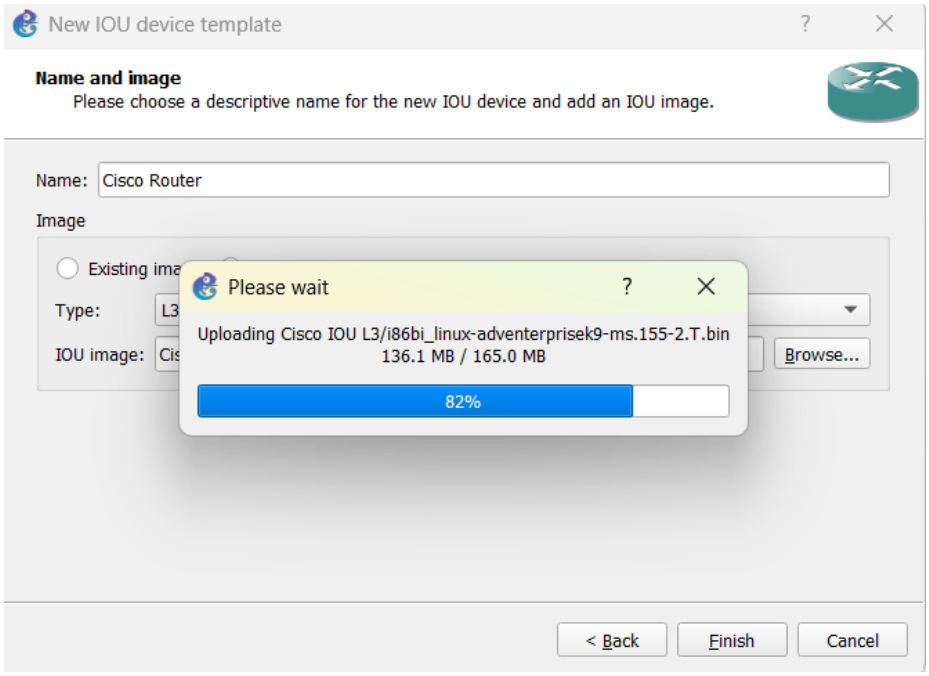
Nota. Elaboración propia. El servidor seleccionado es GNS3 VM para ejecutar la imagen IOU L3 para aprovechamiento de recursos.

Figura 21. Nombre e imagen IOU L3 en GNS3



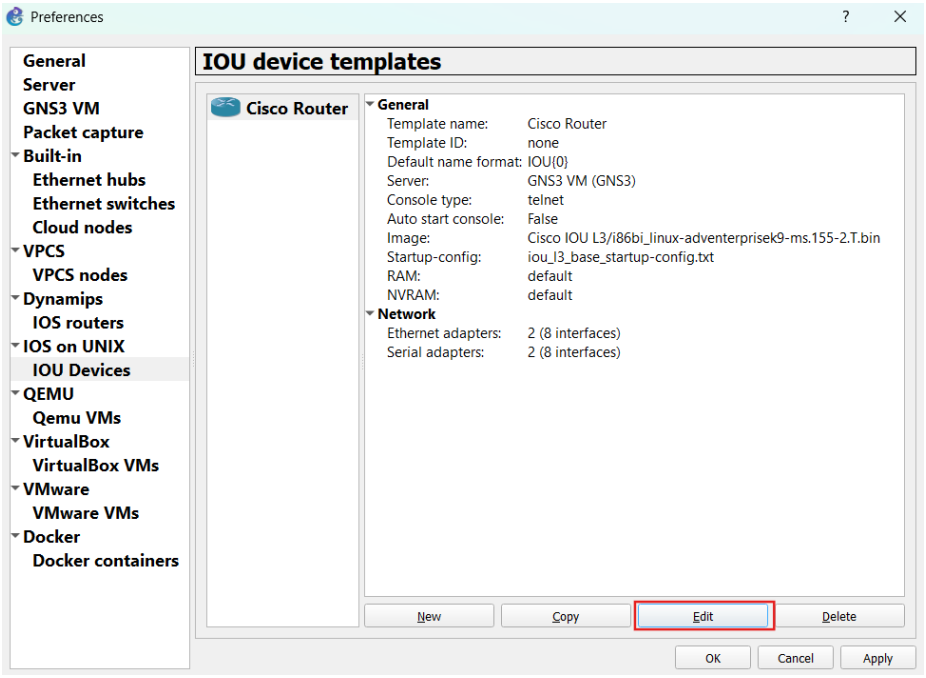
Nota. Elaboración propia. Carga y reconocimiento de imagen Cisco IOU en el servidor GNS3.

Figura 22. Carga de imagen IOU L3 en GNS3



Nota. Elaboración propia. Carga de la imagen IOU L3 en formato .bin para su uso en GNS3.

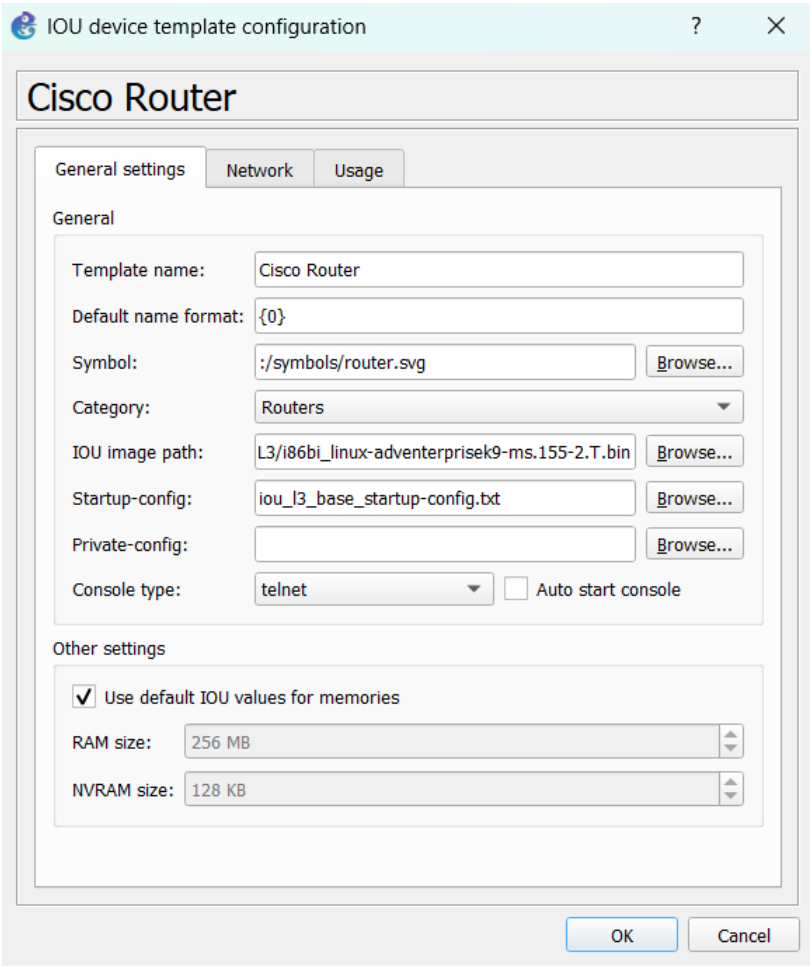
Figura 23. Configuración de la imagen IOU L3 en GNS3



Nota. Elaboración propia. Referencia a la configuración de parámetros dentro de una imagen IOU L3 en GNS3

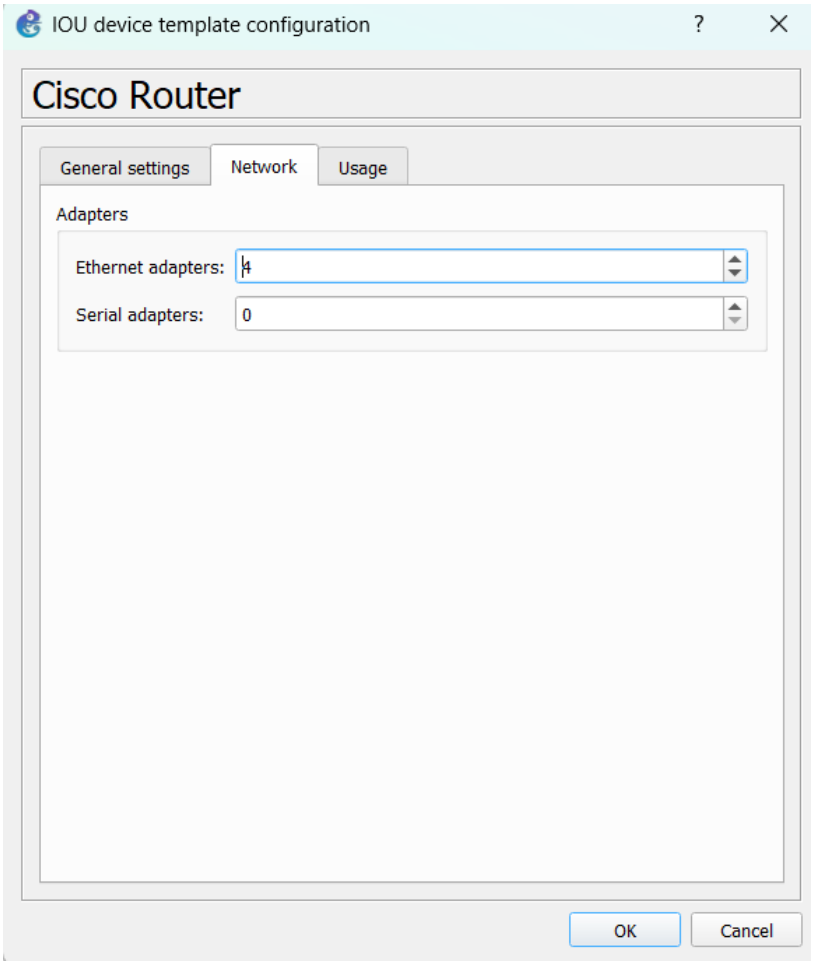
Se establecieron los parámetros básicos de configuración y se mantuvieron los valores predeterminados para las especificaciones de memoria. Asimismo, se configuraron al menos cuatro adaptadores de red para garantizar la conectividad requerida en el entorno de virtualización.

Figura 24. Configuración general de la imagen IOU L3 en GNS3



Nota. Elaboración propia. Se configuran parámetros básicos para la imagen tales como, nombre, símbolo, categoría.

Figura 25. Configuración de red de la imagen IOU L3 en GNS3



Nota. Elaboración propia. Se asignaron al menos 4 adaptadores ethernet para la interconectividad del entorno de virtualización.

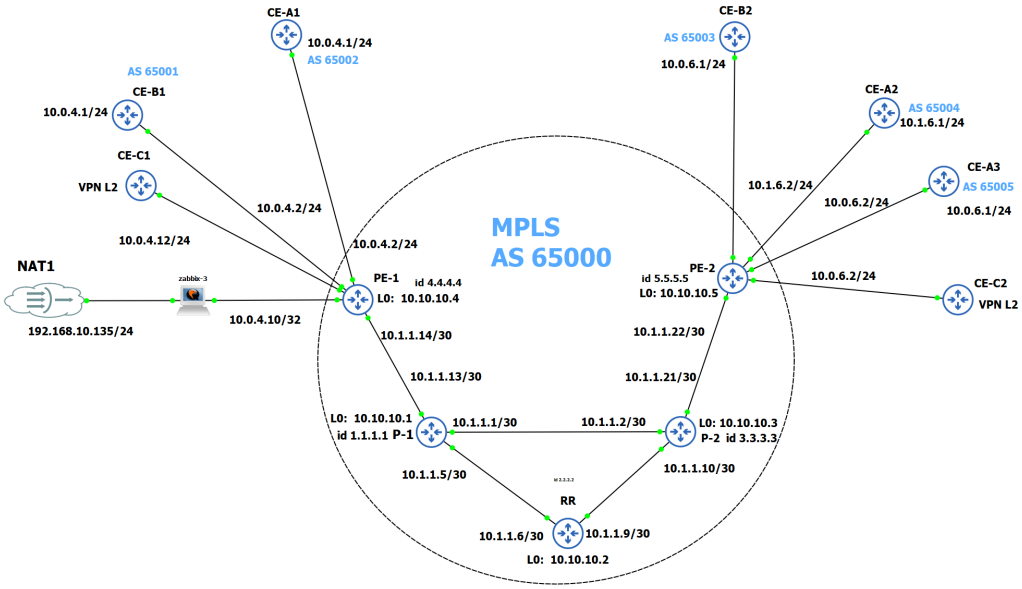
Si las condiciones de la topología lo demandan, es posible ampliar la cantidad de adaptadores de red durante la fase de implementación, accediendo a las preferencias de GNS3 y actualizando los parámetros correspondientes de la imagen IOU.

CAPÍTULO 8

Implementación de la red MPLS

8.1. Topología

Figura 26. Topología MPLS [29] implementada en GNS3 con routers de núcleo y borde



Nota. Elaboración propia. La topología representa el backbone MPLS (AS 65000) con routers PE, P y CE, mostrando la interconexión necesaria para soportar VPNs y monitoreo en el entorno simulado.

8.2. Configuración de equipos

Código 1. Configuración del route reflector (RR)

```
hostname RR
!
interface Loopback0
 ip address 10.10.10.2 255.255.255.255
 ip ospf 1 area 0
!
interface Ethernet0/0
 ip address 10.1.1.6 255.255.255.252
 ip ospf 1 area 0
 mpls ip
 service-policy output LIMIT
 hold-queue 1 out
!
interface Ethernet0/1
 ip address 10.1.1.9 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
router ospf 1
 router-id 2.2.2.2
 network 10.10.10.2 0.0.0.0 area 0
!
router bgp 65000
 bgp log-neighbor-changes
 neighbor 10.10.10.4 remote-as 65000
 neighbor 10.10.10.4 update-source Loopback0
 neighbor 10.10.10.5 remote-as 65000
 neighbor 10.10.10.5 update-source Loopback0
 !
 address-family ipv4
  neighbor 10.10.10.4 activate
  neighbor 10.10.10.4 send-community both
  neighbor 10.10.10.4 route-reflector-client
  neighbor 10.10.10.5 activate
 exit-address-family
 !
 address-family vpnv4
  neighbor 10.10.10.4 activate
  neighbor 10.10.10.4 send-community both
  neighbor 10.10.10.4 route-reflector-client
  neighbor 10.10.10.5 activate
  neighbor 10.10.10.5 send-community both
  neighbor 10.10.10.5 route-reflector-client
 exit-address-family
!
snmp-server community ZABBIX RO acl_snmp
!
mpls ldp router-id Loopback0 force
!
end
```

Nota. Elaboración propia. Nota. Elaboración propia.

Código 2. Configuración del router P1

```
hostname P-1
!
interface Loopback0
 ip address 10.10.10.1 255.255.255.255
 ip ospf 1 area 0
!
interface Ethernet0/0
 ip address 10.1.1.13 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
interface Ethernet0/1
 ip address 10.1.1.5 255.255.255.252
 ip ospf 1 area 0
 ip ospf cost 10
 mpls ip
!
interface Ethernet0/2
 ip address 10.1.1.1 255.255.255.252
 ip ospf 1 area 0
 ip ospf cost 10
 mpls ip
!
router ospf 1
router-id 1.1.1.1
!
ip access-list extended acl_snmp
!
snmp-server community ZABBIX RO
!
control-plane
!
line con 0
 logging synchronous
line aux 0
line vty 0 4
 login local
 transport input ssh
!
!
end
```

Nota. Elaboración propia.

Código 3. Configuración del router P2

```
hostname P-2
!
interface Loopback0
 ip address 10.10.10.3 255.255.255.255
 ip ospf 1 area 0
!
interface Ethernet0/0
 ip address 10.1.1.10 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
interface Ethernet0/1
 ip address 10.1.1.2 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
interface Ethernet0/2
 ip address 10.1.1.21 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
router ospf 1
 router-id 3.3.3.3
 network 10.10.10.3 0.0.0.0 area 0
!
end
```

Nota. Elaboración propia.

Código 4. Configuración del router PE1

```
hostname PE-1
!
vrf definition Client_A
 rd 100:110
 route-target export 100:1000
 route-target import 100:1000
 !
 address-family ipv4
 exit-address-family
!
vrf definition Client_B
 rd 100:120
 route-target export 100:2000
 route-target import 100:2000
 !
 address-family ipv4
 exit-address-family
!
interface Loopback0
 ip address 10.10.10.4 255.255.255.255
 ip ospf 1 area 0
!
```

```
interface Ethernet0/0
 ip address 10.1.1.14 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
interface Ethernet0/1
 vrf forwarding Client_A
 ip address 10.0.4.2 255.255.255.0
!
interface Ethernet0/1.100
 encapsulation dot1Q 100
 ip address 172.16.100.5 255.255.255.252
 ip ospf 1 area 0
!
interface Ethernet0/2
 de\textit{script}ion zabbix
 vrf forwarding Client_B
 ip address 10.0.4.2 255.255.255.0
!
interface Ethernet0/2.100
 encapsulation dot1Q 100
 ip address 172.16.100.1 255.255.255.252
 ip ospf 1 area 0
!
interface Ethernet0/3
 ip address 10.0.4.10 255.255.255.0
 ip ospf 1 area 0
!
interface Ethernet1/0
 de\textit{script}ion CE-C1 (VPWS)
 no ip address
 xconnect 10.10.10.5 100 encapsulation mpls
!
router ospf 1
 router-id 4.4.4.4
!
router bgp 65000
 bgp log-neighbor-changes
 neighbor 10.10.10.2 remote-as 65000
 neighbor 10.10.10.2 update-source Loopback0
 !
 address-family vpnv4
  neighbor 10.10.10.2 activate
  neighbor 10.10.10.2 send-community both
 exit-address-family
 !
 address-family ipv4 vrf Client_A
  neighbor 10.0.4.1 remote-as 65002
  neighbor 10.0.4.1 activate
 exit-address-family
 !
 address-family ipv4 vrf Client_B
  neighbor 10.0.4.1 remote-as 65001
  neighbor 10.0.4.1 activate
```

```
 exit-address-family
!
snmp-server community ZABBIX RO acl_snmp
snmp-server location TG-GNS3
!
end
```

Nota. Elaboración propia.

Código 5. Configuración del router PE2

```
hostname PE-2
!
vrf definition Client_A
 rd 100:110
 route-target export 100:1000
 route-target import 100:1000
 !
 address-family ipv4
 exit-address-family
!
vrf definition Client_B
 rd 100:120
 route-target export 100:2000
 route-target import 100:2000
 !
 address-family ipv4
 exit-address-family
 !
ip cef

!
interface Loopback0
 ip address 10.10.10.5 255.255.255.255
 ip ospf 1 area 0
!
interface Ethernet0/0
 ip address 10.1.1.22 255.255.255.252
 ip ospf 1 area 0
 mpls ip
!
interface Ethernet0/1
 vrf forwarding Client_A
 ip address 10.0.6.2 255.255.255.0
!
interface Ethernet0/1.100
 encapsulation dot1Q 100
 ip address 172.16.100.17 255.255.255.252
 ip ospf 1 area 0
!
interface Ethernet0/2
 vrf forwarding Client_A
 ip address 10.1.6.2 255.255.255.0
!
interface Ethernet0/2.100
 encapsulation dot1Q 100
 ip address 172.16.100.13 255.255.255.252
 ip ospf 1 area 0
!
interface Ethernet0/3
 vrf forwarding Client_B
 ip address 10.0.6.2 255.255.255.0
!
```

```
interface Ethernet0/3.100
 encapsulation dot1Q 100
 ip address 172.16.100.9 255.255.255.252
 ip ospf 1 area 0
!
interface Ethernet1/0
 no ip address
 xconnect 10.10.10.4 100 encapsulation mpls
!
router ospf 1
 router-id 5.5.5.5
 network 10.10.10.5 0.0.0.0 area 0
!
router bgp 65000
 bgp log-neighbor-changes
 neighbor 10.10.10.2 remote-as 65000
 neighbor 10.10.10.2 update-source Loopback0
 !
 address-family vpnv4
  neighbor 10.10.10.2 activate
  neighbor 10.10.10.2 send-community both
 exit-address-family
 !
 address-family ipv4 vrf Client_A
  neighbor 10.0.6.1 remote-as 65005
  neighbor 10.0.6.1 activate
  neighbor 10.1.6.1 remote-as 65004
  neighbor 10.1.6.1 activate
 exit-address-family
 !
 address-family ipv4 vrf Client_B
  neighbor 10.0.6.1 remote-as 65003
  neighbor 10.0.6.1 activate
 exit-address-family
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 6. Configuración del router CE-B1

```
hostname CE-B1
!
ip cef

!
interface Ethernet0/0
 ip address 10.0.4.1 255.255.255.0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 172.16.100.2 255.255.255.252
```

```
!
router bgp 65001
 bgp log-neighbor-changes
 neighbor 10.0.4.2 remote-as 65000
 !
 address-family ipv4
  redistribute connected
  neighbor 10.0.4.2 activate
 exit-address-family
!
ip route 10.0.4.12 255.255.255.255 172.16.100.1
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 7. Configuración del router CE-B2

```
hostname CE-B2
!
ip cef
!
interface Ethernet0/0
 ip address 10.0.6.1 255.255.255.0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 172.16.100.10 255.255.255.252
!
router bgp 65003
 bgp log-neighbor-changes
 neighbor 10.0.6.2 remote-as 65000
 !
 address-family ipv4
  redistribute connected
  neighbor 10.0.6.2 activate
 exit-address-family
!
ip route 10.0.4.12 255.255.255.255 172.6.100.9
ip route 10.0.4.12 255.255.255.255 172.16.100.9
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 8. Configuración del router CE-A1

```
hostname CE-A1
!
ip cef
!
interface Ethernet0/0
 ip address 10.0.4.1 255.255.255.0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 172.16.100.6 255.255.255.252
!
router bgp 65002
 bgp log-neighbor-changes
 neighbor 10.0.4.2 remote-as 65000
 !
 address-family ipv4
  redistribute connected
  neighbor 10.0.4.2 activate
 exit-address-family
!
ip route 10.0.4.12 255.255.255.255 172.16.100.5
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 9. Configuración del router CE-A2

```
hostname CE-A2
!
ip cef
!
interface Ethernet0/0
 ip address 10.1.6.1 255.255.255.0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 172.16.100.14 255.255.255.252
!
router bgp 65004
 bgp log-neighbor-changes
 neighbor 10.1.6.2 remote-as 65000
 !
 address-family ipv4
  redistribute connected
  neighbor 10.1.6.2 activate
 exit-address-family
!
ip route 10.0.4.12 255.255.255.255 172.16.100.13
ip route 10.0.6.0 255.255.255.0 10.1.6.2
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 10. Configuración del router CE-A3

```
hostname CE-A3
ip cef
!
interface Ethernet0/0
 ip address 10.0.6.1 255.255.255.0
!
interface Ethernet0/0.100
 encapsulation dot1Q 100
 ip address 172.16.100.18 255.255.255.252
!
router bgp 65005
 bgp log-neighbor-changes
 neighbor 10.0.6.2 remote-as 65000
 !
 address-family ipv4
  redistribute connected
  neighbor 10.0.6.2 activate
  no neighbor 2001:DB8:A:2::1 activate
 exit-address-family
 !
ip route 10.0.4.12 255.255.255.255 172.16.100.17
!
snmp-server community ZABBIX RO acl_snmp
!
end
```

Nota. Elaboración propia.

Código 11. Configuración del router CE-C1

```
hostname CE-C1
!
interface Ethernet0/0
 ip address 192.168.100.1 255.255.255.0
!
end
```

Nota. Elaboración propia.

Código 12. Configuración del router CE-C2

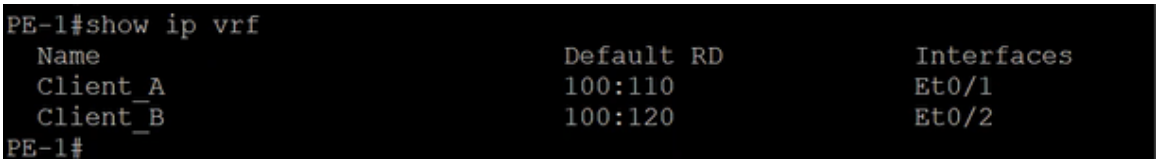
```
hostname CE-C2
!
interface Ethernet0/0
 ip address 192.168.100.2 255.255.255.0
!
end
```

Nota. Elaboración propia.

8.3. Verificación de configuración

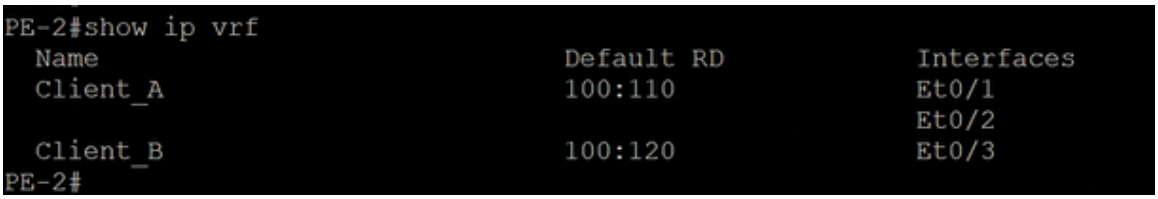
8.3.1. Verificación de VRFs de PE a CE

Figura 27. Verificación de VRF configurados en PE1.



Nota. Elaboración propia. Confirma la existencia de las instancias ClientA y ClientB, con sus RD e interfaces asociadas.

Figura 28. Verificación de VRF configurados en PE2.



Nota. Elaboración propia. Confirma la existencia de las instancias ClientA y ClientB, con sus RD e interfaces asociadas.

Figura 29. Interfaces asociadas a VRF en PE1.

Nota. Elaboración propia. Verificación de asociación de las interfaces físicas a las VRF ClientA y ClientB.

Figura 30. Interfaces asociadas a VRF en PE2.

```
PE-2#
PE-2#show ip vrf interfaces
Interface              IP-Address      VRF
Protocol
Et0/1                  10.0.6.2        Client_A
up
Et0/2                  10.1.6.2        Client_A
up
Et0/3                  10.0.6.2        Client_B
up
PE-2#
```

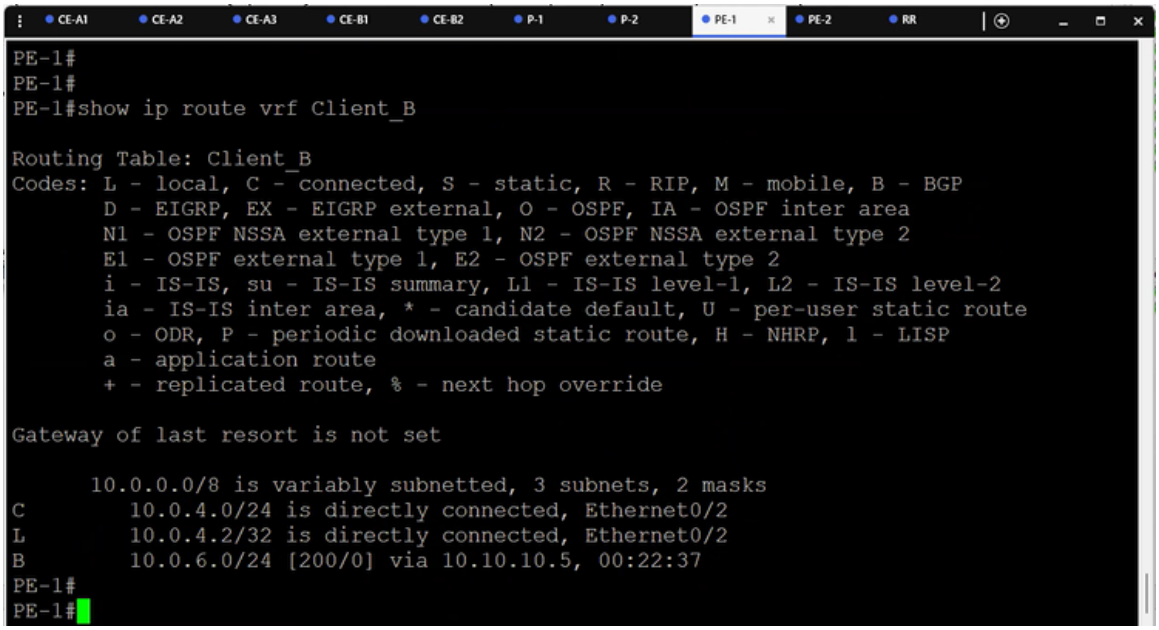
Nota. Elaboración propia. Verificación de asociación de las interfaces físicas a las VRF ClientA y ClientB.

Figura 31. Tabla de enrutamiento VRF ClientA en PE1.

```
PE-1#
PE-1#
PE-1#show ip route vrf Client_A

Routing Table: Client_A
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 4 subnets, 2 masks
C        10.0.4.0/24 is directly connected, Ethernet0/1
L        10.0.4.2/32 is directly connected, Ethernet0/1
B        10.0.6.0/24 [200/0] via 10.10.10.5, 00:21:29
B        10.1.6.0/24 [200/0] via 10.10.10.5, 00:21:29
PE-1#
```

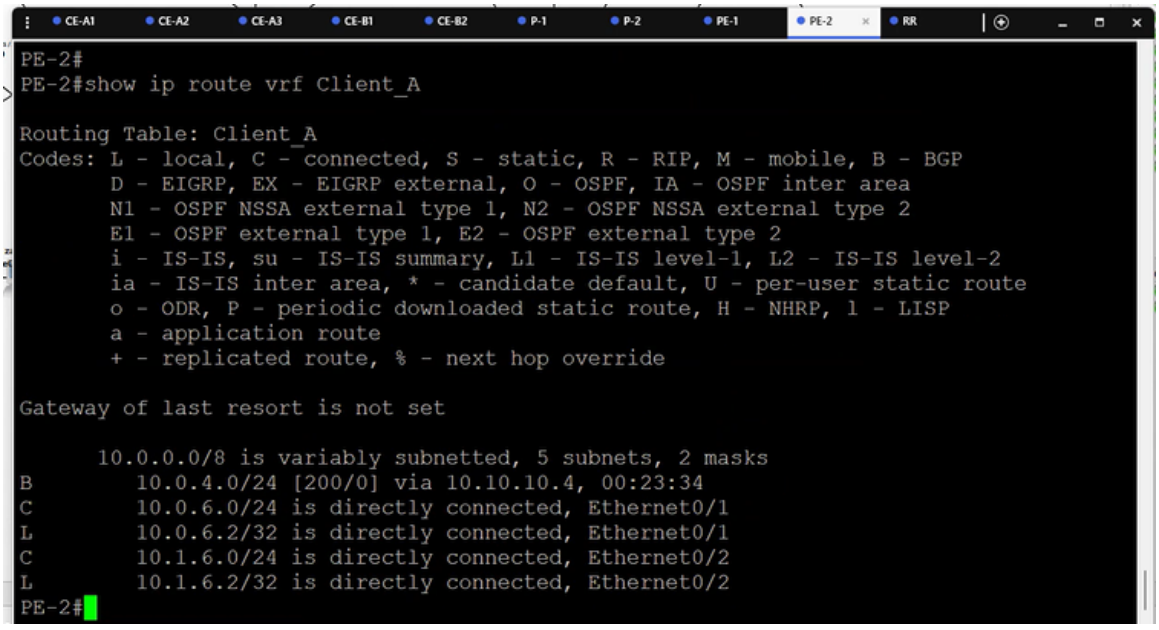
Nota. Elaboración propia. Rutas activas y directamente conectadas, propagación de prefijos en el contexto VRF del PE.

Figura 32. Tabla de enrutamiento VRF ClientB en PE1.



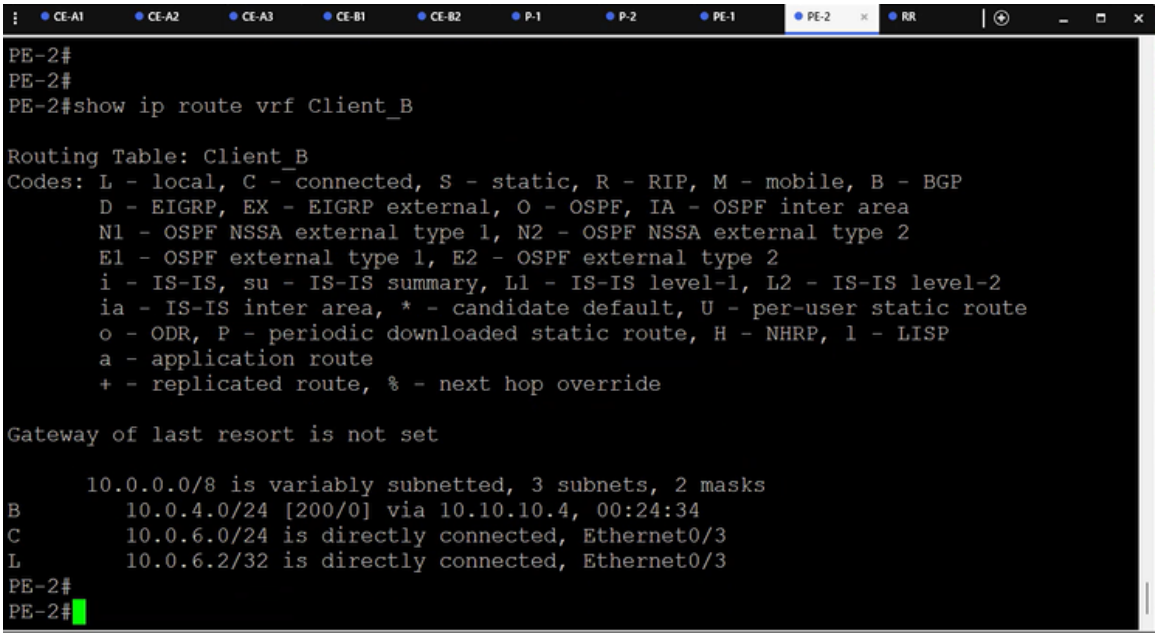
Nota. Elaboración propia. Rutas activas y directamente conectadas, propagación de prefijos en el contexto VRF del PE.

Figura 33. Tabla de enrutamiento VRF ClientA en PE2.



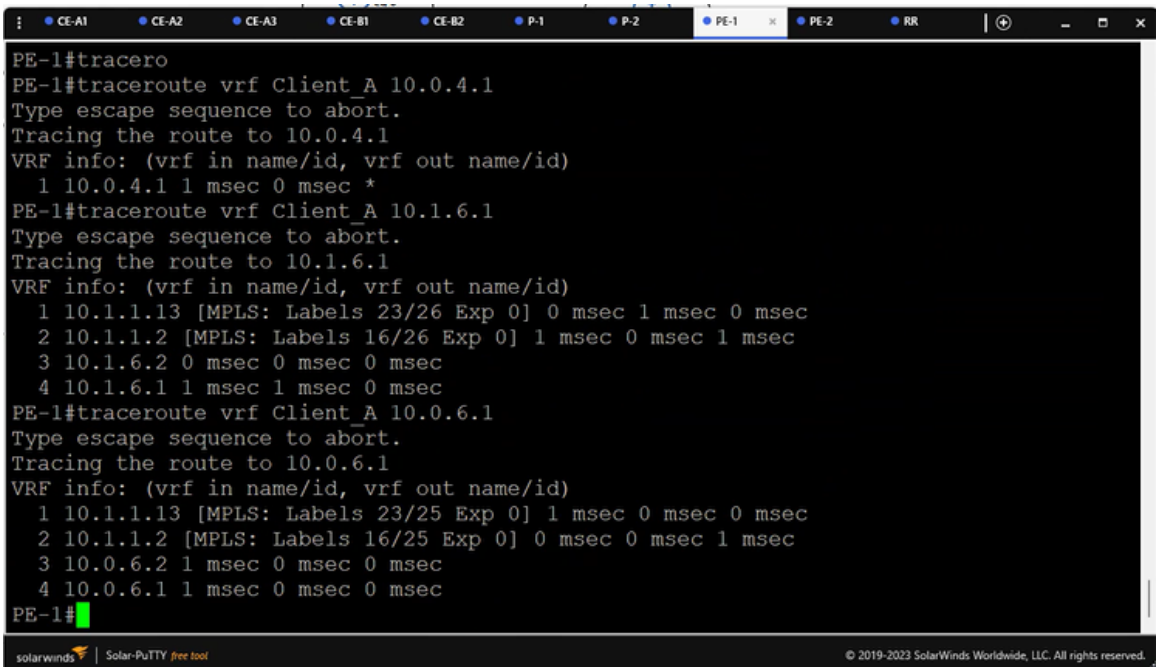
Nota. Elaboración propia. Rutas activas y directamente conectadas, propagación de prefijos en el contexto VRF del PE.

Figura 34. Tabla de enrutamiento VRF ClientB en PE2.



Nota. Elaboración propia. Rutas activas y directamente conectadas, propagación de prefijos en el contexto VRF del PE.

Figura 35. Verificación de rutas en VRF ClientA mediante traceroute en PE1.



Nota. Elaboración propia. Conectividad hacia prefijos remotos, conmutación de etiquetas MPLS y propagación de rutas en el PE1.

Figura 36. Verificación de rutas en VRF ClientB mediante traceroute en PE1.

```
PE-1#traceroute vrf Client_B 10.0.4.1
Type escape sequence to abort.
Tracing the route to 10.0.4.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.0.4.1 1 msec 0 msec *
PE-1#traceroute vrf Client_B 10.0.6.1
Type escape sequence to abort.
Tracing the route to 10.0.6.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.1.13 [MPLS: Labels 23/27 Exp 0] 0 msec 1 msec 0 msec
  2 10.1.1.2 [MPLS: Labels 16/27 Exp 0] 0 msec 1 msec 0 msec
  3 10.0.6.2 0 msec 0 msec 1 msec
  4 10.0.6.1 0 msec 1 msec 0 msec
PE-1#
```

Nota. Elaboración propia. Conectividad hacia prefijos remotos, conmutación de etiquetas MPLS y propagación de rutas en el PE1.

Figura 37. Verificación de rutas en VRF ClientA mediante traceroute en PE2.

```
PE-2#traceroute vr
PE-2#traceroute vrf Client_A 10.0.6.1
Type escape sequence to abort.
Tracing the route to 10.0.6.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.0.6.1 0 msec 0 msec 0 msec
PE-2#traceroute vrf Client_A 10.1.6.1
Type escape sequence to abort.
Tracing the route to 10.1.6.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.6.1 0 msec 0 msec 0 msec
PE-2#traceroute vrf Client_A 10.0.4.1
Type escape sequence to abort.
Tracing the route to 10.0.4.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.1.21 [MPLS: Labels 17/26 Exp 0] 1 msec 0 msec 0 msec
  2 10.1.1.1 [MPLS: Labels 19/26 Exp 0] 0 msec 0 msec 0 msec
  3 10.0.4.2 0 msec 0 msec 0 msec
  4 10.0.4.1 1 msec *  0 msec
PE-2#
```

Nota. Elaboración propia. Conectividad hacia prefijos remotos, conmutación de etiquetas MPLS y propagación de rutas en el PE2.

Figura 38. Verificación de rutas en VRF ClientB mediante traceroute en PE2.

```
PE-2#traceroute vrf Client_B 10.0.6.1
Type escape sequence to abort.
Tracing the route to 10.0.6.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.0.6.1 0 msec 0 msec 5 msec
PE-2#traceroute vrf Client_B 10.0.4.1
Type escape sequence to abort.
Tracing the route to 10.0.4.1
VRF info: (vrf in name/id, vrf out name/id)
  1 10.1.1.21 [MPLS: Labels 17/27 Exp 0] 1 msec 0 msec 0 msec
  2 10.1.1.1 [MPLS: Labels 19/27 Exp 0] 0 msec 0 msec 0 msec
  3 10.0.4.2 0 msec 0 msec 0 msec
  4 10.0.4.1 0 msec *  1 msec
PE-2#
```

Nota. Elaboración propia. Conectividad hacia prefijos remotos, conmutación de etiquetas MPLS y propagación de rutas en el PE2.

Figura 39. Verificación de reenvío CEF en VRF ClientA desde PE1.

```
PE-1#
PE-1#show ip cef vrf Client_A 10.0.4.1 detail
10.0.4.1/32, epoch 0, flags [attached]
  Adj source: IP adj out of Ethernet0/1, addr 10.0.4.1 F408CB48
    Dependent covered prefix type adjfib, cover 10.0.4.0/24
  attached to Ethernet0/1
PE-1#show ip cef vrf Client_A 10.0.6.1 detail
10.0.6.0/24, epoch 0, flags [rib defined all labels]
  recursive via 10.10.10.5 label 25
    nexthop 10.1.1.13 Ethernet0/0 label 23
PE-1#show ip cef vrf Client_A 10.1.6.1 detail
10.1.6.0/24, epoch 0, flags [rib defined all labels]
  recursive via 10.10.10.5 label 26
    nexthop 10.1.1.13 Ethernet0/0 label 23
PE-1#
```

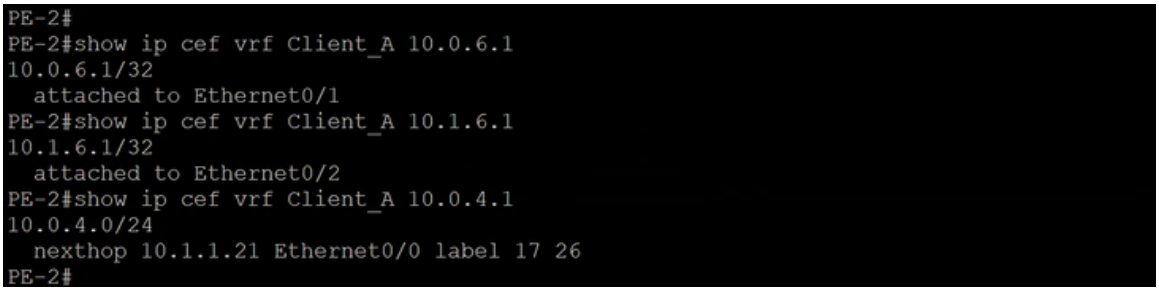
Nota. Elaboración propia. Asociación de prefijos con interfaces y etiquetas MPLS.

Figura 40. Verificación de reenvío CEF en VRF ClientB desde PE1.

```
PE-1#show ip cef vrf Client_B 10.0.4.1 detail
10.0.4.1/32, epoch 0, flags [attached]
  Adj source: IP adj out of Ethernet0/2, addr 10.0.4.1 F408CA18
    Dependent covered prefix type adjfib, cover 10.0.4.0/24
  attached to Ethernet0/2
PE-1#show ip cef vrf Client_B 10.0.6.1 detail
10.0.6.0/24, epoch 0, flags [rib defined all labels]
  recursive via 10.10.10.5 label 27
    nexthop 10.1.1.13 Ethernet0/0 label 23
PE-1#
```

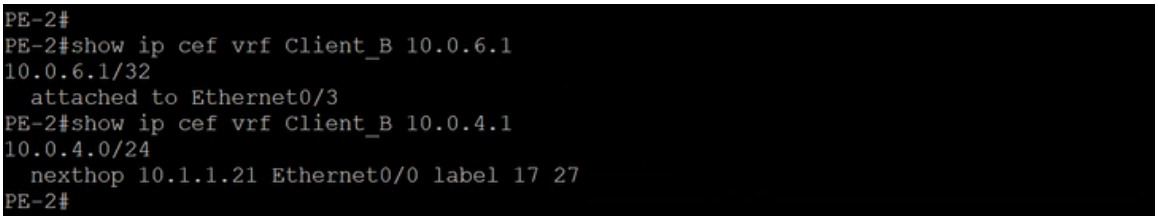
Nota. Elaboración propia. Asociación de prefijos con interfaces y etiquetas MPLS.

Figura 41. Verificación de reenvío CEF en VRF ClientA desde PE2.



Nota. Elaboración propia. Asociación de prefijos con interfaces y etiquetas MPLS.

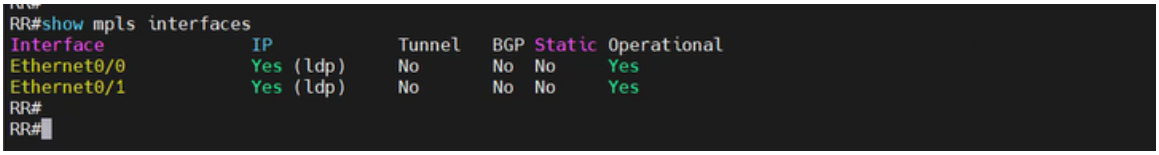
Figura 42. Verificación de reenvío CEF en VRF ClientB desde PE2.



Nota. Elaboración propia. Asociación de prefijos con interfaces y etiquetas MPLS.

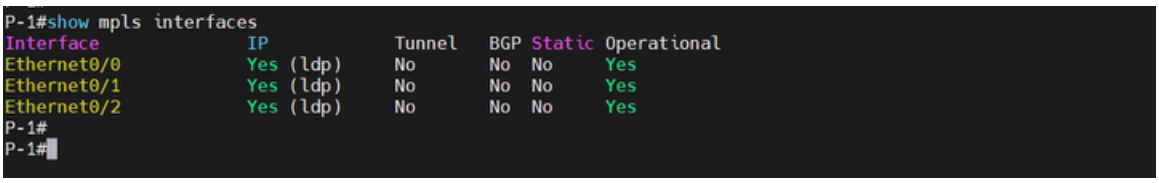
8.3.2. Verificación MPLS LDP

Figura 43. Verificación de interfaces MPLS operativas en el route reflector.



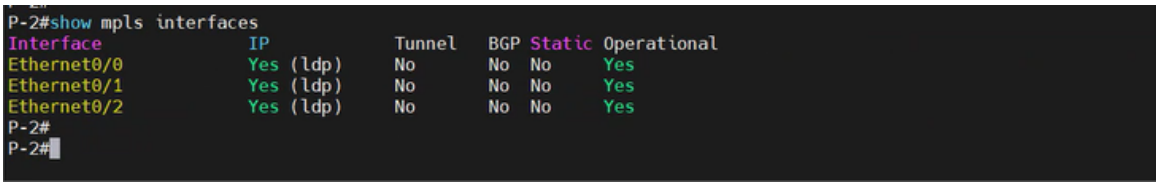
Nota. Elaboración propia. Se confirma que las interfaces Ethernet0/0 y Ethernet0/1 tienen LDP habilitado y se encuentran en estado operativo dentro del dominio MPLS.

Figura 44. Verificación de interfaces MPLS en el router P1.



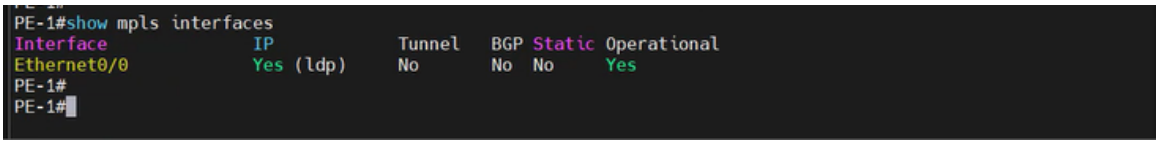
Nota. Elaboración propia. Se confirma que las interfaces Ethernet0/0, Ethernet0/1 y Ethernet0/2 tienen LDP habilitado y están operativas, garantizando la continuidad del plano de transporte MPLS.

Figura 45. Verificación de interfaces MPLS en el router P2.



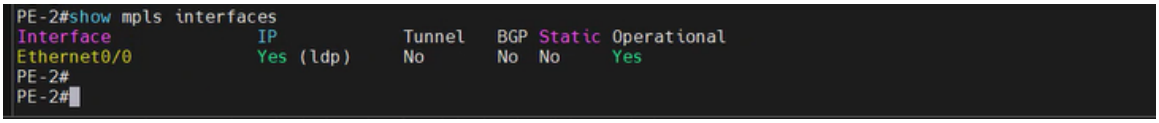
Nota. Elaboración propia. Se confirma que las interfaces Ethernet0/0, Ethernet0/1 y Ethernet0/2 tienen LDP habilitado y están operativas, garantizando la continuidad del plano de transporte MPLS.

Figura 46. Verificación de interfaz MPLS en el router PE1.



Nota. Elaboración propia. Se confirma que la interfaz Ethernet0/0 tiene LDP habilitado y se encuentra en estado operativo, lo que asegura la conectividad dentro del dominio MPLS.

Figura 47. Verificación de interfaz MPLS en el router PE2.



Nota. Elaboración propia. Se confirma que la interfaz Ethernet0/0 tiene LDP habilitado y se encuentra en estado operativo, lo que asegura la conectividad dentro del dominio MPLS.

Figura 48. Tabla de conmutación de etiquetas MPLS usada por el RR para rutas internas del backbone

```
RR#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         16         10.10.10.4/32    0             Et0/0      10.1.1.5
17         Pop Label  10.10.10.3/32    0             Et0/1      10.1.1.10
18         Pop Label  10.10.10.1/32    0             Et0/0      10.1.1.5
19         17         172.16.100.4/30  0             Et0/0      10.1.1.5
20         18         172.16.100.0/30  0             Et0/0      10.1.1.5
21         Pop Label  10.1.1.20/30     0             Et0/1      10.1.1.10
22         19         10.0.4.0/24      0             Et0/0      10.1.1.5
23         Pop Label  10.1.1.12/30     0             Et0/0      10.1.1.5
24         Pop Label  10.1.1.0/30      0             Et0/0      10.1.1.5
           Pop Label  10.1.1.0/30      0             Et0/1      10.1.1.10
25         24         10.10.10.5/32    0             Et0/1      10.1.1.10
26         25         172.16.100.16/30 0             Et0/1      10.1.1.10
27         26         172.16.100.12/30 0             Et0/1      10.1.1.10
28         27         172.16.100.8/30  0             Et0/1      10.1.1.10
RR#
RR#
```

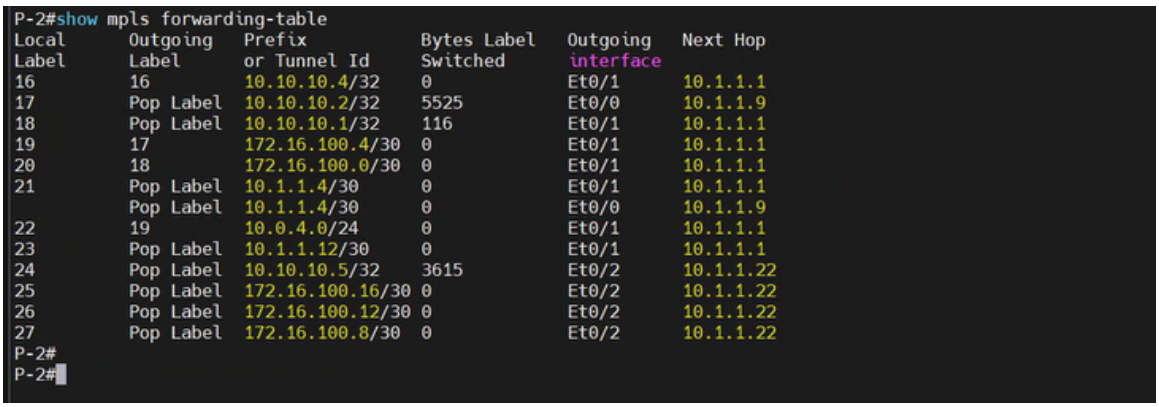
Nota. Elaboración propia. Se muestra las etiquetas locales y salientes con sus interfaces y next hops, lo que confirma el correcto intercambio de etiquetas dentro del dominio MPLS.

Figura 49. Tabla de reenvío MPLS en el router P1.

```
P-1#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         Pop Label  10.10.10.4/32    3076          Et0/0      10.1.1.14
17         Pop Label  172.16.100.4/30  0             Et0/0      10.1.1.14
18         Pop Label  172.16.100.0/30  0             Et0/0      10.1.1.14
19         Pop Label  10.0.4.0/24      0             Et0/0      10.1.1.14
20         24         10.10.10.5/32    0             Et0/2      10.1.1.2
21         Pop Label  10.10.10.3/32    0             Et0/2      10.1.1.2
22         Pop Label  10.10.10.2/32    254           Et0/1      10.1.1.6
23         25         172.16.100.16/30 0             Et0/2      10.1.1.2
24         26         172.16.100.12/30 0             Et0/2      10.1.1.2
25         27         172.16.100.8/30  0             Et0/2      10.1.1.2
26         Pop Label  10.1.1.20/30     0             Et0/2      10.1.1.2
27         Pop Label  10.1.1.8/30      0             Et0/2      10.1.1.2
           Pop Label  10.1.1.8/30      0             Et0/1      10.1.1.6
P-1#
P-1#
*Oct 21 23:52:03.234: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (10.0.4.12)
P-1#
```

Nota. Elaboración propia. Se evidencia la asignación y el intercambio de etiquetas entre los enlaces del core, validando el funcionamiento del plano de reenvío MPLS.

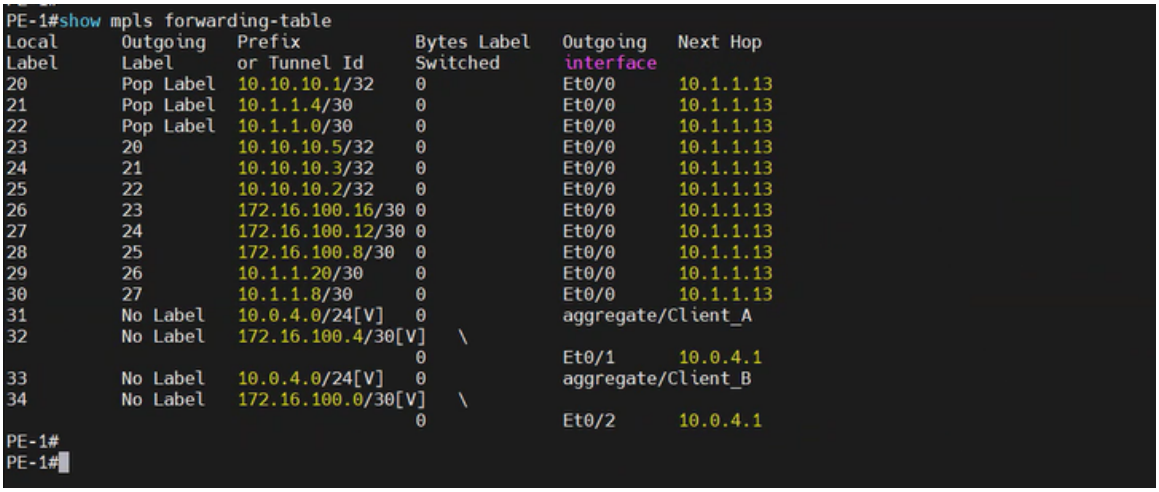
Figura 50. Tabla de reenvío MPLS en el router P2.



```
P-2#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         16         10.10.10.4/32    0             Et0/1      10.1.1.1
17         Pop Label  10.10.10.2/32    5525          Et0/0      10.1.1.9
18         Pop Label  10.10.10.1/32    116           Et0/1      10.1.1.1
19         17         172.16.100.4/30  0             Et0/1      10.1.1.1
20         18         172.16.100.0/30  0             Et0/1      10.1.1.1
21         Pop Label  10.1.1.4/30      0             Et0/1      10.1.1.1
           Pop Label  10.1.1.4/30      0             Et0/0      10.1.1.9
22         19         10.0.4.0/24      0             Et0/1      10.1.1.1
23         Pop Label  10.1.1.12/30     0             Et0/1      10.1.1.1
24         Pop Label  10.10.10.5/32    3615          Et0/2      10.1.1.22
25         Pop Label  172.16.100.16/30 0             Et0/2      10.1.1.22
26         Pop Label  172.16.100.12/30 0             Et0/2      10.1.1.22
27         Pop Label  172.16.100.8/30  0             Et0/2      10.1.1.22
P-2#
P-2#
```

Nota. Elaboración propia. Se muestra el intercambio de etiquetas y los next hops asociados, lo que confirma la correcta propagación del plano de datos en la red MPLS.

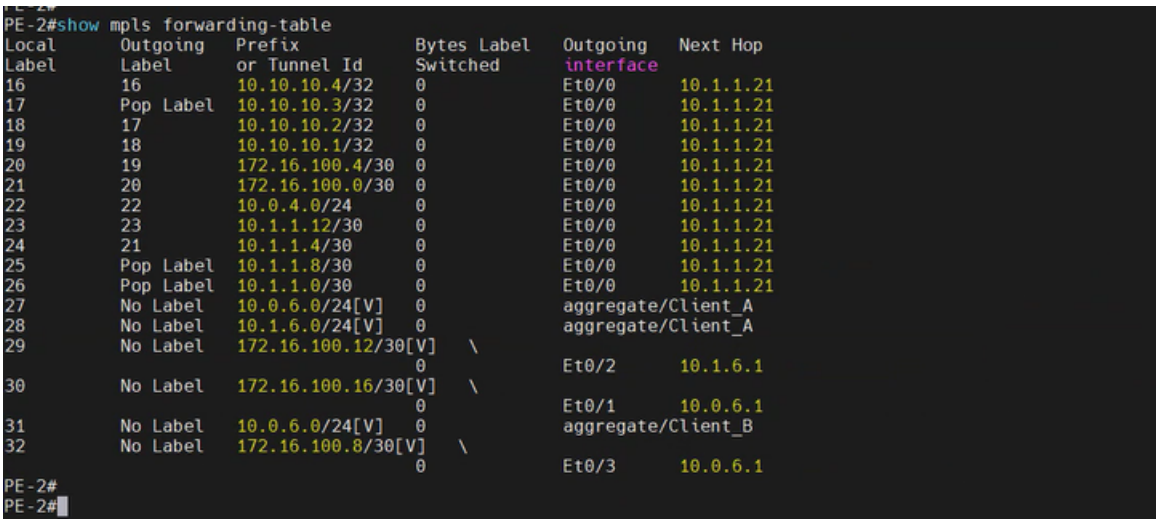
Figura 51. Tabla de reenvío MPLS en el router PE1.



```
PE-1#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
20         Pop Label  10.10.10.1/32    0             Et0/0      10.1.1.13
21         Pop Label  10.1.1.4/30      0             Et0/0      10.1.1.13
22         Pop Label  10.1.1.0/30      0             Et0/0      10.1.1.13
23         20         10.10.10.5/32    0             Et0/0      10.1.1.13
24         21         10.10.10.3/32    0             Et0/0      10.1.1.13
25         22         10.10.10.2/32    0             Et0/0      10.1.1.13
26         23         172.16.100.16/30 0             Et0/0      10.1.1.13
27         24         172.16.100.12/30 0             Et0/0      10.1.1.13
28         25         172.16.100.8/30  0             Et0/0      10.1.1.13
29         26         10.1.1.20/30     0             Et0/0      10.1.1.13
30         27         10.1.1.8/30      0             Et0/0      10.1.1.13
31         No Label   10.0.4.0/24[V]   0             aggregate/Client_A
32         No Label   172.16.100.4/30[V]   \
                                       0             Et0/1      10.0.4.1
33         No Label   10.0.4.0/24[V]   0             aggregate/Client_B
34         No Label   172.16.100.0/30[V]   \
                                       0             Et0/2      10.0.4.1
PE-1#
PE-1#
```

Nota. Elaboración propia. Se muestra las etiquetas asociadas a las VRF Client A y Client B, lo que confirma la correcta segmentación y reenvío de tráfico en el borde del dominio MPLS.

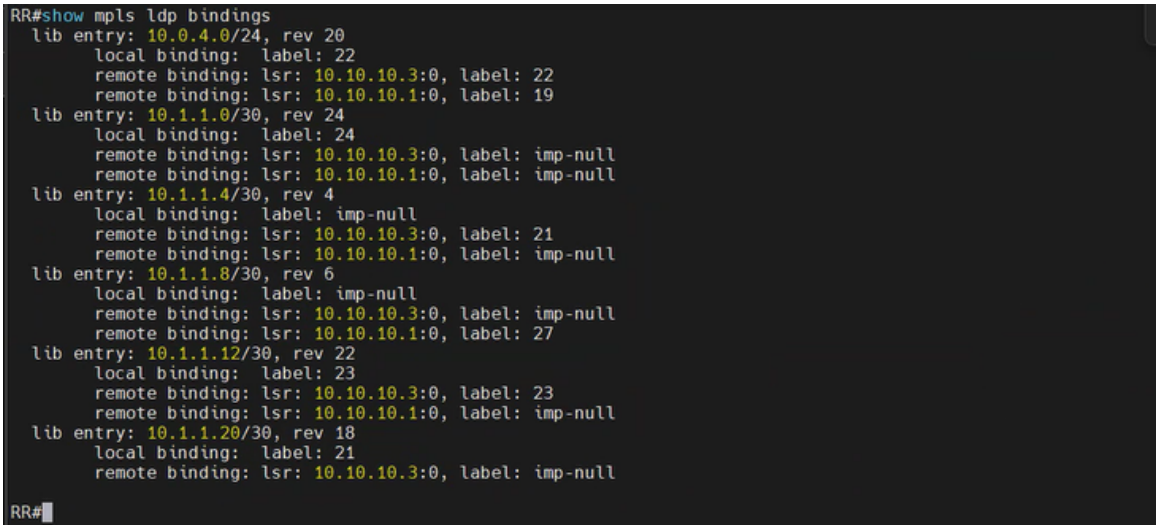
Figura 52. Tabla de reenvío MPLS en el router PE2.



```
PE-2#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
16         16         10.10.10.4/32    0             Et0/0      10.1.1.21
17         Pop Label  10.10.10.3/32    0             Et0/0      10.1.1.21
18         17         10.10.10.2/32    0             Et0/0      10.1.1.21
19         18         10.10.10.1/32    0             Et0/0      10.1.1.21
20         19         172.16.100.4/30  0             Et0/0      10.1.1.21
21         20         172.16.100.0/30  0             Et0/0      10.1.1.21
22         22         10.0.4.0/24      0             Et0/0      10.1.1.21
23         23         10.1.1.12/30     0             Et0/0      10.1.1.21
24         21         10.1.1.4/30      0             Et0/0      10.1.1.21
25         Pop Label  10.1.1.8/30      0             Et0/0      10.1.1.21
26         Pop Label  10.1.1.0/30      0             Et0/0      10.1.1.21
27         No Label   10.0.6.0/24[V]   0             aggregate/Client_A
28         No Label   10.1.6.0/24[V]   0             aggregate/Client_A
29         No Label   172.16.100.12/30[V]   \
                                       0             Et0/2      10.1.6.1
30         No Label   172.16.100.16/30[V]   \
                                       0             Et0/1      10.0.6.1
31         No Label   10.0.6.0/24[V]   0             aggregate/Client_B
32         No Label   172.16.100.8/30[V]   \
                                       0             Et0/3      10.0.6.1
PE-2#
PE-2#
```

Nota. Elaboración propia. Se confirma el intercambio de etiquetas asociado a las VRF Client A y Client B, validando la conectividad entre los sitios del cliente a través del backbone MPLS.

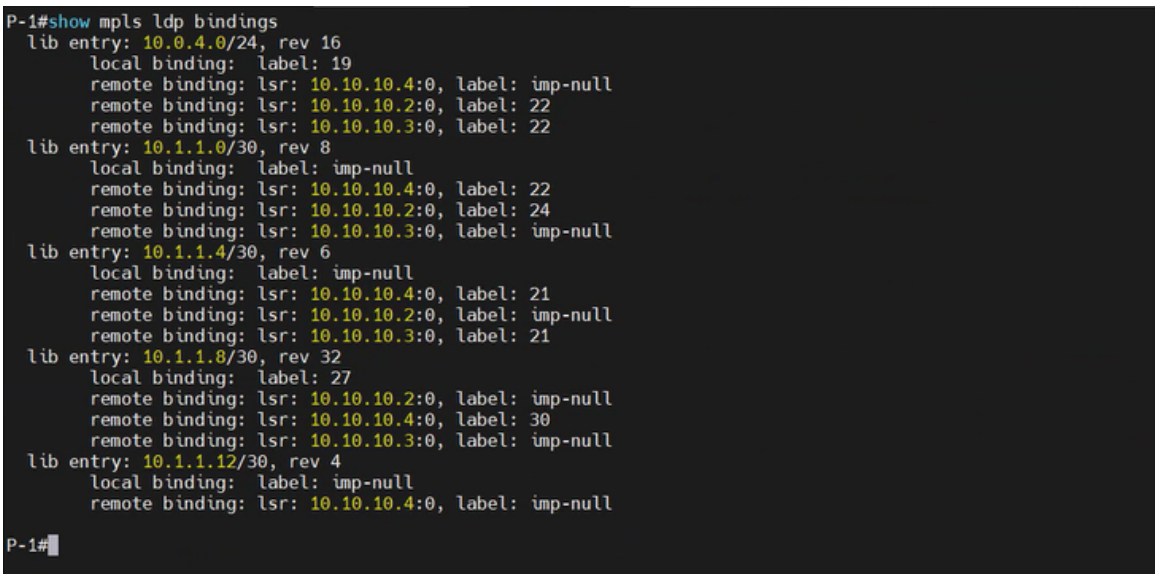
Figura 53. Tabla de asociaciones de etiquetas LDP en el route reflector.



```
RR#show mpls ldp bindings
  lib entry: 10.0.4.0/24, rev 20
        local binding:  label: 22
        remote binding: lsr: 10.10.10.3:0, label: 22
        remote binding: lsr: 10.10.10.1:0, label: 19
  lib entry: 10.1.1.0/30, rev 24
        local binding:  label: 24
        remote binding: lsr: 10.10.10.3:0, label: imp-null
        remote binding: lsr: 10.10.10.1:0, label: imp-null
  lib entry: 10.1.1.4/30, rev 4
        local binding:  label: imp-null
        remote binding: lsr: 10.10.10.3:0, label: 21
        remote binding: lsr: 10.10.10.1:0, label: imp-null
  lib entry: 10.1.1.8/30, rev 6
        local binding:  label: imp-null
        remote binding: lsr: 10.10.10.3:0, label: imp-null
        remote binding: lsr: 10.10.10.1:0, label: 27
  lib entry: 10.1.1.12/30, rev 22
        local binding:  label: 23
        remote binding: lsr: 10.10.10.3:0, label: 23
        remote binding: lsr: 10.10.10.1:0, label: imp-null
  lib entry: 10.1.1.20/30, rev 18
        local binding:  label: 21
        remote binding: lsr: 10.10.10.3:0, label: imp-null

RR#
```

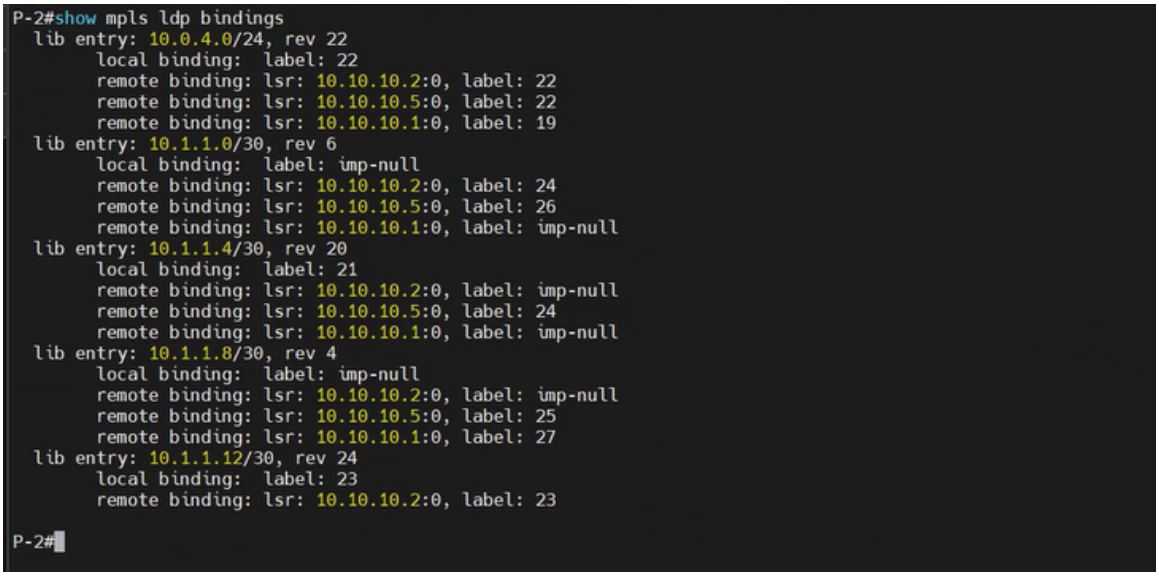
Nota. Elaboración propia. Se muestran las etiquetas locales y remotas asignadas a los prefijos del core, lo que confirma el correcto intercambio de información LDP entre routers MPLS.

Figura 54. Asociaciones de etiquetas LDP en el router P1.



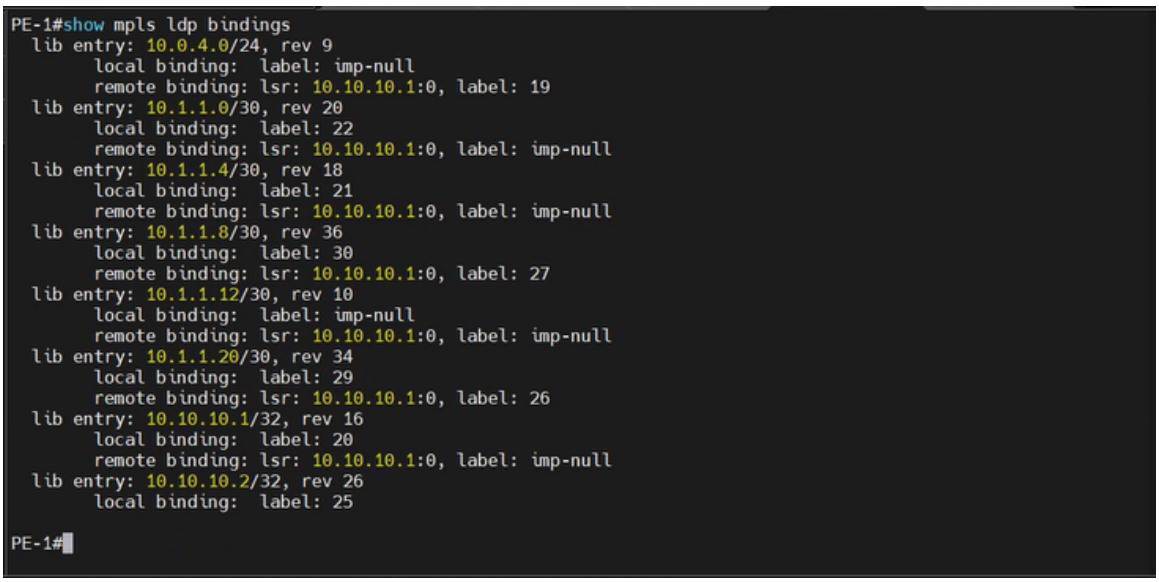
Nota. Elaboración propia. Se presentan las etiquetas locales y remotas intercambiadas con otros routers, lo que confirma la correcta operación del protocolo LDP dentro del backbone MPLS.

Figura 55. Asociaciones de etiquetas LDP en el router P2.



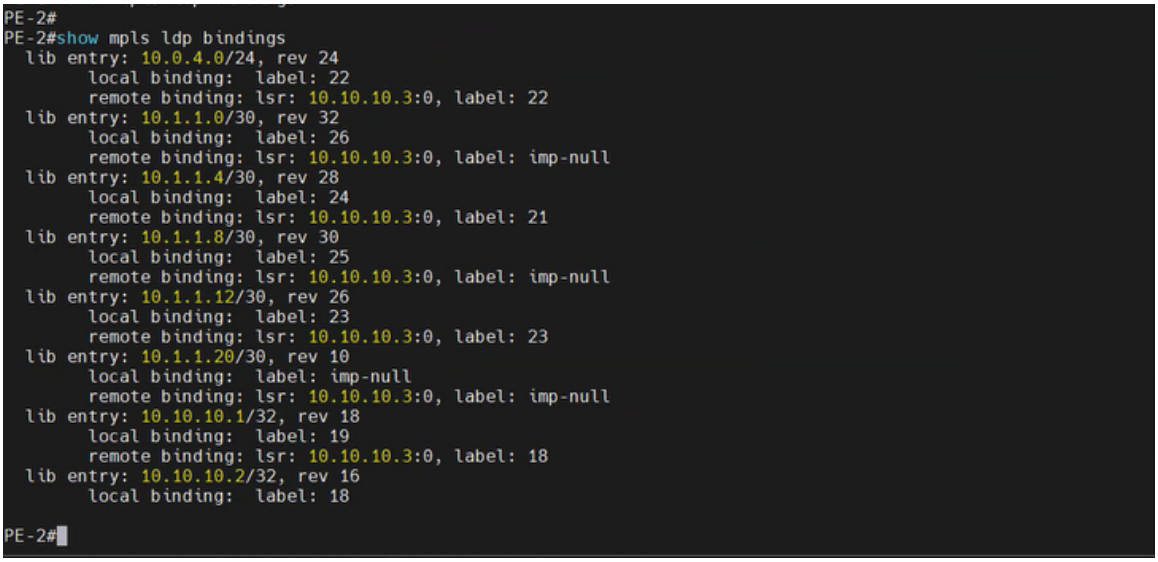
Nota. Elaboración propia. Se presentan las etiquetas locales y remotas intercambiadas con otros routers, lo que confirma la correcta operación del protocolo LDP dentro del backbone MPLS.

Figura 56. Asignación de etiquetas LDP en el router PE1.



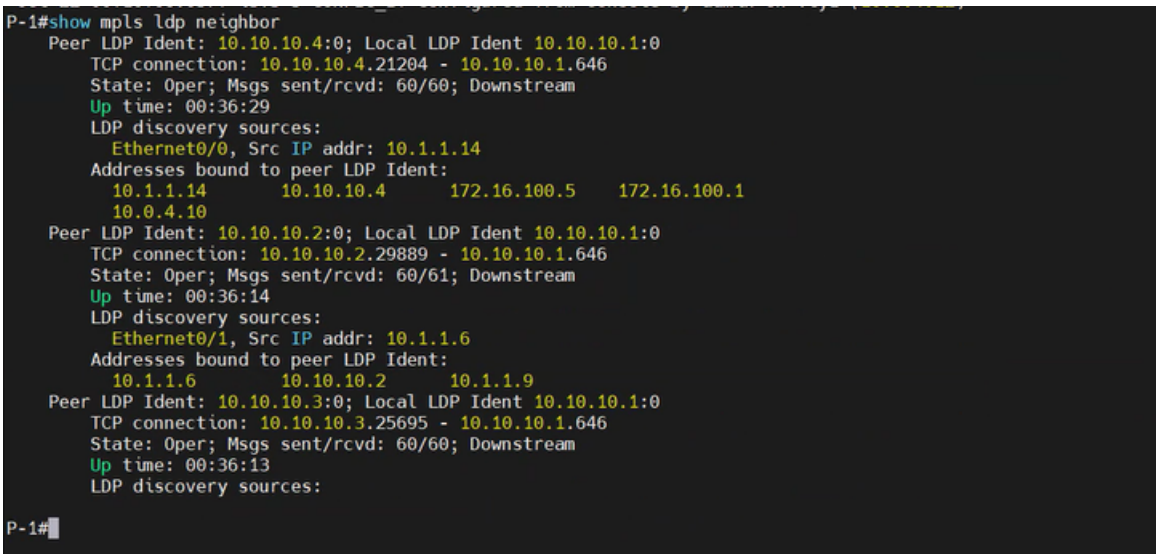
Nota. Elaboración propia. Se muestra las etiquetas locales y remotas asociadas a los prefijos del backbone, lo que confirma la correcta distribución de etiquetas para el reenvío MPLS.

Figura 57. Asignación de etiquetas LDP en el router PE2.



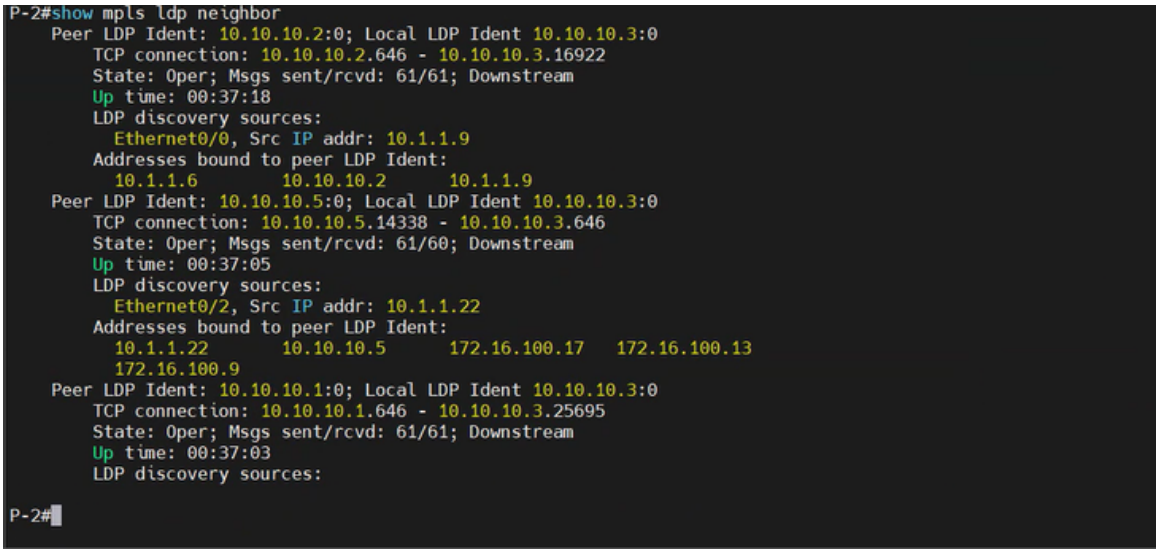
Nota. Elaboración propia. Se muestra las etiquetas locales y remotas asociadas a los prefijos del backbone, lo que confirma la correcta distribución de etiquetas para el reenvío MPLS.

Figura 58. Verificación de vecinos LDP en el router P1.



Nota. Elaboración propia. Se muestra las sesiones LDP establecidas con los routers adyacentes, lo que confirma la correcta propagación de etiquetas en el backbone MPLS.

Figura 59. Verificación de vecinos LDP en el router P2.



Nota. Elaboración propia. Se muestra las sesiones LDP establecidas con los routers adyacentes, lo que confirma la correcta propagación de etiquetas en el backbone MPLS.

Figura 60. Verificación de vecinos LDP en el router PE1.

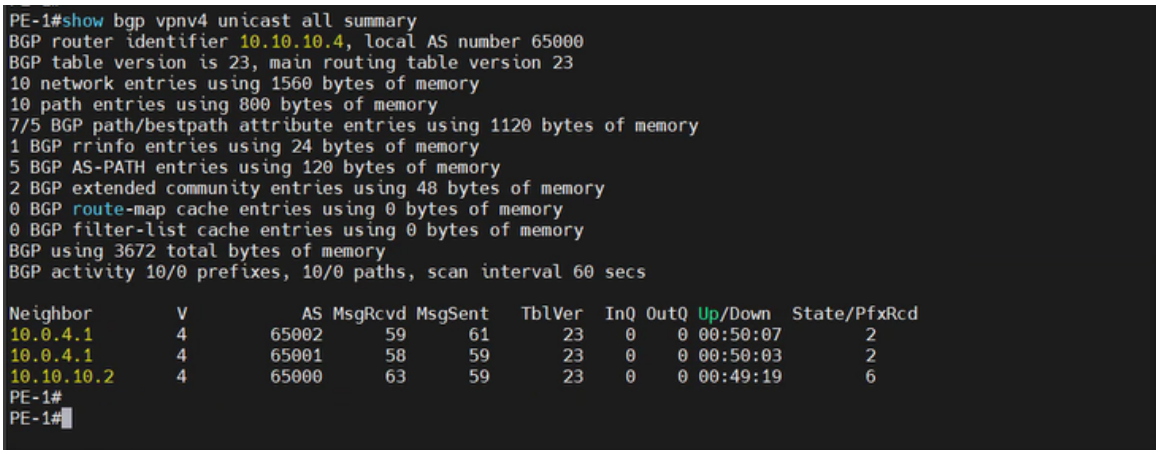
Nota. Elaboración propia. Se confirma la sesión LDP establecida con el router P1, garantizando el intercambio de etiquetas y la continuidad del plano MPLS.

Figura 61. Verificación de vecinos LDP en el router PE2.

Nota. Elaboración propia. Se confirma la sesión LDP establecida con el router P2, garantizando el intercambio de etiquetas y la continuidad del plano MPLS.

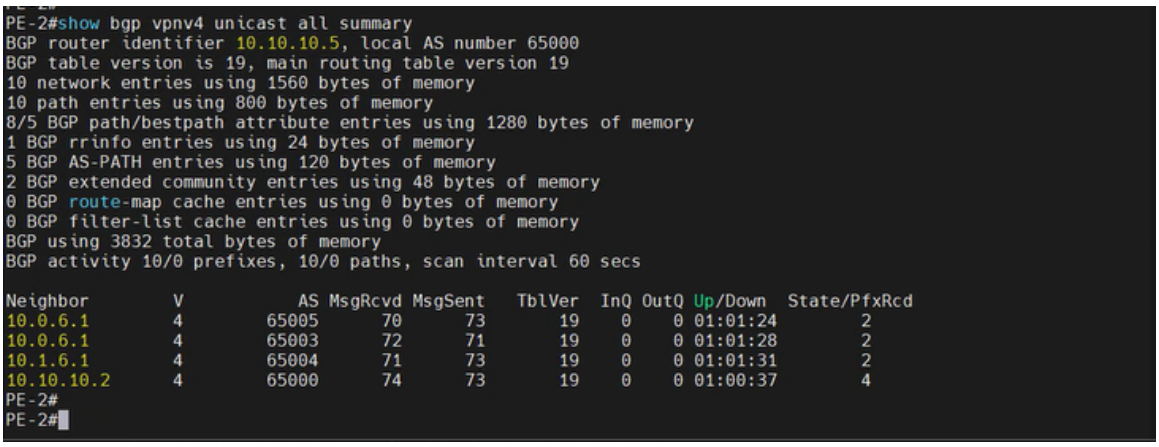
8.3.3. Verificación de PE a PE/RR

Figura 62. Verificación de sesiones BGP en el router PE1.



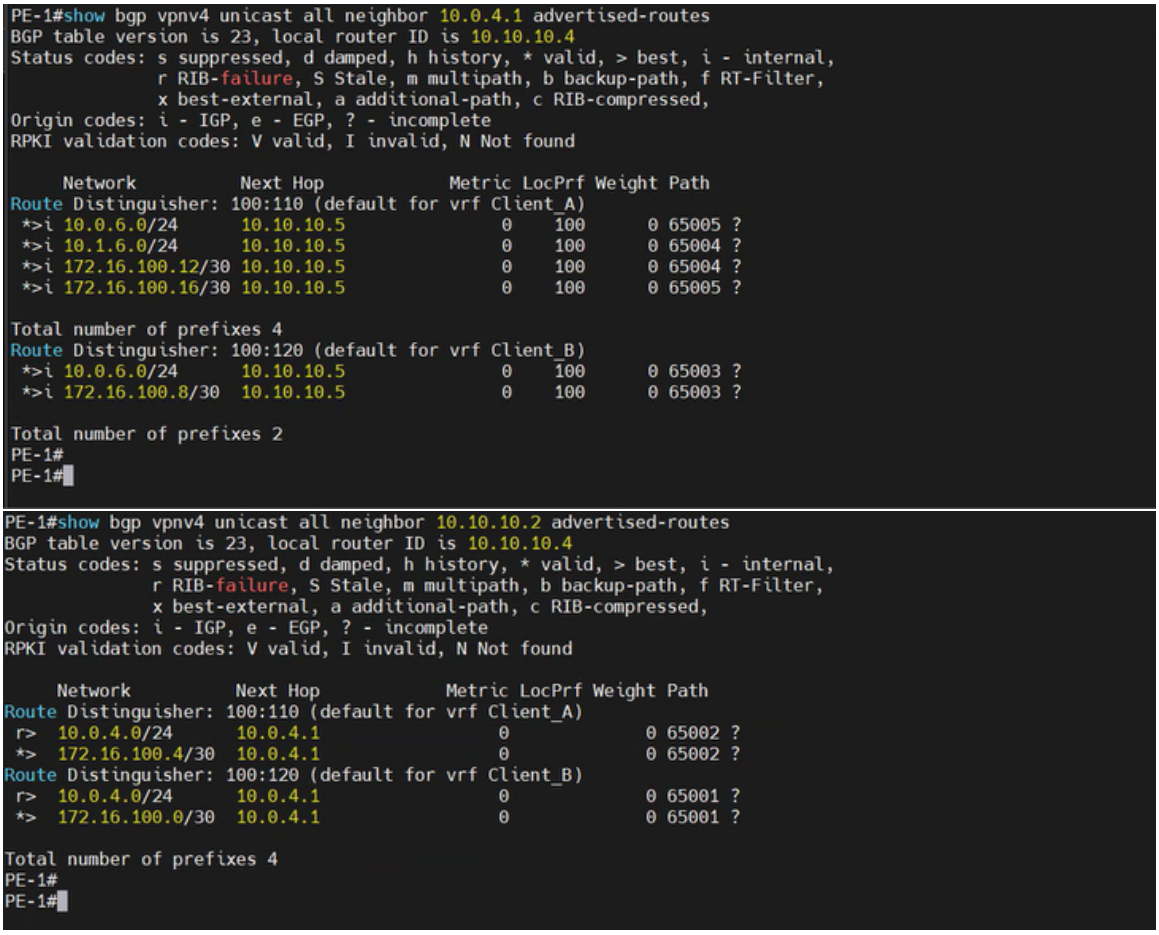
Nota. Elaboración propia. Se muestra los vecinos BGP activos y los prefijos recibidos, lo que confirma la correcta distribución de rutas VPNv4 dentro del dominio MPLS.

Figura 63. Verificación de sesiones BGP en el router PE2.



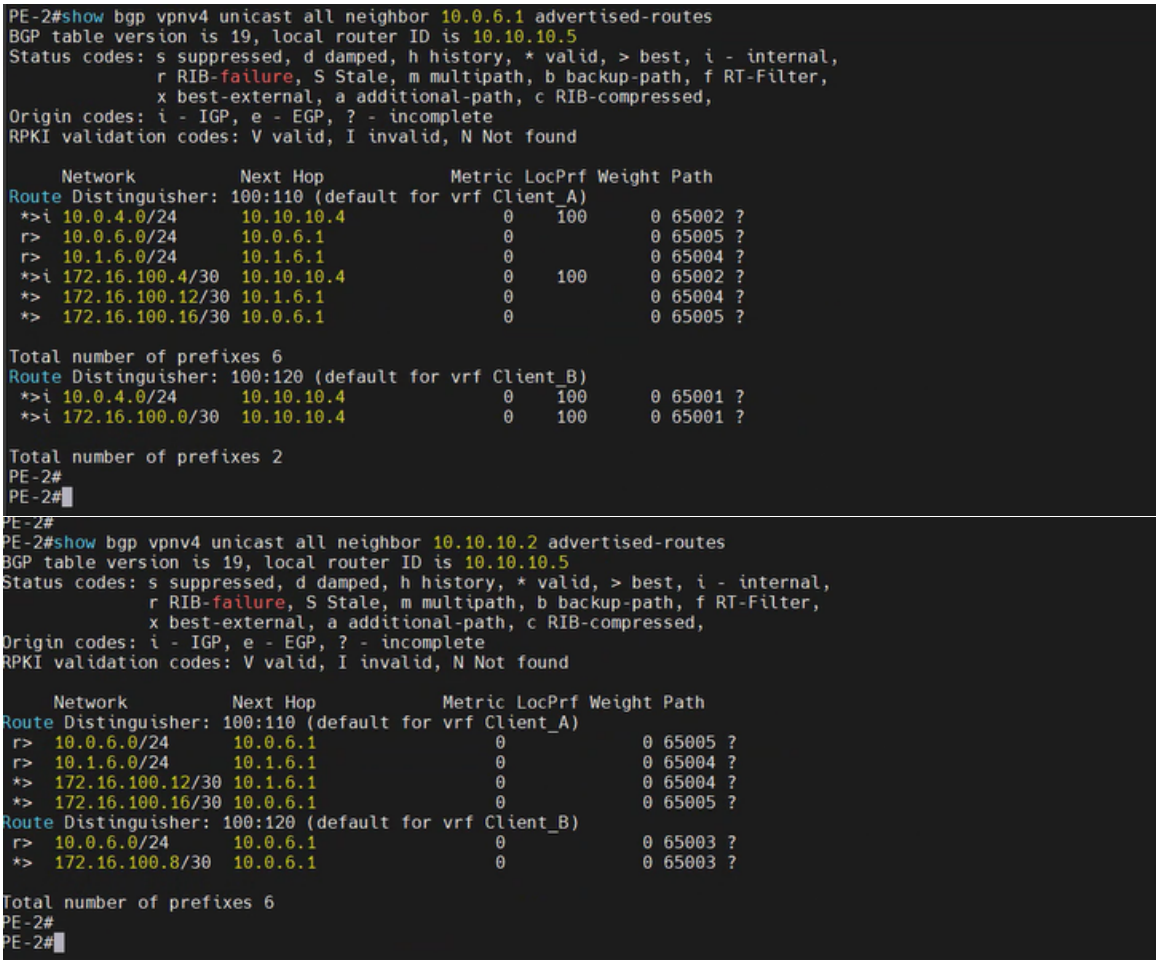
Nota. Elaboración propia. Se muestra los vecinos BGP activos y los prefijos recibidos, lo que confirma la correcta distribución de rutas VPNv4 dentro del dominio MPLS.

Figura 64. Rutas VPNv4 anunciadas por el router PE1 mediante BGP.



Nota. Elaboración propia. Se confirma la exportación de prefijos pertenecientes a las VRF Client A y Client B, validando la propagación de rutas en el dominio MPLS.

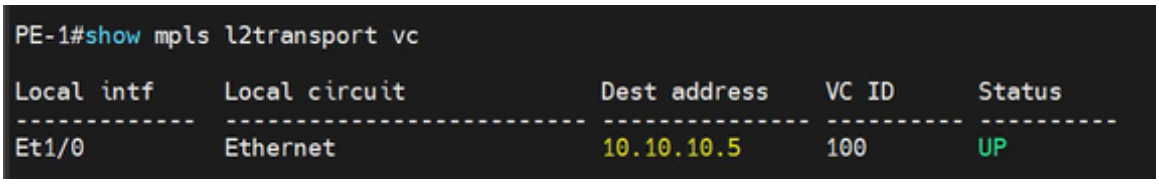
Figura 65. Rutas VPNv4 anunciadas por el router PE2 mediante BGP.



Nota. Elaboración propia. Se confirma la exportación de prefijos pertenecientes a las VRF Client A y Client B, validando la propagación de rutas en el dominio MPLS.

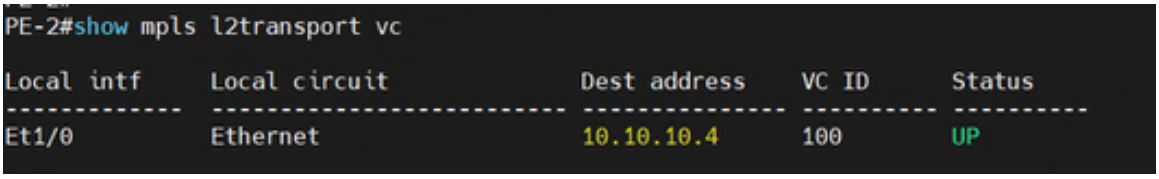
8.3.4. Verificación L2VPN

Figura 66. Verificación de sesiones L2VPN EVPN en el router PE1.



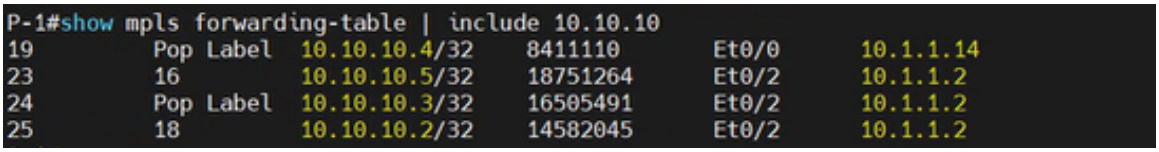
Nota. Elaboración propia. Se muestra los vecinos EVPN activos y los prefijos recibidos, lo que confirma la correcta distribución de rutas EVPN dentro del dominio MPLS.

Figura 67. Verificación de sesiones L2VPN EVPN en el router PE2.



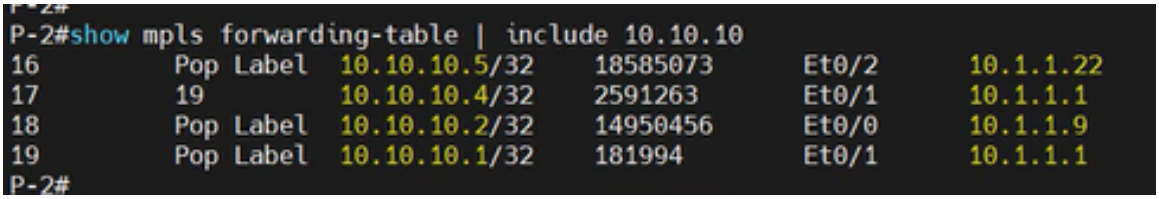
Nota. Elaboración propia. Se muestra los vecinos EVPN activos y los prefijos recibidos, lo que confirma la correcta distribución de rutas EVPN dentro del dominio MPLS.

Figura 68. Verificación de etiquetas hacia los loopbacks en el router PE1.



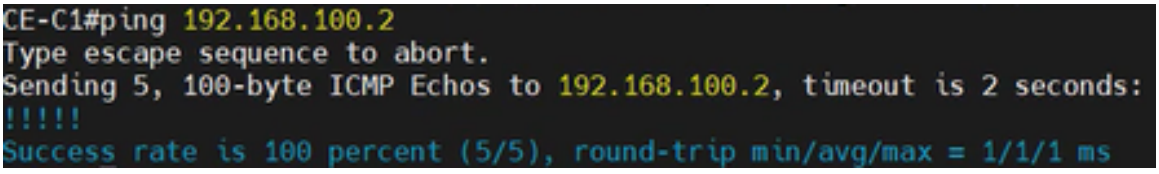
Nota. Elaboración propia. La presencia de rutas con intercambio o eliminación de etiquetas hacia los loopbacks de los PEs confirma la existencia del túnel MPLS necesario para el pseudowire.

Figura 69. Verificación de etiquetas hacia los loopbacks en el router PE2.



Nota. Elaboración propia. La presencia de rutas con intercambio o eliminación de etiquetas hacia los loopbacks de los PEs confirma la existencia del túnel MPLS necesario para el pseudowire.

Figura 70. Verificación de conectividad ICMP entre CE-A1 y CE-A2.



Nota. Elaboración propia. El tráfico IP viaja sobre la red MPLS sin ruteo intermedio, demostrando que los CE están conectados en el mismo dominio Ethernet a través de la L2VPN.

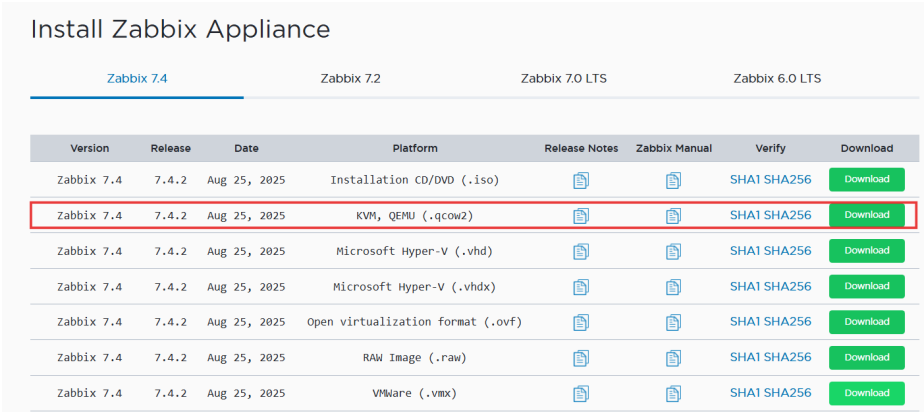
CAPÍTULO 9

Sistema de monitoreo Zabbix

9.1. Instalación de Zabbix Appliance 7.4, plataforma QEMU

Para la implementación del servidor de monitoreo dentro de GNS3 se requiere obtener la imagen preconfigurada de Zabbix Appliance. La descarga se realiza desde el sitio oficial de Zabbix, seleccionando el formato QEMU (.qcow2), compatible para ejecucion dentro de la misma máquina virtual que GNS3 para garantiza una integración directa y estable con la red simulada, evitando así configuraciones manuales adicionales.

Figura 71. Descarga de la Zabbix Appliance en formato QEMU (.qcow2).

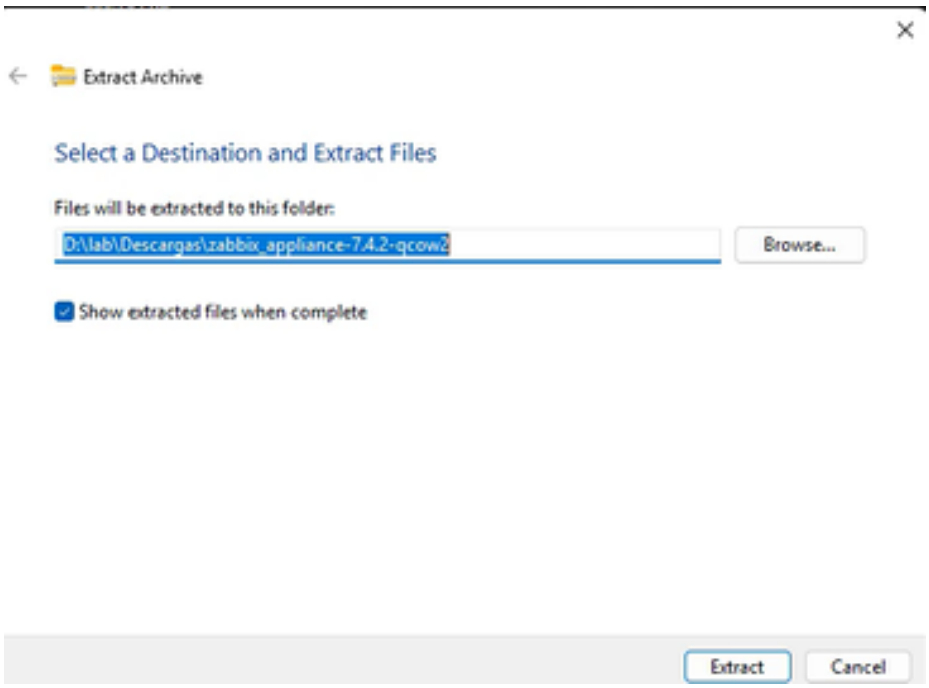


Nota. Elaboración propia. Se selecciona la imagen preconfigurada para KVM/QEMU.

Se procede a extraer el contenido del archivo comprimido en formato WinRAR, obte-

niendo el archivo de imagen (.qcwo2) necesario para la implementación dentro de la máquina virtual en GNS3.

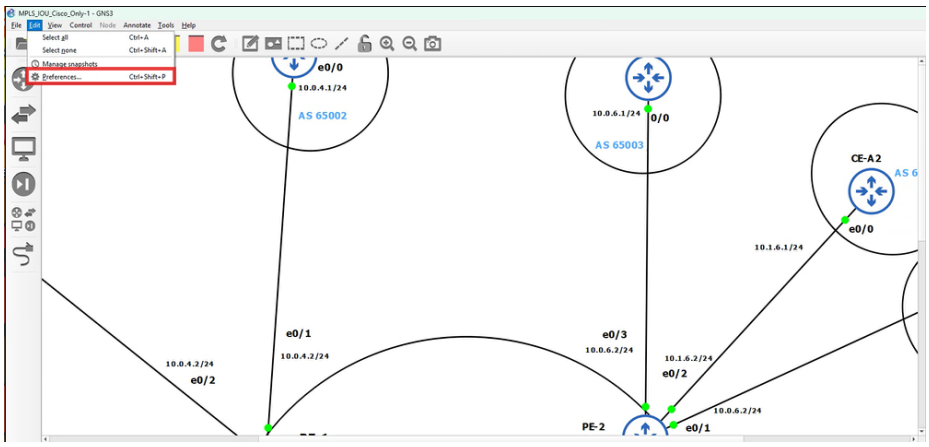
Figura 72. Extracción del WinRAR de la imagen de Zabbix Appliance



Nota. Elaboración propia. Extracción del archivo en formato WinRAR para ser utilizada en GNS3

Se integra la imagen de Zabbix en GNS3 accediendo al menú de preferencias desde la barra superior, seleccionando la opción Edit - Preferences.

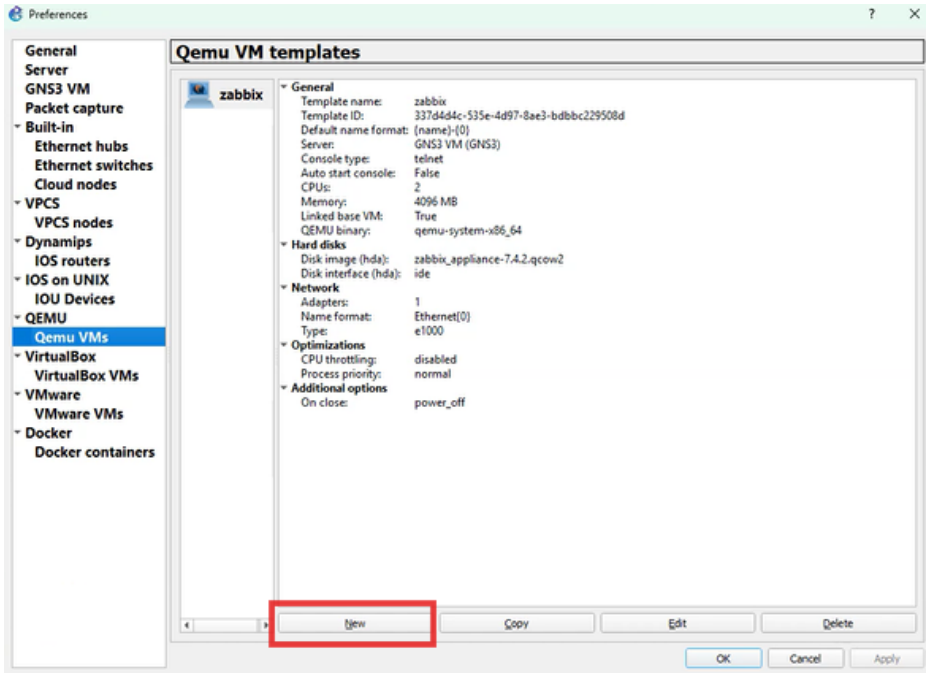
Figura 73. Acceso al menú de preferencias en GNS3



Nota. Elaboración propia. Desde la opción Edit - Preferences se configuran las imágenes y parámetros necesarios para la integración de dispositivos en el entorno de simulación.

Luego, se procedió a crear una nueva máquina virtual en GNS3 mediante el hipervisor QEMU. Esta acción inicia el asistente de configuración para definir los parámetros básicos de la máquina virtual, como el nombre, la memoria asignada, el número de interfaces de red y la imagen de disco a utilizar.

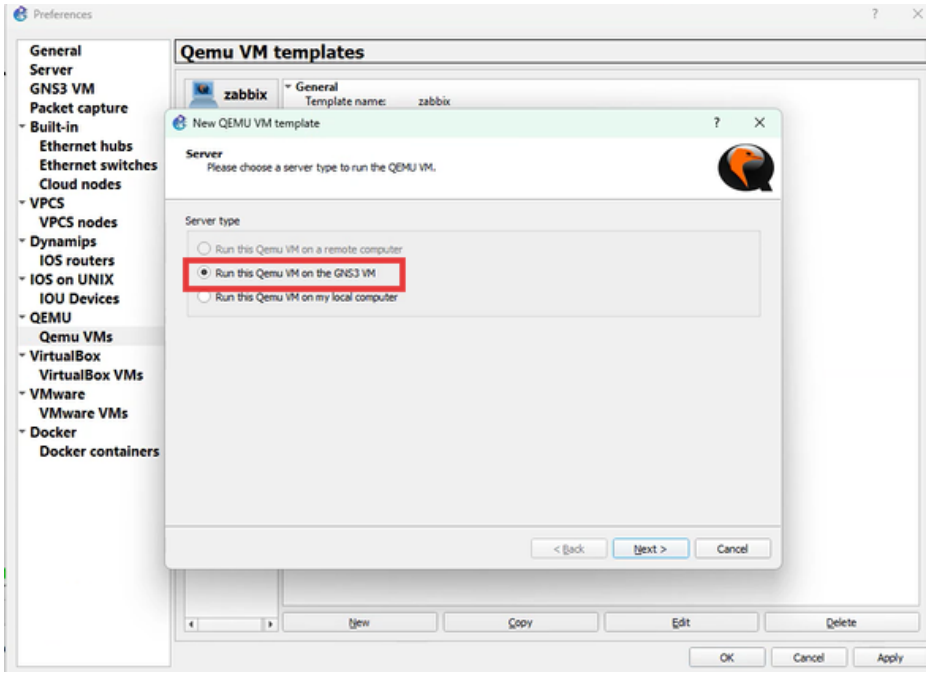
Figura 74. Creación de una nueva máquina virtual QEMU en GNS3.



Nota. Elaboración propia. En la sección QEMU VM *templates*, se selecciona la opción *New* para Incorporación de una nueva plantilla de máquina virtual.

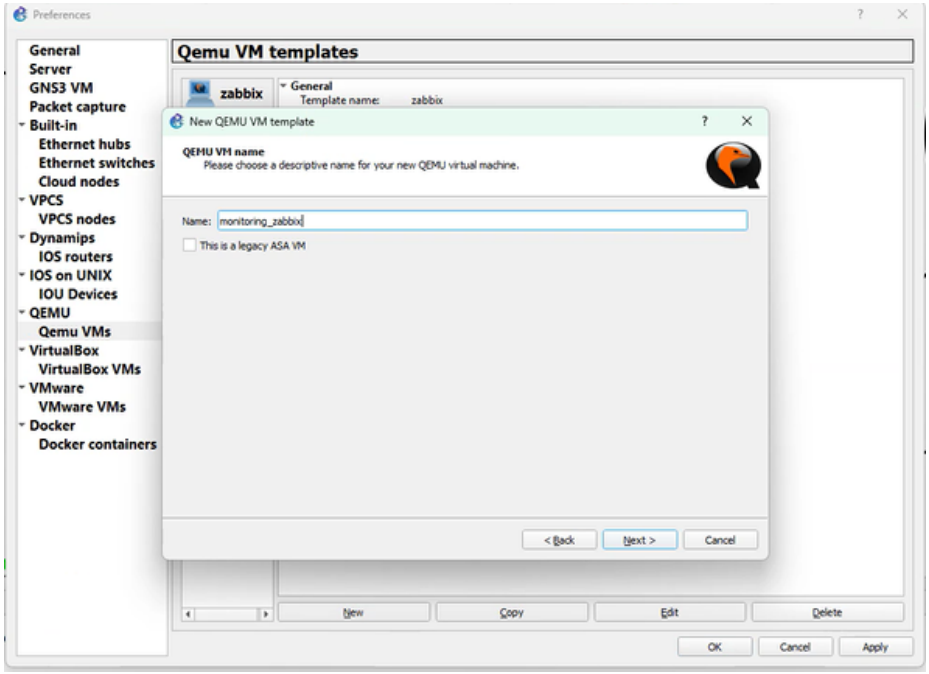
Durante la creación de la máquina virtual, es necesario especificar el entorno en el que se ejecutará. En este caso, se selecciona la opción *Run the QEMU VM on the GNS3 VM*, tal como se muestra en la Figura 75. Esta configuración permite que la máquina virtual se ejecute dentro de la GNS3 VM, garantizando una mejor integración con el entorno así como compatibilidad y un rendimiento más eficiente durante la simulación de red.

Figura 75. Selección de ejecución de la VM QEMU en la GNS3 VM.



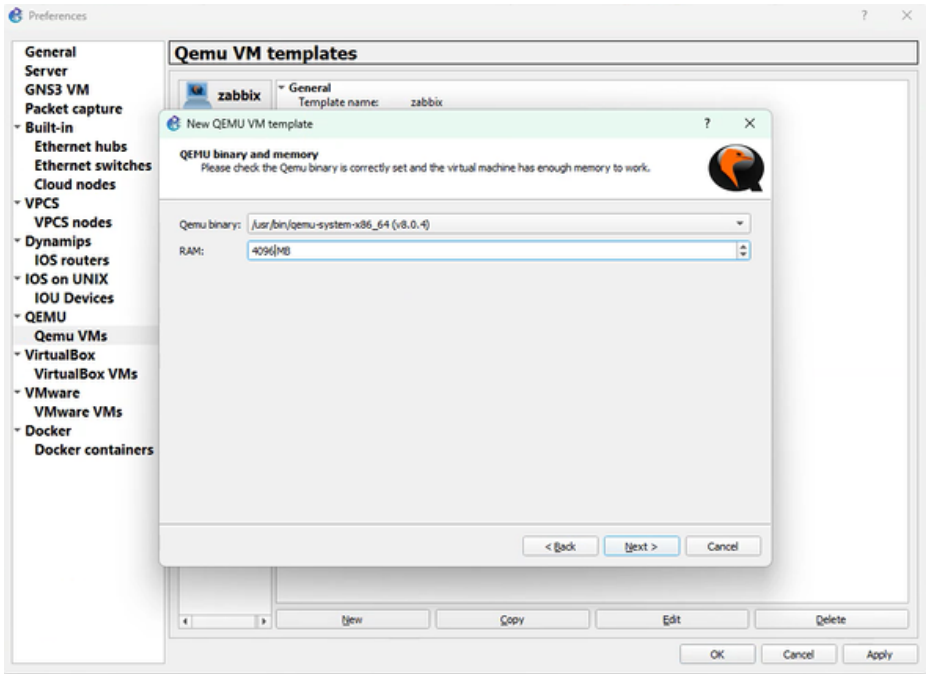
Nota. Elaboración propia. Se configura la máquina virtual para correr dentro de la GNS3 VM para garantizar compatibilidad.

Figura 76. Asignación de nombre a la nueva máquina virtual QEMU.



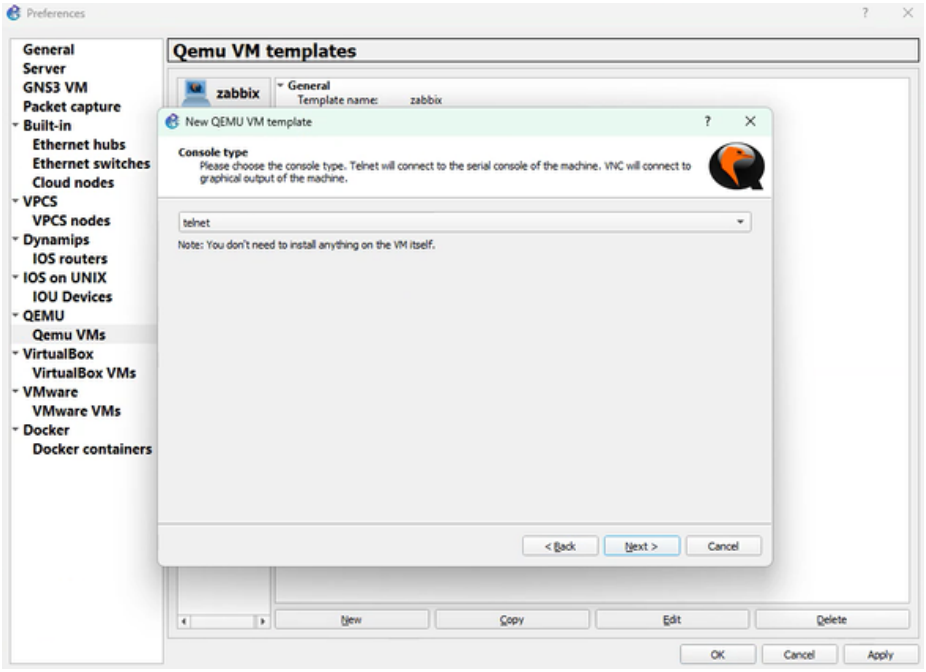
Nota. Elaboración propia. Se define el nombre para identificar la máquina virtual dentro de GNS3.

Figura 77. Configuración de binario QEMU y asignación de memoria RAM.



Nota. Elaboración propia. Se selecciona el binario QEMU correspondiente y se asignan 4096 MB de RAM.

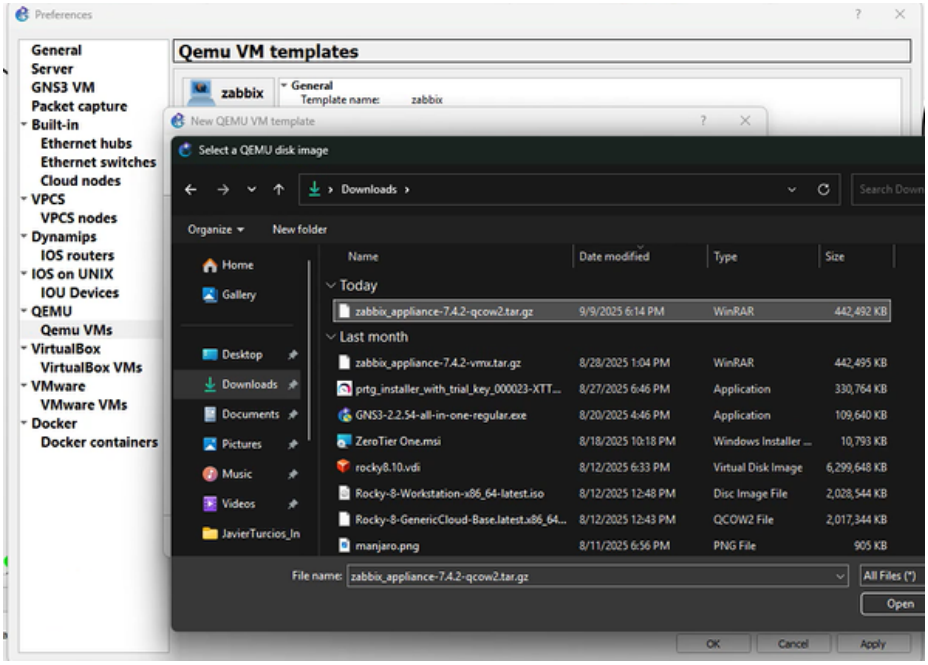
Figura 78. Selección del tipo de consola para la VM QEMU.



Nota. Elaboración propia. Se elige Telnet como método de acceso a la consola, para gestión directa de la máquina virtual desde GNS3.

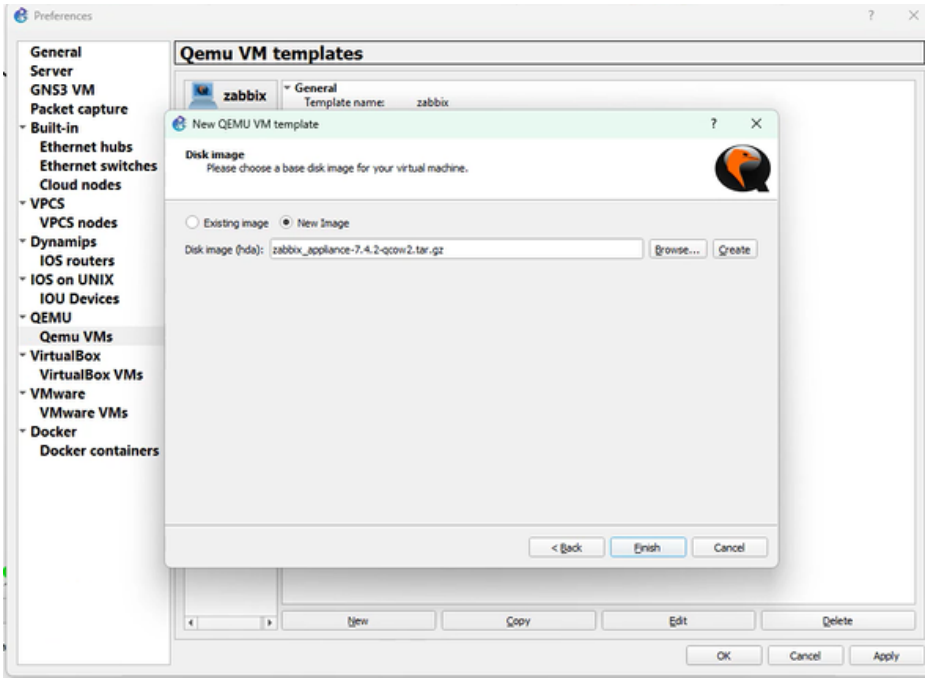
Se selecciona Telnet como tipo de consola, lo que permite gestionar la VM directamente desde la interfaz de GNS3 sin requerir configuraciones adicionales. Facilitando la interacción con el sistema operativo de la máquina virtual, posibilitando la conexión inmediata a través de la consola integrada en GNS3.

Figura 79. Adición de la imagen base para la VM QEMU en GNS3.



Nota. Elaboración propia. Se selecciona la opción New Image y se carga el archivo correspondiente.

Figura 80. Finalización de la adicción de la VM QEMU con la imagen de Zabbix.



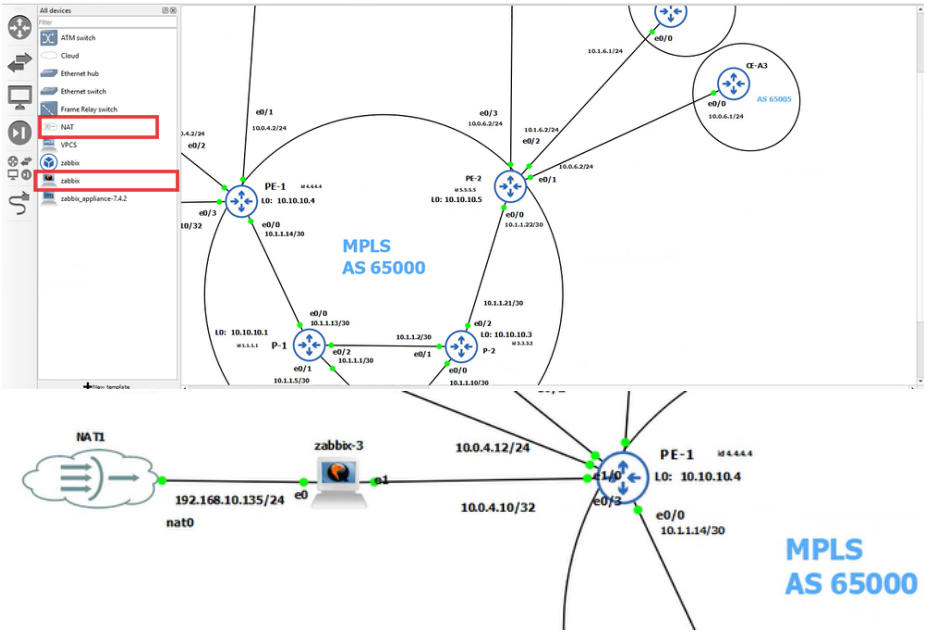
Nota. Elaboración propia. Se carga la imagen zabbix_appliance-7.4.2.qcow2 y se completa, quedando lista la máquina virtual para su despliegue en GNS3.

Para permitir la comunicación entre la red virtual implementada en GNS3 y Zabbix, se añade un nodo NAT (*Network Address Translation*). Se conectó a la interfaz Ethernet0 del servidor, asignándole la dirección 192.168.10.135/24.

A través del nodo NAT, se provee acceso hacia redes externas, como Internet o el sistema operativo de zabbix, sin exponer la topología MPLS virtualizada. A través de NAT, el tráfico generado desde el entorno GNS3 es encapsulado y traducido a una dirección IP del host físico, lo que permite realizar tareas esenciales como la descarga de paquetes y actualizaciones necesarias para el sistema Zabbix, la sincronización horaria mediante NTP y la comunicación con los *script*s de automatización ejecutados fuera del entorno emulado.

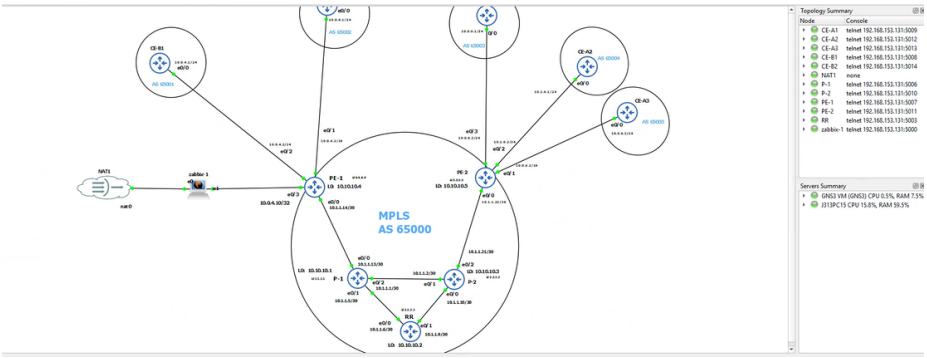
La interfaz NAT actúa como un puente controlado entre la red virtualizada e internet, manteniendo la seguridad y el aislamiento lógico de la infraestructura MPLS.

Figura 81. Inserción de elementos NAT y Zabbix en la topología de GNS3.



Nota. Elaboración propia. Se añaden los nodos NAT y la máquina virtual Zabbix a la red MPLS.

Figura 82. Integración de Zabbix y NAT a la topología MPLS en GNS3.

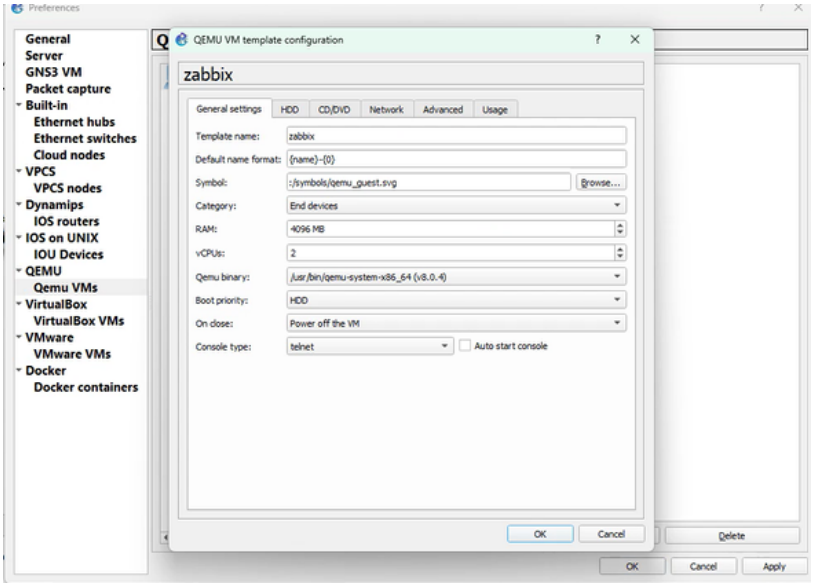


Nota. Elaboración propia. El nodo Zabbix se conecta de forma directa al router PE-1 y al nodo NAT para acceso a Internet.

9.2. Configuración de la máquina virtual Zabbix Appliance

En la figura 83 se muestra la ventana de configuración general de la máquina virtual Zabbix dentro de GNS3. En esta sección se establecen los parámetros básicos de la instancia, tales como el nombre de la VM, la cantidad de memoria asignada (4096 MB), el número de vCPUs y el tipo de consola que se empleará para su administración.

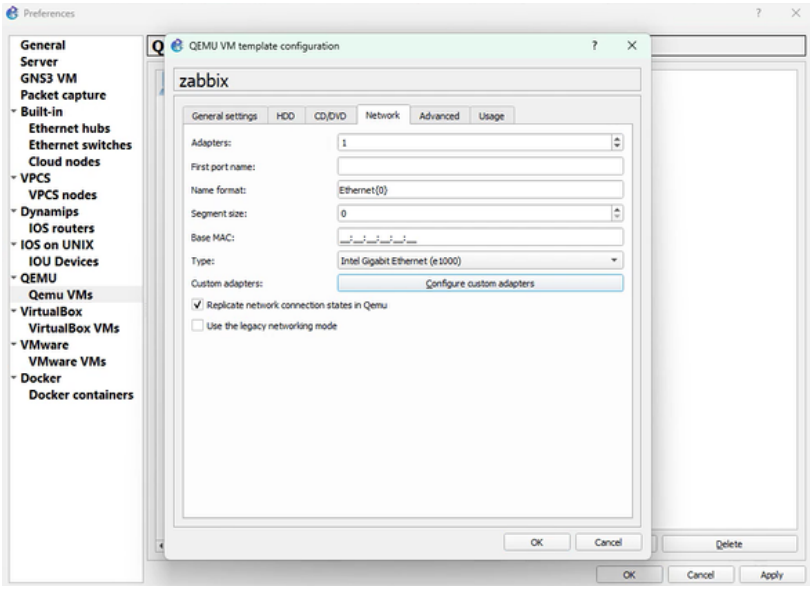
Figura 83. Configuración general de la máquina virtual Zabbix en QEMU.



Nota. Elaboración propia. Se definen parámetros principales como nombre, memoria (4096 MB), número de vCPUs y tipo de consola en GNS3.

Para la configuración de red, se asignan dos adaptadores de red a la máquina virtual Zabbix. El primer adaptador (eth0) se conecta al nodo NAT para proporcionar acceso a Internet, mientras que el segundo adaptador (eth1) se conecta directamente al router PE-1 del core MPLS, lo que permite la monitorización de los dispositivos dentro de la red.

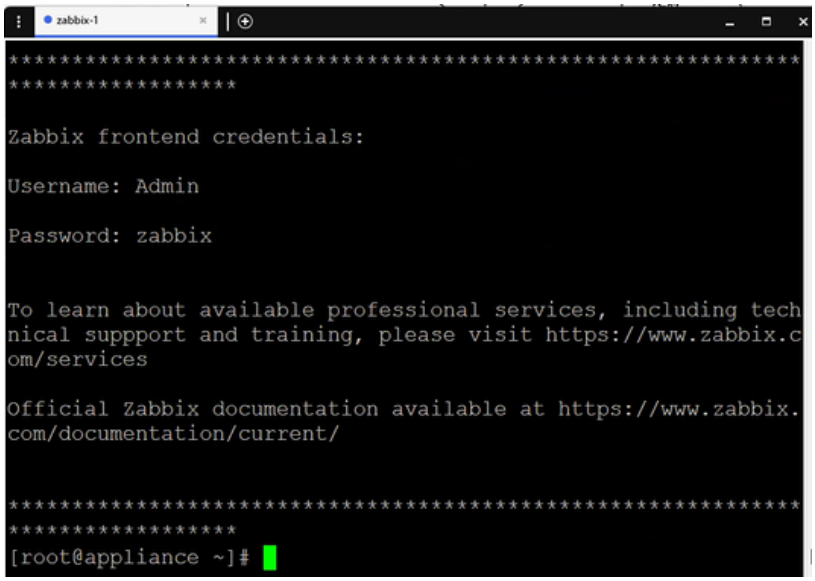
Figura 84. Configuración del adaptador de red.



Nota. Elaboración propia. Se asigna un adaptador Intel Gigabit Ethernet (e1000) como interfaz principal.

Una vez completada la configuración, se procede a iniciar la máquina virtual desde el entorno de GNS3. El acceso se realiza mediante Telnet, utilizando la consola integrada de GNS3. Para la autenticación inicial, se emplean las credenciales predeterminadas proporcionadas por la documentación oficial de Zabbix (admin = root/ password = zabbix).

Figura 85. Inicio de sesión en la consola de Zabbix Appliance.



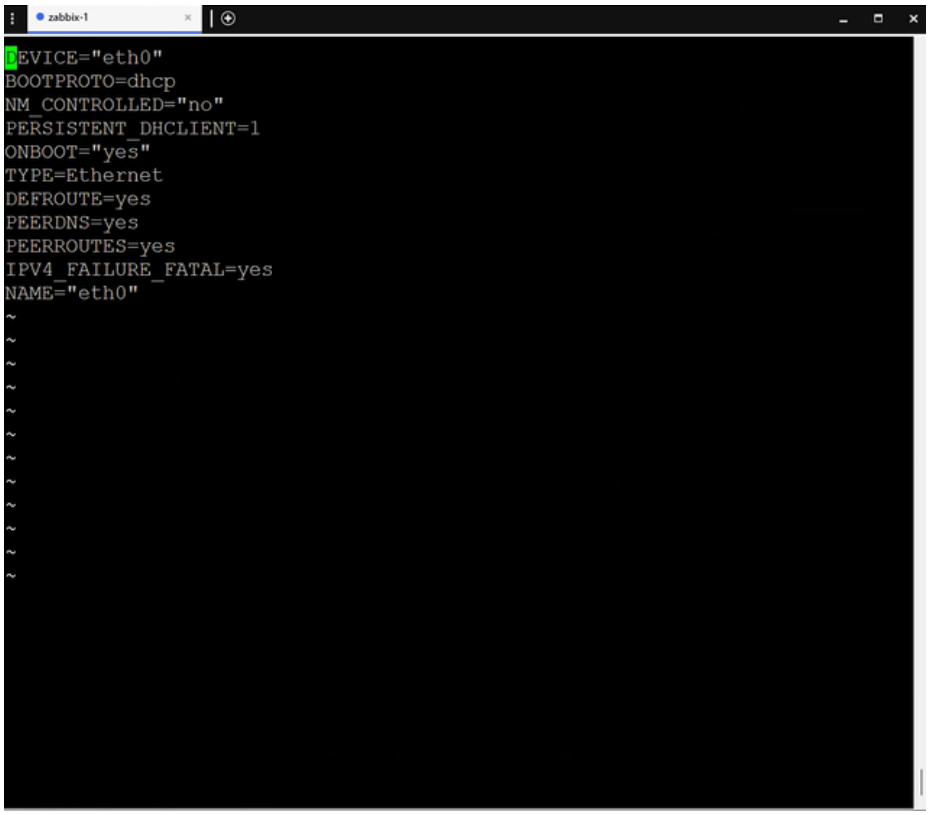
Nota. Elaboración propia. Se accede a la máquina virtual Zabbix con las credenciales predeterminadas (admin = root/ password = zabbix), habilitando la gestión inicial del sistema de monitoreo.

Se procede a configurar el adaptador de red principal (eth0) de la máquina virtual Zabbix. Para ello, se accede al archivo correspondiente dentro del sistema mediante el comando:

vi /etc/sysconfig/network-*script*s/ifcfg-eth0

En este archivo se definen los parámetros básicos de la interfaz, estableciendo el uso del protocolo DHCP. Esta configuración garantiza que la máquina virtual obtenga dinámicamente los parámetros de red necesarios, como dirección IP, puerta de enlace y DNS, lo que permite la conectividad del servidor Zabbix hacia el entorno NAT y, por consiguiente, su acceso a Internet para la descarga de paquetes, actualizaciones del sistema e integración de Inteligencia Artificial.

Figura 86. Configuración de la interfaz de red eth0 en Zabbix Appliance.



Nota. Elaboración propia. Se define la interfaz eth0 con protocolo DHCP y arranque automático, lo que asegura la obtención dinámica de parámetros de red para la conectividad del sistema.

El adaptador de red secundario (eth1) se configura con una dirección IP estática, destinada a la red interna de monitoreo. Para ello, se crea al archivo de configuración correspondiente mediante el comando:

vi /etc/sysconfig/network-*script*s/ifcfg-eth1

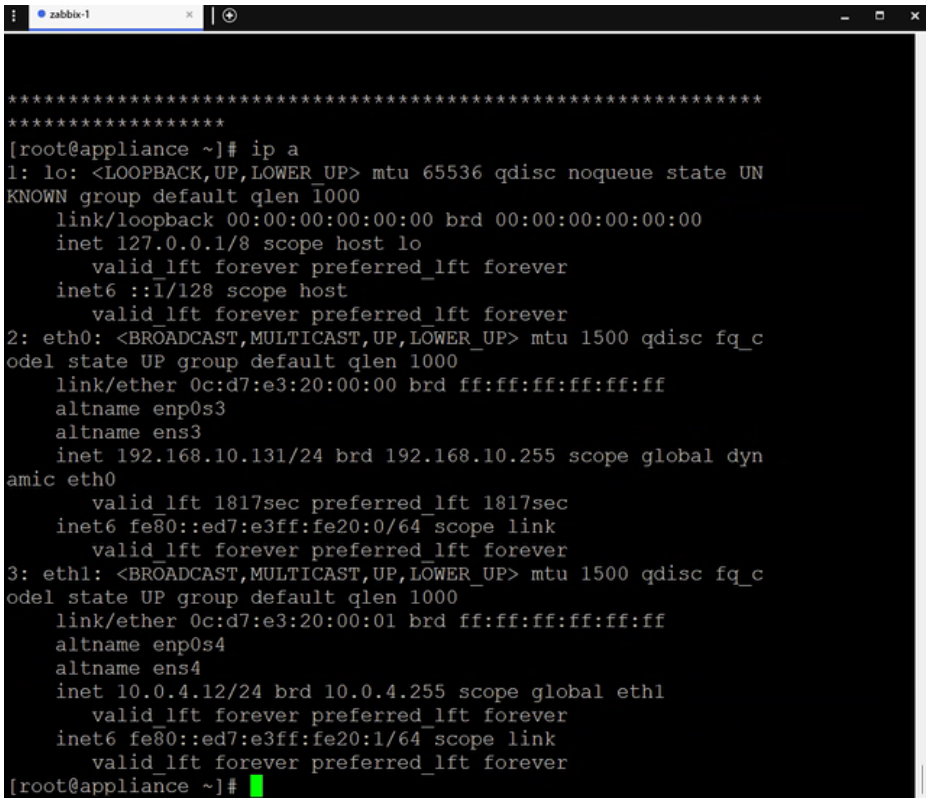
En este archivo se definen de forma manual los parámetros de red, asignando una IP fija y una máscara de subred y activando el arranque automático de la interfaz. Esta configuración permite establecer una conexión estable y permanente entre el servidor Zabbix y los dispositivos o nodos que se requiere sean monitoreados dentro de la topología.

Figura 87. Configuración de la interfaz de red eth1 en Zabbix Appliance.

Nota. Elaboración propia. Se asigna dirección IP estática (10.0.4.12/24), puerta de enlace con IP estática a eth1, lo que asegura la conectividad del servidor Zabbix con la red MPLS.

Se muestra la configuración final de la red utilizando el comando ip a.

Figura 88. Visualización de la configuración de red en Zabbix Appliance.



Nota. Elaboración propia. El comando ip a muestra las interfaces activas, lo que confirma la asignación de IP dinámica en eth0 y estática en eth1 para la integración con la red MPLS.

Se configura la interfaz ethernet 0/3 en el *router* PE-1 dentro de la misma red en la que se configuró el adaptador de red eth1 en Zabbix. Es importante Incorporación de esta interfaz al protocolo OSPF para que la máquina de monitoreo pueda comunicarse con todos los dispositivos del core MPLS.

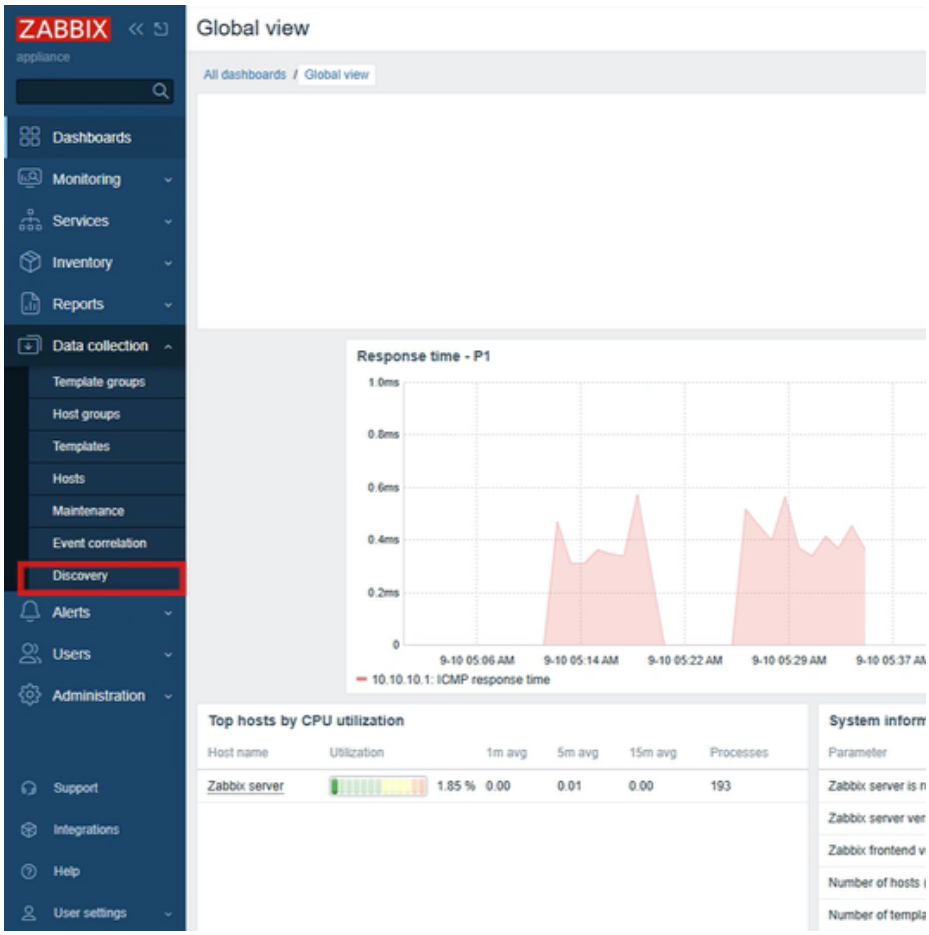
Figura 89. Configuración de la interfaz Ethernet0/3 en el PE-1.

Nota. Elaboración propia. Se asigna la IP 10.0.4.10/24 y se habilita OSPF en el área 0, lo que permite que el servidor Zabbix alcance la red MPLS.

9.3. Incorporación de hosts para monitoreo en Zabbix frontend

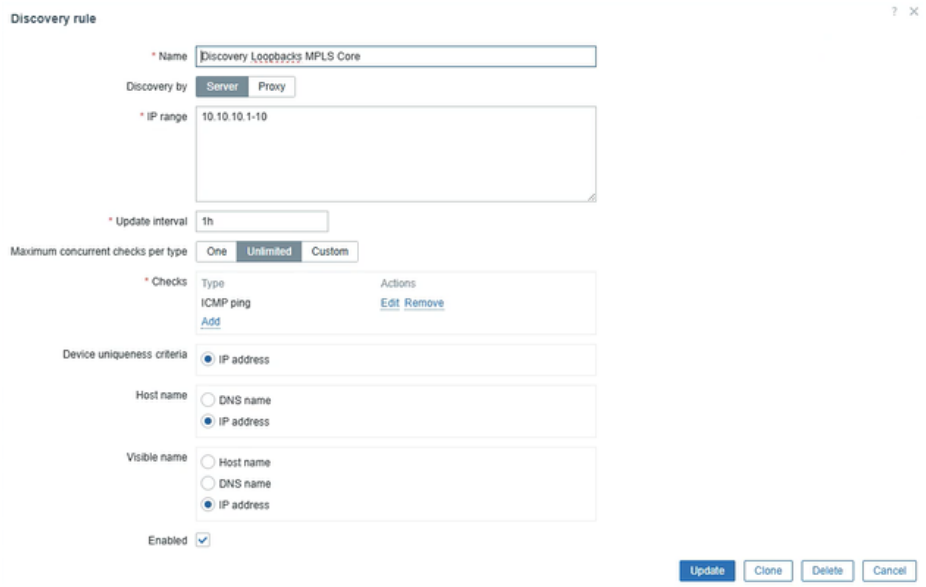
En la interfaz frontend de Zabbix se configura una regla de *auto discovery* con el objetivo de automatizar la incorporación de los hosts del núcleo MPLS y clientes al sistema de monitoreo. Esta regla utiliza las direcciones IP de las interfaces Loopback asignadas a cada router como criterio de identificación, lo que permite que el servidor Zabbix detecte y registre de forma automática los dispositivos activos dentro de la red, optimizando así la gestión y supervisión del entorno MPLS.

Figura 90. Creación de una regla de descubrimiento en Zabbix.



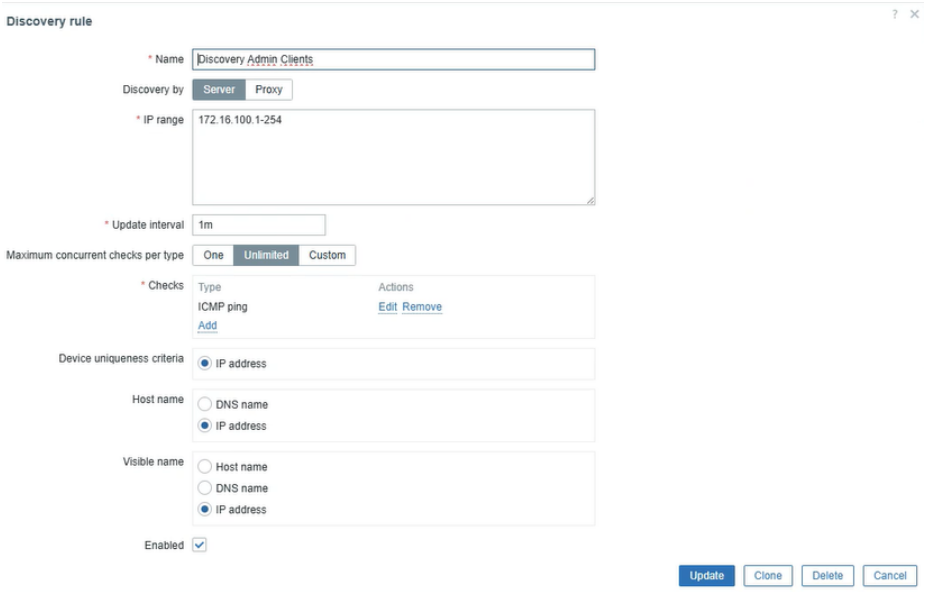
Nota. Elaboración propia. Desde el menú Discovery se configuran reglas que permiten detectar automáticamente hosts y dispositivos en la red MPLS para su monitoreo.

Figura 91. Configuración de la regla de descubrimiento en Zabbix.



Nota. Elaboración propia. Se define una regla para descubrir las direcciones de loopback del core MPLS mediante ICMP, con un rango de IP y un intervalo de actualización de una hora.

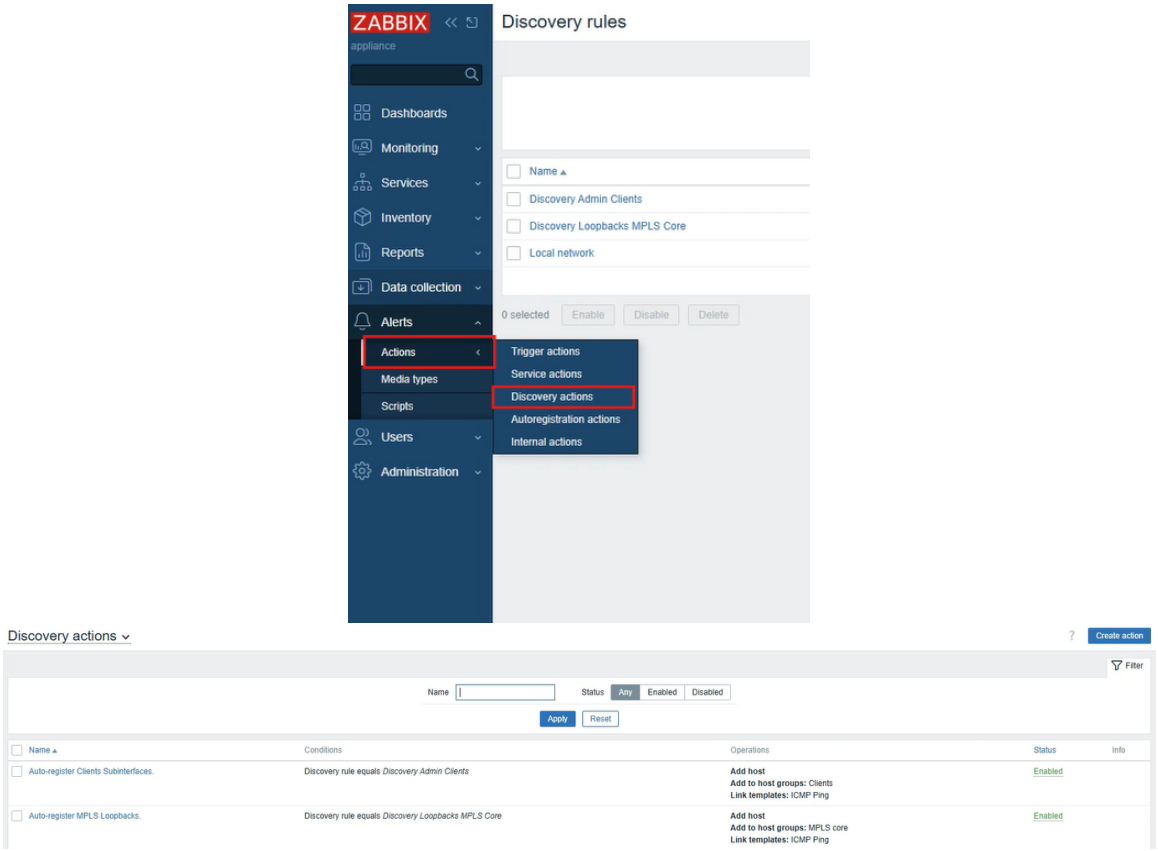
Figura 92. Regla de descubrimiento para clientes en Zabbix.



Nota. Elaboración propia. Se configura una regla Discovery Admin Clients con el rango 172.16.100.1-254, correspondiente a las subinterfaces asignadas para los clientes de la red MPLS.

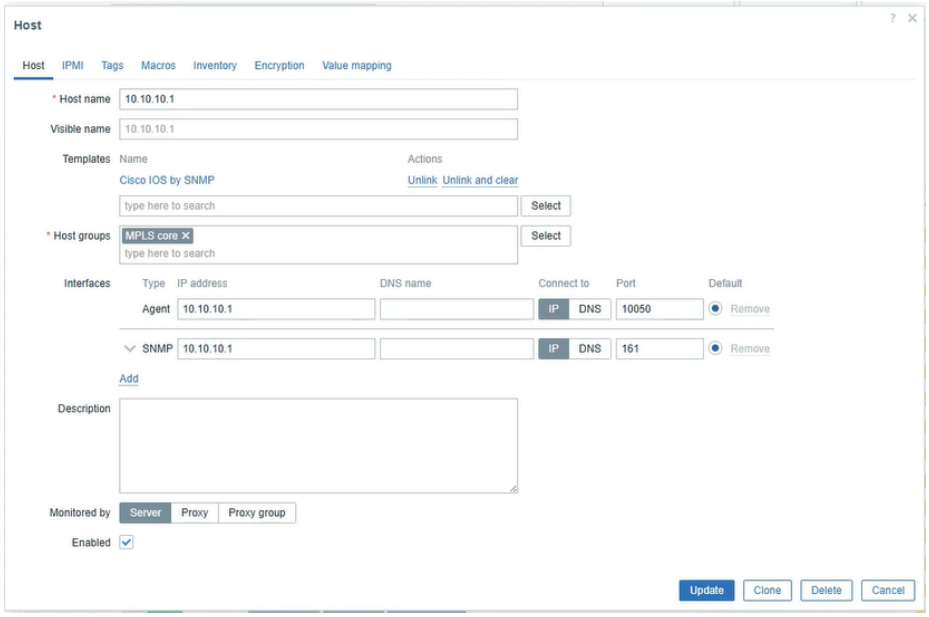
Posteriormente, se configuran las acciones de descubrimiento en el frontend de Zabbix, tal como se muestra en la figura 93. Estas acciones permiten definir las tareas que el sistema ejecutará automáticamente al detectar nuevos hosts dentro de la red. En este caso, desde el menú Actions, se establecen verificaciones ICMP (ping) hacia los dispositivos descubiertos, con el fin de confirmar su disponibilidad y mantener actualizada la base de datos de los equipos monitoreados en la red.

Figura 93. Configuración de acciones en Zabbix para los hosts descubiertos.



Nota. Elaboración propia. Desde el menú Actions se definen pings de verificación ICMP hacia los hosts detectados en la red.

Figura 94. Asociación de plantilla SNMP a un host en Zabbix.

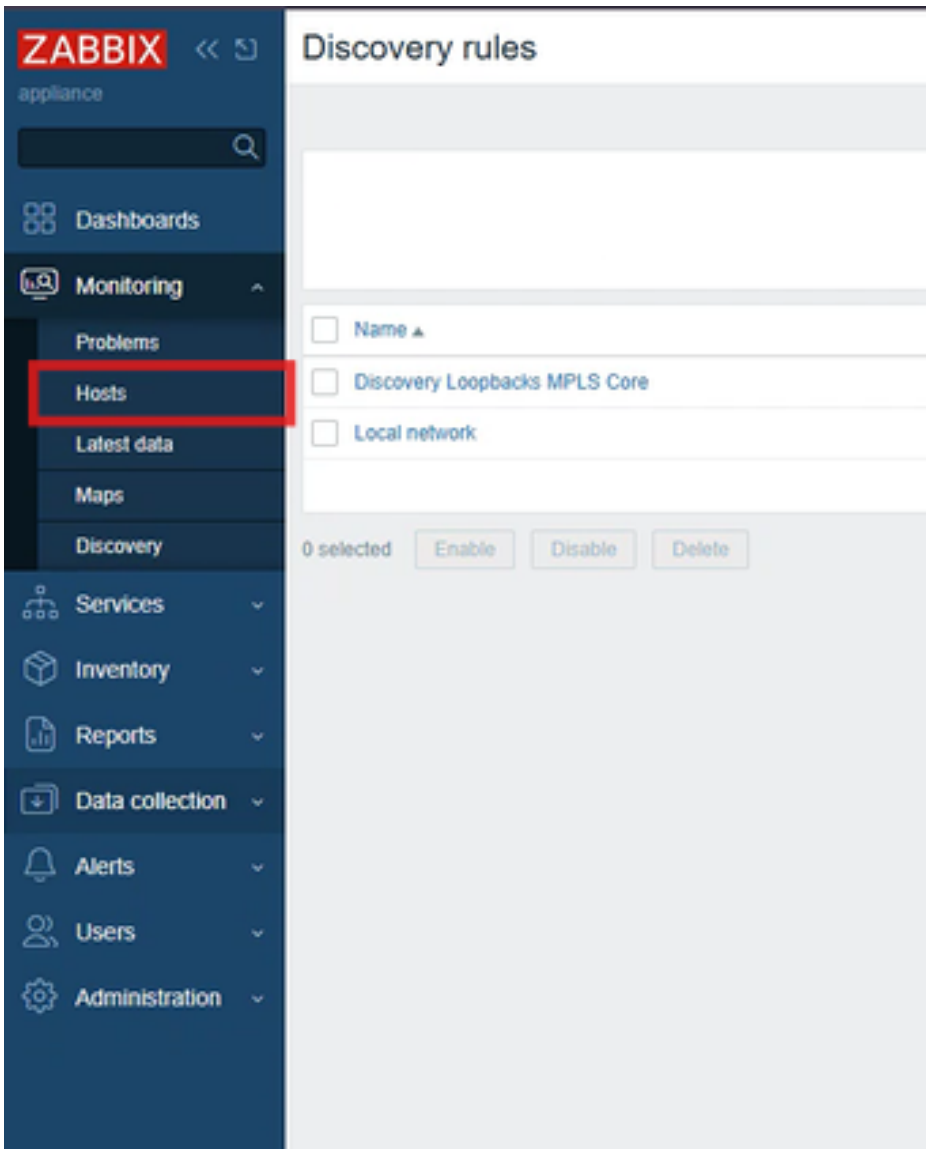


Nota. Elaboración propia. Se añade la plantilla Cisco IOS by SNMP a cada host descubierto, habilitando la recolección de métricas y el monitoreo detallado de los equipos.

Una vez definidas las reglas y acciones de descubrimiento, se procede a la verificación de los hosts detectados en el entorno de monitoreo. Desde el menú Monitoring > Hosts es posible acceder a la lista de dispositivos identificados automáticamente por las reglas configuradas en Zabbix.

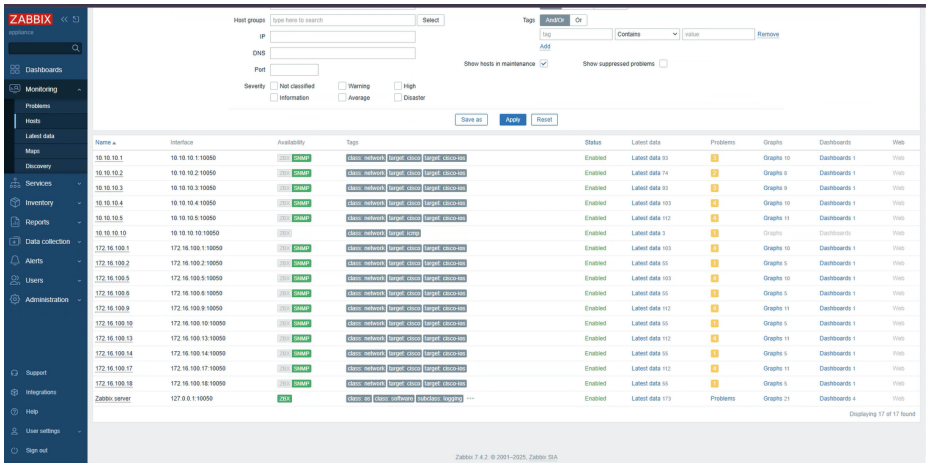
En la figura 96 Se presentan el listado completo de hosts descubiertos, donde el sistema muestra tanto los equipos pertenecientes al núcleo MPLS como los clientes conectados (A, B). Esta visualización permite confirmar la correcta operación del proceso de autodetección y facilita la supervisión centralizada de todos los elementos de la red.

Figura 95. Acceso a la sección de hosts descubiertos en Zabbix.



Nota. Elaboración propia. Desde el menú Monitoring > Hosts se visualizan los dispositivos detectados por las reglas de descubrimiento configuradas.

Figura 96. Listado de hosts descubiertos en la red mediante Zabbix.

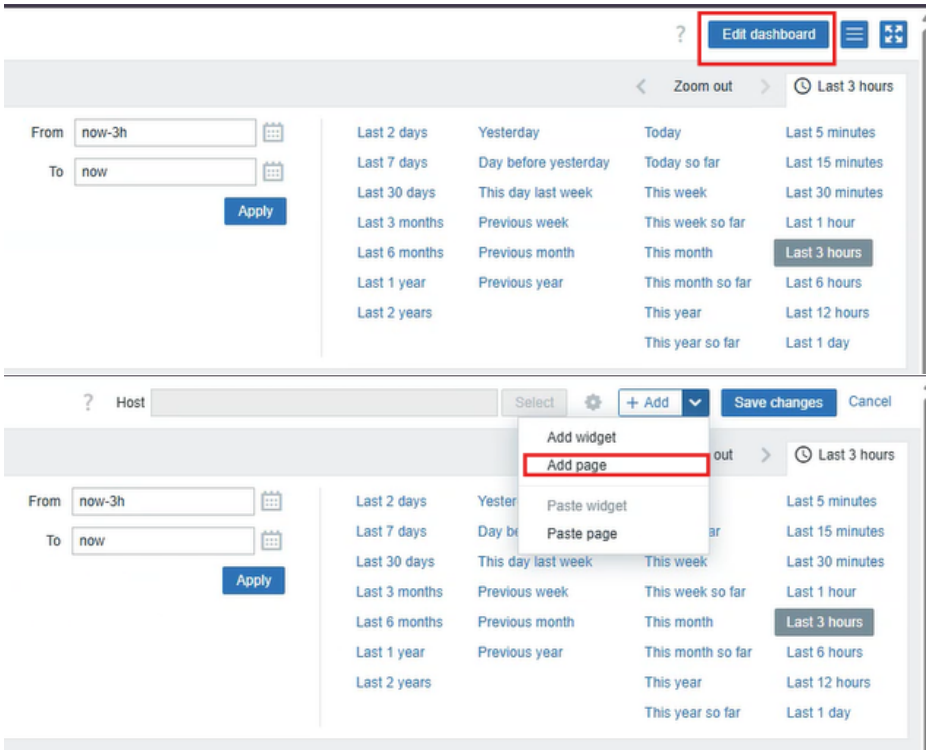


Nota. Elaboración propia. El sistema muestra los dispositivos del core MPLS y clientes detectados.

9.4. Incorporación de gráficas de monitoreo para hosts en Zabbix frontend

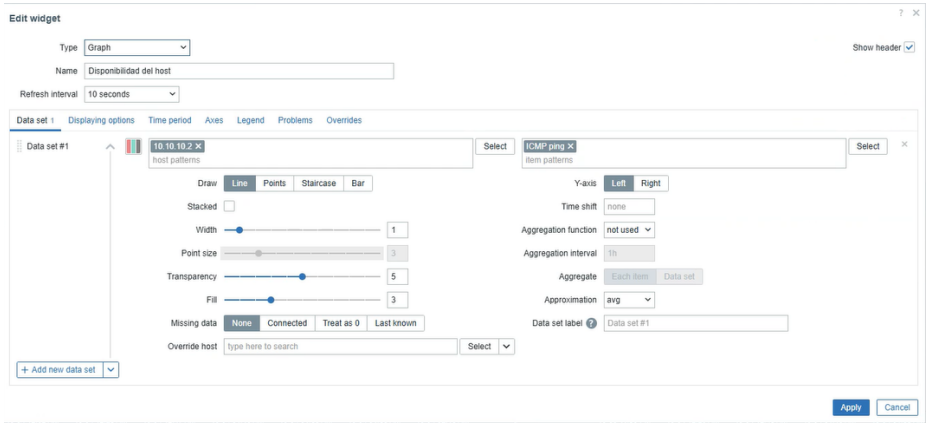
Para complementar el monitoreo de los dispositivos descubiertos, se procede a Incorporación de gráficas de supervisión individual por host dentro del panel principal de Zabbix. Desde el menú *Dashboard* se selecciona la opción *Edit* para modificar el panel de control y crear una página específica para cada dispositivo monitoreado. Esta configuración permite visualizar métricas en tiempo real, tales como tráfico de red o disponibilidad, facilitando el análisis detallado del comportamiento de cada equipo dentro de la red.

Figura 97. Edición del panel de control en Zabbix para Incorporación de páginas por host.



Nota. Elaboración propia. Desde la opción Edit dashboard se crea una página individual para cada host.

Figura 98. Configuración de widget gráfico en el dashboard de Zabbix.



Nota. Elaboración propia. Se añade un widget por host y métrica (ej. ICMP ping) con intervalo de actualización de 10 segundos, lo que permite visualizar en tiempo real la disponibilidad y desempeño de cada dispositivo.

Se repite el proceso para cada Host y métrica que se desee visualizar dentro de Zabbix. Podemos copiar y pegar la página original modificando únicamente a qué host corresponde cada métrica.

Figura 99. Dashboard de Zabbix con páginas individuales por host.

Nota. Elaboración propia. El panel muestra métricas en tiempo real de cada equipo, incluyendo disponibilidad, latencia, pérdida de paquetes y tráfico, facilitando el monitoreo detallado del entorno MPLS.

CAPÍTULO 10

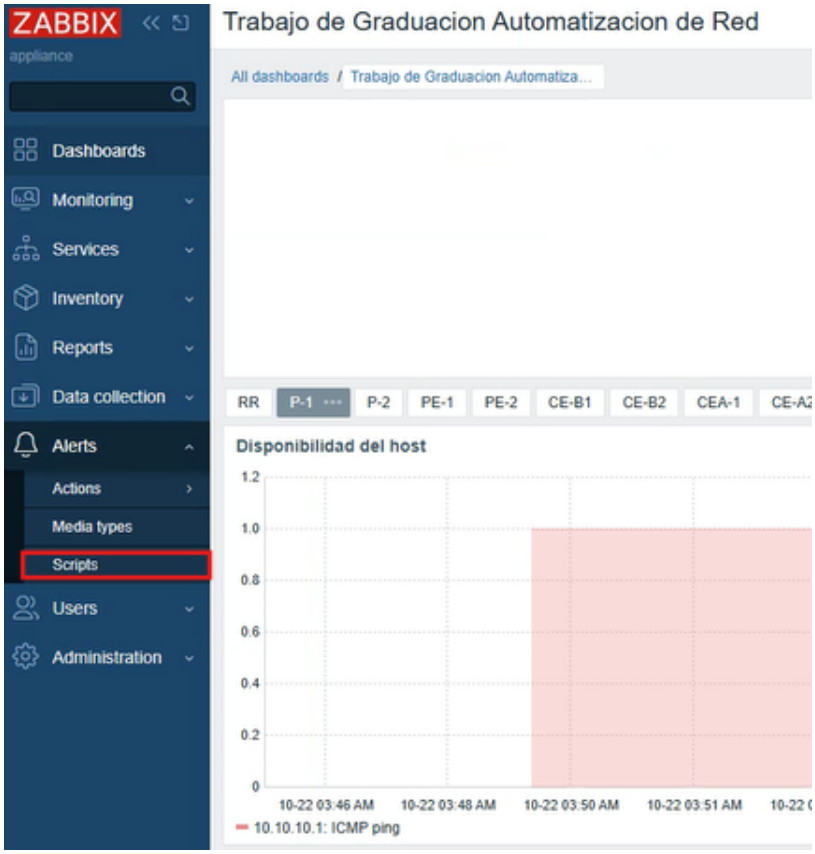
Implementación de inteligencia artificial (Google Gemini)

En este capítulo, se presentan la implementación del *script* de integración con Google Gemini descrito en el Anexo 32. De esta forma, se adapta el enfoque de inteligencia artificial al entorno de monitoreo, lo que permite ejecutar análisis en función de los problemas detectados.

10.1. Configuración del *script* en Zabbix

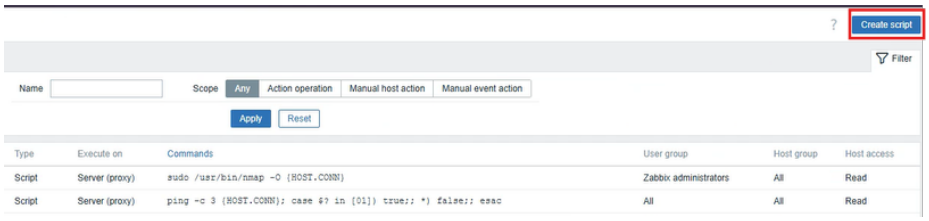
Desde el panel de monitoreo de Zabbix se accede al menú *Scripts*, donde se gestionan las automatizaciones que interactúan con los equipos supervisados.

Figura 100. Panel de monitoreo de Zabbix con acceso a *scripts* personalizados.



Nota. Elaboración propia. Desde el menú *scripts* se gestionan las automatizaciones que ejecutan acciones sobre los equipos monitoreados.

Figura 101. Creación de un nuevo *script* en Zabbix.

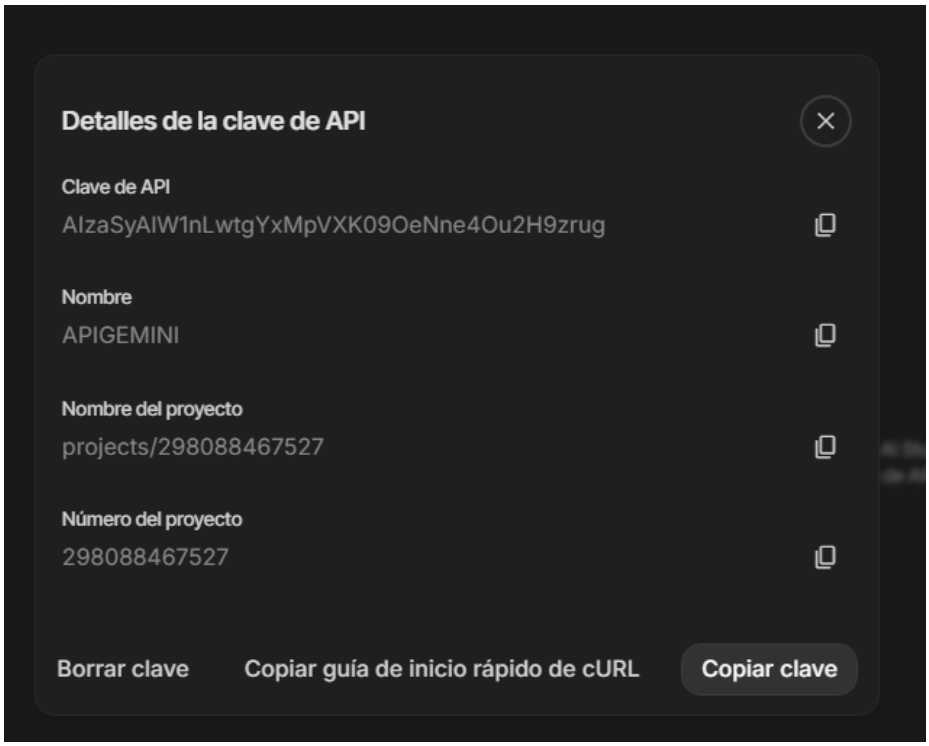


Nota. Elaboración propia. Crear el *script* para Incorporación de el evento personalizado para ejecutar el *script* de Google Gemini.

Para establecer la comunicación entre Zabbix y el modelo de inteligencia artificial Google Gemini, se requiere la generación de una clave API desde la plataforma Google AI Studio. Esta clave actúa como identificador seguro que permite a Zabbix autenticarse y enviar solicitudes al modelo de IA.

En la figura 102 se muestra la creación de la clave API dentro de la consola de Google AI Studio. Una vez generada, esta se incorpora en la configuración del *script* dentro de Zabbix, asociándola al evento del *trigger*. Así, el sistema puede invocar al modelo Gemini para procesar información en tiempo real y apoyar la toma de decisiones en el monitoreo de la red.

Figura 102. Creación de clave API para la integración de Zabbix con Google Gemini.

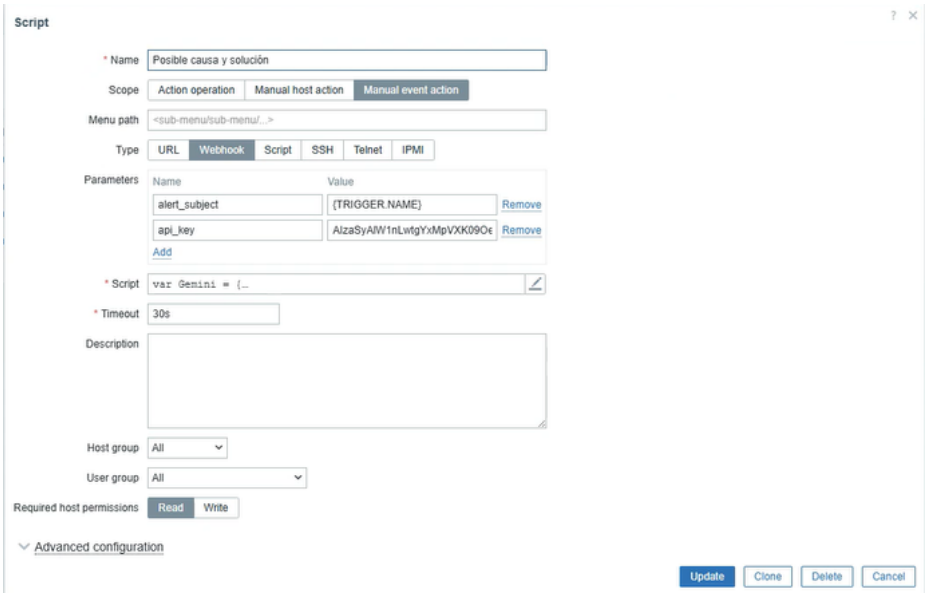


Nota. Elaboración propia. Google AI Studio genera una clave API utilizada por Zabbix para establecer conexión con el modelo Gemini.

Se crea un nuevo *script* del tipo Webhook para permitir el envío de información desde Zabbix hacia un servicio externo mediante peticiones HTTP, la opción Webhook habilita la estructura necesaria para enviar los datos en formato JSON a un endpoint remoto.

Se definen parámetros esenciales como el método POST, la URL del servicio de Google Gemini, y las cabeceras de autenticación, en donde se incluye la API Key que valida la comunicación.

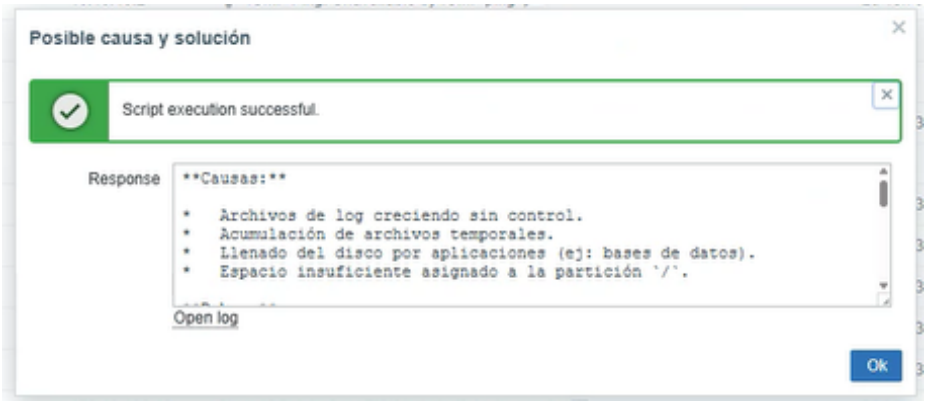
Figura 103. Configuración del *script* para análisis automático de fallos con Google Gemini en Zabbix.



Nota. Elaboración propia. *script* del tipo Webhook que envía el *trigger* al modelo Gemini mediante una API Key.

Es posible referenciar un evento o problema dentro de Zabbix para que la Inteligencia Artificial de Google Gemini genere una respuesta automatizada de diagnóstico o recomendación. El modelo procesa los datos y devuelve una respuesta contextualizada, que puede incluir posibles causas del fallo, sugerencias de verificación o acciones correctivas recomendadas como se muestra en el anexo 34

Figura 104. Ejecución del *script* "Posible causa y solución" en Zabbix con respuesta generada por Google Gemini.



Nota. Elaboración propia. La herramienta de IA analiza el evento detectado y devuelve un diagnóstico con causas probables y posibles soluciones.

CAPÍTULO 11

Exportación de datos

Dentro de la consola del QEMU correspondiente al servidor Zabbix, se accede al archivo principal de configuración del servicio para habilitar la función de exportación de datos. Este procedimiento se realiza mediante el archivo de configuración utilizando el editor de texto vi:

Código 13. Edición del archivo de configuración del servidor Zabbix para habilitar la exportación de datos

```
vi /etc/zabbix/zabbix_server.conf
```

Nota. Elaboración propia.

Se realizan los ajustes mínimos necesarios en el archivo de configuración del servidor de Zabbix para habilitar la exportación de datos. En este caso, se establece el uso del protocolo TCP para la conexión con la base de datos MySQL local, estos parámetros aseguran que el servidor Zabbix pueda escribir y consultar registros de manera continua:

Código 14. Configuración de parámetros de conexión a la base de datos y exportación en Zabbix Server

```
DBHost=127.0.0.1
DBName=zabbix
DBUser=zabbix
DBPassword=contrasena

ExportDir=/var/lib/zabbix/export
ExportFileSize=5M
ExportType=history,trends,events
```

Nota. Elaboración propia.

Se crea una carpeta dedicada para la exportación de datos del servidor Zabbix, para permir mantener una estructura organizada dentro del sistema de archivos, se asignan los permisos adecuados a dicha carpeta para garantizar que el usuario bajo el cual se ejecuta el servicio de Zabbix posea privilegios de lectura y escritura:

Código 15. Creación del directorio de exportación y asignación de permisos en Zabbix

```
sudo mkdir -p /var/lib/zabbix/export
sudo chown zabbix:zabbix /var/lib/zabbix/export
```

Nota. Elaboración propia.

Se arranca el servicio de base de datos MySQL empleando mysqld, configurado para operar sobre el protocolo TCP en la dirección 127.0.0.1. Así establecemos la comunicación local entre el servidor Zabbix y la base de datos.

Código 16. Inicio del servicio MySQL y acceso a la consola de MySQL

```
sudo systemctl enable --now mysqld
systemctl restart zabbix-server
mysql -u root
```

Nota. Elaboración propia.

Dentro del entorno de MySQL, se lleva a cabo la creación de la base de datos y del usuario dedicado para el servidor Zabbix, tal como se muestra a continuación. Se define una base de datos denominada *zabbix* con codificación UTF-8:

Código 17. Creación de la base de datos y usuario para Zabbix en MySQL

```

CREATE DATABASE zabbix CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;
DBHost=127.0.0.1
CREATE USER 'zabbix'@'127.0.0.1' IDENTIFIED BY 'contrasena';
GRANT ALL PRIVILEGES ON zabbix.* TO 'zabbix'@'127.0.0.1';
FLUSH PRIVILEGES;
```

Nota. Elaboración propia.

Se crea el usuario zabbix restringido a la dirección 127.0.0.1, asignándole una contraseña segura y concediéndole todos los privilegios sobre la base de datos. Finalmente, se ejecuta el comando FLUSH PRIVILEGES; para aplicar los cambios realizados y habilitar el acceso del servidor de monitoreo a la base de datos.

Una vez creada la base de datos y configurado el usuario correspondiente, se procede a importar el esquema inicial de Zabbix en MySQL:

Código 18. Importación del esquema inicial de Zabbix en la base de datos MySQL

```
    zcat /usr/share/zabbix/sql-\textit{scripts}/mysql/server.sql.gz | mysql -h
    ↪ 127.0.0.1 -u zabbix -p zabbix
```

Nota. Elaboración propia.

Este procedimiento inicializa correctamente la base de datos y la deja lista para almacenar los datos recolectados por los agentes y las métricas del sistema.

Se inicia el servicio principal del servidor Zabbix, ya que de forma predeterminada únicamente se encuentra activo el agente Zabbix (puerto 10050). El servidor Zabbix escucha en el puerto 10051, el cual es utilizado también por herramientas complementarias como *zabbix sender* para la transmisión de datos desde *scripts* o procesos externos.

Código 19. Reinicio y verificación del estado del servicio Zabbix Server

```
sudo systemctl status zabbix-server -n 40
```

Nota. Elaboración propia.

Se reinicia el servicio zabbix-server, verificar su estado y confirmar que el puerto correspondiente esté correctamente habilitado.

Código 20. Verificación de monitoreo en tiempo real de la creación y crecimiento de archivos de exportación en Zabbix

```
ls -lh /var/lib/zabbix/export
watch -n 2 'ls -lh /var/lib/zabbix/export'
tail -f $(ls -t /var/lib/zabbix/export/* | head -1)
```

Nota. Elaboración propia.

Estos comandos permiten observar tanto la creación de nuevos archivos como el crecimiento progresivo de su tamaño, verificando así el correcto funcionamiento del mecanismo de exportación configurado previamente. Se supervisa el archivo más reciente en formato NDJSON, cuyo contenido se actualiza dinámicamente a medida que Zabbix continúa exportando información

Debería verse una respuesta como la siguiente:

Figura 105. Exportación de datos en tiempo real desde Zabbix en formato NDJSON.

Nota. Elaboración propia. La información de métricas se registra continuamente en archivos NDJSON.

CAPÍTULO 12

Servidor NFS

Con el propósito de permitir el intercambio eficiente de datos entre la máquina virtual principal de GNS3 y la máquina virtual anidada que ejecuta el entorno de supervisión y análisis, se implementó un servidor NFS dentro de la GNS3-VM basado en el reciente trabajo de graduación [30]. Este mecanismo de compartición de archivos proporciona una interfaz de red que permite a un cliente montar un sistema de archivos remoto como si se tratara de una unidad local, garantizando coherencia, persistencia y simplicidad en la transferencia de información entre procesos distribuidos.

12.0.1. Configuración del servidor NFS en la GNS3-VM

El servidor se implementó sobre un sistema operativo Linux que actúa como host de GNS3. Se creó el directorio raíz destinado al intercambio de información con el cliente anidado Zabbix-QCOW:

Código 21. Creación del directorio raíz para NFS en GNS3-VM

```
sudo mkdir -p /srv/share_gns3
sudo chown -R root:root /srv/share_gns3
```

Nota. Elaboración propia.

Posteriormente, se instalaron los servicios necesarios del kernel NFS y se habilitó el nfs-server para su ejecución automática al arranque del sistema.

Código 22. Instalación y activación del servidor NFS en GNS3-VM

```
sudo apt install -y nfs-kernel-server
sudo systemctl enable --now nfs-server
sudo systemctl restart nfs-server
```

```
sudo systemctl status nfs-server --no-pager
```

Nota. Elaboración propia.

El archivo de configuración /etc/exports se modificó para definir los directorios a exportar y las políticas de acceso:

Código 23. Configuración de exportación NFS en GNS3-VM

```
    /srv/share_gns3 192.168.153.0/24(rw,sync,no_root_squash)
```

Nota. Elaboración propia.

Esta configuración permite acceso de lectura y escritura (rw) a las máquinas dentro del segmento 192.168.153.0/24, empleando escritura sincrónica (sync) para garantizar la integridad de los datos y deshabilitando la traducción de privilegios (no_root_squash) para permitir la correcta ejecución de *scripts* administrativos dentro del entorno de pruebas. La política de exportación fue aplicada mediante el comando:

Código 24. Aplicación de la configuración de exportación NFS en GNS3-VM

```
    sudo exportfs -ra
```

Nota. Elaboración propia.

De forma complementaria, se habilitaron los servicios de red requeridos para el intercambio NFS (puertos 2049, 111 y mountd) en el firewall del sistema, lo que asegura que las conexiones provenientes del cliente anidado fueran aceptadas.

12.1. Configuración del cliente NFS en la máquina anidada (Zabbix-QCOW)

La máquina anidada que ejecuta el servicio Zabbix y los *scripts* de ingestión de datos fue configurada como cliente NFS, lo que permite el montaje persistente del sistema remoto. Se creó el punto de montaje local y se estableció la conexión con el servidor:

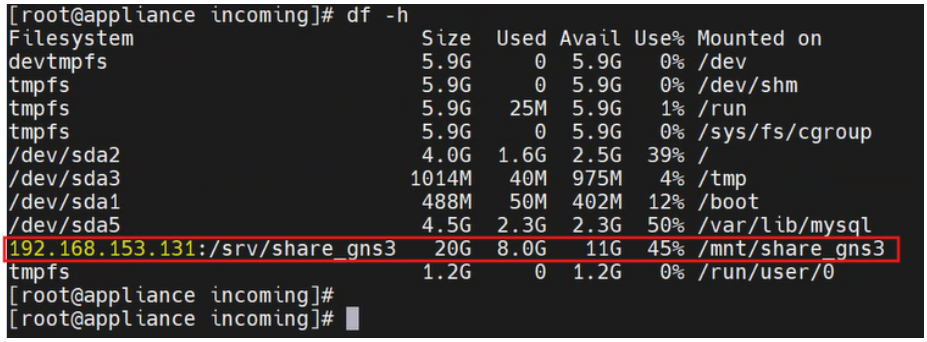
Código 25. Montaje del sistema de archivos NFS en la máquina anidada Zabbix-QCOW

```
sudo mkdir -p /mnt/share_gns3
sudo mount -t nfs 192.168.153.131:/srv/share_gns3 /mnt/share_gns3
```

Nota. Elaboración propia.

De esta forma, la estructura `/mnt/share_gns3/zabbix/incoming` quedó sincronizada con `/srv/share_gns3/zabbix/incoming` en la GNS3-VM, funcionando como un repositorio compartido para los registros y exportaciones de Zabbix. Verificamos que el cliente montó el NFS a través del comando df -h:

Figura 106. Verificación del montaje NFS en Zabbix-QCOW.



Nota. Elaboración propia. El comando df -h confirma que el sistema de archivos NFS está montado correctamente en la ruta designada.

12.2. Transferencia y depuración de datos

Debido a que el archivo history-history-syncer-1.ndjson generado por Zabbix crece continuamente en tiempo real, se implementó un proceso automatizado de rotación, copia y limpieza mediante el servicio logrotate. Este mecanismo genera "*snapshots*" horarios del archivo, los transfiere al servidor NFS y elimina las copias locales, evitando el crecimiento indefinido de la partición interna de la QCOW.

La siguiente configuración fue definida en /etc/logrotate.d/zbx-history-syncer:

Código 26. Configuración de logrotate para la gestión automática de archivos NDJSON en Zabbix-QCOW

```
/home/zabbix/export/history-history-syncer-1.ndjson { #archivo NDJSON gestionado por logrotate
hourly #rotación del archivo NDJSON generada cada hora
rotate 48 #mantiene 48 rotaciones antes de eliminar las más antiguas
missingok #no genera error si el archivo no existe
notifempty #evita rotar si el archivo está vacío
dateext #agrega extensión con fecha y hora a los archivos rotados
dateformat -%Y%m%d%H%M%S #define formato de fecha en el nombre (año, mes, día, hora, minuto, segundo)
copytruncate #copia y trunca sin interrumpir el proceso activo
su zabbix zabbix #ejecuta rotación bajo el usuario y grupo zabbix
postrotate #bloque de comandos a ejecutar después de rotar
    for f in /home/zabbix/export/history-history-syncer-1.ndjson-2*; do #recorre archivos rotados con prefijo 2*
        [ -e "$f" ] || continue #verifica existencia del archivo antes de procesar
        base="$(basename "$f")" #obtiene el nombre base del archivo
        tmp="/mnt/share_gns3/zabbix/incoming/.${base}.part" #ruta temporal en destino compartido
        final="/mnt/share_gns3/zabbix/incoming/${base}" #ruta final del archivo copiado
        /usr/bin/cp -f "$f" "$tmp" && /usr/bin/sync && /usr/bin/mv -f "$tmp" "$final" && /usr/bin/rm -f "$f" #copia, sincroniza, mueve y elimina original
```

```
        done #fin del bucle for
    end\textit{script} #fin del bloque postrotate
} #fin de la configuración logrotate
```

Nota. Elaboración propia.

Esta secuencia asegura que cada rotación copie y trunque el archivo vivo, liberando espacio en la QCOW, transfiera de forma atómica los *snapshots* al directorio compartido `/mnt/share_gns3/zabbix/incoming` y elimine los archivos locales una vez confirmada la copia.

El proceso se complementó con una tarea programada (cron) cada cinco minutos para garantizar la continuidad de la transferencia y limpieza:

Código 27. Tarea cron para la ejecución periódica de logrotate en Zabbix-QCOW

```
crontab -e
*/5 * * * * /usr/sbin/logrotate -f /etc/logrotate.d/zbx-history-syncer
    ↪ >/tmp/zbx-rotate.log 2>&1
```

Nota. Elaboración propia.

12.3. Conversión de archivos NDJSON a CSV dentro de la GNS3-VM

Con el objetivo de optimizar el análisis y el procesamiento de los datos exportados desde Zabbix, se realizó la compactación y la transferencia de los *snapshots* en formato NDJSON hacia un único archivo denominado "main NDJSON", alojado en el servidor NFS. Posteriormente, se desarrolló un *script* en Python encargado de convertir dicho archivo al formato CSV, el cual ofrece una amplia compatibilidad con herramientas de análisis de datos y modelos de aprendizaje automático.

Se implementó un conversor en Python "ndjson2csv.py", ubicado en /srv/share_gns3/zabbix/incoming/ndjson2csv.py, diseñado para transformar registros en formato NDJSON a un archivo CSV. Este proceso estandariza valores nulos y escalares, y extrae campos relevantes para construir filas con el encabezado [host, interface, name, value, clock, groups, path].

Código 28. *Script* de Python para la conversión de archivos NDJSON a CSV

```
cat > /srv/share_gns3/zabbix/incoming/ndjson2csv.py <<'PY' #crea el archivo
    ↪ ndjson2csv.py y escribe el contenido siguiente hasta PY
#!/usr/bin/env python3 #define el intérprete de Python 3 para ejecución directa
import sys, os, json, csv #importa módulos para sistema, archivos, JSON y CSV

def val(x): #función para convertir cualquier valor a texto legible o serializado
    if x is None: #si el valor es nulo
        return "" #retorna cadena vacía
    if isinstance(x, (int, float, bool, str)): #si es tipo básico
        return str(x) #convierte a cadena
```

```
    if isinstance(x, list): #si es una lista
        # Si hay objetos dentro, JSON compacto; si son escalares, a str
        parts = [] #lista temporal de elementos procesados
        for e in x: #recorre cada elemento de la lista
            if isinstance(e, (dict, list)): #si es objeto o lista anidada
                parts.append(json.dumps(e, separators=(',',':'), ensure_ascii=False))
    ↪ #serializa en JSON compacto
            else: #si es valor simple
                parts.append(str(e)) #convierte a texto
        return "|".join(parts) #une los valores
    if isinstance(x, dict): #si es un diccionario
        # Intenta algo legible; si no, JSON
        for k in ("host","name","value","ip"): #busca claves representativas
            if k in x and x[k] not in (None, ""): #si existen y no están vacías
                return str(x[k]) #usa su valor como texto
        return json.dumps(x, separators=(',',':'), ensure_ascii=False) #si no,
    ↪ convierte todo el diccionario a JSON
    return str(x) #valor por defecto para otros tipos

def extract_interface(item_tags): #extrae el nombre de interfaz desde etiquetas del í
    ↪ tem
    try:
        for t in item_tags or []: #recorre la lista de etiquetas
            if isinstance(t, dict) and t.get("tag") == "interface": #si encuentra una
    ↪ etiqueta interface
                return t.get("value","") #devuelve su valor
    except Exception:
        pass #ignora errores
    return "" #retorna vacío si no hay coincidencia

def ensure_header(path, header): #asegura que el archivo CSV tenga encabezado
    write_header = not os.path.exists(path) or os.path.getsize(path) == 0 #verifica
    ↪ si debe escribir encabezado
    f = open(path, "a", newline="", encoding="utf-8") #abre el archivo en modo append
    w = csv.writer(f) #crea escritor CSV
    if write_header: #si necesita encabezado
        w.writerow(header) #lo escribe
    return f, w #retorna archivo y escritor

def process_file(in_path, out_path): #procesa el NDJSON y lo convierte a CSV
    header = ["host","interface","name","value","clock","groups","path"] #define
    ↪ columnas del CSV
    fcsv, writer = ensure_header(out_path, header) #asegura encabezado y abre escritor
    n_ok = n_err = 0 #contadores de éxito y error
    try:
        with open(in_path, "r", encoding="utf-8") as fin: #abre el NDJSON
            for line in fin: #procesa línea por línea
                line = line.strip() #limpia espacios
                if not line: #salta líneas vacías
                    continue
                try:
                    obj = json.loads(line) #intenta cargar JSON
                except Exception:
                    n_err += 1 #suma error
```

```
                    continue

                host = obj.get("host", "") #extrae campo host
                if isinstance(host, dict): #si host es un objeto
                    host = host.get("host", host.get("name","")) #toma nombre o alias

                iface = extract_interface(obj.get("item_tags")) #extrae interfaz
                name  = obj.get("name","") #nombre del ítem
                value = obj.get("value","") #valor
                clock = obj.get("clock","") #marca de tiempo
                groups = obj.get("groups", []) #grupos asociados
                path  = obj.get("path","") #ruta de origen

                row = [ #arma la fila para CSV
                    val(host),
                    val(iface),
                    val(name),
                    val(value),
                    val(clock),
                    val(groups),
                    val(path)
                ]
                writer.writerow(row) #escribe la fila
                n_ok += 1 #incrementa contador de registros válidos
    finally:
        fcsv.close() #cierra archivo CSV

if __name__ == "__main__": #punto de entrada del \textit{script}
    if len(sys.argv) != 3: #verifica que reciba dos argumentos
        print("Usage: ndjson2csv.py <input_ndjson> <output_csv>", file=sys.stderr)
    ↪ #muestra uso correcto
        sys.exit(2) #termina con error
    process_file(sys.argv[1], sys.argv[2]) #ejecuta la conversión
PY #marca el fin del bloque de entrada del \textit{script}

chmod +x /srv/share_gns3/zabbix/incoming/ndjson2csv.py #da permisos de ejecución al
    ↪ \textit{script}
```

Nota. Elaboración propia.

Se implementó un servicio de ingesta automatizado en /usr/local/bin/netml-ingest.sh, encargado de detectar, convertir y gestionar los archivos NDJSON generados por Zabbix dentro del directorio /srv/share_gns3/zabbix/incoming. Se evita escribir en el archivo main.ndjson, manteniéndolo vacío para reducir operaciones y conservar únicamente el archivo CSV consolidado.

Código 29. *Script* de de ingesta automática para conversión y gestión de archivos NDJSON en GNS3-VM

```
sudo tee /usr/local/bin/netml-ingest.sh >/dev/null <<'EOF' #crea/reescribe el
    ↪ \textit{script} en /usr/local/bin, silenciando salida estándar; usa heredoc
    ↪ literal
#!/usr/bin/env bash #shebang: ejecuta con bash del entorno
set -euo pipefail #termina ante error, variables no definidas y falla en pipelines
```

```
4
5  WATCH_DIR="/srv/share_gns3/zabbix/incoming" #directorio observado con inotify
6  MAIN_NDJSON="${WATCH_DIR}/main.ndjson" #archivo NDJSON principal (acumulador opcional)
7  MAIN_CSV="${WATCH_DIR}/main.csv" #archivo CSV principal de salida
8
9  # === FLAGS DE COMPORTAMIENTO ===
10 CONVERT_TO_CSV="true"        #"true": convierte y agrega al CSV principal
11 WRITE_MAIN_NDJSON="false"    #si "true": acumula NDJSON en MAIN_NDJSON
12 TRUNCATE_MAIN_AFTER="true"   #si WRITE_MAIN_NDJSON="true": trunca MAIN_NDJSON tras
       ↪ convertir
13
14 # Archivos principales
15 touch "$MAIN_CSV" #asegura que exista el CSV
16 chmod 664 "$MAIN_CSV" #permisos lectura/escritura para dueño y grupo
17 touch "$MAIN_NDJSON" #asegura que exista el NDJSON principal
18 chmod 664 "$MAIN_NDJSON" #permisos lectura/escritura para dueño y grupo
19
20 append_file() { #función que procesa un archivo NDJSON entrante
21   local f="$1" #ruta del archivo a procesar
22   [[ "$(basename "$f")" =~ ^.*ndjson ]] || return 0 #solo procesa nombres que
       ↪ contengan ndjson
23   [[ "$f" == "$MAIN_NDJSON" || "$f" == "$MAIN_CSV" ]] && return 0 #evita
       ↪ auto-procesarse
24
25   # 1) (Opcional) anexar al MAIN NDJSON
26   if [ "$WRITE_MAIN_NDJSON" = "true" ]; then #si está habilitado acumular NDJSON
27     cat "$f" >> "$MAIN_NDJSON" #anexa el NDJSON fuente al acumulador
28   fi
29
30   # 2) Convertir a CSV con Python
31   if [ "$CONVERT_TO_CSV" = "true" ]; then #si está habilitada la conversión
32     /usr/bin/env python3 "$WATCH_DIR/ndjson2csv.py" "$f" "$MAIN_CSV" || true
       ↪ #convierte; no falla el flujo si Python devuelve error
33   fi
34
35   # 3) Vaciar el MAIN NDJSON
36   if [ "$WRITE_MAIN_NDJSON" = "true" ] && [ "$TRUNCATE_MAIN_AFTER" = "true" ]; then
       ↪ #si corresponde truncar
37     : > "$MAIN_NDJSON" #trunca el archivo (deja tamaño 0) sin borrarlo
38   fi
39
40   # 4) Borrar el archivo fuente para liberar espacio
41   rm -f -- "$f" #elimina el archivo ya procesado
42 }
43
44 # Backfill inicial: procesa lo que ya existe
45 shopt -s nullglob #evita que los patrones vacíos se expandan a sí mismos
46 for f in "${WATCH_DIR}"/history*.ndjson* "${WATCH_DIR}"/history-*.ndjson*; do
       ↪ #recorre NDJSON históricos existentes
47   [ -f "$f" ] && append_file "$f" #si es archivo regular, lo procesa
48 done
49
50 # Watch en vivo
51 inotifywait -m -q -e close_write -e moved_to --format '%w%f' "$WATCH_DIR" | while
```

```
    ↪ read -r path; do #escucha nuevos archivos cerrados/movidos al directorio
  append_file "$path" #procesa cada archivo detectado
done
EOF #fin del heredoc: termina la escritura del \textit{script}

sudo chmod +x /usr/local/bin/netml-ingest.sh #hace el \textit{script} ejecutable
sudo systemctl restart netml-ingest.service #reinicia el servicio que usa el
    ↪ \textit{script}
journalctl -u netml-ingest.service -n 20 --no-pager #muestra los últimos 20 logs del
    ↪ servicio sin paginación
```

Nota. Elaboración propia.

Como verificación, se inspeccionó la primera línea mediante head -n 1 main.csv, lo que confirma la correcta generación de registros en el archivo main.csv, la ausencia de contenido en main.ndjson y el incremento periódico del tamaño del archivo CSV cada cinco minutos, producto de la ejecución automatizada del proceso mediante *cron*.

Figura 107. Verificación del archivo CSV generado a partir de NDJSON en GNS3-VM.

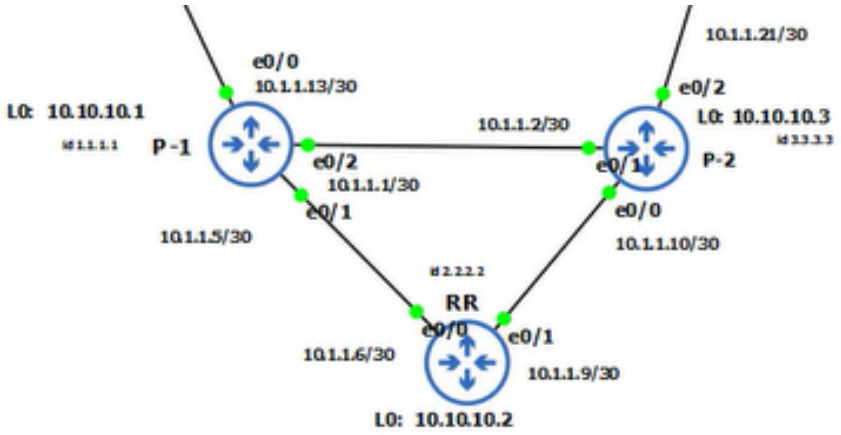
```
gns3@gns3vm:/srv/share_gns3/zabbix/incoming$ tail -n 3 main.csv | awk -F',' '{print NF" columnas -> "$0}'
7 columnas -> 10.10.10.2,e0/1,Interface Et0/1(): Inbound packets with errors,0,1762388096,Discovered hosts|MPL
S,
7 columnas -> 10.10.10.1,e0/1,Interface Et0/1(): Duplex status,3,1762388098,Discovered hosts|MPLS,
7 columnas -> 10.10.10.1,e0/2,Interface Et0/2(): Duplex status,3,1762388098,Discovered hosts|MPLS,
gns3@gns3vm:/srv/share_gns3/zabbix/incoming$ wc -l main.ndjson
0 main.ndjson
gns3@gns3vm:/srv/share_gns3/zabbix/incoming$ wc -l main.csv
315987 main.csv
gns3@gns3vm:/srv/share_gns3/zabbix/incoming$ wc -l main.csv
316228 main.csv
gns3@gns3vm:/srv/share_gns3/zabbix/incoming$
```

CAPÍTULO 13

Modelo predictivo basado en aprendizaje supervisado

Debido a la naturaleza de la red, se tomó la decisión de aplicar los modelos de Aprendizaje de Máquina únicamente al núcleo MPLS, ya que constituye la única sección con redundancia en la red. Esta característica permite provocar la falla controlada de un enlace sin afectar la continuidad operativa del resto de la topología, garantizando así un proceso de entrenamiento adecuado del algoritmo y una estimación más precisa de la probabilidad de falla de cada enlace del núcleo MPLS.

Figura 108. Topología del núcleo MPLS con enlaces redundantes.



Nota. Elaboración propia. El núcleo MPLS presenta enlaces redundantes entre los routers RR y P, lo que permite la simulación de fallas sin interrumpir el servicio.

En el modelo desarrollado, se empleó como variable principal el *Operational Status* de los enlaces, a partir del cual se derivaron indicadores de comportamiento como la frecuencia de caídas, la recencia del último evento y la cantidad de *flaps* registrados.

Aunque el modelo se basa en una única variable, mantiene su carácter de aprendizaje supervisado, al ser entrenado con datos históricos etiquetados: los valores pasados del *Operational Status* actúan como entradas y las ocurrencias futuras de fallas (falla 12h) como variable objetivo.

Un árbol de decisión puede entrenarse válidamente con una sola variable explicativa, generando divisiones internas basadas en condiciones simples, como "downs 24h $>$ 2" o "horas desde última falla $<$ 6". Este enfoque conserva la interpretabilidad y permite establecer relaciones directas entre el historial operativo de un enlace y su probabilidad de falla en distintos horizontes temporales.

De esta manera, el uso del *Operational Status* como variable base, junto con sus derivadas temporales y de frecuencia, constituye una estrategia metodológicamente sólida y defendible, al centrarse en el factor más representativo de la estabilidad del núcleo MPLS [31].

Código 30. *Script* de Python para el modelado predictivo de fallas en enlaces MPLS basado en aprendizaje supervisado

```
#!/usr/bin/env python3 #shebang para ejecutar con Python 3
# -*- coding: utf-8 -*- #declara codificación UTF-8
import os, math, json #módulos estándar: sistema, matemáticas y JSON
import numpy as np #NumPy para operaciones numéricas
import pandas as pd #Pandas para manejo de datos tabulares
from pathlib import Path #manejo de rutas orientado a objetos
from sklearn.tree import DecisionTreeClassifier #árbol de decisión para clasificación
from sklearn.metrics import accuracy_score, precision_recall_fscore_support,
↪ confusion_matrix, classification_report #métricas de evaluación

# -------------------- Paths / Constantes -------------------- #sección de constantes
↪ y rutas
BASE = Path("/srv/share_gns3/zabbix/incoming") #directorio base de trabajo
CSV  = BASE / "main.csv" #ruta del CSV principal de métricas

DOWNCSV   = BASE / "downs_por_enlace.csv" #salida: downs históricos por enlace
PREDCSV   = BASE / "predicciones_ranked.csv" #salida: predicciones finales ordenadas
METRICS   = BASE / "metrics_multi_labels.txt" #salida: métricas por etiqueta

FREQ = "5min" #frecuencia de remuestreo temporal
H12  = 12  # horizonte 12h #constante de referencia de horizonte (no usada explí
↪ citamente)
RANDOM_STATE = 42 #semilla para reproducibilidad
# -------------------- Utilidades -------------------- #funciones utilitarias
def load_oper(csv_path: Path) -> pd.DataFrame: #carga y filtra Operational status
    df = pd.read_csv(csv_path) #lee CSV de entrada
    need = {"host","interface","name","value","clock"} #columnas requeridas
    miss = need - set(df.columns) #detecta columnas faltantes
    if miss: #valida esquema
        raise ValueError(f"Faltan columnas: {miss}") #lanza error si faltan campos
    m = df["name"].str.contains("Operational status", case=False, na=False) #filtra
↪ por ítems de estado operacional
```

```
    df = df[m].copy() #aplica filtro
    df["ts"] = pd.to_datetime(df["clock"], unit="s", utc=True) #convierte epoch a timestamp UTC
    df["oper_status"] = pd.to_numeric(df["value"], errors="coerce") #convierte a número el estado
    df = df[df["oper_status"].isin([1,2])] #mantiene valores 1/2 (UP/DOWN según contexto)
    return df[["host","interface","ts","oper_status"]].sort_values(["host","interface","ts"]) #ordena y devuelve columnas clave

def resample_5m_max(df: pd.DataFrame) -> pd.DataFrame: #remuestrea a 5 min usando máximo
    out = [] #acumulador de dataframes
    for (h,i), g in df.groupby(["host","interface"], sort=False): #agrupa por enlace
        g = g.set_index("ts").sort_index() #indexa por tiempo
        r = g["oper_status"].resample(FREQ).max().ffill().fillna(1) #remuestrea por máximo y completa con último valor
        tdf = pd.DataFrame({"ts": r.index, "oper_status": r.values}) #reconstituye dataframe
        tdf["host"], tdf["interface"] = h, i #anota identificadores
        out.append(tdf.reset_index(drop=True)) #acumula resultado
    tsdf = pd.concat(out, ignore_index=True).sort_values(["host","interface","ts"]) #concatena y ordena
    return tsdf #retorna serie temporal uniforme

def build_signals(tsdf: pd.DataFrame) -> pd.DataFrame: #construye señales interpretables
    """Señales claro/interpretables por timestamp.""" #docstring de función
    def feats(g): #función interna por enlace
        g = g.sort_values("ts").set_index("ts") #ordena e indexa por tiempo
        oper = g["oper_status"] #serie de estado operativo

        # \textit{flaps} (cambios 1<->2 por paso) #comentario de intención
        \textit{flaps}_step = oper.diff().abs().fillna(0) #magnitud de cambio entre muestras (\textit{flaps})
                # horas desde último down #INDENTACIÓN EXTRA: visual; no afecta ejecución
        last = None; hrs=[] #inicializa último DOWN y lista de horas
        for t,v in oper.items(): #itera en el tiempo
            if v==2: last = t #actualiza último DOWN
            hrs.append(np.nan if last is None else (t-last).total_seconds()/3600.0) #calcula horas desde último DOWN
        g["hrs_since_last_down"] = pd.Series(hrs, index=g.index).fillna(1e9) #rellena sin DOWN con valor grande

        # downs recientes #comentario de intención
        is_down = (oper==2).astype(int) #binario: 1 si DOWN
        g["downs_24h"] = is_down.rolling("24H", min_periods=1).sum() #conteo rolling 24h
        g["downs_7d"]  = is_down.rolling("168H",min_periods=1).sum() #conteo rolling 7 d

        # \textit{flaps} recientes (3h) #comentario de intención
        g["\textit{flaps}_3h"] = \textit{flaps}_step.rolling("3H", min_periods=1).sum
```

```
    ↪ () #suma de cambios en 3h
68
69          return g.reset_index() #restaura ts como columna
70
71      fdf = tsdf.groupby(["host","interface"], group_keys=False).apply(feats) #aplica
    ↪ extracción de features por enlace
72      return fdf.sort_values(["host","interface","ts"]).reset_index(drop=True) #ordena y
    ↪  limpia índices
73
74  def future_window_mask(times, t_start, hours): #máscara de ventana futura [t_start,
    ↪ t_start+hours]
75      t_end = t_start + np.timedelta64(hours, 'h') #define fin de ventana
76      return (times > t_start) & (times <= t_end) #condición booleana de inclusión
77
78  def add_multi_labels(fdf: pd.DataFrame) -> pd.DataFrame: #crea etiquetas supervisadas
    ↪ multi-horizonte/evento
79      """Crea etiquetas supervisadas multi-horizonte/multi-evento.""" #docstring
80      def lab(g): #etiquetado por enlace
81          g = g.sort_values("ts").reset_index(drop=True) #ordena por tiempo
82          times = g["ts"].to_numpy() #vector de tiempos
83          op    = g["oper_status"].to_numpy() #vector de estado
84          isdn  = (op==2).astype(int) #binario DOWN
85          flips = np.abs(np.diff(op, prepend=op[0])) #cambios entre muestras
86
87          g["y_down_1h"]  = 0 #etiqueta: habrá DOWN en 1h
88          g["y_down_6h"]  = 0 #etiqueta: habrá DOWN en 6h
89          g["y_down_12h"] = 0 #etiqueta: habrá DOWN en 12h
90          g["y_flap_3h"]  = 0 #etiqueta: \textit{flaps} en 3h (mayor a 2 cambios)
91          g["y_downtime20_12h"] = 0 #etiqueta: del tiempo en DOWN en 12h
92
93          for k, t in enumerate(times): #itera por timestamp
94              m1  = future_window_mask(times, t, 1) #ventana 1h
95              m6  = future_window_mask(times, t, 6) #ventana 6h
96              m12 = future_window_mask(times, t, 12) #ventana 12h
97              g.at[k, "y_down_1h"]  = 1 if isdn[m1].any()  else 0 #marca si hay algún
    ↪ DOWN en 1h
98              g.at[k, "y_down_6h"]  = 1 if isdn[m6].any()  else 0 #marca si hay algún
    ↪ DOWN en 6h
99              g.at[k, "y_down_12h"] = 1 if isdn[m12].any() else 0 #marca si hay algún
    ↪ DOWN en 12h
100
101             m3  = future_window_mask(times, t, 3) #ventana 3h
102             g.at[k, "y_flap_3h"]  = 1 if flips[m3].sum() >= 2 else 0 #marca \textit{
    ↪ flaps} si es mayor o igual a 2 cambios
103
104             if m12.any(): #si existe ventana 12h
105                 dt12 = isdn[m12].mean()  # proporción de ticks en DOWN #promedio como
    ↪ porcentaje del tiempo en DOWN
106                 g.at[k, "y_downtime20_12h"] = 1 if dt12 >= 0.20 else 0 #marca si
    ↪ supera 20%
107         return g #retorna etiquetado por enlace
108     return fdf.groupby(["host","interface"], group_keys=False).apply(lab).reset_index(
    ↪ drop=True) #aplica por enlace
109
```

```
# -------------------- Score determinista (interpretable) -------------------- #sección de score interpretable
def risk_score_row(r): #calcula score determinista 0..1 por fila
    """
    Score 0..1:
      - c24: más downs en 24h => más riesgo (sube rápido)
      - c7 : tendencia 7d
      - cfl: \textit{flaps} recientes (3h)
      - cre: recencia de último down (decae exponencial con 12h)
    """ #documenta componentes del score
    d24 = float(r.get("downs_24h", 0.0)) #downs en 24h
    d7  = float(r.get("downs_7d", 0.0)) #downs en 7d
    fl3 = float(r.get("\textit{flaps}_3h", 0.0)) #\textit{flaps} en 3h
    hrs = float(r.get("hrs_since_last_down", 1e9)) #horas desde último DOWN

    c24 = 1 - math.exp(-0.7*d24) #componente 24h (rápido)
    c7  = 1 - math.exp(-0.25*d7) #componente 7d (lento)
    cfl = 1 - math.exp(-0.9*fl3) #componente \textit{flaps} (sensible)
    cre = math.exp(-hrs/12.0) #componente recencia (decae con 12h)

    score = 0.40*c24 + 0.30*c7 + 0.20*cfl + 0.10*cre #ponderación de componentes
    return max(0.0, min(1.0, score)) #clamp al rango [0,1]

def risk_score_latest(fdf: pd.DataFrame) -> pd.DataFrame: #aplica score al último punto por enlace
    last = fdf.sort_values("ts").groupby(["host","interface"], as_index=False).tail(1) #última muestra por enlace
    out = last[["host","interface","ts","oper_status","downs_24h","downs_7d","\textit{flaps}_3h","hrs_since_last_down"]].copy() #Selección de columnas
    out["prob_falla_12h_pct"] = out.apply(risk_score_row, axis=1)*100.0 #convierte score a %
    out["pred_falla_12h"] = (out["prob_falla_12h_pct"]>=50.0).astype(int) #clasifica por umbral 50%
    return out.sort_values(["host","interface"]).reset_index(drop=True) #ordena y limpia índices

# -------------------- Árbol(es) pequeños por etiqueta -------------------- #modelado ML por etiqueta
def temporal_split(data: pd.DataFrame, target: str, ratio=0.8): #split temporal train/test
    data = data.sort_values("ts") #ordena por tiempo
    n = len(data) #tamaño total
    cut = int(n*ratio) #corte por proporción
    cut = min(max(cut, 1), n-1) #garantiza límites válidos

    def has_pos(d): return int(d[target].sum())>0 #chequea presencia de positivos
    if has_pos(data): #si hay positivos en total
        while cut < n-1 and not has_pos(data.iloc[:cut]): cut += 1 #mueve corte para tener positivos en train
        while cut > 1   and not has_pos(data.iloc[cut:]): cut -= 1 #mueve corte para tener positivos en test
    return data.iloc[:cut], data.iloc[cut:] #retorna particiones

def metrics_text(y_true, y_pred): #calcula métricas de evaluación
```

```
153     acc = accuracy_score(y_true, y_pred) #accuracy
154     pr, rc, f1, _ = precision_recall_fscore_support(y_true, y_pred, average="binary",
    ↪ zero_division=0) #precision/recall/F1
155     cm = confusion_matrix(y_true, y_pred) #matriz de confusión
156     rep = classification_report(y_true, y_pred, digits=4, zero_division=0) #reporte
    ↪ detallado
157     return acc, pr, rc, f1, cm, rep #retorna métricas
158 
159 def train_tiny_tree_for_label(fdf: pd.DataFrame, ycol: str, feat_cols): #entrena árbol
    ↪  pequeño por etiqueta
160     df = fdf.dropna(subset=feat_cols+[ycol]).copy() #elimina nulos en features/target
161     if df.empty or df[ycol].nunique()<2: #requiere variación del target
162         return None, {} #no entrena si no hay clases
163 
164     tr, te = temporal_split(df, ycol, ratio=0.8) #split temporal
165     if tr[ycol].nunique()<2:  # aún sin positivos/negativos #verificación adicional
166         return None, {} #no entrena
167 
168     clf = DecisionTreeClassifier( #configura árbol pequeño
169         criterion="entropy", #criterio de ganancia de información
170         max_depth=2, #profundidad limitada
171         min_samples_leaf=5, #mínimo por hoja
172         class_weight="balanced", #balanceo por clase
173         random_state=RANDOM_STATE #semilla fija
174     )
175     clf.fit(tr[feat_cols], tr[ycol]) #entrena con train
176 
177     yhat = clf.predict(te[feat_cols]) #predice en test
178     acc, pr, rc, f1, cm, rep = metrics_text(te[ycol], yhat) #calcula métricas
179 
180     info = { #paquete de resultados
181         "train_size": len(tr), #tamaño train
182         "test_size": len(te), #tamaño test
183         "positives_train": int(tr[ycol].sum()), #positivos en train
184         "positives_test": int(te[ycol].sum()), #positivos en test
185         "accuracy": acc, "precision": pr, "recall": rc, "f1": f1, #métricas agregadas
186         "confusion_matrix": cm.tolist(), #matriz de confusión serializada
187         "report": rep #reporte sklearn
188     }
189     return clf, info #devuelve modelo y métricas
190 
191 def apply_multi_trees_latest(fdf: pd.DataFrame, base_df: pd.DataFrame, metrics_path:
    ↪ Path) -> pd.DataFrame: #entrena/aplica modelos y mezcla con score
192     """
193     Entrena arbolitos para y_down_1h, y_down_6h, y_down_12h, y_flap_3h,
    ↪ y_downtime20_12h.
194     Devuelve prob final por enlace al último timestamp, combinando ML (promedio) con
    ↪ score determinista.
195     Reglas de guardarraíl: si el enlace es muy estable, ignorar ML y usar el score
    ↪ 100%.
196     """ #docstring de procedimiento
197     # Incorporación de etiquetas múltiples #comentario de paso
198     flabeled = add_multi_labels(fdf) #genera dataset etiquetado
199 
```

```
    # Features (señales interpretables) #Selección de variables
    feat_cols = ["downs_24h","downs_7d","\textit{flaps}_3h","hrs_since_last_down"] #
↪ features candidatas
    feat_cols = [c for c in feat_cols if c in flabeled.columns] #filtra por existencia

    ycols = ["y_down_1h","y_down_6h","y_down_12h","y_flap_3h","y_downtime20_12h"] #
↪ lista de etiquetas

    models = {} #diccionario de modelos por etiqueta
    infos  = {} #diccionario de métricas por etiqueta

    # Entrenar modelo por etiqueta #bucle de entrenamiento
    for y in ycols: #itera etiquetas
        clf, info = train_tiny_tree_for_label(flabeled, y, feat_cols) #entrena árbol
        if clf is not None: #si entrenó
            models[y] = clf #almacena modelo
            infos[y]  = info #almacena métricas

    # Guardar métricas por etiqueta #persistencia de resultados
    with open(metrics_path, "w") as f: #abre archivo de métricas
        for y in ycols: #itera etiquetas
            f.write(f"=== {y} ===\n") #encabezado de sección
            if y in infos: #si hay métricas
                info = infos[y] #recupera info
                f.write(f"TRAIN={info['train_size']} (pos={info['positives_train']}),
↪ " #tamaño y positivos de train
                        f"TEST={info['test_size']} (pos={info['positives_test']})\n")
↪ #tamaño y positivos de test
                f.write(f"Accuracy={info['accuracy']:.4f} | Precision={info['precision
↪ ']:.4f} | " #métricas principales
                        f"Recall={info['recall']:.4f} | F1={info['f1']:.4f}\n") #
↪ continúa métricas
                f.write(f"Confusion Matrix={info['confusion_matrix']}\n") #matriz de
↪ confusión
                f.write(info["report"] + "\n\n") #reporte detallado
            else: #sin modelo entrenado
                f.write("No entrenado (sin variación o sin positivos suficientes)\n\n"
↪ ) #nota de no entrenamiento

    # Último punto por enlace #Selección del corte final
    last = flabeled.sort_values("ts").groupby(["host","interface"], as_index=False).
↪ tail(1).copy() #última fila por enlace
    last = last.fillna(0) #rellena nulos

    # Probabilidad ML (promedio de arbolitos disponibles) #ensamble simple
    ml_probs = [] #acumulador de probabilidades
    for _, row in last.iterrows(): #itera enlaces
        ps = [] #lista de probabilidades por etiqueta
        for y, clf in models.items(): #itera modelos entrenados
            X = row[feat_cols].to_frame().T #vector de features en forma 2D
            try: #intenta probabilidad si disponible
                p = float(clf.predict_proba(X)[:,1][0]) #probabilidad clase positiva
            except Exception: #fallback si no hay predict_proba
                p = float(clf.predict(X)[0]) #usa predicción dura como proxy
```

```
            ps.append(p) #acumula probabilidad
        ens = float(np.mean(ps)) if ps else 0.0 #promedia modelos disponibles
        ml_probs.append(ens) #guarda resultado
    last["prob_ml_multi"] = ml_probs #añade columna de probabilidad ML

    # Mezcla con score determinista #combinación ML + score
    merged = last.merge(base_df[["host","interface","prob_falla_12h_pct"]], #une con score base
                        on=["host","interface"], how="left", suffixes=("","_score")) # left join por enlace
    merged["prob_score"] = merged["prob_falla_12h_pct"].astype(float) / 100.0 # convierte score % a [0,1]

    # Guardarraíl de estabilidad: sin downs en 7d, sin \textit{flaps} en 3h, última falla > 48h => usar SOLO score #regla de override
    stable_mask = (merged["downs_7d"]==0) & (merged["\textit{flaps}_3h"]==0) & (merged["hrs_since_last_down"]>48.0) #condición de estabilidad

    # Prob final #cálculo de probabilidad final
    merged["prob_final"] = np.where( #aplica Selección condicional
        stable_mask, #si estable
        merged["prob_score"], #usa solo score determinista
        0.5*merged["prob_ml_multi"].astype(float) + 0.5*merged["prob_score"].astype(float) #mezcla 50/50
    )

    # Ajustes finales #post-proceso de probabilidad
    merged["prob_falla_12h_pct"] = (merged["prob_final"]*100.0).round(2) #escala a %
    merged["pred_falla_12h"]     = (merged["prob_falla_12h_pct"]>=50.0).astype(int) # clasifica por umbral 50%

    return merged[["host","interface","ts","pred_falla_12h","prob_falla_12h_pct"]]\ # Selección y retorno de columnas
            .sort_values(["host","interface"]).reset_index(drop=True) #ordena y normaliza índices

# -------------------- Main -------------------- #punto de entrada
def main(): #función principal
    os.makedirs(BASE, exist_ok=True) #asegura que exista el directorio base

    print([1] Cargando Operational status) #ERROR DE SINTAXIS: la cadena debe ir entre comillas
    raw = load_oper(CSV) #carga y filtra métricas Operational status
    print(f"Filas crudas: {len(raw)} | hosts: {raw['host'].nunique()} | enlaces: {raw[['host','interface']].drop_duplicates().shape[0]}") #resumen de datos crudos

    print([2] Resample 5min (MAX)) #ERROR DE SINTAXIS: faltan comillas y paréntesis desbalanceados
    tsdf = resample_5m_max(raw) #remuestrea a 5 minutos por máximo

    print([3] Señales claras (downs/\textit{flaps}/recencia)) #ERROR DE SINTAXIS: la cadena debe ir entre comillas
    sdf = build_signals(tsdf) #construye señales interpretables

```

```
    # Diagnóstico: downs totales históricos por enlace #comentario informativo
    downs_total = sdf[sdf["oper_status"]==2].groupby(["host","interface"]).size().
↪ reset_index(name="downs_total")\ #agrega downs por enlace
                    .sort_values(["host","interface"]) #ordena por enlace
    downs_total.to_csv(DOWNCSV, index=False) #exporta downs históricos

    print([4] Score determinista por enlace (último corte)) #ERROR DE SINTAXIS: la
↪ cadena debe ir entre comillas
    base = risk_score_latest(sdf) #calcula score determinista al último punto

    print(Ensamble supervisado multi-etiqueta + mezcla con score) #ERROR DE SINTAXIS:
↪ faltan comillas
    pred = apply_multi_trees_latest(sdf, base, METRICS) #entrena/aplica modelos y
↪ mezcla con score

    pred.to_csv(PREDCSV, index=False) #exporta predicciones

    print("\n--- downs_por_enlace.csv (histórico) ---") #cabecera de bloque de salida
    with pd.option_context('display.max_rows', 100, 'display.width', 160): #ajusta
↪ visualización
        print(downs_total.to_string(index=False)) #imprime tabla de downs

    print("\n--- predicciones_ranked.csv ---") #cabecera de bloque de salida
    with pd.option_context('display.max_rows', 100, 'display.width', 160): #ajusta
↪ visualización
        print(pred.to_string(index=False)) #imprime tabla de predicciones

    print(f"\nArchivos:\n- {DOWNCSV}\n- {PREDCSV}\n- {METRICS}\n Hecho.") #resumen de
↪ artefactos generados

if __name__ == "__main__": #ejecución directa del módulo
    main() #invoca función principal
```

Nota. Elaboración propia.

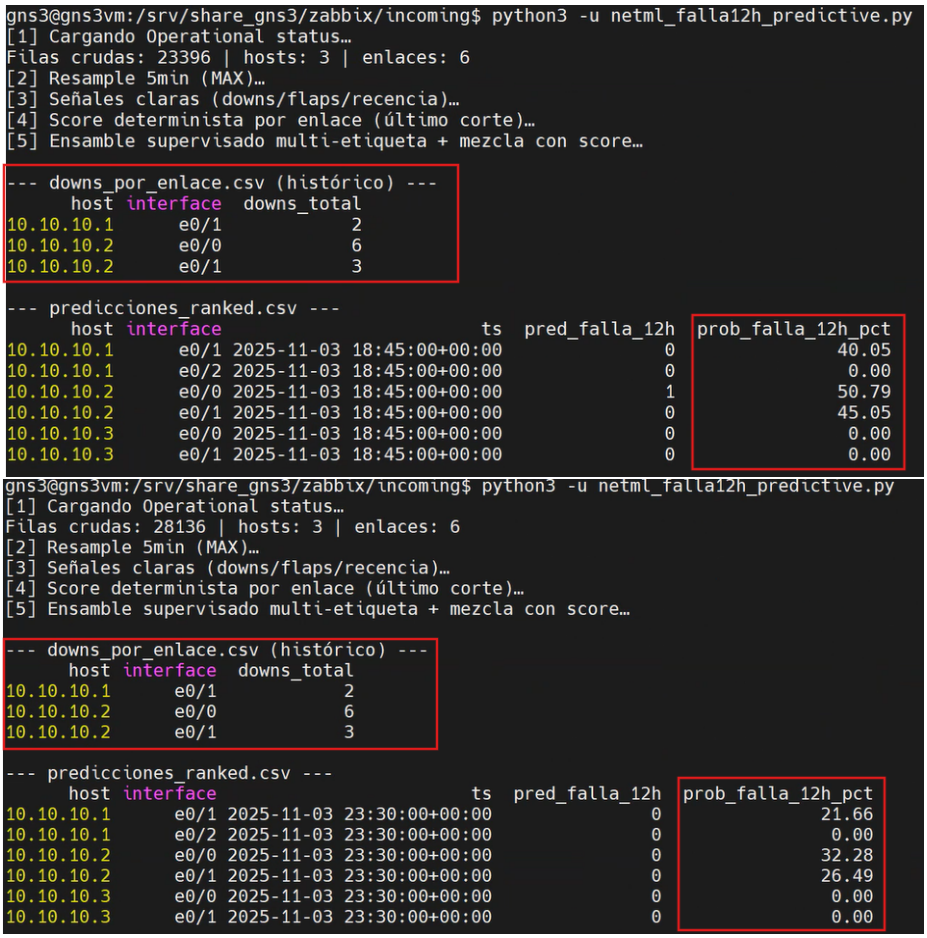
El *script* integra en un único archivo todo el proceso de modelado, utilizando como base la variable *Operational Status* ($1 = UP$, $2 = DOWN$). A partir de ella se generan indicadores temporales como downs 24h, downs 7d, *flaps* 3h y hrs since last down, que describen la frecuencia y recencia de eventos.

Con estas señales se construyen etiquetas supervisadas para distintos horizontes: y down 1h, y down 6h, y down 12h, y flap 3h y y downtime20 12h. Además, se entrena un árbol de decisión independiente por etiqueta (profundidad máxima 2), con división temporal 80/20 y balanceo de clases.

La salida de los modelos se combina con un *score determinista* basado en frecuencia, recencia y *flaps* recientes, aplicando una ponderación 50/50. Si un enlace muestra alta estabilidad (sin caídas en siete días, sin *flaps* y última falla hace más de 48 h), el sistema usa solo el score, evitando falsos positivos.

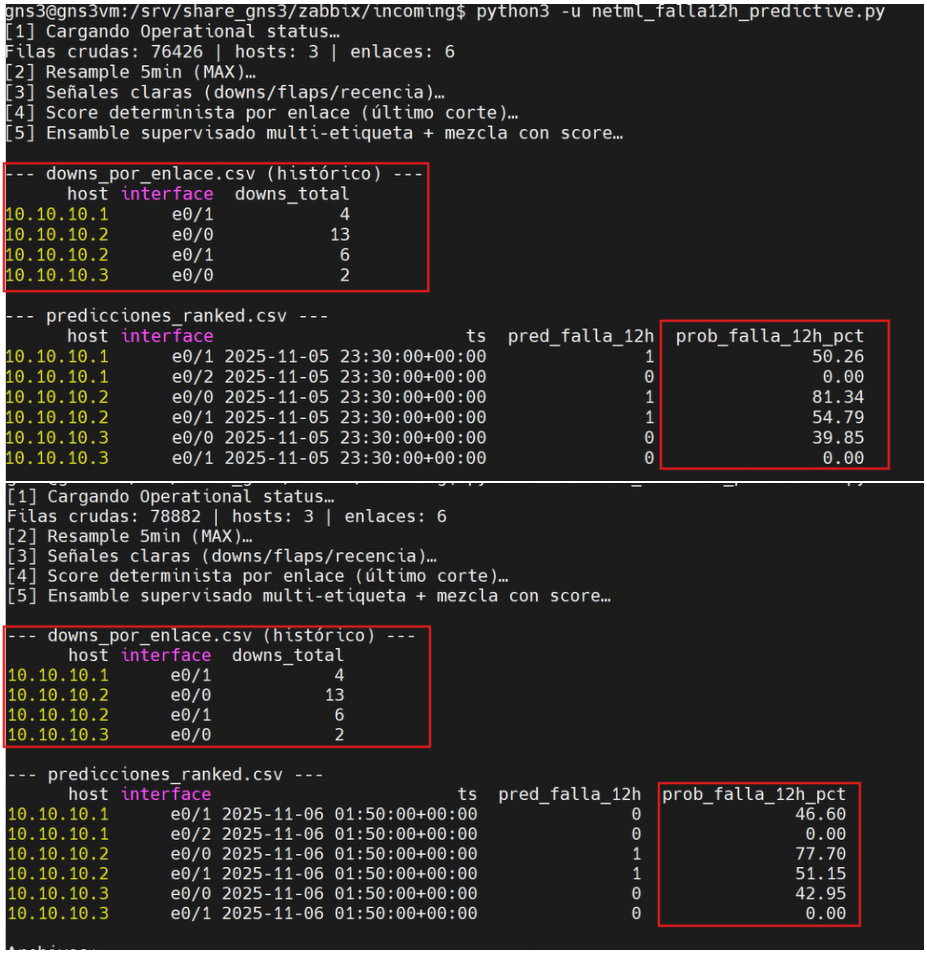
El *script* produce tres salidas: downs por enlace.csv, predicciones ranked.csv y metrics multi labels.txt.

Figura 109. Comparativa entre la primera y segunda ejecución del modelo predictivo de fallas en enlaces del núcleo MPLS en distintos momentos temporales.



Nota. Elaboración propia. Salida del pipeline de predicción de fallas por enlace. Se muestran dos artefactos generados: el histórico de caídas por enlace y el *ranking* de riesgo a 12 h.

Figura 110. Comparativa entre la segunda y tercera ejecución del modelo predictivo de fallas en enlaces del núcleo MPLS en distintos momentos temporales.



Nota. Elaboración propia. Salida del pipeline de predicción de fallas por enlace. Se muestran dos artefactos generados: el histórico de caídas por enlace y el *ranking* de riesgo a 12 h.

La diferencia observada entre ambas ejecuciones del algoritmo, a pesar a que el número histórico de caídas permanece constante, se explica por la naturaleza del modelo supervisado y la influencia de las variables de incluidas en el cálculo del riesgo. En ambas instancias se mantiene el mismo número de eventos históricos de caída, pero las probabilidades de falla varían debido a la evolución temporal de las señales de recencia y estabilidad operacional.

El algoritmo no solo considera la cantidad total de eventos de caída, sino también el tiempo transcurrido desde la última falla, la densidad de *flaps* recientes y la tendencia temporal derivada del *Operational Status*. Es por eso que, aunque el conteo acumulado de downs no haya variado, el modelo interpreta que el contexto temporal ha cambiado, lo que modifica las probabilidades estimadas.

Las métricas obtenidas del modelo presentadas en el anexo 35 evidencian un comportamiento coherente y desbalanceada del conjunto de datos. En horizontes cortos (1h y 6h), el modelo presenta alta exactitud general pero baja sensibilidad, con dificultad para anticipar eventos inmediatos a partir de señales limitadas.

En el horizonte de 12h se observa un incremento en la capacidad de detección de fallas (recall = 0.78), aunque con un aumento de falsos positivos y menor precisión. Es preferible priorizar la detección temprana de posibles fallas. Las etiquetas complementarias, como y flap 3h, mostraron una menor capacidad predictiva debido a la escasez de eventos, mientras que y downtime20 12h no logró entrenamiento por falta de variación en los datos.

CAPÍTULO 14

Algoritmo de optimización dinámica del enrutamiento

Con el objetivo de prevenir interrupciones en la red, se implementó un *script* en Python que utiliza la librería Netmiko para ajustar dinámicamente el costo de las rutas OSPF. Este ajuste se realiza de forma escalonada, priorizando los enlaces según el nivel de riesgo de falla estimado por el algoritmo de Machine Learning.

Código 31. *Script* de Python para la optimización dinámica del enrutamiento OSPF basado en el riesgo de falla de enlaces MPLS breaklines

```
1
2  #!/usr/bin/env python3 # ejecuta con python tres
3  # -*- coding: utf-8 -*- # archivo en codificacion utf ocho
4
5  import os, csv, time, re, traceback # modulos del sistema y utilidades
6  from datetime import datetime # manejo de tiempos
7  from pathlib import Path # rutas de archivos
8
9  import pandas as pd # manejo de datos
10 import numpy as np # calculo numerico
11 from netmiko import ConnectHandler # conexion a equipos de red
12
13 # ---------------------- Paths ---------------------- # seccion de rutas
14 BASE = Path("/srv/share_gns3/zabbix/incoming") # base de trabajo
15 PRED = BASE / "predicciones_ranked.csv"         # archivo de entrada del modelo
16 NETOPT_DIR = BASE / "netopt" # carpeta de trabajo del actuador
17 PLAN = NETOPT_DIR / "cost_plan.csv" # plan de costos a aplicar
18 STATE = NETOPT_DIR / "last_applied.csv" # estado previo aplicado
19 LOGF = NETOPT_DIR / "actuator.log" # archivo de registro
20
21 # ---------------- Credenciales globales ---------- # credenciales y tipo de equipo
22 DEVICE_TYPE = os.environ.get("NETOPT_DEVICE", "cisco_ios") # tipo de plataforma
23 USERNAME    = os.environ.get("NETOPT_USER", "admin") # usuario de acceso
```

```
PASSWORD    = os.environ.get("NETOPT_PASS", "contrasena") # clave de acceso

# ---------------- Parámetros ---------------------- # parametros de actuacion
MAX_COST = int(os.environ.get("NETOPT_MAX_COST", "200"))   # costo mayor para mayor riesgo
MIN_COST = int(os.environ.get("NETOPT_MIN_COST", "10"))    # costo menor para menor riesgo
COOLDOWN_SECONDS = int(os.environ.get("NETOPT_COOLDOWN", "120"))  # espera entre ejecuciones
COST_MIN_STEP = int(os.environ.get("NETOPT_MINSTEP", "0"))        # umbral minimo de cambio

DRYRUN = os.environ.get("NETOPT_DRYRUN", "0") == "1" # modo simulacion
WRITE_MEM = True # guardar configuracion en memoria

# ------------------------------------------------ # funciones de soporte
def log(msg: str): # registra mensajes en archivo y consola
    line = f"{datetime.utcnow().isoformat()}Z {msg}" # compone linea con sello de tiempo
    print(line) # muestra en consola
    try:
        os.makedirs(LOGF.parent, exist_ok=True) # asegura carpeta de registros
        with open(LOGF, "a") as f: f.write(line + "\n") # agrega linea al registro
    except Exception:
        pass # ignora fallas de registro

def ensure_dirs(): # asegura directorios de trabajo
    os.makedirs(NETOPT_DIR, exist_ok=True) # crea carpeta si no existe

def cooldown_ok() -> bool: # verifica tiempo de espera entre planes
    if not PLAN.exists():
        return True # si no hay plan previo permite ejecutar
    mtime = PLAN.stat().st_mtime # obtiene tiempo de modificacion
    return (time.time() - mtime) >= COOLDOWN_SECONDS # compara con ventana de espera

def normalize_iface(name: str) -> str: # normaliza nombre de interfaz para cisco
    """
    Normaliza variantes a 'EthernetX/Y' para Cisco IOS.
    Acepta: 'e0/1', 'E0/1', 'Et0/1', 'Ethernet0/1'...
    """
    if not name:
        return name # retorna si esta vacio
    s = name.strip() # limpia espacios
    m = re.search(r'(\d+/\d+)', s) # extrae numeros de ranura y puerto
    num = m.group(1) if m else s # selecciona numero detectado
    return f"Ethernet{num}" # arma nombre estandar de interfaz

def assign_costs_by_rank(df: pd.DataFrame) -> pd.DataFrame: # asigna costos segun \textit{ranking}
    """
    df columnas: host, interface, prob_falla_12h_pct
    Devuelve df con: rank, desired_cost, iface_cli (normalizada)
    """
    df = df.copy() # trabaja sobre copia
```

```
    # normalizar interfaz a CLI Cisco
    df["iface_cli"] = df["interface"].astype(str).apply(normalize_iface) # agrega columna normalizada

    # ordenar por riesgo desc
    df = df.sort_values(["prob_falla_12h_pct","host","iface_cli"], ascending=[False, True, True]).reset_index(drop=True) # ordena por riesgo y nombre
    df["rank"] = np.arange(1, len(df)+1) # asigna posicion de \textit{ranking}

    n = len(df) # cantidad de enlaces
    if n <= 0:
        return df # retorna vacio si no hay filas

    if n == 1:
        costs = [MAX_COST] # un unico enlace recibe costo maximo
    else:
        costs = np.linspace(MAX_COST, MIN_COST, num=n) # rampa lineal de costos

    df["desired_cost"] = np.round(costs).astype(int) # redondea costos a entero
    return df # devuelve plan con costos deseados

def load_state(): # carga estado previo aplicado
    s = {} # diccionario de estado
    if STATE.exists():
        with open(STATE, newline='', encoding='utf-8', errors='ignore') as f:
            for r in csv.DictReader(f):
                key = (r.get("host","").strip(), r.get("iface_cli","").strip()) # clave por equipo e interfaz
                try:
                    s[key] = int(r.get("last_cost") or "0") # ultimo costo aplicado
                except:
                    pass # ignora errores de parseo
    return s # retorna estado

def save_state(state_dict): # guarda estado actualizado
    with open(STATE, "w", newline='') as f:
        w = csv.writer(f) # escritor csv
        w.writerow(["host","iface_cli","last_cost","ts"]) # encabezado
        for (h,i), c in state_dict.items():
            w.writerow([h,i,c,int(time.time())]) # linea por interfaz

def apply_cost(host: str, iface_cli: str, cost: int): # aplica costo en equipo remoto
    if DRYRUN:
        log(f"[DRYRUN] {host} {iface_cli} -> ip ospf cost {cost}") # solo registra accion en simulacion
        return
    dev = None # manejador de conexion
    try:
        dev = ConnectHandler(device_type=DEVICE_TYPE, host=host,
                             username=USERNAME, password=PASSWORD) # abre sesion
        cmds = [f"interface {iface_cli}", f"ip ospf cost {cost}"] # comandos de configuracion
        out = dev.send_config_set(cmds) # envia configuracion
        if WRITE_MEM:
```

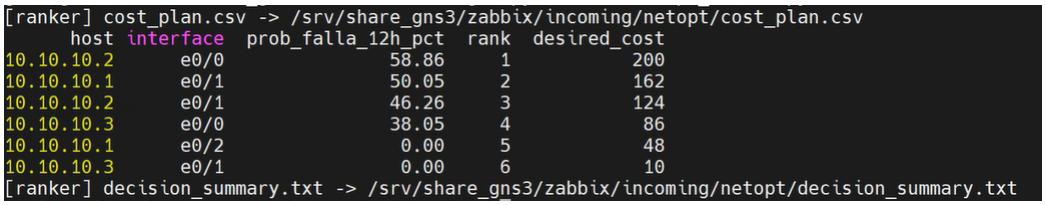
```
            dev.save_config() # guarda configuracion en memoria
        log(f"[APPLIED] {host} {iface_cli} cost={cost}") # registra aplicacion exitosa
    finally:
        if dev:
            try: dev.disconnect() # cierra sesion
            except: pass # ignora error al cerrar

def main(): # flujo principal
    ensure_dirs() # prepara carpetas

    # cooldown para no re-planificar/aplicar cada instante
    if not cooldown_ok():
        log(f"[ranker] Saltando por cooldown {COOLDOWN_SECONDS}s.") # respeta ventana
↪ de espera
        return

    if not PRED.exists():
        raise SystemExit(f"[ranker] No existe {PRED}") # valida entrada del modelo

    df = pd.read_csv(PRED) # lee predicciones
    need = {"host","interface","prob_falla_12h_pct"} # columnas requeridas
    if not need.issubset(df.columns):
        raise SystemExit(f"[ranker] {PRED} no tiene columnas requeridas: {need}") #
↪ valida esquema

    # asignar costos por \textit{ranking}
    ranked = assign_costs_by_rank(df) # calcula costos segun riesgo

    # guardar plan
    ranked[["host","iface_cli","prob_falla_12h_pct","rank","desired_cost"]].to_csv(
↪ PLAN, index=False) # escribe plan a disco
    log("[ranker] Plan generado:") # informa plan generado
    log(ranked[["host","iface_cli","prob_falla_12h_pct","rank","desired_cost"]].
↪ to_string(index=False)) # imprime plan

    # cargar estado previo y aplicar
    state = load_state() # lee estado previo
    log(f"[actuator] DRYRUN={DRYRUN} | devices_en_plan={len(ranked)}") # informa modo
↪ y cantidad de equipos

    for _, r in ranked.iterrows(): # recorre interfaces a aplicar
        host = str(r["host"]) # nombre del equipo
        iface_cli = str(r["iface_cli"]) # interfaz normalizada
        desired = int(r["desired_cost"]) # costo deseado

        last = state.get((host, iface_cli), None) # costo previo
        if last is not None and abs(desired - last) < COST_MIN_STEP:
            log(f"[SKIP] {host} {iface_cli} cambio < {COST_MIN_STEP} (last={last},
↪ desired={desired})") # omite cambios pequenos
            continue

        try:
            apply_cost(host, iface_cli, desired) # aplica cambio en el equipo
            state[(host, iface_cli)] = desired # actualiza estado en memoria
```

```
        except Exception as e:
            log(f"[ERROR] aplicando {host} {iface_cli} -> {desired}: {repr(e)}") # registra error de aplicacion
            log(traceback.format_exc()) # traza del error

    save_state(state) # guarda estado actualizado
    log("[actuator] DONE.") # finaliza ejecucion

if __name__ == "__main__":
    main() # ejecuta principal
```

Nota. Elaboración propia.

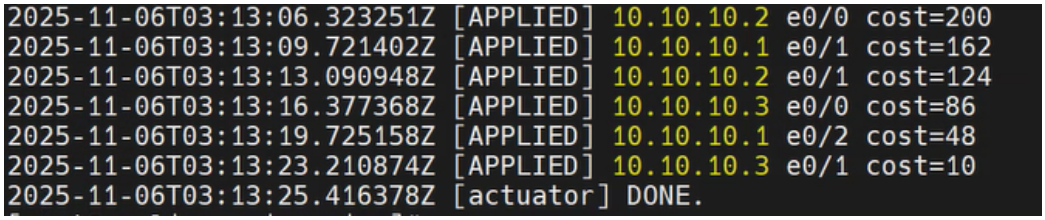
El *script* desarrollado lee directamente el archivo generado por el modelo de Machine Learning, ordenando los enlaces en función de su probabilidad de falla a 12 horas de mayor a menor. Con base en este orden, asigna costos OSPF de manera escalonada mediante una rampa lineal entre los valores máximo y mínimo.

Figura 111. Rampa de costos OSPF asignados a los enlaces MPLS según su *ranking* de riesgo de falla.



Nota. Elaboración propia. Visualización de los costos OSPF a los enlaces MPLS en función de su probabilidad de falla estimada. Los enlaces con mayor riesgo reciben costos más altos, incentivando su evitación en el enrutamiento.

Figura 112. Costos OSPF aplicados a los enlaces MPLS según su *ranking* de riesgo de falla en una ejecución real del *script*.



Nota. Elaboración propia. Costos OSPF efectivamente aplicados a los enlaces MPLS tras la ejecución del *script* de optimización dinámica.

Figura 113. Costo OSPF modificado en el enlace e0/0 del route reflector según la evolución temporal del riesgo de falla estimado.

```
RR#show ip ospf interface Ethernet0/0
Ethernet0/0 is up, line protocol is up
  Internet Address 10.1.1.6/30, Area 0, Attached via Interface Enable
  Process ID 1, Router ID 2.2.2.2, Network Type BROADCAST, Cost: 200
  Topology-MTID    Cost    Disabled    Shutdown      Topology Name
        0           200        no          no            Base
```

Figura 114. Costo OSPF modificado en el enlace e0/1 del route reflector según la evolución temporal del riesgo de falla estimado.

```
RR#show ip ospf interface Ethernet0/1
Ethernet0/1 is up, line protocol is up
  Internet Address 10.1.1.9/30, Area 0, Attached via Interface Enable
  Process ID 1, Router ID 2.2.2.2, Network Type BROADCAST, Cost: 124
  Topology-MTID    Cost    Disabled    Shutdown      Topology Name
        0           124        no          no            Base
```

Figura 115. Costo OSPF modificado en el enlace e0/1 del *router* P1 según la evolución temporal del riesgo de falla estimado.

```
P-1#show ip ospf interface Ethernet0/1
Ethernet0/1 is up, line protocol is up
  Internet Address 10.1.1.5/30, Area 0, Attached via Interface Enable
  Process ID 1, Router ID 1.1.1.1, Network Type BROADCAST, Cost: 162
  Topology-MTID    Cost    Disabled    Shutdown      Topology Name
        0           162        no          no            Base
```

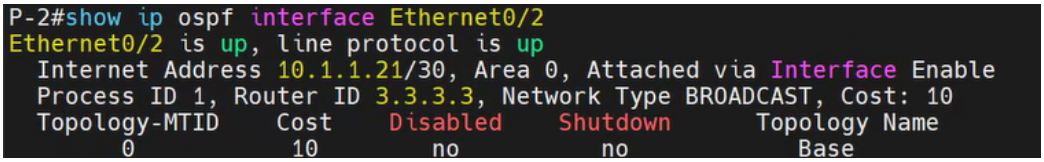
Figura 116. Costo OSPF modificado en el enlace e0/2 del *router* P1 según la evolución temporal del riesgo de falla estimado.

```
P-1#show ip ospf interface Ethernet0/2
Ethernet0/2 is up, line protocol is up
  Internet Address 10.1.1.1/30, Area 0, Attached via Interface Enable
  Process ID 1, Router ID 1.1.1.1, Network Type BROADCAST, Cost: 48
  Topology-MTID    Cost    Disabled    Shutdown      Topology Name
        0           48         no          no            Base
```

Figura 117. Costo OSPF modificado en el enlace e0/1 del *router* P2 según la evolución temporal del riesgo de falla estimado.

```
P-2#show ip ospf interface Ethernet0/1
Ethernet0/1 is up, line protocol is up
  Internet Address 10.1.1.2/30, Area 0, Attached via Interface Enable
  Process ID 1, Router ID 3.3.3.3, Network Type BROADCAST, Cost: 10
  Topology-MTID    Cost    Disabled    Shutdown      Topology Name
        0           10         no          no            Base
```

Figura 118. Costo OSPF modificado en el enlace e0/1 del *router* P2 según la evolución temporal del riesgo de falla estimado.



CAPÍTULO 15

Conclusiones

- Se demostró la implementación de redes privadas virtuales (VPN) de capa 2 y capa 3 sobre una infraestructura MPLS virtualizada, integrando procesos de monitoreo y análisis basados en inteligencia artificial para optimizar su desempeño.
- Se contruyó una red MPLS funcional y escalable mediante la configuración de VRF, LDP y BGP, lo que logró la propagación de rutas VPNv4 entre sitios remotos.
- Se confirmó la estabilidad de las VPN L2 y L3, el correcto aislamiento del tráfico entre clientes (A,B y C) y la eficiencia en el transporte de datos mediante etiquetas.
- Se integró el *software* Zabbix para recolectar métricas operativas de la red, incluyendo latencia, pérdida de paquetes, uso de ancho de banda y disponibilidad de los enlaces.
- Se desarrolló un módulo con inteligencia artificial que analiza los eventos generados por Zabbix y proporciona sugerencias textuales de diagnóstico, depuración y mitigación.
- El modelo predictivo, árbol de decisión, basado en aprendizaje supervisado, logró anticipar fallas en los enlaces del núcleo MPLS con resultados consistentes en distintos horizontes temporales. De esta forma se mantiene sensibilidad ante patrones de inestabilidad sin comprometer la estabilidad del sistema.
- Se validó el funcionamiento del algoritmo de optimización que modifica dinámicamente los costos OSPF en función de la estabilidad de los enlaces, priorizando rutas más confiables y reduciendo los tiempos de convergencia.

CAPÍTULO 16

Recomendaciones

- Escalar la topología MPLS incluyendo IPv6, más routers PE y enlaces redundantes para obtener de mejores escenarios para aplicar el monitoreo y la optimización a toda la red, no solo al núcleo.
- Implementar una VM única para Zabbix en lugar de una appliance para crear particiones separadas, asignar más memoria o CPU y usar bases de datos externas de alto rendimiento. Con ello se evitará la limitación de espacio QCOW2 y se mejorará la estabilidad y escalabilidad del monitoreo.
- Extender el Webhook para que interprete no solo alertas de Zabbix sino también resultados del modelo de predicción, generando sugerencias sobre acciones preventivas o correctivas y transformando el sistema en un asistente operativo inteligente.
- Ampliar el conjunto de variables predictoras, incorporando métricas adicionales de desempeño de red (latencia, pérdida de paquetes o jitter) que permitan mejorar la capacidad explicativa del modelo.
- Evaluar algoritmos adicionales de predicción, como Random Forest, Gradient Boosting o modelos secuenciales que podrían capturar relaciones no lineales y dependencias temporales más complejas que un árbol de decisión tradicional.

CAPÍTULO 17

Referencias

[1] Huawei Enterprise Support Community. «¿Qué es MPLS? Conceptos básicos: enrutamiento y conmutación. »dirección: https://forum.huawei.com/enterprise/intl/es/thread/%C2%BFqu%C3%A9-es-mpls-conceptos-b%C3%A1sicos-enrutamiento-y-conmutaci%C3%B3n/750561039969304576?blogId=750561039969304576.

[2] F. A. Cruz Quimbiulco y R. G. Montaluisa Toalisa, «Análisis y diseño de una red MPLS que soporte IPv6 con protocolos de enrutamiento OSPF, IS-IS y BGP emulando un escenario típico de empresas multinacionales,» Tesis doct., Universidad Politécnica Salesiana, Quito, Ecuador, 2017.

[3] C. P. Chávez Fuentes, «Ruta óptima de tráfico de una red virtual basada en análisis de datos y algoritmo de machine learning,» Tesis doct., Universidad Técnica de Ambato, Ambato, Ecuador, 2021.

[4] Á. González Carrasco, «Integración y optimización de redes MPLS: Un caso práctico,» Tesis doct., Universidad de Alcalá, Madrid, España, 2018.

[5] J. A. Arteaga Cabrera, «Diseño de una red privada virtual con tecnología MPLS,» Tesis doct., Universidad Católica de Santiago de Guayaquil, Guayaquil, Ecuador, 2019.

[6] NetworkLessons.com. «AI and ML in Networking. »dirección: https://networklessons.com/cisco/ccna-200-301/ai-and-ml-in-networking.

[7] V. E. N. Castro, «Desarrollo e implementación de un sistema para el monitoreo automático del estado operativo de los equipos de la red IP-MPLS de la Corporación Nacional de Telecomunicaciones CNT EP,» Tesis de mtría., Escuela Politécnica Nacional, Quito, Ecuador, 2017.

[8] L. D. Ghein, «MPLS Fundamentals: A Comprehensive Introduction to MPLS . Theory and Practice,» *Cisco Press*, págs. 7-55, 2006.

[9] C. Carthern, W. Wilson y N. Rivera, «Multiprotocol Label Switching,» en *Cisco Networks*, Springer, 2021, págs. 951-1016.

[10] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky y S. Uhlig, «Software-Defined Networking: A Comprehensive Survey,» *Proceedings of the IEEE*, vol. 103, n.º 1, págs. 14-76, 2015.

[11] S. Das, A. R. Sharafat, G. Parulkar y N. McKeown, «MPLS with a Simple Open Control Plane,» en *Optical Fiber Communication Conference and Exposition (OFC/NFOEC)*, 2011. dirección: https://yuba.stanford.edu/~nickm/papers/ofc11-saurav-mpls.pdf.

[12] N. Feamster, J. Rexford y E. Zegura, «The Road to SDN: An Intellectual History of Programmable Networks,» en *ACM SIGCOMM Computer Communication Review*, vol. 44, ACM, 2014, págs. 87-98. dirección: https://dl.acm.org/doi/10.1145/2602204.2602219.

[13] N. B. Labs, «Performance Metrics for Communication Systems with Forward Error Correction,» 2011.

[14] I. Minei y J. Lucek, «MPLS-Enabled Applications: Emerging Developments and New Technologies,» *Journal of Network and Systems Management*, vol. 15, n.º 2, págs. 241-256, 2007.

[15] D. Țurcanu, «Quality of Services in MPLS Networks,» *Journal of Engineering Science*, vol. XXVII, n.º 3, págs. 102-110, 2020.

[16] M. Lasserre y V. Kompella, «Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling,» *RFC 4762*, 2007.

[17] Huawei Technologies. «Configuration Guide - MPLS L2VPN. »dirección: https://support.huawei.com/enterprise/en/doc/EDOC1100055018.

[18] Nokia. «Service Router MPLS Guide. »dirección: https://documentation.nokia.com/sr/latest/mpls/mpls.html.

[19] Juniper Networks. «MPLS Applications Configuration Guide. »dirección: https://www.juniper.net/documentation/us/en/software/junos/mpls/topics/concept/mpls-applications-overview.html.

[20] Cisco Systems. «MPLS Layer 3 VPNs: Inter-AS and CSC Configuration Guide. »dirección: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mp_l3_vpns/configuration/xe-16/mp-l3-vpns-xe-16-book/mp-l3-vpns-inter-as.html.

[21] Juniper Networks. «EVPN-VXLAN Integration with MPLS-Based L2VPN. »dirección: https://www.juniper.net/documentation/us/en/software/junos/evpn-vxlan/topics/concept/evpn-vxlan-integration-with-mpls-based-l2vpn.html.

[22] L. Velasco, M. Ruiz, A. Castro, D. King, A. Farrel e Y. Zheng, «Orchestrating Transoceanic Lightpaths in Hybrid Cloud/Fog Environments,» *Journal of Optical Communications and Networking*, vol. 10, n.º 2, A225-A231, 2018. dirección: https://www.osapublishing.org/jocn/abstract.cfm?uri=jocn-10-2-A225.

[23] Cisco Systems. «Multiprotocol Label Switching over ATM (MPLS over ATM).» Accedido: mayo 2025. dirección: https://www.cisco.com/c/es_mx/support/docs/multiprotocol-label-switching-mpls/multiprotocol-label-switching-over-atm-mpls-over-atm/10474-mpls-packflow.html.

[24] Cisco Systems. «MPLS Layer 3 VPNs: Inter-AS and CSC Configuration Guide. »dirección: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/mp_l3_vpns/configuration/xe-16/mp-l3-vpns-xe-16-book/mp-l3-vpns-overview.html.

[25] E. Rosen e Y. Rekhter, «BGP/MPLS IP Virtual Private Networks (VPNs),» *RFC 4364*, 2006. dirección: https://datatracker.ietf.org/doc/rfc4364/.

[26] S. Masood, N. Kumar y J.-W. Kim, «Analysis of artificial intelligence based metaheuristic algorithm for MPLS network optimization,» *2018 20th International Conference on Transparent Optical Networks (ICTON)*, págs. 1-4, 2018.

[27] J. A. Carletto, «Optimización de tráfico en redes multiservicios aplicando técnicas heurísticas,» Tesis de mtría., Universidad Nacional de La Plata, Facultad de Informática, 2022.

[28] F. Rafamantanantsoa, C. Razafindramonja y L. H. Rabetafika, «Analysis and Evaluation of MPLS Network Performance,» *Communications and Network*, vol. 13, págs. 25-35, 2021.

[29] Cisco Systems, Inc. «Configuración de una red privada virtual (VPN) MPLS básica,» Cisco Systems, Inc. dirección: https://www.cisco.com/c/es_mx/support/docs/multiprotocol-label-switching-mpls/mpls/13733-mpls-vpn-basic.html.

[30] K. D. Hernández Guarc, «Automatización de la etapa de síntesis lógica, optimización del proceso de instalación de las aplicaciones de Synopsys y documentación de los pasos de Front-End y Back-End para el diseño del Circuito Integrado El Gran Jaguar,» Trabajo de graduación, Universidad del Valle de Guatemala, Guatemala, 2024.

[31] O. Yildiz y E. Alpaydın, «Univariate and Multivariate Decision Trees,» *Department of Computer Engineering, Bo˘gazi¸ci University, Istanbul*, 2012. dirección: https://www.researchgate.net/publication/267799938_Univariate_and_Multivariate_Decision_Trees.

[32] C. C. (ccaceresoln). «Script IA Help for Zabbix – Script.» Consultado el 22 de octubre de 2025. dirección: https://github.com/ccaceresoln/ScriptIAHelpforZabbix/blob/main/Script.

[33] GNS3 Team. «GNS3 Marketplace Appliances. »dirección: https://www.gns3.com/marketplace/appliances.

CAPÍTULO 18

Anexos

Código 32. Script Webhook para integración de Zabbix con Google Gemini [32]

```

var Gemini = {
    params: {},
    setParams: function(params) {
        if (typeof params !== 'object') {
            return;
        }
        Gemini.params = params;
        if (typeof Gemini.params.api_key !== 'string' || Gemini.params.api_key === '') {
            throw 'API key for Gemini is required.';
        }
        Gemini.params.url =
     ↪ 'https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent';
    },
    request: function(data) {
        if (!Gemini.params.api_key) {
            throw 'API key is missing.';
        }
        var request = new HttpRequest();
        request.addHeader('Content-Type: application/json');

        // Construir URL con API key
        var urlWithKey = Gemini.params.url + '?key=' + Gemini.params.api_key;

        Zabbix.log(4, '[ Gemini Webhook ] Sending request: ' + urlWithKey + '\n' + JSON.stringify(data));
        var response = request.post(urlWithKey, JSON.stringify(data));
        Zabbix.log(4, '[ Gemini Webhook ] Received response with status code ' + request.getStatus() + '\n'
     ↪ + response);

        if (request.getStatus() < 200 || request.getStatus() >= 300) {
            throw 'Gemini API request failed with status code ' + request.getStatus() + '.';
        }

        try {
            response = JSON.parse(response);
        } catch (error) {
            Zabbix.log(4, '[ Gemini Webhook ] Failed to parse response from Gemini.');
```

```
            response = null;
        }
        return response;
    }
};

try {
    var params = JSON.parse(value),
        data = {},
        result = "",
        required_params = ['alert_subject'];

    Object.keys(params).forEach(function(key) {
        if (required_params.indexOf(key) !== -1 && params[key] === '') {
            throw 'Parameter "' + key + '" cannot be empty.';
        }
    });

    // Formatear la consulta para Gemini
    data = {
        contents: [{
            parts: [{
                text: "La alerta: " + params.alert_subject + " ocurrio en Zabbix. " +
                      "Sugiere posibles causas y soluciones para resolver este problema, no hagas un texto
    ↪ grande, " +
                      "solamente unas 10 lineas con causas, ideas, comandos para debug y medidas para
    ↪ mitigar futuros incidentes."
            }]
        }]
    };

    // Configurar la API de Gemini
    Gemini.setParams({ api_key: params.api_key });

    // Hacer la solicitud a Gemini
    var response = Gemini.request(data);

    if (response && response.candidates && response.candidates.length > 0) {
        result = response.candidates[0].content.parts[0].text.trim();
    } else {
        throw 'No response from Gemini.';
    }

    return result;

} catch (error) {
    Zabbix.log(3, '[ Gemini Webhook ] ERROR: ' + error);
    throw 'Sending failed: ' + error;
}
\end

try {
    var params = JSON.parse(value),
        data = {},
        result = "",
        required_params = ['alert_subject'];

    Object.keys(params).forEach(function(key) {
        if (required_params.indexOf(key) !== -1 && params[key] === '') {
            throw 'Parameter "' + key + '" cannot be empty.';
        }
    });

    // Formatear la consulta para Gemini
    data = {
        contents: [{
            parts: [{
                text: "La alerta: " + params.alert_subject + " ocurrio en Zabbix. " +
```

```
                    "Sugiere posibles causas y soluciones para resolver este problema, no hagas un texto
    ↪ grande, " +
                    "solamente unas 10 lineas con causas, ideas, comandos para debug y medidas para
    ↪ mitigar futuros incidentes."
                }]
            }]
    };

    // Configurar la API de Gemini
    Gemini.setParams({ api_key: params.api_key });

    // Hacer la solicitud a Gemini
    var response = Gemini.request(data);

    if (response && response.candidates && response.candidates.length > 0) {
        result = response.candidates[0].content.parts[0].text.trim();
    } else {
        throw 'No response from Gemini.';
    }

    return result;

} catch (error) {
    Zabbix.log(3, '[ Gemini Webhook ] ERROR: ' + error);
    throw 'Sending failed: ' + error;
}
```

Nota. Elaboración propia.

Código 33. Instrucciones para la instalación de imágenes IOU en GNS3 [33]

```
    \begin{itemize}
        \item 1) Login Using SSH in GNS 3 VM
        \item 2) Update repositories - apt-get update after entering root mode using
    ↪ sudo su
        \item 3) Download Phthon - apt-get install python
        \item 4) Download the keygen - wget
    ↪ http://www.ipvanquish.com/download/CiscoIOUKeygen3f.py
        \item 4) Give permission - chmod 0755 CiscoIOUKeygen3f.py
        \item 5) Execute it - python3 CiscoIOUKeygen3f.py
        \item 6) Note the license key.
        \item 7) Browse to cd /opt/GNS3/images/IOU
        \item 8) Create a new file - IOURC.txt
        \item 9) Paste the license key in the following format[license]
        localhost.localdomain = 73635fd3b0a13ad0;
        \item 10) In GNS3 - Preferences -> IOU -> Select the path / paste the key in
    ↪ the same format.

    \end{itemize}
```

Nota. Elaboración propia.

Código 34. Ejemplo de respuesta generada por Google Gemini para un problema de espacio en disco en Zabbix

```
Causas:

*   Archivos de log crecientes (ej: /var/log).
*   Crecimiento inesperado de bases de datos.
*   Archivos temporales no limpiados (/tmp).
```

```
*   Usuarios guardando grandes archivos en /home.

Debug y Soluciones:

*   df -h:  Ver uso del disco.
*   du -hsx /* | sort -rh | head -20:  Identificar directorios grandes.
*   lsof | grep deleted: Encontrar archivos abiertos borrados que ocupan espacio.
*   Limpiar logs: truncate -s 0 /var/log/syslog.
*   Borrar temporales: rm -rf /tmp/*.

Mitigacion:

*   Rotacion de logs (logrotate).
*   Monitoreo de espacio en disco con alertas tempranas.
*   Cuotas de disco para usuarios.
*   Revision periodica del uso del disco.
```

Nota. Elaboración propia.

Código 35. Métricas de desempeño del modelo predictivo de fallas en enlaces MPLS basado en aprendizaje supervisado

```

=== y_down_1h ===
TRAIN=8390 (pos=91), TEST=2098 (pos=95)
Accuracy=0.8408 | Precision=0.0717 | Recall=0.2105 | F1=0.1070

Confusion Matrix = [[1744, 259], [75, 20]]

Class Metrics:
    precision   recall   f1-score   support
  0     0.9588     0.8707     0.9126      2003
  1     0.0717     0.2105     0.1070        95

Accuracy:      0.8408 (2098 samples)
Macro avg:     0.5152   0.5406   0.5098   2098
Weighted avg:  0.9186   0.8408   0.8761   2098


=== y_down_6h ===
TRAIN=8390 (pos=359), TEST=2098 (pos=370)
Accuracy=0.7193 | Precision=0.2141 | Recall=0.2216 | F1=0.2178

Confusion Matrix = [[1427, 301], [288, 82]]

Class Metrics:
    precision   recall   f1-score   support
  0     0.8321     0.8258     0.8289      1728
  1     0.2141     0.2216     0.2178       370

Accuracy:      0.7193 (2098 samples)
Macro avg:     0.5231   0.5237   0.5234   2098
Weighted avg:  0.7231   0.7193   0.7211   2098

```

```
33
34  === y_down_12h ===
35  TRAIN=8390 (pos=904), TEST=2098 (pos=401)
36  Accuracy=0.4824 | Precision=0.2387 | Recall=0.7805 | F1=0.3657
37
38  Confusion Matrix = [[699, 998], [88, 313]]
39
40  Class Metrics:
41      precision   recall   f1-score   support
42    0     0.8882     0.4119     0.5628      1697
43    1     0.2387     0.7805     0.3657       401
44
45  Accuracy:      0.4824 (2098 samples)
46  Macro avg:     0.5635   0.5962   0.4642   2098
47  Weighted avg:  0.7641   0.4824   0.5251   2098
48
49
50  === y_flap_3h ===
51  TRAIN=8390 (pos=208), TEST=2098 (pos=218)
52  Accuracy=0.7898 | Precision=0.1004 | Recall=0.1284 | F1=0.1127
53
54  Confusion Matrix = [[1629, 251], [190, 28]]
55
56  Class Metrics:
57      precision   recall   f1-score   support
58    0     0.8955     0.8665     0.8808      1880
59    1     0.1004     0.1284     0.1127       218
60
61  Accuracy:      0.7898 (2098 samples)
62  Macro avg:     0.4980   0.4975   0.4967   2098
63  Weighted avg:  0.8129   0.7898   0.8010   2098
64
65
66  === y_downtime20_12h ===
67  No entrenado
```

Nota. Elaboración propia.

CAPÍTULO 19

Glosario

***Border gateway protocol* (BGP).** Protocolo de enrutamiento exterior usado junto con extensiones multiprotocolo (MP-BGP) para propagar rutas VPNv4 en MPLS.

***Customer edge router* (CE).** Router de cliente conectado al proveedor de servicios para acceder a la red MPLS.

Graphical Network Simulator 3 (GNS3). Plataforma de simulación que permite emular routers y switches reales para escenarios de red complejos.

***Label distribution protocol* (LDP).** Protocolo que distribuye y asigna etiquetas MPLS entre routers adyacentes.

***Label edge router* (LER).** Router de borde que inserta o retira etiquetas MPLS cuando los paquetes entran o salen de la red.

***Label switching router* (LSR).** Router de núcleo que conmuta paquetes MPLS basándose en etiquetas en lugar de direcciones IP.

***Multiprotocol Label Switching* (MPLS).** Técnica de conmutación por etiquetas que acelera el reenvío de paquetes y soporta aplicaciones como VPN, QoS e ingeniería de tráfico.

***Provider edge router* (PE).** Equipo de frontera del proveedor que conecta la red MPLS con los routers de cliente (CE).

***Quality of service* (QoS).** Mecanismos para priorizar y garantizar niveles de servicio en voz, video y datos en redes MPLS.

***Route distinguisher* (RD).** Identificador único de 8 bytes que se añade a las rutas IP en MPLS VPN para evitar colisiones en el espacio de direcciones entre clientes.

***Route target* (RT).** Atributo BGP usado en MPLS VPN para definir qué rutas deben importarse o exportarse entre instancias VRF.

***Traffic engineering* (TE).** Proceso de mapear el tráfico en una red MPLS para optimizar el uso de enlaces, reducir congestión y mejorar la confiabilidad.

***Virtual private network* (VPN).** Red privada virtual que crea túneles seguros sobre infraestructuras públicas para garantizar confidencialidad e integridad en la comunicación.

***Virtual routing and forwarding* (VRF).** Instancia de tablas de enrutamiento independientes en un router PE que permite segmentar y aislar el tráfico de clientes.

***Virtual switching instance* (VSI).** Instancia virtual de conmutación ethernet usada en VPLS para implementar VPN de capa 2 sobre MPLS.

Zabbix. Herramienta de monitoreo de código abierto utilizada para supervisar rendimiento, disponibilidad y métricas de red en entornos MPLS.