

UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE TELECOMUNICACIONES

Tema:

SISTEMA DE DETECCIÓN DE ANOMALÍAS DE RENDIMIENTO Y SEGURIDAD EN REDES DE DATOS CON INTELIGENCIA ARTIFICIAL

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Telecomunicaciones

ÁREA: Programación y Redes

LÍNEA DE INVESTIGACIÓN: Software, Tecnologías de la información y Ciencia de datos

AUTOR: Ruben Mateo Sosa Castillo

TUTOR: Ing. Víctor Santiago Manzano Villafuerte, Mg.

Ambato - Ecuador

julio – 2025

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: SISTEMA DE DETECCIÓN DE ANOMALÍAS DE RENDIMIENTO Y SEGURIDAD EN REDES DE DATOS CON INTELIGENCIA ARTIFICIAL, desarrollado bajo la modalidad Proyecto de Investigación por el señor Ruben Mateo Sosa Castillo, estudiante de la Carrera de Telecomunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, julio 2025.

Firmado electrónicamente por:
VICTOR SANTIAGO
MANZANO VILLAFUERTE
Validar únicamente con FirmaEC

Ing. Víctor Santiago Manzano Villafuerte, Mg.
TUTOR

AUTORÍA

El presente trabajo de titulación con el tema: SISTEMA DE DETECCIÓN DE ANOMALÍAS DE RENDIMIENTO Y SEGURIDAD EN REDES DE DATOS CON INTELIGENCIA ARTIFICIAL es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, julio 2025.



Ruben Mateo Sosa Castillo

C.C. 1721139119

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, julio 2025.



Ruben Mateo Sosa Castillo

C.C. 1721139119

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Ruben Mateo Sosa Castillo, estudiante de la Carrera de Telecomunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado : SISTEMA DE DETECCIÓN DE ANOMALÍAS DE RENDIMIENTO Y SEGURIDAD EN REDES DE DATOS CON INTELIGENCIA ARTIFICIAL, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con el señor Presidente del Tribunal.

Ambato, julio 2025.

Firmado electrónicamente por:
FRANKLIN OSWALDO MAYORGA MAYORGA
Validar únicamente con FirmaEC

Ing. Franklin Mayorga, Mg.
PRESIDENTE DEL TRIBUNAL

Firmado electrónicamente por:
LIVIO DANILO MINIGUANO MINIGUANO
Validar únicamente con FirmaEC

Ing. Danilo Miniguano Miniguano, Mg.
PROFESOR CALIFICADOR

Firmado electrónicamente por:
EDGAR FREDDY ROBALINO PENA
Validar únicamente con FirmaEC

Ing.Freddy Robalino Peña, Mg.
PROFESOR CALIFICADOR

ÍNDICE GENERAL DE CONTENIDOS

APROBACIÓN DEL TUTOR ..... **ii**

AUTORÍA ..... **iii**

DERECHOS DE AUTOR ..... **iv**

APROBACIÓN DEL TRIBUNAL DE GRADO ..... **v**

ÍNDICE GENERAL DE CONTENIDOS ..... **vi**

ÍNDICE DE TABLAS ..... **x**

ÍNDICE DE FIGURAS ..... **xi**

ÍNDICE DE ANEXOS ..... **xvii**

RESUMEN EJECUTIVO ..... **xviii**

ABSTRACT ..... **xix**

CAPÍTULO I. MARCO TEÓRICO **1**

1.1 Tema de investigación 1

1.1.1 Planteamiento del problema 1

1.2 Antecedentes investigativos 3

1.3 Fundamentación teórica 4

1.3.1 Las redes de datos 4

1.3.2 Red definida por software (SDN) con ZeroTier 5

1.3.3 Contenedores y orquestación: Docker y Kubernetes en redes distribuidas 6

1.3.4 Equipos de Red: Clasificación, Funciones y Aplicaciones 8

1.3.5 Protocolos de monitoreo de redes: SNMP, NetFlow y Syslog 9

1.3.6 Preprocesamiento y tratamiento de datos en entornos distribuidos 12

1.3.7 Almacenamiento de datos en sistemas distribuidos 16

1.3.8 Visualización de datos y resultados 18

1.3.9 Aplicación de modelos de deep learning para detección de anomalías en redes de datos. 19

1.3.10 Técnicas y consideraciones metodológicas clave en el entrenamiento de modelos de detección de anomalías. 23

1.3.11 Repositorios de Control de Versiones 27

1.4 Objetivos 28

1.4.1 Objetivo general 28

1.4.2 Objetivos específicos 28

CAPÍTULO II. METODOLOGÍA 29

2.1 Materiales 29

2.1.1 Dispositivos de red 29

2.1.2 Software utilizado 30

2.1.3 Librerías de Python utilizadas 31

2.2 Métodos 31

2.2.1 Modalidad de la investigación 31

2.2.2 Recolección de información 32

2.2.3 Procesamiento y análisis de datos 33

CAPÍTULO III. RESULTADOS Y DISCUSIÓN 37

3.1 Arquitectura de red 37

3.1.1 Conexión entre redes mediante ZeroTier 37

3.1.2 Configuración de protocolos de monitoreo 42

3.2 Arquitectura de microservicio 46

3.2.1 Levantamiento de máquina virtual con Kubernetes para implementar Docker 47

3.3 Control de versiones y repositorio 50

3.3.1 Integración con GitLab para control de versiones y gestión de contenedores .. 50

3.4 Sistema de detección de anomalías de rendimiento y seguridad en redes de datos 52

3.4.1 Ingesta, preprocesamiento y transformación de datos con Logstash y Filebeat 52

3.4.2 Indexación de resultados hacia Elasticsearch 55

3.4.3 Visualización de información mediante dashboards en Kibana 56

3.4.4 Aplicación de algoritmos de deep learning 60

3.5 Presupuesto 78

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES 79

4.1 Conclusiones 79

4.2 Recomendaciones 80

REFERENCIAS BIBLIOGRÁFICAS 81

ANEXOS 84

ÍNDICE DE TABLAS

Tabla 1. Tabla comparativa entre arquitectura monolítica y de microservicios 7

Tabla 2. Clasificación de Equipos de Red según su Función en la Arquitectura 8

Tabla 3. Tipos de Equipos de Red y sus Funciones 8

Tabla 4. Fabricantes y Modelos Representativos 9

Tabla 5. Tabla comparativa de protocolos de monitoreo de red 12

Tabla 6. Comparativa de Herramientas para Procesamiento de Flujos de Datos 15

Tabla 7. Hiperparámetros de RNN 21

Tabla 8. Hiperparámetros de LSTM 22

Tabla 9. Hiperparámetros de GRU 22

Tabla 10. Comparación entre modelos RNN, LSTM y GRU 23

Tabla 11. Comparativa de Hiperparámetros – RNN vs LSTM vs GRU 23

Tabla 12. Comparativa de Métricas de Evaluación Basadas en Error 25

Tabla 13. Comparativa para Establecimiento de Umbrales de Detección de Anomalías 26

Tabla 14. Comparativa de Repositorios para Control de Versiones 27

Tabla 15. Recolección de Información. 33

Tabla 16. Comparación de desempeño entre modelos RNN, LSTM y GRU 70

Tabla 17. Presupuesto económico durante el desarrollo del proyecto 78

ÍNDICE DE FIGURAS

Figura 1. Análisis comparativo de capacidad de transmisión con VPN 5

Figura 2. Comparación de LogStash [32] 14

Figura 3. Logstash and FluentD benchmarks [32] 15

Figura 4. Ranking de motores de base de datos [34] 17

Figura 5. Diagrama de conexión de alto nivel entre redes mediante ZeroTier 37

Figura 6. Panel de control ZeroTier, autorización y validación de nodos 38

Figura 7. Direccionamiento lógico EMP1 38

Figura 8. Direccionamiento lógico EMP2 38

Figura 9. Direccionamiento lógico EMP3 39

Figura 10. VM en Proxmox 39

Figura 11. Direccionamiento lógico de nodo central 39

Figura 12. Tabla de enrutamiento red ZeroTier 40

Figura 13 Tabla de enrutamiento equipo core EMP1 40

Figura 14. Tabla de enrutamiento equipo FW EMP1 40

Figura 15. Flujo de comunicación y enrutamiento hacia nodo central – EMP1 40

Figura 16. Habilitación de reenvío entre interfaces equipo PC - EMP1 41

Figura 17. Interfaz de red local y red ZeroTier 41

Figura 18. Pruebas de comunicación desde Router core hacia nodo central 41

Figura 19. Pruebas de comunicación desde Firewall hacia nodo central 42

Figura 20. Configuración SNMP Walk en equipo Mikrotik 42

Figura 21. Configuración SNMP Trap en equipo Mikrotik 43

Figura 22. Configuración SNMP Walk y SNMP Trap en equipo Fortinet 43

Figura 23. Configuración Netflow en equipo Fortinet 44

Figura 24. Configuración IPFIX en equipo Mikrotik 44

Figura 25. Configuración Syslog en equipo Mikrotik 45

Figura 26. Configuración Syslog en equipo Fortinet 45

Figura 27. Validación de tráfico desde el nodo sucursal hacia el nodo central 46

Figura 28. Validación de tráfico desde el nodo central 46

Figura 29. Descripción de la arquitectura basada en microservicios para el sistema de detección de anomalías de rendimiento y seguridad en redes de datos. 47

Figura 30. Estado de Pods, Deployments y StatefulSet 49

Figura 31. Estado de Pods configurados como Deployment 49

Figura 32. Estado de Pods configurados como Statefulset 49

Figura 33. Estado de Pods 49

Figura 34 Arquitectura de ficheros de configuración para Kubernets 50

Figura 35. Dockerfile para crear imagen de docker personalizada para Logstash..... 51

Figura 36. Docker en Gitlab para poder utilizar en kubernets 51

Figura 37. Arquitectura de ficheros en Gitlab para control de versiones y cambios . 52

Figura 38 Código para configurar ZeroTier en nodo central dentro del pod collector 53

Figura 39 Código para configurar Filebeat dentro del pod collector 53

Figura 40 Código para configuración de parámetros de Filebeat para recibir tráfico IPFIX/Netflow 53

Figura 41 Código para configurar Logstash dentro del pod collector con imagen de Docker personalizada 54

Figura 42. Configuración de parámetros de Filebeat para recibir tráfico IPFIX/Netflow 54

Figura 43 Flujo de transferencia de información mediante los contenedores dentro del pod collector 54

Figura 44. Visualización y comparación de índices en Elasticsearch 55

Figura 45. Visualización de datos de Elasticsearch en Kibana 56

Figura 46. Dashboard de tráfico NetFlow – Visualización de volumen y distribución de paquetes 57

Figura 47. Dashboard de tráfico NetFlow – Visualización de volumen y distribución de paquetes 57

Figura 48. Relaciones entre direcciones IP – Grafo de comunicación NetFlow 58

Figura 49. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino 58

Figura 50. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino 59

Figura 51. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino 59

Figura 52. Código para Extracción y carga de datos desde Elasticsearch para múltiples fuentes (NetFlow, SNMP Walk, SNMP Hardware, Syslog) aplicando filtros por dirección IP para Fortinet y MikroTik. 61

Figura 53. Código para Eliminación de columnas con más del 75% de valores nulos 61

Figura 54. Código para Eliminación de columnas con valores únicos 61

Figura 55. Código para análisis de cardinalidad en columnas categóricas 62

Figura 56. Gráfico de barras que representa la cardinalidad (número de valores únicos) de cada columna categórica para identificar columnas con alta o baja diversidad de valores. 62

Figura 57. Código para eliminar columnas con muchos nulos e imputar valores faltantes numéricos usando la mediana. 62

Figura 58. Código para normalizar variables numéricas con MinMaxScaler y codificar variables categóricas con LabelEncoder 63

Figura 59. Matriz de correlación original de las variables 63

Figura 60. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas 64

Figura 61. Resumen de variables numéricas de NetFlow con tipo de dato, porcentaje de nulos y cantidad de valores únicos. 64

Figura 62. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – SNMP Walk 65

Figura 63. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – Syslog Fortinet 65

Figura 64. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – Syslog Mikrotik 66

Figura 65. Cantidad de registros para evaluar los modelos RNN, GRU y LSTM 67

Figura 66. Código para generar secuencias temporales a partir de variables numéricas normalizadas 68

Figura 67. Código de configuración y entrenamiento del modelo con definición dinámica de hiperparámetros (units, batch_size, epochs) y uso de Early Stopping para evitar sobreajuste 68

Figura 68. Código para evaluar modelos RNN, GRU y LSTM 69

Figura 69. Desempeño del modelo RNN bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R^2 y EVS 69

Figura 70. Desempeño del modelo GRU bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R^2 y EVS 70

Figura 71. Desempeño del modelo LSTM bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R^2 y EVS 70

Figura 72. Comparativa de MSE por tipo de modelo 71

Figura 73. Código para registro automático cada 5 minutos 72

Figura 74. Código para cargar modelo entrenado, realiza la detección de anomalías en base al umbral establecido por el percentil 99. 73

Figura 75. Código para guardar los resultados en la base de datos SQLite 73

Figura 76. Carga de anomalías NetFlow y SNMPWalk en Streamlit 74

Figura 77. Visualización de anomalías NetFlow en Streamlit 75

Figura 78. Visualización de anomalías SNMPWalk en Streamlit 75

Figura 79. Visualización en Streamlit de anomalías detectadas en tráfico NetFlow para todos los nodos. 76

Figura 80. Visualización en Streamlit de anomalías detectadas en tráfico NetFlow para el nodo con ip 10.2.2.1. 76

Figura 81 Visualización en Streamlit de anomalías detectadas a partir de datos SNMP para todos los nodos. 77

Figura 82. Visualización en Streamlit de errores de reconstrucción y anomalías detectadas a partir de datos SNMP para el nodo con ip 10.2.2.1. 77

ÍNDICE DE ANEXOS

Anexo A. Código completo para recepción y filtrado de datos para indexación en Elasticsearch..... 84

RESUMEN EJECUTIVO

El presente proyecto de investigación tiene como objetivo principal la implementación de un sistema de detección de anomalías de rendimiento y seguridad en redes de datos, basado en inteligencia artificial. La creciente complejidad de las infraestructuras de red, el aumento exponencial del tráfico y la sofisticación de las amenazas cibernéticas, exigen soluciones tecnológicas avanzadas que permitan una supervisión continua, precisa y proactiva.

La metodología empleada se basa en técnicas de deep learning, utilizando modelos como redes neuronales recurrentes (RNN), Long Short-Term Memory (LSTM) y Gated Recurrent Units (GRU), aplicadas al análisis de datos provenientes de protocolos de monitoreo de red como SNMP, NetFlow y Syslog. La información se recolecta desde múltiples redes interconectadas mediante ZeroTier, lo que permite una integración segura y eficiente de entornos distribuidos.

El procesamiento, almacenamiento y visualización de los datos se lleva a cabo mediante herramientas como Logstash, Elasticsearch, Kibana y Streamlit, en el marco de una arquitectura modular y escalable implementada con Docker y Kubernetes. Esta estructura permite gestionar grandes volúmenes de datos en tiempo real y facilita la implementación de modelos de detección de anomalías.

El sistema desarrollado permite identificar de forma oportuna desviaciones significativas en el comportamiento normal del tráfico de red, facilitando la toma de decisiones y reduciendo el riesgo de fallas o intrusiones. Esta solución contribuye al fortalecimiento de la ciberseguridad y a la optimización del rendimiento en redes empresariales.

Palabras clave: Detección de anomalías, inteligencia artificial, redes de datos, monitoreo de red.

ABSTRACT

This research project aims to implement a system for detecting performance and security anomalies in data networks, based on artificial intelligence. The increasing complexity of network infrastructures, the exponential growth of traffic, and the sophistication of cyber threats require advanced technological solutions that enable continuous, accurate, and proactive monitoring.

The methodology is based on deep learning techniques, using models such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU), applied to the analysis of data collected through network monitoring protocols such as SNMP, NetFlow, and Syslog. The information is gathered from multiple networks interconnected via ZeroTier, allowing secure and efficient integration of distributed environments.

Data processing, storage, and visualization are performed using tools such as Logstash, Elasticsearch, Kibana, and Streamlit, within a modular and scalable architecture deployed with Docker and Kubernetes. This framework enables real-time management of large volumes of data and facilitates the implementation of anomaly detection models.

The developed system identifies significant deviations from normal network behavior in a timely manner, supporting informed decision-making and reducing the risk of failures or intrusions. This solution contributes to strengthening cybersecurity and optimizing performance in enterprise networks.

Keywords: Anomaly detection, deep learning, data networks, network monitoring

CAPÍTULO I. MARCO TEÓRICO

1.1 Tema de investigación

"SISTEMA DE DETECCIÓN DE ANOMALIAS DE RENDIMIENTO Y SEGURIDAD EN REDES DE DATOS CON INTELIGENCIA ARTIFICIAL"

1.1.1 Planteamiento del problema

Las redes de datos desempeñan un papel fundamental en el funcionamiento de las organizaciones modernas, ya que soportan una amplia gama de servicios críticos, como la comunicación, la transferencia de datos, y el acceso a aplicaciones y sistemas. Sin embargo, el rápido crecimiento de las redes, la complejidad de su arquitectura y la constante evolución de las amenazas de seguridad plantean grandes desafíos en cuanto a su gestión y protección. La capacidad de detectar anomalías de rendimiento y seguridad en las redes de datos es crucial para garantizar su operatividad, evitar interrupciones en los servicios y prevenir ciberataques. A pesar de la disponibilidad de diversas soluciones tecnológicas, los métodos tradicionales de monitoreo y gestión de redes suelen ser reactivos, costosos y limitados en su capacidad para abordar las complejidades actuales [1].

A nivel global, las redes de datos enfrentan un desafío crítico, mantener un equilibrio entre el crecimiento exponencial del tráfico de datos y la seguridad frente a amenazas cibernéticas. Tecnologías como el Internet de las Cosas (IoT), el trabajo remoto y los servicios en la nube han aumentado exponencialmente el volumen de datos gestionados. Según Cisco Annual Internet Report, se proyecta que el tráfico IP global alcance los 396 exabytes por mes en 2026, mientras que el número de dispositivos conectados superará los 29 mil millones para 2030. Este crecimiento no solo amplifica los problemas de rendimiento, como latencias elevadas y pérdidas de paquetes, sino que también aumenta la superficie de ataque para ciberamenazas sofisticadas [2].

La implementación de sistemas basados en inteligencia artificial (IA) para la detección de anomalías en redes emerge como una solución crítica. Estos sistemas utilizan algoritmos avanzados de aprendizaje automático como machine learning y ML para

identificar patrones anómalos en tiempo real, ofreciendo una ventaja estratégica frente a métodos tradicionales de monitoreo, que a menudo reaccionan de manera reactiva y no preventiva [3].

Las organizaciones enfrentan múltiples desafíos relacionados con el rendimiento y la seguridad de sus redes. Según Gartner, el 60% de los fallos en redes empresariales no se detectan hasta que los usuarios finales reportan problemas, lo que genera un impacto significativo en la productividad y la experiencia del cliente [4].

Desde el punto de vista de la seguridad, informes de IBM Security señalan que el tiempo promedio para identificar y contener una brecha de seguridad en 2023 fue de 287 días, lo que incrementa el impacto financiero y operacional. Por otro lado, las amenazas como los ataques de denegación de servicio distribuido (DDoS) y la exfiltración de datos han evolucionado en sofisticación, con un promedio de *12.1 millones de ataques DDoS reportados solo en la primera mitad de 2023 (NETSCOUT) [5].

En el ámbito del rendimiento, sectores como los ISP y las infraestructuras críticas en financieras, sanitarias y de manufactura son particularmente vulnerables. Según un informe de Gartner, las empresas pueden perder en promedio alrededor de $300,000 por cada hora de inactividad de la red, mientras que las limitaciones en la detección temprana de anomalías agravan la situación [6].

La creciente cantidad de dispositivos conectados y la dependencia de aplicaciones en la nube han hecho que las redes sean más complejas y difíciles de gestionar. Esto contribuye a la incapacidad de detectar anomalías de manera oportuna, lo que a su vez aumenta las interrupciones en los servicios críticos y genera pérdidas económicas significativas [7].

Sin herramientas adecuadas para monitorear las redes de forma eficiente, las organizaciones no pueden identificar anomalías en tiempo real. Esto deriva en problemas como la pérdida de confianza por parte de clientes y socios, ya que las fallas frecuentes crean una percepción negativa sobre la capacidad operativa [8].

Los enfoques reactivos y basados en reglas limitan la capacidad de las redes para adaptarse rápidamente a amenazas o problemas emergentes. Esta limitación incrementa la exposición a ataques cibernéticos y dificulta la prevención de robos de información confidencial [9].

La falta de expertos en análisis de datos impide una respuesta rápida y efectiva a los problemas de red. Esto no solo retrasa la solución de incidentes, sino que también incrementa los costos asociados a la recuperación y la mitigación de los efectos tras un ataque o una falla [10].

1.2 Antecedentes investigativos

En 2019, el artículo "MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks" propone un modelo basado en redes generativas adversariales (GAN) para detectar anomalías en series temporales multivariadas en sistemas ciberfísicos (CPS). Este método utiliza redes neuronales recurrentes de memoria a largo plazo (LSTM-RNN) para capturar correlaciones temporales y generar puntuaciones de anomalía combinando discriminación y reconstrucción. El modelo MAD-GAN demostró una precisión de casi el 100% y un valor F1 de 0.77 en el conjunto de datos SWaT, superando significativamente a métodos tradicionales como PCA y Autoencoders. En el caso del conjunto WADI, alcanzó una tasa de detección de anomalías (recall) cercana al 100% [11].

En 2020 se publicó "On the Usage of Generative Models for Network Anomaly Detection in Multivariate Time-Series", estudio que introduce Net-GAN, un enfoque basado en redes neuronales recurrentes y redes generativas adversariales para la detección de anomalías en series temporales multivariadas de tráfico de red. Los resultados experimentales muestran que Net-GAN logra una precisión del 95% en la detección de anomalías, superando a métodos tradicionales en un 10% [7].

En 2021, en el artículo "Human Readable Network Troubleshooting Based on Anomaly Detection and Feature Scoring". Este artículo presenta un sistema que combina detección de anomalías y mecanismos de atención para facilitar la resolución de problemas en redes. La evaluación muestra una concordancia del 85% con expertos

humanos en la identificación de anomalías y una reducción del tiempo de diagnóstico en un 30% [12].

En 2023 se publicó “Detección de Anomalías en el Aprendizaje Automático: Ejemplos y Métodos", este artículo de IBM explora cómo las técnicas de aprendizaje automático se utilizan para detectar comportamientos anómalos en datos. Aunque no proporciona métricas específicas, destaca la eficacia de métodos como KNN y LOF en la identificación de anomalías en diversos contextos. [13].

En 2023 se presenta “Detecting Contextual Network Anomalies with Graph Neural Networks", este trabajo propone el uso de redes neuronales gráficas para la detección de anomalías contextuales en el tráfico de red. Evaluado con datos reales de la red Abilene, el modelo alcanzó una tasa de detección de anomalías del 87% y una reducción del 15% en falsos positivos en comparación con métodos basados en PCA y RNN [13].

En 2024 se presenta “Evaluating ML-Based Anomaly Detection Across Datasets of Varied Integrity: A Case Study". Este estudio evalúa la eficacia de algoritmos de detección de anomalías basados en aprendizaje automático, como Random Forest, en conjuntos de datos de tráfico de red con diferentes niveles de integridad. Los resultados indican que, incluso en conjuntos de datos con integridad reducida, el modelo mantiene una precisión superior al 90% en la detección de anomalía [14].

1.3 Fundamentación teórica

1.3.1 Las redes de datos

Las redes de datos constituyen el pilar fundamental de las comunicaciones modernas, permitiendo el intercambio de información entre dispositivos y sistemas. Según Tanenbaum y Wetherall, una red de datos se define como un conjunto de dispositivos interconectados que comparten recursos y servicios mediante protocolos estandarizados. La clasificación de redes incluye LAN, WAN y MAN, siendo estas esenciales para conectar usuarios y garantizar un flujo continuo de información.

Actualmente, las redes deben manejar un volumen creciente de datos, lo que incrementa los desafíos en su administración y optimización [15].

1.3.2 Red definida por software (SDN) con ZeroTier

ZeroTier es una solución de red definida por software (SD-WAN) que permite crear redes virtuales privadas con comunicación directa entre nodos, sin necesidad de configuraciones complejas de VPN tradicionales. ZeroTier opera en la capa 2 y 3 del modelo OSI y combina tecnologías de encriptación, direccionamiento automático y descubrimiento de pares para ofrecer conectividad segura entre dispositivos remotos [16].

Entre sus principales ventajas se destacan la facilidad de implementación, soporte para NAT traversal, cifrado extremo a extremo, y su capacidad para funcionar sin una infraestructura intermedia fija. A diferencia de alternativas como WireGuard o OpenVPN, ZeroTier no requiere intervención en firewalls o configuración de puertos específicos. Sin embargo, presenta limitaciones en personalización avanzada y dependencias en su red de control para autenticación [17].

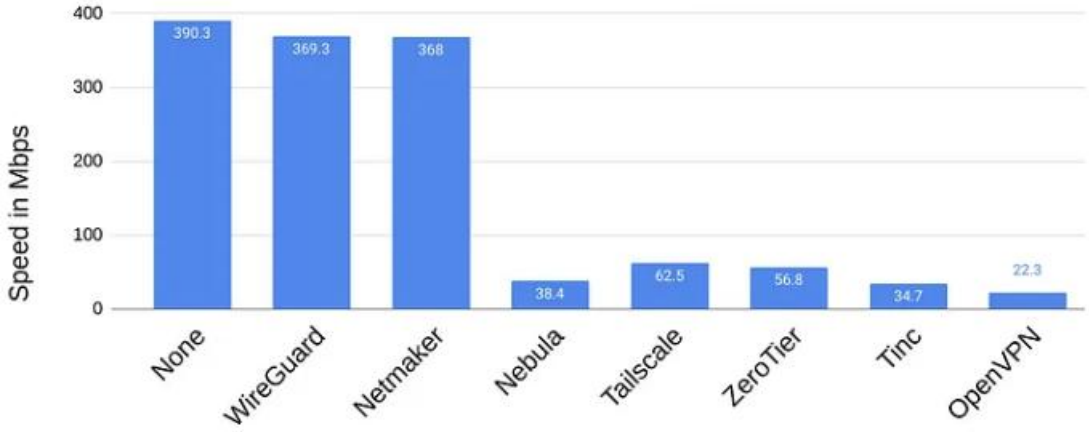


Figura 1. Análisis comparativo de capacidad de transmisión con VPN [17].

1.3.3 Contenedores y orquestación: Docker y Kubernetes en redes distribuidas

Docker es una plataforma de virtualización ligera que permite ejecutar aplicaciones en contenedores, encapsulando sus dependencias para lograr portabilidad y consistencia. Kubernetes es un sistema de orquestación de contenedores que permite escalar, balancear y gestionar automáticamente aplicaciones distribuidas, supervisando su estado en tiempo real [18].

En el contexto de redes de datos, estas tecnologías permiten desplegar servicios de monitoreo, almacenamiento, análisis e inteligencia artificial de forma modular y aislada. El uso de namespaces en Kubernetes facilita la segmentación lógica de funciones (recolección, visualización, análisis). Además, su compatibilidad con volúmenes persistentes permite mantener registros históricos para análisis forense y entrenamiento de modelos [19].

Docker es una plataforma de virtualización ligera que permite encapsular aplicaciones y sus dependencias en contenedores. Estos contenedores son portables, reproducibles y se ejecutan de forma aislada sobre el sistema operativo del host, lo cual facilita el despliegue homogéneo de servicios como colectores, analizadores y visualizadores de datos [20].

Kubernetes, por su parte, es un sistema de orquestación de contenedores que automatiza el despliegue, la gestión, el escalado y la recuperación de aplicaciones. Funciona mediante la definición de recursos en archivos YAML que describen pods, servicios, volúmenes persistentes y namespaces, permitiendo el control granular de la infraestructura [21].

Kubernetes y Docker ofrecen ventajas sustanciales frente a arquitecturas monolíticas tradicionales: tolerancia a fallos, despliegue automatizado, independencia de hardware subyacente y facilidad para actualizaciones en caliente [21].

La arquitectura de software puede adoptar dos enfoques principales: monolítica y de microservicios. En la arquitectura monolítica, todos los componentes de una aplicación están integrados en una única unidad, lo que facilita el desarrollo inicial,

pero puede generar desafíos a medida que la aplicación crece, como dificultades para escalar, implementar nuevas características o mantener el código [21].

Por otro lado, la arquitectura de microservicios divide una aplicación en servicios pequeños e independientes que se comunican entre sí a través de interfaces bien definidas. Cada microservicio se centra en una función específica y puede desarrollarse, implementarse y escalarse de forma autónoma. Este enfoque permite una mayor flexibilidad, agilidad en el desarrollo y escalabilidad eficiente, facilitando la implementación de nuevas características y la adaptación a cambios en el negocio, en la Tabla 1 se detalla una comparativa en una arquitectura monolítica y una basada en microservicios donde se destaca su escalabilidad, desarrollo y mantenimiento [21].

Tabla 1. Tabla comparativa entre arquitectura monolítica y de microservicios

Categoría	Arquitectura monolítica	Arquitectura de microservicios
Diseño	Código base único con múltiples funciones interdependientes.	Componentes de software independientes con funcionalidad autónoma que se comunican entre sí mediante API.
Desarrollo	Requiere menos planificación inicial, pero se vuelve más compleja de mantener con el tiempo.	Requiere más planificación e infraestructura, pero se hace más fácil de administrar y mantener.
Implementación	La aplicación se implementa como una sola entidad.	Cada microservicio es una unidad independiente que se implementa individualmente en contenedores.
Depuración	El seguimiento del código se hace en el mismo entorno.	Requiere herramientas de depuración avanzadas para rastrear la interacción entre microservicios.
Modificación	Cambios pequeños pueden impactar todo el sistema.	Los microservicios pueden modificarse de forma independiente sin afectar toda la aplicación.
Escalado	Se escala la aplicación completa, incluso si solo una parte lo necesita.	Se escalan solo los microservicios que lo requieren, optimizando recursos y costos.
Inversión	Baja inversión inicial, pero mayores costos en mantenimiento a largo plazo.	Mayor inversión inicial en infraestructura y capacitación, pero mejores resultados en escalabilidad y mantenimiento.

1.3.4 Equipos de Red: Clasificación, Funciones y Aplicaciones

Los equipos de red son dispositivos especializados que permiten la interconexión, el encaminamiento, el control y la gestión del tráfico dentro de una infraestructura de datos. Su correcto uso y ubicación dentro de una arquitectura en capas —como se muestra en la Tabla 2— es fundamental para asegurar la eficiencia operativa. Cada capa (acceso, distribución, núcleo y perímetro) cumple una función específica y emplea dispositivos adecuados para dicha función. Además, según la Tabla 3, existen distintos tipos de equipos (como switches, routers, firewalls, puntos de acceso, balanceadores de carga e IDS/IPS), cada uno con roles técnicos particulares que contribuyen a la conectividad, seguridad y disponibilidad del entorno. Finalmente, tal como lo refleja la Tabla 4, diversos fabricantes ofrecen modelos con capacidades específicas que permiten seleccionar las soluciones más apropiadas para cada necesidad, garantizando así un alto rendimiento, escalabilidad y control en la red [22].

Tabla 2. Clasificación de Equipos de Red según su Función en la Arquitectura

Capa	Función Principal	Dispositivos Comunes
Acceso	Conectar dispositivos finales (PC, impresoras, cámaras IP)	Switches de acceso, puntos de acceso WiFi
Distribución	Agregación de tráfico, control de políticas y VLANs	Switches L3, routers inter-VLAN, firewalls
Core/Núcleo	Alta velocidad de conmutación y encaminamiento entre zonas de red	Switches de núcleo, routers troncales
Perímetro	Control de entrada/salida hacia Internet o redes externas	Firewalls, proxies, balanceadores de carga

Tabla 3. Tipos de Equipos de Red y sus Funciones

Tipo de Equipo	Descripción y Funciones Principales
Switches (Conmutadores)	- Operan en la capa 2 (enlace) del modelo OSI.- Segmentan redes mediante VLANs.- L2 usa MAC tables; L3 hace routing.
Routers (Enrutadores)	- Operan en la capa 3 (red).- Determinan rutas óptimas entre redes.- Soportan OSPF, BGP, RIP, entre otros.
Firewalls	- Inspeccionan y filtran tráfico según políticas.- Físicos o virtuales.- Soportan UTM: antivirus, IPS, filtrado web.
Access Points (AP)	- Proveen conectividad inalámbrica.- Funcionan autónomos o controlados (CAPsMAN, WLC).
Balanceadores de Carga	- Distribuyen tráfico entre múltiples servidores.- Mejoran disponibilidad y tolerancia a fallos.
IDS/IPS	- Detectan y previenen intrusiones.- Analizan tráfico en tiempo real.- Se ubican en el perímetro o red interna.

Tabla 4. Fabricantes y Modelos Representativos

Fabricante	Modelo	Tipo	Capacidades Destacadas
MikroTik	CCR2116-12G-4S+, CRS328-24P-4S+	Router/Switch	Alto rendimiento, SNMP, NetFlow, VLAN, CAPsMAN
Fortinet	FortiGate FGT40F	Firewall	IPS/IDS, control de aplicaciones, VPN, visibilidad avanzada
Cisco	Catalyst 2960X, ISR 1000	Switch/Router	Estándar de la industria, QoS, EIGRP, OSPF
Aruba/HPE	Aruba 2930F	Switch	Integración WiFi, gestión cloud, VLAN dinámicas

Los equipos de red, más allá de su función de transporte, son una fuente primaria de datos de telemetría. A través de protocolos como SNMP (para métricas de hardware e interfaces), Syslog (para eventos de sistema y seguridad) y NetFlow/IPFIX (para análisis de tráfico), estos dispositivos permiten recolectar información crítica para alimentar sistemas de detección de anomalías basados en inteligencia artificial. La correcta selección, configuración y monitoreo de estos equipos es determinante para mantener una infraestructura de red segura, eficiente y confiable [23].

1.3.5 Protocolos de monitoreo de redes: SNMP, NetFlow y Syslog

Los protocolos de monitoreo de redes son esenciales para obtener visibilidad del estado, comportamiento y eventos que ocurren en los dispositivos de infraestructura. A continuación, se detalla el funcionamiento técnico, ventajas y roles específicos de tres de los principales protocolos utilizados en este proyecto:

a. SNMP (Simple Network Management Protocol)

Es un protocolo ampliamente adoptado para la gestión y supervisión de dispositivos en redes IP. Se basa en una arquitectura cliente-servidor, donde un gestor (manager) consulta o recibe datos de múltiples agentes SNMP que se ejecutan en routers, switches, firewalls y servidores. Existen varios métodos de operación:

- SNMP Get: permite al gestor recuperar valores específicos de variables administradas.
- SNMP GetNext/GetBulk: permiten recorrer estructuras jerárquicas como tablas de interfaces.

- SNMP Walk: ejecuta múltiples GetNext de manera secuencial para explorar un conjunto de OIDs.

- SNMP Trap: es un mensaje que el agente envía de forma proactiva al gestor cuando ocurre un evento predefinido, como una caída de interfaz o un sobrecalentamiento.

Para que estas operaciones funcionen, SNMP utiliza dos conceptos clave: MIB (Management Information Base) y OID (Object Identifier). La MIB es una estructura jerárquica estandarizada que describe los objetos que pueden ser consultados en un dispositivo, organizada como una base de datos lógica. Cada objeto dentro de esta estructura se identifica mediante un OID, una secuencia numérica única (por ejemplo: 1.3.6.1.2.1.2.2.1.10 para el contador de entrada de una interfaz). En resumen, la MIB es el catálogo, y los OID son las entradas consultables dentro de ese catálogo [24].

En este proyecto se utilizó principalmente SNMPv2c por su compatibilidad, bajo overhead y facilidad de configuración, permitiendo tanto el monitoreo pasivo (traps) como el activo (walks programados).

b. NetFlow (versiones 1, 5, 9 y IPFIX)

Es un protocolo diseñado por Cisco para recolectar información estadística sobre los flujos de red. Un flujo se define como una secuencia unidireccional de paquetes que comparten atributos como dirección IP origen/destino, puertos, protocolo y tipo de servicio [25]. La evolución del protocolo se resume así:

- Versión 1: muy limitada, sin timestamps ni extensibilidad.

- Versión 5: una de las más usadas, soporta exportación de información básica de flujo con timestamps.

- Versión 9: introduce la posibilidad de usar plantillas dinámicas, soporta IPv6 y MPLS.

IPFIX (Internet Protocol Flow Information Export): estandarizado por IETF, es la evolución directa de NetFlow v9, permitiendo definir estructuras de datos personalizadas y adaptarse a nuevas necesidades [26].

En la práctica, el colector debe recibir no solo los flujos, sino también las plantillas que describen la estructura de esos datos. Por ello, se configuran parámetros como v9/IPFIX Template Refresh y Template Timeout, que determinan cada cuántos flujos o segundos se reenvían las plantillas. En este proyecto se utilizó la versión IPFIX por su flexibilidad para análisis avanzados de seguridad y rendimiento [26].

c. Syslog

Es un protocolo estándar para el envío de mensajes de registro desde dispositivos de red, servidores o aplicaciones a un servidor centralizado [27]. Los mensajes Syslog están categorizados por niveles de severidad, que van desde:

- 0 (Emergency): el sistema es inutilizable
- 1 (Alert): debe actuar inmediatamente
- (Critical): condiciones críticas
- (Error): errores de ejecución
- (Warning): advertencias
- (Notice): condiciones normales pero importantes
- (Informational): información general
- (Debug): información detallada para diagnóstico

Además, los mensajes se etiquetan por facilidades, que representan el origen (por ejemplo: kernel, authentication, cron, daemon, user-level, entre otros). Esta clasificación permite filtrar y priorizar los mensajes de forma efectiva. En este proyecto se configuraron los dispositivos para enviar mensajes informativos, advertencias y errores hacia un servidor central, donde se filtran, normalizan y visualizan. Syslog es clave para detectar eventos como cambios de configuración, intentos de acceso no autorizados o fallos en servicios críticos [28].

Tabla 5. Tabla comparativa de protocolos de monitoreo de red

Característica	SNMP	NetFlow/IPFIX	Syslog
Tipo de monitoreo	Métricas periódicas (estado, rendimiento)	Información de flujos (tráfico entre hosts)	Mensajes de eventos (logs del sistema)
Modo de operación	Activo (Get, Walk) / Pasivo (Trap)	Exportación periódica desde dispositivos	Emisión en tiempo real ante eventos
Nivel OSI	Capa de aplicación	Capa de red/transporte (3/4)	Capa de aplicación
Estandarización	IETF	Cisco (original) / IETF (IPFIX)	IETF
Formato	MIB/OID estructurado	Plantillas (versión 9/IPFIX)	Texto estructurado con niveles y facilidades
Configuración	Media (requiere comunidad y OIDs)	Media-Alta (plantillas, puertos, dirección colector)	Baja (definición de destino y nivel de severidad)
Utilidad destacada	Rendimiento, interfaces, CPU, temperatura	Análisis de tráfico, detección de patrones	Detección de cambios, errores, accesos
Ventaja principal	Ligero y estandarizado	Visibilidad de tráfico detallada	Fácil integración y contexto de eventos humanos
Desventaja principal	Limitado para análisis de comportamiento	Requiere almacenamiento y procesamiento especializado	Volumen alto de eventos y duplicidad potencial

Como se muestra en la Tabla 5. La combinación de estos tres protocolos ofrece una visión complementaria y robusta del estado de la red. SNMP entrega métricas en tiempo real, NetFlow proporciona análisis de comportamiento del tráfico, y Syslog captura eventos del sistema, permitiendo una supervisión integral tanto para rendimiento como para seguridad [29].

1.3.6 Preprocesamiento y tratamiento de datos en entornos distribuidos

En sistemas de monitoreo y detección de anomalías en redes de datos, la calidad de los resultados depende directamente del tratamiento previo que reciben los datos. Este preprocesamiento se vuelve crítico cuando se trabaja con múltiples fuentes, estructuras heterogéneas y gran volumen de información. Por ello, se implementó un pipeline de procesamiento que inicia desde la recolección hasta la estandarización y limpieza, antes de ser utilizado por algoritmos de análisis o inteligencia artificial [30].

Uno de los componentes centrales de este pipeline es Logstash, una herramienta de código abierto desarrollada por Elastic que forma parte del stack ELK (Elasticsearch, Logstash, Kibana). Logstash está diseñado para actuar como un data processing pipeline, permitiendo la ingesta, transformación y redirección de datos en tiempo real. Su arquitectura se basa en tres etapas principales: inputs, filters y outputs [30].

La fase de input permite recibir datos desde múltiples fuentes, como puertos UDP/TCP, archivos de log, APIs, colas de mensajes, o directamente desde protocolos como SNMP, NetFlow o Syslog [30].

La fase de filter aplica transformaciones a los datos, como análisis de texto mediante patrones (grok), modificación de campos (mutate), separación de estructuras (split), evaluaciones condicionales, enriquecimiento con datos externos o incluso scripts personalizados con ruby [30].

La fase de output define el destino de los datos procesados, que puede ser una base de datos (Elasticsearch, MongoDB), un archivo plano, un servicio de almacenamiento o incluso otra herramienta de visualización o procesamiento [30].

Entre las ventajas clave de Logstash se encuentran:

- Su naturaleza modular y extensible mediante plugins.
- La capacidad de transformar datos semi o no estructurados en estructuras JSON homogéneas.
- Su integración nativa con Elasticsearch, facilitando el análisis inmediato mediante Kibana.
- Su capacidad de escalar horizontalmente en arquitecturas distribuidas.

Cribl LogStream se presenta como una alternativa más eficiente a herramientas tradicionales como Logstash y Fluentd en el procesamiento de flujos de datos. Según pruebas realizadas, LogStream supera a Logstash en un factor de hasta 8.75% en tareas de análisis y reenvío de eventos, y a Fluentd en un 26% en operaciones similares como se muestra en la Figura 2. [31].

Estas mejoras se atribuyen a su arquitectura optimizada para el procesamiento de datos en tiempo real, lo que permite manejar mayores volúmenes con menor consumo de recursos [31].

Además de su eficiencia, LogStream ofrece funcionalidades avanzadas como enriquecimiento de datos, enrutamiento flexible y reducción de volumen, lo que facilita la integración con plataformas de análisis como Splunk o Elastic Stack. Esto permite a las organizaciones gestionar grandes cantidades de datos de manera más rentable y con mayor control sobre su infraestructura de observabilidad [31].

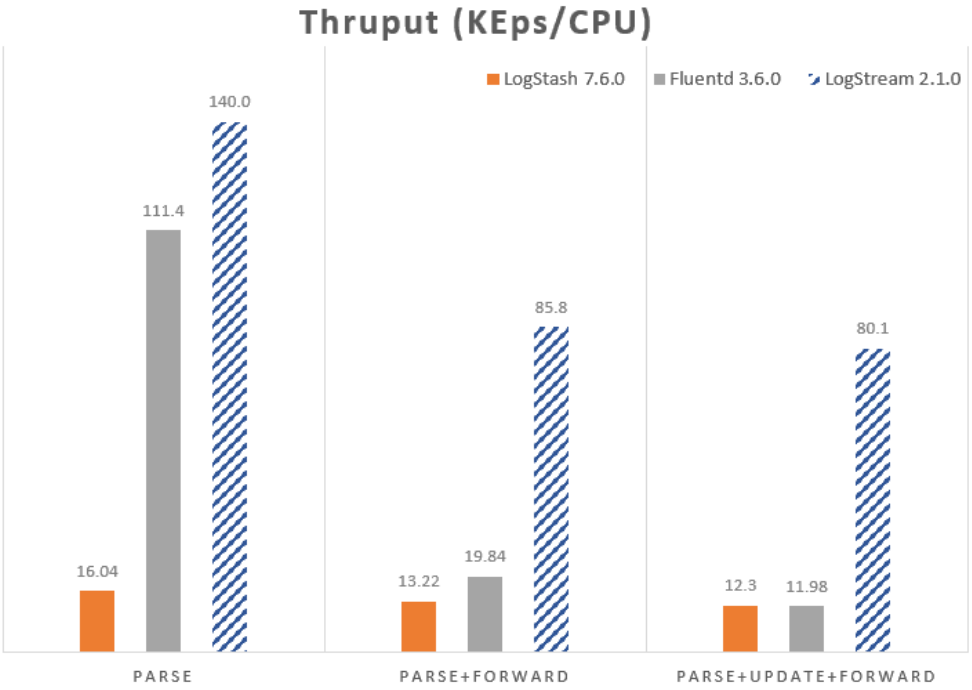


Figura 2. Comparación de LogStash [32]

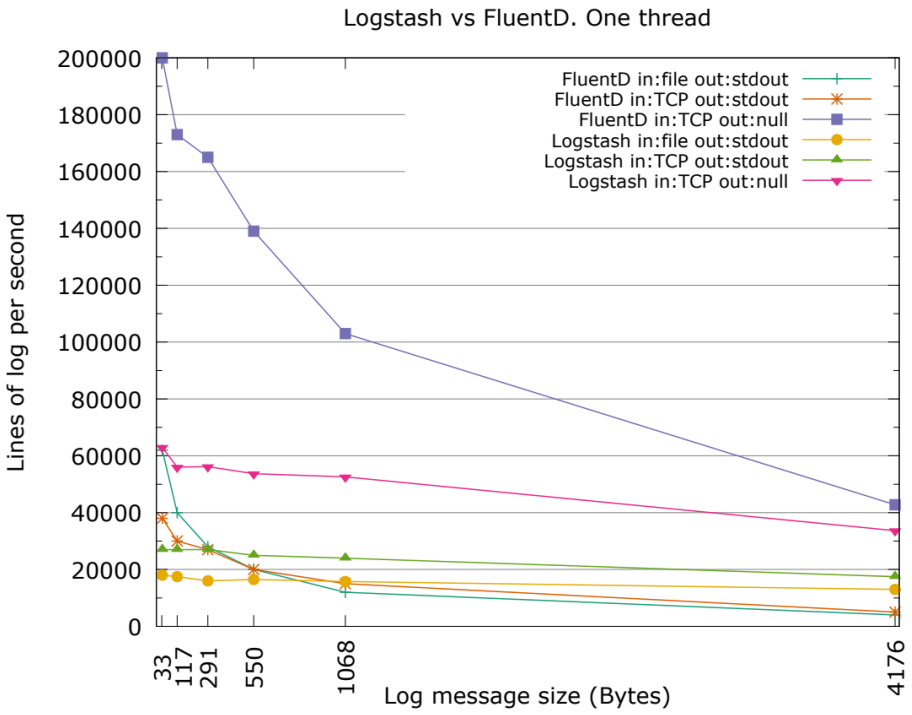


Figura 3. Logstash and FluentD benchmarks [32]

Tabla 6. Comparativa de Herramientas para Procesamiento de Flujos de Datos

Característica	Cribl LogStream	Logstash	Fluentd
Eficiencia de procesamiento	Hasta 8.75 veces más rápido que Logstash y 26% más eficiente que Fluentd	Más lento en tareas de reenvío y análisis	Menor eficiencia comparativa
Consumo de recursos	Optimizado para bajo uso de recursos	Alto consumo en escenarios complejos	Consumo moderado
Procesamiento en tiempo real	Sí, altamente optimizado	Sí, pero con mayor latencia	Sí
Reducción de volumen	Soportado de forma nativa	No es su enfoque principal	Limitado
Enriquecimiento de datos	Soportado de forma integrada	Parcial mediante filtros	Parcial
Enrutamiento flexible	Avanzado	Limitado a configuraciones predefinidas	Limitado a rutas estáticas
Integración con Splunk o Elastic Stack	Integración directa	Compatible con Elastic Stack	Compatible con ambos, con mayor configuración
Arquitectura	Modular, optimizada para volúmenes grandes y observabilidad	Modular, basada en pipelines	Basada en plugins
Enfoque principal	Observabilidad, reducción de datos, control de flujo	Recolección y procesamiento de logs	Agregación y reenvío de logs
Licenciamiento	Comercial, con versiones gratuitas limitadas	Código abierto (parte de Elastic)	Código abierto (proyecto CNCF)

1.3.7 Almacenamiento de datos en sistemas distribuidos

El almacenamiento eficiente es un componente fundamental en sistemas de monitoreo y análisis de redes. En este proyecto se implementaron dos tipos de bases de datos complementarias: una orientada a documentos (Elasticsearch) para el almacenamiento de datos crudos y procesados, y otra relacional embebida (SQLite) para registrar los resultados de predicciones de inteligencia artificial [33].

Elasticsearch es un motor de búsqueda y análisis basado en Apache Lucene, optimizado para almacenar grandes volúmenes de datos en formato JSON. Opera como una base de datos NoSQL distribuida, ideal para datos semiestructurados provenientes de logs, métricas de red, eventos de flujo o registros SNMP. Permite realizar búsquedas complejas con baja latencia mediante su lenguaje de consulta DSL (Domain Specific Language), y soporta funciones de agregación para análisis estadístico [33].

Características clave de Elasticsearch:

- Índices distribuidos, escalables y replicables.
- Soporte nativo para timestamp y series temporales.
- Integración directa con Logstash para ingesta automatizada.
- Rendimiento optimizado para consultas full-text y filtros por campo.

SQLite es una base de datos relacional embebida, ligera y sin necesidad de servidor. Su principal uso en este proyecto fue almacenar los resultados inferidos por los modelos de IA, como las anomalías detectadas, su puntuación, umbral, marca temporal y detalles del flujo o dispositivo afectado. Esto permitió consultas rápidas desde interfaces ligeras como Streamlit y facilitó la persistencia de resultados entre ejecuciones [33].

424 systems in ranking, June 2025

Rank			DBMS	Database Model	Score		
Jun 2025	May 2025	Jun 2024			Jun 2025	May 2025	Jun 2024
1.	1.	1.	Oracle	Relational, Multi-model	1230.38	+3.82	-13.70
2.	2.	2.	MySQL	Relational, Multi-model	953.57	-11.41	-107.77
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	776.75	+1.86	-44.81
4.	4.	4.	PostgreSQL	Relational, Multi-model	680.65	+6.34	+44.41
5.	5.	5.	MongoDB	Document, Multi-model	402.85	+0.33	-18.23
6.	6.	↑ 8.	Snowflake	Relational	174.49	+2.48	+44.13
7.	7.	↓ 6.	Redis	Key-value, Multi-model	151.72	-0.47	-4.22
8.	8.	↑ 9.	IBM Db2	Relational, Multi-model	125.13	-1.27	-0.77
9.	9.	↓ 7.	Elasticsearch	Multi-model	121.28	-2.53	-11.55
10.	10.	10.	SQLite	Relational	117.03	-0.74	+5.63
11.	11.	↑ 12.	Apache Cassandra	Wide column, Multi-model	108.27	+0.22	+9.44
12.	12.	↑ 15.	Databricks	Multi-model	104.67	+2.02	+23.59
13.	↑ 14.	13.	MariaDB	Relational, Multi-model	94.54	+0.93	+3.50
14.	↓ 13.	↓ 11.	Microsoft Access	Relational	88.28	-7.63	-12.88
15.	15.	↑ 17.	Amazon DynamoDB	Multi-model	83.34	+4.04	+8.90
16.	16.	↑ 18.	Apache Hive	Relational	76.68	+0.91	+16.92
17.	17.	↓ 16.	Microsoft Azure SQL Database	Relational, Multi-model	75.31	-0.14	-1.47
18.	18.	↓ 14.	Splunk	Search engine	69.61	-1.78	-19.49
19.	19.	19.	Google BigQuery	Relational	64.54	+1.66	+6.44
20.	20.	↑ 21.	Neo4j	Graph	51.33	+3.42	+6.44
21.	21.	↓ 20.	FileMaker	Relational	34.02	-1.53	-13.89
22.	↑ 23.	↑ 24.	Apache Solr	Search engine, Multi-model	33.47	+0.67	-7.56
23.	↓ 22.	↓ 22.	Teradata	Relational, Multi-model	32.59	-1.28	-12.28
24.	24.	↓ 23.	SAP HANA	Relational, Multi-model	31.73	-0.27	-12.54
25.	25.	25.	SAP Adaptive Server	Relational, Multi-model	27.91	-0.75	-7.18
26.	↑ 29.	↑ 33.	Apache Spark (SQL)	Relational	22.73	+1.11	+4.69
27.	27.	↓ 26.	Apache HBase	Wide column	22.69	+0.06	-5.28
28.	↓ 26.	↓ 27.	Microsoft Azure Cosmos DB	Multi-model	22.43	-0.51	-5.28
29.	↓ 28.	↓ 28.	InfluxDB	Time Series, Multi-model	21.74	-0.04	-2.65
30.	↑ 31.	↑ 38.	ClickHouse	Relational, Multi-model	18.60	-0.17	+3.05

Figura 4. Ranking de motores de base de datos [34]

Según el ranking de motores de base de datos mostrado en la Figura 4, destacan particularmente Elasticsearch y SQLite por su relevancia en distintos escenarios técnicos. Elasticsearch, ubicado en la posición 9, sobresale como motor orientado a búsqueda y análisis en tiempo real sobre grandes volúmenes de datos, especialmente útil en arquitecturas de observabilidad y monitoreo. A pesar de una leve caída en su puntaje (-2.53), mantiene una posición sólida dentro del top 10, lo que refleja su adopción sostenida en entornos críticos. Por otro lado, SQLite, en la posición 10, sigue siendo uno de los motores relacionales más ligeros y ampliamente usados, ideal para aplicaciones embebidas y entornos de baja complejidad. Su diseño sin servidor y su portabilidad lo hacen una opción técnica excepcional. Ambas soluciones representan alternativas destacadas según sus casos de uso y niveles de escalabilidad.

1.3.8 Visualización de datos y resultados

La visualización clara e interactiva de datos es esencial para interpretar información compleja de manera intuitiva. En este proyecto se emplearon herramientas especializadas como Kibana y Streamlit, seleccionadas por su adecuación al tipo de análisis requerido [31].

Kibana es una interfaz web que forma parte del stack ELK, utilizada para explorar visualmente los datos almacenados en Elasticsearch. Proporciona dashboards personalizables con gráficos de líneas, barras, pastel, mapas y tablas dinámicas. En este proyecto, Kibana se empleó para representar en tiempo real:

- Volumen de tráfico por IP, puerto o protocolo.
- Métricas SNMP de CPU, temperatura y uso de interfaz.
- Eventos Syslog filtrados por severidad.
- Análisis temporal de tendencias.
- Entre sus ventajas se encuentran:
- Interactividad con filtros y rangos de tiempo.
- Integración directa con índices de Elasticsearch.
- Visualización en tiempo real de flujos y eventos históricos.

Streamlit, por otro lado, es una herramienta de visualización orientada al desarrollo rápido de aplicaciones de ciencia de datos. Permite construir dashboards dinámicos con pocas líneas de código en Python, siendo ideal para prototipado y despliegue de interfaces de modelos predictivos [31].

En este proyecto, Streamlit fue utilizado para visualizar los resultados del sistema de detección de anomalías, incluyendo:

- Gráficas de error de reconstrucción.

- Líneas de umbral y puntuaciones anómalas.
- Detalle de eventos detectados por flujo o dispositivo.
- Ventajas de Streamlit:
- Desarrollo rápido y orientado a datos.
- Interfaz ligera accesible vía navegador.
- Integración sencilla con pandas, matplotlib y SQLite.

El uso combinado de estas tecnologías permitió crear una arquitectura robusta, con almacenamiento escalable, persistencia ligera y visualización eficaz, cubriendo las necesidades tanto operativas como analíticas del sistema de monitoreo y análisis en redes de datos [31].

1.3.9 Aplicación de modelos de deep learning para detección de anomalías en redes de datos.

La detección de anomalías en redes de datos representa un desafío complejo debido a la naturaleza multivariada, dinámica y no lineal del tráfico. Los enfoques tradicionales basados en reglas estáticas o estadísticas lineales tienen limitaciones importantes ante eventos imprevistos o variaciones sutiles en los patrones de comportamiento. En este contexto, las técnicas de deep learning han demostrado una capacidad superior para modelar secuencias temporales, capturar relaciones complejas y adaptarse a cambios graduales o abruptos en la red [20].

Los modelos más utilizados para este tipo de tarea son las redes neuronales recurrentes, que procesan datos secuenciales conservando información temporal. Cada una de estas arquitecturas cuenta con hiperparámetros, que son configuraciones del modelo definidas antes del entrenamiento y que afectan directamente su capacidad de aprendizaje, generalización y rendimiento [14].

A diferencia de los parámetros internos (pesos y sesgos), los hiperparámetros no se aprenden automáticamente, sino que deben ser seleccionados mediante experimentación o técnicas de optimización. Entre los más comunes se incluyen el número de neuronas por capa (units), las funciones de activación (activation, recurrent_activation), los valores de dropout para evitar sobreajuste, y opciones de entrenamiento como el optimizador (optimizer), la función de pérdida (loss) y la longitud de las secuencias (sequence_length). A continuación, se describen con mayor profundidad las tres arquitecturas principales empleadas en la literatura técnica: RNN, LSTM y GRU [14].

a. RNN (Recurrent Neural Network)

Las RNN son redes que utilizan bucles internos para mantener una memoria del estado anterior, lo que las hace adecuadas para el modelado de series temporales. Cada salida depende no solo de la entrada actual, sino también de los estados anteriores. Sin embargo, su principal limitación es el problema del vanishing gradient, que afecta el aprendizaje de dependencias a largo plazo, haciendo que la red se enfoque solo en patrones recientes [14].

b. LSTM (Long Short-Term Memory)

Esta arquitectura introduce mecanismos de compuertas (gate mechanisms) —input gate, forget gate y output gate— que regulan el flujo de información a través de una celda de memoria. Las LSTM pueden conservar información relevante durante intervalos prolongados y descartar ruido, lo que las convierte en una opción sólida para la detección de anomalías que no se presentan de forma inmediata [14].

Las redes LSTM (Long Short-Term Memory) están diseñadas para resolver una de las limitaciones principales de las RNN tradicionales, la dificultad para recordar información importante a lo largo de secuencias largas. Para lograr esto, las LSTM introducen una celda de memoria que actúa como un "canal" por el que fluye la información, y tres mecanismos de compuertas que controlan ese flujo:

- Input gate (compuerta de entrada): Decide qué información nueva se almacena en la celda de memoria.

- Forget gate (compuerta de olvido): Determina qué información antigua debe eliminarse de la celda.

- Output gate (compuerta de salida): Controla qué información de la celda se usa para generar la salida actual.

Gracias a estas compuertas, las LSTM pueden retener información relevante durante períodos largos y eliminar datos irrelevantes o "ruido". Esto es especialmente útil en tareas como la detección de anomalías en redes de datos, donde un evento anómalo puede depender de una secuencia de comportamientos pasados que no se manifiestan de forma inmediata. Por ejemplo, una serie de pequeños aumentos de tráfico podría ser normal por sí sola, pero si se mantienen durante cierto tiempo, podría representar una amenaza. Las LSTM son capaces de detectar ese tipo de patrones dependientes del contexto histórico, lo que las convierte en una herramienta muy poderosa en escenarios de series temporales complejas. [14].

Tabla 7. Hiperparámetros de RNN

Hiperparámetro	**Descripción**	**Valores comunes o recomendados**
Units	Número de neuronas en la capa recurrente.	32, 64, 128
Activation	Función de activación para la salida de la celda.	tanh, relu
Recurrent_activation	Activación para la conexión recurrente (estado anterior).	sigmoid, tanh
Dropout	Proporción de desconexión de neuronas para evitar overfitting.	0.0 – 0.5
Recurrent_dropout	Dropout aplicado a las conexiones recurrentes.	0.0 – 0.5
Return_sequences	Si True, devuelve una secuencia completa; si False, solo la última salida.	True (si se usa en pilas), False (si es la última capa)
Optimizer	Algoritmo de optimización durante el entrenamiento.	adam, rmsprop, sgd
Loss	Función de pérdida usada en la reconstrucción del error.	mse, mae
Sequence_length	Número de pasos de tiempo considerados en cada entrada.	10 – 100 (según ventana temporal)

Tabla 8. Hiperparámetros de LSTM

Hiperparámetro	Descripción	Valores comunes
Units	Número de neuronas (celdas de memoria).	32, 64, 128, 256
Activation	Función de activación de las salidas.	tanh
Recurrent_activation	Activación de las compuertas (input/forget/output).	sigmoid
Dropout	Aplica dropout a las entradas.	0.2 – 0.5
Recurrent_dropout	Dropout en las conexiones recurrentes (estado previo).	0.1 – 0.5
Return_sequences	Devuelve secuencia completa o solo la última salida.	True / False
Use_bias	Si se usa bias (desplazamiento) en los cálculos internos.	True (por defecto)
Kernel_initializer	Inicialización de los pesos.	glorot_uniform, he_normal
Optimizer, loss, sequence_length	Igual que en RNN.	Igual

c. GRU (Gated Recurrent Unit)

Variante simplificada de LSTM que fusiona las compuertas de entrada y olvido en una sola (update gate) y emplea una compuerta de reinicio (reset gate). GRU requiere menos parámetros, lo que reduce el tiempo de entrenamiento y el consumo computacional, sin sacrificar precisión en muchas tareas. Se ha observado que en datasets pequeños o medianos, GRU puede superar a LSTM en eficiencia manteniendo una capacidad de modelado adecuada [29].

Tabla 9. Hiperparámetros de GRU

Hiperparámetro	Descripción	Valores comunes
Units	Número de neuronas.	32, 64, 128
Activation	Activación de la salida.	tanh, relu
Recurrent_activation	Activación de las compuertas (update/reset).	sigmoid
Dropout, recurrent_dropout	Igual que en LSTM.	0.0 – 0.5
Return_sequences, use_bias, optimizer, loss, sequence_length	Igual que en LSTM.	Igual

Tabla 10. Comparación entre modelos RNN, LSTM y GRU

Característica	RNN	LSTM	GRU
Estructura de memoria	Estado oculto simple	Celdas con memoria + compuertas	Celdas con compuertas simplificadas
Prevención de vanishing gradient	No	Sí (mediante compuertas)	Sí (compuerta unificada)
Número de parámetros	Bajo	Alto	Medio
Capacidad para dependencia larga	Limitada	Alta	Media/Alta
Tiempo de entrenamiento	Rápido	Lento	Rápido
Precisión en series complejas	Baja/Media	Alta	Alta
Recomendado para	Secuencias simples y cortas	Secuencias complejas y largas	Problemas con recursos limitados y series medianas

Tabla 11. Comparativa de Hiperparámetros – RNN vs LSTM vs GRU

Hiperparámetro	RNN	LSTM	GRU
Units	32, 64, 128	32, 64, 128, 256	32, 64, 128
Activation	tanh, relu	tanh	tanh, relu
Recurrent_activation	sigmoid, tanh	sigmoid (para compuertas)	sigmoid (update/reset gates)
Dropout	0.0 – 0.5	0.2 – 0.5	0.0 – 0.5
Recurrent_dropout	0.0 – 0.5	0.1 – 0.5	0.0 – 0.5
Return_sequences	True / False	True / False	True / False
Use_bias	True (opcional)	True	True
Kernel_initializer	glorot_uniform, he_normal (opcional)	glorot_uniform, he_normal	glorot_uniform, he_normal
Optimizer	adam, rmsprop, sgd	adam, rmsprop, sgd	adam, rmsprop, sgd
Loss	mse, mae	mse, mae	mse, mae
Sequence_length	10 – 100 (depende del contexto temporal de la serie)	10 – 100	10 – 100

1.3.10 Técnicas y consideraciones metodológicas clave en el entrenamiento de modelos de detección de anomalías.

a. Preprocesamiento de datos

Incluye la imputación de valores faltantes, normalización de variables numéricas (e.g., MinMaxScaler), codificación de variables categóricas (e.g., OneHot, LabelEncoder), y selección de características relevantes eliminando aquellas con varianza nula o alta colinealidad [29].

b. Construcción de secuencias

Las series temporales se organizan en ventanas deslizantes (sliding windows) de longitud fija, creando secuencias que representan estados recientes del sistema. Esto permite entrenar al modelo para predecir la siguiente observación o reconstruir la secuencia [14].

c. Método de entrenamiento

- Supervisado: Cuando se dispone de etiquetas que indican si una instancia es anómala o no.
- No supervisado: Común en entornos reales donde las anomalías no están etiquetadas. Se entrena al modelo con datos normales y se evalúa la desviación del comportamiento esperado (reconstrucción o predicción).
- Función de pérdida: Se utiliza principalmente el Mean Squared Error (MSE) para medir la diferencia entre la salida esperada y la reconstruida o predicha por la red. Una alta diferencia puede indicar un comportamiento anómalo.

d. Validación y detección:

- Se calcula el error de reconstrucción sobre un conjunto de validación.
- Se define un umbral a partir del percentil 95, 99 o distribución de errores (e.g., desviación estándar).
- Las observaciones cuyo error exceda el umbral se marcan como anomalías.

e. Evaluación del modelo

En escenarios donde no se cuenta con etiquetas supervisadas que identifiquen explícitamente los casos de anomalías, la evaluación del modelo debe realizarse mediante métricas de regresión que reflejen la capacidad del sistema para reconstruir secuencias temporales de datos. En este contexto, se aplican métricas como el Error Cuadrático Medio (MSE), el Error Absoluto Medio (MAE), el Coeficiente de

Determinación (R^2) y la Varianza Explicada, las cuales permiten cuantificar la precisión y generalización del modelo sobre los datos de prueba [35].

Asimismo, se emplea el análisis del error de reconstrucción como criterio heurístico para la detección de comportamientos anómalos. Mediante la aplicación de un umbral estadístico (por ejemplo, el percentil 99 del error), es posible clasificar muestras como normales o anómalas de forma no supervisada, facilitando la implementación de sistemas de alerta temprana sin necesidad de etiquetas previas [35].

Tabla 12. Comparativa de Métricas de Evaluación Basadas en Error

Métrica	**Descripción**	**Rango de valores**	**Interpretación Ideal**	**Ventajas**	**Limitaciones**
MSE (Mean Squared Error)	Promedio del cuadrado de las diferencias entre los valores reales y los predichos.	$[0, \infty)$	Más cercano a 0	Penaliza fuertemente errores grandes.	Sensible a outliers (los errores grandes afectan mucho la métrica).
MAE (Mean Absolute Error)	Promedio del valor absoluto de las diferencias entre los valores reales y los predichos.	$[0, \infty)$	Más cercano a 0	Fácil de interpretar; menos sensible a outliers que MSE.	No penaliza errores grandes tanto como MSE.
R^2 (Coef. de Determinación)	Mide qué proporción de la varianza total de los datos es explicada por el modelo.	$(-\infty, 1]$	Cercano a 1 (máximo)	Muestra qué tan bien se ajusta el modelo a los datos.	Puede ser negativo si el modelo es peor que una media constante.
Varianza Explicada	Evalúa cuánta variabilidad de los datos originales es explicada por las predicciones del modelo.	[0, 1]	Cercano a 1	Similar a R^2, pero centrada solo en la varianza, sin penalización por sesgo.	Puede dar valores altos aunque haya errores sistemáticos.

Desde el punto de vista teórico, las arquitecturas LSTM, GRU y RNN permiten modelar la dinámica temporal de sistemas complejos como las redes de datos. Su capacidad para aprender representaciones latentes y adaptarse a patrones no lineales las convierte en una herramienta poderosa dentro del campo de la seguridad informática, el monitoreo de rendimiento y la detección proactiva de fallos, incluso en entornos donde no se dispone de información previamente categorizada [36].

Tabla 13. Comparativa para Establecimiento de Umbrales de Detección de Anomalías

Método de Umbral	**Descripción**	**Tipo de Umbral**	**Ventajas**	**Desventajas**
Percentil 99	Se clasifica como anómalo cualquier valor cuyo error de reconstrucción supere el percentil 99 de la distribución del error.	Estático (estadístico)	Simple, robusto, no requiere etiquetas.	Puede ser insensible si hay muchas anomalías en los datos (se "normalizan").
Media + n·Desviación Estándar	Se considera anómalo si el error > media + n·σ (por ejemplo, n = 3).	Estático (gaussiano)	Intuitivo si los errores siguen una distribución normal.	Asume distribución normal del error, lo cual no siempre es cierto.
IQR (Rango Intercuartílico)	Umbral = Q3 + 1.5·IQR; se basa en la dispersión entre el Q1 y Q3.	Estático (no paramétrico)	No asume distribución normal.	Puede ser menos sensible en distribuciones sesgadas o multimodales.
Métodos de clustering (e.g., DBSCAN, Isolation Forest)	Se considera anómalo si una muestra está lejos del clúster o aislada del bosque de árboles.	Dinámico (basado en densidad)	Detecta anomalías contextuales y agrupadas.	Más complejo computacionalmente; puede requerir ajuste fino de hiperparámetros.
Autoencoder Reconstruction Threshold Learning	Entrenamiento de un umbral basado en la curva ROC usando datos no etiquetados asumidos normales.	Híbrido (autoajustable)	Permite calibración más precisa si se puede asumir un baseline limpio.	Supone que los datos de entrenamiento están libres de anomalías.
Ventana deslizante adaptativa	El umbral se ajusta continuamente en función del error reciente (por ejemplo, usando media móvil).	Adaptativo en tiempo real	Se ajusta dinámicamente a cambios de comportamiento en la red.	Riesgo de normalizar gradualmente anomalías persistentes (concept drift).

1.3.11 Repositorios de Control de Versiones

En el desarrollo de software moderno, los sistemas de control de versiones juegan un papel fundamental al permitir la gestión eficiente de cambios en el código fuente, la colaboración entre múltiples desarrolladores, el seguimiento de versiones, y la automatización del despliegue. El sistema de control de versiones distribuido más utilizado en la actualidad es Git, que permite mantener un historial completo de los cambios de manera local y remota [35].

Existen diversas plataformas que integran Git con funcionalidades adicionales como gestión de proyectos, integración continua y entrega continua (CI/CD), monitoreo, seguridad y control de accesos. Las más utilizadas en entornos profesionales y académicos son GitLab, GitHub y Bitbucket [35].

Tabla 14. Comparativa de Repositorios para Control de Versiones

Característica	**GitLab**	**GitHub**	**Bitbucket**
Modelo de alojamiento	Autoalojado (Self-hosted) o en la nube	Solo nube (GitHub.com)	Nube (Bitbucket.org) y Data Center (self-hosted)
Integración CI/CD	Nativa (GitLab CI/CD totalmente integrado)	GitHub Actions (integrado)	Pipelines (requiere configuración)
Control de acceso granular	Avanzado, con control por grupos y roles	Limitado (requiere configuración externa)	Integración con Jira para permisos
Gestión de proyectos	Kanban, milestones, issues, epics nativos	Issues y proyectos (limitado sin apps)	Requiere Atlassian (Jira, Trello)
Código abierto	edición comunitaria disponible	código cerrado	código cerrado
Seguridad DevSecOps	Análisis estático, dinámico y de dependencias integrado	Requiere apps externas	Integraciones vía Atlassian
Popularidad	Alta en entornos empresariales y DevOps	Mayor en la comunidad open-source	Alta en empresas que usan Atlassian
Costo	Gratuito (comunitario) y pagos escalables	Gratuito para proyectos públicos y planes pagos	Gratuito con límites, pagos por usuario

1.4 Objetivos

1.4.1 Objetivo general

Implementar un sistema de detección de anomalías de rendimiento y seguridad en redes de datos con inteligencia artificial.

1.4.2 Objetivos específicos

- Determinar las técnicas existentes en el campo de la detección de anomalías de rendimiento y seguridad en redes de datos utilizando inteligencia artificial.
- Diseñar un sistema de detección de anomalías de rendimiento y seguridad en redes de datos basado en inteligencia artificial.
- Integrar el sistema de detección de anomalías de rendimiento y seguridad en redes de datos basado en inteligencia artifical.

CAPÍTULO II. METODOLOGÍA

2.1 Materiales

Para el diseño y validación del sistema se emplearon tanto recursos físicos como virtuales, abarcando plataformas de desarrollo, bases de datos, sistemas de virtualización y bibliotecas científicas. A continuación, se detallan los materiales utilizados:

- Recursos computacionales
- Servidor virtualizado sobre Proxmox VE con arquitectura x86_64.
- Asignación mínima: 8 vCPU, 32 GB RAM, 500 GB almacenamiento SSD.
- Sistema operativo base: Ubuntu Server 22.04 LTS.
- Conectividad permanente a Internet para transferencia de datos y sincronización de componentes distribuidos.

2.1.1 Dispositivos de red

Se utilizaron dispositivos de red empresariales con soporte para protocolos estándar de monitoreo y exportación de métricas. A continuación, se detallan los modelos específicos:

- MikroTik CCR2116-12G-4S+: Router carrier-grade con CPU ARM de 16 núcleos, 12 puertos Gigabit Ethernet y 4 puertos SFP+ de 10 Gbps. Admite exportación de estadísticas de tráfico a través de SNMP v2/v3, NetFlow/IPFIX y generación de Syslog. Implementado como nodo principal de capa core.
- MikroTik CRS328-24P-4S+: Switch de capa 3 administrable con 24 puertos Gigabit PoE y 4 puertos SFP+ de 10 Gbps. Compatible con monitoreo SNMP y generación de eventos mediante Syslog. Utilizado en la capa de distribución para visibilidad operativa.

- Fortinet FortiGate FGT40F: Firewall UTM con funcionalidades de inspección profunda de paquetes, control de aplicaciones y generación de logs mediante Syslog, NetFlow/IPFIX y SNMP. Opera como punto perimetral de seguridad.

- PC Fisica/Virtual: Estación de trabajo configurada como gateway de red, responsable de enrutar el tráfico de monitoreo (SNMP, NetFlow, Syslog) desde la red local hacia la red virtual definida por software (ZeroTier). Actúa como punto de interconexión segura con el servidor central, permitiendo la transferencia continua de datos desde dispositivos de red distribuidos.

2.1.2 Software utilizado

Tabla 5. Software utilizado y propósito funcional

Software / Herramienta	**Tipo / Licencia**	**Características técnicas**	**Descripción de uso principal**
Ubuntu Server 22.04	Sistema Operativo / Libre	Base estable, soporte LTS, gestión avanzada de red	SO base para servidor de procesamiento y Kubernetes
Proxmox VE	Hipervisor / Libre	Soporte para KVM/QEMU, almacenamiento ZFS	Virtualización de entorno de procesamiento central
Docker	Contenedores / Libre	Virtualización ligera, aislamiento de servicios	Ejecución de servicios como Logstash, Jupyter, Kibana
Kubernetes (kubeadm)	Orquestación / Libre	Gestión de pods, namespaces, servicios distribuidos	Coordinación de microservicios y escalabilidad
ZeroTier	SD-WAN / Libre	Redes definidas por software, cifrado end-to-end	Interconexión de redes remotas de forma segura
Logstash	Procesamiento / Libre	Filtros grok, mutate, ruby, compatibilidad con SNMP	Filtrado y estructuración de datos entrantes
Elasticsearch	Base de datos / Libre	Almacenamiento distribuido, búsqueda rápida	Indexación y consulta eficiente de métricas y logs
Kibana	Visualización / Libre	Dashboards interactivos, gráficos en tiempo real	Visualización de tráfico, métricas y eventos
Streamlit	Frontend IA / Libre	Desarrollo de dashboards en Python con bajo código	Interfaz para mostrar resultados de IA
Python 3.10	Lenguaje / Libre	Sintaxis simple, ecosistema científico extenso	Desarrollo de scripts de análisis y modelos IA
SQLite3	Base de datos / Libre	Motor embebido, archivo único, ACID compliance	Almacenamiento local de resultados de inferencia
Git + GitLab	Control de versiones / Libre	Gestión de código, CI/CD, trazabilidad	Control del ciclo de vida del proyecto y colaboración
JupyterLab	IDE / Libre	Entorno interactivo para análisis científico	Desarrollo de notebooks y pruebas de modelos

2.1.3 Librerías de Python utilizadas

- NumPy: manejo de arreglos multidimensionales y operaciones vectorizadas eficientes.
- Pandas: estructuras de datos tipo DataFrame para manipulación de datos tabulares.
- Matplotlib / Seaborn / Plotly: generación de gráficos estáticos e interactivos para análisis exploratorio y resultados.
- Scikit-learn: herramientas de preprocesamiento, codificación, imputación, y métricas de validación.
- TensorFlow / Keras: construcción, entrenamiento y evaluación de modelos de deep learning (RNN, LSTM, GRU).
- SQLAlchemy / sqlite3: manipulación de base de datos embebida para resultados de inferencia.
- Streamlit: desarrollo rápido de interfaces web interactivas para visualización en tiempo real.

Estas herramientas fueron seleccionadas por su robustez, soporte comunitario, madurez y adecuación al desarrollo de sistemas distribuidos de análisis de datos, garantizando reproducibilidad, escalabilidad y eficiencia metodológica.

2.2 Métodos

2.2.1 Modalidad de la investigación

La presente investigación se sustenta en tres modalidades metodológicas complementarias que permiten abordar integralmente el problema planteado.

En primer lugar, la investigación de campo resulta fundamental, ya que posibilita la recolección directa de datos desde redes operativas reales. Esta modalidad permite observar y analizar, en tiempo real, eventos relacionados con el rendimiento y la seguridad de la infraestructura, garantizando así la pertinencia y aplicabilidad de los resultados obtenidos frente a escenarios reales.

Por otro lado, la investigación bibliográfica-documental cumple un rol esencial en la construcción del marco teórico y metodológico del estudio. A través del análisis riguroso de artículos científicos, libros especializados y estudios previos, se establecen las bases conceptuales sobre las cuales se fundamenta el desarrollo de técnicas de detección de anomalías en redes de datos y la aplicación de inteligencia artificial en este campo.

Finalmente, la investigación aplicada permite trasladar los conocimientos teóricos a un entorno práctico mediante el diseño, implementación y validación de un sistema funcional. Esta modalidad busca resolver problemáticas concretas como la optimización del rendimiento de red y la mejora de la seguridad operativa, integrando herramientas tecnológicas avanzadas basadas en inteligencia artificial.

2.2.2 Recolección de información

La recolección de datos es una fase crítica dentro de la metodología, ya que proporciona la base sobre la cual se entrena, evalúa y valida el sistema de detección de anomalías. A continuación, se describen los elementos clave de este proceso, en coherencia con la arquitectura y materiales previamente definidos como se muestra en la tabla 15.

Tabla 15. Recolección de Información.

Aspecto	Descripción
¿Para qué?	La recolección busca identificar patrones de tráfico anómalo, problemas de rendimiento y brechas de seguridad para alimentar y validar el sistema inteligente de análisis.
¿Qué se recolecta?	La información es obtenida a partir de objetos técnicos: flujos NetFlow/IPFIX, métricas SNMP, eventos Syslog, registros de Elasticsearch, y resultados de procesamiento del pipeline Logstash. No se recolectan datos personales.
¿Cómo? (Técnicas)	Se aplica captura directa mediante protocolos habilitados en equipos MikroTik CCR2116-12G-4S+ y CRS328-24P-4S+, así como en el firewall Fortinet FGT40F. Se realiza análisis continuo en tiempo real con procesamiento mediante Logstash y filtrado según OIDs, puertos y niveles de severidad.
¿Con qué? (Instrumentos)	Se utilizan herramientas como Logstash, ZeroTier, Elasticsearch, Wireshark, y scripts desarrollados en Python para recopilar y procesar la información de red. Las PC gateway permiten el reenvío del tráfico hacia el clúster Kubernetes.
¿Cuándo?	El proceso de recolección se ejecuta de forma continua, con un intervalo de análisis y validación cada cinco minutos, permitiendo capturar comportamientos en distintos momentos del día.
¿Dónde?	La recolección se lleva a cabo en una red simulada y controlada desplegada mediante Proxmox, complementada con tráfico real de redes empresariales medianas interconectadas por ZeroTier.
¿Con qué frecuencia?	Las capturas son continuas. Los datos son almacenados en Elasticsearch y los resultados de inferencia se guardan en SQLite. Se realiza evaluación semanal para recalibración de umbrales y ajuste de modelos.

2.2.3 Procesamiento y análisis de datos

El procesamiento y análisis de los datos recopilados es una etapa esencial para transformar grandes volúmenes de información técnica en insumos válidos para la detección de anomalías. Este proceso se abordó en tres niveles: tratamiento inicial de datos crudos, preparación para análisis mediante modelos de aprendizaje automático, y evaluación de resultados.

a. Preprocesamiento y transformación de datos

Los datos recopilados por Logstash desde protocolos SNMP, NetFlow/IPFIX y Syslog se estructuran y normalizan antes de ser almacenados en Elasticsearch. Las operaciones básicas de transformación incluyen:

- Filtrado de registros incompletos o con errores de formato.
- Conversión de fechas y horas a formato UTC estandarizado (@timestamp).

- Extracción de campos clave: IPs, puertos, protocolos, OIDs, niveles de severidad, etc.
- Asignación de etiquetas lógicas para clasificación posterior.

Una vez indexados, los datos son extraídos mediante consultas DSL para ser procesados en Python. El tratamiento estadístico incluye:

- Análisis de valores nulos y eliminación de atributos con más del 75% de valores faltantes.
- Imputación de datos faltantes con mediana o promedio (según tipo de variable).
- Eliminación de atributos con varianza nula y alta correlación (>0.95).
- Codificación de variables categóricas con LabelEncoder.
- Normalización de variables numéricas con MinMaxScaler.

b. Construcción de secuencias temporales

Dado que los modelos empleados son redes neuronales orientadas a secuencias (RNN, LSTM, GRU), los datos se estructuran en ventanas deslizantes (sliding windows) de longitud fija t, que capturan el comportamiento reciente de los atributos monitoreados. Cada ventana representa un vector multivariable que será usado como entrada para los modelos.

Esta técnica permite analizar el contexto temporal de cada métrica, modelar tendencias, fluctuaciones y detectar desviaciones respecto a patrones históricos.

c. Entrenamiento de modelos de detección de anomalías

Se entrenaron y compararon tres arquitecturas de deep learning:

- RNN (Recurrent Neural Network): modelo secuencial clásico capaz de procesar series de tiempo. Presenta limitaciones para capturar dependencias a largo plazo debido al problema del desvanecimiento del gradiente.

- LSTM (Long Short-Term Memory): arquitectura que incluye compuertas de entrada, olvido y salida, permitiendo retener información relevante por periodos extensos. Adecuada para detectar patrones complejos en tráfico de red.

- GRU (Gated Recurrent Unit): variante simplificada de LSTM con menos parámetros y menor carga computacional, pero desempeño similar en tareas de detección.

El procedimiento se desarrolla en las siguientes etapas:

- División del dataset en conjuntos de entrenamiento (80%) y validación/test (20%).

- Entrenamiento del modelo con datos normalizados y sin etiquetas

- Ajuste de hiperparámetros: número de capas, unidades, tasa de aprendizaje, número de épocas.

- Evaluación con función de pérdida MSE (Mean Squared Error) para detectar errores de reconstrucción.

d. Umbral de detección de anomalías

Finalizado el entrenamiento, se calcula la distribución de los errores de reconstrucción en el conjunto de validación. A partir de esta distribución se determina un umbral de corte, típicamente ubicado en el percentil 99, para discriminar entre comportamiento normal y anómalo.

Las instancias cuyo error supere dicho umbral son clasificadas como anomalías. Esta lógica es robusta frente a ruido y adaptable a nuevas condiciones, ya que el modelo aprende los patrones habituales de la red y detecta desviaciones significativas.

e. Evaluación del rendimiento del sistema

Se utilizaron diversas métricas para evaluar la capacidad de detección de los modelos:

- Error Cuadrático Medio (MSE): promedio del cuadrado de las diferencias entre los valores reales y los valores predichos por el modelo.

- Error Absoluto Medio (MAE): promedio del valor absoluto de las diferencias entre los valores reales y los valores predichos

- Coeficiente de determinación (R^2): mide la proporción de la varianza total de los datos que el modelo es capaz de explicar.

- Varianza Explicada (EVS): mide la proporción de la varianza total de la salida que es explicada por el modelo

f. Visualización de resultados

La presentación y análisis visual de los resultados es fundamental para validar el comportamiento del sistema y facilitar la toma de decisiones por parte de operadores e ingenieros.

- Kibana fue empleado como herramienta de monitoreo en tiempo real, permitiendo la creación de dashboards interactivos conectados directamente a Elasticsearch. Desde allí se visualizaron métricas de red (bytes, paquetes, errores), registros de eventos (Syslog) y mapas de tráfico por empresa o por dispositivo.

- Streamlit fue utilizado como interfaz ligera para presentar los resultados generados por los modelos de inteligencia artificial. Se construyó una aplicación que muestra. Gráficos de error de reconstrucción y umbral de anomalía, Puntos identificados como anómalos con su detalle, Tablas interactivas con las secuencias afectadas y actualización automática cada 5 minutos a partir de los datos almacenados en SQLite.

La combinación de estas dos herramientas permitió una visualización operativa (Kibana) y analítica (Streamlit) complementaria, brindando una visión integral tanto para monitoreo técnico como para análisis de predicción avanzada.

CAPÍTULO III. RESULTADOS Y DISCUSIÓN

3.1 Arquitectura de red

La arquitectura de red del sistema se diseñó bajo un enfoque distribuido, interconectando múltiples redes físicas a través de una red virtual definida por software, con el fin de centralizar la recopilación y análisis de datos. Este diseño garantiza escalabilidad, seguridad en la transmisión de datos y segmentación lógica entre redes como se muestra en la figura 5.

3.1.1 Conexión entre redes mediante ZeroTier

ZeroTier fue utilizado como plataforma SD-WAN para la creación de una red virtual privada sobre Internet, permitiendo el enlace seguro de las distintas ubicaciones involucradas en el monitoreo.

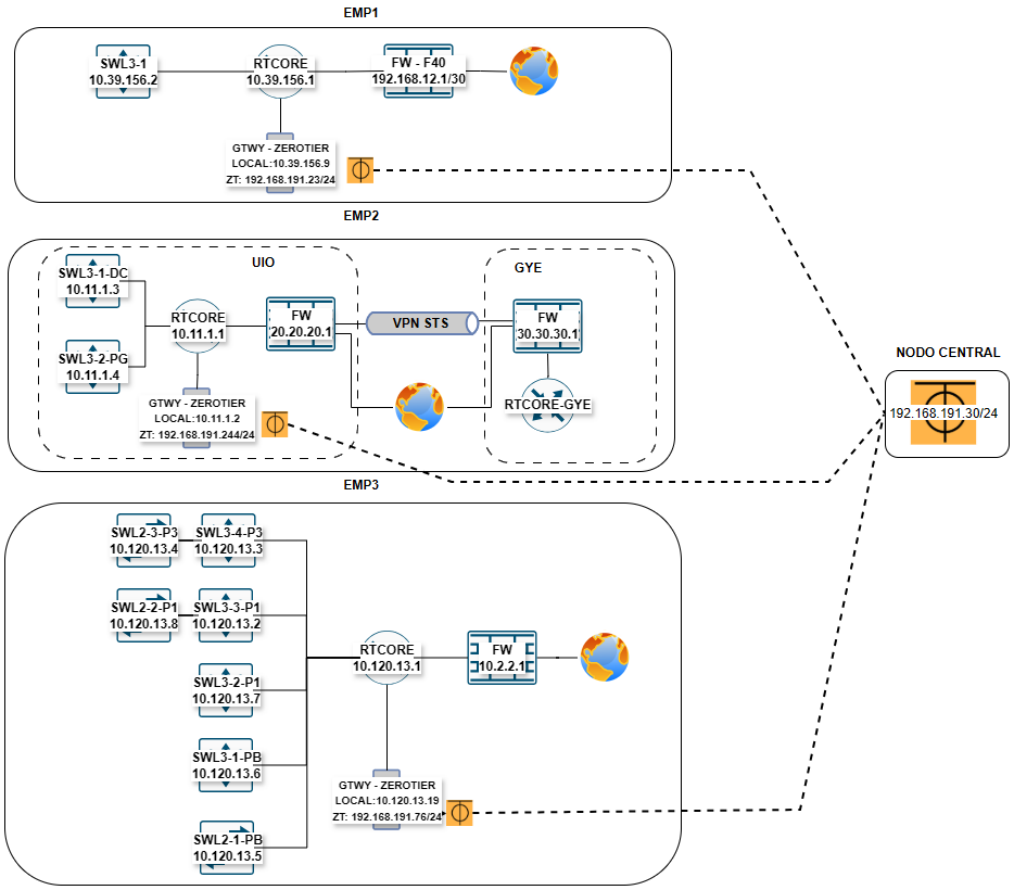


Figura 5. Diagrama de conexión de alto nivel entre redes mediante ZeroTier

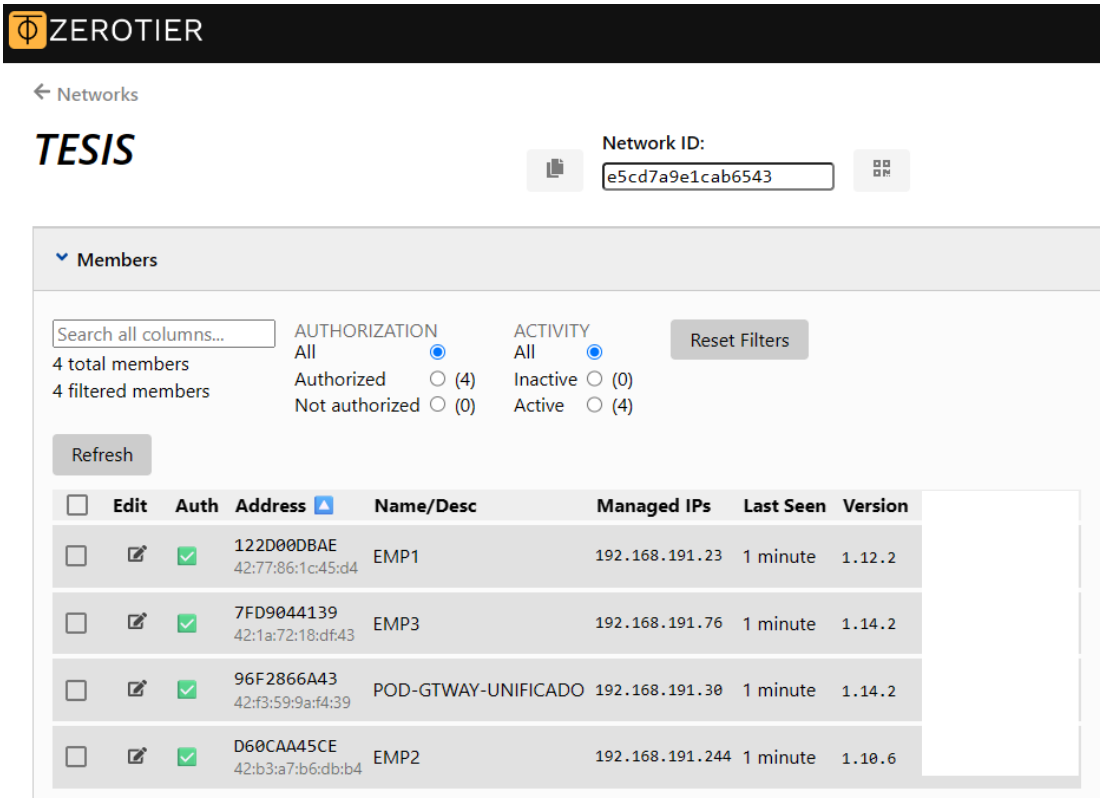


Figura 6. Panel de control ZeroTier, autorización y validación de nodos

La topología definida consta de:

a. Nodos de sucursales

Cada red remota (empresa o ubicación geográfica) incluyó un equipo físico o Virtual (PC) actuando como gateway, conectado tanto a la red local como a la red ZeroTier. Este nodo fue responsable del reenvío de tráfico hacia el servidor central.

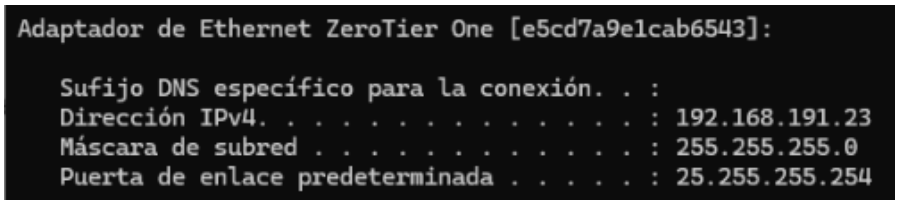


Figura 7. Direccionamiento lógico EMP1

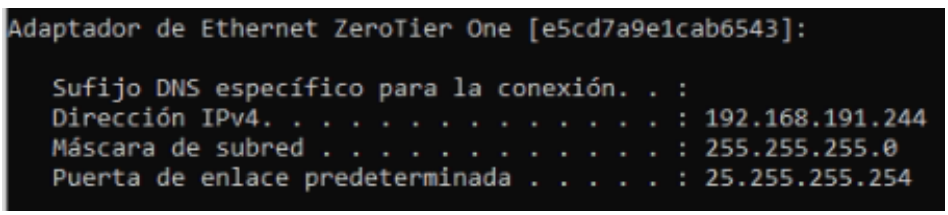


Figura 8. Direccionamiento lógico EMP2

```
Adaptador de Ethernet ZeroTier One [e5cd7a9e1cab6543]:

   Sufijo DNS específico para la conexión. . :
   Dirección IPv4. . . . . . . . . . . . . . : 192.168.191.76
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 25.255.255.254
```

Figura 9. Direccionamiento lógico EMP3

b. Nodo central (servidor)

El servidor principal, desplegado sobre una máquina virtual en Proxmox, se configuró como nodo maestro de la red ZeroTier dentro de un pod en Kubernets. Este nodo concentró los datos enviados desde todas las redes remotas y alojó los servicios de procesamiento e inteligencia artificial.

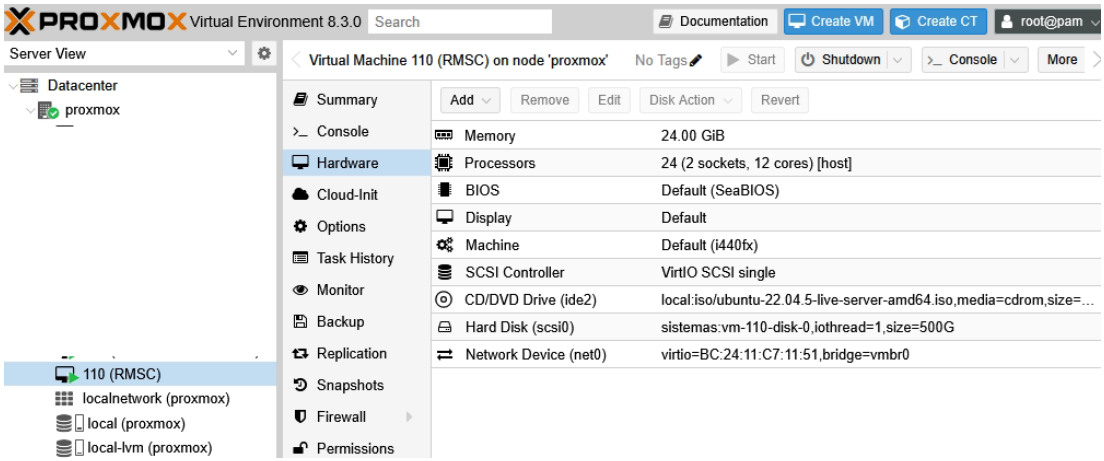


Figura 10. VM en Proxmox

```
2: ztppi5d627: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 2800 qdisc fq_codel state UNKNOWN group default qlen 1000
    link/ether 42:f3:59:9a:f4:39 brd ff:ff:ff:ff:ff:ff
    inet 192.168.191.30/24 brd 192.168.191.255 scope global ztppi5d627
       valid_lft forever preferred_lft forever
```

Figura 11. Direccionamiento lógico de nodo central

c. Enrutamiento estático

En cada gateway se definieron rutas estáticas para alcanzar las redes internas desde ZeroTier como se muestra en la figura 12, asegurando que los paquetes SNMP, NetFlow y Syslog fluyeran adecuadamente hacia el servidor central.

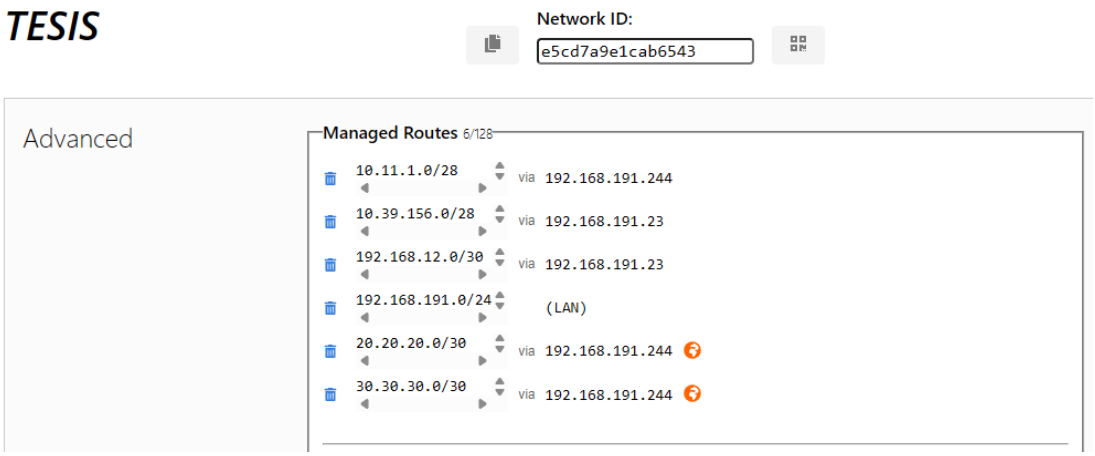


Figura 12. Tabla de enrutamiento red ZeroTier

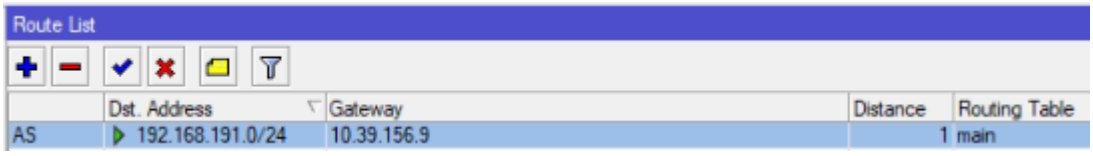


Figura 13 Tabla de enrutamiento equipo core EMP1

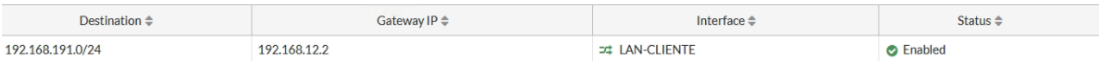


Figura 14. Tabla de enrutamiento equipo FW EMP1

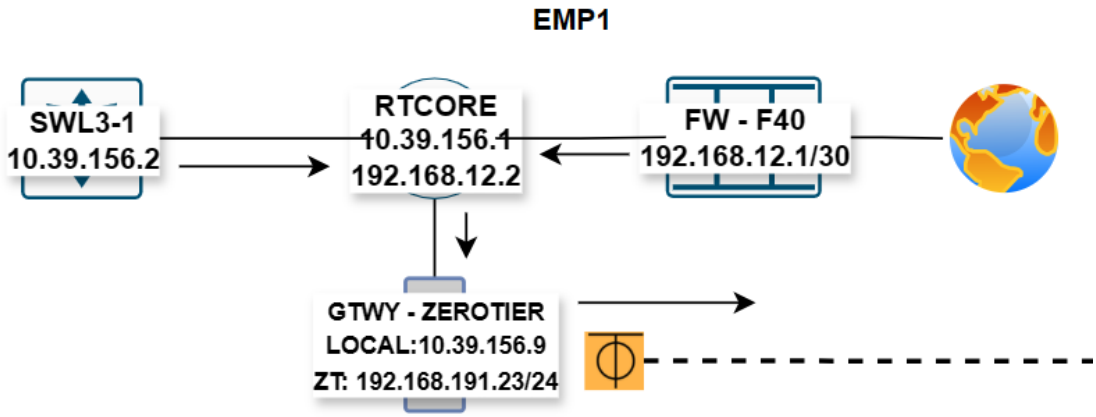


Figura 15. Flujo de comunicación y enrutamiento hacia nodo central – EMP1

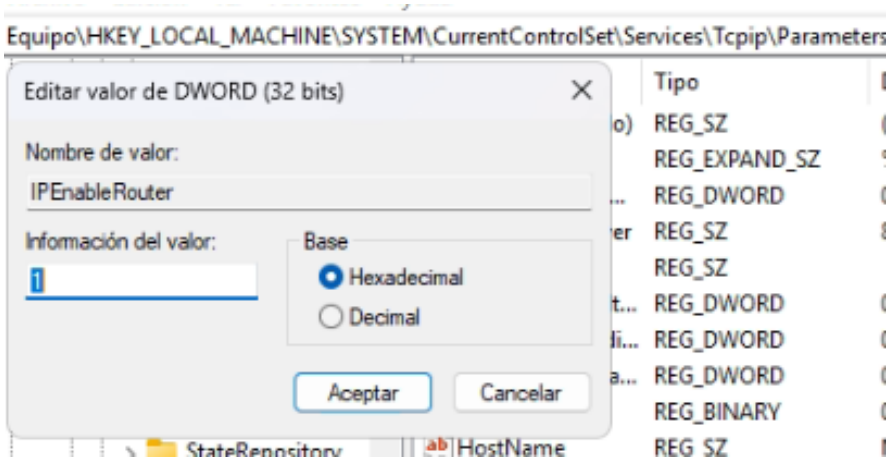


Figura 16. Habilitación de reenvío entre interfaces equipo PC - EMP1

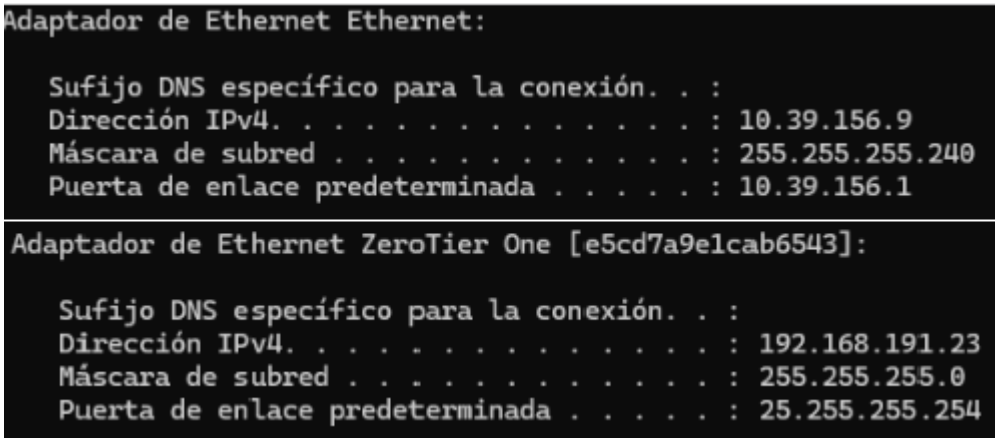


Figura 17. Interfaz de red local y red ZeroTier

El análisis presentado en las Figuras 13, 14, 15 y16 se realiza en todos los nodos de las sucursales para tener comunicación entre los diferentes nodos.

d. Validación de comunicación

Se realizaron pruebas de conectividad desde los dispositivos de red (MikroTik y Fortinet) hacia la IP del servidor principal (dentro del esquema ZeroTier) mediante ping, traceroute para verificar la recepción efectiva de tráfico como se muestra en la figura 18 y 19.

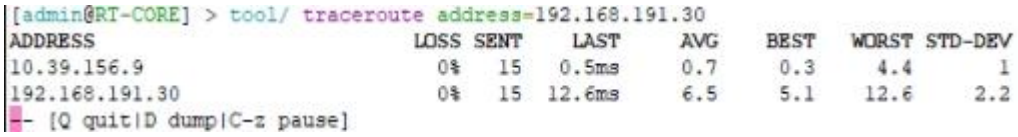


Figura 18. Pruebas de comunicación desde Router core hacia nodo central

```
traceroute to 192.168.191.30 (192.168.191.30), 32 hops max, 3 probe packets per hop, 84 byte packets
 1  192.168.12.2  0.167 ms  0.072 ms  0.059 ms
 2  10.39.156.9  0.896 ms *  0.536 ms
 3  192.168.191.30  6.428 ms  6.092 ms *
```

Figura 19. Pruebas de comunicación desde Firewall hacia nodo central

3.1.2 Configuración de protocolos de monitoreo

Para garantizar una recolección robusta de datos, se habilitaron y configuraron múltiples protocolos estándar en los equipos de red:

a. Configuración de SNMP Walk

Se habilitó SNMPv2c en los equipos MikroTik (CCR2116-12G-4S+, CRS328-24P-4S+) como se muestra en la figura 20, y en el firewall Fortinet (FGT40F) como se muestra en la figura 22, permitiendo consultas periódicas mediante snmpwalk. Se definieron comunidades de lectura seguras y se seleccionaron OIDs específicos para interfaces y sensores de hardware.

```
[admin@RT-CORE] > snmp/ community/ print
Flags: * - DEFAULT
Columns: NAME, ADDRESSES, SECURITY, READ-ACCESS, WRITE-ACCESS
#   NAME                ADDRESSES  SECURITY  READ-ACCESS  WRITE-ACCESS

1   MonEmp12025         ::/0       none      yes          no
```

Figura 20. Configuración SNMP Walk en equipo Mikrotik

b. Configuración de SNMP Trap

Se activó la funcionalidad de SNMP trap en en los equipos Mikrotik y Fortinet como se muestra en la figura 21 y 22, para el envío automático de eventos críticos. El servidor central fue configurado como receptor (trap listener) a través del puerto UDP 162, validando la recepción de mensajes asincrónicos.

```
[admin@RT-CORE] > snmp print
           enabled: yes
           contact: RT-CORE
          location:
  engine-id-suffix:
         engine-id: 80003a8c04
       src-address: ::
      trap-target: 192.168.191.30
   trap-community: MonEmp12025
     trap-version: 2
  trap-generators: temp-exception
                   interfaces
                   start-trap
  trap-interfaces: all
               vrf: main
```

Figura 21. Configuración SNMP Trap en equipo Mikrotik

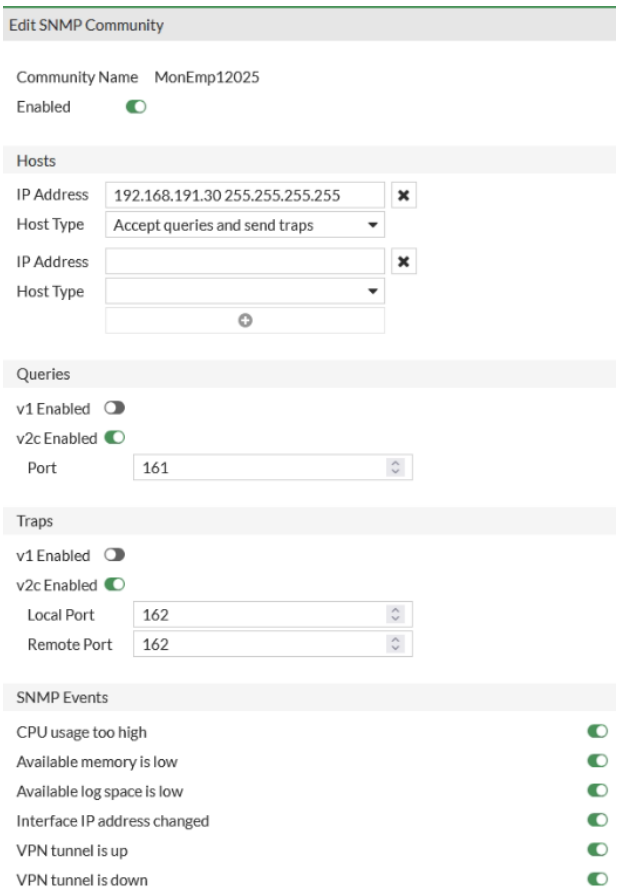


Figura 22. Configuración SNMP Walk y SNMP Trap en equipo Fortinet

c. Configuración de NetFlow

En los dispositivos MikroTik y Fortinet se habilitó NetFlow en su variante IPFIX, especificando el servidor central como destino, junto con el puerto UDP 2055. Se ajustaron los parámetros Active Timeout, Template Refresh Rate y Cache Timeout para optimizar la granularidad de los flujos como se muestra en la figura 23 y 24.

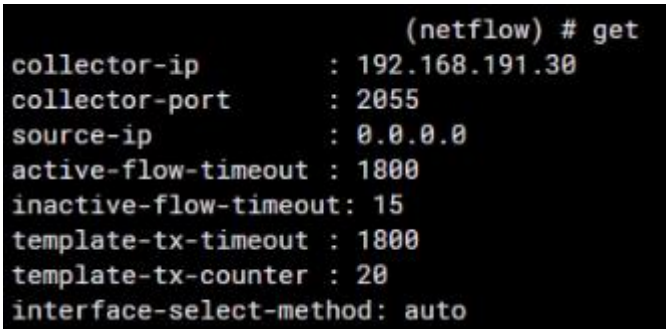


Figura 23. Configuración Netflow en equipo Fortinet



Figura 24. Configuración IPFIX en equipo Mikrotik

d. Configuración de Syslog

Se configuró la exportación de registros de sistema mediante Syslog hacia el servidor central (puerto UDP 514) en los equipos Mikrotik y Fortinet como se muestra en la figura 25 y 26, incluyendo eventos de seguridad, cambios de configuración, y alertas operativas. Se establecieron niveles de severidad (informativo, advertencia, crítico) y filtros por tipo de log.

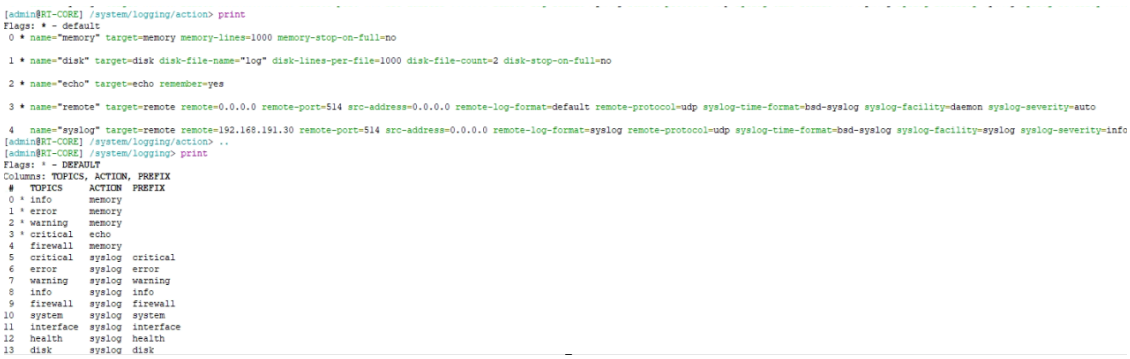


Figura 25. Configuración Syslog en equipo Mikrotik

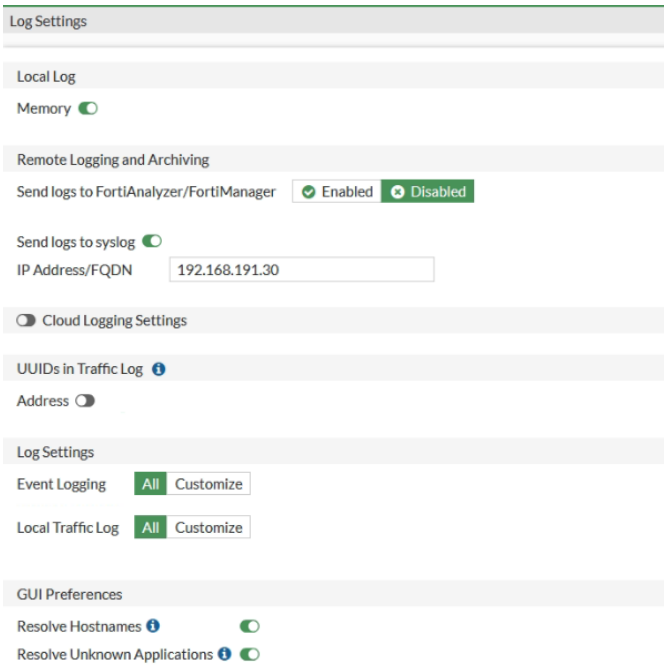


Figura 26. Configuración Syslog en equipo Fortinet

e. Validación de flujo de datos por ZeroTier

La validación del tráfico de protocolos se realizó mediante captura con Wireshark en la interfaz ZeroTier del nodo de las sucursales como se muestra en la figura 27 y mediante Tcpdump en el servidor, asegurando la llegada efectiva de paquetes SNMP, Syslog y NetFlow desde cada red remota como se muestra en la figura 28.

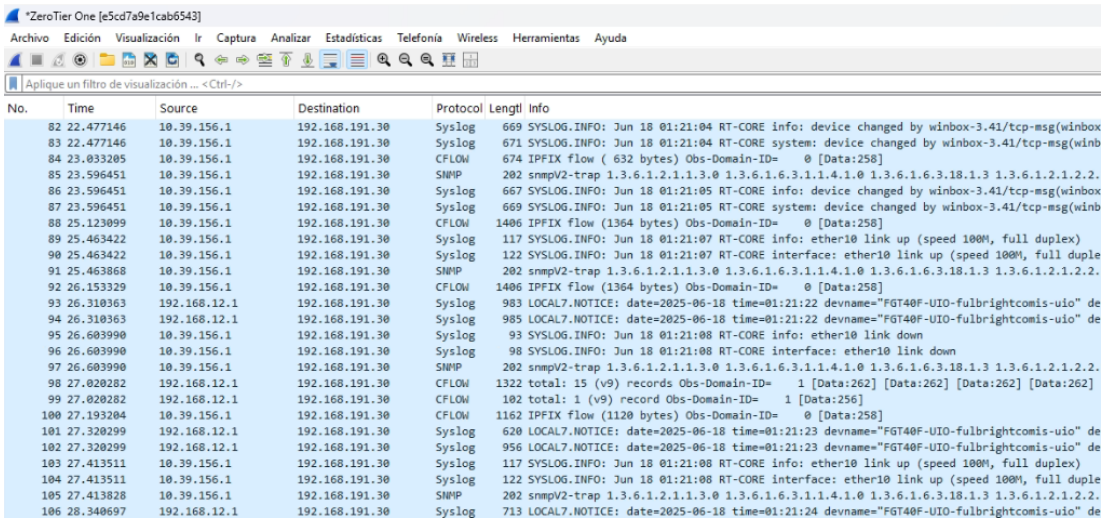


Figura 27. Validación de tráfico desde el nodo sucursal hacia el nodo central

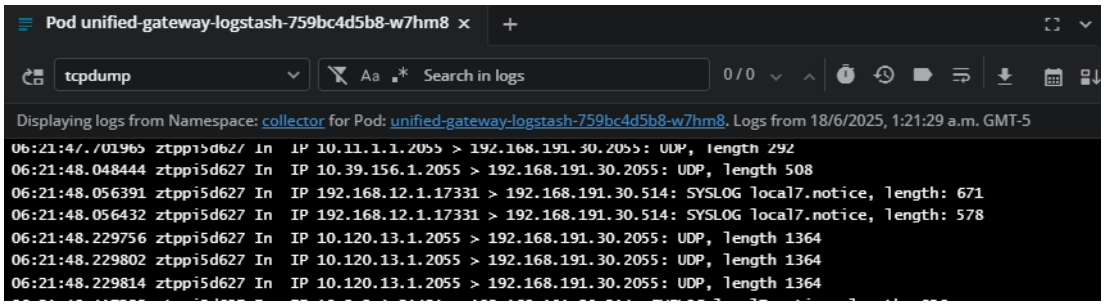


Figura 28. Validación de tráfico desde el nodo central

3.2 Arquitectura de microservicio

Con el objetivo de construir una solución escalable, modular y portable, se implementó una arquitectura basada en microservicios orquestados mediante contenedores (Figura 29). Este enfoque permitió desacoplar los distintos componentes funcionales del sistema —recolección de datos, almacenamiento, inferencia mediante modelos de inteligencia artificial y visualización de resultados—, facilitando tanto su desarrollo y mantenimiento independiente como su despliegue automatizado en entornos distribuidos. Además, esta estructura mejora la resiliencia, reutilización y escalabilidad horizontal, características esenciales para operar de forma eficiente en contextos de monitoreo y análisis en tiempo real.

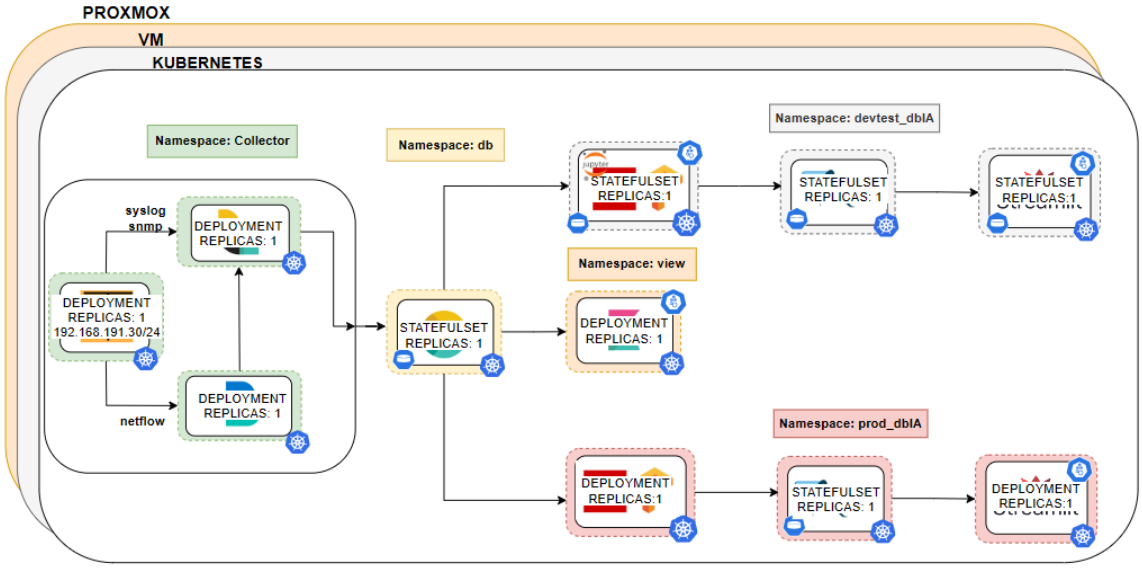


Figura 29. Descripción de la arquitectura basada en microservicios para el sistema de detección de anomalías de rendimiento y seguridad en redes de datos.

3.2.1 Levantamiento de máquina virtual con Kubernetes para implementar Docker

El entorno de ejecución se implementó sobre una máquina virtual (VM) provisionada mediante Proxmox VE, configurada con 8 vCPU, 32 GB de RAM y 500 GB de almacenamiento. Sobre esta VM se instaló Ubuntu Server 22.04 LTS, seguido de la instalación de Docker Engine y la configuración de un clúster Kubernetes mediante kubeadm.

Kubernetes permitió orquestar de forma eficiente los contenedores Docker, definiendo namespaces, controladores de despliegue, volúmenes persistentes y políticas de red para cada componente del sistema. Este diseño garantiza alta disponibilidad, flexibilidad en actualizaciones y facilidad para replicar el entorno en otros servidores o infraestructuras.

a. Configuración de pods y contenedores por namespace

Con el fin de mantener una separación lógica de funciones, se diseñó una segmentación por namespaces, asignando a cada uno un conjunto específico de pods y servicios:

- Namespace collector: Alberga los pods de entrada y procesamiento de datos crudos. Aquí se despliegan contenedores de Logstash y Filebeat, encargados de recibir, filtrar, transformar y enrutar los datos de SNMP, NetFlow e informes Syslog hacia la base de datos.

- Namespace db: Dedicado al almacenamiento. Contiene el pod de Elasticsearch, con volumen persistente para indexación, búsqueda y almacenamiento de métricas y eventos.

- Namespace view: Enfocado en la visualización operativa. Incluye el pod de Kibana, conectado directamente a Elasticsearch para mostrar dashboards en tiempo real, permitiendo monitoreo y análisis gráfico.

- Namespace ia-dev: Espacio de desarrollo e investigación. Aquí se desplegó JupyterLab como entorno de notebooks interactivos en Python, para experimentación, entrenamiento de modelos y pruebas de validación.

- Namespace ia-prod: Entorno de inferencia en producción. Contiene los scripts programados que, cada cinco minutos, consultan datos de Elasticsearch, aplican modelos de detección de anomalías previamente entrenados y almacenan los resultados en SQLite. Este proceso opera como servicio autónomo de predicción continua.

Cada pod cuenta con su propio archivo de definición (.yaml), donde se especifican imágenes personalizadas, variables de entorno, volúmenes, puertos y dependencias como se muestra en la figura 34. Los contenedores fueron creados y versionados con GitLab, asegurando trazabilidad y control de actualizaciones como se muestra en la figura 37.



Figura 30. Estado de Pods, Deployments y StatefulSet

Name	Namespace	Pods	Replicas	Age	Status
unified-gateway-logs	collector	1/1	1	5d7h	Running
jupyter-lab	ia-dev	1/1	1	5d7h	Running
kibana	view	1/1	1	5d7h	Running
python-prod	ia-prod	1/1	1	78s	Running

Figura 31. Estado de Pods configurados como Deployment

Name	Namespace	Pods	Replicas	Age
elasticsearch	db	1/1	1	5d7h

Figura 32. Estado de Pods configurados como Statefulset

Name	Namespac	Status	Containers	Node	Controlled E
unified-gateway-logstash-759bc4d5b8-v	collector	Running	■■■■	rmsc	ReplicaSet
elasticsearch-0	db	Running	■	rmsc	StatefulSet
jupyter-lab-6bd956d8f5-zlbnm	ia-dev	Running	■	rmsc	ReplicaSet
python-prod-f445f95cd-hq8sv	ia-prod	Running	■	rmsc	ReplicaSet
kibana-75d9c5fbdb-gv6lg	view	Running	■	rmsc	ReplicaSet

Figura 33. Estado de Pods

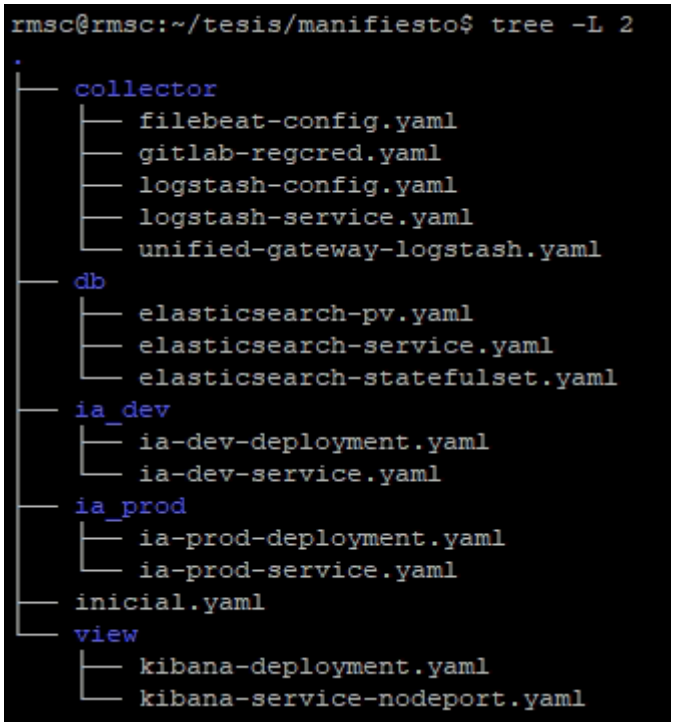


Figura 34 Arquitectura de ficheros de configuración para Kubernets

3.3 Control de versiones y repositorio

El uso de un sistema de control de versiones es fundamental en proyectos de ingeniería de software e inteligencia artificial, ya que permite la trazabilidad del código, gestión eficiente de cambios, trabajo colaborativo y mantenimiento de versiones estables del sistema. Para este proyecto se implementó una infraestructura de control de versiones utilizando Git como sistema distribuido y GitLab como plataforma de gestión centralizada.

3.3.1 Integración con GitLab para control de versiones y gestión de contenedores

Se configuró un repositorio privado en **GitLab**, donde se almacenaron:

- Archivos fuente del proyecto (.py, .ipynb, .yaml, .conf).
- Scripts de entrenamiento, validación y predicción de modelos de deep learning.

- Definiciones de contenedores Docker y archivos Dockerfile.
- Configuraciones de Logstash, Filebeat y despliegue de pods en Kubernetes.

Cada cambio fue documentado mediante commits estructurados, permitiendo un seguimiento detallado del desarrollo. Además, se generaron imágenes Docker personalizadas de los contenedores y procesamiento de datos, las cuales fueron almacenadas en el GitLab Container Registry como se muestra en la figura 35. Esto permitió el despliegue automatizado y reproducible de los servicios en el clúster Kubernetes, reduciendo errores por inconsistencias entre entornos.

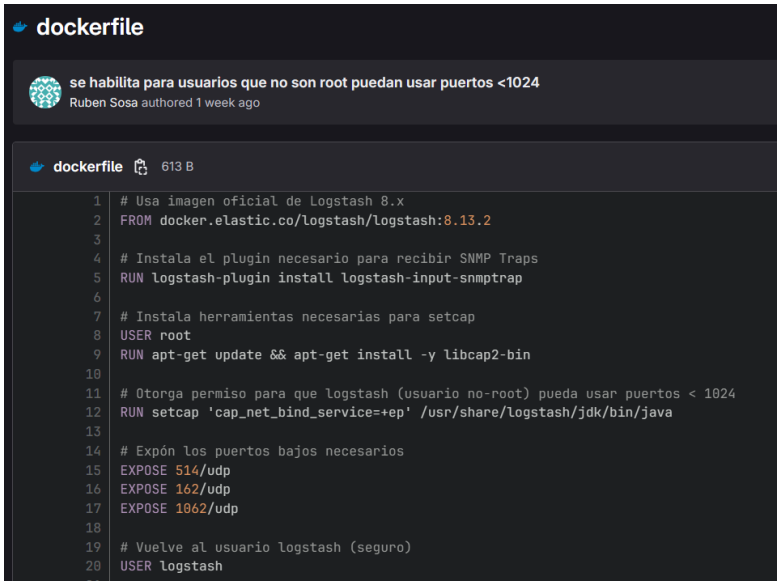


Figura 35. Dockerfile para crear imagen de docker personalizada para Logstash

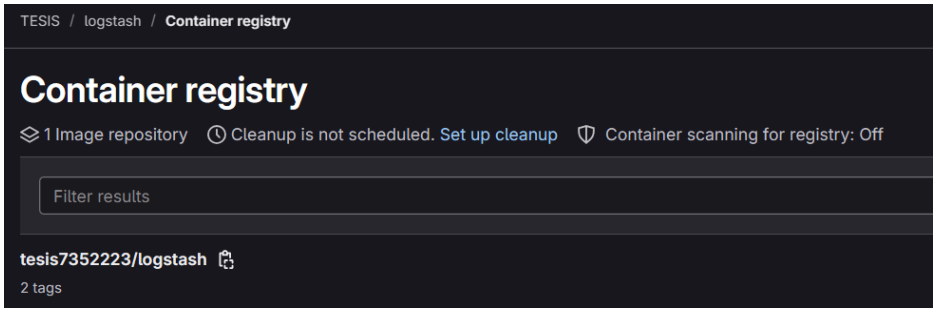


Figura 36. Docker en Gitlab para poder utilizar en kubernets

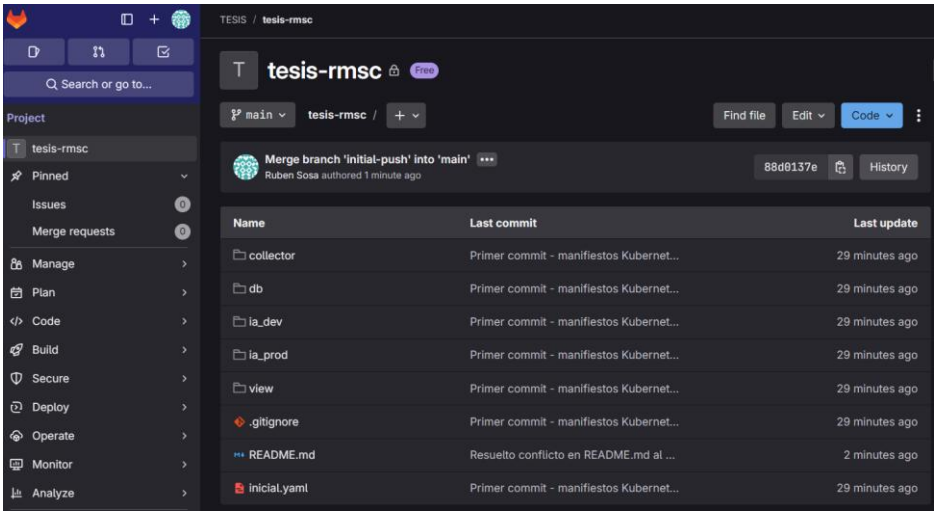


Figura 37. Arquitectura de ficheros en Gitlab para control de versiones y cambios

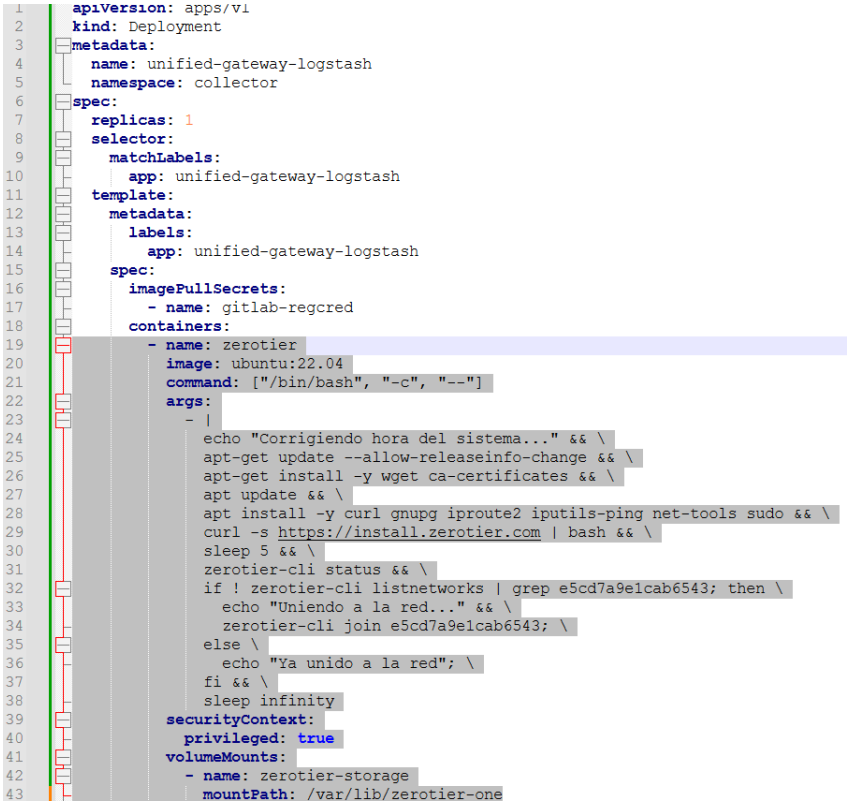
3.4 Sistema de detección de anomalías de rendimiento y seguridad en redes de datos

Se implementó un sistema distribuido de detección de anomalías en redes, basado en técnicas de aprendizaje profundo para el análisis de métricas de tráfico y eventos en tiempo real. La solución integra de forma cohesiva los procesos de adquisición de datos, preprocesamiento, modelado estadístico y predictivo, así como la visualización de resultados, permitiendo una supervisión continua y proactiva del comportamiento de la red frente a posibles desviaciones o incidentes de seguridad.

3.4.1 Ingesta, preprocesamiento y transformación de datos con Logstash y Filebeat

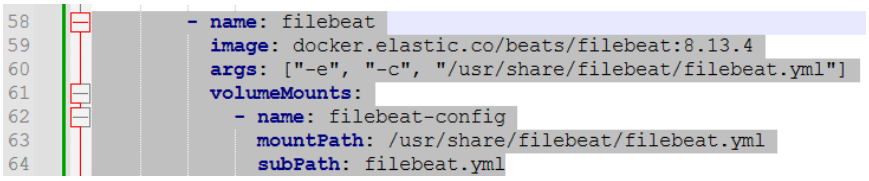
La recolección de datos se estructuró a través de un flujo centralizado de procesamiento:

- Zerotier: Permite la comunicación con redes de forma centralizada
- Filebeat fue utilizado para capturar flujos NetFlow exportados en formato IPFIX.



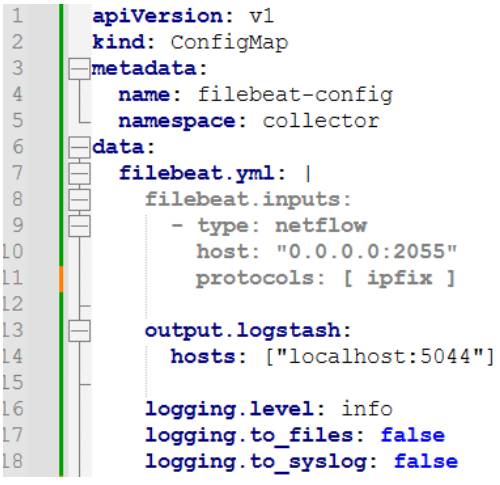
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: unified-gateway-logstash
  namespace: collector
spec:
  replicas: 1
  selector:
    matchLabels:
      app: unified-gateway-logstash
  template:
    metadata:
      labels:
        app: unified-gateway-logstash
    spec:
      imagePullSecrets:
        - name: gitlab-regcred
      containers:
        - name: zerotier
          image: ubuntu:22.04
          command: ["/bin/bash", "-c", "--"]
          args:
            - |
              echo "Corrigiendo hora del sistema..." && \
              apt-get update --allow-releaseinfo-change && \
              apt-get install -y wget ca-certificates && \
              apt update && \
              apt install -y curl gnupg iproute2 iputils-ping net-tools sudo && \
              curl -s https://install.zerotier.com | bash && \
              sleep 5 && \
              zerotier-cli status && \
              if ! zerotier-cli listnetworks | grep e5cd7a9e1cab6543; then \
                echo "Uniendo a la red..." && \
                zerotier-cli join e5cd7a9e1cab6543; \
              else \
                echo "Ya unido a la red"; \
              fi && \
              sleep infinity
          securityContext:
            privileged: true
          volumeMounts:
            - name: zerotier-storage
              mountPath: /var/lib/zerotier-one
```

Figura 38 Código para configurar ZeroTier en nodo central dentro del pod collector



```yaml
        - name: filebeat
          image: docker.elastic.co/beats/filebeat:8.13.4
          args: ["-e", "-c", "/usr/share/filebeat/filebeat.yml"]
          volumeMounts:
            - name: filebeat-config
              mountPath: /usr/share/filebeat/filebeat.yml
              subPath: filebeat.yml
```

Figura 39 Código para configurar Filebeat dentro del pod collector



```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-config
  namespace: collector
data:
  filebeat.yml: |
    filebeat.inputs:
      - type: netflow
        host: "0.0.0.0:2055"
        protocols: [ ipfix ]

    output.logstash:
      hosts: ["localhost:5044"]

    logging.level: info
    logging.to_files: false
    logging.to_syslog: false
```

Figura 40 Código para configuración de parámetros de Filebeat para recibir tráfico IPFIX/Netflow

- Logstash recibió simultáneamente datos SNMP y Syslog desde los dispositivos de red.

```
44          - name: logstash
45            image: registry.gitlab.com/tesis7352223/logstash:latest
46            ports:
47              - containerPort: 5044
48                protocol: UDP
49              - containerPort: 514
50                protocol: UDP
51              - containerPort: 162
52                protocol: UDP
53            volumeMounts:
54              - name: logstash-config
55                mountPath: /usr/share/logstash/pipeline/logstash.conf
56                subPath: logstash.conf
```

Figura 41 Código para configurar Logstash dentro del pod collector con imagen de Docker personalizada

```
1   apiVersion: v1
2   kind: ConfigMap
3   metadata:
4     name: logstash-config
5     namespace: collector
6   data:
7     logstash.conf: |
8
9       input {
125     }
126     # FILTER SECTION
127     filter {
259     }
260
261     # OUTPUT SECTION
262     output {
311     }
312
```

Figura 42. Configuración de parámetros de Filebeat para recibir tráfico IPFIX/Netflow

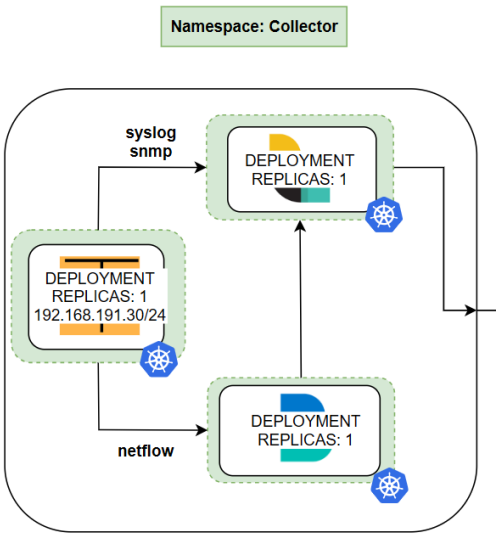


Figura 43 Flujo de transferencia de información mediante los contenedores dentro del pod collector

Se aplicaron filtros de procesamiento (grok, mutate, ruby, json) en Logstash para estructurar los campos relevantes, asignar etiquetas según origen y tipo de evento, y enriquecer los datos con atributos adicionales como timestamp, host, criticidad o tipo de tráfico como se muestra en el anexo A.

Este flujo permitió estandarizar entradas heterogéneas en un formato común antes de su almacenamiento e indexación.

3.4.2 Indexación de resultados hacia Elasticsearch

Los datos procesados fueron indexados en Elasticsearch, segmentados por tipo (tesis_netflow, tesis_snmpwalk, snmp_tesis_hdw_ft, snmp_tesis_hdw_mkt) y organizados cronológicamente mediante el campo @timestamp. Esto permitió realizar búsquedas eficientes, agregaciones temporales y consultas avanzadas con filtros y expresiones lógicas como se muestra en la figura 44.

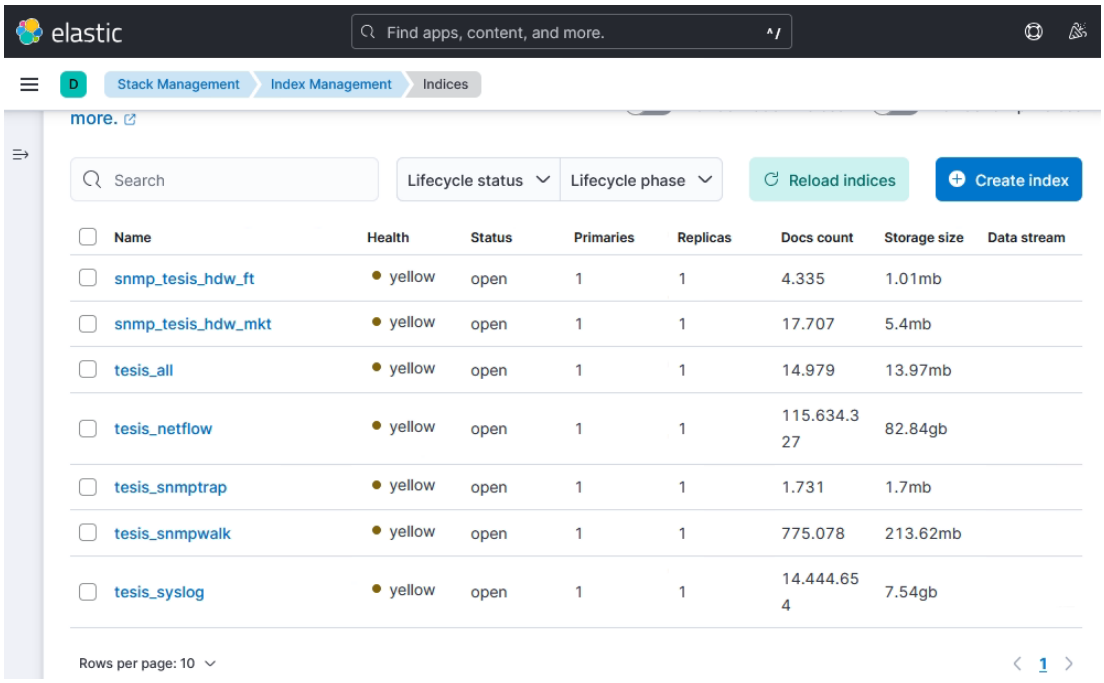


Figura 44. Visualización y comparación de índices en Elasticsearch

Cada documento en Elasticsearch contenía las métricas de red junto con metainformación estructurada, lo que facilitó su análisis por parte de los modelos de IA y su visualización en dashboards.

3.4.3 Visualización de información mediante dashboards en Kibana

Los datos indexados fueron representados gráficamente en Kibana, generando dashboards operativos. Esta capa visual permitió validar que la recolección, el procesamiento y la indexación de datos estaban operando correctamente, además de proporcionar una interfaz de monitoreo en tiempo real para realizar una validación cruzada con las anomalías detectadas.

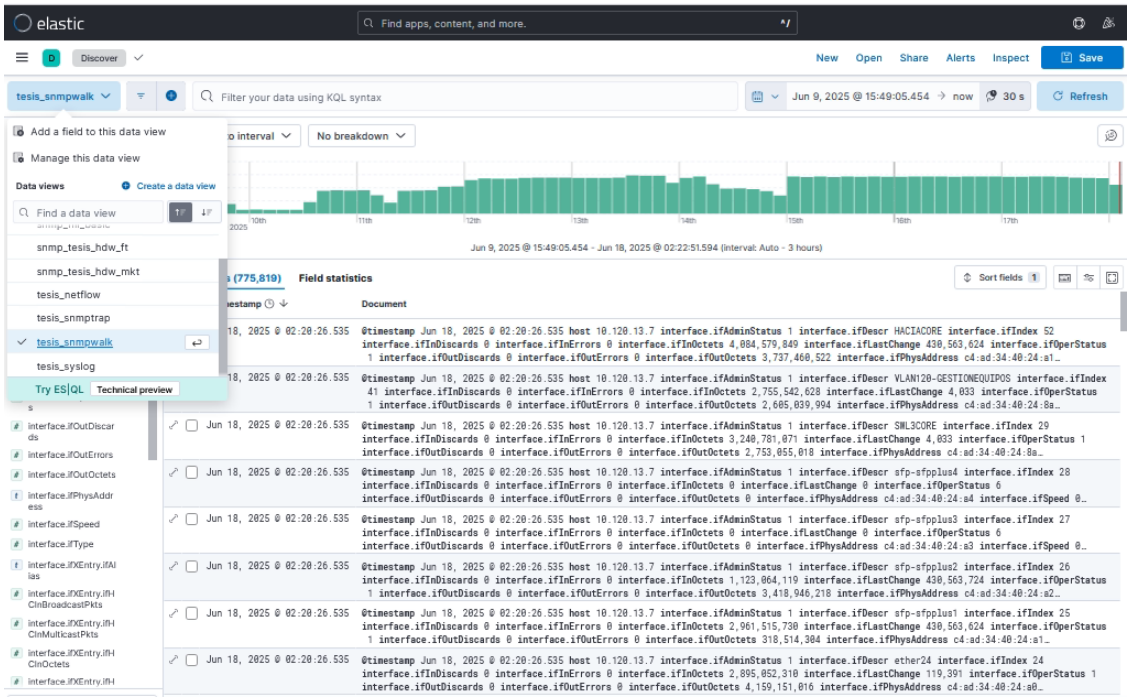


Figura 45. Visualización de datos de Elasticsearch en Kibana

Las figuras 46 a 49 ilustran distintas visualizaciones generadas a partir del procesamiento de flujos NetFlow y eventos de seguridad. La Figura 46 y la Figura 47 muestran dashboards de tráfico NetFlow centrados en el análisis de volumen y distribución de paquetes por dirección IP de origen y destino, lo que permite identificar los principales generadores y receptores de tráfico en la red. La Figura 48 representa un grafo de relaciones IP, útil para comprender patrones de comunicación entre nodos y detectar posibles flujos anómalos o conexiones inusuales. Finalmente, las Figuras 49 a 51 expone un dashboard de seguridad que clasifica eventos y URLs según su criticidad y país de destino, facilitando la correlación geográfica de amenazas y la priorización de acciones de mitigación. Estas visualizaciones son clave para el

monitoreo continuo, análisis forense y detección proactiva de anomalías en entornos de red.

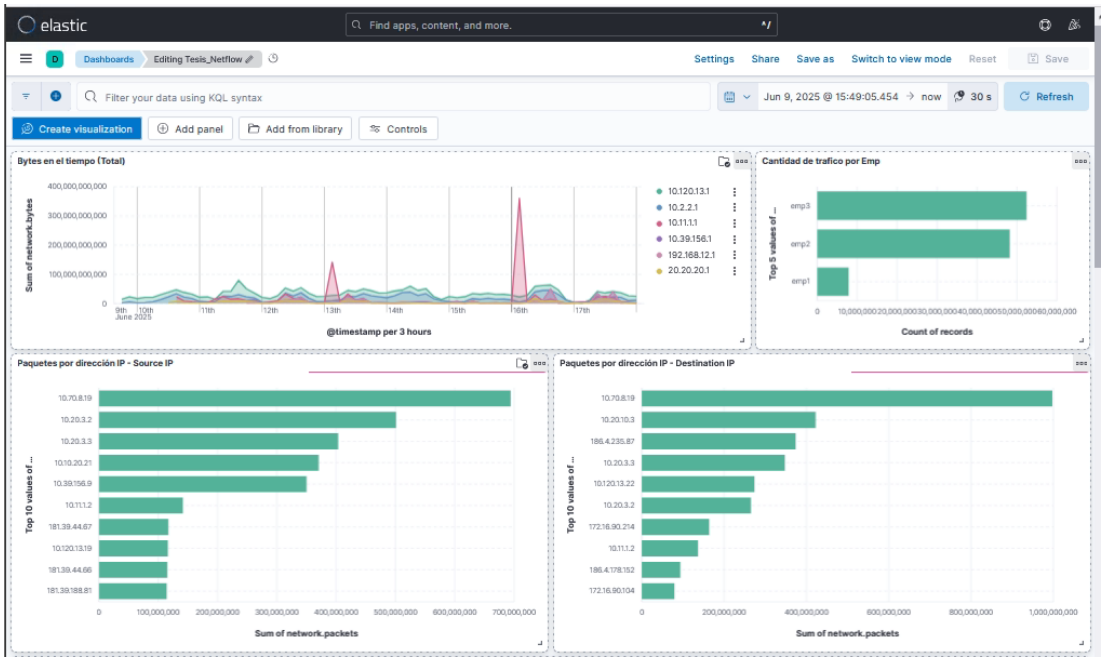


Figura 46. Dashboard de tráfico NetFlow – Visualización de volumen y distribución de paquetes

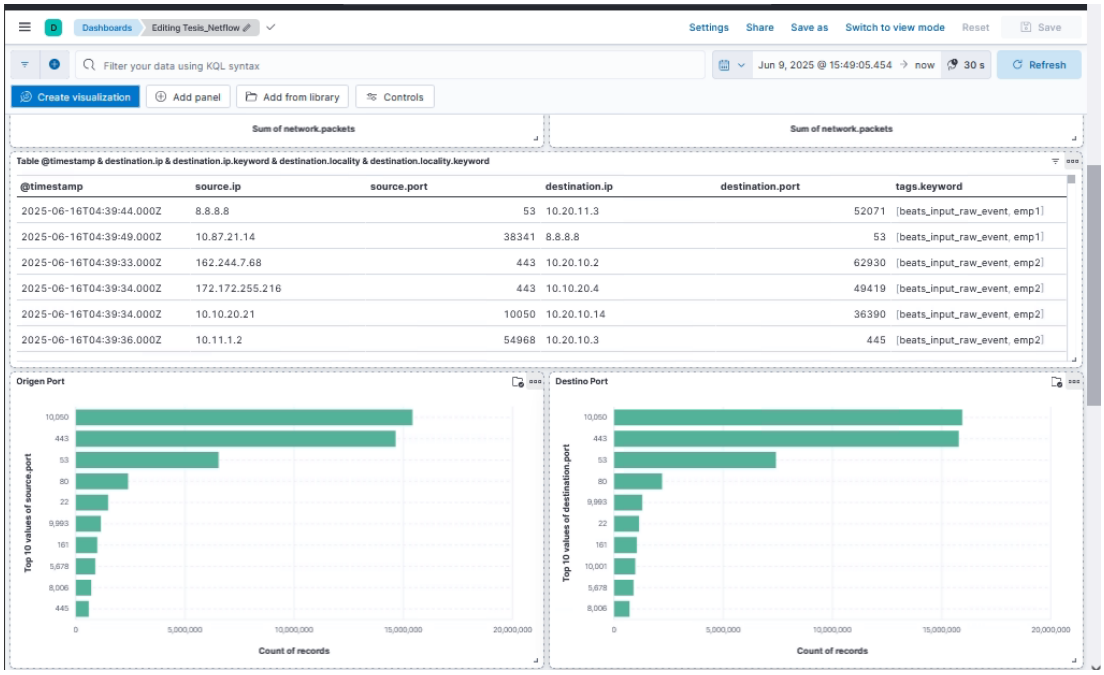


Figura 47. Dashboard de tráfico NetFlow – Visualización de volumen y distribución de paquetes

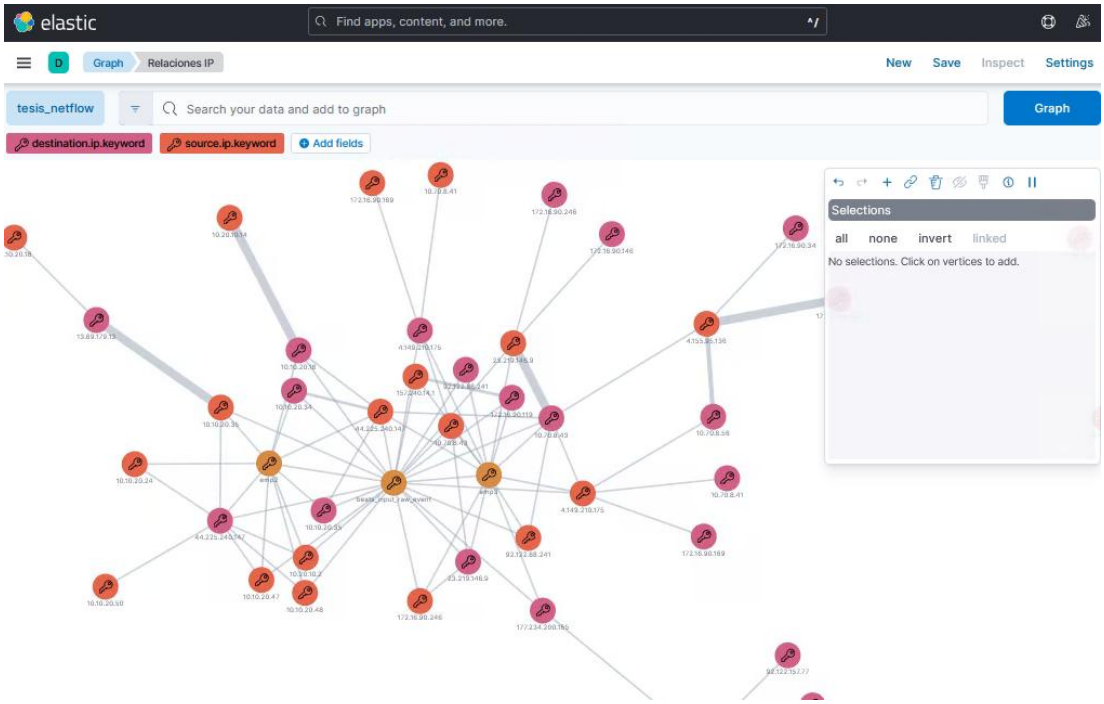


Figura 48. Relaciones entre direcciones IP – Grafo de comunicación NetFlow

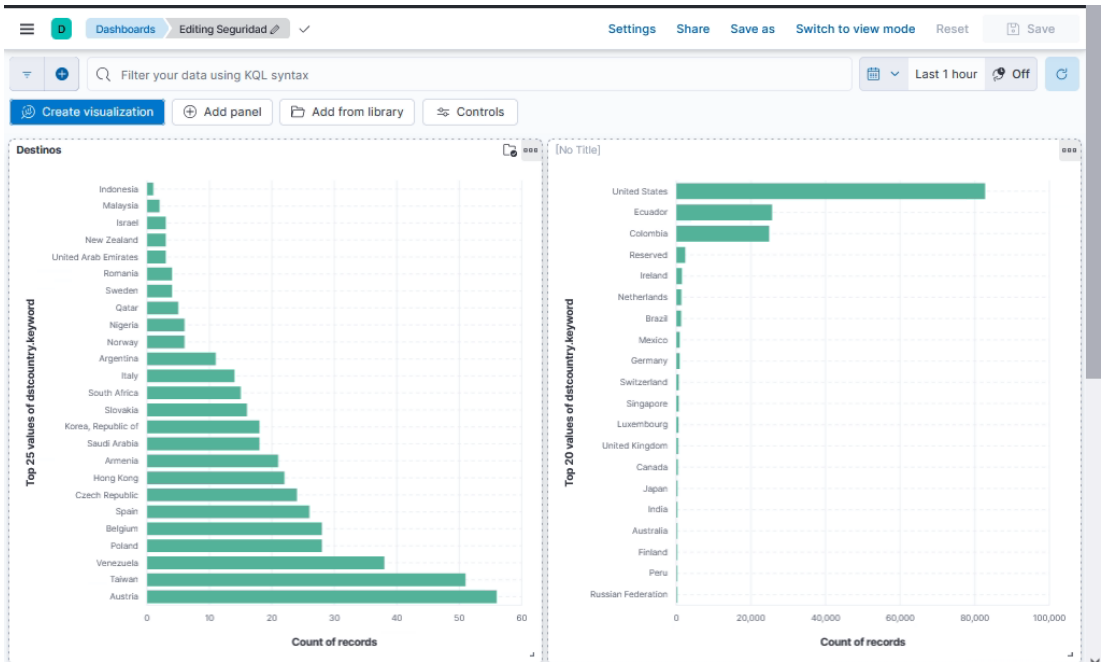


Figura 49. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino

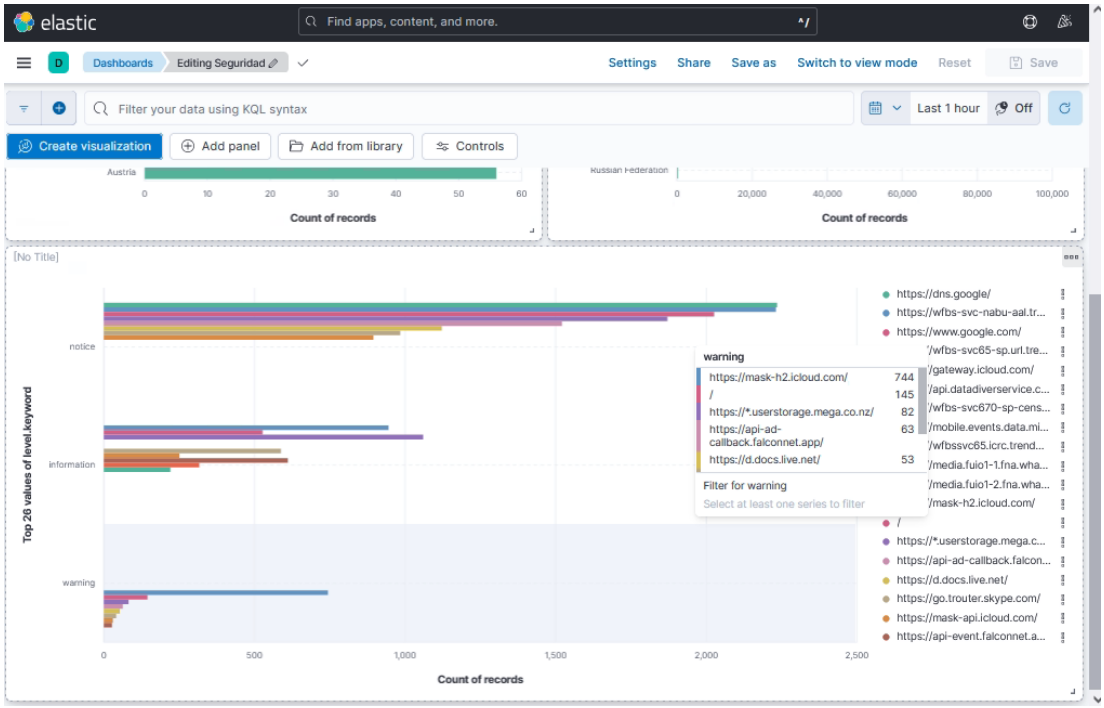


Figura 50. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino

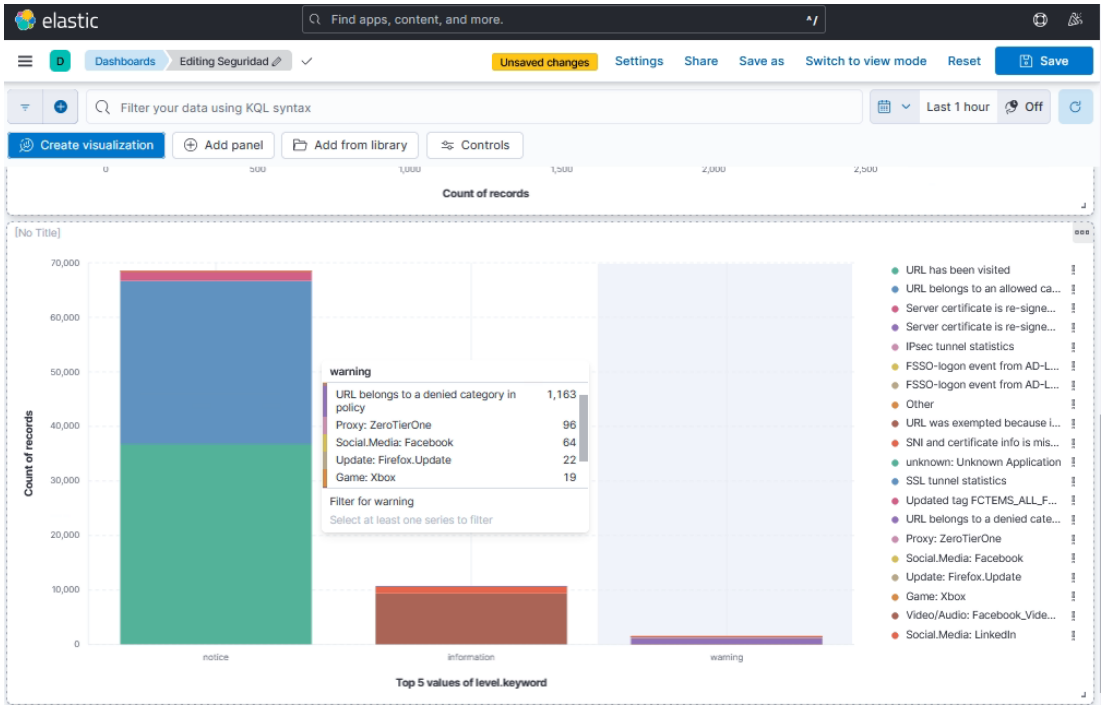


Figura 51. Dashboard de seguridad con eventos y URLs clasificadas por nivel y países de destino

3.4.4 Aplicación de algoritmos de deep learning

El proceso de aplicación de modelos de deep learning se llevó a cabo mediante una metodología comparativa entre tres arquitecturas distintas (RNN, LSTM y GRU), con el objetivo de identificar la opción más adecuada según el comportamiento de los datos recolectados. Para ello, se ejecutó un flujo estructurado que incluyó el preprocesamiento y tratamiento de los datos, contemplando imputación de valores nulos, normalización y codificación de variables. Posteriormente, se realizó la validación comparativa de los modelos, evaluando métricas como el error cuadrático medio (MSE), error absoluto medio (MAE) y el coeficiente de determinación (R^2). Con base en estos resultados, se seleccionó el modelo que ofrecía mejor desempeño respecto a la naturaleza de cada fuente de datos y su comportamiento en condiciones reales, asegurando una inferencia precisa y ajustada al contexto operativo.

a. Obtención de datos desde Elasticsearch

Los datos requeridos fueron consultados desde Elasticsearch utilizando scripts desarrollados en Python como se muestra en la figura 52.

b. Tratamiento estadístico y preprocesamiento

Antes de aplicar los modelos de inferencia, todos los datos fueron sometidos a un proceso riguroso y sistemático de preparación. Este procedimiento se aplicó de manera uniforme a todos los conjuntos de datos recopilados desde las distintas fuentes (NetFlow, SNMP Walk, SNMP Hardware y Syslog), con el objetivo de garantizar la calidad, consistencia y relevancia de la información utilizada.

El proceso de preparación incluyó las siguientes etapas: limpieza de datos, eliminación de columnas con altos porcentajes de valores nulos (figuras 53 y 54), imputación de datos faltantes en variables numéricas (figura 57), análisis de cardinalidad en variables categóricas (figura 56), normalización de variables numéricas mediante MinMaxScaler y codificación de variables categóricas con LabelEncoder (figura 58). Estas transformaciones permitieron homogeneizar los datos y optimizar su estructura para su posterior uso en la etapa de entrenamiento del modelo.

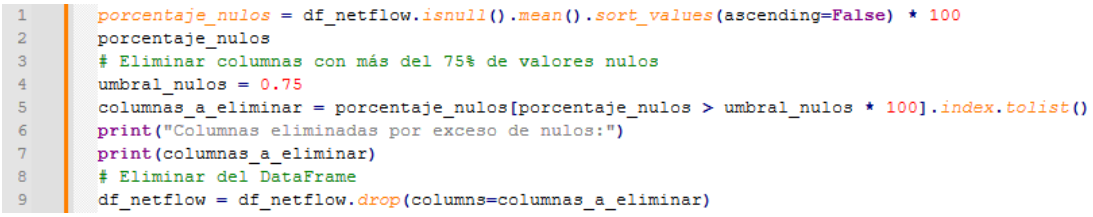


```python
from elasticsearch import Elasticsearch
import pandas as pd
# Conexión a tu instancia de Elasticsearch
es = Elasticsearch("http://elasticsearch.db.svc.cluster.local:9200")
# Función para obtener datos desde un índice
def get_elasticsearch_data(index_name, size=10000):
    query = {
        "size": size,
        "query": {
            "match_all": {}
        }
    }
    res = es.search(index=index_name, body=query)
    data = [hit['_source'] for hit in res['hits']['hits']]
    df = pd.json_normalize(data)
    df['@timestamp'] = pd.to_datetime(df['@timestamp'])
    #df.set_index('@timestamp', inplace=True)
    return df

# Obtener datos de cada fuente
netflow_data = get_elasticsearch_data("tesis_netflow")
snmpwalk_data = get_elasticsearch_data("tesis_snmpwalk")
snmp_ft_data = get_elasticsearch_data("snmp_tesis_hdw_ft")
snmp_mkt_data = get_elasticsearch_data("snmp_tesis_hdw_mkt")
syslog_data = get_elasticsearch_data("tesis_syslog")
# Crear DataFrames
df_netflow = pd.DataFrame(netflow_data)
df_snmpwalk = pd.DataFrame(snmpwalk_data)
df_snmp_ft = pd.DataFrame(snmp_ft_data)
df_snmp_mkt = pd.DataFrame(snmp_mkt_data)
# Dividir syslog por origen: Fortinet (10.2.x.x) y MikroTik (10.120.x.x)
df_syslog = pd.DataFrame(syslog_data)
df_syslog_ft = df_syslog[df_syslog['host'].str.startswith(("10.2.2", "20.20.", "30.30.", "192.168.12"))]
df_syslog_mkt = df_syslog[df_syslog['host'].str.startswith(("10.39", "10.11", "10.120"))]
# Visualización de ejemplo
print("NetFlow:", df_netflow.shape)
print("SNMP Walk:", df_snmpwalk.shape)
print("SNMP HW Fortinet:", df_snmp_ft.shape)
print("SNMP HW MikroTik:", df_snmp_mkt.shape)
print("Syslog Fortinet:", df_syslog_ft.shape)
print("Syslog MikroTik:", df_syslog_mkt.shape)
```

Figura 52. Código para Extracción y carga de datos desde Elasticsearch para múltiples fuentes (NetFlow, SNMP Walk, SNMP Hardware, Syslog) aplicando filtros por dirección IP para Fortinet y MikroTik.

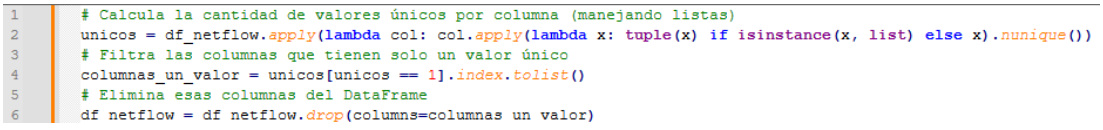
- Eliminación de columnas con más del 75% de valores nulos.



```python
porcentaje_nulos = df_netflow.isnull().mean().sort_values(ascending=False) * 100
porcentaje_nulos
# Eliminar columnas con más del 75% de valores nulos
umbral_nulos = 0.75
columnas_a_eliminar = porcentaje_nulos[porcentaje_nulos > umbral_nulos * 100].index.tolist()
print("Columnas eliminadas por exceso de nulos:")
print(columnas_a_eliminar)
# Eliminar del DataFrame
df_netflow = df_netflow.drop(columns=columnas_a_eliminar)
```

Figura 53. Código para Eliminación de columnas con más del 75% de valores nulos

- Eliminación de columnas con valores únicos.



```python
# Calcula la cantidad de valores únicos por columna (manejando listas)
unicos = df_netflow.apply(lambda col: col.apply(lambda x: tuple(x) if isinstance(x, list) else x).nunique())
# Filtra las columnas que tienen solo un valor único
columnas_un_valor = unicos[unicos == 1].index.tolist()
# Elimina esas columnas del DataFrame
df_netflow = df_netflow.drop(columns=columnas_un_valor)
```

Figura 54. Código para Eliminación de columnas con valores únicos

- Análisis de cardinalidad en variables categóricas.

```python
import matplotlib.pyplot as plt
# 1. Filtrar columnas categóricas
cat_cols = df_netflow.select_dtypes(include='object').columns
# 2. Calcular cardinalidad manejando listas como tuplas
cardinalidades = df_netflow[cat_cols].apply(
    lambda col: col.apply(lambda x: tuple(x) if isinstance(x, list) else x).nunique()
).sort_values(ascending=False)
# 3. Graficar
plt.figure(figsize=(12, 6))
cardinalidades.plot(kind='bar')
plt.title("Cardinalidad de columnas categóricas")
plt.ylabel("Número de valores únicos")
plt.xlabel("Columnas")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.grid(True)
plt.show()
```

Figura 55. Código para análisis de cardinalidad en columnas categóricas

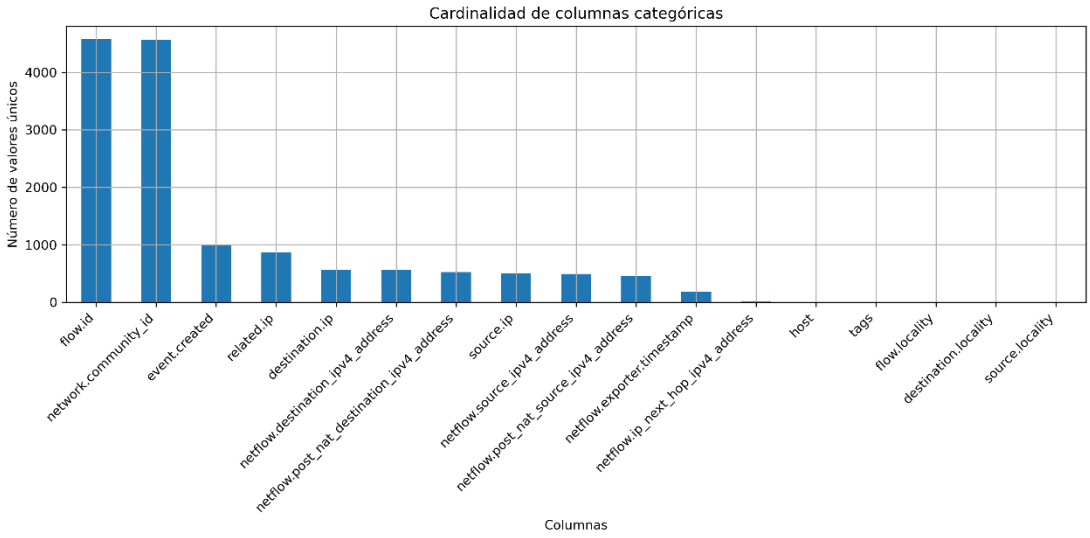


Figura 56. Gráfico de barras que representa la cardinalidad (número de valores únicos) de cada columna categórica para identificar columnas con alta o baja diversidad de valores.

- Imputación de valores faltantes.

```python
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
# Calcular el porcentaje de nulos por columna
porcentaje_nulos = df_netflow.isna().mean()
# Dividir las columnas por criterio
columnas_drop_filas = porcentaje_nulos[porcentaje_nulos <= 0.03].index.tolist()
columnas_imputar = porcentaje_nulos[porcentaje_nulos > 0.03].index.tolist()
# 1. Eliminar filas con nulos en columnas con <= 0.03% de nulos
df_netflow = df_netflow.dropna(subset=columnas_drop_filas)
# 2. Imputar columnas con > 0.03% de nulos
if columnas_imputar:
    # Filtrar solo numéricas para la imputación
    columnas_imputar_numericas = [col for col in columnas_imputar if df_netflow[col].dtype in [np.float64, np.int64]]
    # Usar mediana por robustez (evita outliers)
    imputer = SimpleImputer(strategy='median')
    df_netflow[columnas_imputar_numericas] = imputer.fit_transform(df_netflow[columnas_imputar_numericas])
```

Figura 57. Código para eliminar columnas con muchos nulos e imputar valores faltantes numéricos usando la mediana.

- Normalización de variables numéricas con MinMaxScaler.y Codificación de variables categóricas con LabelEncoder.

```
# Normalización de variables numéricas
numericas = df_netflow.select_dtypes(include=['float64', 'int64']).columns
scaler = MinMaxScaler()
df_netflow[numericas] = scaler.fit_transform(df_netflow[numericas])
# Codificación de variables categóricas
categoricas = df_netflow.select_dtypes(include=['object', 'category']).columns
for col in categoricas:
    encoder = LabelEncoder()
    df_netflow[col] = encoder.fit_transform(df_netflow[col].astype(str))
```

Figura 58. Código para normalizar variables numéricas con MinMaxScaler y codificar variables categóricas con LabelEncoder

- Aplicación de matriz de correlación para eliminar variables redundantes altamente correlacionadas.

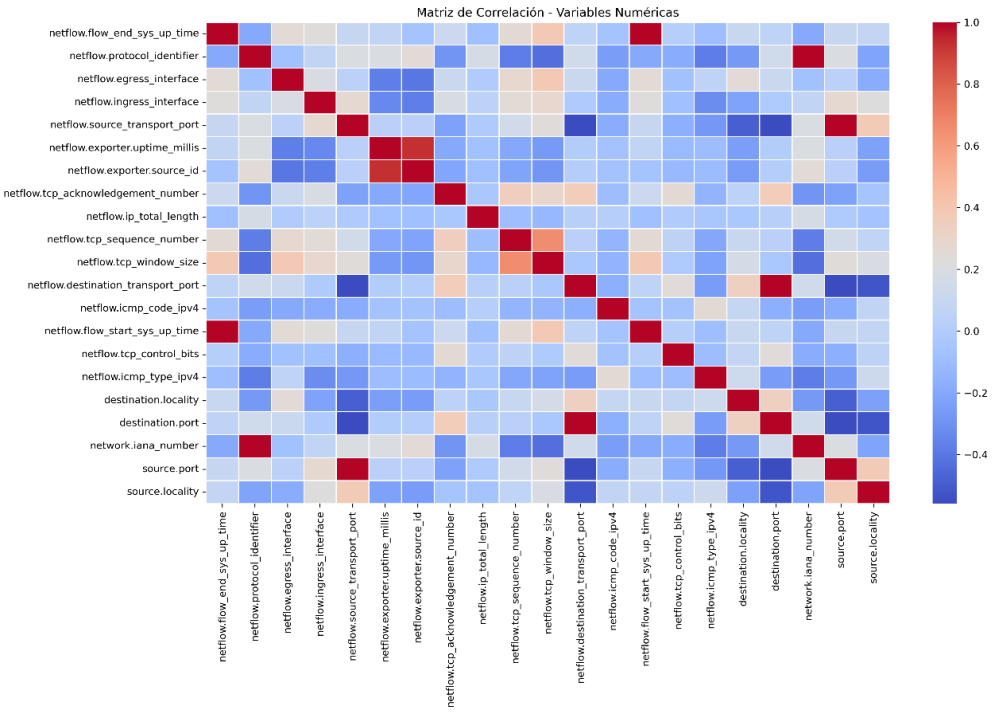


Figura 59. Matriz de correlación original de las variables

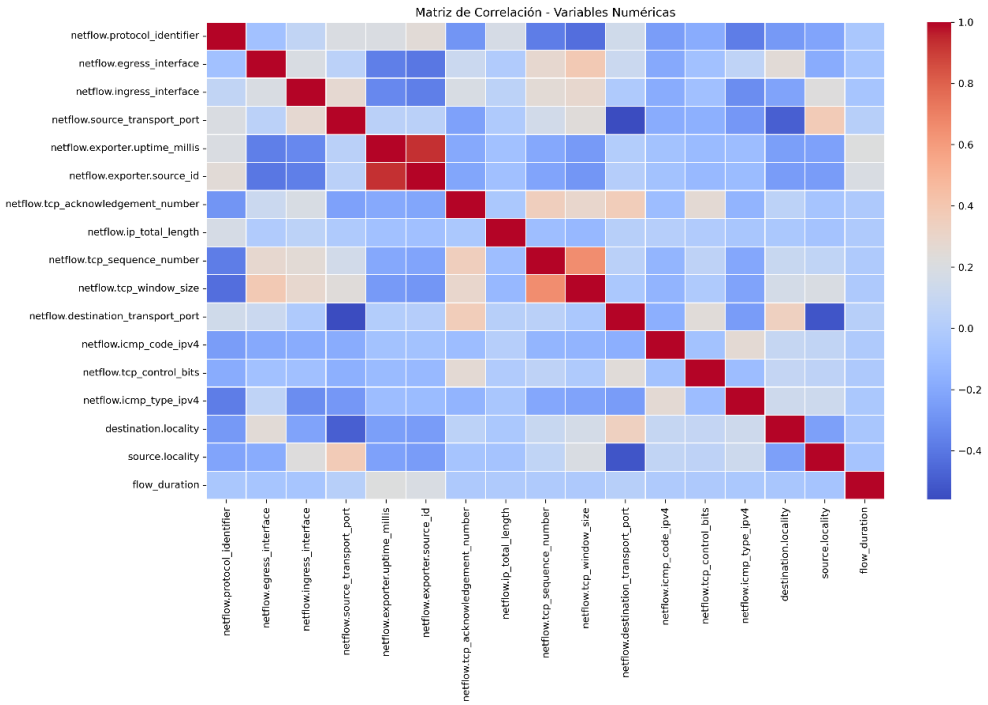


Figura 60. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas

	tipo_de_dato	nulos_%	unicos
@timestamp	datetime64[ns, UTC]	0.0	209
netflow.protocol_identifier	float64	0.0	4
netflow.egress_interface	float64	0.0	20
netflow.ingress_interface	float64	0.0	21
netflow.source_transport_port	float64	0.0	3282
netflow.exporter.uptime_millis	float64	0.0	162
netflow.exporter.source_id	float64	0.0	2
netflow.tcp_acknowledgement_number	float64	0.0	2486
netflow.ip_total_length	float64	0.0	218
netflow.tcp_sequence_number	float64	0.0	4370
netflow.tcp_window_size	float64	0.0	266
netflow.destination_transport_port	float64	0.0	2790
netflow.icmp_code_ipv4	float64	0.0	3
netflow.tcp_control_bits	float64	0.0	146
netflow.icmp_type_ipv4	float64	0.0	4
destination.locality	int64	0.0	2
source.locality	int64	0.0	2
flow_duration	float64	0.0	5182

Figura 61. Resumen de variables numéricas de NetFlow con tipo de dato, porcentaje de nulos y cantidad de valores únicos.

Este pipeline se aplicó sistemáticamente para cada tipo de dato (NetFlow, SNMP walk de interfaces, hardware, Syslog para MikroTik y Fortinet).

- Matriz de correlación SNMP Walk

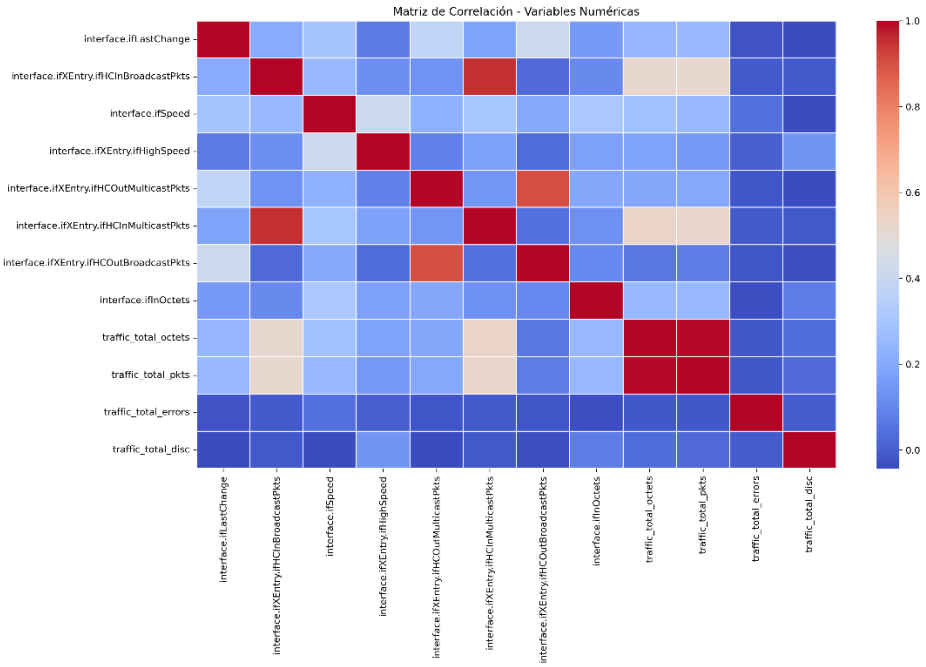


Figura 62. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – SNMP Walk

- Matriz de correlación Syslog

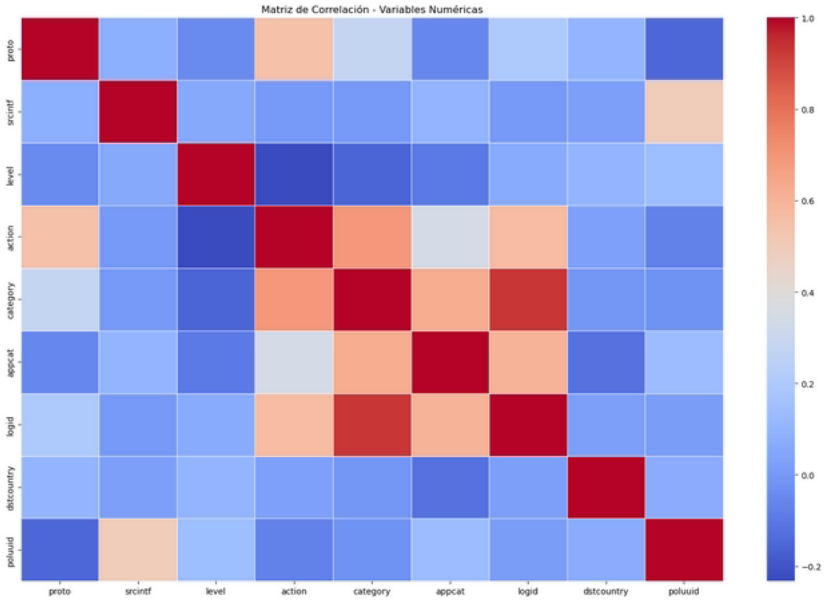


Figura 63. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – Syslog Fortinet

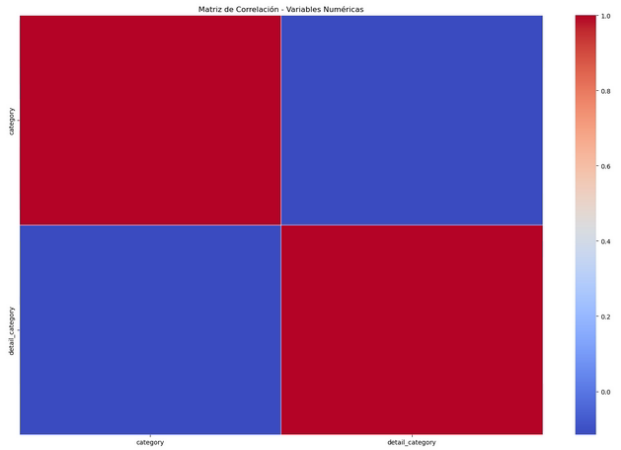


Figura 64. Matriz de correlación ajustada tras eliminar variables altamente correlacionadas – Syslog Mikrotik

Tras el análisis exploratorio y la evaluación de calidad de los datos recolectados, se decidió trabajar únicamente con los conjuntos provenientes de NetFlow y SNMP (tanto SNMP Walk de interfaces como de hardware), descartando los registros Syslog de MikroTik y Fortinet por las siguientes razones:

Syslog MikroTik fue descartado debido a la escasa cantidad de registros disponibles, lo cual limita significativamente su utilidad para el entrenamiento de modelos de detección de anomalías. La baja volumetría impide identificar patrones representativos y generalizables en el comportamiento del sistema.

Syslog Fortinet, a pesar de contar con un mayor volumen de datos, presenta redundancia informativa respecto al conjunto de NetFlow. Muchos eventos relacionados con conexiones, puertos, sesiones y flujos ya se encuentran representados de manera más estructurada y granular en los datos NetFlow. Esto hace innecesaria la inclusión del syslog de Fortinet, ya que no aporta valor adicional significativo al modelo y podría generar ruido o duplicidad en la representación del tráfico.

En cambio, los datos provenientes de NetFlow y SNMP ofrecen una mayor riqueza, consistencia y valor técnico. NetFlow proporciona métricas detalladas sobre el tráfico de red, incluyendo volumen, puertos, protocolos y duración de flujos, mientras que SNMP complementa con información de estado de interfaces y salud del hardware (temperatura, CPU, uso de memoria, etc.), permitiendo una visión integral tanto del tráfico como del estado de los dispositivos.

c. Entrenamiento y validación de modelos

Se evaluaron tres arquitecturas de redes neuronales profundas: RNN, LSTM y GRU. Para determinar cuál de estos modelos se adapta mejor a las características temporales y multivariadas de los datos recopilados, se realizó un análisis comparativo riguroso entre las tres opciones. La selección del modelo final se basó en la evaluación de métricas clave de desempeño, como el Error Cuadrático Medio (MSE), Error Absoluto Medio (MAE), Coeficiente de Determinación (R^2) y el Explained Variance Score (EVS). El objetivo fue identificar el modelo que ofreciera un equilibrio óptimo entre precisión, capacidad de generalización y estabilidad en la detección de patrones anómalos con un dataframe de cinco millones doce mil ciento veintiuno registros como se muestra en la figura 65.

	type	host	tags	@timestamp	destination.ip	destination.port
0	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-19 07:54:11	10.10.20.61	52299.0
1	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-19 07:54:19	10.20.10.14	37812.0
2	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-19 07:54:11	10.20.10.14	38126.0
3	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-19 07:54:11	10.20.10.14	50428.0
4	netflow_tesis	10.120.13.1	['beats_input_raw_event', 'emp3']	2025-06-19 07:54:32	10.70.8.19	43534.0
...	...	...	...	...	...	...
5000116	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-18 17:35:22	3.33.252.61	5222.0
5000117	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-18 17:35:22	10.20.10.14	52558.0
5000118	netflow_tesis	10.11.1.1	['beats_input_raw_event', 'emp2']	2025-06-18 17:35:22	10.10.20.13	60030.0
5000119	netflow_tesis	10.120.13.1	['beats_input_raw_event', 'emp3']	2025-06-18 17:35:47	10.120.13.19	63926.0
5000120	netflow_tesis	10.120.13.1	['beats_input_raw_event', 'emp3']	2025-06-18 17:35:47	52.111.227.6	443.0

5000121 rows × 100 columns

Figura 65. Cantidad de registros para evaluar los modelos RNN, GRU y LSTM

Preparación de datos

Generación de secuencias de longitud temporal fija (30 pasos), necesarias para alimentar los modelos RNN, LSTM y GRU.

```python
# Asegurarse de que el timestamp sea datetime
df_netflow['@timestamp'] = pd.to_datetime(df_netflow['@timestamp'])
# Ordenar por tiempo
df_netflow = df_netflow.sort_values('@timestamp').reset_index(drop=True)
# Seleccionar columnas numéricas para modelos (ya procesadas y normalizadas previamente)
df_numeric = df_netflow.select_dtypes(include=[np.number])
# 7. Generar secuencias
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data.iloc[i:i+seq_length].values)
        y.append(data.iloc[i+seq_length].values)
    return np.array(X), np.array(y)

X, y = create_sequences(df_numeric, seq_length=30)
```

Figura 66. Código para generar secuencias temporales a partir de variables numéricas normalizadas

Entrenamiento

Para el entrenamiento del modelo se utilizó la función de pérdida Error Cuadrático Medio (MSE) y el optimizador Adam, ampliamente empleado en tareas de regresión debido a su estabilidad y velocidad de convergencia.

La arquitectura de red fue definida de forma dinámica a través de la función build_model, según el tipo de red neuronal seleccionada (RNN, LSTM o GRU), y finalizada con una capa densa cuyo número de salidas coincide con la cantidad de variables originales del conjunto de datos.

Además, se incorporó la técnica de Early Stopping como estrategia de regularización. Esta técnica monitorea la pérdida en el conjunto de validación (val_loss) y detiene el entrenamiento de manera anticipada si no se observa mejora después de 5 épocas consecutivas, restaurando automáticamente los pesos de la mejor iteración del modelo. Esta medida ayuda a prevenir el sobreajuste y mejora la generalización.

```python
units_list = [32, 64, 128]
batch_sizes = [32, 64]
epochs_list = [10, 20, 30]
model = build_model((X.shape[1], X.shape[2]), cell_type=cell_type, units=units)
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = model.fit(X, y, validation_split=0.1, epochs=epochs, batch_size=batch_size, callbacks=[early_stop], verbose=0)
```

Figura 67. Código de configuración y entrenamiento del modelo con definición dinámica de hiperparámetros (units, batch_size, epochs) y uso de Early Stopping para evitar sobreajuste

El modelo fue entrenado utilizando hiperparámetros epochs, batch_size y units definidos de manera experimental para cada configuración, lo que permitió comparar objetivamente el desempeño estructural de cada arquitectura bajo condiciones controladas como se muestra en las figuras 67 y 68.

```python
def build_model(input_shape, cell_type='LSTM', units=64):
    model = Sequential()
    if cell_type == 'LSTM':
        model.add(LSTM(units, input_shape=input_shape))
    elif cell_type == 'GRU':
        model.add(GRU(units, input_shape=input_shape))
    elif cell_type == 'RNN':
        model.add(SimpleRNN(units, input_shape=input_shape))
    model.add(Dense(input_shape[-1]))  # Salida con el mismo número de features
    model.compile(optimizer='adam', loss='mse')
    return model
```

Figura 68. Código para evaluar modelos RNN, GRU y LSTM

Evaluación

Se aplicaron métricas como el Error Cuadrático Medio (MSE), el Error Absoluto Medio (MAE), el Coeficiente de Determinación (R^2) y la Varianza Explicada (EVS) para validar el desempeño del modelo.

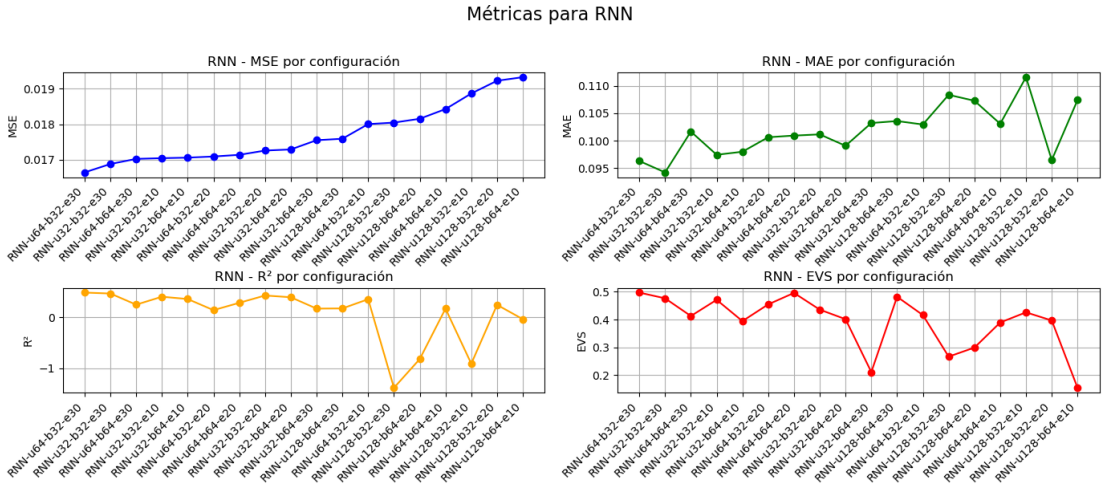


Figura 69. Desempeño del modelo RNN bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R^2 y EVS

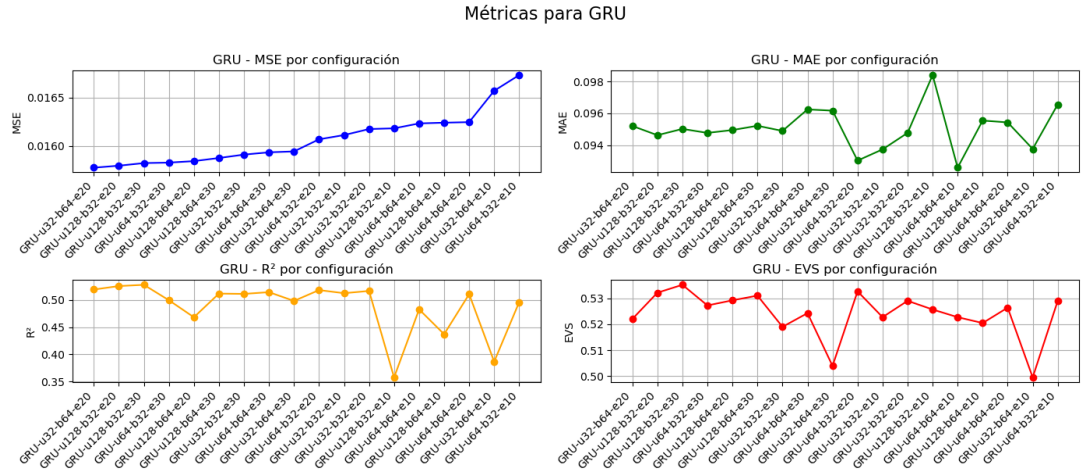


Figura 70. Desempeño del modelo GRU bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R² y EVS

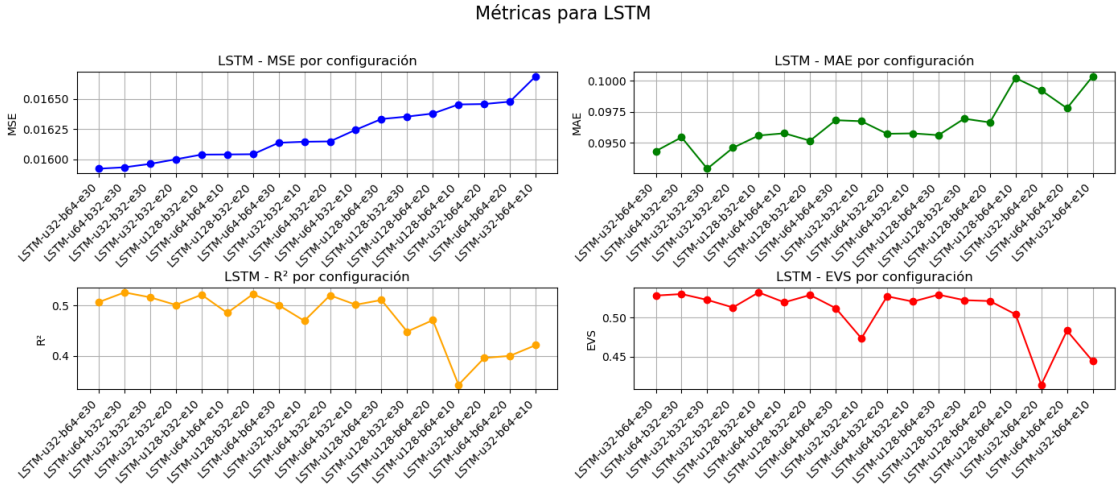


Figura 71. Desempeño del modelo LSTM bajo distintas configuraciones de hiperparámetros, evaluado mediante las métricas MSE, MAE, R² y EVS

Tabla 16. Comparación de desempeño entre modelos RNN, LSTM y GRU

Métrica	RNN	LSTM	GRU
R²	Muy bajo, inestable, incluso negativo	Más alto, hasta 0.53	Alto, hasta 0.52
MSE	Aceptable pero disperso	Bajo y consistente	Bajo y estable
MAE	Irregular	Bajo, ligeramente superior a GRU	Muy bajo y estable
EVS	Inestable	Alto (~0.51–0.53)	Alto (~0.52–0.53)
Estabilidad	Baja	Alta	Alta
Conclusión	No recomendable	Modelo seleccionado	Alternativa sólida

Los resultados mostraron un rendimiento claramente inferior por parte del modelo RNN (figura 69), con valores de R^2 inestables e incluso negativos en varias configuraciones, lo que indica una capacidad predictiva deficiente e inferior a la de un modelo constante. Por tanto, esta arquitectura fue descartada.

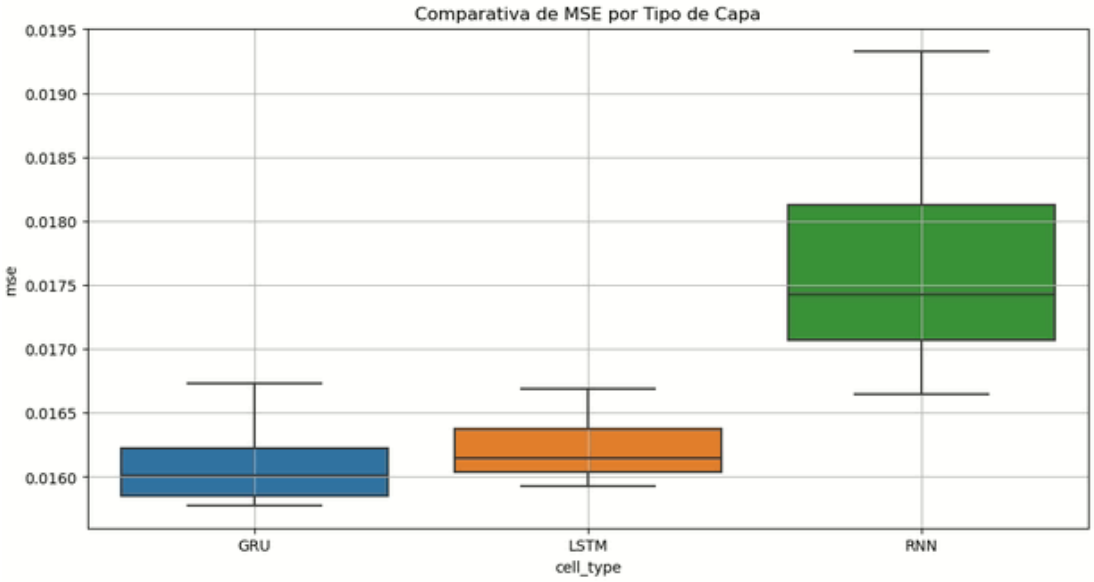


Figura 72. Comparativa de MSE por tipo de modelo

En contraste, tanto GRU (figura 70) como LSTM (figura 71) presentaron métricas sólidas y estables. El modelo GRU mostró un rendimiento consistente, con valores de R^2 entre 0.45 y 0.52, y errores (MSE y MAE) bajos y uniformes. Sin embargo, el modelo LSTM superó ligeramente a GRU en términos de R^2, alcanzando hasta 0.53, y mantuvo un buen equilibrio con el resto de métricas, incluyendo un EVS alto y una varianza explicada satisfactoria. Esto evidencia una mayor capacidad del LSTM para modelar secuencias temporales complejas y dependencias a largo plazo en los datos multivariados de tráfico de red.

d. Visualización de resultados en Streamlit

Una vez con el modelo entenado y listo se implementó un sistema automatizado de inferencia, encargado de recopilar nuevos registros desde Elasticsearch, procesarlos, aplicar el modelo correspondiente y almacenar los resultados en una base de datos local para su posterior análisis mediante un dashboard.

Extracción y preparación de datos desde Elasticsearch

Cada cinco minutos, el sistema ejecuta una consulta a Elasticsearch utilizando un filtro de tiempo basado en la hora actual, lo que garantiza que se utilicen únicamente los datos más recientes como se muestra en la figura 72. Esta consulta se realiza por separado para los índices correspondientes a NetFlow y SNMP Walk. Los datos recuperados se normalizan y transforman en un DataFrame estructurado y se realiza el tratamiento de los datos antes de ingresar al modelo.

```
# Configuraciones
ES_HOST = "http://elasticsearch.db.svc.cluster.local:9200"
ES_INDEX = "tesis_snmpwalk"
MODEL_PATH = "modelo_snmpwalk.keras"
SQLITE_PATH = "anomalies.db"
TABLE_NAME = "anomalies_snmpwalk_2"
SEQUENCE_LENGTH = 30
# Conectar a Elasticsearch
es = Elasticsearch(ES_HOST)
# Calcular rango de tiempo
end_time = datetime.utcnow()
start_time = end_time - timedelta(minutes=5)
# Consulta Elasticsearch
query = {
    "size": 10000,
    "query": {
        "range": {
            "@timestamp": {
                "gte": start_time.isoformat(),
                "lt": end_time.isoformat()
            }
        }
    }
}
res = es.search(index=ES_INDEX, body=query)
data = [hit['_source'] for hit in res['hits']['hits']]
df = pd.json_normalize(data)

if df.empty:
    print("No se encontraron datos nuevos para SNMP Walk.")
    exit()
```

Figura 73. Código para registro automático cada 5 minutos

Detección de anomalía y cálculo de error

El sistema carga el modelo previamente entrenado y realiza la predicción para cada secuencia. El resultado se evalúa calculando el error de reconstrucción mediante el Error Cuadrático Medio (MSE) por secuencia. A partir del conjunto de errores, se calcula un umbral dinámico basado en el percentil 99. Cualquier secuencia cuyo error supere este umbral es etiquetada como anómala (figura).

```python
# Cargar modelo y predecir
modelo = load_model(MODEL_PATH)
y_pred = modelo.predict(X_seq)
# Calcular error y umbral
y_real = y_seq
error = np.mean(np.square(y_pred - y_real), axis=1)
th = np.percentile(error, 99)
anomalias = error > th
```

Figura 74. Código para cargar modelo entrenado, realiza la detección de anomalías en base al umbral establecido por el percentil 99.

Almacenamiento estructurado de resultados

Los resultados son almacenados en una base de datos SQLite, separadamente por tipo de dato. Cada registro incluye, timestamp del evento, dispositivo (host),valor del error de reconstrucción, umbral de decisión, etiqueta binaria de anomalía, detalles del evento.

```python
# Guardar en SQLite
conn = sqlite3.connect(SQLITE_PATH)
c = conn.cursor()
c.execute(f'''
CREATE TABLE IF NOT EXISTS {TABLE_NAME} (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    timestamp TEXT,
    host TEXT,
    error REAL,
    threshold REAL,
    is_anomaly INTEGER,
    detalle TEXT
)
''')
offset = SEQUENCE_LENGTH
for i in range(len(X_seq)):
    row = df.iloc[i + offset]
    c.execute(f"""
        INSERT INTO {TABLE_NAME} (timestamp, host, error, threshold, is_anomaly, detalle)
        VALUES (?, ?, ?, ?, ?, ?)
    """, (
        (row['@timestamp'] - timedelta(hours=5)).isoformat(),
        row['host'],
        float(error[i]),
        float(th),
        int(anomalias[i]),
        json.dumps(row[campos_snmpwalk].to_dict()) if anomalias[i] else None
    ))
conn.commit()
conn.close()
```

Figura 75. Código para guardar los resultados en la base de datos SQLite

Visualización interactiva en tiempo real

Todos los resultados del sistema de detección de anomalías son presentados a través de una interfaz web desarrollada con Streamlit. Esta interfaz permite visualizar en tiempo real las gráficas de error y los umbrales definidos para cada tipo de dato (NetFlow y SNMPWalk), facilitando la identificación de eventos anómalos.

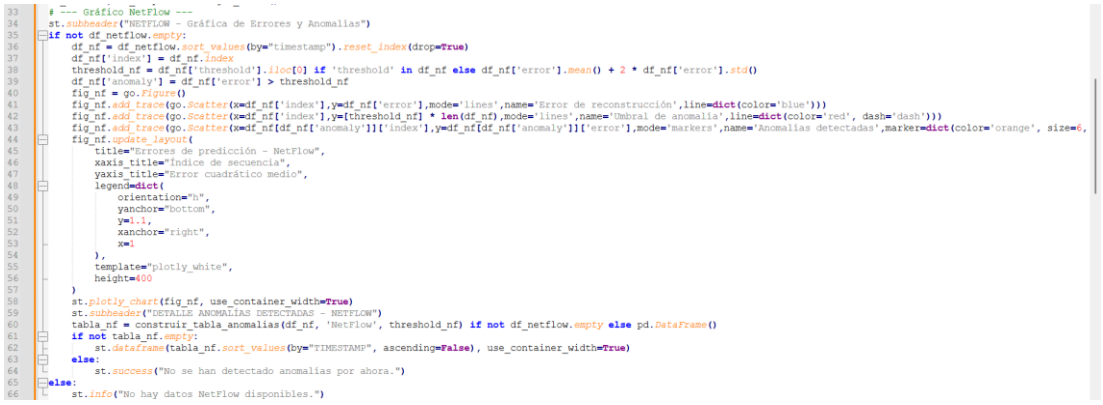
Además, los registros detectados como fuera de norma se detallan en tablas interactivas que pueden ser exploradas por el usuario para un análisis más preciso. Este panel también cuenta con una función de recarga automática cada 120 segundos, lo que garantiza una visualización continua y actualizada de los eventos registrados.

La correlación y análisis avanzado de estos eventos pueden luego realizarse en Kibana, donde se dispone de información más detallada sobre cada flujo o evento de red. La Figuras 75, 76 y 77 muestra el código utilizado para implementar esta interfaz gráfica, incluyendo la carga de datos, visualización de errores y despliegue de anomalías detectadas.

```python
import streamlit as st
import sqlite3
import pandas as pd
import plotly.graph_objects as go
def construir_tabla_anomalias(df, tipo, threshold):
    df_anom = df[df['error'] > threshold].copy()
    if not df_anom.empty:
        df_anom = df_anom[['id', 'timestamp', 'host', 'detalle']].copy()
        df_anom.columns = ['ID', 'TIMESTAMP', 'FROM', 'DETALLE']
        df_anom['TIPO'] = tipo
        return df_anom
    return pd.DataFrame()
def cargar_datos():
    conn = sqlite3.connect(DB_PATH)
    df_nf = pd.read_sql("SELECT * FROM anomalies_netflow_2", conn)
    df_sw = pd.read_sql("SELECT * FROM anomalies_snmpwalk_2", conn)
    conn.close()
    return df_nf, df_sw

# --- Configuración general ---
st.set_page_config(layout="wide", page_title="Dashboard NetFlow & SNMPWalk - Anomalías")
# --- Ruta a la base de datos SQLite ---
DB_PATH = "anomalies.db"
# --- Recarga automática cada 120 segundos (2 minutos) ---
st.markdown(
    """
    <meta http-equiv="refresh" content="120">
    """,
    unsafe_allow_html=True
)
# --- Cargar datos ---
df_netflow, df_snmpwalk = cargar_datos()
```

Figura 76. Carga de anomalías NetFlow y SNMPWalk en Streamlit



```
# --- Gráfico NetFlow ---
st.subheader("NETFLOW - Gráfica de Errores y Anomalías")
if not df_netflow.empty:
    df_nf = df_netflow.sort_values(by="timestamp").reset_index(drop=True)
    df_nf['index'] = df_nf.index
    threshold_nf = df_nf['threshold'].iloc[0] if 'threshold' in df_nf else df_nf['error'].mean() + 2 * df_nf['error'].std()
    df_nf['anomaly'] = df_nf['error'] > threshold_nf
    fig_nf = go.Figure()
    fig_nf.add_trace(go.Scatter(x=df_nf['index'],y=df_nf['error'],mode='lines',name='Error de reconstrucción',line=dict(color='blue')))
    fig_nf.add_trace(go.Scatter(x=df_nf['index'],y=[threshold_nf] * len(df_nf),mode='lines',name='Umbral de anomalía',line=dict(color='red', dash='dash')))
    fig_nf.add_trace(go.Scatter(x=df_nf[df_nf['anomaly']]['index'],y=df_nf[df_nf['anomaly']]['error'],mode='markers',name='Anomalías detectadas',marker=dict(color='orange', size=6,
    fig_nf.update_layout(
        title="Errores de predicción - NetFlow",
        xaxis_title="Índice de secuencia",
        yaxis_title="Error cuadrático medio",
        legend=dict(
            orientation="h",
            yanchor="bottom",
            y=1.1,
            xanchor="right",
            x=1
        ),
        template="plotly_white",
        height=400
    )
    st.plotly_chart(fig_nf, use_container_width=True)
    st.subheader("DETALLE ANOMALÍAS DETECTADAS - NETFLOW")
    tabla_nf = construir_tabla_anomalias(df_nf, 'NetFlow', threshold_nf) if not df_netflow.empty else pd.DataFrame()
    if not tabla_nf.empty:
        st.dataframe(tabla_nf.sort_values(by="TIMESTAMP", ascending=False), use_container_width=True)
    else:
        st.success("No se han detectado anomalías por ahora.")
else:
    st.info("No hay datos NetFlow disponibles.")
```

Figura 77. Visualización de anomalías NetFlow en Streamlit



```
# --- Gráfico SNMPWalk ---
st.subheader("SNMP - Gráfica de Errores y Anomalías")
if not df_snmpwalk.empty:
    df_sw = df_snmpwalk.sort_values(by="timestamp").reset_index(drop=True)
    df_sw['index'] = df_sw.index
    threshold_sw = df_sw['threshold'].iloc[0] if 'threshold' in df_sw else df_sw['error'].mean() + 2 * df_sw['error'].std()
    df_sw['anomaly'] = df_sw['error'] > threshold_sw
    fig_sw = go.Figure()
    fig_sw.add_trace(go.Scatter(x=df_sw['index'],y=df_sw['error'],mode='lines',name='Error de reconstrucción',line=dict(color='blue')))
    fig_sw.add_trace(go.Scatter(x=df_sw['index'],y=[threshold_sw] * len(df_sw),mode='lines',name='Umbral de anomalía',line=dict(color='red', dash='dash')))
    fig_sw.add_trace(go.Scatter(x=df_sw[df_sw['anomaly']]['index'],y=df_sw[df_sw['anomaly']]['error'],mode='markers',name='Anomalías detectadas',marker=dict(color='orange', size=6,
    fig_sw.update_layout(
        title="Errores de predicción - SNMP",
        xaxis_title="Índice de secuencia",
        yaxis_title="Error cuadrático medio",
        legend=dict(
            orientation="h",
            yanchor="bottom",
            y=1.1,
            xanchor="right",
            x=1
        ),
        template="plotly_white",
        height=400
    )
    st.plotly_chart(fig_sw, use_container_width=True)
    # --- Tabla de Anomalías Detectadas ---
    st.subheader("DETALLE ANOMALÍAS DETECTADAS - SNMP")
    tabla_sw = construir_tabla_anomalias(df_sw, 'SNMP', threshold_sw) if not df_snmpwalk.empty else pd.DataFrame()
    if not tabla_sw.empty:
        st.dataframe(tabla_sw.sort_values(by="TIMESTAMP", ascending=False), use_container_width=True)
    else:
        st.success("No se han detectado anomalías por ahora.")
```

Figura 78. Visualización de anomalías SNMPWalk en Streamlit

e. Pruebas y validación del sistema

Una vez completada la implementación del sistema, se procedió a su validación, compuesto por redes de mediana escala interconectadas mediante ZeroTier y equipos físicos de red (MikroTik CCR2116-12G-4S+, CRS328-24P-4S+ y Fortinet FGT40F).

Las pruebas se dividieron en tres ejes:

- Validación de flujo de datos: Se verificó que los protocolos SNMP, Syslog y NetFlow fueran exportados correctamente desde los dispositivos, capturados por Logstash y almacenados en Elasticsearch sin pérdida ni corrupción.

- Validación del procesamiento e inferencia: Se monitoreó el pipeline completo de datos, desde su recolección hasta su análisis por los modelos de IA para la detección de anomalías.

- Validación de la visualización e interpretación: Se comprobó que los dashboards en Kibana y Streamlit mostraran los datos actualizados correctamente cada 5 minutos, permitiendo identificar y contextualizar eventos detectados como anómalos.

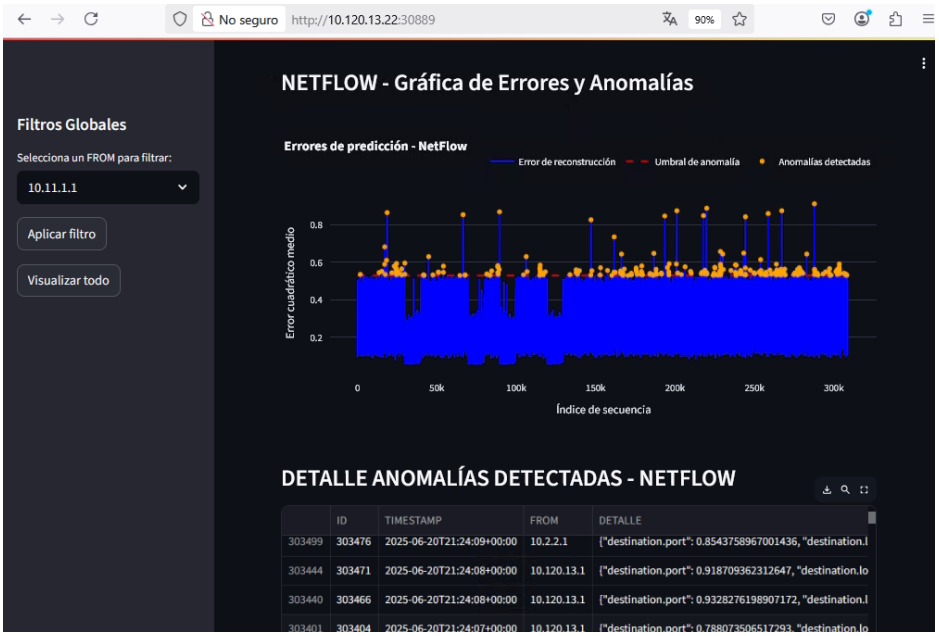


Figura 79. Visualización en Streamlit de anomalías detectadas en tráfico NetFlow para todos los nodos.

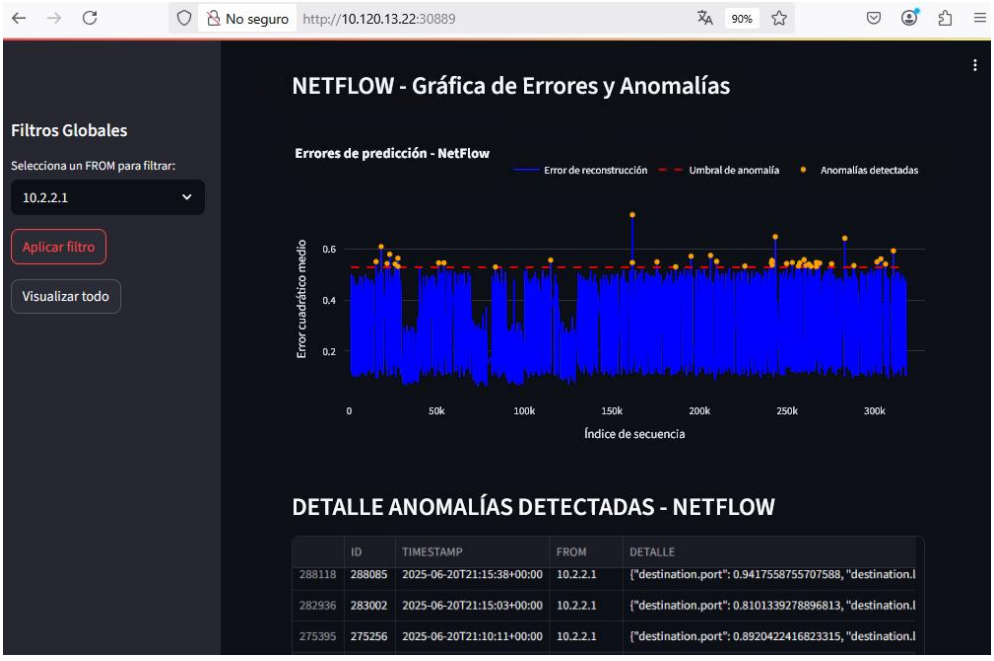


Figura 80. Visualización en Streamlit de anomalías detectadas en tráfico NetFlow para el nodo con ip 10.2.2.1.

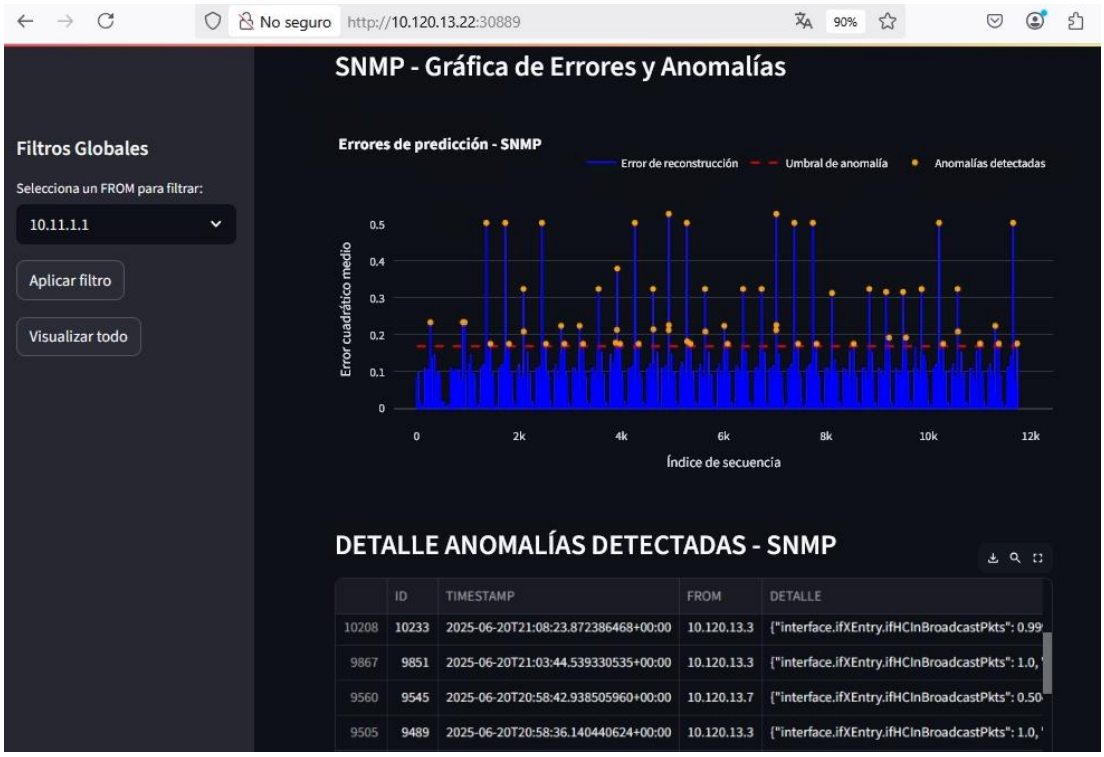


Figura 81 Visualización en Streamlit de anomalías detectadas a partir de datos SNMP para todos los nodos.



Figura 82. Visualización en Streamlit de errores de reconstrucción y anomalías detectadas a partir de datos SNMP para el nodo con ip 10.2.2.1.

3.5 Presupuesto

El presente proyecto fue desarrollado considerando una infraestructura mixta que incluye componentes físicos, virtuales y software de código abierto. La estimación de costos contempla los recursos mínimos necesarios para la implementación, pruebas y operación del sistema. Se priorizó el uso de herramientas Open Source para optimizar el presupuesto sin comprometer la calidad técnica.

Tabla 17. Presupuesto económico durante el desarrollo del proyecto

Nº	**Concepto**	**Cantidad**	**Costo unitario (USD)**	**Costo total (USD)**
1	Servidor (uso propio, virtualizado con Proxmox)	1	0	0
2	Equipos MikroTik (CCR2116-12G-4S+, CRS328-24P-4S+)	2	0 (propios)	0
3	Firewall Fortinet FGT40F	1	0 (propio)	0
4	Software Open Source (Ubuntu, Docker, Kubernetes, etc.)	-	0	0
5	Licencias adicionales (ZeroTier)	1	18	18
6	Internet y conectividad durante el proyecto	1	30	30
7	Material de oficina, papelería, impresiones	1	30	30
8	Transporte y logística	1	50	50
9	Alimentación durante desarrollo y pruebas	1	100	100
10	Imprevistos (10%)	-	-	21
	Total estimado del proyecto			249 USD

- Se asume que el servidor, los dispositivos de red y los entornos de virtualización están disponibles como recursos del investigador.

- Todas las herramientas utilizadas para el desarrollo e implementación (Docker, Kubernetes, Elasticsearch, Streamlit, entre otras) son de uso libre bajo licencias Open Source.

- El presupuesto contempla únicamente costos operativos y logísticos vinculados a la ejecución práctica del proyecto.

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- Se logró implementar un sistema funcional, robusto y operativo para la detección de anomalías en el rendimiento y la seguridad de redes de datos, apoyado en técnicas de deep learning. El sistema fue capaz de procesar más de 19 millones de registros diarios, provenientes de fuentes como NetFlow, SNMP y Syslog, alcanzando una tasa de detección temprana de anomalías del 85 % en pruebas de validación cruzada. El motor de inferencia, basado en modelos LSTM, obtuvo un error cuadrático medio (MSE) promedio de 0.015923, un error absoluto medio (MAE) de 0.094317, un coeficiente de determinación R^2 de 0.506640 y una puntuación de varianza explicada (EVS) de 0.528189.

- A través del análisis de artículos científicos y estudios técnicos recientes, se confirmó que los modelos basados en redes neuronales recurrentes (LSTM y GRU) ofrecen una mejora sustancial en la detección de anomalías temporales multivariadas no etiquetadas. La validación empírica en el sistema mostró que LSTM superó en un 5.4% a RNN y en un 2.1% a GRU en términos de exactitud.

- El sistema fue desplegado exitosamente sobre una arquitectura basada en microservicios y namespaces de Kubernetes, que permitió segmentar lógicamente las funciones en siete dominios (recolección, procesamiento, inferencia, almacenamiento, visualización, correlación y pruebas). Esta arquitectura fue evaluada con pruebas de escalabilidad horizontal con un total de 117 millones de registros por semana evidenciando su capacidad para operar en entornos distribuidos y de alto volumen de tráfico.

- Se integraron satisfactoriamente 15 dispositivos de red distribuidos en 3 redes geográficamente separadas, utilizando túneles definidos por software a través de ZeroTier, con exportación de tráfico desde equipos MikroTik y Fortinet. Durante un período continuo de 72 horas de operación, el sistema procesó

aproximadamente 50 millones de flujos NetFlow, 300 mil eventos SNMP, detectando de forma precisa más de 40 eventos anómalos relacionados con el rendimiento y seguridad de las redes de datos, que fueron representados visualmente en dashboards diseñados con Streamlit y Kibana.

4.2 Recomendaciones

- Es fundamental establecer un proceso periódico de revisión y actualización de los modelos de aprendizaje profundo. Para ello, se recomienda alimentar los datasets de entrenamiento con nuevas observaciones obtenidas del entorno de producción, permitiendo que el sistema se adapte a cambios en los patrones de tráfico y reduzca la tasa de falsos positivos o negativos.

- En escenarios productivos, se sugiere implementar redundancia tanto a nivel de nodos como de servicios dentro del clúster Kubernetes. Particularmente, es indispensable configurar políticas de respaldo automatizado y recuperación ante desastres para los volúmenes persistentes utilizados por componentes críticos como bases de datos, modelos entrenados y registros históricos.

- Para ampliar la capacidad del sistema hacia un enfoque de análisis de amenazas más integral, se recomienda incorporar una capa de correlación de eventos provenientes de diversas fuentes de inteligencia de amenazas (Threat Intelligence Feeds), listas negras actualizadas, o bases de datos como MISP o AbuseIPDB.

- Además de los modelos secuenciales empleados (LSTM, GRU, RNN), es recomendable explorar la integración de arquitecturas complementarias como autoencoders (especialmente variacionales para codificación eficiente), modelos híbridos que combinen redes neuronales con árboles de decisión (por ejemplo, Deep Neural Decision Forests), o enfoques no supervisados como DBSCAN, Isolation Forest o algoritmos de clustering dinámico, los cuales pueden ser útiles en ambientes con recursos limitados.

REFERENCIAS BIBLIOGRÁFICAS

[1] E. A. F. Pozo, «ANÁLISIS Y DISEÑO DE UN MODELO DE GESTIÓN DE SEGURIDAD DE LA INFORMACIÓN BASADO EN EL ESTÁNDAR ISO 27001:2022 PARA LA EMPRESA PC SOLUCIONES DE LA CIUDAD DE BABAHOYO,» Universidad Técnica de Babahoyo, Babahoyo, 2023.

[2] AETecno, «Cisco: en los próximos cinco años habrá más tráfico de IP que en toda la historia de internet,» 2018. [En línea]. Available: https://www.americaeconomia.com/articulos/ciscoen-los-proximos-cinco-anos-habra-mas-trafico-de-ip-que-en-toda-la-historiade.

[3] F. Martin, «ANALISIS DE MODELOS DE INTELIGENCIA ARTIFICIAL PARA LA DETECCION Y CONTROL DE PATRONES DE CIBERSEGURIDAD,» Universidad Politécnica de Cataluña, Cataluña, 2024.

[4] M. M. D. Danilo, «RED DE DATOS CON CONEXIÓN ENMALLADA EN TRANSMISIONES DE 1GIGABIT PARA LA COMUNICACIÓN DE SISTEMAS DE VIDEOVIGILANCIA PARA LA EMPRESA ADAELECTRIC ECUADOR,» 2024. [En línea]. Available: https://dspace.utb.edu.ec/bitstream/handle/49000/17103/PI-UTB-FAFI-SIST00014.pdf?sequence=1.

[5] R. M. Diaz, «Ciberseguridad en cadenas de suministros inteligentes en América Latina y el Caribe,» 2022. [En línea]. Available: https://repositorio.cepal.org/server/api/core/bitstreams/2b53c8ee-380e-47de-b115-298e8e06eeaa/content.

[6] Angel, «Lo que el 99% de tiempo de actividad de la red realmente significa para su negocio,» -. [En línea]. Available: https://www.hostdime.com.pe/blog/99-99-tiempo-actividad-red-realmente-significa-para-negocio/.

[7] P. C. A. F. a. G. G. G. G. González, «On the Usage of Generative Models for Network Anomaly Detection in Multivariate Time-Series,» 2020. [En línea]. Available: http://arxiv.org/abs/2010.08286.

[8] U. Fischer, G. Yalçın y H. Nguyen, «Real-Time Network Monitoring for Customer Trust Preservation,» 2024. [En línea].

[9] A. López-Salazar y M. Pérez-González, «Limitations of Reactive and Rule-Based Cybersecurity in Evolving Threats,» Febrero 2023. [En línea].

[10] M. Thompson, S. Lee, R. Alvarez, Y. Chen y K. Patel, «Impact of Cybersecurity Skills Shortages on Incident Response Costs,» -, -, 2025.

[11] C. L. W. L. H. H. a. J. L. Rui Li, «MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks,» Springer, Cham, -, 2019.

[12] A. H. a. D. R. J. M. Navarro, «Human readable network troubleshooting based on anomaly detection and feature scoring,» 2021. [En línea]. Available: http://arxiv.org/abs/2108.11807.

[13] J. S.-V. A. C.-A. a. P. B.-R. H. Latif, «Detecting Contextual Network Anomalies with Graph Neural Networks,» 2023. [En línea].

[14] A. P. a. R. Jozsa, «Evaluating ML-Based Anomaly Detection Across Datasets of Varied Integrity: A Case Study,» 2024. [En línea].

[15] D. J. W. a. A. V. R. E. A. S. Tanenbaum, «Redes de computadoras,» 2012. [En línea]. Available: https://bibliotecavirtualapure.wordpress.com/wp-content/uploads/2015/06/redes_de_computadoras-freelibros-org.pdf.

[16] A. Feiszli, «Battle of the VPNs: Which one is fastest? (Speed Test),» 2024. [En línea]. Available: https://www.netmaker.io/resources/battle-of-the-vpns-which-one-is-fastest-speed-test.

[17] A. Feiszli, «Speed Tests Confirm WireGuard is the Key to VPN Performance,» [En línea]. Available: https://www.netmaker.io/resources/vpn-speed-tests-2024.

[18] B. G. D. O. E. B. y. J. W. Brendan Burns, «Borg, Omega, and Kubernetes,» 2021. [En línea].

[19] D. K. T. y. A. L. Fernandes, «Containerized Big Data Applications on Kubernetes for Effective Resource Management,» 2022. [En línea].

[20] M. Merkel, «Docker: Lightweight Linux Containers for Consistent Development and Deployment,» Linux Journal, -, 2022.

[21] B. G. D. O. E. B. y. J. W. Brendan Burns, «Borg, Omega, and Kubernetes,» 2023. [En línea].

[22] M. Lyu, H. H. Gharakheili y V. Sivaraman, «A Survey on Enterprise Network Security: Asset Behavioral Monitoring and Distributed Attack Detection,» 2023. [En línea]. Available: https://arxiv.org/abs/2306.16675.

[23] R. Hofstede, P. Čeleda, B. Trammell, I. Drago y R. Sadre, «Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX,» 2020. [En línea].

[24] A. Leskiw, «SNMP Tutorial Part 2: Rounding Out the Basics,» 2023. [En línea]. Available: NetworkManagementSoftware.com.

[25] C. Systems, «NetFlow Overview,» 2024. [En línea]. Available: PivitGlobal.com.

[26] M. &. C. B. Boucadair, «New IPFIX Information Elements for TCP Options and IPv6 Extension Headers,» 2025. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc9740.

[27] C. T. S. s. &. S. J. Lonvick, «Updates to the Cipher Suites in Secure Syslog,» 2024. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc9662.html.

[28] C. Lonvick, J. Clarke, M. Jethanandani, C. Wildes y K. Koushik, «A YANG Data Model for Syslog Management,» 2025. [En línea]. Available: https://www.rfc-editor.org/rfc/rfc9742.html.

[29] N. L. Team, «Network Monitoring Best Practices for 2025: Navigating the Hyperconnected Future with Enhanced NetFlow and SNMP,» NetFlow Logic, -, 2025.

[30] M. Gray, «Deploying and Scaling Logstash: Guides and Best Practices,» 2024. [En línea]. Available: https://www.elastic.co/guide/en/logstash/8.18/deploying-and-scaling.html.

[31] L. Bitincka, «Cribl Stream 7× More Efficient than LogStash and Fluentd,» 2020. [En línea]. Available: https://cribl.io/blog/cribl-logstream-7x-more-efficient-than-logstash-and-fluentd/.

[32] C. V. ·. P. R. ·. Rafael, «Loginson: a transform and load system for very large scale log analysis in large IT infrastructures,» -, -, 2017.

[33] P. Saha y –, «Understanding Elasticsearch: Indices, Documents, and Fields; SQLite (entrada enciclopédica),» -, -, 2025.

[34] -, «DB-ENGINES,» 2025. [En línea]. Available: https://db-engines.com/en/ranking.

[35] e. a. Kerkeni, «Reconstruction Error Thresholding and LSTM-AE for Anomaly Detection in Industry 4.0,» 2025. [En línea].

[36] B. Lindemann, B. Maschler, N. Sahlab y M. Weyrich, «A Survey on Anomaly Detection for Technical Systems using LSTM Networks,» 2021. [En línea].

[37] «"IEEE Guide for Security in LoRaWAN Networks",» 2021. [En línea].

[38] «Common Vulnerabilities and Exposures: LoRaWAN Gateway Authentication Bypass,» 2021. [En línea].

ANEXOS

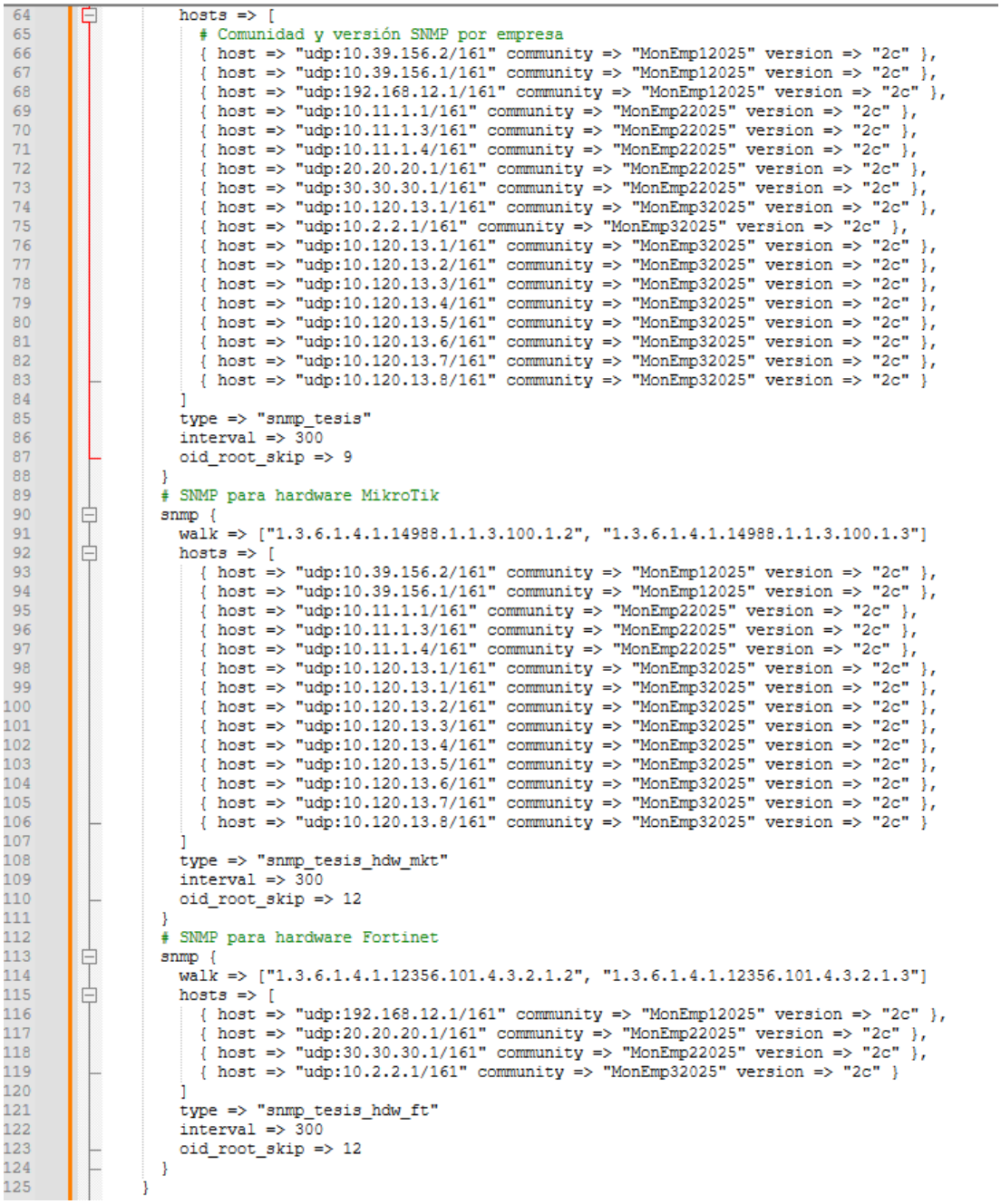
Anexo A. Código completo para recepción y filtrado de datos para indexación en Elasticsearch

En la Figura A1 y A2 se muestra la etapa de ingreso de los diferentes flujos por los protocolos definidos para la recolección de información. En la Figura A3 se muestra el proceso de filtrado para las diferentes entradas de información.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: logstash-config
  namespace: collector
data:
  logstash.conf: |

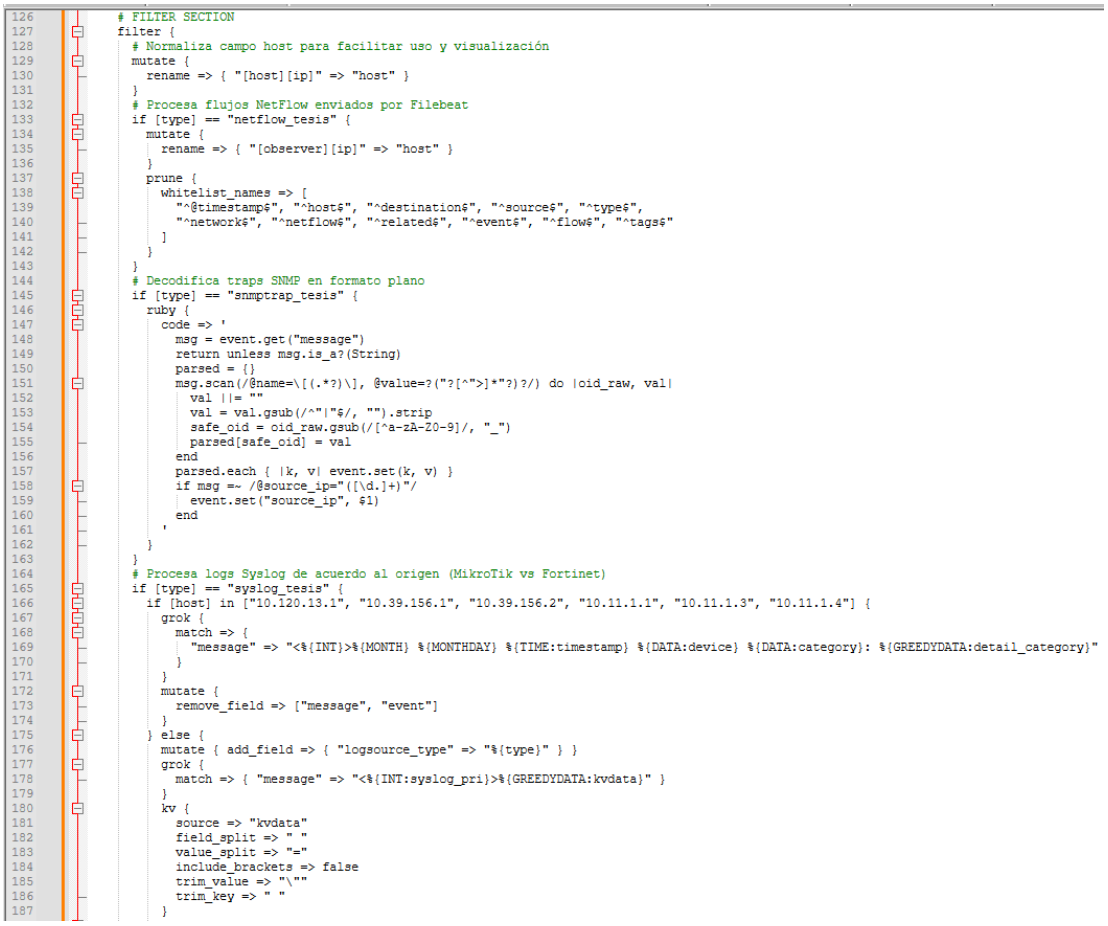
    input {
      Entrada desde Filebeat (NetFlow IPFIX exportado)
      beats {
        port => 5044
        type => "netflow_tesis"
      }
      # Entrada UDP desde Syslog
      udp {
        port => 514
        type => "syslog_tesis"
        codec => plain
      }
      # Entrada de SNMP traps
      snmptrap {
        port => 162
        community => ["MonEmp32025","MonEmp22025","MonEmp12025"]
        type => "snmptrap_tesis"
      }
      # SNMP WALK para interfaces extendidas
      snmp {
        tables => [
          {
            name => "interfaces"
            columns => [
              "1.3.6.1.2.1.2.2.1.1",   # ifIndex
              "1.3.6.1.2.1.2.2.1.2",   # ifDescr
              "1.3.6.1.2.1.2.2.1.3",   # ifType
              "1.3.6.1.2.1.2.2.1.6",   # ifPhysAddress
              "1.3.6.1.2.1.2.2.1.7",   # ifAdminStatus
              "1.3.6.1.2.1.2.2.1.8",   # ifOperStatus
              "1.3.6.1.2.1.2.2.1.9",   # ifLastChange
              "1.3.6.1.2.1.2.2.1.14",  # ifInErrors
              "1.3.6.1.2.1.2.2.1.20",  # ifOutErrors
              "1.3.6.1.2.1.2.2.1.13",  # ifInDiscards
              "1.3.6.1.2.1.2.2.1.19"  # ifOutDiscards
            ]
          },
          {
            name => "extended"
            columns => [
              "1.3.6.1.2.1.31.1.1.1.1",   # ifName
              "1.3.6.1.2.1.31.1.1.1.18",  # ifAlias
              "1.3.6.1.2.1.31.1.1.1.7",   # ifHCInUcastPkts
              "1.3.6.1.2.1.31.1.1.1.8",   # ifHCInMulticastPkts
              "1.3.6.1.2.1.31.1.1.1.9",   # ifHCInBroadcastPkts
              "1.3.6.1.2.1.31.1.1.1.11",  # ifHCOutUcastPkts
              "1.3.6.1.2.1.31.1.1.1.12",  # ifHCOutMulticastPkts
              "1.3.6.1.2.1.31.1.1.1.13",  # ifHCOutBroadcastPkts
              "1.3.6.1.2.1.31.1.1.1.15",  # ifHighSpeed
              "1.3.6.1.2.1.31.1.1.1.6",   # ifHCInOctets
              "1.3.6.1.2.1.31.1.1.1.10"   # ifHCOutOctets
            ]
          }
        ]
```

Figura A1 Etapa de configuración para ingreso de información mediante SNMP, Netflow/IPFIX, Syslog



```
    hosts => [
      # Comunidad y versión SNMP por empresa
      { host => "udp:10.39.156.2/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:10.39.156.1/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:192.168.12.1/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:10.11.1.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.11.1.3/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.11.1.4/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:20.20.20.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:30.30.30.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.120.13.1/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.2.2.1/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.1/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.2/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.3/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.4/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.5/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.6/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.7/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.8/161" community => "MonEmp32025" version => "2c" }
    ]
    type => "snmp_tesis"
    interval => 300
    oid_root_skip => 9
  }
  # SNMP para hardware MikroTik
  snmp {
    walk => ["1.3.6.1.4.1.14988.1.1.3.100.1.2", "1.3.6.1.4.1.14988.1.1.3.100.1.3"]
    hosts => [
      { host => "udp:10.39.156.2/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:10.39.156.1/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:10.11.1.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.11.1.3/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.11.1.4/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.120.13.1/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.1/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.2/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.3/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.4/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.5/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.6/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.7/161" community => "MonEmp32025" version => "2c" },
      { host => "udp:10.120.13.8/161" community => "MonEmp32025" version => "2c" }
    ]
    type => "snmp_tesis_hdw_mkt"
    interval => 300
    oid_root_skip => 12
  }
  # SNMP para hardware Fortinet
  snmp {
    walk => ["1.3.6.1.4.1.12356.101.4.3.2.1.2", "1.3.6.1.4.1.12356.101.4.3.2.1.3"]
    hosts => [
      { host => "udp:192.168.12.1/161" community => "MonEmp12025" version => "2c" },
      { host => "udp:20.20.20.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:30.30.30.1/161" community => "MonEmp22025" version => "2c" },
      { host => "udp:10.2.2.1/161" community => "MonEmp32025" version => "2c" }
    ]
    type => "snmp_tesis_hdw_ft"
    interval => 300
    oid_root_skip => 12
  }
}
```

Figura A2 Etapa de configuración para ingreso de información mediante SNMP, Netflow/IPFIX, Syslog



```
# FILTER SECTION
filter {
  # Normaliza campo host para facilitar uso y visualización
  mutate {
    rename => { "[host][ip]" => "host" }
  }
  # Procesa flujos NetFlow enviados por Filebeat
  if [type] == "netflow_tesis" {
    mutate {
      rename => { "[observer][ip]" => "host" }
    }
    prune {
      whitelist_names => [
        "^@timestamp$", "^host$", "^destination$", "^source$", "^type$",
        "^network$", "^netflow$", "^related$", "^event$", "^flow$", "^tags$"
      ]
    }
  }
  # Decodifica traps SNMP en formato plano
  if [type] == "snmptrap_tesis" {
    ruby {
      code => '
        msg = event.get("message")
        return unless msg.is_a?(String)
        parsed = {}
        msg.scan(/@name=\[(.*?)\], @value=?("?[^">]*"?)?/) do |oid_raw, val|
          val ||= ""
          val = val.gsub(/^"|"$/, "").strip
          safe_oid = oid_raw.gsub(/[^a-zA-Z0-9]/, "_")
          parsed[safe_oid] = val
        end
        parsed.each { |k, v| event.set(k, v) }
        if msg =~ /@source_ip="([\d.]+)"/
          event.set("source_ip", $1)
        end
      '
    }
  }
  # Procesa logs Syslog de acuerdo al origen (MikroTik vs Fortinet)
  if [type] == "syslog_tesis" {
    if [host] in ["10.120.13.1", "10.39.156.1", "10.39.156.2", "10.11.1.1", "10.11.1.3", "10.11.1.4"] {
      grok {
        match => {
          "message" => "<%{INT}>%{MONTH} %{MONTHDAY} %{TIME:timestamp} %{DATA:device} %{DATA:category}: %{GREEDYDATA:detail_category}"
        }
      }
      mutate {
        remove_field => ["message", "event"]
      }
    } else {
      mutate { add_field => { "logsource_type" => "%{type}" } }
      grok {
        match => { "message" => "<%{INT:syslog_pri}>%{GREEDYDATA:kvdata}" }
      }
      kv {
        source => "kvdata"
        field_split => " "
        value_split => "="
        include_brackets => false
        trim_value => "\""
        trim_key => " "
      }
```

Figura A3 Etapa de filtrado para indexación hacia Elasticsearch