UNIVERSIDAD
DE ANTIOQUIA

Estrategia para la implementación de modelos de aprendizaje automático (machine learning) usando arquitecturas basadas en microservicios en el contexto de la seguridad en ciudades inteligentes

David Santiago Guerrero Martínez

Proyecto de investigación presentado para optar al título de Ingeniero Electrónico.

Tutor

Luis German García Morales, (Ph.D.)

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería Electrónica
Medellín, Antioquia, Colombia
2023

Cita	D. S. Guerrero Martínez
Referencia	
Estilo IEEE (2020)	[1] D. S Guerrero Martinez, “Estrategia para la implementación de modelos de aprendizaje automático (machine learning) usando arquitecturas basadas en microservicios en el contexto de la seguridad en ciudades inteligentes”, Trabajo de grado profesional, Ingeniería Electrónica, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2023.

Grupo de Investigación Sistemas Embebidos e Inteligencia Computacional (SISTEMIC)

Seleccione centro de investigación UdeA (A-Z)

Agradecimientos

Primero y ante todo, deseo expresar mi más profundo agradecimiento a mis padres, quienes con su amor, paciencia y apoyo incondicional, me han brindado la fortaleza para continuar y superar los desafíos que se presentaban en mi camino académico.

A Juan y Emanuel, mis hermanos, su compañía y solidaridad han sido esenciales en este proceso. Su aliento y fe en mí me inspiraron a perseguir mis ambiciones con determinación.

Mi gratitud a los profesores de excelencia que marcaron mi trayectoria y formaron en mí una pasión por mi campo profesional. En especial, deseo reconocer la invaluable orientación y apoyo de mi tutor, el Profesor Germán García, y del Profesor José Aedo, quienes con su experiencia y sabiduría, me guiaron y moldearon en cada etapa de este trabajo.

A Camila Martínez, mi novia, cuyo apoyo y amor constante ha sido un pilar en los momentos más difíciles y desafiantes. Su presencia en mi vida ha sido un motor que me impulsó a alcanzar metas que alguna vez parecieron inalcanzables.

Finalmente, a todos mis amigos, conocidos y familiares que han estado allí, en cada paso y en cada logro, les extiendo mi sincero agradecimiento. Su apoyo y aliento han sido fundamentales para alcanzar este hito en mi vida académica.

Gracias a todos por ser parte de este viaje.

Contenido

Lista de Figuras

Lista de Tablas

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IoT	Internet of Things
IDS	Intrusion Detection System
HIDS	Host-based Intrusion Detection System
NIDS	Network-based Intrusion Detection System
SVM	Support Vector Machine
API	Application Program Interface
DoS	Denial of Service
DDoS	Distributed Denial of Service
JWT	JSON Web Token
JSON	JavaScript Object Notation
SSO	Single Sign-On
HTTP	Hypertext Transfer Protocol
ML	Machine Learning

RESUMEN

En la actualidad, las ciudades inteligentes representan una realidad inminente que ofrecen soluciones para mejorar la calidad de vida mediante nuevas tecnologías. Sin embargo, con la creciente dependencia tecnológica, emergen desafíos en seguridad informática tales como la protección de datos ante ataques cibernéticos, la gestión de la privacidad de la información de los usuarios, el manejo eficiente de elevados volúmenes de peticiones, además de afrontar necesidades de escalamiento, resiliencia y confiabilidad en los sistemas. Hoy en día, métodos basados en modelos de aprendizaje automático (ML) son empleados para resolver problemas de seguridad informática presentes en los sistemas de las ciudades inteligentes. Sin embargo, para el manejo eficiente del creciente número de peticiones, además de atender a las necesidades de escalamiento de los sistemas, se hace necesario establecer estrategias y herramientas adecuadas que faciliten la evaluación y desarrollo de modelos de aprendizaje automático.

Este trabajo tiene como objetivo principal desarrollar una estrategia para la implementación y evaluación de modelos de aprendizaje automático enfocados en seguridad informática dentro del contexto de ciudades inteligentes. Para alcanzar este propósito, se planteó una arquitectura basada en microservicios. La iniciativa de este estudio responde a la necesidad de disponer de plataformas escalables y versátiles, que faciliten el despliegue efectivo de modelos ML.

Con la arquitectura planteada en este trabajo, se propuso también el desarrollo de una infraestructura basada en microservicios, con dos servicios fundamentales: uno dedicado al preprocesamiento de datos y otro dedicado a brindar servicios de predicción. Estos microservicios facilitan la ejecución, almacenamiento y evaluación de modelos ML. Como gestor de mensajes, se propuso emplear el software Kafka para brindar una comunicación asincrónica eficiente y efectiva entre los microservicios involucrados. Adicionalmente, la implementación de un Gateway y un balanceador de carga se realizó con la visión de optimizar el manejo integral del tráfico y las solicitudes, asegurando un despliegue de modelos ML en el contexto de ciudades inteligentes que es tanto escalable como versátil.

Como resultado de este trabajo, se desarrolló una infraestructura especializada en la implementación de microservicios empleando Java y Python, así como una base de datos de tráfico de red. Esta última no solo cumple un rol fundamental al alimentar los procesos de predicción, sino que también respalda el caso de estudio específicamente seleccionado para validar la funcionalidad del mecanismo desarrollado. Este conjunto de herramientas y datos no solo facilita el despliegue y la evaluación de modelos de ML, sino que también simplifica la escalabilidad de las arquitecturas involucradas. De esta manera, el presente trabajo contribuye con la implementación de modelos de ML en ciudades inteligentes mediante arquitecturas fáciles de escalar y mantener.

Palabras clave **— Microservicios, aprendizaje automático, seguridad, ciudades inteligentes, Kafka, Gateway, base de datos de tráfico de red.**

ABSTRACT

Nowadays, smart cities represent a reality that offers solutions to enhance the quality of life through new technologies. However, with the rise in technological dependence, challenges in cybersecurity also emerge. These challenges include data protection from cyberattacks, management of user information privacy, efficient handling of high volumes of requests, and the need for scalability, resilience, and system reliability. Today, methods based on machine learning (ML) models address cybersecurity problems in smart city systems. However, to manage the growing number of requests effectively and meet system scalability needs, it is essential to have appropriate strategies and tools that facilitate the evaluation and development of ML models.

This study aims to develop a strategy for the implementation and evaluation of ML models with the focus on cybersecurity within the context of smart cities. To achieve this goal, an architecture based on microservices has been proposed. The initiative of this study stems from the need of having platforms that are scalable and versatile, facilitating the effective deployment of ML models.

With the architecture presented in this work, the development of an infrastructure based on microservices has also been proposed. This infrastructure includes two main services: one dedicated to data preprocessing and another to provide prediction services. These microservices allow for the execution, storage, and evaluation of ML models. To achieve efficient and effective asynchronous communication between the microservices, the software Kafka software has been employed. Additionally, this work incorporated the implementation of a Gateway and a load balancer with the purpose of improving the overall management of traffic and requests. This ensures the deployment of ML models in a smart city that is both scalable and versatile.

As an outcome of this work, an infrastructure has been developed that specializes in the implementation of microservices using Java and Python. This infrastructure is complemented by a network traffic database, which plays an important role in feeding prediction processes and also supports the chosen case study to validate the functionality of the developed mechanism. This collection of tools and data not only simplifies the deployment and evaluation of ML models, but also enhances the scalability of the associated architectures. Consequently, this research contributes to the effective deployment of ML models in smart cities via architectures that are straightforward to scale and maintain.

Keywords — **Microservices, machine learning, security, smart cities, Kafka, Gateway, network traffic database.**

1 INTRODUCCIÓN

En el contexto emergente y dinámico de las ciudades inteligentes, las infraestructuras tecnológicas avanzadas y las soluciones de seguridad robustas son imperativas. Las ciudades inteligentes representan un epicentro donde la tecnología y la infraestructura se entrelazan estrechamente para ofrecer una calidad de vida mejorada a sus habitantes. En este entorno, la seguridad de la información se destaca como una preocupación primordial que requiere atención y soluciones ingeniosas.

Es en este escenario donde los modelos de ML emergen como herramientas prometedoras, ofreciendo soluciones para solventar algunos de los retos de seguridad informática. Sin embargo, implementar efectivamente estos modelos en dispositivos de IoT distribuidos a lo largo de extensas áreas metropolitanas no está exento de desafíos.

Una observación inicial del rendimiento de sistemas monolíticos revela que, aunque pueden procesar con eficacia una serie de peticiones simples, hay dudas significativas sobre su capacidad para manejar demandas más complejas y voluminosas inherentes a un entorno urbano inteligente. Además, se observó que, aunque los sistemas monolíticos pueden, en algunos casos, superar a otras arquitecturas, esto ocurre solo con un número muy bajo de peticiones, que no es representativo de las demandas de una ciudad inteligente real.

Por lo tanto, en este trabajo de investigación, se aborda la falta de plataformas que sean escalables y versátiles, diseñadas específicamente para facilitar el despliegue efectivo de modelos de ML en entornos de ciudades inteligentes. La preocupación no es solo la ausencia evidente de tales plataformas, sino también la necesidad de que estas soluciones sean intuitivas, accesibles, y replicables.

Para abordar estas preocupaciones, en este estudio, se ha conceptualizado y ejecutado una estrategia para la implementación y evaluación de modelos de ML, con un enfoque particular en la seguridad informática en ciudades inteligentes. A través de una arquitectura basada en microservicios, este trabajo presenta un marco de técnicas y prácticas de programación modernas y eficientes, además que también ofrece una ruta para la mejora continua de modelos de ML en el dominio de la seguridad informática. Además, se realizó una selección y evaluación de métricas pertinentes que permiten la mejora de estos modelos en el ámbito de la seguridad de la información.

Con la arquitectura propuesta, se desarrolló un mecanismo que permite la ejecución, almacenamiento y cálculo de métricas de modelos de aprendizaje automático. Este mecanismo se construyó empleando lenguajes de programación tales como Python y Java, así como otras herramientas

tecnológicas, logrando la creación de una infraestructura eficiente y adaptable a las necesidades de las ciudades inteligentes.

En este documento, se proporciona una visión detallada de cada fase de la investigación, desde la identificación del problema hasta la propuesta de soluciones y su respectiva evaluación, todo ello con el propósito de contribuir significativamente al desarrollo y fortalecimiento de la seguridad informática en el entorno de las ciudades inteligentes. Este documento se ha organizado de la siguiente manera:

- **Sección 2 - Planteamiento del Problema:** Esta sección ofrece una visión general del problema a tratar, introduciendo la pregunta de investigación y la hipótesis de trabajo que guiará el desarrollo del estudio.

- **Sección 3 - Objetivos:** Se presentan el objetivo general y los objetivos específicos del estudio, los cuales orientan y delimitan el alcance de la investigación.

- **Sección 4 - Marco Teórico:** En esta parte se ofrece un marco teórico que sustenta el estudio, proporcionando las bases conceptuales y teóricas necesarias para el desarrollo de la investigación.

- **Sección 5 - Metodología:** Se detalla la metodología empleada en el estudio, dividida en cuatro fases cruciales: 1) Planteamiento y Diseño, 2) Especificación, 3) Desarrollo y 4) Despliegue.

- **Sección 6 - Resultados y Análisis:** Esta sección presenta y analiza los resultados obtenidos durante la investigación, ofreciendo un análisis reflexivo sobre los hallazgos del estudio.

- **Sección 7 - Conclusiones:** Finalmente, se ofrecen las conclusiones derivadas del trabajo realizado, proporcionando una reflexión final sobre los logros y limitaciones del estudio, así como posibles mejoras.

2 PLANTEAMIENTO DEL PROBLEMA

2.1. Macro del problema

En la era de la digitalización, las ciudades inteligentes están emergiendo como soluciones sostenibles para abordar desafíos urbanos mediante el uso de tecnologías de información y comunicación[1]. Uno de los principales desafíos en el contexto de las ciudades inteligentes es la seguridad informática, especialmente dado el gran uso de datos sensibles de los ciudadanos.

Los modelos de aprendizaje automático presentan una opción prometedora para mejorar la predicción y detección de eventos de seguridad informática. No obstante, la falta de plataformas especializadas, escalables, y amigables para el usuario, destinadas al despliegue de estos modelos en el ámbito de las ciudades inteligentes, representa un obstáculo importante. Este vacío tecnológico actúa como un cuello de botella para la implementación y expansión efectivas de soluciones inteligentes en entornos urbanos.

2.2. Pregunta de investigación

¿Qué infraestructura tecnológica es necesaria para desarrollar una plataforma que no solo facilite la implementación y evaluación de modelos de aprendizaje automático en procesos de predicción de seguridad informática, sino que también sea escalable, fácil de usar y desplegar, superando las limitaciones de arquitecturas monolíticas?

2.3. Hipótesis de trabajo

Es viable desarrollar una infraestructura basada en microservicios que simplifique la implementación y evaluación de modelos de aprendizaje automático para la seguridad en ciudades inteligentes. Esta infraestructura, que se desarrolla mediante lenguajes de programación como Python y Java, será escalable, fácil de utilizar, instalar y replicar, optimizando la eficacia de las soluciones

de seguridad tradicionales y adaptándose flexiblemente a elevados números de peticiones y otros requisitos de operación en tiempo real.

3 OBJETIVOS

3.1. Objetivo general

Establecer una estrategia para la implementación y evaluación de modelos de aprendizaje automático en procesos de predicción, en el contexto de la seguridad en ciudades inteligentes, haciendo uso de arquitecturas basadas en microservicios y programación en Python y Java.

3.2. Objetivos específicos

1. Establecer una arquitectura basada en microservicios para la implementación de los procesos de predicción de los modelos de aprendizaje automático, empleando buenas prácticas de programación.

2. Seleccionar un conjunto de métricas que permitan la evaluación de los modelos de aprendizaje automático, en el contexto de la seguridad en ciudades inteligentes.

3. Desarrollar un mecanismo basado en la arquitectura y métricas definidas, que permita la ejecución, el almacenamiento y el cálculo de métricas de los modelos de aprendizaje automático, empleando los lenguajes Python y Java, bases de datos no relacionales, sistema de mensajería de publicación-suscripción y la interfaz API REST.

4. Seleccionar, implementar y evaluar un par de modelos de inteligencia computacional como caso de estudio, para validar la funcionalidad del mecanismo desarrollado.

4 MARCO TEÓRICO

La base de esta investigación se fundamenta en varios paradigmas teóricos y prácticos que han surgido durante la última década. Es notable que los conceptos de microservicios, preprocesamiento de datos, APIs REST y su orquestación forman la base para el desarrollo e implementación de modelos de ML. Este capítulo busca arrojar luz sobre estos dominios, otorgando credibilidad y contexto a la metodología.

La implementación de un sistema basado en microservicios para procesos predictivos de modelos de aprendizaje automático requiere una comprensión integral de los principios fundamentales, las metodologías y las tecnologías empleadas. Este marco teórico profundiza en la evolución, los conceptos y los matices de cada componente, aclarando su importancia en el panorama arquitectónico del software moderno.

El estilo arquitectónico de microservicios es una manera particular de diseñar aplicaciones de software como conjuntos de servicios independientemente desplegables. Este enfoque modular descompone sistemas complejos en módulos fácilmente manejables e intercambiables [2].

4.1. Microservicios

Como se ilustra en la Figura **4-1**, la arquitectura basada en microservicios ha revolucionado la forma en que diseñamos y desarrollamos aplicaciones de software modernas. Los microservicios consisten en pequeños servicios que se despliegan y operan de manera independiente, permitiendo una mayor modularidad, escalabilidad e independencia en comparación con las tradicionales arquitecturas monolíticas [3].

Cada nodo dentro de la arquitectura, como el Gateway, servicios individuales, bases de datos asociadas y otros componentes, se ubican y se comunican de manera estratégica para optimizar el flujo de datos y operaciones. A diferencia de los sistemas monolíticos donde una falla puede comprometer todo el sistema, los microservicios ofrecen un grado de aislamiento, mejorando la resiliencia del sistema en su conjunto.

Además, al organizar los microservicios en torno a capacidades empresariales, cada servicio se centra en una única responsabilidad o funcionalidad. Esto no solo facilita la gestión y manteni-

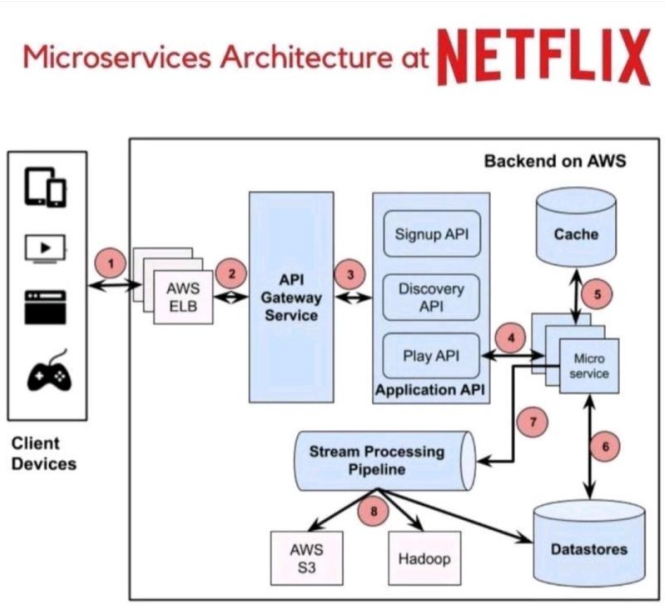

Figura 4-1: Ejemplo de una arquitectura basada en microservicios.

miento, sino que también optimiza la escalabilidad, ya que cada microservicio puede escalarse de manera independiente según la demanda [4].

4.1.1. Microservicios en ML

Integrar modelos de ML en una arquitectura de microservicios introduce desafíos y oportunidades únicas. Los desafíos incluyen la gestión y versión de modelos, la comunicación asíncrona y el preprocesamiento de datos. No obstante, al adoptar un enfoque basado en microservicios, estas tareas se pueden desacoplar de la lógica principal de la aplicación, favoreciendo una escalabilidad sostenida, una mayor resiliencia y facilidad de mantenimiento [5].

4.1.2. APIs RESTful

En el ámbito de los sistemas distribuidos modernos, las APIs RESTful han emergido como la principal técnica de intercambio de datos, estableciendo un estándar de comunicación [6]. En el contexto de los microservicios, las APIs RESTful no solo facilitan el intercambio de datos, sino que también actúan como un mecanismo de orquestación esencial, permitiendo una comunicación fluida y estandarizada entre los distintos servicios [7]. Entre las herramientas disponibles, Flask se destaca como un micro framework web consolidado, opción predilecta para desarrollar backends

ligeros y eficientes. Este, en combinación con otras bibliotecas de Python, posibilita la construcción de APIs RESTful robustas y eficientes, facilitando así la integración y comunicación entre los diferentes componentes en arquitecturas basadas en microservicios [8].

4.1.3. Gateway y Servidor de Registro

Los microservicios, debido a su naturaleza dinámica y distribuida, requieren herramientas especializadas para la gestión de solicitudes y el descubrimiento de servicios. El Servidor de Registro juega un papel crucial, manteniendo un registro activo de todos los microservicios disponibles y facilitando su descubrimiento en tiempo real [9]. Simultáneamente, la Pasarela o Gateway actúa como un punto unificador para las solicitudes, encaminando cada solicitud al microservicio adecuado, optimizando así la distribución de carga y la eficiencia en la comunicación [10].

4.2. Balanceador de Carga

Un balanceador de carga distribuye el tráfico de red entrante a través de múltiples servidores para asegurar que ningún servidor individual se vea sobrecargado con demasiado tráfico. La distribución efectiva de las solicitudes entrantes mejora la fiabilidad y robustez de las aplicaciones, ya que evita que la falla de un único servidor interrumpa el servicio para los usuarios finales [11].

4.3. Sistema de Mensajería

El sistema de mensajería asíncrona es una base para la comunicación de microservicios. El uso de colas de mensajes en sistemas distribuidos permite una mejor tolerancia a fallos, ya que si un servicio falla, el mensaje puede permanecer en la cola y procesarse una vez que el servicio se reanude, asegurando así que no haya pérdida de datos [12]. En este contexto, Kafka se presenta como una plataforma de transmisión de eventos distribuida diseñada para manejar altos volúmenes de datos. Su aplicación en una arquitectura de microservicios respalda eficazmente la comunicación asíncrona, asegurando la consistencia y disponibilidad de datos incluso cuando algunos servicios no están disponibles [13].

4.4. Bases de datos no relacionales

Las bases de datos no relacionales (a menudo llamadas bases de datos NoSQL) han ganado popularidad debido a su escalabilidad y flexibilidad, especialmente al tratar con big data y aplicaciones web en tiempo real. La estructura sin esquema permite un desarrollo rápido y ágil, y la escalabilidad horizontal que ofrecen las hace perfectas para manejar conjuntos de datos grandes o en crecimiento [14].

4.4.1. Bases de datos basadas en documentos

Las bases de datos basadas en documentos, como MongoDB, almacenan información en documentos, comúnmente en formato JSON. Estas bases son particularmente útiles cuando la estructura de los datos puede cambiar con el tiempo, como en aplicaciones web dinámicas [15].

4.4.2. Bases de datos basadas en columnas

Orientadas para soluciones de análisis de grandes volúmenes de datos, las bases de datos basadas en columnas, como Apache Cassandra y HBase, almacenan datos en columnas en lugar de filas. Esto permite realizar operaciones rápidas en grandes conjuntos de datos y es particularmente útil para aplicaciones de análisis [16].

4.4.3. Bases de datos basadas en grafos

Las bases de datos basadas en grafos, como Neo4j, están diseñadas para almacenar datos interconectados. Son especialmente útiles cuando las relaciones entre los datos son tan importantes como los datos en sí. Estas bases de datos son ampliamente utilizadas en redes sociales, sistemas de recomendación y otras aplicaciones que requieren navegación y consulta compleja a través de relaciones [17].

4.5. Virtualización y escalabilidad

La virtualización, particularmente en forma de contenedores, proporciona un entorno ligero, consistente e aislado para ejecutar aplicaciones. Los contenedores no solo aseguran transiciones

fluidas entre diferentes entornos informáticos, sino que también escalan sin esfuerzo, mejorando así la disponibilidad de la aplicación y reduciendo los costos de infraestructura [18].

4.5.1. Tipos de virtualización

4.5.1.1. Virtualización a nivel de sistema

Este enfoque consiste en ejecutar múltiples sistemas operativos en un único host físico. La virtualización a nivel de sistema se realiza principalmente mediante hipervisores, que se dividen en dos categorías: Tipo 1 (o hipervisor bare metal) y Tipo 2 (o hipervisor alojado) [19].

4.5.1.2. Virtualización a nivel de aplicación

Se refiere a soluciones que permiten ejecutar aplicaciones diseñadas para un tipo de sistema operativo en otro sistema diferente, como el caso de Wine en Linux para ejecutar aplicaciones de Windows [20].

4.5.2. Herramientas de virtualización

Además de los contenedores, las herramientas como VMware, Hyper-V y Xen ofrecen soluciones de virtualización a nivel de sistema, permitiendo la creación y administración de máquinas virtuales en infraestructuras físicas [21].

4.5.3. Escalabilidad y gestión de recursos

Con herramientas de orquestación como Kubernetes, es posible garantizar que las aplicaciones escalen de acuerdo a las demandas de tráfico, utilizando recursos de manera eficiente y garantizando alta disponibilidad [18].

4.6. Detección de Intrusiones en IoT

El Internet de las Cosas (IoT) abarca una vasta gama de dispositivos interconectados que intercambian datos sin problemas. La naturaleza de este ecosistema interconectado lo hace altamente susceptible a amenazas externas. Con los atacantes evolucionando continuamente sus tácticas, la necesidad de sistemas de detección de intrusiones robustos y adaptativos se vuelve primordial [22]. La complejidad de los ataques modernos exige contramedidas sofisticadas basadas en aprendizaje automático, diseñadas para detectar patrones, anomalías y amenazas potenciales en los vastos flujos de tráfico de la red [23].

4.7. Métricas y Preprocesamiento en Aprendizaje Automático

En el ámbito del aprendizaje automático y la ciencia de datos, la calidad de los datos y las métricas utilizadas para evaluar modelos son primordiales [24]. Asegurarse de que los datos estén bien preparados y limpios es un paso preliminar que puede influir significativamente en el rendimiento de un modelo [25]. Además, seleccionar la métrica adecuada ayuda a evaluar la eficacia de un algoritmo en diversos escenarios [26]. Esta sección profundiza en las complejidades del preprocesamiento de datos y aclara varias métricas comúnmente utilizadas en el panorama del aprendizaje automático [27].

4.7.1. Aprendizaje Automático y Seguridad de Red

El aprendizaje automático, en su esencia, prospera en patrones. Dada la gran cantidad de datos que fluyen a través de las redes, es poco práctico que los sistemas convencionales monitoreen y analicen todo el tráfico sin automatización. Los modelos de aprendizaje automático, una vez entrenados en conjuntos de datos anotados como BoT-IoT [28, 29] y UNSW-NB15 [30], pueden discernir patrones y señalar anomalías en tiempo real, asegurando que las amenazas potenciales sean rápidamente identificadas y mitigadas [31]. Esta capacidad de adaptarse y aprender del tráfico en curso es lo que distingue a los sistemas de detección de intrusiones basados en aprendizaje automático de sus homólogos convencionales.

4.7.2. Preprocesamiento de Datos

La base de cualquier modelo robusto de ML es la calidad e integridad de los datos en los que se entrena. se destaca la importancia del preprocesamiento para garantizar la consistencia, integridad y relevancia de los datos [32]. Además, los procesos de transformación y limpieza aseguran que

los datos de entrada se alineen con los requisitos del modelo de ML, lo cual es fundamental para lograr una alta precisión en las predicciones [32].

4.7.3. Métricas de Evaluación en Detección de Intrusiones

La eficacia de un sistema de detección de intrusiones no radica únicamente en su capacidad para identificar amenazas, sino en cuán rápido y precisamente lo hace. Varias métricas, tanto tradicionales como específicas del aprendizaje automático, juegan un papel en la evaluación del rendimiento de estos sistemas. Métricas como precisión, exactitud, exhaustividad y la puntuación F1 proporcionan insights sobre el comportamiento del modelo, especialmente en términos de falsos positivos y falsos negativos, que son cruciales en un escenario real de detección de amenazas [23].

4.7.3.1. Matriz de Confusión

Una matriz de confusión es un diseño de tabla específico que permite visualizar el rendimiento de un algoritmo. Cada fila de la matriz representa las instancias en una clase predicha, mientras que cada columna representa instancias en una clase real. Proporciona una vista holística de los resultados de clasificación, destacando las instancias donde las predicciones fueron correctas y donde el modelo cometió errores [33].

4.7.3.2. Exactitud

La exactitud es una representación directa de la corrección global del modelo. Es la proporción del número de predicciones correctas al número total de predicciones. Aunque es ampliamente utilizado, a veces puede ser engañoso, especialmente cuando las distribuciones de clases están sesgadas [34].

4.7.3.3. Sensibilidad

La sensibilidad, también conocida como Tasa de Verdaderos Positivos, es especialmente crítica en contextos donde no detectar un evento es más costoso que una falsa alarma. Cuantifica la capacidad del modelo para identificar todas las instancias relevantes en el conjunto de datos [35].

4.7.3.4. Precisión

La precisión es una medida de la exactitud del modelo. En contextos de seguridad, una mayor precisión asegura que los recursos no se desperdician en falsas alarmas. Esta métrica cuantifica cuántos de los elementos identificados como positivos son realmente positivos [36].

4.7.3.5. Puntuación F1

La puntuación F1 es la media armónica de Precisión y Sensibilidad, ofreciendo un equilibrio entre las dos cuando sus valores divergen. Es especialmente relevante en escenarios donde las clases de datos están desequilibradas, asegurando que los eventos raros no sean pasados por alto ni sobre-predichos [37].

4.7.4. Bases de Datos en Detección de Intrusiones en Red

La evolución y validación de cualquier modelo de aprendizaje automático se basan en la calidad y amplitud del conjunto de datos en el que se entrena. Bases de datos como BoT-IoT y UNSW-NB15 han sido diseñadas específicamente para emular entornos de red del mundo real, conteniendo patrones tanto normales como maliciosos. Esta mezcla diversa asegura que los modelos estén expuestos a una amplia gama de escenarios, permitiéndoles generalizar mejor cuando se despliegan en sistemas del mundo real [30].

5 METODOLOGÍA

El proceso de desarrollo de este proyecto de investigación incluye cuatro fases: planteamiento y diseño, especificación, desarrollo y despliegue. En la Figura **5-1** se muestran las fases y actividades integradas en el proceso de desarrollo de la investigación. Posteriormente, se describen las fases y las actividades del proceso.

5.1. Fase 1: Planteamiento y Diseño

La fase de planteamiento y diseño se centró en el diseño de la arquitectura e infraestructura requeridos para la implementación de los procesos de predicción de los modelos de ML. Dicha arquitectura se fundamentó en los principios esenciales de los sistemas de microservicios, como la modularidad, independencia y escalabilidad del sistema [38].

5.1.1. Actividad 1: Diseño de Arquitectura Basada en Microservicios

El diseño se abordó desde la perspectiva de microservicios. Esta fase se concentró en identificar y estructurar componentes funcionales, considerando principios tales como la modularidad e independencia [39].

1. **Componentes Funcionales:** Se incluyeron preprocesamiento de datos, almacenamiento y despliegue del modelo, manejo de predicciones y registro de resultados.

2. **Comunicación mediante API REST:** Se hizo énfasis en la estructuración y eficiencia en la comunicación entre microservicios y el sistema principal [40].

3. **Gateway y Servidor de Registro:** Se emplearon como intermediarios y facilitadores en la comunicación entre servicios y con peticiones externas.

4. **Balanceo de Carga:** Se implementó una distribución inteligente de las solicitudes para asegurar eficiencia y rendimiento [41].

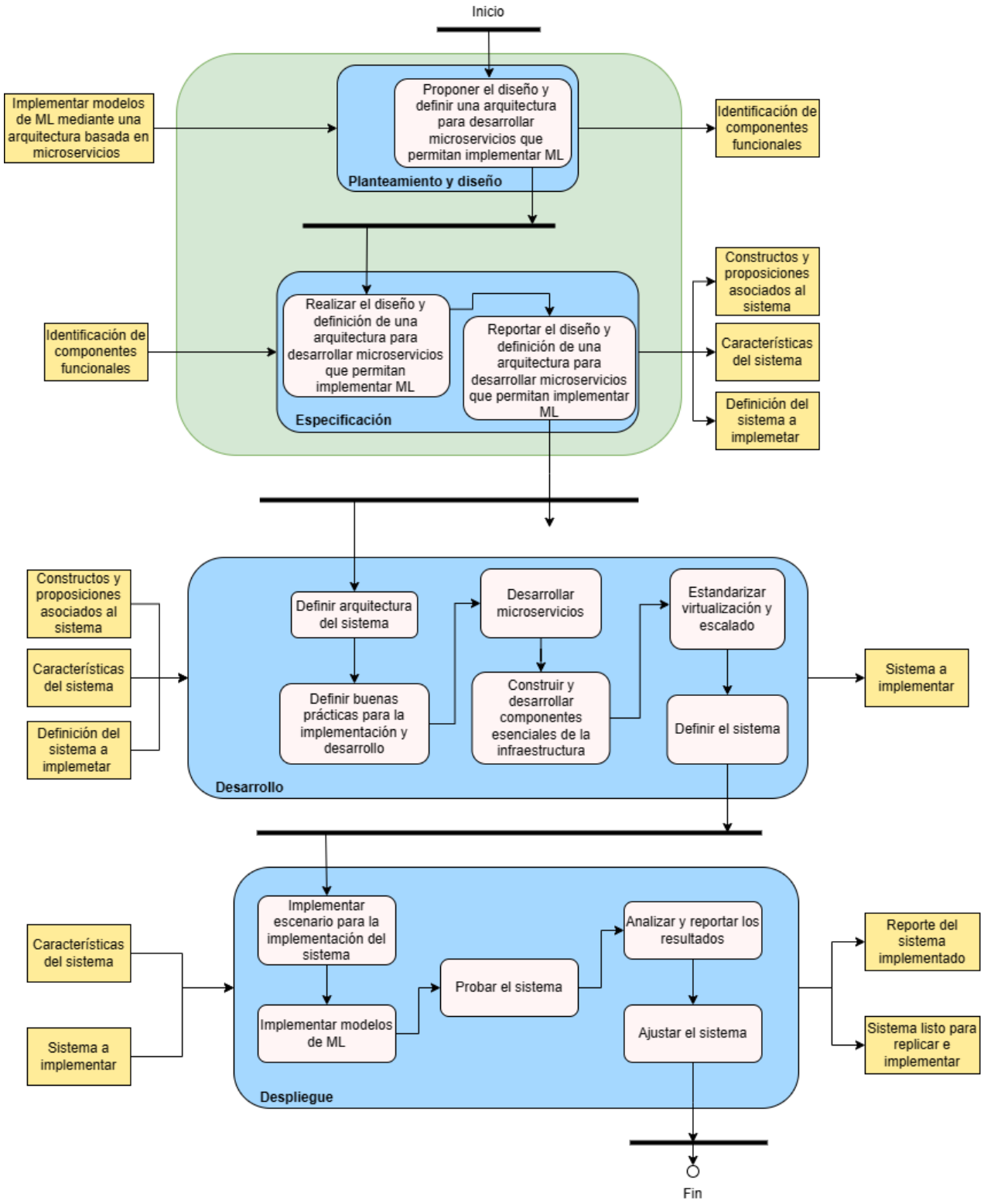

Figura 5-1: Proceso de desarrollo de la investigación.

5.1.2. Actividad 2: Gestión de Datos y Comunicación

Se consideró la estructura y tipo de datos que se gestionarán, así como la manera en que los servicios interactuarán.

1. **Sistema de Mensajería Asíncrona:** Se basó en colas, proporcionando flexibilidad y optimizando la respuesta del sistema [42].

2. **Bases de Datos No Relacionales:** Fueron seleccionadas por su escalabilidad, flexibilidad y simplicidad de implementación [43].

5.1.3. Actividad 3: Despliegue y Escalabilidad

En esta actividad, se detalla el plan para el despliegue de los servicios y cómo garantizar su escalabilidad en función de la demanda. Particularmente, se hace uso de la virtualización mediante contenedores, que proporciona aislamiento, escalabilidad y una gestión eficiente de los recursos [44].

5.2. Fase 2: Especificación

Durante la fase de especificación, se realizó una cuidadosa selección de métricas de rendimiento. Esta selección tuvo como objetivo validar los modelos de aprendizaje automático en el contexto de la seguridad en ciudades inteligentes. Para ello, se consideraron no solo teorías consolidadas, sino también la relevancia y aplicabilidad práctica de cada métrica, teniendo en cuenta las características dinámicas y cruciales de las ciudades inteligentes [45].

5.2.1. Actividad 1: Definición y Justificación de Métricas Seleccionadas

En el proceso de análisis de métricas comúnmente empleadas en el dominio del aprendizaje automático [46], es fundamental elegir aquellas métricas adecuadas para asegurar una interpretación correcta del rendimiento del modelo de aprendizaje automático. Este proceso de selección es crucial no solo para facilitar la toma de decisiones basadas en los resultados del modelo, sino también para garantizar que las métricas seleccionadas estén alineadas con los objetivos específicos del estudio y sean coherentes con el contexto en el que se planea implementar el modelo. A través de una revisión minuciosa de la literatura y considerando las prácticas estándar en el campo del aprendizaje automático, se seleccionaron varias métricas consideradas pertinentes para este

estudio. A continuación, se presentan y justifican estas métricas, destacando su capacidad para abordar los desafíos específicos de nuestro contexto de aplicación, en particular en el ámbito de las ciudades inteligentes.

1. **Matriz de confusión:** Esta métrica es fundamental porque proporciona una perspectiva completa del rendimiento del modelo, con especial enfoque en la clasificación de observaciones. Es valiosa en contextos de seguridad, donde cada clasificación tiene implicaciones tangibles y significativas [47].

2. **Exactitud (Accuracy):** Funciona como un indicador inmediato de la eficiencia global del modelo, siendo una métrica de referencia para evaluar la exactitud de las predicciones [48].

3. **Sensibilidad (Recall):** Esencial para evaluar la habilidad del modelo para detectar eventos o situaciones críticas, la sensibilidad es crucial en contextos donde cada detección puede tener un impacto significativo en la seguridad pública [49].

4. **Precisión (Precision):** Mientras que la sensibilidad se centra en la correcta detección de eventos cruciales, la precisión asegura que dichas detecciones sean acertadas, minimizando así el riesgo de utilizar recursos de manera ineficiente [50].

5. **Puntuación F1 (F1-score):** Esta métrica es esencial en situaciones donde existen clases desequilibradas. Al combinar precisión y sensibilidad, la puntuación F1 garantiza una evaluación justa de todas las clases [51].

La selección de estas métricas tuvo como objetivo proporcionar una visión integral del comportamiento del modelo. Además, las métricas seleccionadas sirven como guía para futuras optimizaciones y modificaciones. Con esto en mente, se buscó asegurar que cualquier modelo implementado no solo fuera robusto y exacto, sino también efectivo dentro del cambiante panorama de las ciudades inteligentes [52].

5.3. Fase 3: Desarrollo

5.3.1. Estrategia de Implementación y Arquitectura

La implementación del sistema se fundamentó en una arquitectura de microservicios. Esta estrategia fue diseñada para alcanzar los objetivos específicos del proyecto y para asegurar una ejecución eficaz de los modelos de aprendizaje automático. Se seleccionaron tecnologías y lenguajes como Python y Java, en combinación con bases de datos no relacionales, un sistema de mensajería de publicación-suscripción y una interfaz API REST.

El núcleo del sistema consistió en dos microservicios esenciales, destinados al preprocesamiento y la predicción de datos. Estos microservicios se desarrollaron utilizando Flask, un framework de Python, que probó ser adecuado por su eficiencia en la creación rápida de servicios. La versatilidad de Flask facilitó la integración de modelos de aprendizaje automático, mejorando así el funcionamiento principal del sistema.

5.3.2. Comunicación y Mensajería

Se estableció una comunicación externa con los microservicios a través de una API REST, la cual proporciona interfaces para acceder y consumir los servicios brindados por cada microservicio. Esta estrategia promovió la interoperabilidad con otros sistemas y aplicaciones.

Para garantizar una comunicación continua y fiable entre los microservicios, se incorporó un sistema de mensajería basado en colas y un enfoque asincrónico. Se eligió a Kafka como el broker de mensajería, creando así un canal seguro y eficiente para la transmisión de datos y eventos entre microservicios. Con un sistema de tipo FIFO, Kafka mantuvo la integridad de los mensajes y la tolerancia a fallos.

5.3.3. Almacenamiento de Datos

Cada microservicio del sistema estuvo vinculado directamente con una base de datos no relacional MongoDB. Esta elección permitió un almacenamiento de datos ágil y escalable. MongoDB es conocido por gestionar grandes volúmenes de información sin sacrificar el rendimiento, lo que lo hace ideal para infraestructuras basadas en microservicios.

5.3.4. Virtualización y Gestión

Para optimizar la implementación y gestión de los microservicios, se optó por la virtualización a través de contenedores Docker. Además, se utilizó un servidor de descubrimiento, Eureka, desarrollado en Java por Netflix. Este servidor funcionó como un punto centralizado para todas las APIs de los microservicios, simplificando así su gestión y configuración.

Se incorporó un microservicio con la funcionalidad de API Gateway usando el framework de Java, Spring Framework, junto con el servicio Spring Cloud Gateway. Este componente garantizó la integridad y seguridad del sistema, al servir como único punto de entrada para todas las solicitudes externas.

5.3.5. Balanceadores de Carga

Con el objetivo de optimizar el rendimiento y asegurar la escalabilidad, se introdujeron dos balanceadores de carga. Estos balanceadores distribuyeron el tráfico de manera equitativa entre microservicios, garantizando un servicio ininterrumpido a los usuarios.

5.3.6. Conclusión del Desarrollo

La estrategia basada en microservicios, ilustrada en la Fig. **5-2**, junto con las tecnologías y lenguajes seleccionados, demostró ser robusta y eficiente. La combinación de Kafka, MongoDB, Docker, el servidor de registro y el API Gateway, consolidaron un sistema fiable y eficaz.

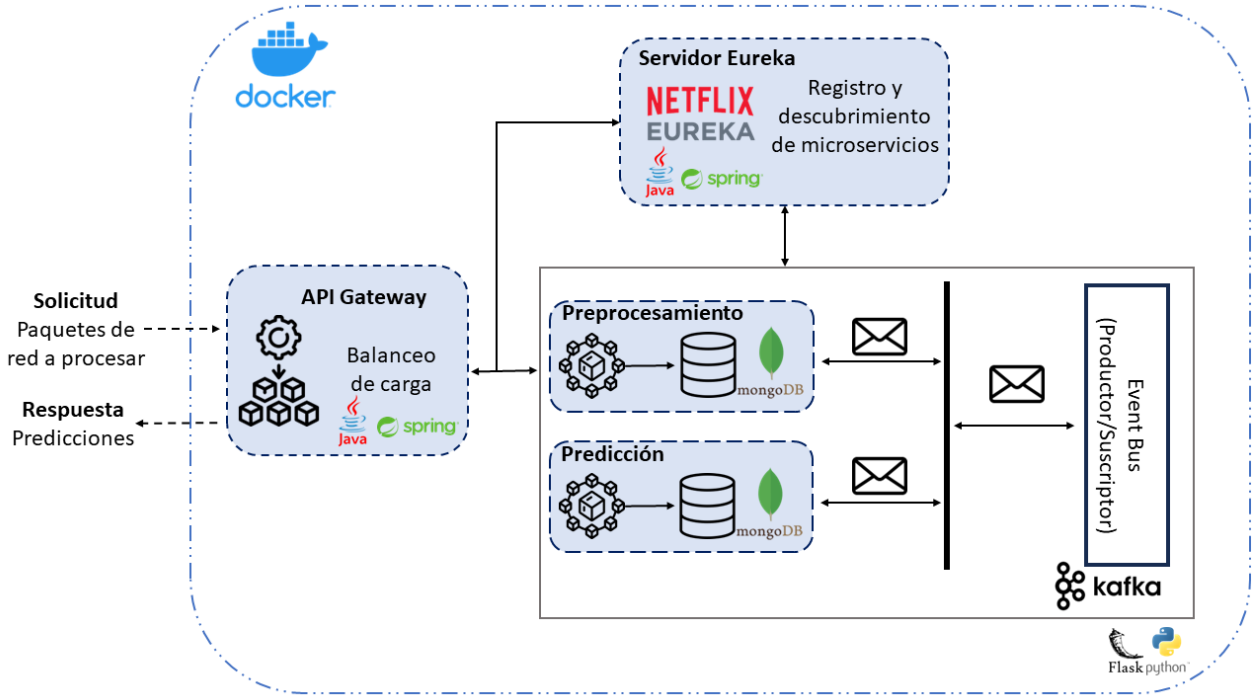

Figura 5-2: Arquitectura del sistema de implementación de modelos de ML.

Ahora, el sistema desarrollado para la implementación de modelos de aprendizaje automático consta de dos microservicios centrales:

5.3.7. Microservicio de preprocesamiento

El preprocesamiento de datos es esencial en la preparación de cualquier modelo de aprendizaje automático. Para abordar esta necesidad, se ha desarrollado un microservicio específico. Su función primordial radica en la implementación de las operaciones necesarias para adecuar los datos entrantes, independientemente del modelo de ML que se vaya a utilizar.

Cuando se pretende implementar un nuevo modelo de ML, se verifica si se requiere alguna función de estandarización específica. Aquí, se ha incorporado un método para almacenar funciones de estandarización, identificadas por un nombre único. Esta característica permite la diferenciación y carga personalizada de funciones según las necesidades del modelo. El esquema de almacenamiento se detalla en la Fig. **5-3**. El método de almacenamiento en la base de datos MongoDB se encuentra enlazado a un endpoint que recopila el nombre de la función y la función en un archivo .pkl

El preprocesamiento se materializa mediante un método que recibe un JSON conteniendo los datos originales a ser procesados. Posteriormente, estos datos son transformados para adaptarse a los requerimientos del modelo correspondiente. En este enfoque, se puede desarrollar un método de preprocesamiento único para cada modelo. El microservicio tiene la capacidad de serializar los datos procesados y transmitirlos a través de Kafka al microservicio de predicción. Esta función opera en un hilo de ejecución independiente, lo que garantiza su carácter asincrónico.

Una vez que el microservicio de predicción procesa los datos y genera una respuesta, la cual incluye resultados de clasificación y tiempos de predicción para cada modelo, el microservicio de preprocesamiento almacena esta respuesta. Cada registro se asocia con un identificador que refleja el tipo de modelo al que corresponde.

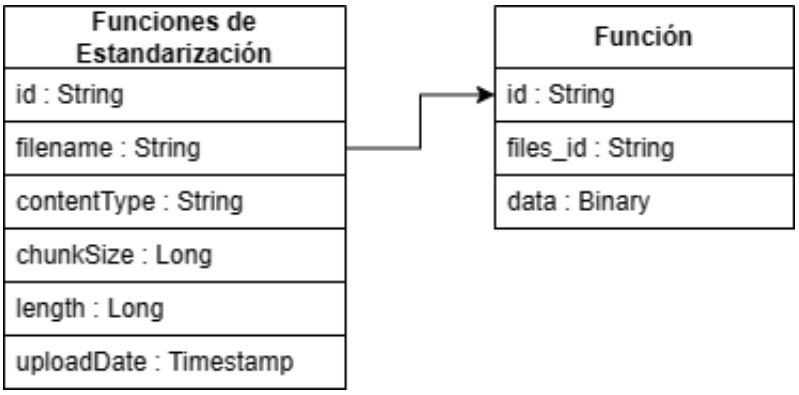

Figura 5-3: Diagrama de la entidad de parámetros de estandarización.

A continuación, se detalla uno de los endpoints disponibles en este microservicio:

· Almacenar los parámetros de estandarización:

POST <ip>:<port>/seguridad/preprocesamiento/save/param/standardization
Body: form-data (Files) param: archivo de estandarización (.pkl) name: nombre del archivo de estandarización
Respuesta: Almacenamiento correcto status: 200 Mensaje: Save succesfully

5.3.8. Microservicio de predicción

Este microservicio se ha diseñado para llevar a cabo la fase de predicción una vez que los datos han sido debidamente procesados. Su estructura se divide en dos funciones de gran relevancia. En primer lugar, se encarga de almacenar los diversos modelos de aprendizaje automático (ML) que se van a implementar. Para lograr esto, se ha establecido un endpoint específico que acepta archivos .pkl junto con el nombre correspondiente del modelo a ser almacenado, en la Fig **5-4**. se detalla el esquema de almacenamiento.

La segunda función del microservicio reside en la utilización efectiva de estos modelos previamente almacenados. Cuando recibe los datos preprocesados a través de Kafka, junto con su identificador único, el microservicio ejecuta el modelo pertinente y devuelve una lista de valores que refleja la predicción realizada. Esta respuesta es enviada de manera eficiente y asincrónica, lo que garantiza un proceso de predicción continuo y fluido en el sistema implementado.

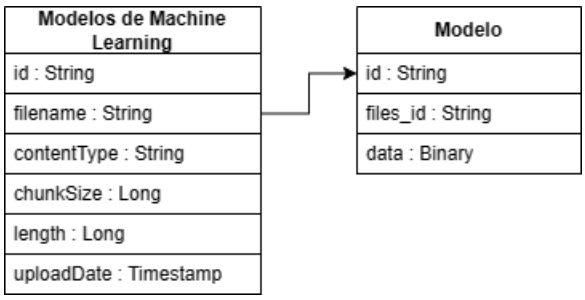

Figura 5-4: Diagrama de la entidad de modelos de ML.

A continuación, se detalla el endpoint disponible en este microservicio:

· Almacenar modelo de ML:

POST <ip>:<port>/seguridad/prediccion/save/model/{name-model}
Body: form-data (Files) param: archivo del modelo de clasificación (.pkl) name: nombre del archivo de estandarización
Respuesta: Almacenamiento correcto status: 200 Mensaje: Update succesfully

5.4. Fase 4: Despliegue

Se evaluó el sistema de manera activa, adoptando dos modelos diferenciados de inteligencia computacional. El corazón de esta evaluación estuvo centrado en la creación de un escenario orientado hacia la identificación en tiempo real de posibles ataques en el tráfico de red. El propósito principal de esta metodología era asegurar una identificación precisa y expedita de las amenazas emergentes, facilitando de esta forma decisiones inmediatas, tales como la restricción de acceso a un dispositivo específico o sistema vinculado al Internet de las Cosas (IoT). Con esta capacidad de identificación temprana de actividades no autorizadas, el sistema se encontraba en una posición ventajosa para desencadenar respuestas correctivas de manera autónoma. El protocolo de evaluación se desglosa en tres actividades fundamentales.

5.4.1. Actividad 1: Implementación de los modelos de ML en el sistema de detección de intrusos basado en microservicios

El mecanismo para detectar ataques se basó en técnicas de aprendizaje automático, responsables de examinar el tráfico de red y de diferenciar entre flujos normales y aquellos generados por posibles ataques. Para que el sistema pudiera realizar esta diferenciación correctamente, fue esencial someterlo previamente a un proceso de entrenamiento usando un conjunto de datos específicos de tráfico de red detallados a continuación:

5.4.1.1. Actividad 1.1: Base de datos BoT-IoT

Este conjunto de datos fue diseñado específicamente para favorecer el análisis de tráfico de red en el contexto del IoT. Una característica distintiva de BoT-IoT no solo radica en proporcionar datos, sino también en ofrecer un entorno de simulación que reproduce el comportamiento real de dispositivos IoT conectados en una red.

Dicho entorno de simulación fue meticulosamente estructurado y contaba con máquinas virtuales designadas como atacantes, cuya tarea era emular y ejecutar diferentes acciones maliciosas. Entre estas acciones se encuentran, pero no se limitan a, los ataques de denegación de servicio (DoS y DDoS) y actividades asociadas a la recopilación no autorizada y sustracción de información. En la TABLA **5-1**, se detalla la distribución de los registros en el conjunto de datos BoT-IoT.

Clase	**Numero de registros**
Normal	643
Ataque	3667879
Total	**3668522**

Tabla 5-1: DISTRIBUCIÓN DE LOS REGISTROS EN EL SUBCONJUNTO DEL 5 % DE LA BASE DE DATOS BOT-IOT

5.4.1.2. Actividad 1.2: Base de datos UNSW-NB15

La base de datos UNSW-NB15 ha sido diseñada para la evaluación y validación de sistemas de detección de intrusos o ataques en redes. Este recurso se destaca por su capacidad de mimetizar un entorno de red actualizado y contemporáneo. La creación de la UNSW-NB15 no se basó únicamente en patrones de tráfico convencionales; en cambio, se empleó la herramienta IXIA PerfectStorm para generar una mezcla de actividades normales reales, propias de la era digital actual, junto con comportamientos de ataque sintéticos. Estos últimos se diseñaron a partir de un diccionario de acceso público, el cual cataloga una amplia variedad de vulnerabilidades y exposiciones de seguridad informática conocidas [30].

Para proporcionar una perspectiva más clara sobre la estructura y la variedad de registros contenidos en UNSW-NB15, la TABLA **5-2** desglosa detalladamente la distribución de estos registros en el conjunto de datos.

Clase	**Numero de registros**
Normal	2218764
Ataque	321283
Total	**2540047**

Tabla 5-2: DISTRIBUCIÓN DE LOS REGISTROS DE LA BASE DE DATOS UNSW-NB15

5.4.1.3. Actividad 1.3: Creación de una Base de Datos Propia

Para entender y analizar de manera más específica el tráfico de red en nuestro escenario de prueba, se decidió construir una base de datos personalizada. Esta recopila información detallada

de los paquetes de red que circulan por dicho entorno.

Para lograr este fin, se recurrió a la herramienta Argus [53]. Este software, especializado en auditorías de red, permite una captura en tiempo real del flujo de paquetes, redirigiendo la información directamente a una de las interfaces de red del sistema.

Una vez capturado, el tráfico se almacena inicialmente en el formato binario propio de Argus. Posteriormente, mediante la misma herramienta, se procede a extraer características relevantes que describan con precisión a los paquetes de red. Estos datos refinados se almacenan en una base de datos MongoDB. Por razones de accesibilidad y análisis, la información se exporta posteriormente a un archivo CSV.

Para determinar qué características se debían utilizar en los modelos de aprendizaje automático, se llevó a cabo un análisis comparativo entre las bases de datos BoT-IoT y UNSW-NB15. Se identificaron y seleccionaron las características que ambas bases compartían. Estas características comunes también se extrajeron de nuestra base de datos personalizada.

Entre las caracteristicas denotadas en la Tabla Tabla **5-4**, se observan inicialmente 27 características que fueron vitales para la construcción de nuestros modelos de ML, que se enfocaron en una clasificación binaria, con el objetivo principal de identificar posibles ataques en la infraestructura de red. Es importante señalar que, de estas características, 15 son entregadas directamente por la herramienta Argus. Las 12 características restantes no se derivan directamente, sino que se calculan utilizando expresiones específicas basadas en las proporcionadas por Argus.

5.4.1.4. Actividad 1.4: Preprocesamiento y entrenamiento de modelos.

Actualmente, con las bases de datos preparadas, procedemos al preprocesamiento de los datos. Esta etapa es crucial para adaptar y refinar los datos, preparándolos para la construcción eficiente de modelos mediante técnicas de aprendizaje automático. Las acciones de preprocesamiento se implementan en todas las bases de datos utilizadas. A continuación, se describen las principales acciones emprendidas:

a. **Corrección de Etiquetas en el Protocolo ARP:** Según el análisis sobre la base de datos BoT-IoT realizado por Peterson et al. [28], se descubre que el protocolo ARP no está directamente asociado con las acciones de ataque identificadas. Sin embargo, existen instancias erróneamente etiquetadas como ataques bajo este protocolo. Para garantizar la precisión y fiabilidad de la información, Re etiquetamos estas instancias, clasificándolas como tráfico normal en lugar de ataques. Esta acción optimiza la calidad del conjunto de datos y garantiza su pertinencia en la detección de amenazas reales.

b. **Normalización de los Datos:** Comprendemos que las características en un conjunto de

datos pueden variar significativamente en escala y comportamiento. Para nivelar el terreno y garantizar que cada característica reciba la importancia adecuada en los modelos de aprendizaje, normalizamos los datos. Específicamente, transformamos las características para que sigan una distribución normal, con una media de 0 y una desviación estándar de 1. Este proceso de estandarización es fundamental para mantener la coherencia y la relevancia de las variables.

c. **Eliminación de Características No Informativas:** Basándonos en investigaciones previas [29], entendemos que algunas características, aunque técnicamente relevantes, no aportan valor al proceso de modelado predictivo. Estas características, en su mayoría, se relacionan con la identificación de paquetes, como las direcciones y puertos de origen y destino. Al eliminar esta información, minimizamos el ruido y las distracciones en el conjunto de datos, reduciendo simultáneamente la dimensionalidad. Esta acción puede potenciar la precisión del modelo y simplificar su interpretación.

Una vez finalizado el meticuloso proceso de preprocesamiento de los datos, avanzamos hacia las fases esenciales de entrenamiento, evaluación y ejecución de los modelos de aprendizaje automático. Durante la construcción de nuestro modelo de detección de ataques en sistemas IoT, decidimos apostar por cuatro algoritmos de clasificación supervisados, fundamentándonos en un exhaustivo análisis de la literatura especializada. Estos algoritmos son:

- **Árboles de Decisión:** Una herramienta poderosa que permite decisiones basadas en condiciones específicas del conjunto de datos, implementada a través de DecisionTreeClassifier de la librería Sklearn [54].

- **Bosques Aleatorios:** Este enfoque agrupa varios árboles de decisión para obtener una predicción más precisa y robusta, utilizando RandomForestClassifier de Sklearn [55].

- **Regresión Logística:** Un método estadístico utilizado para modelar relaciones entre una variable dependiente categórica y una o más variables independientes.

- **SVM (Support Vector Machine):** SVM es una técnica que busca el mejor hiperplano que separe las clases de datos, y en nuestro caso, se implementó con el SGDClassifier de Sklearn [56], que puede adaptarse para operar como SVM.

Después de someter nuestros modelos al proceso de entrenamiento, procedemos a almacenarlos en el microservicio de predicción. Esta decisión estratégica asegura que los modelos estén listos y disponibles para ejecuciones posteriores. Es fundamental destacar que este almacenamiento no es simplemente un paso de archivado; representa una integración activa con nuestra infraestructura, permitiendo que el microservicio acceda de manera eficiente a los modelos y los despliegue con precisión cuando sea necesario.

5.4.2. Actividad 2: Extractor de tráfico de red

El Extractor de Tráfico de Red actúa como un pilar fundamental de la infraestructura, siendo responsable de capturar y analizar los paquetes que circulan a través de una interfaz de red. En términos sencillos, su tarea principal es registrar detalles esenciales de cada paquete para facilitar el aprendizaje automático.

Se utiliza Open Argus, una herramienta de código abierto, para llevar a cabo la captura y el análisis de paquetes de red. Esta herramienta no solo organiza los paquetes en flujos de red periódicos, sino que también realiza un seguimiento bidireccional, capturando datos en ambas direcciones entre dispositivos de origen y destino. Para identificar estos flujos de red, utiliza cinco criterios clave: direcciones IP de origen y destino, puertos de origen y destino, y tipo de protocolo.

Después de que Argus recopila estos flujos de red, un script en Python se encarga de canalizarlos hacia el detector de ataques. Para mantener una transferencia eficaz y segura, el script opera con la librería asyncio, permitiendo el procesamiento asincrónico de los datos en bloques de 100 registros. Cada bloque se transmite al microservicio para análisis y se monitoriza el tiempo que lleva el proceso.

El script de python se divide en tres funciones principales:

1. **get_stream_reader:** Esta función establece una conexión de entrada/salida entre dos procesos utilizando la clase StreamReader del módulo asyncio. Sirve como un puente para leer datos del flujo de entrada estándar.

2. **send_api_request:** Funcionando de forma concurrente, esta función gestiona una cola de solicitudes. Utiliza el módulo aiohttp para realizar peticiones HTTP asincrónicas, enviando datos para ser procesados y monitorizando el tiempo que toma el análisis.

3. **main:** Es donde está la ejecución principal del programa. Inicialmente, invoca a get_steam_reader para obtener datos de la herramienta Argus. Luego, gestiona la cola de solicitudes, procesando datos en bloques y colocándolos en la cola cuando es necesario.

En esencia, el Extractor de Tráfico de Red, con la ayuda de Open Argus y el script de Python, garantiza una captura y procesamiento eficiente de datos de red para nuestras necesidades de aprendizaje automático.

5.4.3. Actividad 3: Detector de ataques basado en microservicios

Dentro del marco de nuestra investigación, se han integrado modelos de aprendizaje automático (ML) en una infraestructura que opera con base en microservicios. Estos microservicios han sido desarrollados específicamente para facilitar diversas funcionalidades.

5.4.3.1. Microservicio de preprocesamiento

Este microservicio se encarga principalmente de la recepción y tratamiento inicial de paquetes de datos, que llegan en bloques de 100 registros y en formato JSON. Su propósito se extiende más allá de ser un simple receptor:

· **Transformación de Datos:** Al recibir los datos, se lleva a cabo una serie de transformaciones para adaptarlos conforme a los parámetros de preprocesamiento ya establecidos. Estas transformaciones están alineadas con la metodología descrita en la Fig. **5-3**, permitiendo el cálculo de características adicionales, que son detalladas en la TABLA **5-3**.

Categoria	Caracteristica	Descripcion
Estadísticas de conexión	TnBPSrcIP	Numero total de bytes por IP origen en 100 conexiones
	TnBPDstIP	Número total de bytes por IP destino en 100 conexiones
	TnP_PSrcIP	Número total de paquetes por IP origen en 100 conexiones
	TnP_PDstIP	Número total de paquetes por IP destino en 100 conexiones
	TnP_PerProto	Número total de paquetes por protocolo en 100 conexiones
	TnP_Per_Dport	Numero total de paquetes por puerto de destino en 100 conexiones
	AR_P_Proto_P_SrcIP	Tasa promedio por protocolo por IP origen en 100 conexiones (Calculado por paquetes/duracion)
	AR_P_Proto_P_DstIP	Tasa promedio por protocolo por IP destino en 100 conexiones
	N_IN_Conn_P_SrcIP	Número de conexiones entrantes por IP origen en 100 conexiones
	N_IN_Conn_P_DstIP	Número de conexiones entrantes por IP destino en 100 conexiones
	AR_P_Proto_P_Sport	Tasa promedio por protocolo por puerto de origen en 100 conexiones
	AR_P_Proto_P_Dport	Tasa promedio por protocolo por puerto de destino en 100 conexiones

Tabla 5-3: CARACTERÍSTICAS CALCULADAS A PARTIR DE LAS ENTREGADAS POR ARGUS

· **Estandarización:** Los datos pasan por un proceso de estandarización que garantiza su coherencia con el set de entrenamiento usado durante la construcción de los modelos ML.

· **Serialización y Transmisión:** Una vez que los datos han sido adecuadamente procesados, se vuelven a serializar en formato JSON. Posteriormente, a través de un tópico específicamente creado y configurado en Kafka denominado "data-preprocessing", estos datos son enviados al microservicio de predicción.

· **Funcionalidad Predictiva:** Al recibir los datos, el microservicio de predicción aplica los modelos de aprendizaje automático para discernir si la actividad registrada corresponde a un comportamiento regular o a un potencial ataque. Como resultado, no solo obtenemos una clasificación, sino también el tiempo que cada modelo toma en realizar su respectiva predicción.

· **Almacenamiento de Datos:** Todo dato procesado, así como las predicciones y métricas asociadas, se almacenan en una base de datos MongoDB bajo el nombre "preprocesamiento". La estructura y diseño de esta base de datos se ilustra en la Tabla **5-4**, donde se destaca la colección principal "data". Esta colección es la encargada de guardar las características representativas de cada paquete de red, las predicciones derivadas de cada modelo ML, el tiempo requerido para procesar bloques de 100 registros, y la clasificación real de los registros.

Tabla 5-4: DIAGRAMA DE LAS ENTIDADES EN EL MICROSERVICIO DE PREPROCESAMIENTO.

Data	
id	Mongo ID
stime	decimal
proto	string
proto_number	int
pkts	int
bytes	int
state	string
state_number	int
ltime	decimal
dur	decimal
spkts	int
dpkts	int
sbytes	int
dbytes	int
TnBPSrcIP	int
TNBPDstlIP	int
TnP_PSrcIP	int
TnP_PDstIP	int
TnP_PerProto	int
TnP_Per_Dport	int
AR_P_Proto_P_SrcIP	decimal
AR_P_Proto_P_DstIP	decimal
N_IN_Conn_P_DstIP	int
N_IN_Conn_P_SrcIP	int
AR_P_Proto_P_Sport	decimal

AR_P_Proto_P_Dport	decimal
tag	int
prediction_dt	int
time_prediction_dt	decimal
prediction_lr	int
time_prediction_lr	decimal
prediction_rf	int
time_prediction_rf	decimal
predicition_svm_linear	int
time_prediction_svm_linear	decimal

A continuación, se detalla el nuevo endpoint disponible en este microservicio:

Tabla 5-5: PREPROCESAMIENTO Y PREDICCIÓN DE 100 REGISTROS

GET <ip>:<port>/seguridad/preprocesamiento/data/standardization/ <param1>/<param2>/<param3>
Los tres últimos valores de la API corresponden a los parámetros utilizados en el momento de las acciones de ataque, en caso de no ser necesario enviar un valor de 0.

Body: Raw (JSON)
{
"stime":{"0":1667596695.270251,"1":1667596695.414449,...,"99":1667596716.776453},
...
"dbytes": {"0":222,"1":322,...,"99":151}
}

Respuesta: Valores de predicción y tiempo de predicción
Status: 200 OK
Body: JSON
{
[1,1,1,...,1,0.002948045],
...
[1,0,0,...,1,0.003280401]
}

Se recibe una lista con las predicciones de los cuatro modelos en el siguiente orden: árboles de decisión, regresión logística, bosques aleatorios y SVM. Cada posición de la lista contiene un vector con los resultados de la predicción del modelo correspondiente, donde 1 representa tráfico de ataque y 0 tráfico normal; al final del vector se encuentra el tiempo que tarda el modelo en entregar los resultados.

5.4.3.2. Microservicio de predicción

En el diseño integral de nuestra infraestructura, un componente esencial es el microservicio de predicción. Este se encarga de manejar y aplicar los diversos modelos de aprendizaje automático (ML) que se encuentran almacenados en él. Estos modelos, de vital importancia para nuestro sistema, han sido entrenados con anterioridad, siguiendo las especificaciones detalladas en la sección 5.3.8. Las funcionalidades y el proceso de Trabajo del microservicio es el siguiente:

- **Integración con Kafka:** El microservicio está integrado con el broker Kafka. A través de esta conexión, recibe los datos que serán sometidos a predicción.

- **Suscripción a Tópicos:** Este microservicio se suscribe específicamente al tópico "data-preprocessing", originado en el microservicio de preprocesamiento. Dicha suscripción garantiza que cada vez que se introduzcan datos en este tópico, el microservicio de predicción sea notificado y pueda actuar en consecuencia.

- **Conversión y Análisis:** Al recibir los datos, estos son inmediatamente transformados en un dataframe. Posteriormente, se ingresan a los modelos de ML para realizar el análisis y obtener las respectivas predicciones. Durante este proceso, también se mide el tiempo que cada modelo demora en realizar su predicción, proporcionando un indicador crucial de eficiencia.

- **Compilación de Resultados:** Tras el proceso de predicción, se compilan los resultados. Esta compilación incluye las predicciones de los cuatro modelos utilizados y el tiempo asociado a cada uno. Luego, estos datos se serializan en formato JSON y se publican en el tópico "data-prediction".

- **Integración con el Microservicio de Preprocesamiento:** Cabe mencionar que el microservicio de preprocesamiento se encuentra suscrito al tópico "data-prediction". Esto facilita una comunicación fluida entre ambos servicios, permitiendo que el microservicio de preprocesamiento reciba y utilice los datos publicados sin inconvenientes.

- **Requisitos para su Funcionamiento:** Es fundamental para el correcto funcionamiento del microservicio de predicción, contar con la librería scikit-learn instalada. Además, es imperativo tener acceso a los modelos de ML que fueron entrenados previamente, garantizando que las predicciones sean coherentes y precisas.

A continuación, se detallan el nuevo endpoint disponible en este microservicio:

- Almacenar el tiempo que tarda en responder los microservicios al procesar 100 registros

POST <ip>:<port>/seguridad/prediccion/save/time/argus/<time>
El parámetro <time> de la API corresponde al tiempo desde la captura de 100 registros hasta obtener la respuesta con las predicciones de los datos.
Respuesta: Almacenamiento correcto Status: 200 Body: True

5.4.3.3. Microservicio de Gateway

El microservicio de Gateway desempeñó un papel crucial en la administración y direccionamiento de las solicitudes entrantes hacia los microservicios correspondientes. Se implementó utilizando el Spring Framework, y, en particular, se aprovechó Spring Cloud Gateway para dirigir el tráfico hacia los microservicios de predicción y preprocesamiento. Tres características principales sustentaron su eficacia y seguridad:

- **Configuración de Rutas:** Con las capacidades de Spring Cloud Gateway, se definieron rutas específicas para gestionar el tráfico. Estas rutas aseguraron que cada solicitud se dirigiera al destino apropiado, proporcionando una navegación eficiente y coherente en el sistema.

- **Seguridad:** Se otorgó prioridad a la seguridad en la infraestructura. Mediante la inclusión de la clase SpringSecurityConfig, se asignaron permisos específicos a cada ruta. Así, para acceder a ciertos endpoints, no solo era necesario autenticarse, sino también poseer un token JWT OAuth 2 válido, brindando una capa adicional de protección contra accesos no autorizados.

- **Redireccionamiento Inteligente:** Con la exposición de la aplicación a través de un puerto específico, se implementó un mecanismo que redirigió eficientemente el tráfico externo hacia el servidor. Este proceso optimizado mejoró la velocidad de respuesta y el procesamiento, y también mejoró la accesibilidad y experiencia del usuario.

Este microservicio, al integrar estas características, no solo funcionó como un enrutador de tráfico sino también como un guardián de seguridad y eficiencia, asegurando un rendimiento óptimo y seguro.

5.4.3.4. Servidor de registro

En la arquitectura propuesta para el proyecto, se identificó al servidor de registro como un elemento esencial para garantizar la eficiencia y escalabilidad de los servicios. Se optó por Eureka de Netflix, construido sobre el marco de Spring, estableciendo un ecosistema en el que los microservicios (preprocesamiento, predicción y gateway) se integraron y coordinaron de manera óptima. Al examinar las características y ventajas de Eureka, se resalta su valor fundamental:

· **Balanceo de Carga:** Eureka proporcionó mecanismos avanzados de balanceo de carga, enfrentando uno de los principales desafíos en sistemas distribuidos. Distribuyó solicitudes de manera uniforme entre las diferentes instancias de microservicios, asegurando tiempos de respuesta rápidos y evitando cuellos de botella.

· **Descubrimiento de Servicios:** La capacidad de descubrimiento y comunicación entre múltiples servicios es esencial en entornos distribuidos. Eureka facilitó esta interacción, permitiendo que los servicios descubrieran y se comunicaran entre sí sin problemas.

La elección de Eureka demostró ser acertada, ya que se integró eficientemente con el ecosistema general y ofreció una serie de beneficios intrínsecos que potenciaron la eficiencia y escalabilidad de la arquitectura.

5.4.3.5. Contenerización y despliegue

En la metodología, la contenerización y el despliegue se abordaron con meticulosidad y precisión, garantizando la robustez y eficiencia de los servicios. Se ha elegido Docker como herramienta principal, debido a su confiabilidad y flexibilidad en la creación y gestión de contenedores.

- **Configuración de Servicios Esenciales:** Para elementos clave como la base de datos y Kafka, se diseñaron dos archivos `docker-compose.yml` distintos. Estos archivos no solo facilitan la orquestación de dichos servicios, sino que también aseguran que las configuraciones se ajusten a las necesidades específicas y que el despliegue se realice sin inconvenientes.

- **Contenerización de Microservicios:**

  - **Dockerfiles Individuales:** Cada microservicio es respaldado por un `Dockerfile` dedicado. Este archivo describe meticulosamente las instrucciones y dependencias necesarias para contenerizar el microservicio de manera adecuada. Esto garantiza que cada servicio esté encapsulado con todas las herramientas y configuraciones que necesita para operar de manera óptima.

  - **Docker-Compose Centralizado:** A nivel de arquitectura, todos estos microservicios están interrelacionados y, por lo tanto, requieren una coordinación efectiva. Para ello, en el repositorio principal, disponemos de un archivo `docker-compose.yml` centralizado. Este archivo orquesta el despliegue coordinado de todos los microservicios, asegurando una interacción fluida entre ellos y reduciendo las posibilidades de conflictos o errores durante el despliegue.

- **Entorno de Despliegue Especializado:** Todo este ecosistema de servicios y microservicios se despliega en una máquina virtual diseñada específicamente para análisis de datos. Al alojar

esta VM en un servidor local, obtenemos ventajas considerables: un control total sobre el entorno, la capacidad de realizar ajustes rápidos y una escalabilidad eficiente, asegurando que la infraestructura pueda adaptarse a cargas de trabajo más grandes o a nuevos requisitos según surjan.

6 Resultados y Análisis

6.1. Características Identificadas y Descripción de la Base de Datos

Estas características derivadas brindan una visión más profunda del comportamiento del tráfico de red, facilitando la detección de posibles amenazas.

Categoría	**Característica**	**Descripción**
Información de tiempo	Stime	Tiempo de inicio del flujo
	Ltime	Tiempo de la ultima actividad del flujo
	Dur	Duración del flujo
Información de protocolo	Proto	Protocolo utilizado en el flujo
Información de direcciones y puertos	Saddr	Dirección IP de origen
	Daddr	Dirección IP de destino
	Sport	Puerto de origen
	Dport	Puerto de destino
Información de trafico	Pkts	Cantidad de paquetes
	Bytes	Cantidad de bytes
	Spkts	Cantidad de paquetes desde la dirección IP de destino
	Dpkts	Cantidad de paquetes desde la dirección IP de destino
	Sbytes	Cantidad de bytes desde la dirección IP de origen
	Dbytes	Cantidad de bytes desde la dirección IP de destino
Información de estado	State	Estado del flujo

Tabla 6-1: CARACTERÍSTICAS GENERADAS POR LA HERRAMIENTA ARGUS

6.2. Análisis Comparativo de Bases de Datos

Al analizar las tres bases de datos mencionadas, se observó una distribución desigual entre los registros de tráfico normal y aquellos asociados a ataques.

· En BoT-IoT, apenas el 0,018 % de los registros corresponden a tráfico normal.

· En UNSW-NB15, el 87,26 % de los registros se identifican como tráfico normal.

· Mientras que, en nuestra base de datos propia, el tráfico normal constituye un 9,49 %.

6.3. Equilibrio y Preparación de los Datos

Con el propósito de equilibrar las muestras para el entrenamiento y las pruebas, se procedió a construir una base de datos balanceada usando una submuestra que garantizara una distribución equitativa de los datos.

Para BoT-IoT, se conservaron todos los registros de tráfico normal y se eligieron aleatoriamente 200.000 registros etiquetados como ataques. En el caso de UNSW-NB15, solo se conservaron 199.357 registros de tráfico normal. Por último, en nuestra base de datos personalizada, se mantuvieron todos los registros de tráfico normal y se seleccionaron aleatoriamente 341.423 registros de tráfico etiquetados como ataque.

Con este método, se alcanzó una distribución balanceada en las bases de datos, resultando en 541.423 registros para tráfico normal y un número igual, 541.423 registros, para el tráfico de ataque.

6.4. Microservicios: Una Infraestructura para la Implementación de Modelos de ML

A lo largo de la presente investigación, se optó por una metodología teórica dirigida al desarrollo de una infraestructura basada en microservicios. Dicha infraestructura está diseñada primordialmente para llevar a cabo tareas de preprocesamiento y análisis detallado del tráfico de red. El propósito central de estas operaciones es la identificación temprana y precisa de intrusiones o anomalías que pudieran comprometer la seguridad de la red. La arquitectura propuesta integra diversos elementos esenciales para la optimización de los procesos mencionados. Entre estos, se incluyen un gateway, cuya función es facilitar la comunicación y transferencia eficiente de datos; y un balanceador de carga, diseñado para distribuir de manera equitativa y estratégica las solicitudes recibidas entre los microservicios disponibles. Asimismo, se desarrollaron dos microservicios principales, los cuales no solo son fácilmente replicables y coherentes, sino que también son sencillos de mantener y actualizar gracias a su estructura modular y a la implementación de prácticas de desarrollo ágil. Cada uno de los microservicios incorporados en la infraestructura cuenta con su propia base de datos. Esto no solo promueve la autonomía y la independencia entre los servicios,

sino que también facilita la gestión y manipulación de la información. Para garantizar una comunicación asincrónica efectiva y robusta entre los microservicios, se implementó un sistema de colas de mensajes, seleccionando para ello Kafka por su rendimiento y confiabilidad comprobados. Este conjunto de características confecciona una infraestructura que se destaca por su agilidad y ligereza. Su diseño permite una adaptación y replicación eficiente en una variedad de escenarios y contextos operativos, ofreciendo soluciones versátiles frente a las necesidades emergentes en entornos de ciudades inteligentes.

6.5. Rendimiento de los Microservicios

Tras el desarrollo y despliegue de los microservicios y la infraestructura, se propusieron cuatro escenarios para evaluar el rendimiento del sistema, comparándolo con un monolito que posee funcionalidades equivalentes. Esta evaluación permite verificar la cantidad de peticiones exitosas y analizar los tiempos de respuesta de ambos sistemas. Para ello, se utilizó la petición definida en la Tabla **5-5**, registrando como respuesta el tiempo de ejecución de la arquitectura basada en microservicios y de un monolito típico. La Tabla **6-2** muestra los cinco primeros valores de tiempo de respuesta registrados para una petición.

Tabla 6-2: TIEMPOS DE EJECUCIÓN DE UN MONOLITO Y EL SISTEMA BASADO EN MICROSERVICIOS

processing_time_monolito	**processing_time_microservices**
3.421356678	1.393592119
3.812394381	1.73174572
3.325146437	2.136006355
3.396124125	2.31491375
2.115172148	2.181381702

Además muestra los resultados de:

- **count:** Indica el número total de peticiones realizadas a cada servicio (100 para ambos).
- **mean:** Muestra el tiempo medio de respuesta.
- **std:** Indica la desviación estándar del tiempo de respuesta, reflejando la variabilidad de los datos.
- **min:** Tiempo mínimo de respuesta registrado.
- **25 %, 50 %, 75 %:** Primer, segundo (mediana) y tercer cuartiles de la distribución de tiempos de respuesta.

- **max:** Tiempo máximo de respuesta registrado.

6.5.1. 100 peticiones consecutivas

El análisis de rendimiento con 100 peticiones consecutivas tiene el objetivo de evaluar cómo cada arquitectura se comporta bajo carga repetitiva y continua. Es importante mencionar que, si bien 100 peticiones no representan una carga extrema, sí permiten obtener una perspectiva inicial del comportamiento de cada sistema frente a demandas consecutivas.

En la Tabla **6-3**, se desglosan los tiempos de respuesta para ambas arquitecturas. Es notorio que, en términos de promedio (mean), el monolito tiene un tiempo de respuesta ligeramente inferior (aproximadamente 0.03 segundos más rápido) que el sistema basado en microservicios. Esta diferencia podría ser atribuida a la sobrecarga de gestión inherente de los microservicios, como la comunicación entre ellos, lo cual no está presente en un monolito.

Tabla 6-3: ANÁLISIS DE RESPUESTA DE 100 PETICIONES LINEALES A DOS SERVICIOS: UN MONOLITO Y EL SISTEMA DE MICROSERVICIOS

	processing_time_monolito	**processing_time_microservices**
count	100	100
mean	0.179670861	0.208702509
std	0.008510555	0.007122109
min	0.168210745	0.200571537
25 %	0.172811031	0.205328345
50 %	0.17736435	0.207275391
75 %	0.1866045	0.209622383
max	0.220294714	0.263301849

Por otro lado, la Figura **6-1** ilustra visualmente el comportamiento de cada arquitectura a lo largo de las 100 peticiones. Se observa que el monolito presenta un patrón más estable y coherente en sus tiempos de respuesta. Sin embargo, el sistema basado en microservicios muestra picos más pronunciados en algunas peticiones, lo que indica una mayor variabilidad en su rendimiento.

6.5.2. 1000 peticiones consecutivas

Para ampliar el alcance del análisis y evaluar cómo se comportan las arquitecturas bajo una demanda más intensa, se realizó una prueba con 1000 peticiones consecutivas. Esta prueba es esencial para comprender el rendimiento del sistema cuando se somete a una carga más prolongada y repetitiva, y para identificar posibles áreas de mejora.

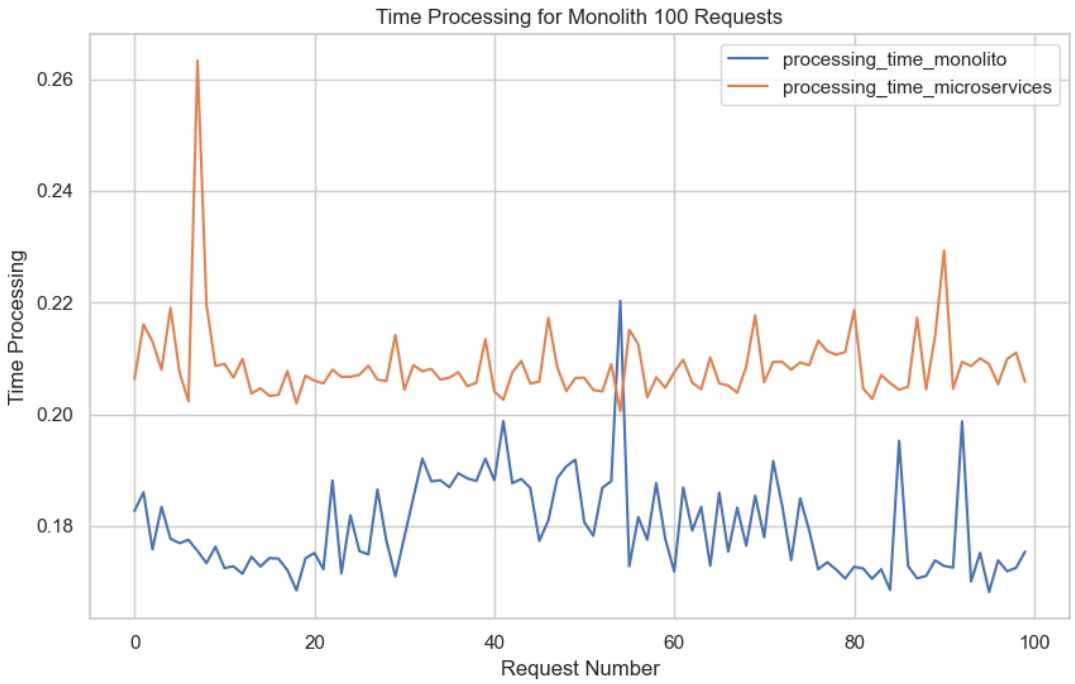

Figura 6-1: Comparación de tiempos de respuesta entre el sistema monolítico y el sistema basado en microservicios.

La Tabla **6-4** muestra los resultados de los tiempos de respuesta para las 1000 peticiones realizadas tanto al sistema monolítico como al basado en microservicios. En comparación con la prueba de 100 peticiones, es evidente que, aunque el tiempo promedio de respuesta (mean) del monolito sigue siendo más rápido, la diferencia se ha reducido ligeramente. Esto sugiere que, a medida que la cantidad de peticiones aumenta, el rendimiento del sistema basado en microservicios tiende a estabilizarse y acercarse al del monolito. Además, al analizar la desviación estándar (std), se observa que ambos sistemas presentan valores similares, lo que indica una consistencia comparable en la variabilidad de sus tiempos de respuesta. Es importante notar que, a pesar de las ventajas en escalabilidad y flexibilidad que ofrecen los microservicios, pueden presentar tiempos de respuesta ligeramente mayores debido a la complejidad adicional en la gestión y comunicación entre los diferentes servicios.

En la Figura **6-2**, se ilustra la comparación de tiempos de respuesta entre las dos arquitecturas a lo largo de las 1000 peticiones. Se puede apreciar que el patrón de comportamiento del monolito sigue siendo relativamente estable, con algunas fluctuaciones menores. En contraste, el sistema basado en microservicios muestra variaciones más pronunciadas en ciertos puntos, pero en general, su rendimiento tiende a mejorar con el transcurso de las peticiones. Esta tendencia sugiere que, con el tiempo, es probable que el sistema basado en microservicios no solo alcance, sino que posiblemente supere al monolito en términos de eficiencia de tiempo de respuesta.

Tabla 6-4: ANÁLISIS DE RESPUESTA DE RESPUESTA DE 1000 PETICIONES LINEALES A DOS SERVICIOS: UN MONOLITO Y EL SISTEMA DE MICROSERVICIOS

	processing_time_monolito	**processing_time_microservices**
count	1000	1000
mean	0.173038657	0.206829506
std	0.006277969	0.006487913
min	0.16533637	0.197475433
25 %	0.169862509	0.203870416
50 %	0.171570301	0.205447078
75 %	0.174138605	0.20749104
max	0.225871086	0.263898849

6.5.3. 1000 peticiones simultáneas

El análisis de rendimiento con 1000 peticiones simultáneas busca entender cómo se comportan las dos arquitecturas cuando enfrentan una alta demanda simultánea. Este escenario es crucial para comprender la capacidad de cada sistema de manejar ráfagas de tráfico y demandas pico, situaciones típicas en entornos de producción. La Tabla **6-5** resume los resultados obtenidos. Se puede observar que el sistema monolítico gestionó 468 de las 1000 peticiones con un tiempo medio de respuesta de 2.32 segundos, siendo su tiempo máximo de 4.24 segundos. Por otro lado, el sistema de microservicios atendió 514 peticiones, pero con un tiempo medio significativamente mayor, alcanzando los 40.93 segundos. El dato más notable aquí es el tiempo máximo, que llegó a ser de 670.30 segundos, indicando que algunas peticiones enfrentaron largos periodos de espera o congestión.

Tabla 6-5: ANÁLISIS DE RESPUESTA DE RESPUESTA DE 1000 PETICIONES SIMULTANEAS A DOS SERVICIOS: UN MONOLITO Y EL SISTEMA DE MICROSERVICIOS

	processing_time_monolito	**processing_time_microservices**
count	468	514
mean	2.325419566	40.93993083
std	0.789441842	86.14884672
min	0.182293415	0.227177858
25 %	1.858524263	1.996881128
50 %	2.417278529	8.766057372
75 %	2.883807898	33.10580099
max	4.240494967	670.3078642

La Figura **6-3** ilustra visualmente esta comparativa. A primera vista, es evidente que el sistema basado en microservicios presentó picos de tiempo de respuesta mucho más altos en comparación

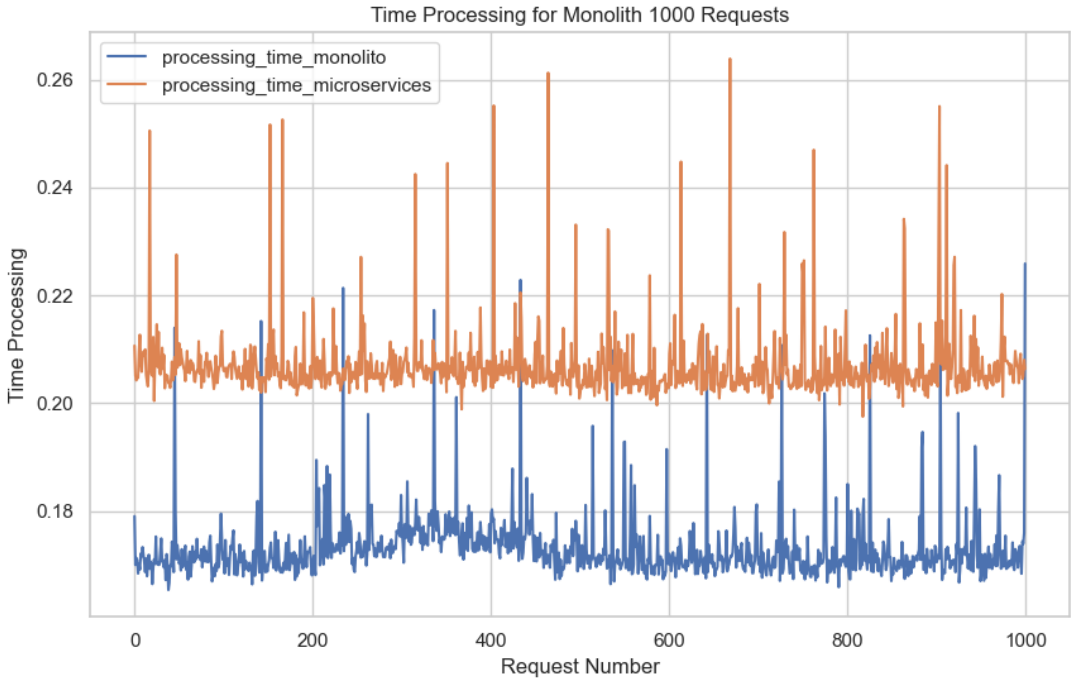

Figura 6-2: Comparación de tiempos de respuesta entre el sistema monolítico y el sistema basado en microservicios.

con el monolítico. Estos picos pueden ser indicativos de cuellos de botella o problemas de gestión de recursos en el sistema de microservicios.

6.5.4. 1000 peticiones simultaneas escalando el sistema de microservicios

Para evaluar el comportamiento del sistema bajo cargas extremas y su capacidad de escalamiento, se realizaron pruebas con 1000 peticiones simultáneas, ajustando el sistema de microservicios para aprovechar la capacidad de escalado horizontal. El escalado horizontal, en sistemas distribuidos, es la práctica de añadir más instancias para distribuir la carga, en lugar de aumentar la capacidad de una sola instancia (escalado vertical).

En el contexto del diseño de sistemas, entender la respuesta a estas cargas extremas y la capacidad de escalado es fundamental para asegurar una operación estable, especialmente en situaciones donde se esperan picos de demanda.Los resultados presentados en la Tabla **6-6** muestran estadísticas descriptivas de los tiempos de respuesta obtenidos. Se observa que el sistema monolítico tuvo tiempos de respuesta con menos variabilidad, mientras que el sistema de microservicios mostró una mayor dispersión en sus tiempos, lo que podría indicar diferencias en la capacidad de manejo de

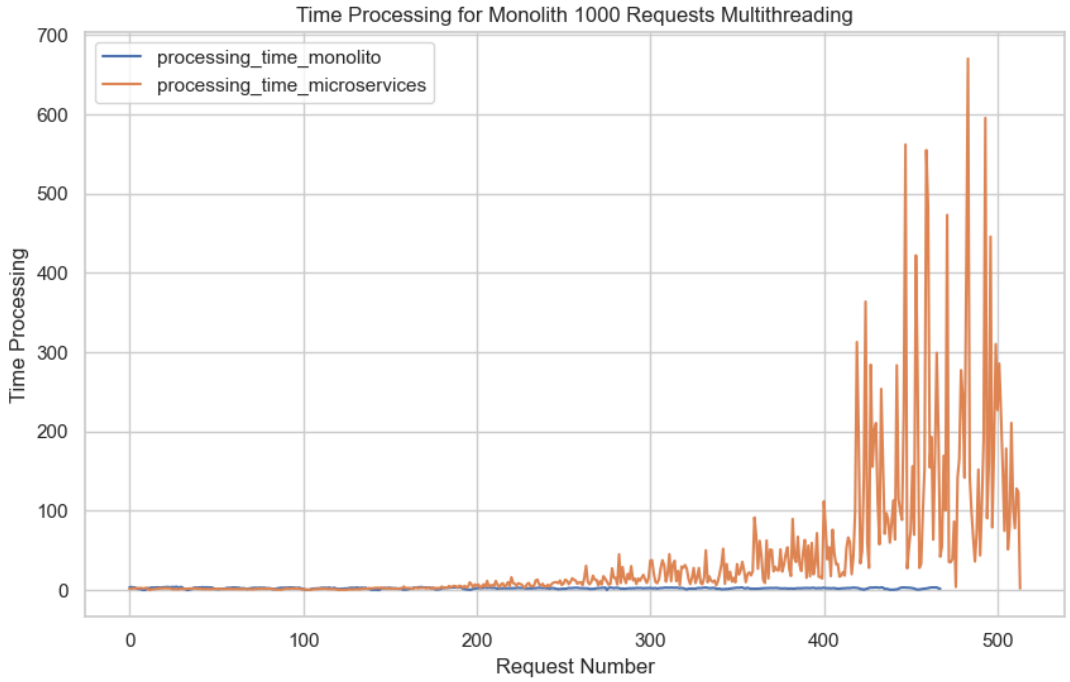

Figura 6-3: Comparación de tiempos de respuesta entre el sistema monolítico y el sistema basado en microservicios.

la carga entre las instancias escaladas. El sistema monolítico, al no ser escalado, mostró tiempos de respuesta promedio de 26.14 milisegundos, con una variación estándar de 6.80. El tiempo más corto fue de 12.11 milisegundos, mientras que el más largo fue de 42.08 milisegundos.

Por otro lado, el sistema de microservicios escalado tuvo tiempos de respuesta promedio más altos, siendo de 43.93 milisegundos. Sin embargo, el rango de respuesta fue más amplio, oscilando desde 1.50 hasta 227.01 milisegundos. La Figura **6-4** ilustra la comparación entre ambos sistemas. Es notable cómo el sistema de microservicios, a pesar de ser escalado, presentó picos más altos en tiempos de respuesta en ciertas situaciones, lo que sugiere que, si bien la escalabilidad puede ayudar a mejorar el rendimiento general, existen otros factores en juego que afectan la respuesta del sistema. Es esencial entender estos factores y ajustar el sistema en consecuencia para optimizar su rendimiento en situaciones de alta demanda.

Tabla 6-6: ANALISIS DE RESPUESTA DE 1000 PETICIONES SIMULTANEAS A DOS SERVICIOS: UN MONOLITO Y EL SISTEMA DE MICROSERVICIOS ESCALADO

	processing_time_monolito	**processing_time_microservices**
count	57	327
mean	26.14197751	43.92724899
std	6.803384946	35.98811574
min	12.10999703	1.503571033
25 %	21.75717211	17.84910524
50 %	27.3231318	32.62061214
75 %	31.47635388	57.89106345
max	42.08078361	227.0137391

6.6. Análisis sobre el rendimiento de los Microservicios

El proceso de transición y adaptación de sistemas monolíticos a microservicios es un tema que hoy en día da bastante de que hablar en la industria del software. Los resultados presentados anteriormente ilustran una pequeña muestra de lo que significa esta decisión. La escalabilidad, en particular, es una de las características más destacadas de una arquitectura basada en microservicios.

Para plataformas que anticipan un alto nivel de concurrencia o una demanda masiva de peticiones, es imperativo contar con una infraestructura que pueda escalar horizontalmente para acomodar el incremento en las demandas de procesamiento. Un sistema monolítico, aunque puede ser eficiente en ciertos escenarios, puede encontrar desafíos cuando se enfrenta a picos de tráfico o cargas intensas y consecutivas. En cambio, una arquitectura de microservicios, diseñada adecuadamente, puede distribuir el tráfico y procesar múltiples peticiones simultáneamente, asegurando tiempos de respuesta consistentes y menores fallos bajo carga.

Es fundamental subrayar la importancia del escalamiento horizontal. Aunque un sistema pueda escalar, es vital que componentes críticos, como el balanceador de carga y el gateway, sean también escalados para evitar que estos se saturen. Si un balanceador de carga no se escala correctamente, puede terminar siendo el punto de fallo del sistema, reduciendo la eficiencia de la arquitectura basada en microservicios.

Los resultados también indican que, incluso si hay un aumento en el tiempo de respuesta, la arquitectura sigue siendo disponible para atender peticiones. Esta disponibilidad y resiliencia son cruciales en escenarios de alta demanda. Aunque puede haber una pérdida en el rendimiento inicial, con un adecuado escalamiento horizontal, mediante la implementación de más microservicios y un balanceo eficiente, esta situación puede ser optimizada.

No obstante, cabe mencionar que una limitante en las pruebas realizadas fue la disponibilidad

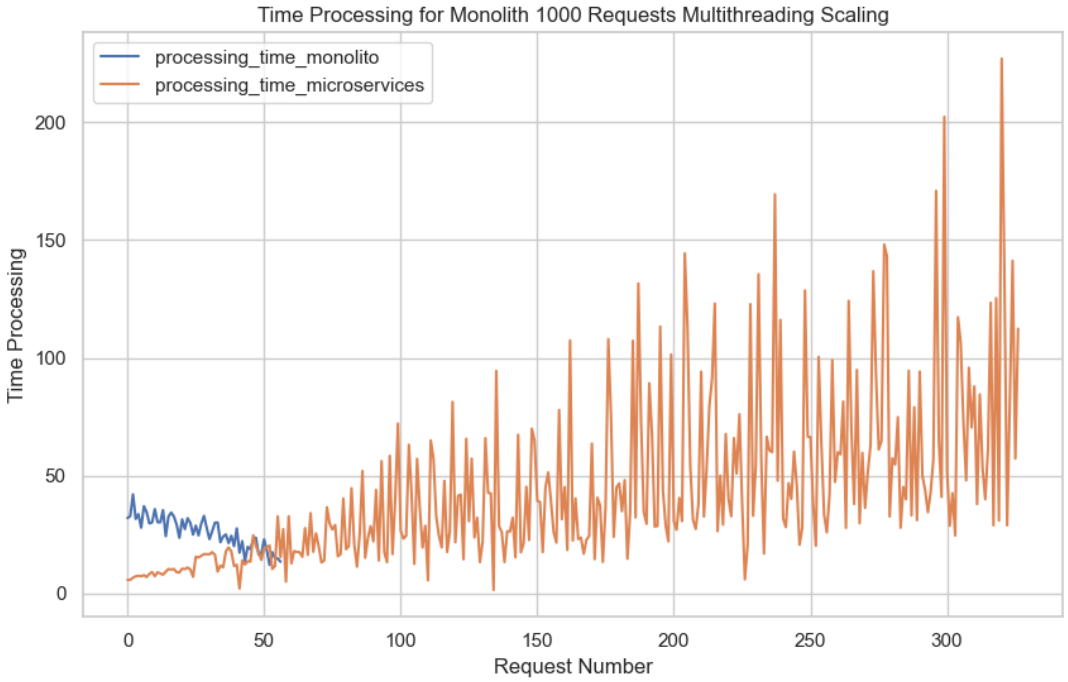

Figura 6-4: Comparación de tiempos de respuesta entre el sistema monolítico y el sistema basado en microservicios.

de recursos hardware. Las replicas, aunque efectuadas, se montaron en una misma máquina física. Esto implica que, aunque haya capacidad para manejar más peticiones, los recursos como CPU y memoria son compartidos, lo cual podría no reflejar el rendimiento óptimo de la arquitectura en un escenario real con múltiples máquinas físicas.

6.7. Trabajos a Futuro

A lo largo de este trabajo se abordo en una propuesta de como implementar una infraestructura para el despliegue modelos de analisis de datos en microservicios orientado en seguridad informatica en ciudades inteligentes. Sin embargo, a continuación, se presentan algunas recomendaciones y áreas de interés para futuros trabajos en este campo:

- **Evaluación del balanceador y gateway**: Sse empleó un único tipo de balanceador de carga y gateway. Sería interesante evaluar el comportamiento y rendimiento del sistema utilizando diferentes configuraciones y tipos de balanceadores y gateways, con el objetivo de optimizar aún más la distribución de peticiones y el rendimiento del sistema.

- **Pruebas con mayores cargas de peticiones**: Aunque se realizaron algunas pruebas con bastante carga, sería relevante someter al sistema a cargas de peticiones aún mayores. Esto permitiría evaluar su comportamiento en escenarios más extremos y verificar si es necesario realizar ajustes adicionales en la arquitectura o en el código para mantener la estabilidad y rendimiento del sistema.

- **Despliegue en diferentes máquinas físicas**: Para evitar la compartición de recursos de CPU y memoria en un único equipo, sería recomendable desplegar los microservicios en diferentes máquinas físicas. Esto ofrecería una visión más realista del verdadero potencial de escalabilidad de la arquitectura y podría mejorar significativamente el rendimiento y la resiliencia del sistema.

- **Integración con infraestructura en la nube**: El despliegue de los microservicios en entornos de nube permitiría aprovechar toda la infraestructura y herramientas que los proveedores de nube ofrecen. Esto incluye balanceadores de carga avanzados, bases de datos distribuidas, monitoreo en tiempo real y, lo más importante, la posibilidad de escalar rápidamente según las demandas.

7 CONCLUSIONES

1. A través de la implementación y evaluación de modelos de aprendizaje automático en el contexto de la seguridad informática de ciudades inteligentes, se puede mostrar cómo aprovechar la programación moderna y las arquitecturas basadas en microservicios para mejorar la eficacia y eficiencia de los procesos de analítica.

2. La arquitectura basada en microservicios es flexible y fácil de escalar para poder implementar sistemas de predicción en el contexto de seguridad informática y en general, implementar modelos de analítica en entornos con grandes volúmenes de datos.

3. Los microservicios permiten separar responsabilidades, facilitando el mantenimiento y mejorando la posibilidad de escalabilidad en caso de tener que emplear procesos de manera paralela, beneficiándose de ventajas como la comunicación asincrónica entre componentes para responder de manera efectiva.

4. Las métricas seleccionadas son necesarias para garantizar que los modelos de aprendizaje automático estén respondiendo de acuerdo a lo esperado. Las métricas brindan la capacidad de monitorear y mejorar la precisión y eficiencia de los modelos implementados.

5. El mecanismo desarrollado, que incorpora las arquitecturas y métricas definidas, logra responder adecuadamente y muestra flexibilidad. Al utilizar Python y Java, junto con bases de datos no relacionales y sistemas de mensajería de publicación-suscripción, se ha creado un sistema que es capaz de ejecutar modelos de aprendizaje automático y que también almacena y calcula métricas relevantes.

6. La selección, implementación y evaluación de modelos de inteligencia computacional validan que es sencillo de implementar modelos de ML en sistemas distribuidos y fuertemente usados, además sirve como un caso de estudio que demuestra que el sistema es apto para adaptarse a diferentes modelos y contextos de aplicación, no solo en el ámbito de seguridad informática si no implementar cualquier tipo de modelo de predicción en ambientes con grandes volúmenes de información o procesamiento.

7. Se muestra que en términos de escalabilidad y resiliencia, los sistemas basados en microservicios tienen una clara ventaja sobre los sistemas monolíticos, especialmente cuando se manejan grandes volúmenes de peticiones o datos.

8. La facilidad de replicación, escalabilidad y mantenimiento del sistema propuesto, en conjunto con su robustez y adaptabilidad, se plantea como una solución viable y prometedora para abordar desafíos de seguridad informática en ciudades inteligentes y en general, implementando modelos de analítica en escenarios complejos o de gran concurrencia de información.

Bibliografía

[1] Ignacio Rodríguez, María Campo-Valera, and Víctor Fajardo. *Conectando el Futuro: Ciudades Inteligentes, IoT y la Transformación de la Sociedad Urbana*. 05 2023. ISBN 9788413352572. doi: 10.24310/mumaedmumaed.27.

[2] L. Bass, J. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. IEEE Press Series on Systems and Software Engineering. 2015.

[3] N. Dragoni, S. Dustdar, S. Larsen, and M. Mazzara. *Microservices: Yesterday, today, and tomorrow*, pages 195–216. Springer, 2017.

[4] C. Richardson. Pattern: Microservice architecture. http://microservices.io/patterns/microservices.html, 2017. Accessed: 09-06-2023.

[5] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. IEEE Communication Networks and Systems. 2015.

[6] R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[7] F. P. Costa, A. M. Delgado, and J. M. da Silva e Souza. Modeling microservices-based systems. *Software: Practice and Experience*, 50(3):275–309, 2020.

[8] M. L. Ginsberg. Flask web development with python tutorial. *IEEE Web Services Transactions*, 2014.

[9] J. Smith and D. Roberts. Using eureka for microservice registration and discovery. *IEEE Cloud Computing*, 3(4):42–49, 2016.

[10] E. Evans. *Decomposing the monolith: Gateway*, pages 123–130. Addison-Wesley, 2004.

[11] M. Malkowski and R. H. Katz. Scaling and optimizing as-a-service platforms using a reference architecture. *IEEE Internet Computing*, 15(3):8–13, 2011.

[12] S. Newman. Building microservices: designing fine-grained systems. *IEEE Software*, 33(3): 91–93, 2016.

[13] N. Debs. Real-time analytics with storm and cassandra. In *IEEE Open Source Database Conference*, 2015.

[14] P. Zikopoulos and C. Eaton. Understanding big data: analytics for enterprise class hadoop and streaming data. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):300–307, 2012.

[15] Kyle Banker. Mongodb in action. 2011.

[16] Avinash Lakshman and Prashant Malik. *Cassandra: A Decentralized Structured Storage System*, volume 44. ACM, 2010.

[17] Emil Eifrem Ian Robinson, Jim Webber. *Graph Databases*. O'Reilly Media, 2013.

[18] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, omega, and kubernetes. *IEEE Communications Magazine*, 33(4):50–57, 2016.

[19] J. E. Smith and R. Nair. The architecture of virtual machines. *IEEE Computer*, 38(5):32–38, 2005.

[20] M. Fetterman A. Ho, C. Clark, A. Warfield, and S. Hand. Practical taint-based protection using demand emulation. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 29–41, 2006.

[21] B. Dragovic P. Barham, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauera, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[22] S. Garcia. Data preprocessing techniques for classification without feature selection. In *IEEE European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 97–102, Apr 2016.

[23] J. Heaton. Evaluation metrics for machine learning. In *IEEE International Conference on Data Science and Machine Learning*, pages 112–118, Aug 2017.

[24] Z. Zhang. Machine learning foundations and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):983–987, May 2018.

[25] S. Garcia. Data preprocessing techniques for classification without feature selection. In *IEEE European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 97–102, Apr 2016.

[26] J. Heaton. Evaluation metrics for machine learning. In *IEEE International Conference on Data Science and Machine Learning*, pages 112–118, Aug 2017.

[27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[28] J. M. Peterson, J. L. Leevy, and T. M. Khoshgoftaar. A review and analysis of the bot-iot dataset. In *Proceedings - 15th IEEE International Conference on Service-Oriented System Engineering, SOSE 2021*, pages 20–27, Aug 2021. doi: 10.1109/SOSE52839.2021.00007.

[29] J. L. Leevy, J. Hancock, T. M. Khoshgoftaar, and J. Peterson. Detecting information theft attacks in the bot-iot dataset. In *Proceedings - 20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021*, pages 807–812, 2021. doi: 10.1109/ICMLA52953.2021.00133.

[30] N. Moustafa and J. Slay. Unsw-nb15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, Dec 2015. doi: 10.1109/MILCIS.2015.7348942.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[32] S. J. Russell and P. Norvig. *Data Preprocessing*, pages 82–92. Prentice Hall, 3rd edition, 2009.

[33] S. Sumathi and S. N. Sivanandam. Introduction to data mining and its applications. *IEEE Transactions on Systems, Man, and Cybernetics*, 36(6):1453–1454, 2006.

[34] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *IEEE International Conference on Machine Learning*, pages 233–240, 2006.

[35] T. Fawcett. An introduction to roc analysis. *IEEE Pattern Recognition Letters*, 27(8):861–874, 2006.

[36] K. Bache and M. Lichman. Uci machine learning repository. *IEEE Journal of Machine Learning Research*, 2013.

[37] C. Drummond and R. C. Holte. Cost curves: An improved method for visualizing classifier performance. *IEEE Machine Learning*, 65(1):95–130, 2006.

[38] S. Newman. *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2015.

[39] M. Richards. *Microservices vs. Service-Oriented Architecture*. O'Reilly Media, Inc., 2016.

[40] M. Masse. *REST API design rulebook*. O'Reilly Media, Inc., 2011.

[41] S. Luo, J. Wu, E. Ding, X. Liao, and Z. Hu. A review of automatic load-balancing and load-distribution approaches in cloud environments. *Journal of Systems and Software*, 100: 129–140, 2014.

[42] G. Hohpe and B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley, 2004.

[43] J. Han, E. Haihong, G. Le, and J. Du. Survey on nosql database. In *2011 6th international conference on Pervasive computing and applications*, pages 363–366. IEEE, 2011.

[44] D. Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[45] M. Batty et al. Smart cities of the future. *The European Physical Journal Special Topics*, 214 (1):481–518, 2013.

[46] S. Raschka. *Python machine learning*. Packt Publishing Ltd, 2015.

[47] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[48] C. Sammut and G. Webb. *Encyclopedia of Machine Learning*. Springer, 2011.

[49] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, page 233–240, 2006.

[50] D. M. W. Powers. Evaluation: from precision, recall and f-factor to roc, informedness, markedness and correlation. 2007.

[51] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45(4):427–437, 2006.

[52] I. Hashem et al. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2016.

[53] Argus Developer Community. *Argus - The Network Flow Auditor*. Qosient, LLC, 2023. URL https://openargus.org/.

[54] sklearn.tree.decisiontreeclassifier — documentación de scikit-learn 1.1.2, 2023. URL https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. accessed Aug. 07, 2023.

[55] sklearn.ensemble.randomforestclassifier — scikit-learn 1.1.2 documentation, 2023. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier. accessed Aug. 07, 2023.

[56] sklearn.linear_model.sgdclassifier — scikit-learn 1.1.2 documentation, 2023. URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html. accessed Aug. 07, 2023.