UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

DISEÑO DE UNA API PARA INTEGRAR PROCESOS DE PROCESAMIENTO DE LENGUAJE NATURAL DENTRO DEL GRUPO DE INVESTIGACIÓN GIHP4C DE LA UNIVERSIDAD POLITÉCNICA SALESIANA - SEDE CUENCA

AUTOR:

JOHN HENRY TENESACA CRIOLLO

DIRECTORA:

GABRIEL ALEJANDRO LEÓN PAREDES

CUENCA – ECUADOR
2024

Autor:

John Henry Tenesaca Criollo
Ingeniero en Ciencias de la Computación.
Candidato a Magíster en Software con Mención en Diseño de Arquitectura de Sistemas por la Universidad Politécnica Salesiana – Sede Cuenca.
Jtenesacac2@est.ups.edu.ec

Dirigido por:

Gabriel Alejandro León Paredes
Ingeniero de Sistemas.
Máster Universitario en Ingeniería Web.
Doctor en Tecnologías de Información.
gleon@ups.edu.ec

DEDICATORIA

Dedico este proyecto a mi familia, cuyo apoyo incondicional ha sido clave para superarme y crecer en todo momento. En especial, a mi hijo Juan, mi mayor inspiración y motivación, quien me impulsa a esforzarme cada día para ser un padre ejemplar.

Con cariño,
John Tenesaca.

AGRADECIMIENTO

Al culminar este proyecto de titulación, quiero expresar mi más profundo agradecimiento a mi familia, quienes han sido fundamentales en mi crecimiento personal y profesional. También extiendo mi gratitud a la Universidad Politécnica Salesiana, que cuenta con un cuerpo docente de excelencia en el área de sistemas. En especial, quiero destacar al Ing. Gabriel León, un modelo a seguir no solo por su profesionalismo, sino también por su calidad humana y su capacidad para guiar a los demás.

Reconozco que mi éxito profesional ha sido posible gracias al apoyo de muchas personas, y deseo hacer una mención especial a mi madre, Silvia, quien ha sido el pilar fundamental en mi vida y ha influido profundamente en mi desarrollo y valores.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Diseño de una API para integrar procesos de Procesamiento de Lenguaje Natural dentro del Grupo de Investigación GIHP4.

Autor(es):

John henry tenesaca criollo

Resumen

A lo largo de varios años, el Grupo de Investigación GIHP4C ha desarrollado diversas investigaciones enfocadas en el Procesamiento del Lenguaje Natural (PLN), acumulando conocimiento sobre las herramientas tecnológicas emergentes en este campo. Estas iniciativas han proporcionado una comprensión profunda de cómo las herramientas de PLN pueden emplearse para anticipar y comprender las reacciones de los usuarios en diferentes contextos. Las aplicaciones de PLN desarrolladas por el grupo ha sido implementadas de manera independiente para satisfacer necesidades específicas, para mejorar la eficiencia en el mantenimiento y la prestación de servicios de estas aplicaciones, se ha propuesto una solución integradora basada en una API.

La creación de un API Gateway centralizado ofrecerá el acceso a diversas aplicaciones de PLN, mejorando la gestión y el mantenimiento de estas herramientas. Conjuntamente, permitirá el balanceo de carga, la monitorización del flujo de datos y la creación de nuevas instancias de manera sencilla, asegurando una alta disponibilidad y rendimiento de los servicios que proporcionará a los investigadores del grupo GIHP4C la capacidad de agregar nuevos endpoints de procesamiento del lenguaje natural (PLN) permitiendo aumentar la flexibilidad y adaptabilidad de la infraestructura del grupo GIHP4C.

Se ha diseñado una arquitectura utilizando Traefik, Docker, Docker Compose, FastAPI y PostgreSQL estas herramientas fueron escogidas para implementar un API Gateway que exponga los servicios a través de un único punto de entrada así garantizando la alta disponibilidad por medio de la gestión de múltiples instancias y eliminando un único punto de fallo. El API Gateway se encargará de la orquestación y el enrutamiento de las solicitudes a los diversos servidores del grupo de investigación GIHP4C, asegurando una distribución del tráfico eficiente y segura.

La implementación del API Gateway optimiza el punto de entrada para todos los proyectos del grupo de investigación GIHP4C, mejorando la administración y el consumo de los servicios para futuros desarrollos en PLN esta implementación facilita la integración, el balanceo de carga y la monitorización, garantizando una infraestructura escalable y altamente disponible. Esto potencia la capacidad del grupo para innovar y adaptarse rápidamente a nuevas necesidades tecnológicas, mejorando la eficiencia operativa y la calidad de los servicios ofrecidos.

Palabras clave:

- Alta disponibilidad, API Gateway, Balanceo de carga, Diseño de API, Integración Continua, Monitorización de datos, Procesamiento del Lenguaje Natural (PLN)

Abstract

Over the course of several years, the GIHP4C Research Group has developed various investigations focused on Natural Language Processing (NLP), accumulating knowledge on emerging technological tools in this field. These initiatives have provided a deep understanding of how NLP tools can be employed to anticipate and understand user reactions in different contexts. Each of the NLP applications developed by the group has been implemented independently to meet specific needs at different times. To improve the efficiency of maintenance and service delivery for these applications, an integrated solution based on an API has been proposed.

The creation of a centralized API Gateway will provide access to various NLP applications, improving the management and maintenance of these tools. Additionally, it will enable load balancing, data flow monitoring, and the creation of new instances easily, ensuring high availability and performance of services. This will give the group's researchers the ability to add new NLP processing endpoints, increasing the flexibility and adaptability of the group's technological infrastructure.

To address these challenges, an architecture has been designed using Traefik, Docker, Docker Compose, FastAPI, and PostgreSQL. These tools were selected to implement an API Gateway that exposes services through a single entry point, guaranteeing high availability by managing multiple instances and eliminating a single point of failure. The API Gateway will handle the orchestration and routing of requests to the various servers of the research group, ensuring efficient and secure traffic distribution.

The implementation of the API Gateway optimizes the entry point for all of the research group’s projects, enhancing the administration and consumption of services for future developments in NLP. Furthermore, this implementation facilitates integration, load balancing, and monitoring, ensuring a scalable and highly available infrastructure. This empowers the group’s ability to innovate and adapt quickly to new technological needs, improving operational efficiency and the quality of the services offered.

Keywords:

• High availability, API Gateway, Load balancing, API Design, Continuous Integration, Data Monitoring, Natural Language Processing (NLP)

1. Introducción

Antecedentes

El Grupo de Investigación GIHP4C a lo largo de varios años, ha llevado a cabo múltiples investigaciones en el campo del Procesamiento del Lenguaje Natural (PLN) implicando en el desarrollo de diversas aplicaciones tecnológicas. Cada una de estas aplicaciones fue creada de manera independiente, enfocándose en satisfacer necesidades específicas. Esta metodología, aunque efectiva para abordar problemas inmediatos, ha generado una serie de desafíos significativos que afectan la eficiencia operativa y la conexión del ecosistema tecnológico del grupo GIHP4C.

Determinación del problema

Descripción del problema

La independencia en el desarrollo ha llevado a una fragmentación de las aplicaciones de PLN, lo que dificulta la gestión centralizada. Cada aplicación tiene su propia arquitectura, metodología de implementación y mantenimiento, lo que incrementa la complejidad administrativa y operacional. Integrar nuevas funcionalidades o tecnologías emergentes en este entorno fragmentado es un desafío significativo. La falta de una infraestructura común dificulta la escalabilidad y la capacidad de adaptación rápida a nuevas demandas tecnológicas o cambios en los requisitos del proyecto.

La ausencia de un punto de control unificado para el monitoreo y la gestión del flujo de datos entre las diferentes aplicaciones de PLN genera ineficiencias y dificultades, para garantizar un rendimiento óptimo y una alta disponibilidad de los servicios.

Los usuarios e investigadores del grupo enfrentan dificultades al interactuar con múltiples interfaces y sistemas dispares. Esto afecta negativamente la experiencia del usuario y puede ralentizar la adopción de nuevas herramientas o la integración de los datos procesados.

Estos problemas subrayan la necesidad de una solución que permita una gestión centralizada, eficiente y escalable de las diversas aplicaciones de PLN desarrolladas por el grupo GIHP4C.

La implementación de una solución integradora basada en un API Gateway es vista como un paso esencial para superar estas barreras y optimizar la infraestructura tecnológica del grupo. Esta solución no solo permite mejorar la eficiencia operativa y la calidad de los servicios, sino que también potencia la capacidad del grupo para innovar y adaptarse rápidamente a las nuevas necesidades tecnológicas.

Formulación del problema

¿Cómo puede el Grupo de Investigación GIHP4C superar los desafíos derivados de la fragmentación en el desarrollo de aplicaciones de Procesamiento del Lenguaje

Natural (PLN), para lograr una gestión centralizada, eficiente y escalable de su ecosistema tecnológico?

Justificación del problema

La fragmentación actual en el desarrollo de aplicaciones de Procesamiento del Lenguaje Natural (PLN) dentro del Grupo de Investigación GIHP4C representa un obstáculo significativo para la innovación y la prestación de servicios de alta calidad. En efecto, la proliferación de aplicaciones independientes ha generado una arquitectura dispersa y compleja, dificultando la gestión unificada de los servicios de PLN. Esta situación se traduce en una mayor carga operativa, una menor reutilizabilidad de código y una limitada capacidad para responder a las demandas cambiantes del mercado.

Con el fin de superar estas limitaciones, proponemos la implementación de una arquitectura basada en API Gateway. De este modo, se centralizará la gestión de todas las API de PLN del GIHP4C, proporcionando una interfaz unificada que abstrae la complejidad de la infraestructura subyacente. En consecuencia, se obtendrán los siguientes beneficios: mayor cohesión en la arquitectura, simplificación de la gestión, mejora de la escalabilidad y promoción de la reutilizabilidad de componentes.

La integración de un API Gateway se posiciona como una solución estratégica para optimizar los servicios de PLN del GIHP4C. Al abordar los problemas de fragmentación y complejidad, esta arquitectura permitirá al grupo acelerar el desarrollo de nuevas aplicaciones, mejorar la calidad de los servicios existentes y fortalecer su posición en la investigación. En particular, la implementación de un API Gateway permitirá al GIHP4C ofrecer servicios de PLN más robustos, escalables y adaptables a las necesidades de un entorno tecnológico en constante evolución.

Delimitación del problema

La delimitación del problema, se enfoca en la implementación de un API Gateway diseñado para proporcionar un punto de entrada centralizado y optimizar la integración de nuevas aplicaciones en el entorno de Procesamiento del Lenguaje Natural (PLN). Este API Gateway se desarrollará con el objetivo de mejorar la alta disponibilidad de los servicios existentes y futuros del grupo de investigación, permitiendo así una gestión eficiente del tráfico y garantizando la escalabilidad. Se establece claramente que el enfoque se centra exclusivamente en el diseño y despliegue del API Gateway, asegurando su operación independiente de las arquitecturas de servicios preexistentes dentro del grupo.

Objetivos

Objetivo General

Diseñar, implementar y mejorar la eficiencia de las aplicaciones de Procesamiento del Lenguaje Natural (PLN) del Grupo de Investigación (GIHP4C), a través de la creación y aplicación de una API integral.

Objetivo Específicos

- Realizar una revisión exhaustiva del estado del arte, investigando diversas aplicaciones relacionadas con el tema, con el propósito de identificar y extraer las mejores herramientas, metodologías y arquitecturas para su posterior aplicación en el desarrollo de las soluciones propuestas.

- Diseñar, desarrollar e integrar un API basado en el conocimiento adquirido durante la revisión del estado del arte. Este enfoque permitir la creación del API, manteniendo diversos patrones lógicos que faciliten la integración con las diferentes APIs de las aplicaciones de Procesamiento del Lenguaje Natural (PLN) gestionadas por el Grupo de Investigación GIHP4C.

- Evaluar e implementar el API mediante pruebas que garanticen su funcionamientó optimo. Esto se lograŕa al permitir a los investigadores del Grupo GIHP4C integrar aplicaciones de Procesamiento del Lenguaje Natural (PLN) en el API, demostrando así su correcto desempeño y funcionalidad para los usuarios finales.

Trabajos Relacionados

Agente de seguridad de microservicio basado en API Gateway en Edge Computing

Los autores [1] integraron un API gateway en plataformas de edge computing para mejorar la seguridad y autenticación de clientes. Utilizaron un agente de seguridad de microservicios basado en Kong, facilitando la autenticación, autorización y enrutamiento de solicitudes en un entorno de red privada. Implementaron el API Gateway con Kong, actuando como intermediario entre clientes y microservicios, usando tokens JWT para la autorización. Esto permitió un control de acceso eficiente sin sobrecargar los microservicios, manteniéndolos livianos, y mejoró la privacidad del cliente mediante un sistema de autenticación basado en tokens. El sistema mejoró la seguridad y eficiencia en edge computing. El uso de tokens JWT permitió una autorización efectiva sin sobrecargar los microservicios, manteniéndolos ligeros. Además, el mecanismo basado en tokens preservó la privacidad del cliente, eliminando la necesidad de manejar credenciales, mejorando la seguridad general. La implementación de un API Gateway no solo optimizaría la gestión centralizada de las aplicaciones de PLN, sino que también incrementaría la seguridad y eficiencia, tal como se ha observado en casos de plataformas de edge computing.

Análisis comparativo de Monolith, Microservice API Gateway y Microservice Federated Gateway en una aplicación basada en web que utiliza GraphQL API

Por otro lado investigadores [2] compararon tres arquitecturas de software: Monolítica, Microservicio con Patrón de Agregación de Gateway y Microservicio con Patrón de Gateway Federado. Para llevar a cabo esta comparación, se implementaron pruebas de carga y concurrencia que evaluaron las ventajas y desventajas de cada arquitectura en términos de rendimiento, complejidad y fiabilidad. Para realizar la comparación, se utilizó GraphQL para exponer un único punto de entrada para consultas, integrando las arquitecturas con patrones de gateway. Esta metodología permitió evaluar cómo cada arquitectura manejaba la complejidad y el uso de recursos mediante pruebas ajustadas de carga y concurrencia, optimizando los casos y tamaños de prueba. Como resultado, se encontró que la elección de la arquitectura influye significativamente en el

rendimiento, escalabilidad y mantenibilidad de las aplicaciones web. El estudio proporcionó una comprensión más profunda de cada arquitectura, ayudando a los desarrolladores a seleccionar la opción más adecuada según las necesidades del proyecto, basándose en parámetros como rendimiento y complejidad. La comparación de arquitecturas en este análisis permite comprender mejor las implicaciones de adoptar una arquitectura de microservicios con un API Gateway, como se ha propuesto. Este enfoque destaca el impacto positivo que puede tener en la escalabilidad, reducción de la complejidad y mejora del rendimiento de las aplicaciones de procesamiento de lenguaje natural (PLN).

Gestión de API Gateway basada en arquitectura de microservicios

Se reviso a partir del artículo [3], se analizó cómo la arquitectura de microservicios y el API Gateway pueden mejorar la gestión de servicios en comparación con la arquitectura monolítica. Implementaron el API Gateway para manejar la complejidad de las API y mejorar la interacción con microservicios, gestionando aspectos como autenticación, balanceo de carga y monitoreo.

Para lograr esto, se utilizó el API Gateway como el único punto de entrada a los microservicios, encapsulando la estructura interna del sistema y proporcionando funciones como verificación de permisos y almacenamiento en caché. Esto simplificó las interacciones entre el cliente y los microservicios, ocultando la complejidad del backend. Los resultados mostraron que el uso del API Gateway redujo la complejidad para los clientes, simplificó el código y mejoró el rendimiento general del sistema. Al centralizar la gestión y control de las API, se redujo el tiempo de respuesta del sistema y se permitió que los microservicios se concentraran en el procesamiento de la lógica de negocio. El artículo proporciona una comprensión clara de que un API Gateway no solo centralizaría la gestión de las aplicaciones de PLN, sino que también optimizaría el rendimiento y la eficiencia, simplificando la interacción con los servicios y permitiendo que las aplicaciones se concentren en su lógica de negocio.

Una estrategia de diseño de puerta de enlace API optimizada para la persistencia y el acoplamiento

En relación al siguiente articulo [4], se abordó la complejidad y costos asociados con la gestión de microservicios y APIs. Proporcionaron una solución optimizada para los API Gateways al mejorar su marco y añadir un mecanismo de persistencia en la base de datos, buscando reducir el acoplamiento y los costos de mantenimiento.

Para lograr esto, el artículo implementó un diseño de API Gateway que separa los complementos de los servicios centrales y utiliza una base de datos persistente. Esta arquitectura permite una menor complejidad y acoplamiento entre servicios, y facilita la gestión de microservicios al reducir los costos de desarrollo y mantenimiento. Los resultados demostraron que la optimización del marco y la persistencia en la base de datos mejoraron la eficiencia del API Gateway. La separación de servicios y el diseño de microservicios permitieron una reducción significativa en los costos de desarrollo y mantenimiento, además de una mejora en el rendimiento general del sistema. La estrategia de diseño optimizado del API Gateway descrita en el artículo refuerza la premisa de que implementar un Gateway no solo centralizaría la gestión de las aplicaciones de PLN, sino que también aumentaría su eficiencia, reduciría el acoplamiento y los costos de mantenimiento, y optimizaría el rendimiento a largo plazo.

Autenticación y autorización de usuario final en arquitectura de microservicio

En el articulo [5], se abordaron diferentes métodos de autenticación y autorización en sistemas de software. En un entorno monolítico, se utiliza un sistema basado en sesiones para verificar credenciales y gestionar el acceso, lo que puede ser simple pero plantea problemas de rendimiento y escalabilidad.

Para superar estos desafíos, se implementó un enfoque de gestión de sesiones distribuidas y autenticación basada en tokens. Las sesiones distribuidas incluyen técnicas como la replicación y la gestión centralizada, mientras que la autenticación basada en tokens elimina la necesidad de almacenamiento de sesiones y mejora la escalabilidad del servidor. Los resultados mostraron que el uso de autenticación basada en tokens en una arquitectura de microservicios mejora significativamente la escalabilidad y la flexibilidad. La separación del servicio de autenticación y la eliminación del almacenamiento de sesiones reducen la complejidad y los costos de mantenimiento, adaptándose mejor a entornos distribuidos. La autenticación basada en tokens y la separación del servicio de autenticación descritas en el artículo refuerzan la propuesta de implementar un API Gateway en las aplicaciones de PLN. Este enfoque no solo mejora la escalabilidad del sistema, sino que también reduce la complejidad y optimiza los costos de mantenimiento.

2. Marco teórico referencial

El API Gateway, como elemento clave en la arquitectura de sistemas distribuidos, actúa como intermediario entre los clientes y los servicios backend, ofreciendo así una capa adicional de control y seguridad. Su principal función es gestionar todas las solicitudes entrantes y salientes, facilitando el acceso a los servicios subyacentes y proporcionando funcionalidades avanzadas como autenticación, autorización, enrutamiento y monitoreo [1]. A través de estas capacidades, el API Gateway asegura una correcta gestión de las interacciones dentro del ecosistema, alineando las necesidades del grupo de investigación con los recursos del sistema.

Una vez entendido esto, el API Gateway intercepta y procesa todas las solicitudes de los clientes antes de enviarlas a los servicios backend de PLN. La arquitectura propuesta permitirá que dichas solicitudes sean gestionadas previamente con herramientas como Traefik, que habilita el balanceo de carga y el redireccionamiento de las peticiones. Durante este proceso, las solicitudes son analizadas conforme a políticas predefinidas, lo que permite ejecutar funciones como validación de datos y enrutamiento eficiente hacia los servicios correspondientes. Posteriormente, el API Gateway devuelve la respuesta al cliente, facilitando una interacción transparente entre el cliente y los servicios backend. Este enfoque centralizado permite una administración eficiente y coherente en entornos distribuidos, maximizando el rendimiento y la escalabilidad del sistema [2].

En el marco de arquitecturas basadas en microservicios, el API Gateway desempeña un papel crucial como punto único de entrada para coordinar las comunicaciones entre los diversos servicios. Nos enfocaremos en cómo el API Gateway habilita la interacción entre microservicios, y brinda características esenciales respondiendo así a la problemática del escalamiento en los proyectos de investigación de GIHP4C. Además, abordaremos aspectos críticos como la escalabilidad y la alta

disponibilidad son requisitos críticos en sistemas de producción. Este marco teórico también investigará las estrategias y técnicas más efectivas para escalar y mantener la disponibilidad del API Gateway, considerando la distribución de carga, la replicación de datos y la redundancia de servidores [5]. Así mismo, se analizarán diferentes arquitecturas de implementación de API Gateway, como las arquitecturas centralizadas, distribuidas e híbridas, evaluando su impacto en la escalabilidad, disponibilidad y rendimiento del sistema [6].

La monitorización y análisis de métricas son aspectos fundamentales para asegurar el rendimiento continuo y la disponibilidad de un API Gateway. Este estudio explorará herramientas como Traefik y Portainer que permiten la supervisión en tiempo real y el análisis detallado de métricas relacionadas con el tráfico, la latencia y el uso de recursos del sistema [7]. Además, se discutirá la implementación de API Gateways en entornos basados en contenedores como Docker y Kubernetes, destacando los beneficios en términos de portabilidad, escalabilidad y facilidad de despliegue [8].

Se examinarán los desafíos específicos que plantea la Internet de las Cosas (IoT) en la gestión de la comunicación entre dispositivos y servicios backend, y cómo la arquitectura de microservicios y los API Gateways pueden ser aplicados para abordar la integración, escalabilidad y seguridad en estos entornos [9].

Por otro lado, es importante señalar que el API Gateway desempeña un papel fundamental al considerar implementaciones en la administración de microservicios. Herramientas como NGINX y Amazon Web Services (AWS) demuestran que el API Gateway es esencial para la gestión eficiente de servicios en estos entornos, ya que proporciona un punto de entrada unificado que facilita el control de acceso, el monitoreo y la autenticación centralizada [1]. Además, diversas investigaciones en sistemas distribuidos han establecido que los API Gateways optimizan la comunicación entre servicios, permitiendo una abstracción de la infraestructura subyacente y contribuyendo a la escalabilidad, al tiempo que mantienen la consistencia en las comunicaciones internas del sistema.

A nivel de rendimiento, existen estudios que respaldan el uso de balanceadores de carga como Traefik y plataformas de contenedores como Docker, dada su capacidad para gestionar la demanda dinámica de los sistemas de microservicios y su flexibilidad en despliegues escalables [2].

En resumen, estos fundamentos teóricos demuestran la capacidad del API Gateway no solo para mejorar la eficiencia operativa, sino también para adaptarse a las necesidades cambiantes del sistema, sentando una base sólida para su adopción en entornos complejos, como el caso del grupo de investigación GIHP4C.

3. Desarrollo del proyecto

Con el avance de las tecnologías y las necesidades para ofrecer un servicio de calidad por parte de Grupo de investigación se planteó desplegar una arquitectura robusta y moderna, optimizando significativamente el rendimiento y la eficiencia. Aprovechando tecnologías avanzadas como balanceo de carga, API Gateway, bases de datos, contenedores y sistemas de monitoreo, se busca una gestión efectiva y escalabilidad dinámica de múltiples instancias, adaptándose a las necesidades específicas del grupo de investigación. Estas soluciones se explorarán en profundidad en este documento.

Fases del desarrollo del proyecto

La arquitectura basada en contenedores proporciona una visión integral de su implementación en la Figura 1, diseñada para resolver la problemática mediante la gestión de diversas fases técnicas. Estas fases están estructuradas en seis componentes principales, ofreciendo una perspectiva global del sistema.

Dichas fases son:

1. Funcionamiento del API Gateway
2. Implementación del balanceador de carga
3. Api Gateway
4. Monitor de contenedor
5. Integración de la base de datos

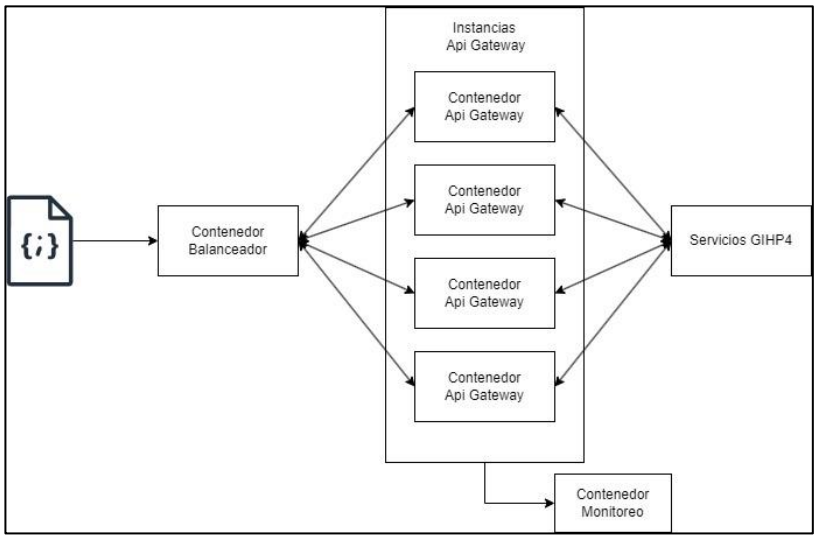


Figura 1: Diagrama arquitectura basada en contenedores

Fase 1: Funcionamiento del API Gateway

En la fase del funcionamiento se estructura en las siguientes etapas:

1. **Recepción de la Petición**:

   La recepción de peticiones está gestionada por Traefik, el cual configurado para operar a través del puerto 80, facilita el balanceo de carga de todas las solicitudes entrantes. De este modo, Traefik dirige estas peticiones hacia los contenedores individuales del API Gateway disponibles, asegurando una distribución equilibrada y eficiente del tráfico.

2. **Api Gateway**

   Una vez que las peticiones llegan al API Gateway, se inicia un proceso crucial para su correcto enrutamiento y gestión.

   Los pasos que se realizan son:

   1. **Manejo de la petición:**

      Una vez recibida la petición, se procede a gestionar su ID, el cual permite identificar el tipo de servicio solicitado. El API Gateway valida si la solicitud contiene un ID de servicio válido. Este ID es fundamental, ya que define el servicio específico al cual debe ser dirigida la petición, como se muestra en la Figura 2.

   2. **Consulta en la Base de Datos:**

      Si el Id de servicio está presente, se procede a consultar la base de datos. El API Gateway busca esta identificación en la tabla correspondiente (por ejemplo, Servicio) para obtener detalles importantes como el Id, URL, Nombre y estado del servicio.

3. **Ejemplo de Concatenación:**

   Por ejemplo, supongamos que tenemos una solicitud con el Id de servicio 123 y la URL /api/obtener/usuarios. Utilizando la información recuperada de la base de datos.

   Esto genera el API completo que apunta al recurso deseado, por ejemplo: “http://localhost/api/obtener/usuarios”.

4. **Redirección de la Petición:**

   Una vez formado el API completo, el API Gateway redirige la petición hacia el endpoint específico correspondiente al servicio encontrado en la base de datos. Este proceso asegura que cada solicitud sea dirigida correctamente al servidor adecuado, optimizando así el flujo de trabajo y la eficiencia del sistema se presenta en la Figura *2*.

   Este flujo de trabajo garantiza que el API Gateway pueda manejar dinámicamente las peticiones entrantes, basándose en la configuración almacenada en la base de datos y asegurando una respuesta rápida y precisa para cada solicitud.

5. **Monitor**

   Una vez que las peticiones han sido gestionadas, es crucial verificarlas utilizando el monitor proporcionado por Portainer, el cual muestra los logs en tiempo real de todas las peticiones entrantes al API Gateway. Este proceso asegura un funcionamiento óptimo al permitir el balanceo y la administración adecuada a través del API Gateway. Además, monitorear las peticiones garantiza que sean enviadas correctamente a los servidores del grupo de investigación, asegurando así la integridad y el rendimiento del sistema se presenta en la Figura *2*.

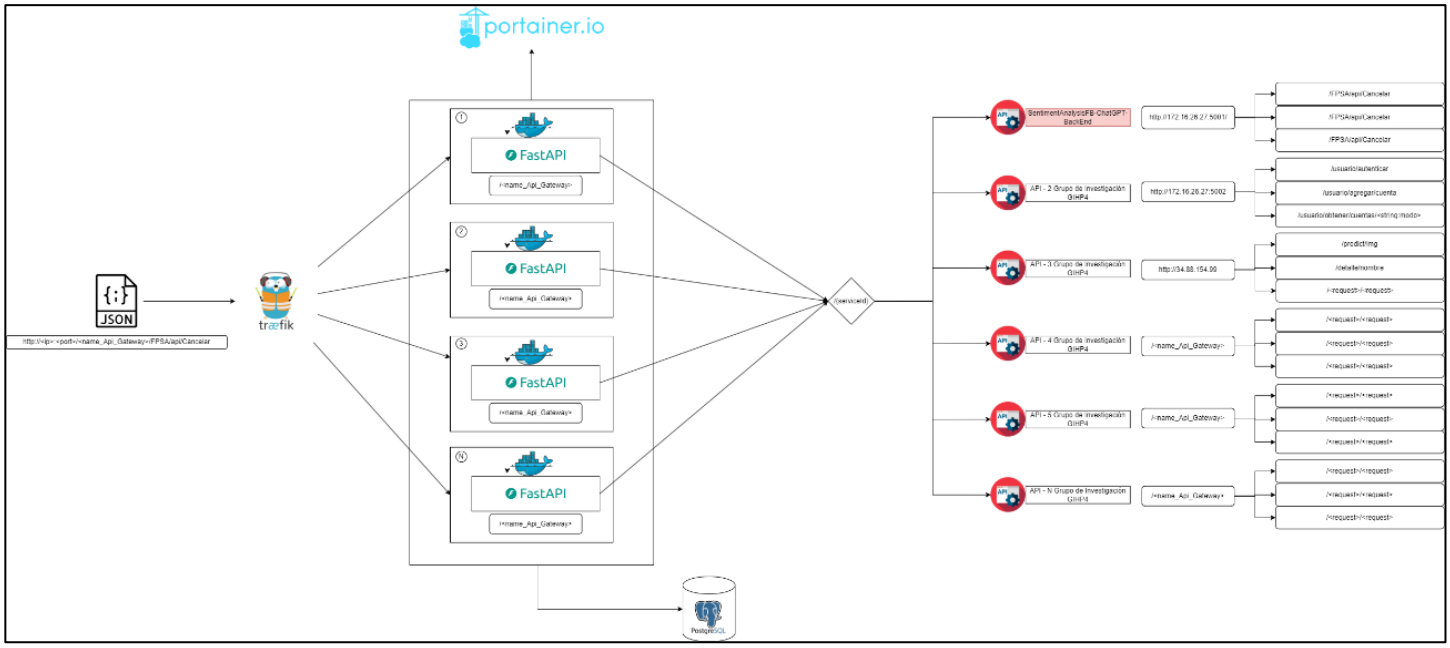


Figura 2: Arquitectura Api Gateway

La arquitectura mostrada en la Figura 2 ilustra cómo se facilita la creación de múltiples instancias sin preocuparnos por el puerto en el que se expongan. Con una arquitectura diferente, sería necesario especificar y registrar manualmente cada puerto en Traefik para lograr un balanceo de carga efectivo. Sin embargo, en esta implementación ofrece la ventaja de manejar automáticamente el registro y balanceo de cada instancia, independientemente del puerto en el que se encuentre.

Este enfoque permite desplegar cuantas instancias sean necesarias sin limitación por puerto, ya que Traefik se encarga de descubrir y gestionar dinámicamente los servicios. Adicionalmente se verifica de manera continua el estado de cada contenedor, permitiendo que solo las instancias activas reciban solicitudes, lo cual previene el envío de peticiones a instancias inactivas o caídas.

Esta capacidad de autogestión en la detección y registro de servicios no solo optimiza el balanceo de carga, sino que también garantiza una mayor disponibilidad y eficiencia en el manejo de solicitudes. Más adelante, se explorará en profundidad la justificación técnica detrás de la elección de cada herramienta en esta arquitectura.

Fase 2: Implementación del Balanceador de Carga

Para gestionar múltiples instancias y asegurar alta disponibilidad y rendimiento, se ha implementado un balanceador de carga junto con un API Gateway, tal como se muestra en la Figura 1, que presenta un diagrama básico de la arquitectura. Esta configuración optimiza el control y permite una escalabilidad dinámica de los recursos conforme a los requerimientos del Grupo de Investigación GIHP4.

Traefik facilita la arquitectura actuando como un proxy inverso y balanceador de carga su función principal es dirigir el tráfico entrante a múltiples instancias del API Gateway de manera eficiente. Permite gestionar dinámicamente las reglas de enrutamiento y la configuración de servicios, optimizando así la disponibilidad y el rendimiento de la infraestructura.

El balanceo de carga es basado en round-robin por defecto. Este método distribuye equitativamente las solicitudes de los clientes entre las diferentes instancias disponibles del servicio registrado, asegurando la distribución uniforme de la carga de trabajo.

Configuración Contenedor Traefik

Configuración para implementar Traefik como proxy inverso y balanceador de carga en un entorno Docker en el Código 1.

```yaml
traefik:
  image: traefik:v2.5
  command:
    - "--api.insecure=true"  # Habilita el dashboard de Traefik
    - "--providers.docker=true"
    - "--entrypoints.web.address=:80"
  ports:
    - "80:80"
    - "8080:8080"  # Dashboard de Traefik
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
  networks:
    - monitoring
```

Código 1: Configuración contenedor traefik

Mediante la configuración presentada en el Código 1 presentamos que **"providers.docker=true**" habilita Docker como proveedor de servicios en Traefik y agregamos el punto de entrada por el puerto 80 con **"--entrypoints.web.address=:80"**.

Destacamos el uso de Traefik, que nos permite supervisar todas las conexiones a través de "http://localhost:8080/dashboard/#/" desde esta plataforma podemos obtener información detallada del estado de las instancias.

Configuración del Balanceador Api Gateway

Para gestionar el balanceo de carga cuando tenemos múltiples réplicas y solo conocemos el puerto interno que se va a exponer, pero no el puerto externo, utilizamos la configuración de Traefik. Esta configuración nos permite mantener el control del puerto interno mientras Traefik maneja el balanceo de carga.

```
"traefik.http.services.app.loadbalancer.server.port=8000"
```

Código 2:Configuración del balanceador

Mediante el Código *2*, podemos controlar el puerto interno (en este caso, el puerto 8000) sin necesidad de conocer el puerto externo. Traefik se encarga de gestionar el balanceo de carga de manera eficiente. Todas las solicitudes entrarán a través del puerto 80, lo que facilita el despliegue y el control de las peticiones.

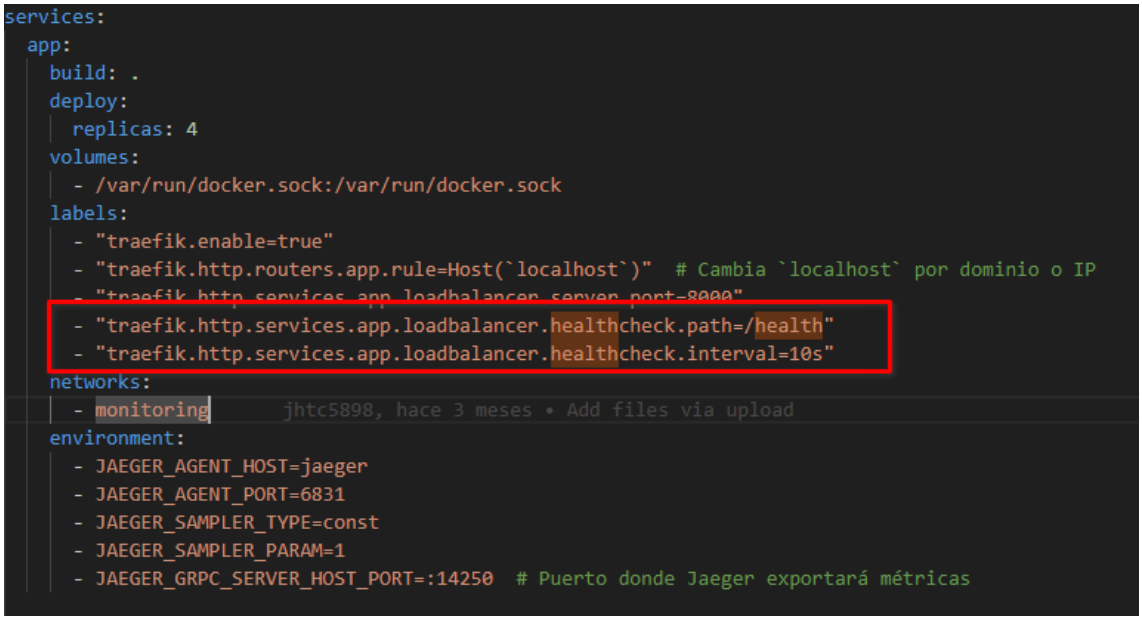
Además de configurar el balanceador de carga, es importante manejar una red (network) adecuada para garantizar la comunicación eficiente y segura entre los diferentes servicios y réplicas. La red permite que los contenedores se comuniquen entre sí de manera efectiva y facilita la gestión del tráfico interno.

Toda esta configuración de Traefik debe ser incluida en las etiquetas (labels) como se visualiza en el Código 3, el contenedor que se desea balancear. Esto asegura un funcionamiento correcto y eficiente cuando se levanten varias réplicas del mismo contenedor. Las etiquetas permiten a Traefik identificar y gestionar adecuadamente cada instancia del contenedor, facilitando el balanceo de carga y la redirección de peticiones de manera efectiva.

```yaml
app:
  build: .
  deploy:
    replicas: 4
    update_config:
      delay: 10s
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.app.rule=Host(`localhost`)"  # Cambia `localhost` por dominio o IP
    - "traefik.http.services.app.loadbalancer.server.port=8000"
  networks:
    - monitoring
```

Código 3:Código configuración de labels

Con el objetivo de garantizar la alta disponibilidad, Traefik permite una configuración avanzada para verificar la disponibilidad de las instancias. Esta configuración se aplica directamente a nivel de los contenedores como se presenta en el Código 4:



Código 4: Validación Traefik disponibilidad de instancias

Esta configuración permite la validación de instancias realizando una solicitud al servicio /health expuesto. De esta manera se garantiza que solo se incluyan instancias en funcionamiento en el proceso de equilibrio de carga. Si alguna instancia no responde, será excluida del grupo, optimizando la distribución del tráfico. Esta funcionalidad, administrada por Traefik, es crucial para mantener una alta disponibilidad y resiliencia en las arquitecturas de microservicios.

Para poner en perspectiva, cuando Traefik envía periódicamente solicitudes al punto final /health de cada instancia, aquellas que responden con un estado 200 OK permanecen en la rotación de balanceo de carga. Por el contrario, si una instancia devuelve un error o se agota el tiempo de espera, Traefik la excluye de la rotación hasta que vuelva a estar en buen estado. Este mecanismo ayuda a mantener la confiabilidad del sistema al enrutar el tráfico solo a servicios en buen estado.

Acceso Interfaz Web de Traefik

Para acceder a la interfaz web de Traefik, se debe ingresar a la URL “http://localhost:8080” en el navegador se presenta en la Figura 3.

Esta interfaz proporciona una vista detallada del estado del balanceador de carga, permitiendo ver los servicios disponibles, los routers configurados y las reglas de enrutamiento aplicadas. Permite monitorear el estado de salud de los servicios en tiempo real. Esto es especialmente útil para la administración y la resolución de problemas, ya que ofrece una visualización clara y centralizada de todas las operaciones de balanceo de carga gestionadas por Traefik.

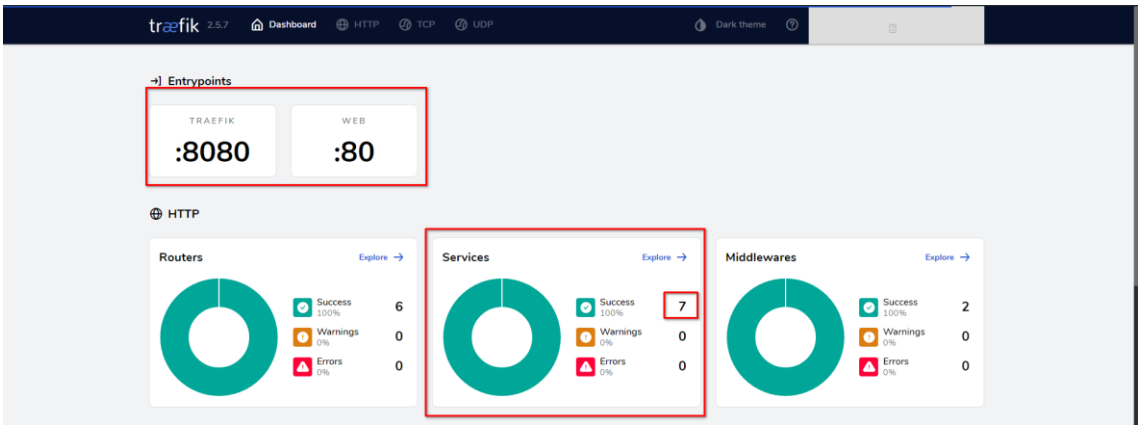


Figura 3:Interfaz Traefik

Fase 3: API Gateway

A medida que la implementación se expanda y se desarrollen nuevas aplicaciones, el grupo de investigación proyecta exponer sus hallazgos de manera que fomente el estudio y el interés de los estudiantes. Este enfoque no solo impulsará el crecimiento del grupo, sino que también cultivará una cultura de aprendizaje y colaboración en el ámbito de la investigación, fortaleciendo la conexión entre la teoría y la práctica.

En este contexto, será crucial establecer una capa de seguridad adecuada mediante el uso de HTTPS. En lugar de exponer cada uno de los servicios de forma individual, la implementación de un API Gateway operando bajo HTTPS se presenta como una solución más eficiente y segura. Esto asegurará que todos los servicios internos permanezcan protegidos durante su uso, tanto por parte de los estudiantes como de otras entidades externas que requieran acceso a la información. De este modo, se garantizará un tratamiento seguro de los datos, lo que fomentará la confianza en el uso de los recursos del grupo de investigación.
Con el tiempo, y a medida que se consolide la infraestructura, se podrá ofrecer un acceso aún más amplio y seguro a los servicios, permitiendo que el grupo se adapte a las crecientes demandas del entorno académico y profesional. Esta estrategia no solo facilitará la colaboración, sino que también sentará las bases para una implementación escalable que responda a las necesidades futuras de los estudiantes y la comunidad investigadora.

Este proceso implica generar un certificado SSL que garantice la seguridad de las comunicaciones. Sin embargo, si se manejan múltiples endpoints, será necesario crear certificados SSL adicionales para cada uno de ellos.

Esta estrategia de escalabilidad no solo permite un crecimiento ordenado de las aplicaciones, sino que también asegura que el acceso a los servicios se mantenga seguro y eficiente, alineándose con las mejores prácticas de la industria.

Tomando en cuenta el beneficio planteado para que todas las peticiones ingresen a través del puerto 80 y se distribuyan de manera balanceada entre los contenedores disponibles del API Gateway que funcionara como punto de entrada para las peticiones. Donde el API Gateway es un componente crítico que actúa como punto de entrada único para las solicitudes externas dirigidas a un conjunto de servicios internos.

Debemos tomar en cuenta que el api Gateway nunca debería manejar la lógica del negocio ya que el objetivo principal es mantener esta lógica en los servicios.
Tenemos que tener en cuenta que se mantiene varias instancias para no tener un único punto de fallo.

Tomando en cuenta como se debe manejar, su función principal es gestionar la recepción de las solicitudes del cliente, enrutar estas solicitudes al servicio adecuado, combinar las respuestas y devolverlas al cliente final.

Implementación del API Gateway

El manejo de peticiones mediante el API Gateway se realiza a través del envío de URLs específicas. El API Gateway utiliza estos URLs para identificar el punto de entrada (IdServicio) correspondiente y enrutar la petición de manera adecuada como se visualiza en la Figura 4.

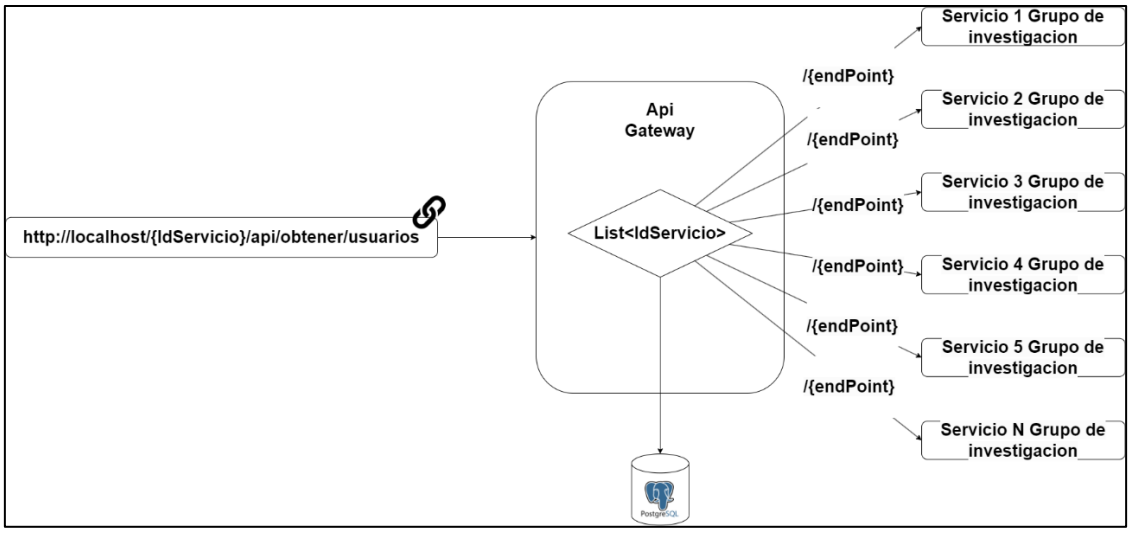


Figura 4: Api Gateway

En este ejemplo, {IdServicio} en la Figura 4 actúa como una clave que permite al API Gateway identificar cual IP del Grupo de Investigación debe dirigir la petición. Este mecanismo facilita el control y la administración de las solicitudes, asegurando que cada petición sea enviada a la IP del servidor correspondiente, gestionado adecuadamente por el grupo de investigación donde se tienen desplegados los servicios pertinentes de PLN.

El resto del URL (/api/obtener/usuarios) especifica el endpoint del API que contiene la lógica de la aplicación y define las operaciones específicas a ejecutar. Esto permite que el API Gateway no solo determine el servidor adecuado a través de {IdServicio}, sino que también dirija la petición a la funcionalidad específica requerida dentro de ese servicio.

Los beneficios que ofrece del API Gateway dentro del grupo de investigación:

- **Centralización del Tráfico**: Todas las peticiones pasan por el API Gateway, lo que centraliza la gestión y facilita el control del tráfico de red.
- **Gestión Eficiente de Microservicios**: Simplifica la comunicación y administración de microservicios, permitiendo una exposición controlada y eficiente de los servicios internos.

La planificación de la arquitectura presentada en la Figura *4* está diseñada para permitir la gestión eficiente de múltiples instancias a través de un balanceador de carga, una base de datos centralizada y un sistema de monitoreo, garantizando un servicio de alta calidad para el grupo de investigación. Este enfoque integral facilita la escalabilidad y la robustez de la infraestructura, asegurando que el sistema pueda adaptarse a futuras tendencias y demandas tecnológicas

La estructura propuesta se enfoca en una arquitectura duradera y adaptable, capaz de soportar y evolucionar con el tiempo. Para lograrlo, se han implementado diversos servicios que sientan las bases sólidas del API Gateway, proporcionando una plataforma eficiente y segura para los servicios del grupo de investigación.

En un futuro, cuando el desarrollo alcance un nivel avanzado, esta infraestructura podrá ser clave para la comercialización de las investigaciones realizadas, ampliando el impacto del grupo y su contribución al conocimiento científico y tecnológico.

Servicios Expuestos por el API Gateway

El API Gateway proporciona una serie de servicios esenciales para la gestión y administración de los microservicios en la infraestructura del grupo de investigación.

A continuación, se detallan los principales servicios expuestos y sus funcionalidades:

Crear Nuevo Punto de entrada

Objetivo: Permite crear un nuevo punto de entrada en la infraestructura del grupo de investigación.

Endpoint: POST http://localhost/nuevoServicio

Funcionalidad:

- **Adición de Nuevos Servicios**: Este servicio facilita la incorporación de nuevos puntos de entrada en la infraestructura. Los puntos de entrada pueden ser gestionados y utilizados en futuros proyectos del grupo de investigación.
- **Registro en Base de Datos**: Los usuarios pueden registrar un nuevo servicio en la base de datos. Para ello, deben proporcionar los siguientes detalles:
    - **Nombre**: Nombre del nuevo servicio.
    - **URL**: Dirección del servicio.
    - **Estado**: Estado actual del servicio (activo, inactivo, etc.).

Uso:

1. Envía una solicitud POST al endpoint especificado.
2. Incluye los detalles del nuevo servicio en el cuerpo de la solicitud.
3. El servicio se registra en la base de datos y está disponible para su gestión y uso en proyectos futuros.

Eliminar Punto de entrada

Objetivo: Permite desactivar un punto de entrada que ya no está en uso o que no debe ser expuesto sin eliminarlo permanentemente de la base de datos.

Endpoint: POST http://localhost/deleteServicio

Funcionalidad:

Desactivación del Punto de Entrada: En lugar de eliminar el punto de entrada de la base de datos, se actualiza su estado a false. Esto asegura que el servicio no sea considerado en los enrutamientos futuros.

Gestión Flexible: La desactivación permite una administración más flexible del punto de entrada, facilitando su reactivación si fuera necesario y evitando la pérdida de datos históricos relacionados con el servicio.

Uso:

1. **Enviar Solicitud**: Realiza una solicitud POST al endpoint proporcionado.
2. **Incluir Datos**: Proporciona el identificador del punto de entrada que deseas desactivar en el cuerpo de la solicitud.
3. **Actualizar Estado**: El sistema cambia el estado del punto de entrada a false, lo que impide su inclusión en el enrutamiento de solicitudes sin eliminarlo permanentemente.

Activar punto de entrada

Objetivo: Permite reactivar un punto de entrada que había sido desactivado anteriormente, restaurando su accesibilidad y funcionalidad en la infraestructura.

Endpoint: POST http://localhost/activarServicio

Funcionalidad:

- **Reactivación del Punto de Entrada**: Actualiza el estado del punto de entrada a true, permitiendo que vuelva a ser accesible y se incluya en el enrutamiento de solicitudes.
- **Restauración de Funcionalidad**: Facilita la reactivación de puntos de entrada que habían sido temporalmente desactivados, asegurando que puedan ser utilizados nuevamente según las necesidades del sistema.

Uso:

1. **Enviar Solicitud**: Realiza una solicitud POST al endpoint especificado.
2. **Incluir Datos**: Proporciona el identificador del punto de entrada que deseas activar en el cuerpo de la solicitud.
3. **Actualizar Estado**: El sistema cambia el estado del punto de entrada a true, permitiendo que el servicio sea restaurado y esté disponible para el enrutamiento de peticiones.

Editar punto de entrada

Objetivo: Permite actualizar los detalles de un punto de entrada existente en la infraestructura para reflejar cambios en la URL, nombre o estado, adaptándose a nuevas configuraciones o servidores.

Endpoint: POST http://localhost/editarServicio

Funcionalidad:

- **Actualización de Detalles**: Permite modificar la información del punto de entrada, incluyendo la URL (IP del servidor), nombre y estado, utilizando el identificador del punto de entrada.
- **Adaptación a Nuevos Servidores**: Facilita la actualización de la IP del punto de entrada, especialmente útil cuando se despliegan proyectos en nuevos servidores o se realizan cambios en la infraestructura.

Uso:

1. **Enviar Solicitud**: Realiza una solicitud POST al endpoint especificado.
2. **Incluir Datos**: Proporciona el identificador del punto de entrada junto con los nuevos detalles que deseas actualizar (URL, nombre, estado) en el cuerpo de la solicitud.
3. **Actualizar Información**: El sistema aplica los cambios solicitados y actualiza la información del punto de entrada en la infraestructura.

Obtener Todos los puntos de entrada

Objetivo: Permite obtener una lista completa de todos los puntos de entrada actualmente activos en la infraestructura, proporcionando una visión general para facilitar la administración y el monitoreo.

Endpoint: GET http://localhost/obtenerServicios

Funcionalidad:

- **Visión General**: Proporciona una lista detallada de todos los puntos de entrada disponibles y operativos en la infraestructura del grupo de investigación.
- **Administración y Monitoreo**: Facilita la gestión continua y el monitoreo en tiempo real de los puntos de entrada, ayudando a mantener la integridad y eficiencia del sistema.

Uso:

1. **Enviar Solicitud**: Realiza una solicitud GET al endpoint especificado.
2. **Recibir Datos**: Obtén una respuesta que contiene una lista de todos los puntos de entrada activos, con detalles relevantes para su administración y monitoreo.
3. **Gestionar Información**: Utiliza la lista obtenida para realizar análisis, ajustes y supervisión continua de la infraestructura.

Información del contenedor

Objetivo: Permite obtener información detallada sobre el estado y funcionamiento de los contenedores en la infraestructura, incluyendo el API Gateway y sus contenedores asociados.

Endpoint: GET http://localhost/obtenerInformacionContenedor

Funcionalidad:

- **Estado y Funcionamiento**: Proporciona información detallada sobre el estado actual del API Gateway y los contenedores relacionados, permitiendo una visión completa del entorno.
- **Balanceo de Carga**: Ofrece datos útiles en escenarios de balanceo de carga, donde es crucial conocer el estado y la asignación de las solicitudes a los contenedores específicos.

Uso:

1. **Realizar Solicitud**: Envía una solicitud GET al endpoint especificado.
2. **Recibir Información**: Obtén una respuesta que incluye detalles sobre el estado y el rendimiento de los contenedores, así como del API Gateway.
3. **Analizar y Gestionar**: Utiliza la información para supervisar el estado del sistema, diagnosticar problemas y gestionar el entorno de contenedores de manera eficiente.

Fase 4: Monitor de contenedor

El monitor del contenedor se gestiona a través de Portainer, una plataforma de administración de contenedores que proporciona una interfaz gráfica para visualizar y gestionar el estado de los contenedores.

Permite:

1. Visualización del Estado de los Contenedores:

   Portainer ofrece una interfaz gráfica intuitiva que permite monitorear el estado y el rendimiento de los contenedores en tiempo real. Proporciona información detallada sobre el uso de recursos, como CPU, memoria y red, facilitando la identificación de posibles problemas de rendimiento.

2. Acceso a la Consola del Contenedor:

   Permite acceder a la consola de los contenedores directamente desde la interfaz de Portainer. Esto es útil para realizar tareas de soporte, solucionar problemas y revisar cómo se están manejando las peticiones dentro del contenedor.

3. Monitoreo y Registro de Logs

   Portainer permite visualizar y analizar los logs generados por los contenedores. Esta funcionalidad es crucial para el diagnóstico de problemas y la auditoría de las operaciones realizadas por el API Gateway se visualiza en la Figura 5.

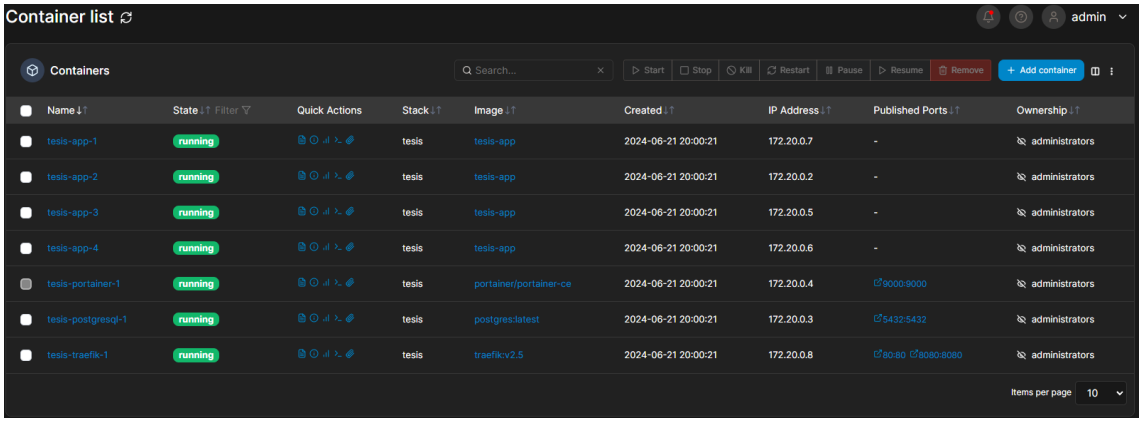


Figura 5: Interfaz Portainer

Con el propósito de asegurar una efectiva y exhaustiva monitorización de las peticiones, se han implementado Prometheus y Jaeger, herramientas esenciales para observar el comportamiento del sistema en tiempo real y gestionar arquitecturas distribuidas.

Prometheus se encarga de recopilar métricas precisas, como la latencia, la tasa de errores y el uso de recursos (como CPU y memoria) que afectan a las instancias. A través de su potente motor de consulta y un sistema de alertas configurables, se permite a los usuarios detectar rápidamente problemas de rendimiento, como cuellos de botella, que puedan estar afectando a la experiencia del usuario o la eficiencia de los servicios, se debe ingresar a la URL “http://localhost:3000/” en el navegador, se presenta en la Figura 6.

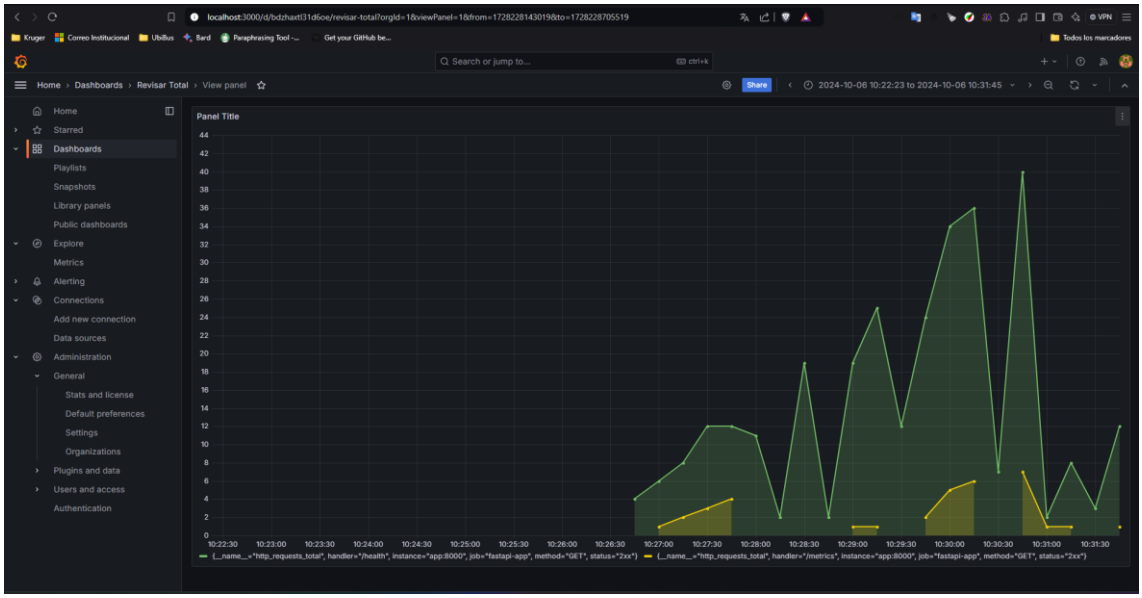


Figura 6: Métricas Prometheus

Jaeger por otra parte, está diseñado específicamente para realizar tracing distribuido, permitiendo rastrear de extremo a extremo las peticiones que atraviesan múltiples microservicios. Esto resulta invaluable cuando es necesario identificar en qué parte del recorrido de una solicitud tienen lugar retrasos o errores, algo fundamental en sistemas distribuidos complejos, se debe ingresar a la URL “http://localhost:16686/” en el navegador, se presenta en la Figura 6.

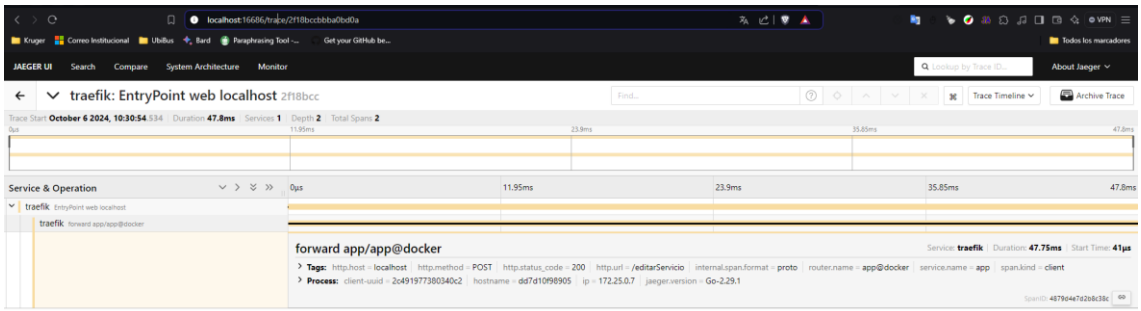


Figura 6:Interfaz Jaeger

Al combinar ambas herramientas, se obtiene una visibilidad completa ya que no sólo se monitorean métricas de alto nivel (como tiempo de respuesta y carga del sistema) con Prometheus, sino que también es posible analizar el flujo detallado de cada transacción con Jaeger facilitando la depuración y optimización de los servicios.

Fase 5: Integración de la Base de Datos

El objetivo principal de la conexión del API Gateway con la base de datos es mantener un registro centralizado de todos los puntos de entrada en la tabla Servicio. Esta tabla contiene la información detallada de todos los puntos de entrada gestionados por el grupo de investigación. Al recibir una petición, el API Gateway consulta esta tabla para validar la existencia de un idServicio relacionado en la base de datos. Una vez encontrada la correspondencia, la petición se redirige al endpoint adecuado utilizando la IP del servidor correspondiente almacenada en la base de datos.

Para la gestión de esta información, se ha optado por utilizar una base de datos PostgreSQL debido a sus capacidades avanzadas y a la libertad que ofrece al ser una solución open source. PostgreSQL proporciona un rendimiento robusto, escalabilidad y características de seguridad que son esenciales para el manejo eficiente de las peticiones y la administración de los servicios.

La tabla Servicio contiene las siguientes columnas clave se visualiza en la Figura 7:

- **id**: Identificador único del punto de entrada. Permite determinar a qué servicio se debe redirigir la petición.
- **url**: Dirección IP del servidor interno del grupo de investigación. Define el destino de la redirección.
- **nombre**: Identificador descriptivo del punto de entrada. Facilita la identificación y el manejo administrativo de los puntos de entrada.
- **estado**: Indica si el puntos de entrada está activo o inactivo. Permite al API Gateway saber si el puntos de entrada puede recibir peticiones.

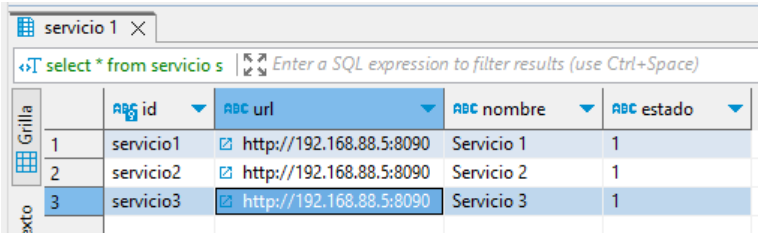


Figura 7:Tabla administrativa

4. Resultados y discusión

Una vez completada la fase de implementación de la investigación, se procedió a evaluar el uso del API Gateway, el cual actúa como intermediario entre los clientes y los servicios backend, regulando el flujo de solicitudes y mejorando el control de acceso. Esta herramienta fue esencial para optimizar la gestión de solicitudes en arquitecturas distribuidas.

Se utilizaron soluciones como Traefik y plataformas de monitoreo, como Portainer, para medir el impacto del API Gateway en términos de escalabilidad y rendimiento. Los resultados obtenidos proporcionaron una visión clara del comportamiento del sistema bajo diversas cargas de trabajo y escenarios, destacando la eficiencia del API Gateway como mediador.

Los escenarios de rendimiento y carga fueron realizados con JMeter, y se ejecutaron en un equipo con las siguientes especificaciones:

- **Sistema operativo:** Linux
- **Pruebas realizadas:** JMeter

A continuación, se presentan los distintos escenarios de prueba ejecutados:

1. **Escenario de despliegue inicial**: Se realizó la configuración y despliegue del API Gateway en el entorno del grupo de investigación, integrándolo con los servicios del sistema y verificando su funcionamiento. Se definieron configuraciones reutilizables para facilitar futuras implementaciones, asegurando tiempos de despliegue cortos y eficientes como se demuestra en la Tabla 1 los tiempos de la implementación son cortos. Este enfoque garantiza que la integración del API Gateway en nuevos proyectos sea rápida y sencilla, optimizando su uso a largo plazo.

Etapa	Descripción	Tiempo estimado	Comentarios
Preparación del Entorno	Configuración del entorno de servidor y dependencias	10 min	Incluye la actualización del sistema e instalación completa del paquete para contenedores (Docker).
Despliegue del API Gateway	Despliegue Docker del Api Gateway	5 min	Incluye el tiempo de descarga desde el repositorio y el despliegue automatizado con Docker.
Configuración del API Gateway con Servicios	Tiempo de integración por cada servicio.	10 servicios =10min	Tiempo requerido para integrar cada servicio en el API Gateway, considerando la adición de 10 servicios. Tiempo individual de 1 minuto.
Pruebas de Funcionamiento	Pruebas de carga y funcionamiento del API Gateway.	20 min	Pruebas de los servicios para la administración del API Gateway.
Despliegue Final	Publicación y activación en producción.	30 min	Pruebas en producción manejo de las peticiones.
	Total:	**1 hora 15 min**	

Tabla 1:Tiempos estimados para el despliegue del API Gateway

2. **Escenario de consumo de servicios PLN**: En este escenario, se evaluó el rendimiento del API Gateway al gestionar solicitudes para las aplicaciones de Procesamiento de Lenguaje Natural (PLN) que maneja el Grupo de Investigación. En la Tabla 2, se destaca el tiempo de respuesta del API Gateway al recibir y enviar la petición, excluyendo el tiempo que requiere el servicio PLN. Esto permite obtener datos precisos sobre la gestión que realiza el API Gateway.

Aspecto Evaluado	**Descripción**	**Métrica**	**Resultado**
Volumen de Solicitudes	Número total de solicitudes procesadas durante la prueba.	Cantidad (número)	1000
Tiempo de Respuesta	Tiempo promedio que tarda el API Gateway en responder	Milisegundos (ms)	100ms
Tasa de Éxito de Solicitudes	Porcentaje de solicitudes exitosas frente al total.	Porcentaje (%)	100%
Carga Concurrente	Número máximo de solicitudes simultáneas gestionadas.	Cantidad (número)	100

Tabla 2:Consumo de servicios PLN

3. **Escenario de alta disponibilidad:** Se sometió al sistema a pruebas de alta disponibilidad con la finalidad de asegurar que permanezca accesible y operativo incluso en caso de fallos o interrupciones.

   Traefik realiza verificaciones periódicas, consultando el endpoint /health de cada contenedor cada 10 segundos. Esto permite evaluar el estado de las instancias y detectar posibles fallos. En caso de que una instancia experimente algún error, Traefik la excluye del balanceador de carga hasta que la verificación sea exitosa nuevamente. Basando en esto los resultados en el escenario de alta disponibilidad se resume en baja probabilidad de fallo como se presenta en la Tabla 3 .

Prueba	Descripción	Métrica	Resultado Esperado	Resultado Obtenido
Prueba de Failover Manual	Simulación de la caída de una instancia del API Gateway.	Tiempo de redirección (segundos)	Menos de 5 segundos	4 segundos
Prueba de Failover Automático	Redirección automática en caso de fallo.	Tiempo de redirección (segundos)	Menos de 2 segundos	3 segundos
Prueba de Carga	Evaluación del rendimiento bajo alta carga de solicitudes simultáneas.	Solicitudes por segundo (RPS)	Sin degradación del servicio	Sin degradación del servicio
Consistencia de Respuestas	Validación de respuestas entre instancias durante el failover.	Porcentaje de coincidencias (%)	100% de coincidencias	100% de coincidencias

Tabla 3: Resultados escenario alta disponibilidad

5. Conclusiones

En conclusión, el desarrollo del API Gateway para el Grupo de Investigación GIHP4C representa un avance significativo en la eficiencia y gestión de aplicaciones distribuidas. Este API ha creado un punto de entrada único, mejorando la organización de las solicitudes y optimizando el rendimiento del sistema. Además, ha facilitado la integración de múltiples microservicios, aportando características críticas para una interacción efectiva.

La revisión exhaustiva del estado del arte ha sido esencial para identificar herramientas y metodologías que promuevan el desarrollo de soluciones eficientes. En particular, la elección de Traefik como balanceador de carga y proxy inverso ha sido clave para gestionar el tráfico de manera dinámica. Esta herramienta no solo asegura alta disponibilidad, sino que también maximiza el uso de recursos, permitiendo que el sistema se adapte a picos de demanda sin sacrificar el rendimiento.

Por otro lado, la implementación de contenedores mediante Docker ha mejorado la portabilidad y escalabilidad del sistema, simplificando el despliegue y la gestión de aplicaciones. Estos elementos, combinados, han permitido que el Grupo de Investigación aborde de manera efectiva sus desafíos operativos, proporcionando una capa adicional de control que facilita la gestión de interacciones en un ecosistema cada vez más interconectado.

A futuro, la infraestructura permite que el grupo pueda expandirse fácilmente, aprovechando la capacidad de Traefik para gestionar múltiples instancias sin restricciones de puerto. Esta capacidad de autogestión, combinada con un balanceo de carga eficiente, no solo optimiza la disponibilidad y el uso de los recursos, sino que también asegura una infraestructura sólida y adaptable. La implementación actual sienta las bases para futuros desarrollos en el procesamiento del lenguaje natural, impulsando tanto la innovación como la eficiencia en la investigación.

A medida que el API Gateway crezca para atender un mercado más amplio y se requieran tiempos de respuesta más cortos, es recomendable integrar Redis como una capa de caché para almacenar en memoria los destinos de las peticiones. Esto permitiría redirigirlas de forma eficiente, evitando búsquedas innecesarias y reduciendo significativamente los tiempos de respuesta. Asimismo, para una gestión más robusta de seguridad y autenticación basada en tokens JWT, se recomienda implementar Keycloak como servidor de identidad. Keycloak centralizaría la autenticación y el control de acceso, asegurando que las peticiones se manejen de manera segura y conforme a los permisos establecidos, facilitando también la integración con Redis para almacenar tokens activos y validar rápidamente cada petición entrante.

En resumen, la implementación del API Gateway no solo transforma la gestión de aplicaciones en el Grupo de Investigación GIHP4C, sino que también sienta las bases para futuros desarrollos en el campo del procesamiento del lenguaje natural, impulsando la innovación y la eficiencia en la investigación.

6. GLOSARIO

- **API (Interfaz de Programación de Aplicaciones)**: Conjunto de reglas y protocolos que permite la interacción entre diferentes aplicaciones de software.
- **API Gateway**: Una capa que actúa como un punto de entrada único para gestionar y dirigir el tráfico hacia múltiples APIs, facilitando la integración y la gestión centralizada.
- **PLN (Procesamiento del Lenguaje Natural)**: Subcampo de la inteligencia artificial que se centra en la interacción entre computadoras y humanos a través del lenguaje natural.
- **Fragmentación**: Situación en la que múltiples aplicaciones operan de forma independiente, dificultando su integración y gestión centralizada.
- **Arquitectura**: Estructura organizativa de un sistema que define su comportamiento y cómo interactúan sus componentes.
- **Metodología de implementación**: Conjunto de procedimientos y técnicas utilizadas para desarrollar y desplegar aplicaciones.
- **Mantenimiento**: Proceso de gestión de las actualizaciones y correcciones de un sistema o aplicación para asegurar su correcto funcionamiento.
- **Complejidad administrativa**: Dificultades en la gestión y supervisión de aplicaciones o sistemas que tienen múltiples componentes o requieren diversas configuraciones.
- **Escalabilidad**: Capacidad de un sistema para manejar un crecimiento en la carga de trabajo, aumentando su capacidad sin comprometer el rendimiento.
- **Infraestructura común**: Conjunto de recursos compartidos que permiten la operación y gestión de aplicaciones y servicios.
- **Rendimiento óptimo**: El funcionamiento más eficiente y efectivo de un sistema o aplicación, que maximiza su capacidad y recursos.
- **Alta disponibilidad**: Diseño de un sistema que asegura su funcionamiento continuo y sin interrupciones, incluso en caso de fallos.
- **Interfaz de usuario**: El medio a través del cual un usuario interactúa con una aplicación, incluyendo el diseño y la experiencia del usuario.

- **Innovación**: Proceso de desarrollo de nuevas ideas y aplicaciones que mejoran un producto o servicio existente.
- **Cohesión**: Medida de cuán estrechamente están relacionados los componentes de un sistema; una alta cohesión implica que los componentes trabajan bien juntos.
- **Reutilizabilidad de código**: Capacidad de utilizar componentes de software en diferentes aplicaciones sin necesidad de reescribir el código.
- **Pruebas**: Métodos utilizados para garantizar que una aplicación o sistema funcione correctamente y cumpla con los requisitos establecidos.
- **Despliegue**: Proceso de poner una aplicación o sistema en funcionamiento, haciéndolo accesible para los usuarios.
- **Gestión del tráfico**: Supervisión y control del flujo de datos que circula por un sistema o red.
- **Ecosistema tecnológico**: Conjunto de tecnologías y aplicaciones interrelacionadas que operan dentro de un entorno específico.

Referencias

1. Xu, R.; Jin, W.; Kim, D. Microservice Security Agent Based On API Gateway in Edge Computing. *Sensors* **2019**, *19*, 4905. https://doi.org/10.3390/s19224905
2. K. Adrio, C. N. Tanzil, M. C. Lianto and Z. E. Rasjid, "Comparative Analysis of Monolith, Microservice API Gateway and Microservice Federated Gateway on Web-based application using GraphQL API," 2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), Palembang, Indonesia, 2023, pp. 654-660, doi: 10.1109/EECSI59885.2023.10295809. keywords: {Microservice architectures;Computer architecture;Logic gates;Throughput;Software;Time measurement;Time factors;software architecture;GraphQL API;monolithic;Microservices;software testing analysis},
3. Zhao, J. T., Jing, S. Y., & Jiang, L. Z. (2018, September). Management of API gateway based on micro-service architecture. In *Journal of Physics: Conference Series* (Vol. 1087, No. 3, p. 032032). IOP Publishing. https://iopscience.iop.org/article/10.1088/1742-6596/1087/3/032032/meta
4. Zuo, X., Su, Y., Wang, Q., & Xie, Y. (2020). An API gateway design strategy optimized for persistence and coupling. *Advances in Engineering Software*, *148*, 102878. https://www.sciencedirect.com/science/article/abs/pii/S0965997820304452
5. He, X., & Yang, X. (2017, October). Authentication and authorization of end user in microservice architecture. In *Journal of Physics: Conference Series* (Vol. 910, No. 1, p. 012060). IOP Publishing. https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012060/meta
6. "Applied-ML," GitHub, [Online]. Available: https://github.com/eugeneyan/applied-ml. [Accessed: 03-Aug-2024].
7. P. Singh, "Python Natural Language Processing Cookbook," Packt Publishing, 2020. [Online]. Available: https://www.packtpub.com/product/python-natural-language-processing-cookbook/9781838987312. [Accessed: 03-Aug-2024].
8. "NLP Progress," GitHub, [Online]. Available: https://github.com/sebastianruder/NLP-progress. [Accessed: 03-Aug-2024].

9. “AWS Lambda y API Gateway,” Amazon Web Services, [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 03-Aug-2024].
10. Reactive Programming, “API Gateway - Reactive Programming,” 2023. Disponible en: https://www.reactiveprogramming.com/api-gateway
11. ICHI.PRO, “Microservicios Spring Boot - API Gateway,” 2023. Disponible en: https://www.ichi.pro/microservicios-spring-boot-api-gateway
12. M. Adams and C. Kunz, “Microservices and API Gateway,” 2018. Disponible en: https://scholar.google.com/scholar?q=Microservices+and+API+Gateway
13. Patel, “Security in API Gateway,” 2018. Disponible en: https://scholar.google.com/scholar?q=Security+in+API+Gateway
14. Newman, “Scalability and High Availability in API Gateway,” 2015. Disponible en: https://scholar.google.com/scholar?q=Scalability+and+High+Availability+in+API+Gateway
15. Richardson, “API Gateway Implementation Architectures,” 2016. Disponible en: https://scholar.google.com/scholar?q=API+Gateway+Implementation+Architectures
16. Singh, “Monitoring and Metrics in API Gateway,” 2016. Disponible en: https://scholar.google.com/scholar?q=Monitoring+and+Metrics+in+API+Gateway
17. Newman, “Container-based API Gateway Development,” 2015. Disponible en: https://scholar.google.com/scholar?q=Container-based+API+Gateway+Development
18. Richardson, “Microservices and API Gateway Architectures in IoT,” 2016. Disponible en: https://scholar.google.com/scholar?q=Microservices+and+API+Gateway+Architectures+in+IoT