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:

ARTÍCULOS PROFESIONALES DE ALTO NIVEL

TEMA:

DISEÑO DE UNA ARQUITECTURA DE LA OBSERVABILIDAD EN MICROSERVICIOS PARA INSTITUCIONES FINANCIERAS

AUTORES:

EDISSON PAÚL REDROVÁN REYES
MARLON ENRIQUE ULLOA AMAYA

DIRECTOR:

MIGUEL ÁNGEL QUIROZ MARTÍNEZ

CUENCA – ECUADOR
2026

Autores:

Edisson Paúl Redrován Reyes
Ingeniero de Sistemas.
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Cuenca.
eredrovan@est.ups.edu.ec

Marlon Enrique Ulloa Amaya
Ingeniero de Sistemas.
Candidato a Magíster en Software por la Universidad Politécnica Salesiana – Sede Cuenca.
eredrovan@est.ups.edu.ec

Dirigido por:

Miguel Ángel Quiroz Martínez
Ingeniero en Sistemas.
Máster Universitario en Ciencias y Tecnologías de la Computación.
Máster en Ingeniería con Especialidad en Sistemas de Calidad y Productividad.
mquiroz@ups.edu.ec

Diseño de una Arquitectura de la Observabilidad en Microservicios para Instituciones Financieras

Autor(es):

Edisson Paúl Redrován Reyes
Marlon Enrique Ulloa Amaya

Resumen

Este trabajo propone desarrollar una arquitectura de microservicios equipada con telemetría avanzada, pensando especialmente en las necesidades de las instituciones financieras, donde la observabilidad del sistema y la capacidad de mantener operaciones estables son aspectos críticos. Lo que buscó fue comprobar si la propuesta realmente funcionaba y qué tan bien se desempeñaba, para lo cual, se construyó un prototipo donde se puso a prueba en varios escenarios controlados que intentaban replicar situaciones del mundo real. Para llevar a cabo el experimento, se diseñaron y armaron cuatro configuraciones distintas del sistema. Cada una tenía diferentes niveles de instrumentación y los servicios estaban conectados entre sí de maneras diferentes. Los resultados obtenidos evidencian que cuando el sistema contaba con telemetría, se pudo detectar problemas mucho antes de que se volvieran graves, hacer un seguimiento detallado de cada transacción desde el inicio hasta el final, y lo más importante, reducir drásticamente el tiempo que tomaba entender qué estaba fallando cuando algo salía mal. Y todo esto sin que el sistema en general perdiera velocidad o eficiencia. En cambio, cuando probamos las versiones sin telemetría, las cosas se complicaban bastante. Era mucho más difícil localizar dónde estaba el problema exactamente, y cuando surgía algún imprevisto, la respuesta del equipo operativo era notablemente más lenta. Después de analizar todos estos resultados, se concluye que la arquitectura propuesta sí es factible desde el punto de vista técnico, puede crecer según las necesidades y es lo suficientemente sólida como para implementarse en bancos y otras entidades financieras. Lo que aporta es una mejora real en dos frentes: por un lado, permite entender mucho mejor qué está pasando dentro del sistema en cada momento, y por otro, facilita enormemente el cumplimiento de todas las regulaciones y requerimientos operativos que son tan estrictos en este sector.

Abstract

This work proposes the development of a microservices architecture equipped with advanced telemetry, designed especially for the needs of financial institutions, where system observability and the ability to maintain stable operations are critical aspects. The aim was to verify whether the proposal actually worked and how well it performed. To this end, a prototype was built and tested in several controlled scenarios that attempted to replicate real-world situations. To carry out the experiment, four different system configurations were designed and assembled. Each had different levels of instrumentation, and the services were connected to each other in different ways. What we found was quite revealing: when the system had telemetry, we were able to detect problems long before they became serious, track each transaction in detail from start to finish, and most importantly, drastically reduce the time it took to understand what was going wrong when something went wrong. And all this without the overall system losing speed or efficiency. In contrast, when we tested the versions without telemetry, things became quite complicated. It was much more difficult to pinpoint exactly where the problem was, and when something unexpected arose, the response from the operations team was noticeably slower. After analyzing all these results, we concluded that the proposed architecture is technically feasible, can grow according to needs, and is robust enough to be implemented in banks and other financial institutions. It provides real improvement on two fronts: on the one hand, it allows for a much better understanding of what is happening within the system at any given moment, and on the other, it greatly facilitates compliance with all the regulations and operational requirements that are so strict in this sector.

Palabras clave: Microservicios; Observabilidad; Telemetría; Istio; Prometheus; Grafana; Kiali; Kubernetes; Monitoreo de rendimiento; Sector financiero.

1. Introducción

La creciente complejidad de los sistemas basados en microservicios ha elevado la importancia de implementar mecanismos efectivos de observabilidad, especialmente en entidades financieras donde la disponibilidad, seguridad y rendimiento de los servicios son críticos. La dificultad radica en que, a medida que las arquitecturas de microservicios escalan, se vuelve más desafiante monitorear, diagnosticar y optimizar el comportamiento del sistema en tiempo real, afectando la experiencia del usuario y la estabilidad operativa (Borges et al., 2024c).

En sistemas basados en microservicios, una sola petición puede atravesar múltiples servicios desarrollados y desplegados de forma independiente, lo que dificulta la detección y el diagnóstico de fallos (Soldani and Brogi, 2022a). Cuando el monitoreo se configura de manera incompleta o con supuestos erróneos, algunas fallas quedan ocultas y aumentan tanto la latencia como los costos operativos (Borges et al., 2024b). La supervisión tradicional, centrada en métricas aisladas y registros no correlacionados resulta insuficiente frente a la complejidad de las dependencias y de las comunicaciones en tiempo de ejecución, de ahí la necesidad de enfoques de observabilidad que hagan visibles las relaciones causa efecto (Madupati, 2023).

La urgencia de este problema es mayor en el sector financiero, donde interrupciones breves pueden traducirse en pérdidas económicas y daños reputacionales significativos. En este contexto, la observabilidad no es un complemento, sino un mecanismo para sostener la confiabilidad operativa y acelerar la recuperación ante incidentes (Borges et al., 2024a; Mahida, 2024). Pese a ello, muchas decisiones de diseño siguen basándose en la intuición y en prácticas heredadas, sin métodos comparables para valorar su efectividad ni para justificar sus costos de adopción (Borges et al., 2024c).

La observabilidad no solo mejora el rendimiento técnico, si no que contribuye a la trazabilidad exigida por normativas financieras (como PCI DSS e ISO 27001), convirtiéndose en un factor estratégico para la sostenibilidad operativa.

Actualmente, las instituciones financieras carecen de un marco unificado que les permita correlacionar métricas, trazas y logs en arquitecturas distribuidas, lo que dificulta la detección oportuna de fallos y el cumplimiento de requisitos regulatorios. Este estudio busca diseñar una arquitectura de observabilidad en microservicios que aborde este vacío. Se implementa un prototipo funcional que integra monitoreo de métricas, trazabilidad distribuida y análisis operativo, y se evalúa su desempeño en términos de consumo de recursos, visibilidad extremo a extremo y facilidad de integración en clústeres modernos. El estudio compara configuraciones con distintos niveles de instrumentación y acoplamiento para estimar, bajo carga controlada, su impacto en detección temprana de anomalías, localización de errores y tiempos de diagnóstico. Como contribución principal, se ofrece un marco práctico y reproducible para decidir qué señales recolectar, cómo correlacionarlas y qué compromisos aceptar entre costo de instrumentación y beneficio operativo.

2. Estudios Previos

A lo largo de los últimos años, varios investigadores han venido trabajando en diferentes formas de mejorar la observabilidad cuando se trabaja con arquitecturas de microservicios, y esto ha cobrado particular importancia en sectores donde no hay margen para el error, como es el caso del financiero. Para poder presentar un panorama claro de lo que se ha hecho hasta ahora, se ha organizado la revisión de estos trabajos previos agrupándolos por temáticas principales.

2.1 Panorama empírico e industrial

Waseem et al. (2021) presentan un estudio práctico basado en 106 encuestas a profesionales, y también 6 entrevistas a expertos, con un enfoque en patrones como API Gateway y Backend-for-Frontend, métricas operacionales (uso de recursos, balanceo de carga) y prácticas como logs centralizados, monitoreo de excepciones y

health-checks como principales para gestionar la complejidad operativa. Así también, Li et al. (2022) realizan encuestas de prácticas industriales e identifican retos persistentes como la sobrecarga de datos y la complejidad de instrumentación, ofreciendo una visión realista de la adopción en producción. Complementariamente, Steidl et al. (2024) sistematizan prácticas de la industria para detección de anomalías en operación, destacando métricas y patrones recurrentes en contextos de alta complejidad tecnológica.

De manera similar, Soldani and Brogi (2022b) comparan enfoques de detección de fallos (trazas, logs, métricas e IA), destacando su aplicabilidad en sectores regulados. En esa línea, Faseeha et al. (2025) proponen una taxonomía de frameworks de observabilidad que clasifica soluciones por tipo de señal y nivel de monitoreo. Como base conceptual, Widerberg and Johansson (2022) sintetizan los fundamentos de observabilidad cloud-native y herramientas clave (p. ej., Prometheus y Kubernetes).

2.2 Stacks y arquitecturas cloud-native para observabilidad

Varios trabajos documentan implementaciones prácticas en Kubernetes. Bennett and Whitaker (2022) describen un stack con Prometheus y Grafana, abordando escalabilidad y entornos efímeros; Mensah and Ofori (2023) detallan pipelines con Fluentd y Loki (recolección, transformación y visualización de logs) para grandes volúmenes. En arquitecturas con service mesh, Nunes et al. (2020) integran Istio, Jaeger, Prometheus y Kiali (validado en OpenShift OKD), diseño extrapolable a finanzas por sus exigencias de escalabilidad y cumplimiento. En paralelo, Kratzke (2022) analizan casos de uso cloud-native donde el uso consistente de logs y trazas mejora diagnóstico, auditoría y resiliencia.

2.3 Marcos integradores y trazabilidad de extremo a extremo

Pensando en cadenas de petición complejas, Lee et al. (2023) proponen un marco integral que combina métricas, trazas y logs para aislar fallos de punta a punta y acelerar el diagnóstico bajo SLA estrictos. En la misma línea, Wang et al. (2022) presentan KMamiz, solución no intrusiva sobre Kubernetes/Istio/Zipkin que construye grafos de dependencia y evalúa calidad arquitectónica con foco en trazabilidad precisa.

2.4 Detección de anomalías y localización de causa raíz

Métodos basados en trazas: Yu et al. (2023) introducen TraceRank, un algoritmo inspirado en PageRank que explota trazas para construir grafos de dependencia y localizar servicios defectuosos. De forma complementaria, Li et al. (2021) emplean correlaciones sobre trazas para reducir ruido y acelerar troubleshooting, mientras Yang et al. (2023) incorporan semántica de tipo de petición y endpoint para mejorar la precisión de causa raíz. Para escenarios a gran escala, Liu et al. (2021) proponen MicroHECL, una herramienta eficiente para localización de fallos en sistemas extensos.

Aprendizaje automático con enfoques multimodales: Desde IA explicable, Akmeemana et al. (2025) presentan GAL-MAD, basado en Graph Attention Networks para detección de anomalías con interpretabilidad, alineada con exigencias regulatorias. En el espectro no supervisado, Liu et al. (2020) emplean redes bayesianas profundas para detectar anomalías sin etiquetado previo. Integrando múltiples señales, Du et al. (2018) combinan métricas y trazas para diagnóstico reactivo en entornos con contenedores, y Gupta (2021) muestran cómo orquestar OpenTelemetry en Kubernetes junto con técnicas de IA para detectar degradaciones (latencia, dependencias).

2.5 Optimización de instrumentación, muestreo y costo operacional

La eficiencia operacional es abordada por Chen et al. (2024), quienes introducen TraceMesh para muestreo orientado a eventos raros, mejorando precisión con ahorro de recursos. En cuanto a impacto, Nõu et al. (2025) cuantifican la sobrecarga de la trazabilidad distribuida, insumo clave para decisiones informadas bajo alta carga. Con un enfoque adaptativo, Pathak et al. (2024) presentan SALO, que ajusta dinámicamente el nivel de logging (sidecars/controladores) reduciendo volumen sin perder trazabilidad. Para evaluar trade-offs, Borges et al. (2024c) proponen Oxn, herramienta que inyecta fallos y ajusta configuración de observabilidad para medir efectos cuantitativos.

Como caso aplicado, Okpako et al. (2025) realizan una implementación con Java/Spring Boot y Docker que integra Zipkin para mejorar la trazabilidad entre microservicios, con beneficios directos en tiempo de depuración y rendimiento que se puede transferir a sectores financieros.

3. Metodología

Este estudio se apoya en un prototipo experimental que simula condiciones operativas de una institución financiera. Sobre esa base se implementó y validó una arquitectura de observabilidad para microservicios, poniendo a prueba bajo escenarios de alta disponibilidad y baja latencia sus capacidades de monitoreo, trazabilidad y diagnóstico de fallos; además, se ejecutaron rutas de tráfico y picos de carga para observar cómo respondían los componentes ante situaciones límite. El estudio de caso no solo comprobó el desempeño técnico, sino que permitió contrastar la propuesta con un contexto afín al de las entidades financieras marcado por requisitos regulatorios y operativos, para valorar con mayor precisión su pertinencia y aplicabilidad en escenarios reales.

En primera instancia, se realizó un análisis de la literatura en bases de datos académicas de alto impacto, como SCOPUS, cuidando tanto la amplitud como la relevancia de los resultados. Para garantizar la exhaustividad y la pertinencia, además, se empleó una cadena de búsqueda que combinó términos centrales y sinónimos, es que incluir variantes como visibility o insight evita dejar fuera trabajos cercanos en enfoque pero distintos en vocabulario, se utilizó la siguiente cadena de búsqueda:

```
("observability" OR "monitoring" OR "visibility" OR "insight") AND
("architecture" OR "framework" OR "design" OR "structure") AND
("microservices" OR "service-oriented" OR "distributed systems" OR
"cloud-native") AND
("financial" OR "banking" OR "finance" OR "financial services") AND
("institutions" OR "organizations" OR "entities" OR "companies")
```

La búsqueda arrojó 33 artículos. Tras aplicar criterios de exclusión de artículos que no estén alineados con los objetivos del trabajo se descartaron estudios centrados exclusivamente en sistemas monolíticos o en Big Data sin componente de observabilidad, se seleccionaron 24 trabajos para análisis detallado. Esta fase permitió identificar las tecnologías y prácticas más recurrentes del ecosistema de observabilidad y ofreció las bases conceptuales para el diseño de la arquitectura.

El análisis de la literatura y los requisitos concluyó que un enfoque basado en Service Mesh (Istio + Envoy) combinado con pipelines de telemetría mediante OpenTelemetry Collector y Fluent Bit resultaba óptimo, ya que desacopla de manera eficiente la recolección de datos de su almacenamiento y análisis.

Por otro lado, el diseño se propone seguir el Modelo C4 (Contexto, Contenedores, Componentes y Código, en este caso, Deployment), el cual permite representar el sistema en diferentes niveles de abstracción, facilitando la comprensión de la arquitectura propuesta por diferentes stakeholders.

- Context View (Nivel 1): Se definió la interacción entre los roles clave (Site Reliability Engineer, Desarrollador, Oficial de Seguridad) y la Plataforma de Observabilidad, estableciendo los límites y el contexto del sistema.
- Container View (Nivel 2): Se identificaron los principales componentes (Contenedores): Prometheus (métricas), Jaeger (trazas), Loki/Tempo

(logs/trazas) y Grafana (visualización), y su despliegue como pods en el clúster Kubernetes.

- Component View (Nivel 3): Se detalla la composición funcional de OpenTelemetry Collector (receivers, processors, exporters), responsable de normalizar, enriquecer y encaminar señales (metrics, traces, logs) hacia los backends correspondientes. Esta capa permite desacoplar a los productores de telemetría de las herramientas de almacenamiento y análisis, aspecto relevante para cumplir requisitos de auditoría y segregación de datos en el dominio financiero.
- Deployment View (Nivel 4): Se mapean los contenedores a un despliegue reproducible en Minikube, agrupando las instancias a través de el namespace y aplicando configuraciones declarativas mediante manifiestos y Helm charts. De esta manera, la topología no solo sigue buenas prácticas si no que también permite aproximar a un entorno productivo en aspectos clave como el aislamiento entre servicios.

El diseño final se basa en principios de modularidad y separación de responsabilidades para que cada componente se aísle y escale de forma independiente, enfocándose en una función específica. La arquitectura resultante garantiza resiliencia, flexibilidad y control, condiciones necesarias para entornos financieros de alta disponibilidad. Independientemente, la plataforma tiene los siguientes componente por separado: Istio para obtener telemetría a nivel de malla; Prometheus para métricas; Tempo para trazas; Loki para logs; y Grafana/Kiali para visualización, consulta y diagnóstico. La combinación de estas piezas soporta la correlación entre señales y reduce el acoplamiento entre servicios de aplicación y herramientas de observabilidad.

Se consolidó así un diseño orientado a la resiliencia, en el que cada componente puede escalar de forma independiente y la telemetría fluye con seguridad hacia distintos backends. Además, esta separación de responsabilidades facilita ajustes finos sin mover el resto del sistema; por ejemplo, disponer de pods dedicados a la

observabilidad para las métricas, sin intervenir los pods de los microservicios observados.

Con esa base, se pasó a la implementación en un entorno local con Minikube, emulando un clúster de Kubernetes productivo. Esta elección permitió recrear un escenario controlado de microservicios financieros con balanceo de carga, aislamiento por namespaces y despliegues reproducibles de los componentes de observabilidad. Al trabajar de esta manera, se pueden realizar cambios, observar su impacto, latencia, consumo, tasas de error y revertir con precisión cuando hace falta.

El despliegue de la arquitectura se realizó en distintos pasos:

1. Orquestación y malla de servicios: Se instaló Istio en modo ambient mesh, habilitando el plano de control (istiod) y el ingress gateway. Los sidecars Envoy permitieron la recolección de métricas y trazas de forma no intrusiva.
2. Automatización de instalación: Se emplearon Helm Charts y manifiestos YAML para la instalación de Prometheus, Grafana, Jaeger, Loki y Tempo. Esta estrategia garantiza reproducibilidad, versionado y facilidad de actualización de los componentes.
3. Pipeline de telemetría: El OpenTelemetry Collector fue configurado con:
   3.1. Receivers: OTLP y Prometheus.
   3.2. Processors: normalización de métricas y anonimización de datos sensibles.
   3.3. Exporters: Prometheus (métricas), Loki (logs) y Tempo (trazas).
4. Plataforma de visualización: Grafana se implementó como punto central de observación, integrando dashboards personalizados con métricas financieras clave, consultas PromQL, logs filtrados por servicio y correlación de trazas distribuidas.

Cada componente fue ejecutado en pods independientes dentro del namespace observability. La comunicación entre servicios se aseguró mediante mTLS (mutual TLS) habilitado en Istio, reforzando el alineamiento con requerimientos regulatorios del sector financiero.

3.1 Entorno de pruebas

Diseño experimental. Se empleó un diseño A/B para medir el impacto de la observabilidad en tiempo de ejecución: (A) Baseline sin malla ni canalización de telemetría, y (B) Observabilidad con Istio (mTLS) + OpenTelemetry + Prometheus/Loki/Tempo. Ambas condiciones usaron el mismo clúster, versiones y recursos. El warm-up fue de 20 s y el perfil de carga se ejecutó con k6 (stages: 30 s a 200, 60 s a 500, 30 s a 0). Los endpoints evaluados corresponden al flujo GET /productpage del sistema de referencia.

4. Resultados

Tras la implementación del prototipo, se validó que la arquitectura propuesta demostró ser plenamente funcional, cumpliendo con los requerimientos establecidos durante la fase de análisis y diseño. La estructura definida permitió corroborar la coherencia entre los objetivos planteados y el comportamiento observado en el entorno experimental, evidenciando que la solución satisface los principios de observabilidad, escalabilidad y resiliencia considerados desde su concepción.

Como resultado del proceso de diseño arquitectónico, se obtuvieron las siguientes vistas estructuradas según el modelo C4:

Figura 1: Diseño de Contexto

En la Figura 1 se muestra el diseño de contexto, el cual representa la visión más general del sistema, mostrando su posición dentro del ecosistema organizacional y las interacciones con actores y sistemas externos. En esta vista se identifican los principales participantes: el usuario final, los sistemas externos de monitoreo y observabilidad (por ejemplo, Prometheus y Grafana), y los servicios empresariales del dominio financiero que generan los datos de interés.

El sistema central basado en el colector de observabilidad y el service mesh actúa como intermediario, capturando, procesando y distribuyendo la información entre los diferentes módulos. Esta representación facilita comprender los límites del sistema y las dependencias externas que influyen en su operación, garantizando que el diseño cumpla con los principios de acoplamiento débil e interoperabilidad controlada.

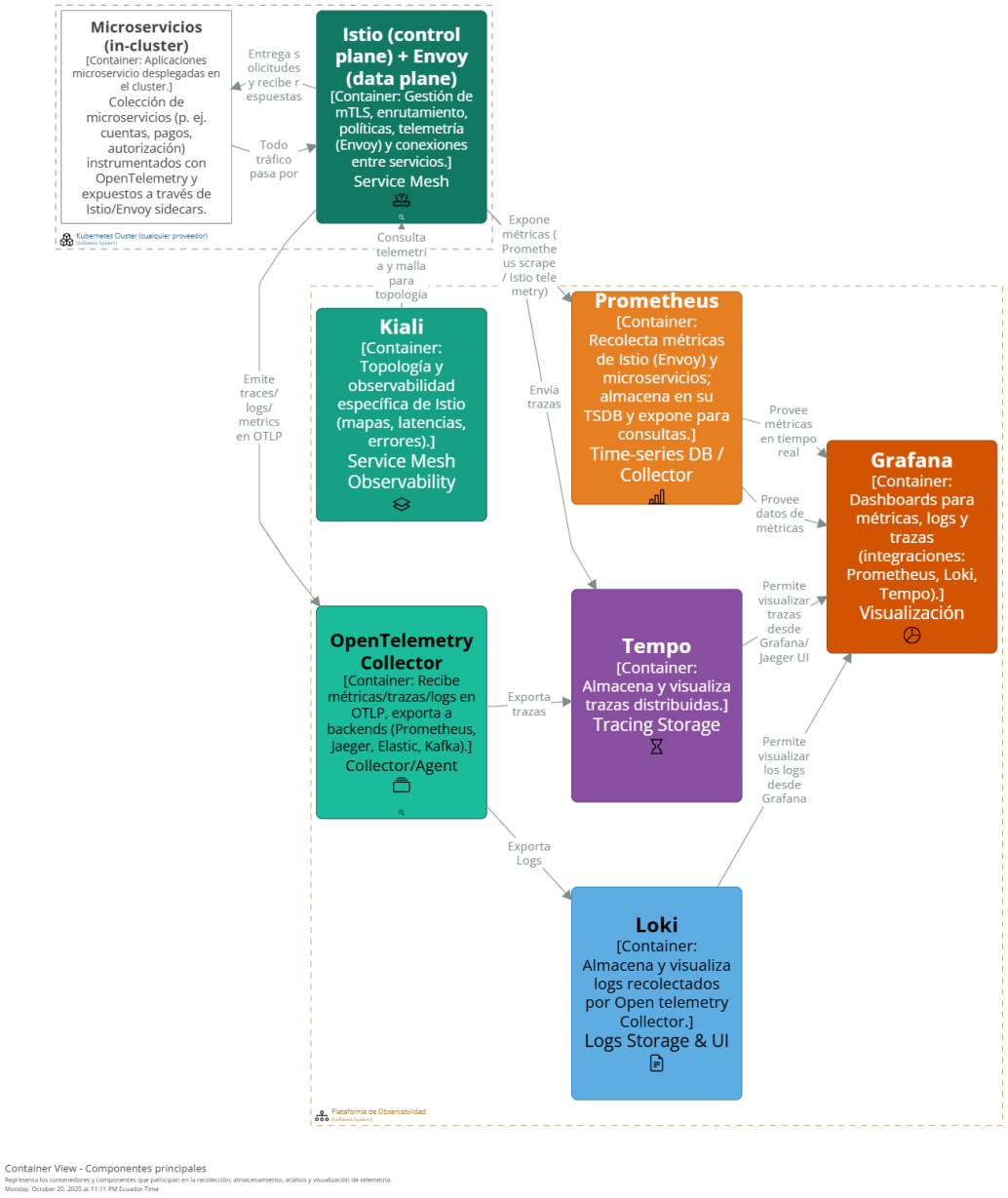

Figura 2: Diseño de Contenedores

Por otro lado, en la Figura 2, se hace referencia al diseño de contenedores que descompone el sistema en sus principales unidades lógicas de ejecución, cada una con una responsabilidad claramente definida. Entre los contenedores más relevantes se encuentran:

- El OpenTelemetry Collector, encargado de recibir, procesar y exportar las métricas y trazas provenientes de los distintos servicios.

- El Service Mesh (Istio), que implementa el control de tráfico, la autenticación mutua y la comunicación segura entre microservicios mediante el uso de proxies Envoy.
- Los Servicios de Dominio Financiero, que representan las aplicaciones principales del sistema y generan la información transaccional y de negocio.
- El Almacén de Configuración y Telemetría, donde se definen las políticas de tráfico, seguridad y observabilidad, y que se encuentra soportado en Kubernetes mediante Custom Resource Definitions (CRDs).
- Los Servicios de Monitoreo Externos, como Prometheus y Jaeger, los cuales consumen los datos exportados para visualización y análisis.

Esta vista permite observar las fronteras tecnológicas, los protocolos de comunicación (gRPC, HTTP/REST, mTLS) y las dependencias que existen entre los módulos. La segmentación propuesta impulsa una arquitectura desacoplada, escalable y portable dentro del entorno Kubernetes.

Además, las Figuras 3 y 4, representan el diseño de componentes, es decir, profundizan en la estructura interna de los contenedores principales, destacando la interacción entre los módulos funcionales que conforman el sistema.

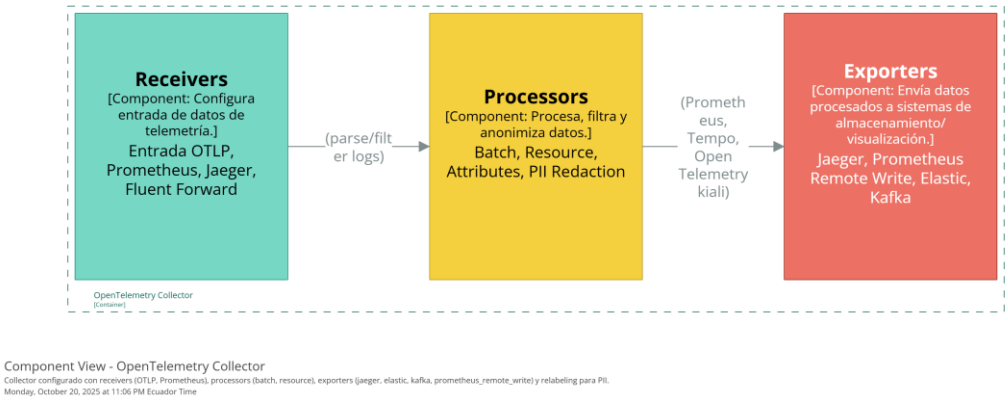

Figura 3: Diseño de Componentes de Open Telemetry

En el caso de OpenTelemetry Collector de la Figura 3, se detallan sus tres componentes fundamentales:

- Receivers, que se encargan de recibir los datos de observabilidad en diferentes formatos (OTLP, Prometheus, Jaeger, entre otros).
- Processors, los cuales son los responsables de transformar, agregar o filtrar los datos antes de su envío.
- Exporters, envían la información procesada hacia los sistemas de visualización externos.

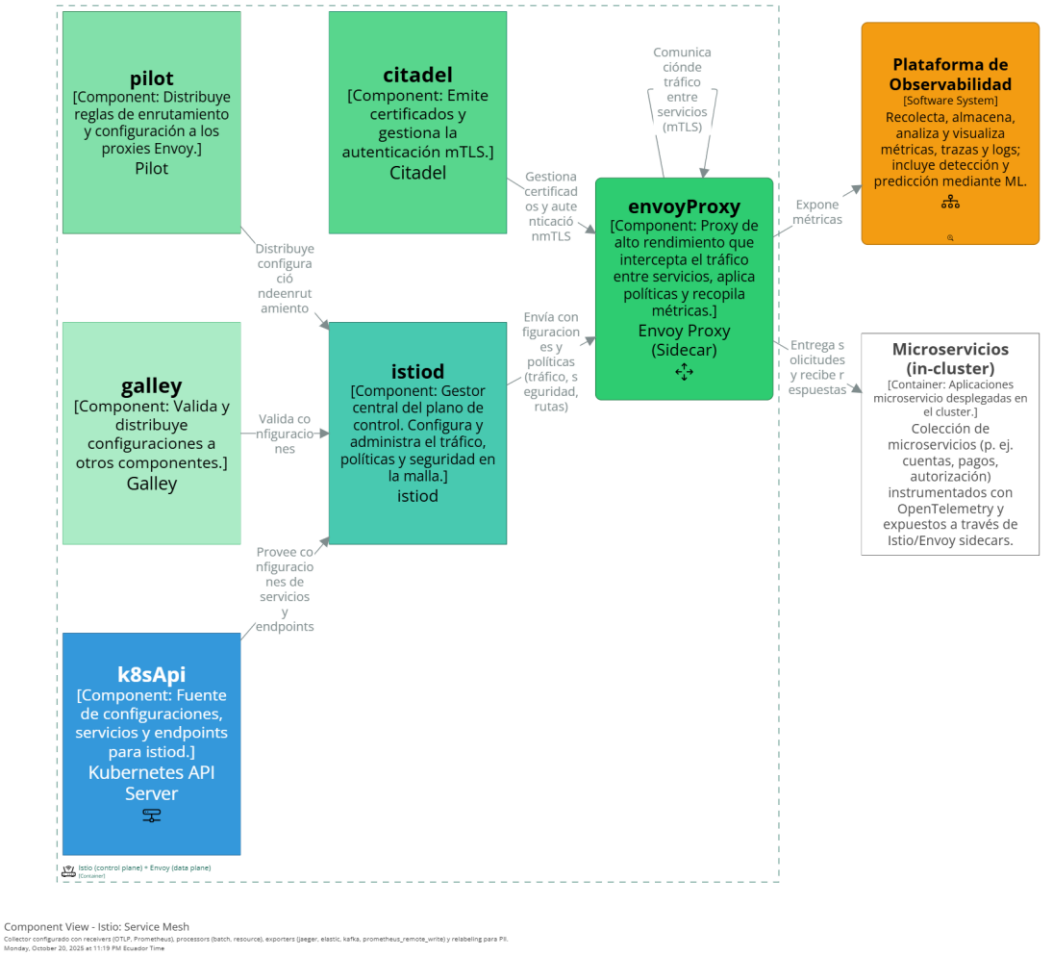

Figura 4: Diseño de Componentes de Istio

Por otro lado, Istio Service Mesh que se muestra en la Figura 4 contiene dos planos operativos:

- Control Plane, compuesto por Istiod, Pilot, Citadel y Telemetry, los cuales gestionan la configuración, autenticación y políticas de seguridad.

- Data Plane, conformado por los proxies Envoy Sidecar, que se encargan de interceptar y enrutar el tráfico entre los microservicios.

Estas figuras permiten comprender cómo los componentes interactúan para lograr una observabilidad integral del sistema, asegurando trazabilidad, resiliencia y control sobre las comunicaciones en tiempo real.

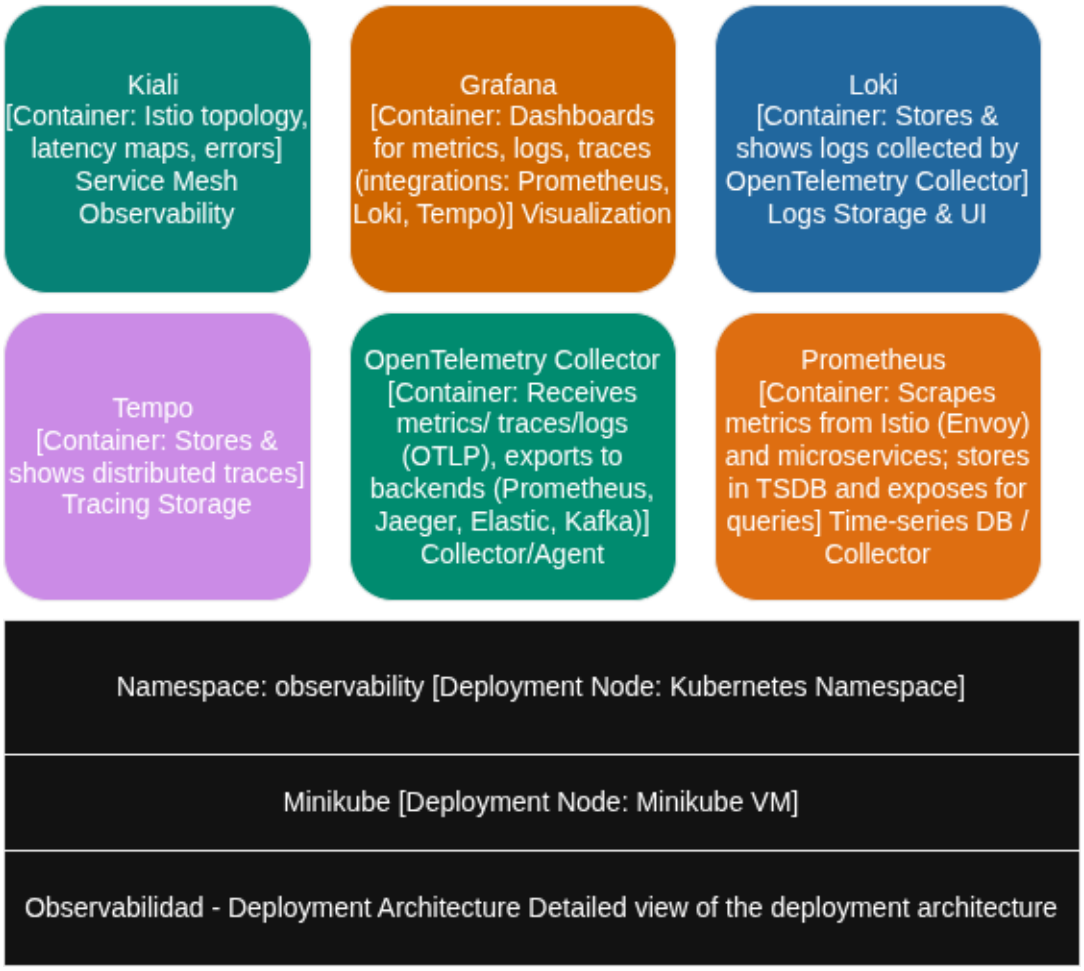

Figura 5: Diseño de Despliegue

Por último, se consideró necesario representar cómo quedaría realmente desplegado todo el sistema en un ambiente de ejecución. Para esto, se diseñó la arquitectura de despliegue sobre un clúster de Kubernetes, tal como se muestra en la Figura 5. La Figura 5 muestra cómo se distribuyeron los servicios en entorno de ejecución: están los servicios propios del dominio financiero, el colector que se

encarga de la observabilidad, y los proxies Envoy, cada uno corriendo en su propio pod y comunicándose entre ellos a través de la red interna del clúster.

Con este esquema de despliegue se logran dos cosas importantes: primero, se puede ver claramente cómo los componentes de infraestructura se relacionan con las partes lógicas del sistema, y segundo, queda evidente cómo todo lo diseñado en las etapas previas se traduce en algo concreto y funcional. Gracias a esta configuración se pudo integrar todo el prototipo de manera ordenada y ejecutarlo de forma controlada, asegurándose de que en cualquier momento se puede hacer seguimiento a los flujos de datos y medirlos sin problema.

Además, la evaluación de la arquitectura se realizó en torno a tres dimensiones principales: correctitud funcional, rendimiento y escenarios específicos del dominio financiero.

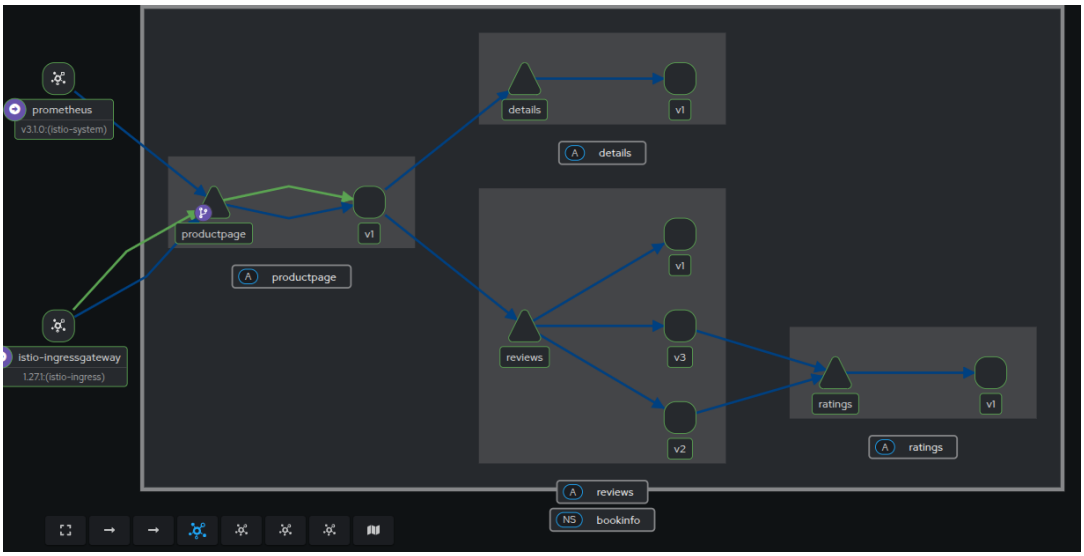

Figura 6: Grafo de tráfico en Kiali

La Figura 6 muestra la herramienta Kiali que se utilizó para validar la topología y el enrutamiento de del aplicativo a probar. El grafo de tráfico muestra las dependencias entre los diferentes servicios, así como el ingress y Prometheus. Además de la estructura, Kiali resume el caudal (req/s) y las tasas de éxito/error por arista, lo que permitió confirmar que el flujo nominal se mantiene estable y que los

fallos inyectados se reflejan como incrementos de 4xx/5xx en los enlaces afectados. Esta vista fue útil para localizar rápidamente el servicio impactado y acotar el dominio de análisis antes de revisar trazas detalladas.

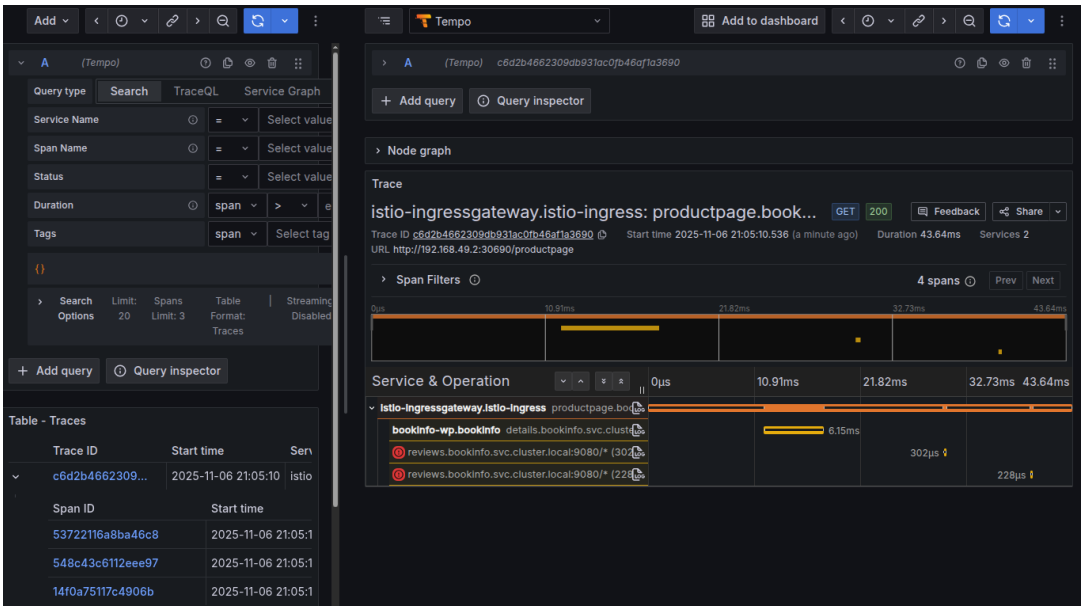

Figura 7: Traza de petición en Grafana/Tempo

La Figura 7 muestra que con Grafana/Tempo se inspeccionaron las trazas. Ante una falla inducida en el camino entre uno de los servicios de el aplicativo, la traza evidenció la ruta critica y resaltó el servicio afectado con estado de error. Esta información permitió establecer una correlación directa entre el pico de errores observado en Kiali y la operación concreta que los originó. Cuando se dispone de logs en este caso via Loki/OTel, la navegación entre trazas y logs se facilita cerrar el bucle causal con los mensajes de aplicación del mismo evento.

Correctitud funcional. Se validó la recolección, almacenamiento y visualización de telemetría en sus tres categorías:

- Métricas: capturadas por Prometheus y visualizadas en Grafana (CPU, memoria, latencia por microservicio).
- Logs: centralizados en Loki, con búsqueda por etiquetas de servicio y severidad.

- Trazas: recopiladas en Tempo, permitiendo detectar cuellos de botella en transacciones que involucran múltiples microservicios.

Rendimiento y eficiencia. Se generaron cargas de trabajo controladas mediante bucles de curl y pruebas con K6, midiendo:

- Latencia promedio por petición (ms).
- Throughput del sistema (req/s).
- Consumo de CPU y memoria de los pods de Prometheus, Loki y Collector.
- Tiempo de respuesta de consultas PromQL y de búsqueda de logs.

Los resultados preliminares indican que la incorporación de Istio introduce un overhead promedio del 7–10 % en la latencia, consistente con lo reportado en la literatura. No obstante, este costo se justifica por el nivel de visibilidad y control ganado.

Se registraron: (i) Throughput (req/s) a partir de k6.http_reqs.rate; (ii) Latencia (ms) promedio, mediana y percentiles p90/p95 desde k6.http_req_duration; (iii) Tasa de éxito (códigos 2xx) vía k6 check_status_200. Adicionalmente (no mostrado en la tabla) se recolectaron métricas de recursos (CPU/mem) de istio-proxy, otel-collector, prometheus, loki para cuantificar el coste de observabilidad.

La Tabla 1 resume los resultados agregados por condición. El overhead relativo de latencia se calcula como (p95B − p95A) / p95A × 100 %, y el cambio de throughput como (ThrB − ThrA) / ThrA × 100 %.

Tabla 1: Comparativa A/B con k6 (A = línea base, B = observabilidad)

Métrica	A	B	Dif. abs.	Delta%
Throughput (req/s)	279,08	269,31	−9,77	−3,50%
Latencia promedio (ms)	845,19	881,39	36,20	4,28%

Mediana (ms)	832,10	736,01	−96,09	−11,55%
p95 (ms)	1625,60	2069,68	444,08	27,32%
Tasa de fallos (%)	0,0000	0,0062	0,0062	
CPU (mCPU)	1,77	2,98	1,207	68,18%
Memory (MiB)	785,50	792,62	7,13	0,90%

Nota. La tabla presenta la comparación entre los escenarios A (línea base) y B (con observabilidad integrada) empleando k6. Se muestran los cambios en las métricas de rendimiento y latencia en términos absolutos y porcentuales.

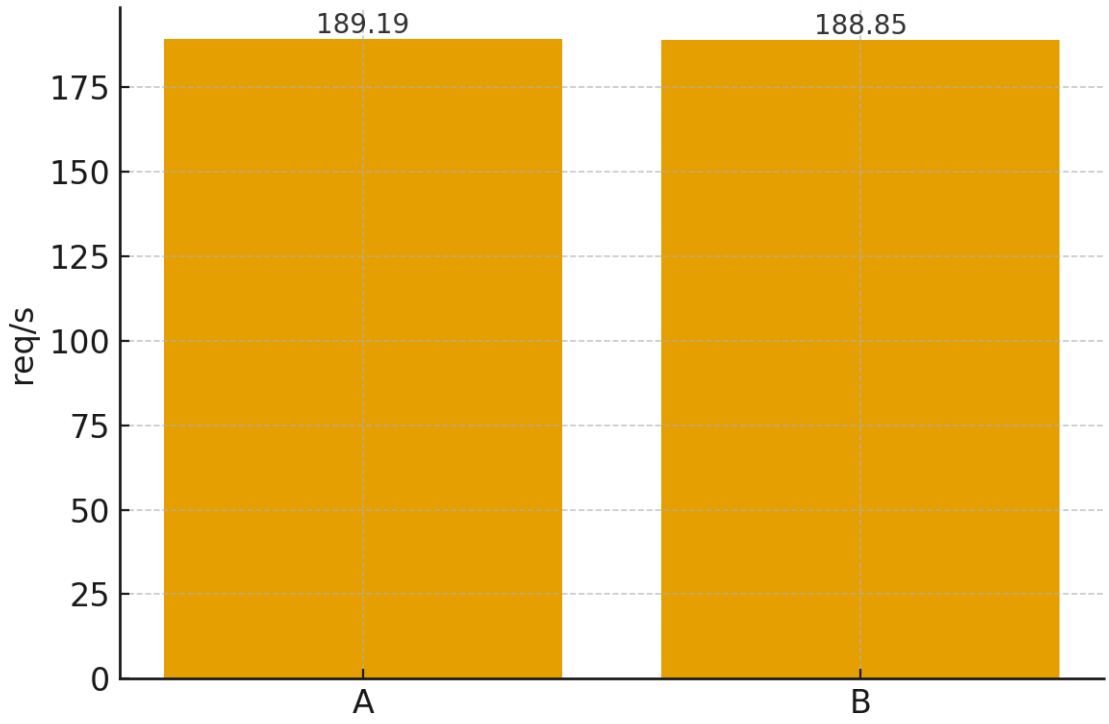

Figura 8: Comparación escenario A vs B - Throughput

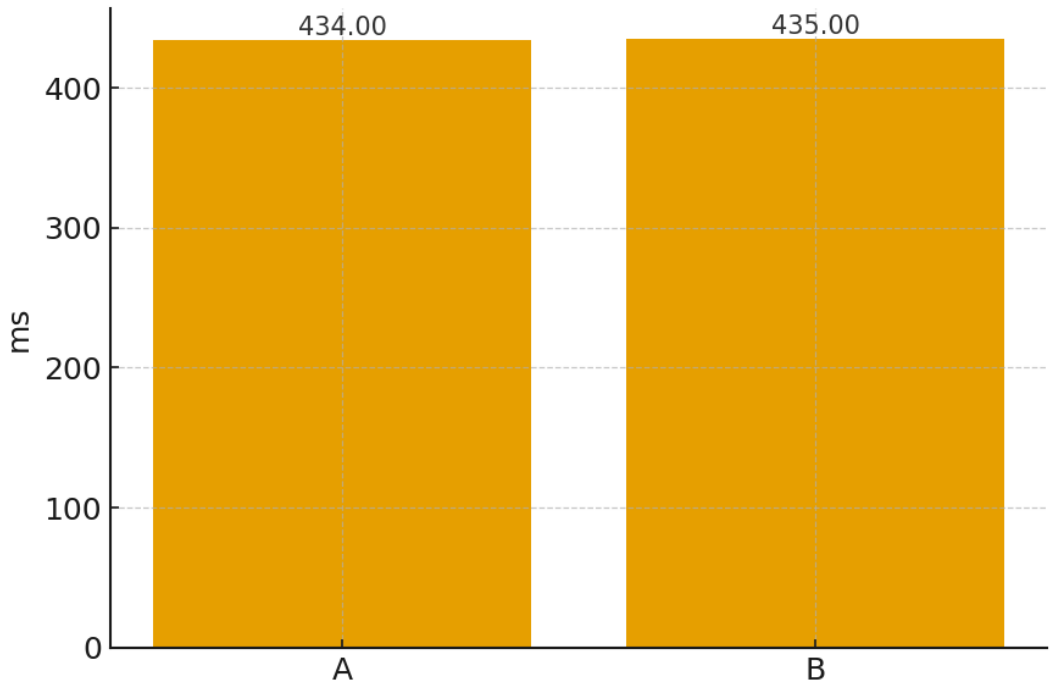

Figura 9: Comparación escenario A vs B - Latencia

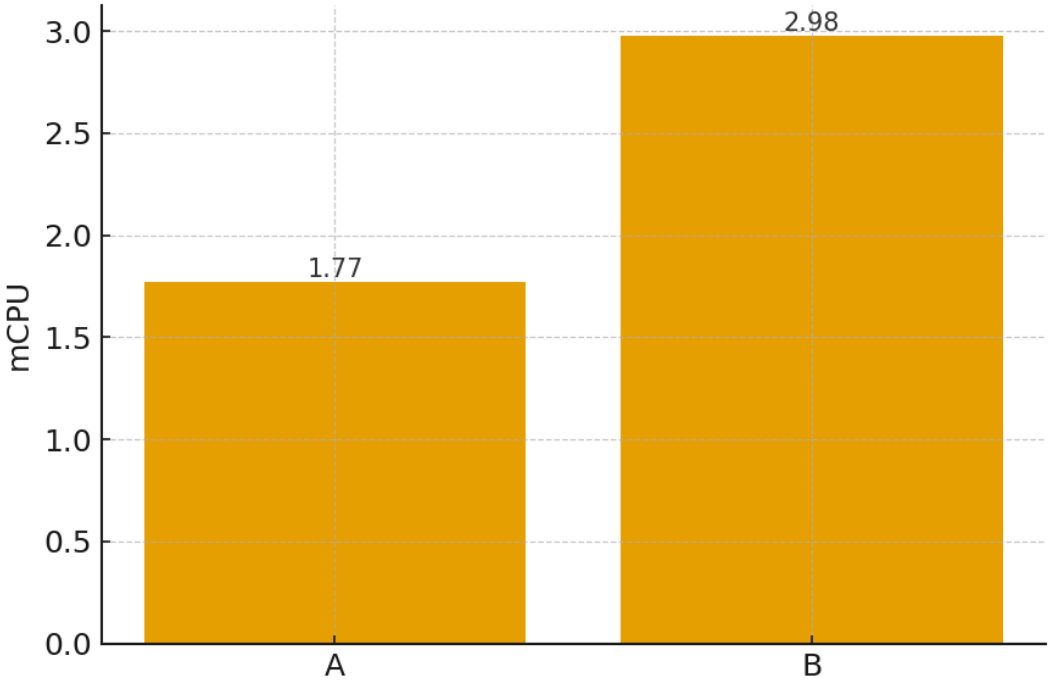

Figura 10: Comparación escenario A vs B - CPU

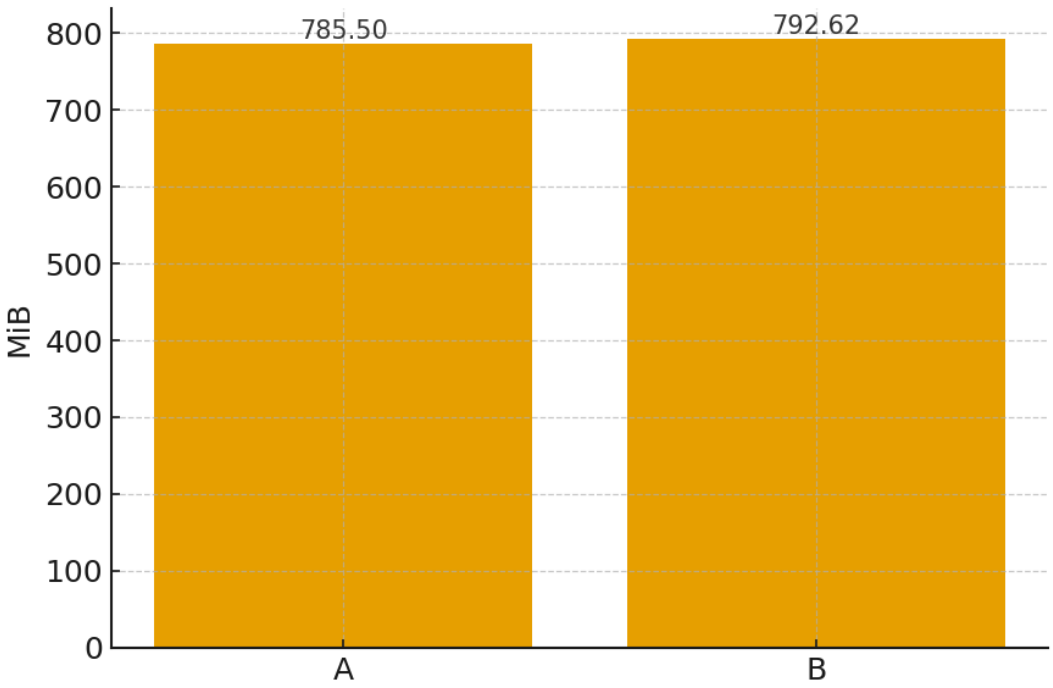

Figura 11: Comparación escenario A vs B - Memoria

En conjunto, las Figuras 8–11 muestran que habilitar la observabilidad preserva el throughput dentro de un margen estrecho, mientras que el efecto se concentra en la latencia de cola (p95), más sensible a la instrumentación que las métricas centrales. El aumento en el uso de CPU y el moderado aumento en el consumo de memoria son consistentes con el procesamiento y transporte de la telemetría, esto no indica un consumo desproporcionado de recursos del sistema. Desde la perspectiva operativa, estos hallazgos respaldan la adopción de la observabilidad, ajustando con cautela las políticas de tiempo de espera, reintento y muestreo de trazas, especialmente cuando los SLO se establecen en percentiles altos.

Interpretación. La activación de la observabilidad mantuvo un throughput similar (variación de −3,5 %), con mejora en la mediana (−11,55 %) pero un incremento de colas altas (p90 +19,4 %, p95 +27,3 %), y un ligero aumento de la media (+4,3 %). Se observaron dos respuestas no exitosas en (B) (0.006 %), mientras que (A) no registró fallos. Estos resultados son consistentes con el costo esperado del plano de datos y la emisión de telemetría, que impactan más a los percentiles altos que a la mediana.

La penalización en p95 sugiere que la observabilidad introduce latencias adicionales en colas altas, lo que coincide con estudios previos sobre mallas de servicio y trazabilidad distribuida. La mejora en la mediana puede atribuirse a efectos de warm-up, cachés y adaptación del sistema bajo carga.

Escenarios financieros simulados

Se reprodujeron escenarios vinculados a las necesidades del sector:

- Alta disponibilidad: se simularon fallos de pods, verificando la rápida recuperación automática de Kubernetes y la continuidad del monitoreo.
- Cumplimiento regulatorio: se validó la anonimización de datos sensibles mediante procesadores en OpenTelemetry Collector.
- Seguridad: se comprobó la encriptación de tráfico con mTLS entre sidecars, requisito esencial para auditorías financieras.

Estos experimentos llevaron a confirmar que la arquitectura que propuesta sí funciona en la práctica, puede crecer según sea necesario y es lo suficientemente sólida como para usarse en instituciones financieras reales. Lo que se encontró es que al incorporar telemetría unificada a través de OpenTelemetry e Istio Service Mesh, el sistema adquiere capacidades de observabilidad que realmente marcan la diferencia: pudiendo detectar problemas antes de que se agraven, seguir cada transacción de principio a fin sin perder detalle, y cuando algo falla, diagnosticarlo rápidamente mientras está ocurriendo.

Ahora bien, ¿qué pasaría si no se tuviese esta telemetría integrada? La situación cambiaría drásticamente. Existiría dependencia de logs dispersos por todos lados, métricas que cada servicio guarda por su cuenta, y un montón de herramientas diferentes que no necesariamente hablan el mismo idioma. Esto haría que conectar los puntos entre distintos eventos fuera una pesadilla, y el tiempo que tomaría resolver un incidente se dispararía. Peor aún, sin una capa de observabilidad que funcione de manera coherente, sería casi imposible anticiparse a problemas de rendimiento antes de que afecten a los usuarios, y costaría muchísimo cumplir con

las políticas operativas y regulaciones que en el sector financiero exigen: poder rastrear cada operación de punta a punta.

Entonces, al observar todo el panorama, los resultados hablan por sí solos: implementar esta arquitectura con telemetría integrada no solo hace que el sistema sea más resiliente y opere de manera más eficiente, sino que también simplifica enormemente la tarea de monitorear todo, y ayuda a alcanzar ese nivel de cumplimiento técnico y regulatorio que el sector financiero demanda sin excepción.

Las pruebas se realizaron en un entorno controlado con Minikube, por lo que los resultados deben extrapolarse con cautela a entornos de producción reales. No se incluyeron métricas de costo energético ni de ancho de banda de red.

Conclusiones

Los resultados del prototipo evidencian que la integración de observabilidad mediante Istio y OpenTelemetry mejoran la trazabilidad y resiliencia de los sistemas financieros distribuidos. Además, la implementación ha mostrado que una arquitectura de observabilidad apoyada de herramientas como Istio, Prometheus, Grafana y Kiali proporcionaron una visibilidad integral del comportamiento de los microservicios en Kubernetes sin tener que modificar el código de los mismos. Istio capturó telemetría de tráfico de extremo a extremo desde la malla de servicios; Prometheus aportó métricas continuas y comparables en el tiempo; y Kiali hizo más legible el flujo del sistema de los microservicios al contar con vistas topológicas. Grafana, además, permitió aterrizar esa información en paneles llevando a tener un panorama del comportamiento de los microservicios. En conjunto, estas piezas se integraron permitiendo tener un plano de observabilidad cumpliendo el objetivo general de diseñar una arquitectura modular y extensible; incluso, ante una degradación puntual es posible ver el pico de latencia, seguir el salto entre servicios y contrastarlo con el consumo de recursos en segundos. Disponer de una visión operativa integral facilita la toma de decisiones rápidas en entornos como el

financiero, donde los márgenes de error son mínimos y la respuesta oportuna marca la diferencia.

En las operaciones diarias de los servicios, al combinar diagramas de flujo y métricas, se recorta el tiempo para detectar y aislar incidentes. De un vistazo, se volvieron más evidentes los cuellos de botella, los tropiezos en la comunicación entre servicios y esos picos de latencia que antes se escondían entre logs. Además, esta sinergia impulsó de manera consistente el tiempo medio de detección y el tiempo medio de recuperación. La mejora en disponibilidad y resiliencia no vino de una única herramienta, sino de correlacionar sistemáticamente señales que antes se miraban por separado.

El diseño modular y declarativo facilitó la implementación en un clúster de kubernetes de manera local. Al tratar la configuración como código, se automatizaron la instrumentación, la verificación posterior a la implementación y parte de las pruebas de rendimiento. Este enfoque optimiza las operaciones cuando instrumentamos microservicios para que sean observables y reducimos el riesgo de configuraciones en el despliegue.

La arquitectura propuesta cumple con las premisas de visibilidad, monitoreo y resiliencia, ofreciendo una base sólida para su adopción en entornos financieros. Su modularidad y extensibilidad la convierten en una plataforma viable para futuras integraciones con mecanismos de analítica predictiva y automatización de respuestas ante incidentes.

AGRADECIMIENTOS

Agradecemos al tutor del proyecto por su guía constante, sus observaciones puntuales y el cuidado puesto en cada revisión. Sus comentarios nos ayudaron a afinar la metodología y a sostener el rigor técnico de los resultados.

Extendemos el reconocimiento a los docentes de la Maestría en Software, cuyas asignaturas sentaron las bases conceptuales y prácticas necesarias para formular, diseñar e implementar la solución propuesta. Este trabajo recoge, además, el esfuerzo sostenido de los autores que a lo largo del proyecto combinamos estudio, experimentación y discusión para integrar ingeniería de software, arquitectura de sistemas y observabilidad aplicada.

Finalmente, agradecemos a la institución académica por el entorno de aprendizaje y los recursos tecnológicos que hicieron posible la culminación de este estudio. Sin ese apoyo logístico y académico, el avance no habría sido el mismo.

Referencias

L. Akmeemana, C. Attanayake, H. Faiz, and S. Wickramanayake. Gal-mad: Towards explainable anomaly detection in microservice applications using graph attention networks. arXiv preprint arXiv:2504.00058, 2025.

S. Bennett and J. Whitaker. Monitoring and observability of microservices using kubernetes and prometheus. 2022.

M. C. Borges, J. Bauer, and S. Werner. Oxn-automated observability assessments for cloud-native applications. In 2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C), pages 167–170. IEEE, 2024a.

M. C. Borges, J. Bauer, S. Werner, M. Gebauer, and S. Tai. Informed and assessable observability design decisions in cloud-native microservice applications. In 2024 IEEE 21st International Conference on Software Architecture (ICSA), page 69–78. IEEE, June 2024b. doi: 10.1109/icsa59870.2024.00015.

M. C. Borges, J. Bauer, S. Werner, M. Gebauer, and S. Tai. Informed and assessable observability design decisions in cloud-native microservice applications. In 2024 IEEE 21st International Conference on Software Architecture (ICSA), pages 69–78. IEEE, 2024c.

Z. Chen, Z. Jiang, Y. Su, M. R. Lyu, and Z. Zheng. Tracemesh: Scalable and streaming sampling for distributed traces. In 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), pages 54–65. IEEE, 2024.

Q. Du, T. Xie, and Y. He. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In International Conference on Algorithms and Architectures for Parallel Processing, pages 560–572. Springer, 2018.

U. Faseeha, H. J. Syed, F. Samad, S. Zehra, and H. Ahmed. Observability in microservices: An in-depth exploration of frameworks, challenges, and deployment paradigms. IEEE Access, 2025.

A. Gupta. Scalable distributed tracing and performance optimization in microservices. ESP Journal of Engineering Technology Advancements, pages 210–224, 2021.

N. Kratzke. Cloud-native observability: The many-faceted benefits of structured and unified logging-a case study. 2022.

C. Lee, T. Yang, Z. Chen, Y. Su, and M. R. Lyu. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pages 1750–1762. IEEE, 2023.

B. Li, X. Peng, Q. Xiang, H. Wang, T. Xie, J. Sun, and X. Liu. Enjoy your observability: an industrial survey of microservice tracing and analysis. Empirical Software Engineering, 27(1): 25, 2022.

Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, et al. Practical root cause localization for microservice systems via trace analysis. In 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), pages 1–10. IEEE, 2021.

D. Liu, C. He, X. Peng, F. Lin, C. Zhang, S. Gong, Z. Li, J. Ou, and Z. Wu. Microhecl: High-efficient root cause localization in large-scale microservice systems. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pages 338–347. IEEE, 2021.

P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, et al. Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks. In 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pages 48–58. IEEE, 2020.

B. Madupati. Observability in microservices architectures: Leveraging logging, metrics, and distributed tracing in large-scale systems. Metrics, and Distributed Tracing in Large-Scale Systems (November 30, 2023), 2023.

A. Mahida. Integrating observability with devops practices in financial services technologies: A study on enhancing software development and operational resilience. International Journal of Advanced Computer Science & Applications, 15(7), 2024.

K. Mensah and A. Ofori. Microservices observability pipelines in kubernetes with fluentd and loki. 2023.

A. Nõu, S. Talluri, A. Iosup, and D. Bonetta. Investigating performance overhead of distributed tracing in microservices and serverless systems. In Companion of the 16th ACM/SPEC International Conference on Performance Engineering, pages 162–166, 2025.

J. S. Nunes, S. C. Sampaio, R. S. Malaquias, and I. de Morais Barroca Filho. Deploying the observability of the sigsaude system using service mesh. In 2020 20th International Conference on Computational Science and Its Applications (ICCSA), pages 9–15. IEEE, 2020.

M. Okpako, O. Babatunde, and V. Eguavoen. Exploring tracing in microservice applications: Leveraging zipkin for enhanced observability. International Journal of Research and Scientific Innovation, pages 384–395, 02 2025.

D. Pathak, M. Verma, A. Chakraborty, and H. Kumar. Self adjusting log observability for cloud native applications. In 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), pages 482–493. IEEE, 2024.

J. Soldani and A. Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. ACM Comput. Surv., 55(3), Feb. 2022a.

J. Soldani and A. Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. ACM Computing Surveys (CSUR), 55(3):1–39, 2022b.

M. Steidl, B. Dornauer, M. Felderer, R. Ramler, M.-C. Racasan, and M. Gattringer. How industry tackles anomalies during runtime: Approaches and key monitoring parameters. In 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pages 364–372. IEEE, 2024.

Y.-T. Wang, S.-P. Ma, Y.-J. Lai, and Y.-C. Liang. Analyzing and monitoring kubernetes microservices based on distributed tracing and service mesh. In 2022 29th Asia-Pacific Software Engineering Conference (APSEC), pages 477–481. IEEE, 2022.

M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez. Design, monitoring, and testing of microservices systems: The practitioners perspective. Journal of Systems and Software, 182: 111061, 2021.

A. Widerberg and E. Johansson. Observability of cloud native systems. Digitala Vetenskapliga Arkivet, 2022.

J. Yang, Y. Guo, Y. Chen, and Y. Zhao. Tracenet: Operation aware root cause localization of microservice system anomalies. In 2023 IEEE International Conference on Communications Workshops (ICC Workshops), pages 758–763. IEEE, 2023.

G. Yu, Z. Huang, and P. Chen. Tracerank: Abnormal service localization with dis-aggregated end-to-end tracing data in cloud native systems. Journal of Software: Evolution and Process, 35 (10):e2413, 2023.