Universidad ORT Uruguay
Facultad de Ingeniería

Desarrollo de una prueba de concepto de prácticas AIOps

Entregado como requisito para la obtención del título de Licenciatura en Sistemas.

Facundo Aguerre Guerisoli – 187426

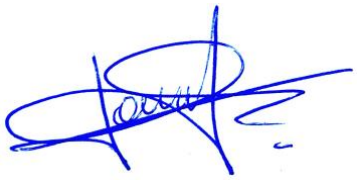
Tutor: Martin Solari

2025

Declaración de autoría

Yo, Facundo Aguerre Guerisoli, declaro que el trabajo que se presenta en esa obra es de nuestra propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizába el trabajo integrador de la Licenciatura en Sistemas;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente propia;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente qué fue contribuido por otros, y qué fue contribuido por mi persona;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Facundo Aguerre

15 de setiembre de 2025

Agradecimientos

A mi familia, mis amigos y mi novia por su apoyo incondicional durante todo mi recorrido académico. Sin su apoyo no podría haber llegado hasta aquí.

A mi tutor, Martin Solari, por su compromiso y acompañamiento constante, que hicieron posible la realización de este trabajo.

A Victor Lutz, por su apoyo y consejos durante toda la carrera.

A Enrique Gretter, compañero de tantos desafíos y aprendizajes compartidos.

Abstract

Los sistemas de software modernos son cada vez más complejos y requieren altos niveles de disponibilidad y observabilidad. DevOps integra desarrollo y operaciones a través de prácticas como automatización, infraestructura como código y entrega continua, con el fin de aumentar la velocidad de entrega y la confiabilidad de los sistemas. AIOps amplía este enfoque mediante el uso de inteligencia artificial aplicada a datos telemétricos. La prueba de concepto se desarrolló extendiendo un proyecto académico usado en los cursos universitarios de ingeniería de software de forma de evalúar cómo AIOps puede fortalecer la disponibilidad, observabilidad y resiliencia de un sistema. La propuesta consistió en instrumentar una arquitectura con OpenTelemetry, integrarla a un pipeline de CI/CD y construir una infraestructura de pruebas capaz de simular escenarios de negocio y fallos controlados. Sobre estos datos se aplicó Isolation Forest, un algoritmo no supervisado de detección de anomalías entrenado con métricas, logs y trazas. Los resultados muestran que AIOps permite detectar patrones anómalos tempranos, mejorar la correlación entre señales, reducir la dependencia de umbrales estáticos y habilitar respuestas proactivas. En conclusión, su aplicación en un contexto DevOps constituye un mecanismo viable para mejorar la continuidad operativa, la resiliencia y la observabilidad de sistemas modernos.

Palabras clave

Ingenieria de Software, Agilidad, DevOps, Observabilidad, Disponibilidad, Resiliencia, AIOps, Isolation Forest

Indice

1. Introducción

La ingeniería de software ha evolucionado rápidamente durante la era de la transformación digital. Las organizaciones se enfrentan al desafío de entregar valor a un ritmo acelerado, adaptarse a mercados cambiantes garantizando la experiencia del usuario sin interrupciones.

De forma de afrontar estos desafíos, la agilidad en la ingeniería de software surgió como una alternativa a los enfoques tradicionales que se caracteriza por ciclos de desarrollo largos, poca flexibilidad ante cambios y distanciamiento de entregas con las necesidades del cliente. El Manifiesto Ágil publicado en 2001 estableció prácticas y valores claves como la colaboración cercana con el cliente, capacidad de adaptación al cambio y entrega continua de software [1].

Sin embargo, las metodologías ágiles no resolvían del todo el desafío de llevar ese software de manera rápida y segura a entornos de producción. La transición a producción para algunas organizaciones tiende a ser un proceso estresante que incluye actividades manuales propensas a errores e, incluso, correcciones de última hora. Esta brecha entre el desarrollo de software ágil y el despliegue en producción se traduce en el fenómeno conocido como "muro de la confusión", donde los equipos de desarrollo persiguen la rapidez y los equipos de operaciones priorizan la estabilidad, con objetivos muchas veces opuestos [2] [3].

1.1. Contexto de DevOps y AIOps

DevOps puede considerarse una evolución natural de los métodos de desarrollo ágil. DevOps busca integrar las actividades de desarrollo (Dev) y operaciones (Ops) en un solo equipo para reducir el tiempo entre la confirmación de un cambio y su puesta en producción, garantizando al mismo tiempo una alta calidad y estabilidad [4]. A través de la automatización de la entrega continua, colaboración y comunicación continua entre equipos de desarrollo e infraestructura como código, DevOps permitió a las organizaciones incrementar la velocidad de entrega, mejorar la confiabilidad de los sistemas y alinear mejor el software con las necesidades del negocio.

En forma paralela, los entornos de TI modernos presentan una complejidad creciente, caracterizados por sistemas diversos y variables, arquitecturas distribuidas y volúmenes de datos cada vez mayores, ha exacerbado los desafíos asociados a las operaciones de TI tradicionales [5]. Los equipos de DevOps se han vuelto dinámicos, con mayor automatización inteligente y constante adaptación cultural de forma de poder afrontar a las nueva tecnologías [3]. Las organizaciones requieren que sus sistemas y servicios estén disponibles las 24 horas del día, los 7 días de la semana. Para eso, la necesidad de obtener información en tiempo real sobre el rendimiento del sistema se ha vuelto crucial.

La disponibilidad se refiere a la capacidad de un sistema para encontrarse operativo y preparado para ejecutar sus funciones cuando es requerido. La disponibilidad es un atributo de calidad del software que implica confiabilidad, además del concepto de recuperación [6]. La observabilidad es un atributo de calidad que busca entender el estado interno del sistema basada en tres pilares: logs, traces y métricas, ya no es una cuestión de último momento, sino un enfoque principal de las prácticas de DevOps. La observabilidad es un soporte esencial de la disponibilidad ya que facilita la detección y diagnóstico de problemas que afectan la continuidad del servicio y refuerza la corresponsabilidad de los equipos DevOps en los sistemas en producción. En este marco, la observabilidad se convierte en un soporte clave de la disponibilidad. Al permitir identificar y comprender incidentes de forma temprana, contribuye a mantener la continuidad del servicio y refuerza la corresponsabilidad de los equipos DevOps en los sistemas en producción [7].

Nuevas tendencias como la IA, han permitido a los equipos DevOps lograr mayor eficiencia, seguridad e innovación a través del procesamiento de datos históricos. AIOps (Inteligencia Artificial para Operaciones de TI) es la aplicación de diversas prácticas y herramientas para ayudar a las organizaciones de TI la toma de decisiones de forma proactiva [8].

AIOps optimiza el proceso de monitorización mediante la automatización inteligente, correlacionando datos de diferentes fuentes, detectando patrones e identificando problemas antes de que provoquen interrupciones. Para las organizaciones que dependen de un mayor rendimiento y tiempo de actividad, el cambio de operaciones reactivas a proactivas transforma su forma de gestión [9].

La comprensión del estado operativo de un sistema en tiempo real depende de los datos que este produce de manera continua. Entre ellos, se destacan las métricas, logs y las trazas. En este contexto, el uso de algoritmos de aprendizaje automático aplicados sobre métricas, logs y trazas permite afrontar problemas como la detección de anomalías en tiempo real.

Con AIOps, el uso de algoritmos de aprendizaje profundo permite identificar patrones complejos y detectar desviaciones, lo que posibilita una reacción proactiva antes de reactiva [9]. De esta manera, las organizaciones logran reducir tiempos de inactividad, minimizar falsas alarmas y mejorar la fiabilidad de sus servicios, consolidando la transición desde una operación reactiva hacia una gestión inteligente y predictiva.

1.2. Objetivos de investigación

El objetivo del presente trabajo, en el marco del fin de carrera de Licenciatura en Sistemas, es desarrollar una aplicación de software que sirva de prueba de concepto para la incorporación de prácticas de AIOps.

El propósito es analizar cómo la integración de técnicas de inteligencia artificial aplicadas a los datos telemétricos (logs, métricas y trazas) puede potenciar los atributos de calidad de los sistemas de software, en particular su observabilidad y disponibilidad, favoreciendo la detección temprana y eficiente de fallas.

En primer lugar se realiza un análisis del marco teórico, a través de la revisión de definiciones y prácticas vinculadas con DevOps. Entre las mismas, se destaca la evolución de la observabilidad, el surgimiento de AIOps y sus prácticas. En segundo lugar, diseña una arquitectura de software que integre procesos de integración y despliegue continuo con telemetría estandarizada, de forma de garantizar un flujo automatizado de información y habilitar capacidades avanzadas de observabilidad. Como tercer lugar, se identifican estándares, marcos conceptuales y herramientas que permitan sustentar la aplicación de observabilidad y AIOps. De esta forma se evalúan ventajas, limitaciones y posibilidades de integración en entornos reales de desarrollo de software. En cuarto lugar, se definen métricas de negocio que permiten construir un contexto realista para la prueba de concepto, posibilitando evaluar el comportamiento del sistema

tanto en su aspecto operativo como en su impacto organizacional. En quinto lugar, se desarrolla la implementación detallando la incorporación de telemetría, la integración de modelos de inteligencia artificial y el procesamiento de datos para la detección de anomalías. Sexto y último lugar, se ejecutan pruebas en un entorno controlado donde se inyectan fallas de forma de analizar resultados y generar conclusiones sobre las prácticas de AIOps implementadas.

2. Marco Teorico

2.1. DevOps

Durante años los equipos de desarrollo y operaciones permanecieron desconectados con objetivos y responsabilidades distintas. Problemas de comunicación entre los equipos y difusión en la integración e implementación de software en producción generaban problemas como la falta de entrega de valor al cliente [2].

DevOps es un movimiento cultural y conjunto de prácticas que combinan el desarrollo de software (Dev) y las operaciones de TI (Ops). Su objetivo es facilitar la cooperación entre los equipos, fomentando una cultura donde la colaboración es esencial y cada miembro del equipo comparte responsabilidades, buscando la mejora continua. Con esto, se procura acortar el ciclo de vida del desarrollo de sistemas al tiempo que se entregan nuevas funcionalidades, correcciones y actualizaciones con frecuencia en estrecha alineación con los objetivos de negocio [2].

Los equipos de DevOps utilizan indicadores de rendimiento, entre los que destacan las métricas DORA. Estas métricas sirven como un mecanismo para vincular las prácticas de desarrollo con los objetivos estratégicos del negocio. Estas métricas se han consolidado como referentes en la evaluación del desempeño de los procesos de entrega continua y despliegue. Las cuatro métricas principales son:

- Deployment Frequency: Con qué frecuencia una organización despliega releases a producción con éxito.
- Mean Lead Time for Changes (MLTC): Cuanto tiempo pasa para que un cambio de código (commit) llegue a producción.
- Mean Time to Recovery (MTTR): Porcentaje de despliegues que fallan en producción.
- Change Failure Rate: Cuánto tiempo transcurre para recuperar la producción de un fallo [10].

2.1.1. Entrega continua

La integración y entrega continua (CI/CD), es una de las prácticas fundamentales dentro de la metodología DevOps. La integración continua (CI) implica la integración frecuente de cambios de código en un repositorio compartido, acompañada de pruebas automatizadas y ejecutadas en entornos similares a producción, lo que permite detectar errores de manera temprana y obtener retroalimentación rápida. La entrega continua (CD) introduce el concepto de deployment pipeline, asegurando que tanto el código como la infraestructura se mantengan siempre en un estado desplegable y listo para producción.

En conjunto, las prácticas de CI/CD permiten automatizar las tareas de pruebas e implementación, reduciendo los problemas en los despliegues y aumentando la velocidad de entrega y mejorando la calidad de los entregables [2].

2.1.2. Infraestructura como código (IaC)

La Infraestructura como Código (IaC) consiste en gestionar y aprovisionar la infraestructura a través de definiciones de código en lugar de configuraciones manuales. Esto permite estandarizar entornos y reducir errores humanos. Este enfoque no solo incrementa la velocidad de entrega y la confiabilidad, sino que además otorga a los equipos la capacidad de automatizar el ciclo completo desde el desarrollo hasta la operación asegurando la consistencia entre entornos [2].

Por otra parte, la monitorización continua es una práctica esencial en DevOps. Consiste en supervisar de forma proactiva el rendimiento, la disponibilidad y el consumo de recursos en cada etapa del ciclo de vida del software.

La monitorización de DevOps opera con el objetivo principal de detectar y resolver problemas de forma temprana, previniendo así de forma proactiva las interrupciones del sistema o la degradación del servicio [11].

El aumento de la complejidad de las aplicaciones las vuelve más dinámicas y distribuidas. Al mismo tiempo, existe la exigencia de garantizar disponibilidad, calidad del servicio y continuidad 24x7. Esto exige a los equipos de DevOps comprender en tiempo real el

comportamiento de los sistemas en producción para detectar anomalías, anticipar fallos y asegurar la continuidad operativa sin afectar la experiencia del usuario.

Las soluciones tradicionales de automatización resultan insuficientes y presentan limitaciones en términos de escalabilidad, robustez y eficacia operativa. La enorme cantidad de datos generados por los sistemas incrementa la dificultad de comprender en tiempo real el estado de los sistemas, lo que hace imprescindible contar con nuevas capacidades de observabilidad y análisis avanzado. La observabilidad ha dejado de ser una práctica reactiva y se ha convertido en un componente estratégico para garantizar la disponibilidad continua y la eficiencia en entornos distribuidos, donde confluyen métricas de negocio, seguridad y rendimiento en una misma visión integrada [12].

2.2. Observabilidad

La observabilidad en arquitecturas de sistemas es la capacidad de inferir el estado interno y los modos de fallo de un sistema distribuido complejo únicamente a partir de sus salidas externas.

Es factible considerar la observabilidad como una evolución natural de la monitorización y se ha asociado con una práctica que va más allá de la simple recopilación de datos de telemétricos. Mediante la observabilidad, los equipos de desarrollo y operaciones tienen una visión profunda y contextualizada de su comportamiento en tiempo real [13] [14].

La telemetría constituye la base sobre la que se construye la observabilidad [15], recopila y analiza logs, métricas y trazas de diversas fuentes remotas. Los logs registran eventos con marca de tiempo que permiten entender el comportamiento histórico del sistema. Las métricas son medidas cuantificables del rendimiento, como el uso de CPU, memoria o número de solicitudes. Las trazas por en cuanto, siguen una solicitud a lo largo de sus componentes, mostrando cómo interactúan los distintos servicios. De esta forma, la telemetría sirve de ayuda a las organizaciones para implementar mejoras basadas en información en sus procesos y sistemas.

Las métricas son medidas cuantificables del rendimiento, como uso de CPU, memoria o número de solicitudes. Los logs registran eventos con marca de tiempo que permiten

entender el comportamiento histórico del sistema. La telemetría se puede definir en tres pasos; la recopilación de datos, transmisión de datos y análisis y monitoreo [16].

Los datos recopilados, mediante fuentes como sensores, programas y otras herramientas de monitoreo, reúnen datos como el comportamiento de los usuarios, métricas y las condiciones del entorno en actividad. La trasmisión de estos datos recopilados son enviados a un sistema central encargado de su recolección donde también puede ocurrir la sanitización de los datos permitiendo una mejor integración entre fuentes. Finalmente los datos son procesados, analizados y exhibidos a fin de brindar información sobre el estado de los sistemas monitoreados, su rendimiento y su comportamiento [16].

2.2.1. Log

Los logs son mensajes con marca de tiempo que las aplicaciones emiten para registrar eventos en un momento dado [17]. Tradicionalmente, los desarrolladores y operadores han recurrido ampliamente a logs para comprender el comportamiento del sistema.

Un log puede tener distintas estructuras. Su formato puede variar (texto plano, JSON estructurado, etc.), lo que a veces dificulta su consulta y manejo [9].

Una cuestión debatida en la industria es dónde y cómo registrar los logs, ya que no existe un consenso absoluto. Según [18], en la industria deberían loguear en ubicaciones del código donde se maximice el valor de los logs para tareas de mantenimiento, diagnóstico y monitoreo del sistema. Sin embargo, no existe un consenso real de donde es la mejor ubicación o capa a realizarlo. De acuerdo al estudio, los dos tipos de contextos más utilizados se dan en situaciones inesperadas y puntos críticos de acción. Por otro lado, algunos equipos prefieren logs muy verbosos (niveles de log más detallados y en mayor cantidad) en cada componente, mientras otros optan por generar logs solo en puntos clave (por ejemplo, al inicio y fin de una petición) para evitar ruido excesivo.

En la práctica moderna, se busca que los logs estén correlacionados con las trazas para agregar contexto. Se recomienda adjuntarlos a una traza o span específico, de modo que sepamos dónde y durante qué operación se generó cada log [14] [19].

Los logs suelen clasificarse por niveles de severidad para distinguir su propósito:

- DEBUG / TRACE
  - Información de depuración muy detallada.
  - Desde las llamadas HTTP realizadas hasta consultas a librerías.
  - Usualmente este nivel se desactiva en producción para disminuir la verbose.
  - Utilizado mayoritariamente en ambientes de desarrollo local o staging.
- INFO
  - Eventos informativos generales. Por ejemplo, inicio/fin de procesos importantes en producción. Nivel más alto que DEBUG/TRACE.
- WARNING
  - Se utilizan para situaciones anómalas no fatales. Por ejemplo, intentos inválidos de un usuario que podrían indicar un problema o situaciones de posible riesgo.
- ERROR
  - Nivel utilizado para emitir errores de funcionamiento, excepciones o fallos en servicios. Por ejemplo, emisión de error ante falla por llamada un servicio third-party con status code 500 el cual no permite continuar el proceso.
- FATAL / CRITICAL
  - Representa los errores graves que obligan a detener la aplicación o un componente crítico del sistema.
  - Se utiliza, por ejemplo, cuando falla la conexión a la base de datos principal o cuando no hay posibilidad de recuperación sin intervención manual.
  - En la mayoría de los casos, estos logs requieren alertas inmediatas y pueden estar integrados con sistemas de monitoreo para activar notificaciones.
- OFF
  - Este nivel se utiliza para desactivar por completo el registro de logs.
  - No genera ningún evento en el sistema de logging.

- Suele aplicarse en configuraciones especiales donde el rendimiento es crítico o para entornos donde el logging no está permitido por políticas de seguridad [20].

El uso adecuado de los niveles evita tanto la sobrecarga de alertas y problemas de reconocimiento de errores fácilmente. Además la categorización permite filtrar logs generados de acuerdo a su magnitud, de forma de enfocarse en los más relevantes.

Cada log debe contar con suficiente contexto. El contexto no es más que datos que se consideren importantes para la comprensión del estado del sistema. Por ejemplo, un identificador del servicio o módulo, identificador de la petición o transacción, usuario o proceso involucrado, y otros detalles (ej. dirección IP, hilo de ejecución, etc.) que faciliten la búsqueda y correlación de datos para el fácil entendimiento del estado del sistema.

Un log bien estructurado y basado en buenas prácticas, puede contar con distintos formatos e información. Sin embargo, el común entre herramientas de observabilidad es utilizar estructura de JSON. Un ejemplo de esto puede verse en la Figura 1: Ejemplo de log en formato JSON.

La estructura entre los registros destacados:

- El registro timestamp indica el momento exacto en que ocurrió el evento, mientras que level refleja la severidad. Estos elementos son fundamentales para ordenar los eventos y priorizar la respuesta según su criticidad.
- El registro trace_id y span_id permiten correlacionar el evento con un flujo mayor dentro de un sistema distribuido. De esta forma, los logs se integran con trazas y métricas, aportando una visión completa de la operación favoreciendo la observabilidad del sistema.
- El registro service y environment especifican en qué aplicación o microservicio ocurrió el evento y en qué contexto. Esta información resulta clave para sistemas distribuidos, donde múltiples componentes generan logs simultáneamente.
- El registro message resume en lenguaje humano lo sucedido, sirviendo como referencia rápida para el usuario.

El resto de los registros, suelen ser introducidos como información adicional y enriquecedora del evento.

```
{
  "timestamp": "2024-08-04T12:34:56.789Z",
  "level": "INFO",
  "service": "user-authentication",
  "environment": "production",
  "message": "User login successful",
  "context": {
    "userId": "1",
    "username": "johndoe",
    "client_ip": "192.168.1.1",
    "user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0
Safari/537.36"
  },
  "trace_id": "abcd-efgh-ijkl-mnop",
  "span_id": "123-ijkl-sdkjfsd-5453",
  "duration": 200,
  "request": {
    "method": "POST",
    "url": "/api/v1/login",
    "headers": {
      "Content-Type": "application/json",
      "Accept": "application/json"
    },
    "body": {
      "username": "username",
      "password": "******"
    }
  },
  "response": {
    "statusCode": 200,
    "body": {
      "success": true,
      "token": "jwt-token-here"
    }
  }
}
```

Figura 1: Ejemplo de log en formato JSON

2.2.2. Metricas

Las métricas son mediciones numéricas sobre el desempeño o estado de un sistema a lo largo del tiempo. Se pueden representar a través de contadores, promedios o tasas que sirven como indicadores claves de salud del sistema [21]. A diferencia de los

logs, se almacenan como series temporales, o sea, valores numéricos por tiempo y suelen estar agregadas para reducir volumen y facilitar su análisis [15].

Su estructura suele incluir nombre, valor, marca de tiempo y metadata como el servicio o instancia que reporta, lo cual permite realizar búsquedas más precisas o agregarlas fácilmente [19].

Las métricas son especialmente útiles para monitoreo proactivo y alertas, porque permiten establecer umbrales cuantitativos. Los indicadores de nivel de servicio (SLI) son métricas representativas de la calidad del servicio (por ejemplo, porcentaje de solicitudes exitosas) [22]. Es común derivar estos SLIs agregando métricas específicas que contabilicen cierta acción: por ejemplo, contabilizando las llamadas de solicitudes exitosas vs. totales para calcular una tasa de éxito [23].

Las empresas usualmente definen objetivos de nivel de servicio (SLOs) sobre estas métricas. Los SLO proporcionan un marco para definir objetivos claros en torno al rendimiento de las aplicaciones. Esto ayuda a los equipos a ofrecer una experiencia coherente al cliente, a equilibrar el desarrollo de funciones con la estabilidad de la plataforma y a mejorar la comunicación con los usuarios internos y externos. Un ejemplo práctico de SLO puede ser la definición de un objetivo de 99.9% de solicitudes exitosas en un mes, para evaluar si se cumplen las expectativas de fiabilidad [24].

En la

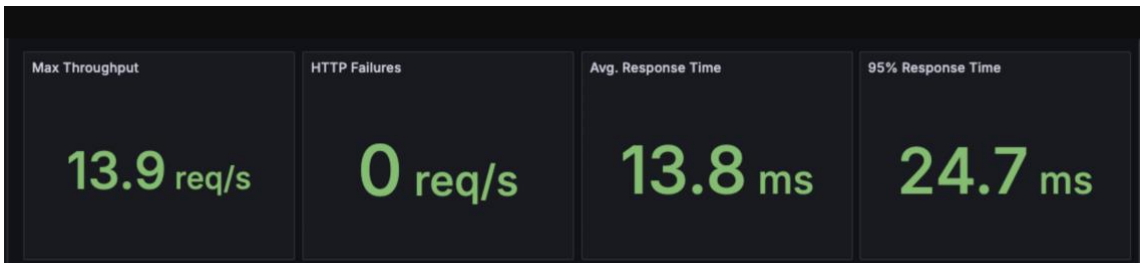


Figura 2: Ejemplo de métricas se define ejemplos de indicadores representativos de la calidad del servicio.

- Max Throughput: Mide la máxima cantidad de solicitudes que el sistema puede procesar en un período de tiempo determinado (En el ejemplo, requests por segundo).

- HTTP Failures: Representa el número de solicitudes HTTP que no se completaron exitosamente (En el ejemplo, solicitudes con código 4xx o 5xx).
- Avg. Response Time: Es el tiempo promedio que tarda el sistema en responder a una solicitud.
- 95% Response Time: indica el tiempo de respuesta bajo el cual se encuentran el 95% de las solicitudes. Mide la latencia percibida por la mayoría de los usuarios y ayuda a detectar picos de demora.

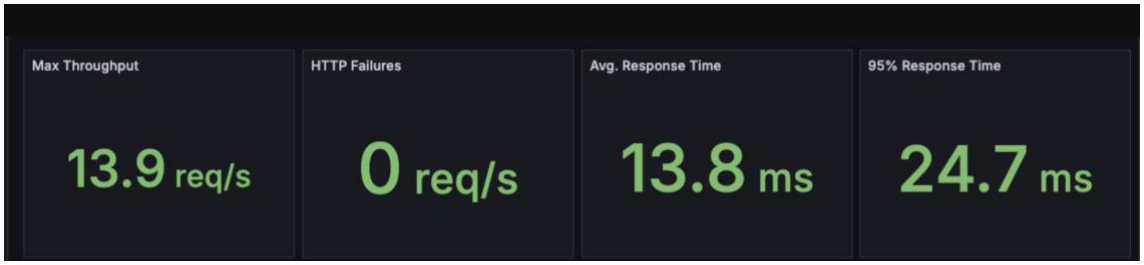


Figura 2: Ejemplo de métricas

2.2.3. Trazas y Spans

Una traza distribuida es el registro completo del recorrido que realiza una petición a través de un sistema compuesto por múltiples servicios. Consiste en un conjunto de segmentos o spans relacionados entre sí, que en conjunto representan la ejecución de esa transacción distribuida, desde el inicio hasta el fin [25].

Un span representa una unidad de trabajo u operación individual. Esto es, por ejemplo, una consulta a base de datos, una llamada a API interna, o un cálculo intensivo. Incluye además información relevante como lo es el nombre de la operación, marca de tiempo y metadatos en forma de atributos.

El primer span de la traza se denomina span raíz (root), correspondiente típicamente a la entrada inicial de la petición (por ejemplo, la llegada de una solicitud HTTP al frontend). Luego, a medida que esa petición invoca a otros servicios o métodos, se generan spans hijos que capturan cada una de esas operaciones, formando un árbol que refleja la causalidad y secuencia de llamadas [15].

Las trazas permiten seguir extremo a extremo cómo una solicitud atraviesa múltiples servicios. Todos los spans asociado a la traza compartirá un mismo identificador (Trace

ID) y cada span tiene además su propio identificador único (Span ID) con a su vez, una referencia al Trace ID de su padre (excepto el root) para reconstruir la jerarquía [15] [23].

Un ejemplo de trazas y spans puede ser visto en la Figura 3: Ejemplo de traza y spans vía Grafana. La traza está a la solicitud POST a la ruta api/StockRequest con HTTP 400. Cuenta con un ID único (336afc565f8c8b685d13bc55834ce3b5), una marca de tiempo (2025-09-03 21:37:21.053) y duración total (16.08ms). El numero de span asociados a la traza son 9, cada uno con su ID único y duración, entre otros atributos. El span en color rojo CreateStockRequest de duración 5.93 ms refleja la operación fallida.

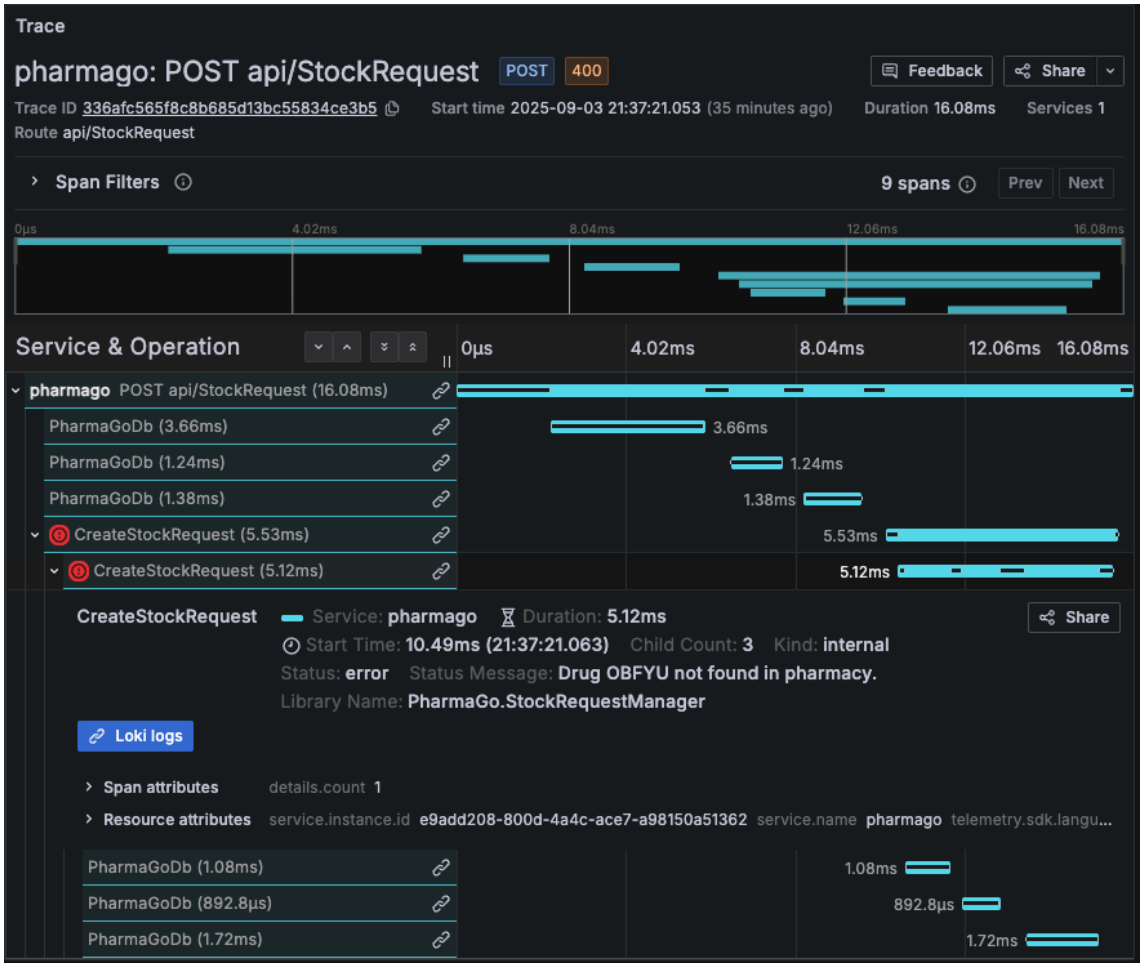


Figura 3: Ejemplo de traza y spans vía Grafana.

La trazabilidad distribuida en sistemas de microservicios es fundamental de forma de identificar el origen de una degradación de rendimiento o fallo cuando una sola acción del usuario atraviesa decenas de componentes [15]. Es posible, ante un problema, revisar

una muestra de trazas de transacciones lentas o con error y ubicar rápidamente en qué servicio y en qué paso se produjo el retraso o la excepción. Esto permite a los equipos mejor capacidad de control, permitiendo una mayor observabilidad y una reducción drástica del tiempo de recuperación del sistema ante eventuales fallas.

2.3. AIOps

La creciente complejidad de los entornos de Tecnologías de la Información (TI), caracterizados por arquitecturas distribuidas, sistemas heterogéneos y un volumen masivo de datos, ha impulsado la necesidad de nuevas estrategias de gestión y automatización.

AIOps es la aplicación de Big data, Machine Learning y otras tecnologías analíticas avanzadas a las funciones de operaciones de TI. Su objetivo es automatizar tareas operativas, analizar grandes volúmenes de datos en tiempo real y extraer información procesable para mejorar la eficiencia del equipo de operaciones en producción [8].

Los sistemas tradicionales de monitorización han demostrado ser insuficientes para gestionar cantidades masivas y diversas de datos. Además de métricas de rendimiento e incidentes de seguridad en tiempo real. Los retrasos en la detección y mitigación de fallos repercuten directamente en el usuario final, mediante la afectación de la disponibilidad del sistema [9].

Las organizaciones que han incorporado prácticas de AIOps reportan beneficios significativos en múltiples dimensiones de la gestión de TI. En el ámbito de los datos, AIOps facilita la integración y depuración de datos heterogéneos, facilitando la construcción de conjuntos de datos confiables y reutilizables. Al mismo tiempo, contribuye a la estandarización de los pipelines de entrega y a la mejora de la observabilidad al detectar anomalías de forma temprana y correlacionar eventos a través de distintas capas del sistema. De esta forma, los equipos de operaciones disponen de información procesable en tiempo real, lo que incrementa su capacidad de respuesta y eleva la calidad de los servicios en producción [26].

AIOps adquiere relevancia al situarse en el centro de los incidentes que ponen en riesgo la continuidad operativa. Su contribución se refleja en la capacidad de reforzar atributos de calidad como la disponibilidad y la observabilidad. Si bien AIOps aún se encuentra en

una fase de maduración, ha mostrado un potencial significativo para abordar situaciones complejas donde los enfoques tradicionales resultan insuficientes, en particular en la anticipación de fallos, la identificación de anomalías y el análisis de causas raíz [26] [27].

2.3.1. Problemáticas abordadas por AIOps

2.3.1.1. Detección de anomalías

La detección de anomalías consiste en detectar anomalías, valores atípicos o, en general, eventos anormales del sistema a través del análisis de datos de telemetría (métricas, logs y trazas), de una o más fuentes.

Las técnicas tradicionales resultan insuficientes frente al creciente volumen y variedad de datos. Los métodos tradicionales suelen limitarse a fuentes estructuradas y con baja cantidad de volúmen, sin embargo, existen fuentes como informes de incidentes, comentarios de usuarios y documentación que suelen ser beneficiosas en contexto e información [27].

De acuerdo a estudios recientes, existe una amplia variedad de modelos de aprendizaje automático para la detección de anomalías. Debido a los volúmenes de datos manejados, las redes neuronales profundas tienden a ser más efectivas. Las redes neuronales profundas permiten identificar anomalías sutiles y complejas que los enfoques tradicionales podrían pasar por alto, mejorando así la precisión en la detección. Esto resulta en una respuesta proactiva y mayor fiabilidad del sistema [27].

Procesamiento de Lenguaje Natural (NLP), permite el análisis de datos no estructurados, como registros, comentarios de usuarios e informes de incidentes, que a menudo contienen gran cantidad de información pero son difíciles de procesar con herramientas tradicionales. Con esto, la capacidad de las redes neuronales profundas para manejar secuencias largas y complejas con datos estructurados y no estructurados, proporciona una visión más completa del estado del sistema. Sin embargo, las redes neuronales profundas presentan desafíos significativos. El principal desafío es la necesidad de grandes volúmenes de datos etiquetados para entrenar adecuadamente estos modelos. Otro desafío es el costo computacional que conlleva el desarrollo y puesta a producción de los mismos [28].

Por otro lado, algoritmos no supevisados como Isolation Forest (IF) aíslan observaciones anómalas mediante particiones aleatorias. Con esto, evita la dependencia de etiquetas en dominios con telemetría masiva y alta cardinalidad. Su principal fortaleza es detectar desviaciones poco frecuentes respecto del comportamiento reciente, lo que permite priorizar alertas sin instrumentación adicional ni esquemas complejos de etiquetado. En la práctica, se han demostrado resultados positivos cuando los logs, métricas y trazas se agregan en ventanas temporales (p. ej., tasas de error HTTP, variaciones de tiempos de, latencias por endpoint/servicio) siendo consideradas en conjunto. De esta forma, ofrece entrenamiento e inferencia eficientes incluso con tráfico cambiante. IF no aprende de forma explícita los patrones temporales (picos por hora o día), por lo que conviene trabajar con ventanas de tiempos y (servicio, endpoint, etc) y re-ajustar el detector cuando cambie el régimen de tráfico. Otro desafío es la calibración del parámetro contamination, el cual permite definir la proporción de valores atípicos del dataset, impacta directamente la tasa de falsos positivos [29] [30].

2.3.1.2. Predicción de fallos

La predicción de fallos implica el análisis de datos históricos para detectar la probabilidad de que un sistema de software experimente un fallo en un plazo futuro. Este proceso ayuda a los operadores del sistema a identificar y abordar los problemas de forma proactiva antes de que se conviertan en fallos reales.

Mientras que las técnicas tradicionales de predicción requieren grandes volúmenes de datos estructurados y suelen limitarse a reglas estáticas o modelos simples, las prácticas de AIOps permiten algoritmos avanzados que permiten capturar patrones complejos en flujos de datos heterogéneos y dinámicos. Técnicas como el aprendizaje profundo basado en arquitecturas LSTMs (Long Short-Term Memory Networks) y Transformer, las cuales permiten el despliegue de modelos, han demostrado una mejora en la predicción y captura de patrones complejos en tiempo real de los distintos datos de telemetría. Con esto, los equipos han obtenido resultados prometedores; menos falsas alarmas, un mantenimiento predictivo mejorado, reducción del tiempo de inactividad, minimización de intervención humana y la mejora de la gestión de la infraestructura en la nube entre otras [31]. En particular, los modelos basados en arquitecturas LSTM y Transformer han demostrado gran efectividad en la identificación de relaciones temporales y secuenciales, lo que

habilita prácticas de mantenimiento predictivo más precisas y reduce la tasa de falsas alarmas.

No obstante, la implementación a gran escala enfrenta retos como la latencia en la inferencia de modelos complejos y el elevado coste computacional asociado [10].

2.3.1.3. Análisis de causa raíz

El análisis de causa raíz (RCA) es un proceso crítico para identificar las causas fundamentales de incidentes en entornos complejos. El objetivo es minimizar el tiempo de clasificación (MTTT) y, a la vez, contribuir a la reducción del Tiempo Medio de Resolución (MTTR).

2.3.1.3.1. Incidente

El ciclo de vida de un incidente en ingeniería de software comprende cuatro etapas fundamentales. En primer lugar, la detección, que puede producirse tanto por la experiencia de los usuarios como por sistemas de monitorización automatizados encargados de supervisar el estado y el rendimiento del servicio. La etapa de clasificación consiste en que el incidente es derivado al equipo más adecuado de acuerdo con sus características y la experiencia técnica disponible. Seguidamente se realiza el análisis de la causa raíz, etapa en la que se examinan registros, métricas de rendimiento, dependencias y guías de resolución con el fin de identificar el origen del problema. Finalmente, se ejecuta la etapa de mitigación, la cual consiste en la aplicación de acciones correctivas basadas en la causa identificada, cuyo propósito es restablecer la funcionalidad del servicio y reducir el impacto negativo en el negocio.

El análisis de causa raíz (RCA) no automatizado se centra tradicionalmente en la creación de reglas que cualquier miembro de DevOps pueda seguir para resolver incidentes repetidos. Sin embargo, no es escalable crear reglas y un flujo de trabajo de procesos independientes para cada tipo de incidente repetido cuando los sistemas son grandes y complejos [26].

Las técnicas de RCA se pueden clasificar de acuerdo a tres clases: técnicas basadas en logs, técnicas basadas en trazas y técnicas de monitorización.

Las técnicas basadas en trazas y monitorización alcanzan un mayor nivel de detalle para las anomalías consideradas y las causas raíz identificadas, en comparación con la única técnica disponible basada en logs [27].

Diversos estudios concluyen que el análisis de causa raíz basado en gráficos causales permite identificar las causas raíz más probables mediante técnicas de recorrido del grafo, análisis de propagación de anomalías o métodos de inferencia causal. Esta técnica es actualmente una de las más utilizadas en entornos de centros de datos y aplicaciones en la nube, debido a su capacidad para capturar relaciones complejas entre componentes y priorizar posibles causas raíz de forma eficiente [27].

Con esto, las prácticas de AIOps resuelven gran parte del desafío al correlacionar datos de múltiples fuentes, detectar patrones e identificar problemas antes de que afecten la operación, reduciendo significativamente los tiempos de respuesta [9].

Las métricas DORA son favorecidos a partir de la aplicación de prácticas de AIOps. A través de la automatización de tareas rutinarias y de la detección proactiva de fallos, se logra una mejora del Mean Time to Recovery (MTTR) y una disminución del Change Failure Rate.

El análisis avanzados y modelos predictivo permiten que los sistemas integrados a AIOps puedan ofrecer información y recomendaciones basadas en datos en tiempo real, mejorando la capacidad de los equipos de TI para tomar decisiones informadas y responder de manera efectiva a los problemas emergentes enalteciendo el MTTR y los fallos en producción [10] [31].

Paralelamente, AIOps introduce una capa de automatización inteligente que optimiza la integración de cambios de código, la ejecución de pruebas y la predicción de impactos. Esto contribuye a una reducción del MLTC, permitiendo a los equipos tener un rápido accionar [32].

La implementación de AIOps enfrenta desafíos importantes. La calidad de los datos sigue siendo un requisito crítico, dado que la eficacia de los modelos de inteligencia artificial depende de la precisión y completitud de la información. A esto se suma la complejidad de integrar soluciones AIOps en sistemas de TI preexistentes, las dificultades asociadas

a la interoperabilidad y las demandas computacionales que elevan los costes operativos. También emergen retos vinculados con la ciberseguridad, ya que los modelos de IA pueden ser vulnerables a ataques adversarios o errores de configuración.

Por tanto, si bien estas capacidades aceleran los ciclos de entrega, la incorporación de prácticas de AIOps requiere una adaptación profunda de los procesos de trabajo, nuevas competencias técnicas en los equipos y un modelo de gobernanza que asegure la sostenibilidad del enfoque. La ausencia de estas condiciones puede generar implementaciones defectuosas, costos innecesarios y pérdida de confianza en la operación.

3. Diseño de prueba de concepto

3.1. Contexto del proyecto

Para desarrollar este trabajo, es necesario partir de un proyecto base de complejidad intermedia, lo suficientemente completo para habilitar mejoras y evaluaciones posteriores del sistema con operativa real. Por tal, el sistema seleccionado posee una escala y dificultad acordes a una etapa formativa en la cual se promueven habilidades avanzadas de Ingeniería de Software como el diseño y arquitectura, persistencia de datos, API Rest e interfaz de usuario.

El proyecto académico corresponde a las asignaturas Diseño de Aplicaciones 2 e Ingeniería de Software Ágil 2 de la carrera de Ingeniería en Sistemas de la Universidad ORT Uruguay. El proyecto poseé una arquitectura de tres niveles. Está compuesto por una base de datos relacional en SQL Server y un backend desarrollado como API REST en .NET. Utiliza el ORM Entity Framework Core con modalidad 'Code First' para la persistencia de entidades. El frontend fue desarrollado con Angular CLI 14.2.6, ejecutándose sobre Node y NPM.

Para la incorporación de prácticas de AIOps, el proyecto cuenta con una arquitectura monolítica. Una arquitectura monolítica estructura una aplicación como una única unidad, con todos los componentes como la interfaz de usuario, la lógica de negocio y el acceso a datos, integrados en un solo bloque de código [33].

La observabilidad en arquitecturas monolíticas implica monitorizar y comprender el comportamiento de una aplicación única y acoplada, lo que es más sencillo que en arquitecturas distribuidas debido a su menor complejidad. Sin embargo, la observabilidad encuentra un potencial mayor en la trazabilidad entre servicios y en la correlación de datos telemétricos.

El monolito aún así, resulta ser un escenario válido para la prueba de concepto ya que concentra la lógica en un único despliegue, reduce las dependencias y facilita las pruebas. Permite además, instrumentar de forma homogénea logs, métricas y trazas aportando señales para validar las prácticas de AIOps.

La definición de métricas de negocio sobre los procesos de compra, ventas y gestión de stock permiten vincular la operativa del sistema con objetivos estratégicos de disponibilidad y observabilidad.

3.2. Definicion de metricas de negocio

El sistema recibe el nombre de PharmaGo, cuyo propósito es gestionar farmacias, medicamentos y operaciones asociadas a la compra y reposición de stock. Ofrece diversas funcionalidades a los distintos usuarios como la administración del catálogo y la operatoria de venta; además, contempla perfiles interno y externos. Los perfiles internos a la farmacia (administradores, dueños y empleados) crean invitaciones, cargan/actualizan medicamentos y solicitan reposición de stock para aprobación del dueño. Para usuarios externos permite buscar y comprar productos según disponibilidad por farmacia.

Las métricas de negocio son esenciales para vincular el comportamiento operativo con los objetivos estratégicos. Su monitoreo proporciona indicadores clave de desempeño que trasciende lo técnico y reflejan el valor organizacional. Por eso, se definen un conjunto de objetivos los cuales serán utilizados en la prueba de concepto:

Objetivo de negocio 1: Cantidad total de ventas por hora.

El objetivo es conocer el flujo de ventas por unidad de tiempo. La métrica correspondiente es pharmago_sales_total, definida como un contador de ventas confirmadas. La unidad de medida es el monto en dólares (USD).

Esta métrica está asociada a la observabilidad, dado que permite monitorear el comportamiento del negocio en tiempo real e identificar caídas abruptas o picos inusuales que pueden anticipar problemas en el sistema.

Objetivo de negocio 2: Cantidad de solicitudes de reposición de stock por día.

El objetivo es medir cuántas solicitudes de reposición se generan en cada farmacia durante un período determinado. La métrica asociada es pharmago_stock_requests_created_total,

definida como un contador que acomula solicitudes creadas por usuarios del sistema. La unidad de medida es el número entero de solicitudes.

La métrica se relaciona con el atributo de observabilidad, ya que permite detectar patrones de demanda y anticipar posibles sobrecargas que afecten la continuidad del servicio.

Objetivo de negocio 3: Cantidad de compras en estado "pendiente" por farmacia.

El objetivo es identificar el volumen de operaciones pendientes en cada farmacia. Para ello se utilizan dos métricas: pharmago_pending_created_total y pharmago_pending_resolved_total, cuya diferencia indica las compras que aún no han sido procesadas. La unidad de medida es el número de operaciones pendientes.

El indicador puede asociarse a la disponibilidad del sistema en términos de que si el número de compras pendientes alcanza los umbrales definidos por el negocio, puede comprometer la experiencia del usuario. Lo cual puede repercutir a cuellos de botellas o indisponibilidad de alcance al servicio.

Objetivo de negocio 4: Tiempo promedio de compras en estado "pendiente" por farmacia.

El objetivo es medir cuánto tiempo permanece una compra en estado pendiente antes de resolverse. La métrica utilizada es pharmago_pending_duration_seconds que registra la duración de la compra en segundos. La unidad de medida es el tiempo promedio en segundos o minutos. La métrica puede relacionarse con la disponibilidad porque refleja la eficiencia en el procesamiento de pedidos.

3.3. Diseño de arquitectura

A nivel arquitectónico, el proyecto es un monolíto. Está compuesto por una base de datos relacional en SQL Server y un backend desarrollado como API REST en .NET 6.0 que posteriormente fue migrado a .NET 9.0. Utiliza Entity Framework Core con modalidad 'Code First' para la persistencia de entidades. El frontend fue desarrollado con Angular CLI 14.2.6, ejecutándose sobre Node y NPM .

El proyecto cuenta con conceptos fundamentales de la ingeniería de software, priorizando atributos de calidad como la mantenibilidad y la extensibilidad. Entre las decisiones de diseño destacan el uso de inyección de dependencias a través de una Service Factory. Además el desacoplamiento del manejo de excepciones y la aplicación de patrones de diseño como fachada y repositorio genérico para las entidades.

De modo de alcanzar los objetivos planteados, se incorpora un conjunto de decisiones que buscan enaltecer los atributos de calidad: observabilidad, disponibilidad y resiliencia, además de facilitar la evolución hacia prácticas avanzadas de AIOps.

En primera instancia, se realiza la incorporación de capacidades de entrega continua (CI/CD). Para eso, es se implementa un flujo automatizado en GitHub Actions. El script contempla la construcción de imágenes Docker, la ejecución de pruebas unitarias y el despliegue en una instancia de Amazon EC2 mediante integración con Amazon Elastic Container Registry (ECR). Este pipeline permite al sistema ser ágil en la integración de cambios y reducción de tiempos de entrega en producción.

3.3.1. OpenTelemetry

OpenTelemetry (OTel) es un marco open-source y un estándar abierto de telemetría. Su propósito propósito es estandarizar la instrumentación del código y la forma en que se generan, recolectan y exportan datos de trazas, métricas y logs hacia distintas plataformas de observabilidad [34].

OpenTelemetry está compuesto por distintos elementos que permiten cubrir el ciclo completo de observabilidad. Dispone de APIs y SDKs específicos para distintos lenguajes, lo que permite una instrumentación uniforme. OpenTelemetry define OpenTelemetry Protocol (OTLP) un protocolo optimizado para el transporte de telemetría que admite tanto gRPC como HTTP. El uso de un protocolo unificado facilita la comunicación entre aplicaciones instrumentadas, agentes y backends, minimizando problemas de interoperabilidad. Por otro lado, define un conjunto de convenciones semánticas que aseguran la estandarización de etiquetas y formatos en los datos generados.

El núcleo de la arquitectura es el Collector; un proceso independiente diseñado para recibir, procesar y reenviar datos de telemetría. A diferencia de las bibliotecas ligadas al código, el Collector opera como un servicio externo que puede integrarse en la infraestructura de manera flexible. El Collector soporta telemetría en múltiples formatos, como por ejemplo Jaeger, Prometheus, Zipkin, Syslog, entre otros. El Collector permite la configuración de pipelines para la transformación de datos previo a exportarla a distintos sistemas. Admite la aplicación de filtros, agregaciones y políticas de muestreo, lo que enriquece la información telemétrica. La principal ventaja del Collector es la capacidad de desacoplar la instrumentación del código de los detalles de exportación. Permitiendo así cambiar la implementación sin modificar las aplicaciones. Por ello, en entornos productivos se considera una buena práctica adoptarlo como intermediario [35].

Existe en el mercado una serie de competidores y estándares definidos utilizados por la industria. Por ejemplo, a diferencia de plataformas propietarias como Datadog, OpenTelemetry ofrece un modelo de apertura y flexibilidad. OpenTelemetry al ser open-source, permite la posibilidad de exportar datos a múltiples destinos simultáneamente, lo que brinda independencia frente a proveedores específicos. En cambio, Datadog es un sistema cerrado que, si bien admite integraciones con estándares como OTel, concentra la ingesta de datos principalmente en su propia plataforma. Esto genera una mayor dependencia de su ecosistema propio. OpenTelemetry no presenta costos directos a no ser los necesarios para el despliegue y mantenimiento de la infraestructura. Por su parte, Datadog opera bajo un modelo de precios basado en hosts, volumen de datos ingeridos y funcionalidades habilitadas. Este enfoque puede derivar en costos significativos en entornos de gran escala [35].

El hecho de ser un proyecto open-source respaldado por una comunidad activa, ofrece independencia tecnológica frente a soluciones pagas, evitando el riesgo de dependencia de un único proveedor. De este modo, permite enaltecer su capacidad de interoperabilidad entre herramientas y plataformas. Esto permite una evolución continua del estándar, con actualizaciones frecuentes que priorizan aspectos de seguridad y compatibilidad [34].

OpenTelemetry facilita el intercambio de métricas, logs y trazas entre sistemas heterogéneos, ofreciendo un marco común que unifica prácticas en un ecosistema diverso de observabilidad. Este enfoque resulta especialmente relevante en aplicaciones nativas

de la nube y arquitecturas distribuidas, donde la consistencia en la telemetría es un requisito fundamental para el análisis transversal [35].

El modulo encargado de la implementación del está las prácticas se recomienda ser cohesivo y con bajo acoplamiento . De esta forma, la capacidad de reuso es alta y permite la reutilización a través del sistema. De cara al proyecto, se implementó un nuevo compoenente especifico que desarrolla las prácticas de observabilidad. OpenTelemetry cuenta con una serie de prácticas que permiten su aplicación efectiva.

Se introdujo un componente denominado Observability con el fin de centralizar y desacoplar toda la lógica relacionada con la telemetría del núcleo funcional del sistema. En la Figura 4: Diagrama de componentes se puede corroborar que esta adhesión, se promueve la creación de componentes cohesivo, reutilizables y con responsabilidades bien delimitadas. Esto favorece el mantenimiento, extensibilidad y evolución del estandar dentro del proyecto [36].

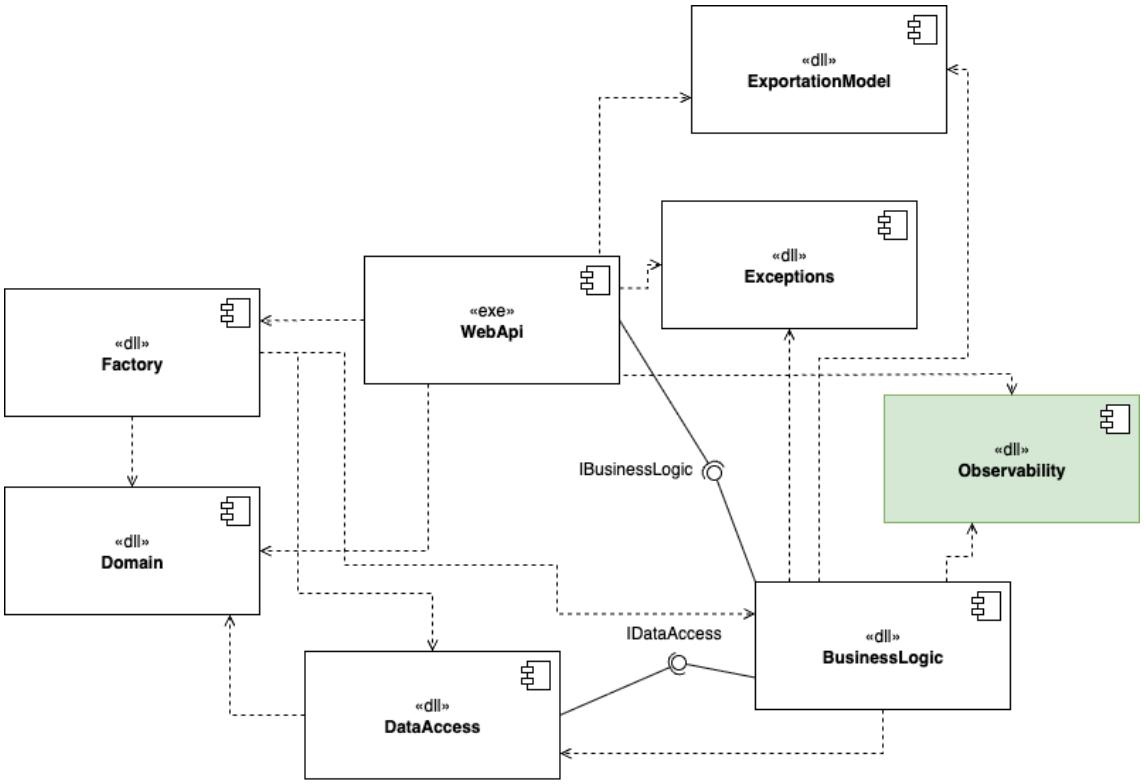


Figura 4: Diagrama de componentes

Por otra parte, la instrumentación en OpenTelemetry puede realizarse de forma manual o automática. La instrumentación manual consiste en añadir fragmentos de código específicos dentro de la aplicación para capturar métricas o trazas personalizadas. Este enfoque ofrece un nivel de control detallado, ya que permite recolectar indicadores de negocio o métricas específicas del dominio, como el tiempo de ejecución de una consulta o el número de ítems en un carrito de compras. Sin embargo, requiere mayor esfuerzo y conocimiento por parte de los desarrolladores. La instrumentación automática, por otro lado, se apoya en agentes o bibliotecas prediseñadas que capturan datos estándar sin necesidad de modificar el código fuente. Este enfoque permite recolectar información como uso de CPU, consumo de memoria, latencia de peticiones o tasas de error de manera rápida y sencilla. Aunque menos flexible que la manual, su ventaja radica en la facilidad de adopción inicial y en la capacidad de proporcionar valor inmediato. En conjunto, ambas modalidades pueden coexistir: la instrumentación automática como base para obtener telemetría estándar de manera ágil, y la manual como complemento para capturar métricas críticas de negocio [37].

Del mismo modo, un aspecto crítico es la correlación de señales. OpenTelemetry recomienda incluir el trace_id en los logs de aplicación, de manera que se pueda correlacionar los datos fácilmente desde una métrica a la traza correspondiente y de allí a los logs detallados. Esta correlación nativa constituye una de las prácticas más poderosas del estándar, ya que ofrece una vista unificada de la telemetría [38].

El muestreo (sampling) es otra práctica clave para lograr observabilidad eficiente. En sistemas de gran escala, recolectar todos los datos puede resultar prohibitivo, por lo que se recomienda aplicar estrategias de muestreo. OpenTelemetry soporta dos tipos principales:

- Head Sampling: decide al inicio de la traza si será muestreada, lo que permite descartar solicitudes de bajo interés desde el comienzo.
- Tail Sampling: analiza la traza completa antes de decidir, lo que posibilita reglas más complejas, como conservar únicamente las trazas en las que participó un servicio crítico o aquellas con errores.

Un diseño adecuado de muestreo permite reducir volumen y costos sin sacrificar la capacidad de diagnóstico. Mal implementado, puede introducir invisibilizar fallos importantes, de modo que su configuración debe ajustarse cuidadosamente a cada sistema [38] [34].

3.4. Herramientas

La selección de herramientas como Prometheus, Loki, Tempo, Grafana y OpenTelemetry (Collector) permite la implementación de observabilidad de forma eficiente, flexible y estandarizada. En conjunto, permiten cubrir de manera integrada las tres señales fundamentales de la observabilidad: logs, métricas y trazas.

Estas soluciones ofrecen escalabilidad y compatibilidad además de integrar señales heterogéneas bajo una interfaz común, facilitando el análisis de incidentes, la mejora de la disponibilidad y el fortalecimiento de la confiabilidad en sistemas complejos y distribuidos [39] [40].

3.4.1. Prometheus

Prometheus es una base de datos de series temporales diseñada para la recolección y almacenamiento de métricas. Se basa en un modelo pull, donde se consulta periódicamente los endpoints expuestos por las aplicaciones instrumentadas. Estas métricas se almacenan como valores numéricos etiquetados, lo que permite clasificar y analizar fácilmente el comportamiento de sistemas y servicios.

En este proyecto, Prometheus se integra como fuente de datos principal para métricas en Grafana, facilitando la visualización y la correlación con otras señales del sistema. Su potente lenguaje de consultas, PromQL, posibilita tanto la generación de visualizaciones como la definición de alertas, brindando soporte a la detección temprana de anomalías y a la automatización de respuestas [41] [42].

3.4.2. Loki

Loki es un sistema escalable de agregación y búsqueda de logs, diseñado bajo un enfoque de bajo costo y alta eficiencia. A diferencia de otras soluciones de logging que indexan todo el contenido de los registros, Loki solo indexa sus etiquetas principales

(como servicio, instancia o flujo). Esta estrategia de diseño reduce los requisitos de almacenamiento y procesamiento, lo que lo hace eficiente en cuanto a los requisitos de infraestructura. Su lenguaje de consulta, LogQL, inspirado en PromQL, facilita la relación directa entre logs, métricas y trazas, fortaleciendo así el análisis integral de incidentes. Esta característica lo posiciona como una herramienta especialmente adecuada para arquitecturas que generan grandes volúmenes de datos [43] [44].

3.4.3. Tempo

Tempo es un sistema de trazas distribuido orientado al almacenamiento económico y escalable de trazas. Su diseño evita la dependencia de índices complejos, lo que le permite gestionar grandes volúmenes de spans sin comprometer el rendimiento.

Tempo es compatible con estándares de telemetría como OTLP, Jaeger y Zipkin, lo que asegura la integración con distintos entornos de instrumentación. A partir de su alta capacidad de almacenamiento, es posible comprender el recorrido de una petición en entornos distribuidos permitiendo enaltecer la disponibilidad del sistema [45] [46].

3.4.4. Gafana

Grafana actúa como la interfaz unificada del stack, proporcionando dashboards interactivos, exploración de métricas, logs y trazas, además de la configuración de alertas. Su principal característica es la capacidad de integrar múltiples fuentes de datos.

Grafana ofrece dashboards interactivos que permiten visualizar métricas en tiempo real, explorar logs y analizar trazas distribuidas. Además, Grafana facilita la configuración de alertas basadas en reglas de PromQL y la combinación de múltiples fuentes de datos. Este nivel de extensibilidad posiciona a Grafana como un componente clave en ecosistemas distribuidos y diversos, donde los equipos de operaciones requieren una vista consolidada y transversal de su infraestructura [47] [40].

Para ilustrar cómo se combinan todos estos elementos, en la Figura 5: Diagrama arquitectónico de herramientas de Observabilidad se presenta una arquitectura típica integrando OpenTelemetry con el stack de Grafana (Grafana, Loki, Tempo) y Prometheus. Las flechas sólidas representan el flujo de datos de telemetría (trazas,

métricas, logs) a través del protocolo OTLP (gRCP/HTTP). Las flechas punteadas representan consultas de Grafana hacia los backends correspondientes utilizando PromQL. En esta arquitectura, cada microservicio instrumentado (Service A, Service B) incluye el SDK de OpenTelemetry. Este SDK captura logs, métricas y trazas y los envía al OpenTelemetry Collector, usando el protocolo OTLP generalmente sobre gRPC (o HTTP)

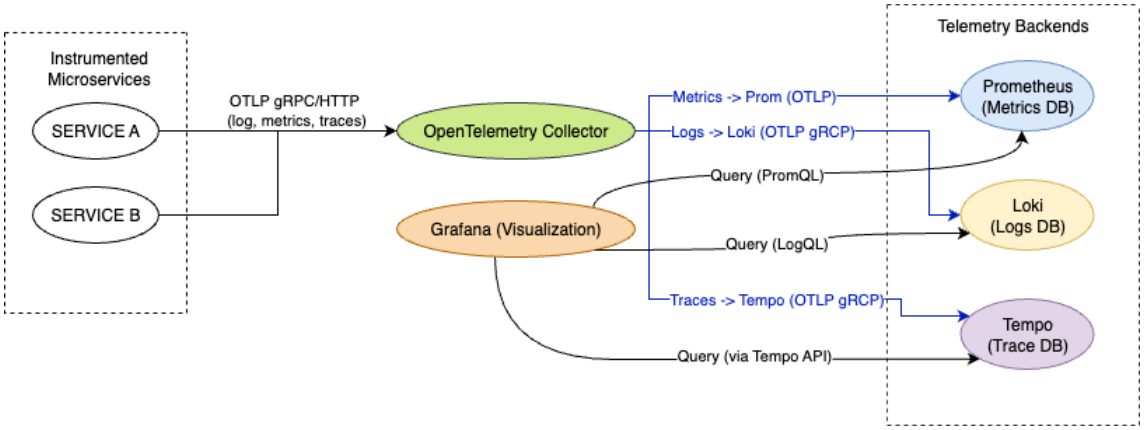


Figura 5: Diagrama arquitectónico de herramientas de Observabilidad

3.5. Modelo de AI

Una de los problemas abordados por AIOps es la detección de anomalías. Con el auge de la inteligencia artificial, existe una diversa variedad de modelos de aprendizaje automático los cuales permiten realizar detecciones rápidas permitiendo a los equipos ser proactivos ante la aparición de posibles fallas.

Para el módulo de AI encargado de la aplicación del algoritmo se realiza la implementación de distintas fases. Primeramente es necesario definir el objetivo y un problema al cual se apunta obtener una solución. El objetivo debe estar asociado a las necesidades del negocio y debe ser posible cuantificar. En segundo lugar, la recopilación y preparación de datos. Un componente vital para el correcto funcionamiento de cualquier software basado en Machine Learning es la cantidad y calidad de los datos. Para la fuente de datos serán utilizadas las señales telemétricas recolectadas y transformadas en primera instancia por las herramientas de observabilidad. Posteriormente previo al entrenamiento del modelo se deben realizar prácticas de limpieza a los datos. En la fase siguiente, con

la información recibida, se procede a la elección del modelo open-source. Existe una amplia variedad como Random Forest, Isolation Forest, Regresión logística, Reinforcement Learning y otros, cada uno adecuado para diferentes tipos de problemas.

3.5.1. Isolation Forest

El algoritmo open-source de aprendizaje automático no supervisado Isolation Forest (IF) resulta una elección adecuada en vistas de los objetivos planteados. IF se basa en la idea de que las anomalías son datos poco frecuentes y diferentes, por lo que resultan más fáciles de aislar que las observaciones normales. El método funciona creando múltiples árboles de decisión aleatorios y evaluando qué tan rápido un dato queda separado del resto. Los valores que se aíslan con pocas divisiones son marcados como posibles anomalías. Este enfoque permite detectar comportamientos inusuales de manera rápida y eficiente, incluso en grandes volúmenes de datos [29] [48].

IF se ve beneficiado al mantener volúmenes de datos elevado para su entrenamiento, esto se debe a que al tener volúmenes limitados suele generar sobreajustar pudiendo generar falsos positivos que impacten a la toma de decisión por los equipos. Al ser un algoritmo no supervisado, no depende de etiquetas previas, lo que lo hace especialmente útil en escenarios de detección temprana de incidentes [49].

Por otra parte, IF no requiere supuestos previos estrictos sobre la distribución de los datos. Esto otorga flexibilidad sobre fuentes heterogéneas y señales con distinta granularidad como los datos telemétricos (logs, metricas y trazas). Esta característica es fundamental a la hora del entendimiento de anomalías sutiles y multivariables.

Por ejemplo, el sistema puede mostrar un incremento del 15 % en la métrica de compras pendientes por farmacia. Si se dependiera únicamente de un umbral fijo, como disparar una alerta al superar las 100 compras pendientes, esta variación pasaría inadvertida mientras el total se mantenga por debajo del límite. Sin embargo, al considerar señales adicionales como un aumento leve de logs con errores 403 al confirmar compras y mayores retrasos en las trazas del servicio de aprobación de pedidos, la situación puede ser diferente. De forma aislada, estas evidencias no activarían ninguna alerta, pero

analizadas en conjunto, IF puede reconocer la combinación como un patrón anómalo que anticipa una posible degradación del flujo de negocio.

La cuarta fase comprendida en la implementación del modelo consiste en el entrenamiento del modelo. Los datos utilizados en esta etapa representan las operaciones del negocio en un estado “normal”. En quinto lugar se procede la evaluación del modelo, siendo esta fase crucial para determinar su funcionamiento y validez. Se consideran nuevos datos no conocidos de forma de evaluar su eficiencia [50].

4. Implementación

4.1.Contenedores de aplicación

Con el auge de los sistemas cloud-based la contenedores han surgido como la evolución natural y eficiente de la virtualización. Al favorecer el aislamiento de dependencias y la entrega continua permite que los despliegues se vuelven reproducibles y trazables. Esto se una condición necesaria para obtener datos de observabilidad confiables. Contar con las aplicaciones en contenedores es una caracteristica y mejora fundamental para el desarrollo de prácticas.

Para llevar a cabo esta mejora en el proyecto, se implementó como etapa inicial la estandarización de la ejecución del sistema. Fue necesaria la creación de imágenes Docker para cada componente por separado. Una imagen encapsula sistema operativo, runtime y librerías. El resultado es un artefacto portable y versionable que se ejecuta igual en Docker Compose. Docker Compose promueve la definición y ejecución de aplicaciones multi-contenedores.

El principal desafío en el desarrollo de los contenedores vino a través de la conexión entre la API y la base de datos. La base de datos necesitó levantarse primero y estabilizarse. Esto en parte se debe a que la instanciación de la base de datos requiere de un script auxiliar que permita la insercción de datos. Un ejemplo de esto, es la necesidad de contar con un usuario administrador en el sistema.

Se tiene como resultado que el entorno completo del sistema es iniciado bajo un único comando de Docker Compose, asegurando la automatización y rápida instanciacion de los componentes.

4.2. CI/CD Pipeline

El proceso de Integración Continua y Entrega/Despliegue Continuos (CI/CD) se ha consolidado como piedra angular del desarrollo de software moderno. En AIOps, los pipelines convierten la entrega de software en un ciclo automatizado de datos y aprendizaje automático. Por ello, la adopción de CI/CD en este proyecto resulta necesaria para sustentar las prácticas de AIOps.

Esta mejora fue desarrollada en el proyecto en un conjunto de etapas. El desarrollo de pipeline suele ser iteractivo en cuanto a configuracion de variables de entorno necesarias. Usualmente, las variables se agregan de acuerdo a la necesidad de forma de mantener restringida la cantidad expuesta por motivos de seguridad. Las credenciales de Amazón Web Services son conmunmente mandatorias en etapas tempranas.

Siendo un pipeline básico, se definen dos grandes etapas:

- Etapa: Build – Test
    - Se ejecutan los test unitarios de la aplicación. Dado el alcance del proyecto, únicamente se ejecutan pruebas unitarias de la API. En caso de fallo, el pipeline se interrumpe.
    - Mediante el uso de la herramienta Docker Compose se realiza la etapa de copilación de las imagenes. En caso de error, el pipeline se interrupe permitiendo al desarrollador tener rápida visibilidad del estado.
    - Finalmente se realiza el envío de las imágenes creadas a Amazon Elastic Container Registry (ECR) donde son alojadas previo a su despliegue.
- Etapa: Deploy
    - El despliegue tiene como pre-condicion la etapa Build-Test.
    - El despliegue del sistema consiste en obtención de las imágenes Docker desde ECR y posterior inicialización de los contenedores mediante Docker Compose.

El pipeline es ejecutado mediante un evento “push” de Github sobre la rama “develop”. Esta decisión se encuentra basada en la supresión de complejidad del pipeline para su ejecución.

4.3. Implementación de telemetría

La implementación del telemetría conlleva distintas etapas, las cuales contaron con desafíos diversos. Por ejemplo, la migración de versiones descontinuadas a versiones actualizadas y mejoradas de librerías y plataformas.

La API utilizada como backend, fue desarrollada en el framework .NET 6.0. Debido a esto, en primera instancia se desarrolló el manejo de logs, traces y métricas a tavés del estándar de OpenTelemetry recomendado para dicha versión.

Durante la implementación las trazas y métricas no tenían el comportamiento esperado. El primer hallazgo fue que .NET 6.0 dejó de recibir soporte oficial por parte de Microsoft en noviembre de 2024. Esto implica que no se publican más actualizaciones de seguridad ni correcciones de errores críticos, lo que convierte su uso en un riesgo en producción. Por otra parte, la necesidad de mantener las bibliotecas actualizadas se vuelve especialmente crítica en proyectos que incorporan herramientas de rápida evolución como OpenTelemetry. Las herramientas como Grafana, Loki, Tempo y Prometheus exigen un versionado actualizado para poder explotar de mejor forma sus capacidades. Si bien OpenTelemetry sigue declarando compatibilidad con .NET 6.0, su documentación más reciente recomienda explícitamente utilizar .NET SDK 8.0 o superior para seguir correctamente las guías de instrumentación, reflejando así una migración tácita hacia entornos más modernos [51].

Tras la migración de la API a .NET 9.0 conjuntamente con la actualización a su ultima versión de las herramientas, el sistema comenzó a emitir, recolectar y exportar datos telemetricos satisfactoriamente.

La etapa de implementación de observabilidad fue mucho más que una integración técnica. Supuso un proceso de aprendizaje profundo sobre cómo instrumentar correctamente una aplicación .NET moderna, comprender la naturaleza declarativa de OpenTelemetry y superar barreras de compatibilidad.

4.4. Aplicación de modelo de AI

La implementación del módulo de AI basado en Isolation Forest se realizó utilizando la librería scikit-learn (sklearn), un framework open-source de aprendizaje automático ampliamente adoptado en la comunidad de ciencias de datos [29] [30].

Los datos que conforman el dataset de entrenamiento corresponden a información telemétrica recolectada durante las simulaciones de operación en estado <normal>, es decir, sin fallos críticos ni anomalías deliberadas. Estos datos incluyen métricas, logs y

trazas capturados por el sistema de observabilidad instrumentado con OpenTelemetry. La cantidad de datos viene dada por la simulación de tres días de operación del sistema.

Para la construcción de las variables de entrada se diseñó la función build_features, cuyo propósito fue transformar los logs, métricas y trazas en variables numéricas y categóricas normalizadas. Esta etapa fue clave para garantizar que las señales heterogéneas pudieran ser interpretadas de forma unificada por el modelo. Se incluyeron tanto atributos técnicos como códigos HTTP y latencias. También fueron incluidas métricas de negocio.

El entrenamiento del modelo se realizó a través de la práctica común de dividir el dataset en 70 % de datos para entrenamiento y 30 % para validación. Así es posible evaluar la capacidad del algoritmo de sobreajustar sobre datos no conocidos. El sobreajuste sucede cuando el modelo aprende los datos de entrenamiento pero falla al momento de predecir nuevos resultados con nuevos datos [52]. El parámetro contamination, que define la proporción estimada de anomalías en los datos fue definido con el valor automático estándar de forma de simplificar la prueba.

Un desafío al momento del entrenamiento consistió en reducir el ruido de los datos ya que no toda desviación representa un incidente real. Por ejemplo, un alza en una métrica especifica puede verse relacionado a un evento de negocio que impuse esta alza. Para esto, a traves de prácticas de limpieza y normalización de valores extremos. Por otro lado, una calibración errónea del parametro de contamination puede llevar a falsos positivos o falsos negativos, por lo que se requirió un conjunto de iteraciones donde finalmente se concluyó que el valor estandar se alinea el dominio de negocio.

En la evaluación con datos nuevos, simulados con anomalías identificadas como cuellos de botellas el modelo demostró su capacidad de detectar patrones que escapaban a una operación normal. Por ejemplo, al correlacionar logs referidos al inicio de una solicitud de concreción de compra con demoras en trazas del servicio de aprobación. Esto identifica patrón anómalo que anticipaba degradación. Desde el punto de vista técnico, esto confirmó la ventaja del enfoque multivariable frente a reglas aisladas que solamente contemplan ún valor especifico sin tener en cuenta un contexto definido de negocio.

5. Infraestructura de prueba

5.1. Escenario de prueba

El escenario de prueba se diseñó para simular en condiciones controladas y reproducibles, la operación del sistema en un ambito realista. Con esto, se procura medir la eficiencia de las prácticas de AIOps desarrolladas con el fin de medir el impacto en la disponibilidad, resiliencia y observabilidad.

Por un lado, se procura evaluar la arquitectura diseñada para la capacidad del sistema de ser observable. Permitiendo así, evidenciar el impacto en los equipos de operaciones de forma de mejorar su eficiencia ante distintos posibles casos como la operación nomal, una posible anomalía o falla. Por otro lado, la evaluación de las prácticas de desarrolladas de AIOps de forma de medir su incidencia y efectividad ante los posibles escenarios posibles. Estos escenarios son vistos en perspectiva de la operativa de negocio (Por ejemplo, aumento de flujo de compras en una farmacia), o la operativa técnica la cual involucra fallas del sistema (Por ejemplo, caídas del sistema o bugs).

La estrategia se centra en la simulación mediante scripts de distintos escenarios los cuales permitan medir el impacto de las prácticas y herramientas. Los scripts de prueba son el elemento básico de la automatización. Un script de prueba consiste en una serie de comandos o eventos almacenados en un archivo de lenguaje de script para ejecutar e informar los resultados. Puede contener decisiones lógicas que afectan la ejecución del script, creando múltiples rutas posibles, valores constantes y variables cuyos valores cambian durante la reproducción. La ventaja del proceso de desarrollo de scripts de prueba es que estos pueden repetir la misma instrucción muchas veces en bucles, cada vez con datos diferentes [53]. Esto permite realizar simulaciones fácilmente sobre la API, como por ejemplo, carga de datos, flujos de negocio, condiciones anómalas, etc. De este modo, las pruebas dejan de ser un ejercicio aislado y se convierten en un mecanismo de aprendizaje continuo que retroalimenta el diseño, la operación y las prácticas de AIOps.

Los escenarios de prueba que se efectuan en tres tipos de simulación. La carga inicial prepara un estado realista del dominio. Esto es por ejemplo, creación farmacias o usuarios las cuales sirvan como set de datos reutilizables garantizando repetibilidad y control de

variables en distintas corridas. La simulación de negocio refleja flujos cotidianos como la solicitudes de reposición, creación de compras o aprobaciones. Esto se realiza a través de distintas coordinaciones de ejecucion y fases de carga temporizadas lo cual permite definir una variedad de fases en la operativa. Estas fases están alineadas con las métricas de negocio de forma de "estresar" el sistema generando oportunidades de evaluación para las prácticas. Como tercer tipo de simulación, se encuentra la siembra de fallos donde se introducen errores controlados. Unos ejemplos de estos errores pueden ser la generación de solicitudes malformadas, acciones no autorizadas, excepciones deliberadas y desconexión de dependencias. Con esto, es posible observar degradación del sistema y posterior recupeación, correlación entre trazas y logs, etc.

El valor de este enfoque no solo es posible identificarlo en la reproducción condiciones de operativa real sino que se obtiene un conjunto de datos observables sobre el que es posible entrenar y evaluar técnicas de AIOps. Por ejemplo, la detección de anomalías ante un aumento de solicitudes con código HTTP 500.

La ejecución de pruebas basadas en escenarios cercanos a la operativa real generan un aporte fundamental para evaluar la efectividad de las prácticas de AIOps. Al reproducir flujos de negocio, se generan evidencias que permiten medir la capacidad del sistema de detectar anomalías, correlacionar señales y anticipar incidentes antes de que lleguen a producción. Este enfoque no solo facilita la validación de los mecanismos de observación y análisis inteligente, sino que también repercute directamente en los atributos de calidad: la disponibilidad se ve fortalecida al evaluar la continuidad y los tiempos de recuperación. La resiliencia por su parte, se enaltece al comprobar la tolerancia a fallos y la capacidad de retornar al estado estable luego de la degradación. La observabilidad por su parte mediante la correlación de métricas, logs y trazas muestran con precisión el comportamiento interno del sistema.

5.2. Implementación de las pruebas

Los scripts utilizados para los tres enfoques distintos fueron desarrollados de forma que puedan ser ejecutados de forma independiente. Estas pruebas a nivel de API, donde no se simula la interfaz gráfica sino las solicitudes al servidor.

5.2.1. Script 1 – Carga inicial de datos

El primer script se encarga de preparar el escenario base para las pruebas, poblando el sistema con datos iniciales necesarios. El script realiza las siguientes acciones automatizadas (utilizando llamadas HTTP al servidor local del backend):

- Inicia sesión como administrador.
- Crear farmacias al sistema. El script guarda en memoria la lista de farmacias creadas con sus IDs y nombres.
- Genera invitaciones de usuario para cada farmacia en el sistema. Por cada farmacia se crea primero una invitación para un usuario Owner (propietario) de esa farmacia, y luego al menos 2 invitaciones para usuarios Employee (empleados) de esa misma farmacia.
- Crear efectivamente las cuentas de usuario a partir de invitaciones generadas. El script guarda todos los usuarios creados en una lista USERS
- Login para cada usuario empleado para obtener tokens de acceso individuales. Esto simula que cada empleado inicia sesión en el sistema.
- Crear el inventario inicial de medicamentos (drugs) en cada farmacia.
- Guardar las listas resultantes en archivos JSON locales para su uso. Estos datos guardados a nivel local sirven de datos de entradas para la simulación de negocio. pueden alimentar el siguiente script.

Al final del script contamos con un sistema poblado con varias farmacias y usuarios con distintos roles, similar a un entorno real de uso donde varias farmacias operan en paralelo.

5.2.2. Script 2 – Simulación operativa

El segundo script se refiere a la simulación de uso del sistema. Se simulan las operaciones de negocio diarias que realizan los usuarios (empleados y propietarios de farmacias). El diseño del script está alineado con los objetivos de métricas de negocio en mente:

- Cantidad de solicitudes de reposición de stock por mes.
- Cantidad de compras en estado “pendiente” por farmacia.
- Valor total de ventas por hora.

- Tiempo promedio de las compras en estado “pendiente” por farmacia.

La simulación genera datos que alimentan directamente estas métricas:

- Cantidad de solicitudes de reposición de stock por mes.
- Compras pendientes por farmacia.
- Valor total de ventas por hora.
- Tiempo promedio de compras pendientes por farmacia

A partir de los datos previamente cargados por script 1, se realizan múltiples iteraciones de acciones, midiendo la respuesta del sistema bajo diferentes patrones de carga. La estructura del script se divide en tres secciones fuertemente marcadas:

- Iniciar sesión de múltiples usuarios al mismo tiempo. Los token se agrupan por farmacia y roles para fácil manejo.
- Operaciones de negocio a través de métodos. El script define una serie de funciones que encapsulan llamadas HTTP a la API para realizar operaciones específicas. Estas funciones se encargan de invocar los endpoints de la API durante la simulación, automatizando lo que en la vida real serían acciones en la interfaz de usuario. Permitiendo así simular transacciones completas:
    - El empleado hace solicitudes de reposición de stock
    - Aprobación o rechazo de solicitudes de stock lo cual genera un cambio de estado de la solicitud de “pendiente” a “aprobado”.
    - Concreción de una compra.

  Por ejemplo: Un StockRequest representa una solicitud de reposición de stock que un empleado pide al propietario (por ejemplo, cuando la farmacia se queda sin cierto medicamento y pide más). Para eso, una función con el nombre de create_stock_request(token_empleado, details), realiza una solicitud HTTP con método POST al endpoint /api/StockRequest con un cuerpo JSON conteniendo detalles de medicamentos y cantidades solicitadas. La función envía la solicitud (autenticada con el token de un empleado) y espera una respuesta de código 200.

- Simulación de modos concurrentes.

A través de funciones basadas en threads, se realizan múltiples acciones de forma aleatoria y en paralelo de forma de simular distintas simulaciones de escenarios reales que pueden existir.

Por un lado se encuentra la simulación del proceso de solicitud y aprobación de stock que involucra tanto a los empleados como a los dueños. A través de la función definida simulate_stock_requests, se simula que en una farmacia se realizan n solicitudes de reposición de stock seguidas, y luego el propietario revisa esas solicitudes. Así, esta función encapsula el flujo completo de Solicitudes de Stock: un empleado pide reabastecimiento (varias veces) y el dueño las atiende (principalmente aprobando). En la simulación completa, se lanzará esta función varias veces para distintas farmacias concurrentemente. Del mismo modo, se realiza la simulación de compras por parte de usuarios anónimos. Para eso la función simulate_purchases simula n compras de clientes en una farmacia dada.

De forma de realizar la simulación y generar un flujo acorde a lo real, se implementó una serie de fases programadas con distintos modos de carga. Cada modo corresponde a un patrón de carga específico que modifica la intensidad de las operaciones simuladas, con el objetivo de recrear distintos escenarios de uso y pruebas de estrés.

La secuencia definida para la ejecución es la siguiente:

- <normal> (5 minutos)
- <flat> (5 min)
- <drop> (2 min)
- <normal> (5 minutos)
- <flat> (5 min)
- <spike> (1 min)
- <normal> (5 minutos)
- <flat> (5 min)
- <anomaly> (30 segundos)
- <normal> (5 minutos)
- <flat> (5 min)
- <anomaly_p7> (30 segundos)

- <normal> (5 minutos)
- <flat> (5 min)
- <bottleneck> (5 min)
- <normal> (5 minutos)
- <flat> (5 min)

El modo <normal> representa el comportamiento típico del sistema bajo una carga de uso esperada. Para cada farmacia la simulación genera un número moderado/alto de operaciones simulando un tráfico regular donde los empleados hacen algunas solicitudes de reposición y atienden a varios clientes, de forma continua. Es útil para ver el desempeño del sistema en condiciones estables pudiendo tener medir el umbral normalizado de las métricas de negocio.

Por otro lado, el modo <flat> corresponde a un periodo de actividad moderada/baja. Se genera un nivel bajo de operaciones lo cual simula poca demanda. El objetivo es observar cómo se comporta el sistema con carga mínima asegurando que no existan problemas en de disponibilidad de recursos ante repentinos cambios. A nivel métricas, este modo lleva a que las solicitudes y ventas bajen comparado al modo <normal> prácticamente a cero.

En el modo <drop> se simula un corte completo de actividad durante un intervalo de tiempo limitado. Con esto se puede representar un escenario de interrupción de operaciones pero sin fallas. Con esto, se puede simular en los dashboard de visualización por monitoreo una baja que puede disparar alertas de ser necesario. De cara de logs, metricas y traces no se genera datos.

Las simulaciones de picos de cargas sirven para estresar la capacidad de la aplicación. Prueba si el sistema soporta un volumen alto de tráfico en poco tiempo. Así cómo también comprobar la respuesta de las bases de datos a decenas de inserciones y consultas concurrentes midiendo si existe degradado de servicios. El modo <spike> genera este comportamiento concentrado en una parte del sistema. Se enfocan en un estrés intenso en una sola farmacia emulando un posible caso de uso real como por ejemplo una promoción o ciberataque. En términos de métricas de negocio, un pico se refleja en el incremento de ventas por hora y del número de compras pendientes. Además de un posible aumento de rechazos por falta de stock.

El modo <anomaly> procura poner en práctica la introducción de comportamientos anómalos específicos para simular fallos lógicos o situaciones de negocio inusuales. Esto nos permite probar no solo comprobar el sistema a nivel observabilidad, sino también de proceso y reacción del equipo. El modo en cuestión genera un volumen inusualmente alto de solicitudes de reposición de stock. El objetivo con esto es ver cómo el sistema y especialmente el módulo de gestión de stock maneja tal flujo de pedidos. Así se podría simular un error de sistema que duplica solicitudes o un ciberataque.

El modo <anomaly_p7> es un caso particular sobre la farmacia con ID 7. Procura forzar una situación anómala de compras de clientes las cuales se mantienen en un estado "pendiente" por un tiempo indeterminado. El propósito conceptual es generar un caso donde compras en una farmacia no sean atendidas lo cual puede repercutir a nivel negocio en falta de personal o porque el sistema dejó de procesarlas. Así la métrica "cantidad de compras en estado pendiente por farmacia" sube anormalmente en una farmacia. Con esto se probaría si detectamos acumulación de compras pendientes (posible indicador de un problema de flujo).

El modo <bottleneck> provocar un aumento del tiempo promedio de compras pendientes para una farmacia aleatoria. De esta forma, dicha métrica sube anormalmente generando una simulación a una anomalía que puede ser leída a nivel negocio como caída del servicio, lo cual afecta a los usuarios finales. Este modo permite a las prácticas de AIOps identificar esta situación generando la alerta correspondiente a los equipos de operaciones.

5.2.3. Script 3 – Siembra de fallo

El tercer punto de la simulación pretende introducir deliberadamente fallos en el sistema para evaluar su resiliencia y observabilidad bajo errores.

A través de la provocación de un error controlado en algún componente del sistema durante la simulación permite medir cómo se registra (logs) el fallo y cómo reacciona el sistema a nivel de monitoreo. Se inyectan errores para comprobar que el sistema no solo soporta cargas, sino que también falla de forma controlada y alerta apropiadamente a los operadores [54].

En práctica, un script de inyección de fallos actúa de acuerdo a las formas que se procura poner a prueba:

- Realizar peticiones malformadas o con datos incorrectos a propósito para generar errores 400/404 en la API. Esto permite medir el correcto manejo de las devoluciones de mensajes para conocer fácilmente el error. Se analiza luego el correcto error devuelto (Codigo HTTP) y explicación de la falla. Además, el correspondiente log asociado que permite medir a posterior, el volúmen en un servicio de observabilidad.
- Realizar acción prohibida a nivel lógico. Esto puede ser, por ejemplo, un empleado intentando realizar tareas del dueño. Acorde al sistema, esto sería indebido por lo que se debería emitir 403 (Forbidden) y medir el registro del intento a nivel logs. Usualmente es utilizado este tipo de pruebas para seguridad del sistema.
- Forzar falló críticamente al sistema. Comúnmente indicado con un código HTTP 500. Para eso, se envía un payload conocido que provoca la excepción no controlada en el servidor. Comúnmente asociado a un bug conocido en el sistema. Esto verifica si el sistema captura excepciones o si se cae el servicio. Ayudando a tener mediciones acordes al sistema.
- Simular la caída de un servicio dependiente. Voluntariamente se apaga un servicio crítico como lo es la base de datos de forma que se permita medir cómo reacciona el sistema ante un eventual caso.

  Debido a la arquitectura actual, simular una “desconexión” a la base de datos genera en el sistema una serie de fallos en cascada que deberían ser fácilmente identificables por logs y traces.

6. Resultados

6.1. Implementación

La implementación del proyecto permitió consolidar una arquitectura moderna y alineada a las prácticas de AIOps. La estandarización en contenedores mediante Docker y Docker Compose habilitó un entorno reproducible y automatizado. El pipeline de CI/CD integró etapas de build, test y deploy garantizando la entrega continua y la disponibilidad de los contenedores, reduciendo la distancia entre desarrollo y operaciones. Esto resulta fundamental para los equipos y su entrega de valor en próducción de forma automatizada.

La implementación de telemetría estandarizada con OpenTelemetry por otro lado, constituyó una mejora del sistema alcanzando un nivel de observabilidad que permitió evidenciar comportamientos tanto técnicos como de negocio. Así, se brindaron las bases necesarias para la aplicación de un modelo de AI a estos datos.

El desarrollo de un Dashboard en Grafana permitió realizar el seguimiento de las métricas de negocio definidas, obteniendo información visual en tiempo real del estado de las métricas durante la ejecución de la simulación. Esta representación permitió el fácil monitoreo de valores críticos para el negocio.

El desarrollo de un modelo de AI como el Isolation Forest brindó se transformó en una herramienta fundamental para transformar y analizar los datos telemétricos heterogéneos los cuales pueden luego ser utilizados para la toma de decisiones. Esto confirma el valor del modelo como complemento a umbrales fijos, aportando a los equipos de DevOps una nueva herramienta para enaltecer resiliencia y disponibilidad del sistema.

6.2. Ejecución de las pruebas

La construcción de una infraestructura de pruebas permitió validar el sistema en condiciones cercanas a la operación real habilitando la capacidad de generar telemétricos de calidad para las prácticas de AIOps. A diferencia de un enfoque tradicional centrado en pruebas unitarias, los tres scripts diseñados aportaron un marco que incluye el contexto para evaluar tanto procesos de negocio como escenarios de falla.

Cada script fue desarrollado de forma independiente a nivel de API, lo que aseguró repetibilidad y control sobre cada escenario.

La secuencia de ejecuciones de pruebas controladas conforman la simulación de la operativa del sistema en un ambiente similar a producción. De esta forma, se genera un aumento significativo en la cantidad de logs, métricas y trazas los cuales son procesados en el marco de OpenTelemetry.

Con la ejecución de la simulación de forma continua durante 70 minutos, se obtiene una cantidad alta de datos telemétricos. Aproximadamente 2.600 registros de logs, 13.400 registros de mediciones de métricas y 2.700 spans asociados a las trazas, son los valores para una ejecución total. Este número parece razonable de cara a un sistema constituído en dos capas, sin embargo refleja el desafío de los equipos de DevOps al momento de trabajar con altos volúmenes de datos en arquitecturas distribuidas y sistemas más complejos.

Gracias al uso de la herramienta de visualización Grafana, es posible realizar la correlación entre los datos telemétricos, de forma de detectar posibles anomálias que puedan surgir en la operación en tiempo real. Por ejemplo, en la Figura 6: Métrica - Valor total de Ventas por Hora, se puede reconocer el aumentro pronunciado de este valor. En especial, la farmacia con ID 1 recibe un aumento pronunciado de ventas cercano al doble en un corto período de tiempo.

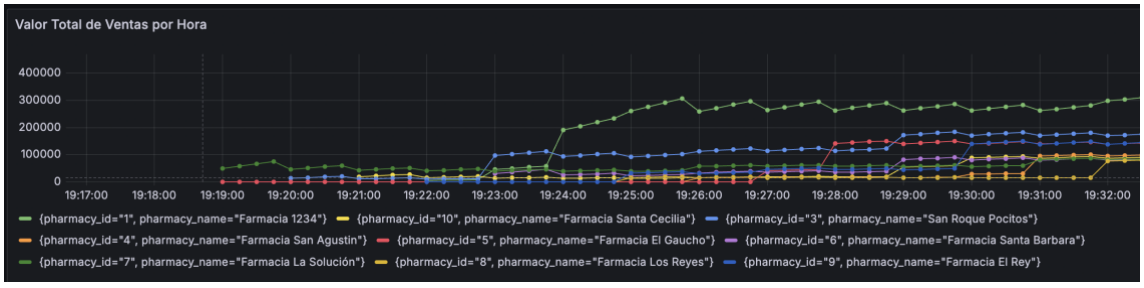


Figura 6: Métrica - Valor total de Ventas por Hora

Del mismo modo, la Figura 7: Métrica - Cantidad de compras pendientes por farmacia representa el valor alcanzado por la métrica “Cantidad de compras “pendientes” por farmacia” durante la simulación. Se observa un valor en rojo notoriamente mayor (79) correspondiente a la farmacia conn ID 1.

Figura 7: Métrica - Cantidad de compras pendientes por farmacia

Estos tipos de escenarios, donde las métricas se encuentran por fuera de sus umbrales normales o poseen comportamientos anómalos, son habituales de suceder en entornos de producción. El análisis de estos valores permiten tomar decisiones de forma rápida y conciza.

De este modo, la infraestructura de pruebas se consolida como un elemento esencial para evaluar el cumplimiento de los objetivos de negocio, generando beneficios tangibles para la organización. La posibilidad de replicar automáticamente escenarios anómalos y de visualizar en tiempo real el comportamiento de las métricas permite pasar de una postura reactiva a una proactiva, fortaleciendo la capacidad de anticipación y mejora continua.

6.3. Detección de anomalías

Todos los escenarios monitoreados se vieron potenciados con la aplicación del modelo de inteligencia artificial, especialmente gracias a la correlación de los datos telemétricos. Esta integración permitió detectar anomalías considerando múltiples factores que, por lo general, las herramientas tradicionales de observabilidad tienden a pasar por alto, aportando así una capa adicional de profundidad en el análisis y comprensión del comportamiento del sistema.

Un ejemplo de este proceso puede observarse en la Figura 8: Ejemplo de predicción de anomalía de cuello de botella, donde se presenta la inferencia de una anomalía identificada por el modelo Isolation Forest correspondiente a la farmacia con ID 1. La anomalía fue clasificada con severidad alta y categorizada como business_bottleneck, aludiendo a un cuello de botella en el flujo de compras pendientes. El modelo detectó que

la cantidad de solicitudes creadas crecía a un ritmo mayor que las resueltas, aumentando el tiempo medio de resolución. Este resultado fue posible gracias al análisis de patrones extraídos de los logs que reflejaban la acumulación de solicitudes sin cierre y trazas distribuidas que evidenciaban demoras en operaciones específicas dentro del sistema.

La combinación de estas fuentes de información permitió al modelo distinguir un comportamiento anómalo que difícilmente habría sido detectado de forma aislada. Por esto, los equipos de DevOps pueden identificar de manera proactiva un cuello de botella operativo y anticipar medidas correctivas antes de que el impacto se reflejara en los indicadores de negocio.

En sintesis, los resultados confirman que la aplicación de AI sobre datos telemétricos de calidad incrementa significativamente la capacidad del sistema para detectar, compreder y prevenir anomalías, validando su valor como componente esencial dentro de las prácticas modernas de AIOps.


```
  {
    "id": "an-2025-10-20 00:13:00+00:00-1",
    "timestamp": "2025-10-20 00:13:00+00:00",
    "pharmacy_id": "1",
    "pharmacy_name": "all",
    "signal": "multivariate_ops",
    "model": "isolation_forest",
    "category": "business_bottleneck",
    "title": "Cuello de botella de negocio: pendientes se acumulan",
    "severity": "high",
    "message": "Las compras 'pendientes' crecen más rápido que se
resuelven y su tiempo medio aumenta.",
    "evidence": {
      "total_events": 51,
      "http_4xx": 0,
      "http_5xx": 0,
      "http_err_ratio": 0.0,
      "app_err_ratio": 0.0,
      "avg_span_ms": 10.752,
      "pending_created": 19,
      "pending_resolved": 13,
      "pending_backlog": 6,
      "pending_avg_s": 32.194
    }
  }
```


Figura 8: Ejemplo de predicción de anomalía de cuello de botella

7. Conclusiones

La industria del desarrollo de software está en constante cambio debido a la necesidad productos que demuestren mayor eficiencia, disponibilidad y calidad. Al facilitar la colaboración y reducir la brecha tradicional entre operaciones y desarrollo, DevOps se ha convertido en un elemento indispensable en este ecosistema en constante evolución. Con la llegada de la IA, aplicar estas capacidades a la operación en entornos DevOps se conoce como AIOps y aporta un potencial de mejora.

7.1. Reflexión sobre el impacto de AIOps

El desarrollo de la prueba de concepto constituyó un ejercicio integral donde se combinaron prácticas de DevOps, telemetría estandarizada y un modelo de inteligencia artificial para la detección de anomalías. Esta experiencia permitió evaluar el sistema en condiciones cercanas a la operación real en producción y generar insumos valiosos para comprender su disponibilidad, resiliencia y observabilidad. A partir de este proceso, fue posible reflexionar sobre el impacto de las prácticas de AIOps, análisis que se presenta en las siguientes secciones.

7.1.1. Contexto DevOps

Las prácticas de AIOps requieren el contexto DevOps como sosten de forma de ser aplicadas con éxito. No se trata únicamente de incorporar algoritmos de Machine Learning sobre datos, sino de contar con un ecosistema cultural y técnico que permita que esos datos sean confiables, representativos de la operación real y con valor al negocio.

En el corriente trabajo se observa el estrecho vinculo mediante las dependencias requeridas para el desarrollo de las prácticas de AIOps. La incorporación de un pipeline de integración y entrega continua fue condición necesaria para sostener la prueba de concepto. La automatización de la infraestructura mediante IaC, junto con la ejecución de pruebas y el despliegue continuo, no solo simplificaron la entrega de software, sino que habilitaron la generación de telemetría confiable para el entrenamiento y evaluación de los modelos de AIOps. Es posible observar que la corresponsabilidad entre desarrollo y operaciones hace evidente la importancia de definir métricas de negocio que vinculen

el comportamiento técnico con el impacto organizacional, resultando imprescindible para el éxito. De esta forma, se evitó reducir a las prácticas de AIOps a un ejercicio técnico logrando integrarlas a un marco donde las decisiones responden tanto a criterios técnicos como de negocio.

7.1.2. Observabilidad

El desarrollo de la observabilidad como atributo de calidad se observa como un prerrequisito indispensable para la incorporación de prácticas de AIOps. El hecho de obtener datos telemétricos no es suficiente; se requiere la capacidad de correlacionarlos, analizarlos y transformarlos en información que sea provechosa para la toma de decisiones.

Evolucionar arquitecturas existentes hacia la observabilidad y crear nuevas que la integren desde su concepción requiere esfuerzos técnicos y organizacionales. Agregar telemetría conlleva un esfuerzo técnico y económico significativo que las empresas no suelen estar preparadas para abordar. Cambios en el diseño de las aplicaciones, incorporación de librerías, adaptación de pipelines y gestión de un volumen creciente de datos son algunos de los desafíos comunes a afrontar. Incorporar observabilidad significa cambiar la forma en que los equipos entienden la operación. Ya no se trata de simples métricas de recursos como CPU o memoria, sino de indicadores de negocio directamente vinculados con la experiencia del usuario. Este cambio demanda un esfuerzo extra: decidir qué medir, cómo hacerlo y con qué propósito. Sobrecargar el sistema de señales puede generar ruido y costos innecesarios, mientras que subinstrumentarlo conduce a la pérdida de entendimiento operativa [2] [54]. La irrupción de estandares open-source como OpenTelemetry, han permitido afrontar parte de estos desafíos reduciendo costos y simplificando parte de la complejidad técnica mediante distintas abstracciones.

La adopción de OpenTelemetry como estandar en el desarrollo de este trabajo permitieron estandarizar la instrumentación de métricas, logs y trazas en un único marco conceptual. Sin embargo, su implementación en el proyecto evidenció las dificultades antes mencionadas. Fue necesario adaptar el sistema a un cambio arquitectónico complejo, versiones actualizadas de librerías y resolver problemas de compatibilidades entre capas. De esta forma se evidencia una realidad frecuente en proyectos de software; las

aplicaciones no están preparadas para ser observadas de manera nativa, y habilitar esta capacidad supone modificar y rediseñar componentes críticos.

El desarrollo de la observabilidad evidenció que no significa simplemente de trabajar con herramientas sino que se requiere adoptar un nuevo paradigma. En arquitecturas distribuidas o basadas en microservicios, esta visión se vuelve aún más crítica. La correlación de métricas, logs y trazas es lo que permite comprender incidentes complejos y detectar anomalías que atraviesan múltiples servicios.

La aplicación de AIOps carece de sustento sin observabilidad. La calidad de los algoritmos depende de la calidad de las señales, y esta calidad solo se alcanza cuando la observabilidad se convierte en un atributo de calidad. En la prueba de concepto, el esfuerzo invertido en instrumentar, estandarizar y correlacionar los datos fue tan importante como el desarrollo del modelo de inteligencia artificial.

7.1.3. Metricas asociados a objetivos de negocio

En ingeniería de software se disponen de una amplia variedad de métricas. Entre ellas, se encuentran las métricas tradicionales que son ligadas al rendimiento de los sistemas como CPU y memoria. Las métricas DORA permiten evaluar la productividad y su capacidad de entrega de los equipos de DevOps y las métricas de aplicación, centradas en el comportamiento de los servicios en ejecución. Sin embargo, con frecuencia estas métricas se definen de forma aislada, sin una correlación clara con los objetivos de negocio. Esto genera indicadores que describen partes del sistema o del proceso, pero que no reflejan el verdadero impacto en las organizaciones ni ofrecen información útil para la toma de decisiones.

Por esto, las métricas deben estar diseñadas desde los objetivos de negocio. De esta forma adquieren sentido y valor. Una métrica como “total de ventas confirmadas por hora” no debe ser vista solo como una cifra operativa, sino una medida directa del desempeño del negocio. De la misma forma que la frecuencia de despliegues por parte de los equipos no representa únicamente su velocidad, sino su capacidad de proveer mejoras constantes en producción. Cuando los indicadores están alineados a los objetivos estratégicos, los

equipos logran interpretar mejor los síntomas técnicos, priorizar incidentes y enfocar sus esfuerzos en lo que realmente sostiene la continuidad del servicio.

Esta cambio de perspectiva permite demostrar por ejemplo, que los datos telemétricos deben ser implementados de forma que tengan un propósito claro. La generación de logs simplemente por tenerlos o recolectar métricas indiscriminadas genera ruido, eleva los costos de almacenamiento y, sobre todo, entorpece la capacidad de los equipos de responder ante incidentes. Una alerta que no está asociada a un objetivo real termina siendo ignorada y esto repercute en los equipos de operaciones.

En la prueba de concepto realizada, la definición de métricas de negocio guió la construcción de escenarios de prueba y permitió evaluar el sistema en condiciones cercanas a la operativa real en producción. Indicadores como la cantidad de compras pendientes por farmacia o los tiempos promedio de resolución de una compra no solo sirvieron para medir resultados, sino que también orientaron las simulaciones y prácticas de AI utilizadas en la busqueda de mejorar la disponibilidad y resiliencia del sistema.

Por tanto, toda estrategia de AIOps debe pensarse desde los objetivos de negocio, ya que son ellos los que otorgan sentido a la observabilidad y convierten a la AI en un recurso accionable para los equipos DevOps.

7.1.4. Infraestructura de pruebas

En la industria del software, las pruebas automátizadas suelen asociarse principalmente a casos de prueba funcionales o unitarios. Esto es una etapa necesaria dentro del ciclo de desarrollo, pero rara vez se lo concibe como una infraestructura en sí misma. Sin embargo, de cara a la incorporación de prácticas de AIOps, contar con una infraestructura de testing sólida se vuelve un requisito esencial para su validación.

La ejecución de pruebas unitarias sobre fragmentos de código no es suficiente. Debido a la complejidad creciente de los nuevos sistemas, se hace indispensable construir un entorno capaz de simular condiciones operativas realistas de forma automática y repetible. Por tanto, las pruebas no solo contemplan entradas y salidas, sino que también toma relevancia el contexto en donde se desarrollan y bajo los objetivos de negocio que las impulsan. En particular, resulta necesario recrear entornos de producción complejos,

donde se concentran múltiples servicios, dependencias externas y cargas variables. De esta forma los equipos puedan anticipar fallos y validar la resiliencia en escenarios cercanos a la realidad en producción.

La capacidad de generar datos sintéticos comparables con los de producción e introducir fallos controlados es esencial para lograr generar simulaciones con contexto. Las simulaciones automatizadas no son pruebas aisladas, sino mecanismos sistemáticos de validación. Cada simulación se diseña para ejecutarse de forma repetible y escalable permitiendo la retroalimentación constante de datos para entrenamiento y validación del modelo de AI.

La inserción de fallos deliberados es un factor fundamental en este tipo de infraestructura. La inserción deliberada de fallos no debe entenderse como un riesgo, sino como una práctica que fortalece la capacidad de respuesta de los equipos de operaciones. La experiencia de Netflix con Chaos Engineering demuestra que provocar errores controlados en entornos reales permite a los ingenieros validar la tolerancia a fallos de sus sistemas y detectar vulnerabilidades invisibles bajo condiciones normales de operación. El enfoque de este tipo de pruebas cambia de un proceso de verificación a un proceso continuo de aprendizaje sobre resiliencia [2].

La incorporación de es lógica en la prueba de concepto aporta beneficios concretaos y tangibles. Por un lado, permite evaluar no solo el comportamiento funcional del sistema, sino también su capacidad de observación, recuperación y continuidad frente a incidentes inesperados. Cada una de estas fallas aporta señales que evaluadas por el modelo de AI permitieron evaluar no solo así mismo, sino la resiliencia del sistema y la eficacia de la observabilidad implementada. De cara a los equipos, otorga métricas prácticas sobre tiempos de detección, propagación de fallos y capacidad de recuperación, que son esenciales para relacionar observabilidad con disponibilidad. De este modo, la inserción de fallos controlados se convierte en una herramienta valiosa para reforzar la confianza en el sistema y para entrenar a los equipos en escenarios de respuesta más cercanos a la realidad operativa.

Contar con una infraestructura de testing no es un complemento, sino un componente esencial en las prácticas de AIOps. Realizando pruebas automáticas antes de llegar a

producción, reduce riesgos y fortalece atributos de disponibilidad y resiliencia. La simulación es un recurso estratégico que acompaña tanto al desarrollo como a la operación.

7.1.5. Impacto del módulo de AI

Haber realizado la prueba de concepto permite evaluar las posibilidades y limitaciones de AIOps. La utilidad obtenida luego de la aplicación de distintas prácticas depende de diversos factores. Un ejemplo de estos puede ser los cambios arquitectónicos requeridos por AIOps y el volumen/calidad de los datos necesarios para la implementación del modelo.

La implementación del módulo de AI basado en Isolation Forest (IF) responde a la necesidad de detectar anomalías sin depender de etiquetas previas y con capacidad de adaptación a datos heterogéneos en tiempo real. El algoritmo se ajusta de forma natural a los datos telemétricos permitiendo un análisis integrado de señales que antes se evaluaban de manera aislada. En comparación a umbrales estáticos, IF reconoce combinaciones sutiles de patrones que revelaban degradación temprana del servicio aportando un nivel de análisis profundo de las operaciones. Al acortar la distancia entre el inicio de la degradación y la detección accionable, habilita respuestas antes de que el problema se convierta en caída o en violación sostenida de SLO, elevando la capacidad del servicio a en resiliencia y disponibilidad. Esto permite ajustes en la operación en patrones inusuales que permiten la reducción de falsos positivos y priorizar anomalías con mayor impacto. De esta forma los equipos de operaciones pueden enfocar sus esfuerzos en incidentes relevantes, fortaleciendo la resiliencia organizacional además de la técnica.

Las prácticas de AIOps no reemplazan la necesidad de métricas bien definidas, ni corrige deficiencias en la instrumentación. La AI a través de sus prácticas permite lidiar con grandes volumenes de datos y obtener información que sistemas tradicionales no serían capaces. Si los datos tienen poca calidad o no están alineados con los objetivos de negocio, los resultados tendrán poca relevancia para el negocio. Por esto, AIOps debe entenderse como un multiplicador que potencia prácticas de observabilidad y DevOps ya establecidas con el foco siempre en las necesidades del negocio. Con un ecosistema

DevOps consolidado y con observabilidad estandarizada, los algoritmos de inteligencia artificial se convierten en un complemento que aporta eficiencia y rapidez.

AIOps aporta valor cuando se integra de forma consciente en un flujo donde las métricas están alineadas al negocio, la observabilidad es robusta y el pipeline DevOps garantiza repetibilidad. Con estas condiciones se convierte en una práctica capaz de mejorar la eficiencia operativa y apoyar la continuidad del servicio. Sin embargo, cualquier debe en estas prácticas puede resultar en un riesgo alto a la organización. Generando una herramienta costosa aislada del negocio y con alta complejidad en su mantenimiento. En sintesis, AIOps es útil, pero su verdadero potencial emerge solo cuando se lo entiende como parte de un todo y no como una solución independiente especifica a operaciones.

7.2. Líneas futuras de investigación

El trabajo realizado abre oportunidades para continuar la investigación en distintas direcciones.

Una primera línea consiste en extender la prueba de concepto hacia una arquitectura de microservicios. Esto permitiría validar la aplicabilidad de las prácticas de AIOps en un entorno distribuido y complejo, donde la coordinación de múltiples servicios incrementa los desafíos de observabilidad y resiliencia.

En segundo lugar, resulta relevante explorar otros módulos y algoritmos de inteligencia artificial para el análisis de señales telemétricas. Además de métodos clásicos como clustering o técnicas basadas en redes neuronales, nuevas posibilidades con modelos generativos de lenguaje (LLMs) aplicados al análisis de logs y correlación de eventos. De esta forma poder ofrecer distintas perspectivas para la detección de anomalías en datos telemétricos heterogéneos.

Otra posible línea de avance es la expansión de la simulación del entorno de producción y la inyección de incidentes. La generación de escenarios más realistas y con mayor contexto, con cargas variables y fallos controlados permitiría disponer de un conjunto de datos más amplio para ser utilizado por las prácticas AIOps.

En último lugar, es necesario estudiar el impacto de las prácticas de AIOps en la performance de los equipos DevOps. Este análisis debería contemplar métricas DORA asociadas a la productividad de más de un integrante y en un contexto industrial real. Su aporte permitiría evaluar no solo los beneficios y el valor de AIOps en las organizaciones, sino su adopción como herramienta.

8. Referencias bibliográficas

[1] «Principios del Manifiesto Ágil». Accedido: 19 de agosto de 2025. [En línea]. Disponible en: https://agilemanifesto.org/iso/es/principles.html

[2] G. Kim, P. Debois, J. Willis, J. Humble, y J. Allspaw, *The DevOps handbook: how to create world-class agility, reliability, & security in technology organizations*, First edition. Portland, OR: IT Revolution Press, LLC, 2016.

[3] V. H. Das Chowdary, A. Shanmukh, T. P. Nikhil, B. S. Kumar, y F. Khan, «DevOps 2.0: Embracing AI/ML, Cloud-Native Development, and a Culture of Continuous Transformation», en *2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN)*, may 2024, pp. 673-679. doi: 10.1109/ICPCSN62568.2024.00112.

[4] T. Visser, «Measuring Agile/DevOps team performance», [En línea]. Disponible en: https://ceur-ws.org/Vol-3414/paper-3.pdf

[5] S. Venkataramanan *et al.*, «AIOps in Action: Streamlining IT Operations Through Artificial Intelligence», [En línea]. Disponible en: https://www.researchgate.net/publication/391850093_AIOps_in_Action_Streamlining_IT_Operations_Through_Artificial_Intelligence_1

[6] L. Bass, P. Clements, y R. Kazman, *Software Architecture in Practice*. Harlow: Pearson Education, 2013.

[7] J. Gawad, «The Future of DevOps: Key Trends, Innovations and Best Practices in 2025», DevOps.com. Accedido: 12 de agosto de 2025. [En línea]. Disponible en: https://devops.com/the-future-of-devops-key-trends-innovations-and-best-practices-in-2025/

[8] «¿Qué es la AIOps? | IBM». Accedido: 7 de mayo de 2025. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/aiops

[9] «Beyond DevOps: The Evolution Toward Intelligent IT Operations with AIOps and MLOps», *RCSAS*, vol. 5, n.º 4, 2025.

[10] «DORA | DORA's software delivery metrics: the four keys». Accedido: 13 de septiembre de 2025. [En línea]. Disponible en: https://dora.dev

[11] F. E. Metioui, «Mastering DevOps Monitoring: Best Practices and Strategies», Medium. Accedido: 23 de agosto de 2025. [En línea]. Disponible en: https://medium.com/@fouadpro2002/mastering-devops-monitoring-best-practices-and-strategies-3af0546043ac

[12] «Best Observability Platforms Reviews 2025 | Gartner Peer Insights». Accedido: 23 de agosto de 2025. [En línea]. Disponible en: https://www.gartner.com/reviews/market/observability-platforms

[13] «¿Qué es la observabilidad? | IBM». Accedido: 11 de agosto de 2025. [En línea]. Disponible en: https://www.ibm.com/mx-es/topics/observability

[14] A. Cunha Azevedo Moreira, «An observability approach for microservices architectures based on OpenTelemetry», 2023, [En línea]. Disponible en: https://repositorium.sdum.uminho.pt/bitstream/1822/92699/1/Andre%20Cunha%20Azevedo%20Moreira.pdf

[15] «Observability primer», OpenTelemetry. Accedido: 20 de junio de 2025. [En línea]. Disponible en: https://opentelemetry.io/docs/concepts/observability-primer/

[16] «¿Qué son los datos de telemetría? | Una guía integral sobre datos de telemetría». Accedido: 12 de mayo de 2025. [En línea]. Disponible en: https://www.elastic.co/es/what-is/telemetry-data
[17] «Logs», OpenTelemetry. Accedido: 22 de junio de 2025. [En línea]. Disponible en: https://opentelemetry.io/docs/concepts/signals/logs/
[18] Q. Fu *et al.*, «Where do developers log? an empirical study on logging practices in industry», en *Companion Proceedings of the 36th International Conference on Software Engineering*, Hyderabad India: ACM, may 2014, pp. 24-33. doi: 10.1145/2591062.2591175.
[19] «Three Pillars of Observability: Logs, Metrics & Traces Defined», Sematext. Accedido: 20 de junio de 2025. [En línea]. Disponible en: https://sematext.com/glossary/three-pillars-of-observability/
[20] R. Kuc, «Logging Levels: What They Are & How to Choose Them», Sematext. Accedido: 24 de agosto de 2025. [En línea]. Disponible en: https://sematext.com/blog/logging-levels/
[21] «Metrics | OpenTelemetry». Accedido: 24 de agosto de 2025. [En línea]. Disponible en: https://opentelemetry.io/docs/concepts/signals/metrics/
[22] S. Gunja, «What are SLOs? How service-level objectives work with SLIs to deliver on SLAs», Dynatrace news. Accedido: 22 de junio de 2025. [En línea]. Disponible en: https://www.dynatrace.com/news/blog/what-are-slos/
[23] A. Villela, «Observability Mythbusters: Logs and Metrics Aren't Enough», Medium. Accedido: 20 de junio de 2025. [En línea]. Disponible en: https://adri-v.medium.com/observability-mythbusters-logs-and-metrics-arent-enough-4ed6f7653cef
[24] Datadog, «Service Level Objectives», Datadog Infrastructure and Application Monitoring. Accedido: 22 de junio de 2025. [En línea]. Disponible en: https://docs.datadoghq.com/es/api/latest/service-level-objectives/
[25] «Traces», OpenTelemetry. Accedido: 24 de agosto de 2025. [En línea]. Disponible en: https://opentelemetry.io/docs/concepts/signals/traces/
[26] Q. Cheng *et al.*, «AI for IT Operations (AIOps) on Cloud Platforms: Reviews, Opportunities and Challenges», 10 de abril de 2023, *arXiv*: arXiv:2304.04661. doi: 10.48550/arXiv.2304.04661.
[27] J. Soldani y A. Brogi, «Anomaly Detection and Failure Root Cause Analysis in (Micro)Service-Based Cloud Applications: A Survey», 26 de mayo de 2021, *arXiv*: arXiv:2105.12378. doi: 10.48550/arXiv.2105.12378.
[28] «Algomox Blog | Real-Time Anomaly Detection with NLP in AIOps». Accedido: 7 de mayo de 2025. [En línea]. Disponible en: https://www.algomox.com/resources/blog/real_time_anomaly_detection_with_nlp_in_aiops/
[29] F. T. Liu, K. M. Ting, y Z.-H. Zhou, «Isolation Forest», en *2008 Eighth IEEE International Conference on Data Mining*, Pisa, Italy: IEEE, dic. 2008, pp. 413-422. doi: 10.1109/ICDM.2008.17.
[30] J. Amat Rodrigo, «Detección de anomalías con Isolation Forest y python», Diciembre 2020, Accedido: 6 de septiembre de 2025. [En línea]. Disponible en: https://cienciadedatos.net/documentos/py22-deteccion-anomalias-isolation-forest-python
[31] D. K. D. Pal, V. R. R. Alluri, S. Thota, V. S. M. Bonam, S. Chitta, y M. Shaik, «AIOps: Integrating AI and Machine Learning into IT Operations», [En línea].

Disponible en: https://www.researchgate.net/publication/389136333_AIOps_Integrating_AI_and_Machine_Learning_into_IT_Operations
[32] Navin Sabharwal , Gaurav Bhardwaj, *Hands-On AIOps: Best Practices for Implementing AIOps in Your Organization*. [En línea]. Disponible en: https://www.oreilly.com/library/view/hands-on-aiops-best/9781484282670/html/524834_1_En_BookBackmatter_OnlinePDF.xhtml
[33] «What is Monolithic Architecture? | IBM». Accedido: 3 de septiembre de 2025. [En línea]. Disponible en: https://www.ibm.com/think/topics/monolithic-architecture
[34] «Documentation», OpenTelemetry. Accedido: 22 de junio de 2025. [En línea]. Disponible en: https://opentelemetry.io/docs/
[35] «OpenTelemetry vs. Datadog: Key Differences Explained», Last9. Accedido: 20 de junio de 2025. [En línea]. Disponible en: https://last9.io/blog/opentelemetry-vs-datadog/
[36] «10 Observability Best Practices Every DevOps Should Implement», Middleware. Accedido: 1 de septiembre de 2025. [En línea]. Disponible en: https://middleware.io/blog/observability/best-practices/
[37] «OpenTelemetry Instrumentation: Manual vs. Automatic (with Examples)», Lumigo. Accedido: 11 de agosto de 2025. [En línea]. Disponible en: https://lumigo.io/opentelemetry/opentelemetry-instrumentation-manual-vs-automatic-with-examples/
[38] «OpenTelemetry best practices: A user's guide to getting started with OpenTelemetry», Grafana Labs. Accedido: 14 de junio de 2025. [En línea]. Disponible en: https://grafana.com/blog/2023/12/18/opentelemetry-best-practices-a-users-guide-to-getting-started-with-opentelemetry/
[39] «The Grafana Stack», Grafana Labs. Accedido: 23 de junio de 2025. [En línea]. Disponible en: https://grafana.com/about/grafana-stack/
[40] N. nur septama, «Instrumenting Microservices with Open Telemetry & Grafana Stack», Medium. Accedido: 20 de junio de 2025. [En línea]. Disponible en: https://medium.com/@nandanurseptamaa/instrumenting-microservices-with-open-telemetry-grafana-stack-fe154942cdff
[41] «What is Prometheus? | Grafana Cloud documentation», Grafana Labs. Accedido: 26 de agosto de 2025. [En línea]. Disponible en: https://grafana.com/docs/grafana-cloud/introduction/what-is-observability/prometheus/
[42] «Prometheus data source | Grafana documentation», Grafana Labs. Accedido: 6 de septiembre de 2025. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/datasources/prometheus/
[43] «Loki overview | Grafana Loki documentation», Grafana Labs. Accedido: 3 de junio de 2025. [En línea]. Disponible en: https://grafana.com/docs/loki/latest/get-started/overview/
[44] «Configure the Loki data source | Grafana documentation», Grafana Labs. Accedido: 6 de septiembre de 2025. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/datasources/loki/
[45] «Grafana Tempo OSS | Distributed tracing backend», Grafana Labs. Accedido: 3 de junio de 2025. [En línea]. Disponible en: https://grafana.com/oss/tempo/
[46] «Introducción a Grafana Traces y Tempo», Grafana Labs. Accedido: 6 de septiembre de 2025. [En línea]. Disponible en: https://grafana.com/es/go/webinar/introduccion-a-grafana-traces-y-tempo/

[47] «About Grafana | Grafana documentation», Grafana Labs. Accedido: 6 de septiembre de 2025. [En línea]. Disponible en: https://grafana.com/docs/grafana/latest/introduction/

[48] F. T. Liu, K. M. Ting, y Z.-H. Zhou, «Isolation-Based Anomaly Detection», *ACM Trans. Knowl. Discov. Data*, vol. 6, n.º 1, pp. 1-39, mar. 2012, doi: 10.1145/2133360.2133363.

[49] C. Yan, «Anomaly Detection Using Isolation Forest: A Comprehensive Guide», Medium. Accedido: 7 de septiembre de 2025. [En línea]. Disponible en: https://chrisyandata.medium.com/anomaly-detection-using-isolation-forest-a-comprehensive-guide-6b73513e854a

[50] N. Nipa, N. Bouguila, y Z. Patterson, «A Comparative Study of Log-Based Anomaly Detection Methods in Real-World System Logs»:, en *Proceedings of the 10th International Conference on Internet of Things, Big Data and Security*, Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2025, pp. 141-152. doi: 10.5220/0013367000003944.

[51] R. Bhandari (MSFT), .«NET 6 will reach End of Support on November 12, 2024», .NET Blog. Accedido: 7 de septiembre de 2025. [En línea]. Disponible en: https://devblogs.microsoft.com/dotnet/dotnet-6-end-of-support/

[52] «¿Qué es el sobreajuste? | IBM». Accedido: 13 de septiembre de 2025. [En línea]. Disponible en: https://www.ibm.com/es-es/think/topics/overfitting

[53] M. Hanna, N. El-Haggar, y M. Sami, «A Review of Scripting Techniques Used in Automated Software Testing», *IJACSA*, vol. 5, n.º 1, 2014, doi: 10.14569/ijacsa.2014.050128.

[54] «¿Qué es la ingeniería del caos? | IBM». Accedido: 2 de agosto de 2025. [En línea]. Disponible en: https://www.ibm.com/es-es/think/topics/chaos-engineering