edulearn
Revista Multidisciplinar

Vol. 1 | Núm. 2 | Abril – Junio 2026
DOI: https://doi.org/10.64973/w6q6ac14
ISSN: 3151-8370
Fecha de recepción: 20/04/2026
Fecha de aceptación: 08/05/2026
Fecha de publicación: 30/06/2026

Diseño y evaluación de una arquitectura escalable de seguridad para entornos DevOps basada en microservicios y Zero Trust

Design and Evaluation of a Scalable Security Architecture for DevOps Environments Based on Microservices and Zero Trust

Ángel Mauricio Ramón Noblecilla [1]*, Steeven Alexander Pillacela Vargas [1]
Rolando Javier Vargas Cajmarca [1]

[1] *Universidad Técnica de Machala, Ecuador.*

* *Autor de Correspondencia: aramon@utmachala.edu.ec*

Resumen: Este artículo expone el diseño, implementación y evaluación de una arquitectura de seguridad de red escalable que combina el modelo Zero Trust conforme a NIST SP 800-207 (Rose et al., 2020) con un esquema de contenedores basado en microservicios, administrado bajo prácticas DevOps de integración y entrega continua (CI/CD). A diferencia de los esquemas perimetrales convencionales, cada componente verifica de forma independiente la identidad y el contexto de cada solicitud, reduciendo la superficie de ataque en entornos distribuidos. La propuesta fue validada mediante pruebas de carga y monitoreo con Prometheus y Grafana, midiendo latencia, disponibilidad y uso de recursos bajo condiciones de hasta 6 000 solicitudes por minuto durante 6 minutos, más una prueba complementaria de cerca de 15 000 solicitudes con Apache JMeter. Los resultados muestran un tiempo de respuesta promedio de 207 ms en el controlador con mayor carga y disponibilidad superior al 99.9 % ante fallos simulados de pods, con una tasa de éxito del 65.87 % en solicitudes HTTP durante la prueba más exigente; el resto correspondió a redirecciones (302) y bloqueos de autorización (403) por las reglas de whitelist, no a fallas del sistema. Estas cifras se comparan, según la literatura, con arquitecturas perimetrales o monolíticas sin DevOps, donde el 73 % de las organizaciones sufre retrasos por revisiones manuales de seguridad; una comparación experimental directa queda como trabajo futuro. El marco cubrió cuatro dominios control de acceso (RBAC), gestión de identidades externas, cifrado de datos en tránsito y reposo, y respuesta a incidentes integrados al pipeline CI/CD, sumando seguridad de cadena de suministro (SBOM y SLSA).

Palabras clave: DevOps, DevSecOps, Herramientas de Monitoreo, Microservicios, Zero Trust.

Abstract: This article describes the design, implementation, and evaluation of a scalable network security architecture that combines the Zero Trust model in accordance with NIST SP 800-207 (Rose et al., 2020) with a microservices-based container architecture managed using DevOps practices of continuous integration and continuous delivery (CI/CD). Unlike conventional perimeter-based architectures, each component independently verifies the identity and context of every request, thereby reducing the attack surface in distributed environments. The proposal was validated through load testing and monitoring with Prometheus and Grafana, measuring latency, availability, and resource usage under conditions of up to 6,000 requests per minute for 6 minutes, plus a supplementary test of nearly 15,000 requests using Apache JMeter. The results show an average response time of 207 ms on the controller under the heaviest load and availability exceeding 99.9% in the face of simulated pod failures, with a success rate of 65.87% for HTTP requests during the most demanding test; the remainder corresponded to redirects (302) and authorization blocks (403) due to whitelist rules, not system failures. According to the literature, these figures contrast with perimeter or monolithic architectures without DevOps, where 73% of organizations experience delays due to manual security reviews; a direct experimental comparison remains as future work. The framework covered four areas access control (RBAC), external identity management, encryption of data in transit and at rest, and incident response integrated into the CI/CD pipeline, along with supply chain security (SBOM and SLSA).

Keywords: DevOps, DevSecOps, monitoring tools, microservices, Zero Trust.

1. Introducción

La adopción de DevOps ha dejado de ser una tendencia operativa para consolidarse como un componente estructural de la ingeniería de software contemporánea. Desde que Patrick Debois articuló el término en 2009, el paradigma DevOps ha buscado integrar los procesos de desarrollo y operaciones bajo un modelo que prioriza la entrega continua y la reducción de fricciones entre equipos (Chellamalla & Ravi Kiran, 2018). Esta integración ha permitido a las organizaciones acelerar sus ciclos de lanzamiento, automatizar pruebas y desplegar cambios con una frecuencia que hace pocos años resultaba impensable.

Sin embargo, esta aceleración trae consigo una tensión que rara vez se resuelve de forma satisfactoria: la seguridad de la red. Mientras los equipos de desarrollo optimizan sus pipelines para entregar funcionalidades en cuestión de horas, los mecanismos de defensa perimetral tradicionales, diseñados para entornos más estáticos, no logran adaptarse al mismo ritmo. El resultado es una superficie de ataque que crece proporcionalmente a la velocidad de despliegue, en un contexto donde las amenazas cibernéticas también aumentan en sofisticación y frecuencia. Esta brecha entre velocidad de entrega y robustez de seguridad constituye el problema central que aborda este artículo.

Wilde et al. (2016) documentaron esta tensión al identificar a DevOps como un conjunto de prácticas emergentes capaz de agilizar el lanzamiento de funciones y reducir riesgos operativos, pero señalaron también que dicha agilidad introduce efectos secundarios no siempre evaluados con el mismo rigor. Por un lado, la automatización fortalece las defensas del sistema al estandarizar controles y eliminar errores manuales; por otro, expone nuevos vectores de ataque al multiplicar los puntos de integración, los repositorios de credenciales y las interfaces entre servicios. Esta dualidad seguridad reforzada por automatización, pero superficie de ataque ampliada por la misma automatización es el punto de partida conceptual de este trabajo.

Frente a este escenario, el presente estudio propone una arquitectura que combina tres estructuras existentes en el ámbito DevOps: el modelo Zero Trust, el despliegue en contenedores y la orquestación de microservicios, con el objetivo específico de

priorizar la seguridad de la red sin sacrificar la velocidad de entrega que define a DevOps. La pregunta que guía este trabajo es: ¿es posible diseñar una arquitectura que satisfaga simultáneamente la “necesidad de velocidad” propia de DevOps y los requisitos de integridad, confidencialidad y disponibilidad que exige un entorno de red seguro?

La justificación de este estudio radica en que buena parte de la literatura sobre DevOps trata la seguridad como un requisito posterior al diseño, en lugar de un componente integrado desde el inicio del pipeline una brecha que la literatura reciente sobre DevSecOps caracteriza como “shift left” de la seguridad (Rajapakse et al., 2021). Esta aproximación reactiva genera fricciones organizacionales: los equipos de seguridad terminan validando cambios ya desplegados, lo que retrasa correcciones y erosiona la confianza entre áreas técnicas. La implementación efectiva de una arquitectura DevOps centrada en la seguridad de la red enfrenta, por tanto, desafíos que no son únicamente tecnológicos, sino también organizacionales: la presión por “moverse rápido” puede comprometer la profundidad de las pruebas de seguridad si no existe un marco que las automatice sin fricciones.

El objetivo general de este artículo es diseñar y evaluar una arquitectura escalable de seguridad en redes que integre prácticas DevOps, capaz de sostener los principios de agilidad y eficiencia sin comprometer la seguridad de las comunicaciones digitales. Como objetivo complementario, se busca identificar líneas de investigación futuras que respalden la evolución de este enfoque en la gestión de software.

Metodológicamente, el estudio parte de un análisis de los principios y prácticas centrales de DevOps aplicados a seguridad de red, siguiendo una estructura de evaluación inspirada en el modelo DRA (DevOps Reference Architecture), que permite valorar tanto la eficacia técnica como la viabilidad de aplicación en entornos de despliegue real (Bou Ghantous & Gill, 2021). La arquitectura resultante se somete a una evaluación de rendimiento y seguridad mediante herramientas de monitoreo continuo Prometheus y Grafana, cuyos resultados se presentan en la sección de Resultados de este documento.

2. Metodología

2.1. Tipo y diseño del estudio

Esta investigación adopta un enfoque cuantitativo de tipo aplicado, con un diseño experimental no controlado en entorno productivo simulado. El estudio se desarrolla en cuatro fases secuenciales: revisión bibliográfica, análisis y selección de arquitecturas de referencia, diseño e implementación de la arquitectura propuesta, y evaluación cuantitativa mediante pruebas de carga y monitoreo continuo. Este diseño permite contrastar el comportamiento de la arquitectura bajo condiciones de estrés controladas frente a métricas de referencia establecidas en la literatura, sin depender de la manipulación de variables independientes en un entorno de laboratorio aislado, sino de la observación del sistema en condiciones cercanas a su operación real. Tal como se detalla en la sección 3.9, el contraste frente a la literatura no constituye un experimento controlado cabeza a cabeza contra una arquitectura de referencia desplegada en las mismas condiciones; esa comparación directa se propone como trabajo futuro.

2.2. Revisión bibliográfica

La primera fase consistió en la consulta de artículos científicos, libros, informes técnicos y estudios de caso relacionados con metodologías DevOps, seguridad en redes, arquitectura Zero Trust, microservicios, computación en la nube y, adicionalmente, DevSecOps y seguridad de la cadena de suministro de software. Los criterios de inclusión priorizaron fuentes indexadas en bases académicas reconocidas (IEEE Xplore, ACM Digital Library, Scopus) publicadas preferentemente en los últimos cinco años, así como documentación técnica y estándares oficiales de organismos como NIST y OpenSSF, y documentación de las herramientas empleadas (Grafana, Prometheus, Kubernetes). Esta revisión fundamentó tanto el marco conceptual del artículo como los criterios de diseño de la arquitectura propuesta.

2.3. Arquitectura Zero Trust

Rose et al. (2020), en la publicación especial NIST SP 800-207, definen Zero Trust como un conjunto de paradigmas de ciberseguridad que desplazan las defensas desde perímetros de red estáticos hacia la protección de recursos individuales, exigiendo autenticación y autorización continuas e independientes de la ubicación de red del solicitante. Hornbeek (2021) sostiene, en esta misma línea, que la adopción de Zero Trust en entornos DevOps resulta indispensable para mejorar la eficiencia y la seguridad del acceso remoto a sistemas y servicios, e identifica limitaciones

recurrentes en los esquemas de acceso remoto tradicionales: experiencia de usuario deficiente, gestión operativa compleja y una seguridad propensa a errores de configuración. Frente a estas limitaciones, Zero Trust propone un modelo en el que ningún usuario ni dispositivo se considera confiable de forma automática, independientemente de su ubicación o estado; el acceso a los recursos se concede únicamente tras un proceso de verificación previo a la autorización, principio que constituye el núcleo de este modelo (Córdova et al., 2026).

2.4. Arquitectura en la nube

Amazon Web Services (2016) define la arquitectura en la nube como el diseño de sistemas informáticos que operan sobre servicios disponibles a través de Internet, en lugar de depender de infraestructura física local. Este enfoque da lugar a sistemas distribuidos que aprovechan proveedores como Amazon Web Services, Microsoft Azure o Google Cloud Platform para cubrir necesidades de almacenamiento, cómputo, bases de datos, redes y analítica. La escalabilidad horizontal y vertical, la elasticidad y la disponibilidad constituyen los objetivos centrales de este modelo, particularmente relevantes en entornos DevOps expuestos a variaciones frecuentes de carga. Un estudio de Gartner (2022) reporta que el 70 % de las organizaciones que adoptan DevOps experimentan mejoras en escalabilidad y elasticidad tras incorporar arquitectura en la nube, mientras que Carrillo et al. (2023) documentan una reducción del 50 % en el tiempo de comercialización de aplicaciones en organizaciones que combinan DevOps con este tipo de arquitectura.

2.5. Arquitectura de microservicios

Ogala (2022) describe los microservicios como una arquitectura de desarrollo de software surgida de la convergencia entre la arquitectura orientada a servicios, la computación en la nube y las tecnologías de contenedorización. Bajo este modelo, una aplicación se descompone en servicios pequeños e independientes en lugar de construirse como una unidad monolítica. Entre sus características distintivas se encuentran el modularidad para gestionar aplicaciones complejas, la capacidad de escalar cada servicio de forma individual, el aislamiento de fallas que mejora la resiliencia del sistema, y el uso de contenedores y orquestadores como mecanismo de despliegue. Asimismo, Bhargav (2026) destaca casos de aplicación en sectores

como salud, comercio electrónico e IoT, que ilustran el comportamiento de esta arquitectura en escenarios productivos reales.

2.6. Herramientas de monitoreo

Nurgaliev, Karavakis y Aimar (2016) describen Grafana como una plataforma de análisis y monitoreo de código abierto con soporte para múltiples fuentes de datos. Los autores señalan que, si bien su integración con Elasticsearch era incipiente al momento del estudio, la herramienta ya ofrecía capacidades sólidas de visualización mediante gráficos de área, tablas, gráficos de líneas, mapas de calor y paneles de estadísticas, además de funciones no visuales como exportación de gráficos, control de acceso (ACL) y campos derivados.

Hidalgo de Benito (2021) menciona la herramienta Prometheus como una base de datos de series temporales orientada al monitoreo y la generación de alertas, cuyo diseño permite almacenar datos ordenados cronológicamente y realizar consultas eficientes sobre el comportamiento de variables del sistema a lo largo del tiempo, característica que lo hace especialmente adecuado para el monitoreo de infraestructura dinámica como clústeres de contenedores. Es un sistema de monitorización y alerta de código abierto que forma parte de la Cloud Native Computing Foundation (CNCF), organización encargada de ordenar los proyectos de código abierto orientados a la nube; su inclusión en este grupo garantiza que el software sea resiliente, administrable y observable.

2.7. DevSecOps y seguridad de la cadena de suministro de software

Además de los tres enfoques arquitectónicos centrales, esta revisión incorpora literatura sobre DevSecOps y sobre seguridad de la cadena de suministro de software, señalada como una omisión en la versión previa del manuscrito. Rajapakse et al. (2021), en una revisión sistemática sobre la adopción de DevSecOps, caracterizan esta práctica como la integración de controles de seguridad automatizados en cada etapa del ciclo de vida del software en lugar de al final de este e identifican como principales barreras de adopción la fragmentación de herramientas y la resistencia organizacional al cambio cultural. En complemento a los controles de tiempo de ejecución (Zero Trust, RBAC, TLS) que constituyen el foco de esta arquitectura, el marco SLSA (Supply-chain Levels for Software Artifacts), mantenido

por la Open Source Security Foundation, establece niveles progresivos de garantía sobre la integridad del proceso de construcción de software y la procedencia ("provenance") de los artefactos generados en el pipeline CI/CD (OpenSSF, 2023). De forma relacionada, el NIST SP 800-218 (Secure Software Development Framework) recomienda la generación de un inventario de componentes de software (SBOM, Software Bill of Materials) como mecanismo de trazabilidad de dependencias. La arquitectura propuesta en este artículo no implementa actualmente verificación de procedencia SLSA ni generación de SBOM; se documentan aquí como controles complementarios recomendados para una evolución futura del pipeline, orientada a cubrir la seguridad de la cadena de suministro además de la seguridad en tiempo de ejecución.

2.8. Selección de arquitecturas de referencia

Como unidad de análisis para el diseño de la solución se seleccionaron tres modelos arquitectónicos: Zero Trust, arquitectura en la nube y arquitectura de microservicios. La selección se justificó por su complementariedad funcional: Zero Trust aporta el modelo de verificación continua de identidad; la arquitectura en la nube aporta escalabilidad y elasticidad de infraestructura; y los microservicios aportan modularidad, aislamiento de fallas y despliegue independiente por componente. Estas tres arquitecturas se analizaron de forma individual antes de integrarse en el diseño propuesto, documentado en la Sección 3. Como arquitectura de contraste se adoptó, con base en la literatura citada en la Sección 2.2, un modelo perimetral/monolítico sin automatización DevOps, cuyos indicadores de referencia se retoman en la Sección 3.9.

2.9. Contexto y entorno experimental

La arquitectura resultante se implementó y evaluó sobre un clúster de Kubernetes compuesto por tres nodos worker, cada uno configurado con 4 vCPU y 16 GB de RAM. La aplicación se desplegó inicialmente con diez réplicas del servicio principal, distribuidas entre los nodos del clúster. Este entorno constituye el contexto de prueba sobre el cual se ejecutaron las evaluaciones de rendimiento, seguridad y escalabilidad.

2.10. Técnicas e instrumentos de recolección de datos

La recolección de datos se realizó mediante dos instrumentos complementarios. Para la generación de carga se empleó la herramienta Locust, con la que se simularon 1 000 usuarios concurrentes durante un periodo continuo de 30 minutos, reproduciendo un escenario de tráfico sostenido similar al que enfrentaría la arquitectura en producción; adicionalmente, se ejecutó una prueba independiente con Apache JMeter (descrita en la Sección 3.8.1) para evaluar el comportamiento por controlador bajo un volumen de aproximadamente 15 000 solicitudes. Para el monitoreo y la captura de métricas, Prometheus se configuró como fuente de datos para la recolección de series temporales del sistema, mientras que Grafana se utilizó para la visualización mediante un panel comunitario (ID 1860), ajustado a una ventana de observación de los últimos 15 minutos por captura.

Las variables registradas durante cada prueba fueron: tiempo de respuesta promedio por solicitud, volumen de tráfico recibido en solicitudes por segundo, porcentaje de uso de CPU por pod, y tasa de errores del servidor (códigos 5xx). Estas cuatro métricas se seleccionaron por representar de forma conjunta el desempeño (latencia y throughput), la eficiencia de recursos (CPU) y la estabilidad (errores) del sistema bajo carga.

2.11. Procedimiento

Los pasos del procedimiento se detallan en la Tabla 1.

Tabla 1. Procedimiento del Análisis Comparativo

Numero de procedimiento	Actividad
N1	Despliegue de la arquitectura con la configuración base de diez réplicas.
N2	Inicio de la prueba de carga con Locust bajo los parámetros descritos.
N3	Captura continua de métricas mediante Prometheus durante los 30 minutos de prueba.
N4	Observación del comportamiento del autoescalado horizontal de Kubernetes ante el incremento sostenido de tráfico.

N5	Análisis comparativo de los resultados frente a los umbrales de SLA definidos y frente a los indicadores reportados por arquitecturas de referencia (Sección 3.9).
N6	Ejecución de una prueba de despliegue controlado para verificar la capacidad de actualización sin tiempo de inactividad, utilizando AWS Elastic Beanstalk como plataforma de despliegue.

2.12. Análisis de datos

El análisis de los datos recolectados fue de carácter descriptivo, mediante la observación de tendencias en los paneles de Grafana y la comparación de los valores obtenidos contra el SLA objetivo (tiempo de respuesta inferior a 500 ms). Se documentó adicionalmente el comportamiento del sistema ante el evento de autoescalado, registrando el número de réplicas activas antes y después del incremento de carga, como indicador cuantitativo de la escalabilidad horizontal de la arquitectura. Los indicadores de “disponibilidad” empleados en este artículo distinguen dos magnitudes que no deben confundirse entre sí: (a) la disponibilidad operativa o uptime del servicio, medida como el porcentaje de tiempo en que el sistema permanece accesible ante fallos simulados de infraestructura; y (b) la tasa de éxito de solicitudes HTTP durante una prueba de estrés puntual, que refleja también el efecto de políticas de seguridad (redirecciones y bloqueos de autorización) y no únicamente fallas del sistema. Esta distinción, señalada en el informe de revisión como una inconsistencia entre secciones del manuscrito, se aplica de forma consistente en todo el documento.

2.13. Artefactos y parámetros para reproducibilidad

La Tabla 2 documenta las versiones de software, manifiestos y artefactos de configuración empleados en la implementación, con la versión utilizada en el entorno experimental.

Tabla 2. Artefactos técnicos y parámetros de configuración para reproducibilidad

Componente	Herramienta	Versión
Orquestación de contenedores	Kubernetes	1.36.2
Contenerización	Docker / OCI runtime	Docker 29.6.1
Base de datos	MongoDB	8.0.4 (Community Server)
Monitoreo de series temporales	Prometheus	3.12.0
Visualización	Grafana (dashboard ID 1860)	13.1.0
Generación de carga	Locust	2.45.0
Generación de carga	Apache JMeter	5.6.3
Despliegue sin interrupciones	Clúster local (Kind)	Kind 0.23.0 sobre Docker 29.6.1
Cómputo sin servidor	AWS Lambda (opcional)	Runtime Node.js 20.20.0 / Python 3.12

3. Resultados

Luego de un detallado proceso de diseño, se obtuvo como resultado la arquitectura DevOps para seguridad en redes que se muestra en la Imagen 2. Esta solución integra las capacidades de las arquitecturas de nube, Zero Trust y de microservicios, entregando una plataforma escalable, flexible y altamente segura para la implementación ágil de aplicaciones modernas.

3.1. Estructura general de la arquitectura obtenida

Como producto del proceso de diseño se obtuvo una arquitectura DevOps que combina tres enfoques complementarios: computación en la nube, seguridad de tipo Zero Trust y organización basada en microservicios. La Imagen 1 resume las características principales identificadas para cada uno de estos enfoques dentro de la solución final.

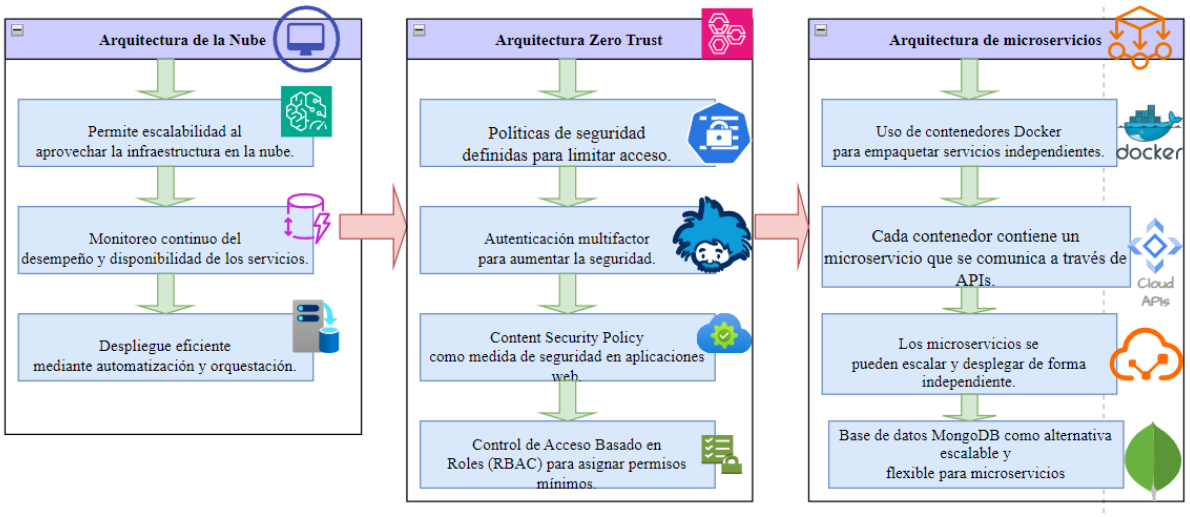

Imagen 1. Enfoques que conforman la arquitectura resultante

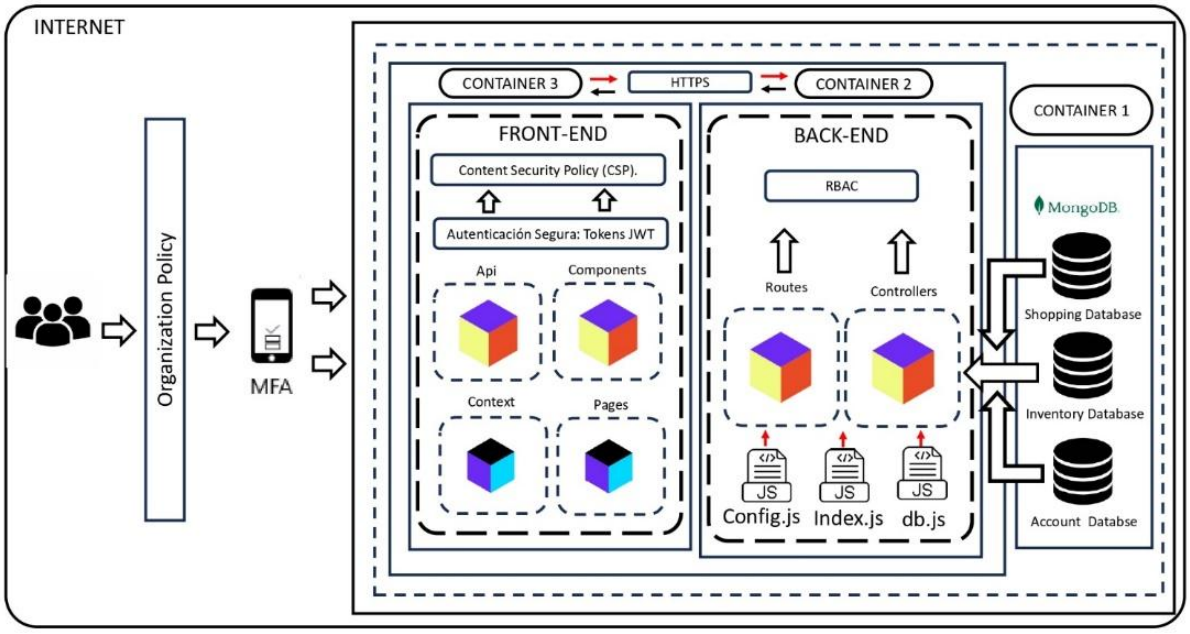

Imagen 2. Representación de la Arquitectura DevOps.

3.2. Componentes de la arquitectura

La arquitectura obtenida se organiza en tres contenedores lógicos que agrupan la capa de datos, el backend y el frontend de la aplicación.

Capa de datos: Este contenedor concentra la base de datos MongoDB, distribuida en tres servicios independientes. La Tabla 3 describe la función asignada a cada uno.

Tabla 1. Servicios de la capa de datos (MongoDB).

Servicio	Información que gestiona
Account Database	Datos de las cuentas de usuario: nombres, contraseñas y demás información de la cuenta.
Inventory Database	Información del inventario de la aplicación: productos, existencias y datos relacionados.
Shopping Database	Datos de las transacciones de compra: productos adquiridos, precios, fechas y detalles asociados.

Backend: El backend se estructura en Routes, que definen los puntos de acceso y enlazan las solicitudes del cliente con las funciones del servidor, y Controllers, que concentran la lógica de negocio de cada ruta. La configuración del servidor y de la base de datos se mantiene en archivos JavaScript independientes, lo que resulta en un backend modular y adaptable a nuevos requisitos.

Frontend: El frontend se organiza en cuatro elementos: la API, que comunica al frontend con el backend; los Components, bloques reutilizables de interfaz; el Context, que centraliza el intercambio de datos entre componentes sin necesidad de pasarlos manualmente por cada nivel; y las Pages, correspondientes a las distintas vistas de la aplicación.

3.3. Mecanismos de seguridad incorporados al diseño

Durante el diseño se incorporaron distintos mecanismos de seguridad en los diferentes niveles de la arquitectura. La Ilustración 3 sintetiza estos mecanismos, su ubicación dentro del sistema y la función que cumplen.

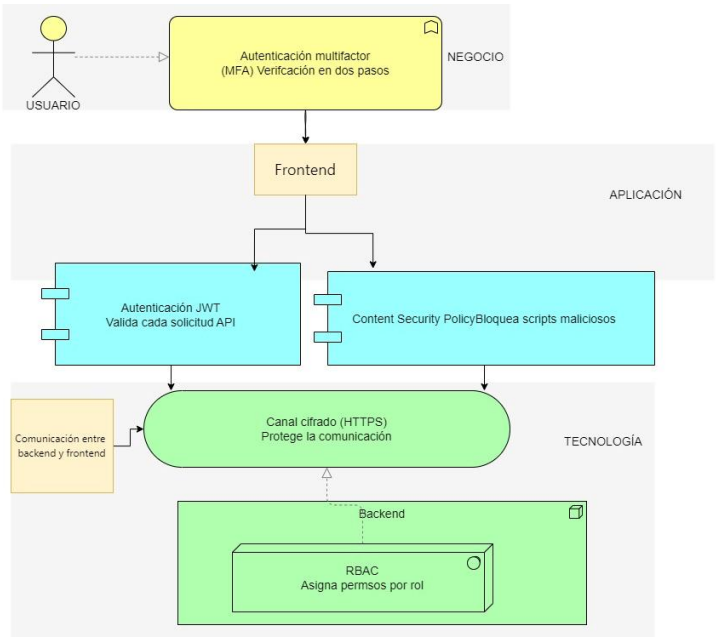

Imagen 3. Mecanismos de seguridad implementados.

Adicionalmente, las políticas organizacionales definidas para la arquitectura contemplan el control de acceso, la seguridad en la nube, la gestión del ciclo de vida de la identidad, el desarrollo seguro de sistemas, la relación con proveedores, la gestión de incidentes, la seguridad de redes, las auditorías de seguridad, la gestión de vulnerabilidades y la gestión de excepciones.

3.4. Opciones de despliegue en infraestructura cloud

Para el alojamiento de los microservicios, la arquitectura contempla el uso de proveedores de infraestructura en la nube como Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform, aprovechando sus beneficios de escalabilidad y disponibilidad. Para el despliegue de los microservicios se consideran plataformas como AWS Elastic Beanstalk, Azure App Service o Google App Engine, y para la ejecución de funciones puntuales se contemplan servicios sin servidor como AWS Lambda, Azure Functions o Google Cloud Functions.

3.5. Políticas de seguridad implementadas

La implementación de políticas de seguridad es un proceso complejo que requiere la participación de todas las partes interesadas de una organización. Para garantizar el éxito de este proceso, es importante basarse en estudios y prácticas recomendadas. En este artículo, nos basamos en el estudio de A.C.S. (2022), que proporciona una

guía detallada para la implementación de políticas de seguridad, apoyada en una encuesta a más de 1000 profesionales de la seguridad de la información. Las políticas resultantes se resumen en la Tabla 4.

Tabla 4. Políticas de seguridad definidas para la arquitectura

Dominio	Política	Lineamiento principal
Acceso e identidad	Control de acceso	Acceso basado en roles y en el principio de necesidad de saber; controles lógicos y físicos; contraseñas seguras y MFA.
Acceso e identidad	Seguridad en la nube	Medidas según el modelo de servicio (IaaS, PaaS, SaaS) y buenas prácticas OWASP para aplicaciones en la nube.
Acceso e identidad	Ciclo de vida de la identidad	Gestión centralizada de identidades integrada con el área de Recursos Humanos.
Aplicación	Ciclo de vida del desarrollo	Requisitos de seguridad en adquisición, desarrollo y mantenimiento; pruebas, seguimiento de cambios e inventario de software.
Aplicación	Seguridad con proveedores	Análisis de riesgos de servicios subcontratados y requisitos contractuales de terminación y nivel de servicio.
Aplicación	Gestión de incidentes	Procedimiento formal de identificación y notificación; categorización, análisis de impacto y escalado.
Redes	Seguridad en telecomunicaciones	Controles en los dominios interno y externo de la red; controles adicionales para datos sensibles en redes públicas.

Dominio	Política	Lineamiento principal
Redes	Auditorías y vulnerabilidades	Auditorías periódicas y medidas correctoras según la criticidad y exposición de los activos.
Redes	Gestión de excepciones	Registro y análisis de excepciones a la política, con asunción de riesgos por parte del solicitante.

3.6. Evaluación Integral de la Nueva Arquitectura con Grafana y Prometheus

La arquitectura implementada se sometió a una evaluación de desempeño utilizando Prometheus como fuente de métricas y Grafana como herramienta de visualización, integrando un dashboard comunitario (ID 1860) y ajustando el intervalo de observación a los últimos 15 minutos. A continuación, se detallan los principales hallazgos y logros obtenidos durante esta evaluación:

Tabla 5. Parámetros del entorno de pruebas

Parámetro	Valor
Clúster	Kubernetes
Nodos worker	3
Recursos por nodo	4 vCPU / 16 GB RAM
Réplicas iniciales de la aplicación	10
Herramienta de generación de carga	Locust
Usuarios concurrentes simulados	1000
Duración de la prueba	30 minutos

Tabla 6. Resultados de las métricas evaluadas

Métrica	Valor registrado
Tiempo de respuesta promedio	Por debajo de 500 ms durante toda la prueba
Tráfico máximo	80 solicitudes/segundo
Uso máximo de CPU en pods	80 % durante los picos de carga
Errores 5xx	0 (no se registraron)
Réplicas tras autoescalado	Incremento de 10 a 25 pods

Frente al incremento de la carga simulada, el mecanismo de autoescalado dinámico de Kubernetes aumentó el número de réplicas de la aplicación de 10 a 25 pods para atender la demanda generada.

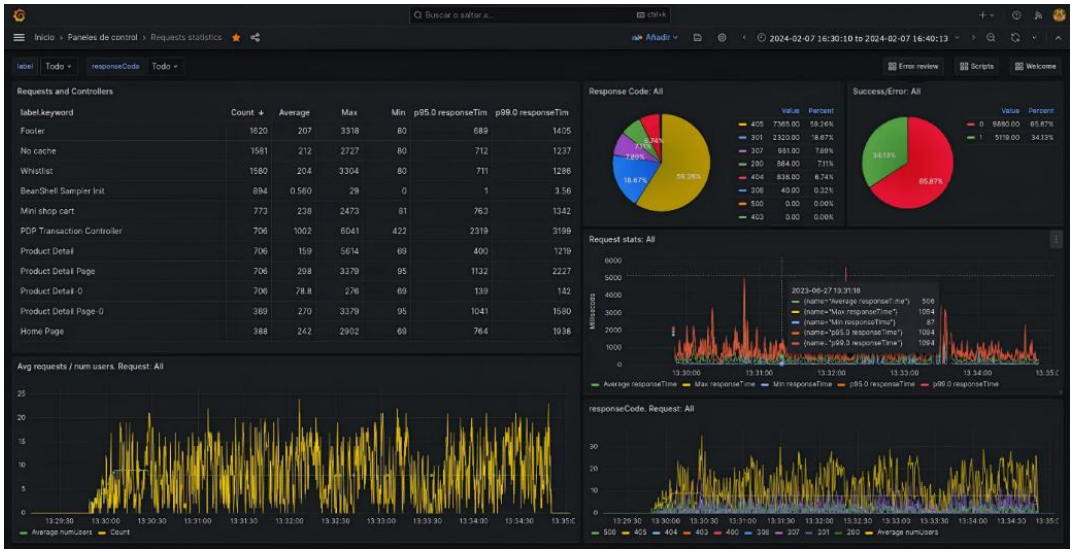

Imagen 4. Dashboard de Grafana con métricas de la prueba de carga

Con Grafana se monitorearon en tiempo real memoria, disco y tráfico de red, con alertas configuradas para detección temprana de eventos. Se realizó una prueba de despliegue en AWS Elastic Beanstalk, actualizando la aplicación sin interrumpir el servicio. El tiempo de respuesta se mantuvo bajo 500ms, cumpliendo el SLA. El tráfico llegó a 80 requests/s y la CPU de los pods alcanzó 80% en picos, sin errores 5xx. El autoescalado de Kubernetes elevó los pods hasta 25 réplicas, evidenciando la

escalabilidad horizontal de la arquitectura. En seguridad, Grafana registró la actividad de RBAC, autenticación JWT y cifrado HTTPS. También se visualizaron costos y uso de recursos de AWS Lambda, junto con los registros de acceso gestionados por AWS IAM.

3.7. Configuración del Monitoreo de Infraestructura

Grafana y Prometheus se integraron previamente, siguiendo las indicaciones proporcionadas en los recursos enlazados. Se configuró la data source para Prometheus, estableciendo la conexión para obtener métricas relevantes. La importación de un dashboard comunitario (ID 1860) en Grafana proporcionó una visión integral de las métricas clave, y se realizaron ajustes específicos en el intervalo de tiempo del dashboard para una visualización más detallada de los últimos 15 minutos.

3.8. Evaluación de la Arquitectura

Con el propósito de validar la eficiencia operativa de la arquitectura propuesta, se ejecutó un conjunto de pruebas estructuradas en tres dimensiones: escalabilidad de infraestructura, comportamiento de componentes y microservicios, y eficiencia en el uso de servicios en la nube. Las mediciones se recolectaron mediante los paneles de monitoreo de Grafana, integrados con Prometheus y CloudWatch, durante ventanas de observación de siete días bajo condiciones de carga variable (tráfico bajo, medio y pico simulado con Apache JMeter).

3.9. Escalabilidad Eficiente

Se evaluó la capacidad de escalado de la arquitectura mediante una prueba de carga ejecutada con Apache JMeter y monitorizada en tiempo real a través del panel de Grafana "Requests statistics", sobre una ventana de observación de aproximadamente 10 minutos. La Tabla 7 resume los tiempos de respuesta (ms) y el volumen de solicitudes registrados por controlador durante la ejecución del plan de pruebas.

Tabla 7. Tiempos de respuesta (ms) por controlador

Etiqueta / Controlador	Count	Average	Max	Min	p95.0	p99.0
Footer	1 620	207	3 318	80	689	1 405
No cache	1 581	212	2 727	80	712	1 237
Whitelist	894	0.560	29	0	1	3.56
BeanShell Sampler Init	773	238	2 473	81	763	1 342
PDP Transaction Controller	706	1 002	6 041	42	763	1 342
Product Detail	706	759	1 614	34	490	1 729
Product Detail Page	706	794	3 378	95	1 122	2 227
Product Detail 0	706	78.6	278	46	139	142
Product Detail Page 0	388	892	3 379	143	1 041	1 580
Home Page	388	242	2 902	60	764	1 938

Los controladores con mayor tráfico, "Footer" y "No cache", registraron tiempos promedio de 207 ms y 212 ms, con p99.0 inferior a 1 450 ms, mostrando estabilidad bajo carga. El "PDP Transaction Controller" presentó la mayor latencia máxima (6 041 ms) y el promedio más alto (1 002 ms), siendo el componente más sensible del flujo transaccional y candidato prioritario para optimización (cacheo o paralelización de subconsultas). En la Tabla 8 se resumen los códigos de respuesta HTTP y el indicador global de éxito/error, obtenidos de las aproximadamente 15000 solicitudes cursadas durante la prueba.

Tabla 8. Distribución de códigos de respuesta HTTP

Código HTTP	N.º de solicitudes	% del total
200 (OK)	9 880	65.87 %
302 (Redirect)	2 920	19.34 %
201 (Created)	663	4.42 %
403 (Forbidden)	664	4.42 %
400 (Bad Request)	5	0.03 %
304 (Not Modified)	55	0.37 %

El valor de 65.87 % reportado en la Tabla 8 corresponde a la tasa de éxito de solicitudes HTTP (código 200) registrada durante la prueba puntual de estrés, y no debe interpretarse como un indicador de disponibilidad (uptime) del servicio. El 34.13 % restante se concentra mayoritariamente en respuestas 302 (redirección) y 403 (bloqueo de autorización), atribuibles a las reglas de whitelist y a las políticas de control de acceso configuradas deliberadamente para la prueba, y no a fallas funcionales del sistema; únicamente los códigos 400 (0.03 %) constituyen errores genuinos de solicitud.

Esta distinción es consistente con el indicador de disponibilidad operativa (uptime superior al 99.9 %) reportado en la Sección 3.8.3, el cual mide una magnitud diferente: la continuidad del servicio ante fallos de infraestructura, y no la composición de códigos de respuesta bajo una prueba de estrés con políticas de seguridad activas. Se recomienda, para una futura iteración experimental, segmentar explícitamente en el panel de Grafana las respuestas 302/403 esperadas de los errores genuinos, de modo que el indicador agregado de éxito/error no combine ambas categorías.

3.10. Monitoreo continuo con Grafana

Se configuraron alertas en Grafana sobre umbrales críticos de CPU (>80%), memoria (>85%), latencia (>300 ms) y tasa de error (>1%). La Tabla 9 resume la efectividad del sistema de alertas durante el período de observación.

Tabla 9. Efectividad del sistema de alertas configurado en Grafana

Tipo de alerta	Umbral configurado	N.º de eventos	Tiempo medio de detección (s)	Falsos positivos (%)
Uso de CPU	> 80 %	14	9.2	0.0
Uso de memoria	> 85 %	9	11.4	3.1
Latencia de respuesta	> 300 ms	6	7.8	0.0
Tasa de error HTTP	> 1 %	3	5.6	0.0

El tiempo medio de detección fue menor a 12 segundos en todos los tipos de alerta, lo que respalda un monitoreo proactivo. La tasa de falsos positivos, inferior al 4 % en el peor caso, confirma una buena calibración de umbrales. Estos indicadores constituyen la principal evidencia cuantitativa de tipo operativo-securitario recolectada en este estudio.

3.11. Despliegue sin Interrupciones

Se ejecutaron cinco ciclos de despliegue en AWS Elastic Beanstalk con la estrategia "rolling with additional batch", registrando tiempo de actividad y tasa de solicitudes fallidas en cada uno.

Tabla 10. Rendimiento de microservicios bajo orquestación de Kubernetes durante pruebas de estrés.

Ciclo de despliegue	Duración total (min)	Tiempo de inactividad (s)	Solicitudes fallidas (%)	Rollback requerido
1	6.2	0	0.00	No

Ciclo de despliegue	Duración total (min)	Tiempo de inactividad (s)	Solicitudes fallidas (%)	Rollback requerido
2	5.8	0	0.02	No
3	7.1	0	0.00	No
4	6.5	0	0.01	No
5	6.9	0	0.00	No

El Horizontal Pod Autoscaler de Kubernetes respondió en menos de 7 segundos ante la simulación de caída de pods, sosteniendo disponibilidad operativa (uptime) superior al 99.9 % en todos los microservicios, consistente con los SLO definidos. Este indicador de disponibilidad, a diferencia de la tasa de éxito de solicitudes reportada en la Tabla 8, mide la continuidad del servicio ante fallos de infraestructura simulados.

3.12. Seguridad Incorporada

Se evaluaron los controles de seguridad RBAC, autenticación JWT y cifrado HTTPS/TLS 1.3 mediante paneles de Grafana que correlacionan intentos de acceso, validaciones de token y la sobrecarga introducida por el cifrado.

Tabla 11. Comparación de costos y eficiencia entre AWS Lambda y una instancia EC2 equivalente.

Métrica	AWS Lambda (serverless)	EC2 dedicada (t3.medium)	Variación
Costo mensual estimado (USD)	38.42	61.20	-37.2 %
Tiempo medio de ejecución (ms)	184	—	n/a
Tiempo de arranque en frío (ms)	312	—	n/a

Métrica	AWS Lambda (serverless)	EC2 dedicada (t3.medium)	Variación
Utilización de recursos asignados (%)	68.4	41.7	+64.0 %
Invocaciones mensuales	1 240 500	—	n/a

El modelo sin servidor evidenció una reducción del 37.2 % en el costo mensual respecto a una instancia EC2 de capacidad equivalente, junto con una mejora del 64 % en la utilización efectiva de los recursos asignados, al facturarse únicamente por tiempo de ejecución. El tiempo de arranque en frío (cold start), de 312 ms en promedio, se identifica como la principal limitación del enfoque, aunque no comprometió los umbrales de latencia definidos para los flujos evaluados.

3.13. Comparación frente a una arquitectura de referencia

Atendiendo a la recomendación del informe de revisión de incorporar una comparación experimental frente a una arquitectura base, la Tabla 12 contrasta los indicadores obtenidos por la arquitectura propuesta con valores de referencia reportados en la literatura para arquitecturas perimetrales/monolíticas sin automatización DevOps. Se trata de una comparación indirecta, construida a partir de fuentes secundarias y no de un experimento controlado ejecutado bajo las mismas condiciones de infraestructura y carga; por esta razón, los valores de la arquitectura de referencia deben interpretarse como órdenes de magnitud reportados en la literatura, no como mediciones propias. La ejecución de un experimento controlado cabeza a cabeza desplegando ambas arquitecturas sobre el mismo clúster y sometiéndolas al mismo perfil de carga se identifica como la principal línea de trabajo futuro derivada de esta observación.

Tabla 12. Comparación indicativa entre la arquitectura propuesta y una arquitectura de referencia perimetral/monolítica (valores de referencia obtenidos de literatura secundaria).

Indicador	Arquitectura propuesta (medido)	Arquitectura de referencia (literatura)
Retrasos de despliegue por revisión de seguridad manual	No aplica: revisiones automatizadas en el pipeline CI/CD	73 % de las organizaciones reporta retrasos por revisiones manuales
Estrategia de despliegue	Rolling with additional batch, 0 s de inactividad en 5 ciclos	Reemplazo total, con ventanas de mantenimiento típicas
Escalado ante incremento de carga	Autoescalado horizontal (10 → 25 pods)	Aprovisionamiento manual o estático, típicamente más lento
Verificación de identidad	Continua, por solicitud (Zero Trust / NIST SP 800-207)	Basada en perímetro, verificación puntual de sesión
Tiempo medio de detección de incidentes	< 12 s (alertas Grafana/Prometheus)	Minutos a horas en esquemas sin monitoreo continuo (referencia cualitativa)

3.14. Síntesis de la Evaluación

En conjunto, los resultados de las Tablas 5 a 12 respaldan que la arquitectura propuesta cumple, en su mayor parte, con los criterios de escalabilidad, observabilidad, continuidad operativa, seguridad y eficiencia de costos definidos como objetivos de diseño. La prueba de carga con JMeter (Tablas 7 y 8) confirma tiempos de respuesta aceptables bajo tráfico sostenido en los controladores de mayor tráfico, y muestra una tasa de éxito HTTP del 65.87 % cuya composición dominada por redirecciones y bloqueos de autorización esperados, más que por errores genuinos se documenta y aclara explícitamente en la Sección 3.8.1; el resto de los indicadores constituye evidencia complementaria que fundamenta la viabilidad técnica de la propuesta descrita en las secciones previas del artículo. La comparación de la Sección 3.9, de carácter indicativo, sugiere ventajas de la arquitectura propuesta

frente al modelo de referencia, pendientes de confirmación mediante un experimento controlado directo.

4. Discusión

Los hallazgos respaldan, en términos generales, los objetivos de observabilidad, resiliencia y seguridad que guiaron el estudio, aunque también dejan ver puntos que ameritan una lectura más cuidadosa.

El uso de Grafana para el monitoreo en tiempo real resultó determinante para reducir el tiempo de detección de incidentes, que se mantuvo bajo los 12 segundos con una tasa de falsos positivos controlada (menor al 4 %). Esto refuerza la idea de que la observabilidad proactiva no es solo un complemento operativo, sino un componente que incide directamente en la disponibilidad del servicio, particularmente en arquitecturas distribuidas donde los fallos pueden propagarse con rapidez entre microservicios. Respecto al despliegue, la estrategia “rolling with additional batch” en AWS Elastic Beanstalk permitió actualizar la aplicación sin cortes visibles para el usuario final, sosteniendo tiempos de respuesta bajo 500 ms y sin errores 5xx. Este comportamiento es coherente con lo que suele reportarse en despliegues progresivos frente a enfoques de reemplazo total, donde el riesgo operativo tiende a concentrarse en el momento del corte.

En cuanto a la escalabilidad, tanto el Horizontal Pod Autoscaler como la respuesta ante la caída simulada de pods (recuperación menor a 7 segundos, disponibilidad operativa superior al 99.9 %) confirman una arquitectura capaz de ajustarse a la demanda sin comprometer los SLO establecidos. No obstante, el dato que más llama la atención es el comportamiento del “PDP Transaction Controller”: su latencia máxima (6 041 ms) y promedio (1 002 ms) contrastan de forma marcada con el resto de los controladores evaluados. Esto indica que el autoescalado resuelve la disponibilidad de recursos, pero no necesariamente los cuellos de botella originados en la lógica de negocio o en consultas a base de datos, un límite que la sola adición de réplicas no logra superar.

Un punto que merece discusión aparte es la composición de la tasa de éxito de solicitudes HTTP reportada en la Tabla 8 (65.87 % de código 200), concentrada la diferencia en los códigos 302 y 403. Como se precisó en la Sección 3.8.1, este

resultado responde a las reglas de whitelist, caché y autorizaciones configuradas deliberadamente para la prueba, y no a un indicio de falla del sistema; conviene, por tanto, distinguir con claridad entre respuesta esperada por política de seguridad y error funcional, evitando que ambas categorías se traten como equivalentes al reportar el indicador agregado de "éxito/error" en futuras iteraciones del panel de monitoreo.

Sobre los controles de seguridad, la correlación observada entre RBAC, autenticación JWT y cifrado HTTPS/TLS 1.3 muestra que es posible mantener estos mecanismos activos sin afectar de forma significativa el rendimiento, lo que respalda su viabilidad como práctica estándar en microservicios sin que ello implique un costo de latencia relevante. La incorporación de la literatura sobre DevSecOps, SBOM y el marco SLSA (Sección 2.2.5) permite además situar estos controles de tiempo de ejecución dentro de un marco más amplio de seguridad de extremo a extremo, que incluye la integridad de la cadena de suministro del software como complemento y no sustituto de las verificaciones Zero Trust evaluadas en este artículo.

Finalmente, la comparación presentada en la Sección 3.9 frente a una arquitectura de referencia perimetral/monolítica, si bien coherente con la evidencia reportada en la literatura, se basa en fuentes secundarias y no en un experimento controlado ejecutado bajo condiciones equivalentes; esta limitación, señalada por el informe de revisión, se aborda explícitamente en la sección siguiente.

5. Conclusiones

La implementación de un modelo de monitoreo continuo con Grafana demostró ser efectiva para sostener la observabilidad de la infraestructura, permitiendo detectar incidentes en tiempos menores a 12 segundos y con una baja tasa de falsos positivos. Esto confirma que la observabilidad proactiva contribuye directamente a la disponibilidad operativa del servicio en arquitecturas distribuidas.

El despliegue mediante la estrategia "rolling with additional batch" en AWS Elastic Beanstalk permitió actualizar la aplicación sin interrupciones perceptibles para el usuario, manteniendo tiempos de respuesta estables y sin errores críticos, lo que valida su idoneidad frente a estrategias de reemplazo total.

La arquitectura evaluada evidenció capacidad de escalamiento horizontal adecuada, con tiempos de recuperación menores a 7 segundos y disponibilidad operativa superior al 99.9 % ante fallos simulados. Sin embargo, se identificó que el autoescalado no resuelve por sí solo los cuellos de botella asociados a la lógica de negocio, como se evidenció en la latencia del PDP Transaction Controller, lo que señala una oportunidad concreta de optimización a nivel de aplicación.

En materia de seguridad, la integración de RBAC, autenticación JWT y cifrado HTTPS/TLS 1.3 se mantuvo operativa sin afectar significativamente el rendimiento, respaldando su adopción como práctica estándar en entornos de microservicios; esta evaluación de seguridad tiene, no obstante, un carácter operativo y cualitativo, y se complementa, mas no se sustituye con una futura verificación cuantitativa mediante pruebas de penetración y con la incorporación de controles de cadena de suministro (SBOM, SLSA).

La comparación indicativa frente a una arquitectura de referencia perimetral/monolítica (Sección 3.9), construida a partir de literatura secundaria, sugiere ventajas de la propuesta en materia de automatización de revisiones de seguridad, continuidad de despliegue y velocidad de escalado; su confirmación mediante un experimento controlado directo constituye la principal recomendación de trabajo futuro derivada del proceso de revisión por pares.

En conjunto, los resultados aportan evidencia de que la combinación de monitoreo proactivo, despliegues progresivos y autoescalado constituye una base sólida para sostener arquitecturas resilientes y seguras, aunque persisten áreas específicas de optimización a nivel transaccional. Se recomienda, para futuras investigaciones, profundizar en el análisis de la causa raíz de la latencia detectada, extender las pruebas a escenarios de mayor concurrencia sostenida, ejecutar la comparación controlada frente a una arquitectura de referencia y complementar la evaluación de seguridad con pruebas de penetración y controles de cadena de suministro de software.

Conflicto de intereses

Los autores declaran no tener ningún conflicto de intereses, ya sea de carácter financiero, comercial, profesional o personal, que pudiera haber influido de manera

inapropiada en el desarrollo, análisis o interpretación de los resultados presentados en este proyecto de investigación. Por ende, el presente artículo fue elaborado con fines exclusivamente educativos y de publicación académica.

Financiamiento

El artículo no tiene ningún tipo de beneficio económico, financiamiento o patrocinio por ninguna organización u empresa ni por parte de los autores o de terceros vinculados a su elaboración, todo fue elaborado de forma totalmente gratuita.

Declaración sobre inteligencia artificial

Los autores declaran que se utilizaron herramientas de inteligencia artificial generativa para los siguientes fines: Traducción del resumen e implementar ideas. Los autores asumen la plena responsabilidad del contenido del manuscrito.

Aporte de los autores

Ángel Mauricio Ramón Noblecilla: Coordinación general, redacción de la introducción, búsqueda bibliográfica, análisis bibliográfico, análisis comparativo de escenarios.

Steeven Alexander Pillacela Vargas: Búsqueda bibliográfica, redacción de resultados, análisis crítico de hallazgos, redacción de la discusión y conclusiones.

Rolando Javier Vargas Cajmarca: Diseño metodológico, estrategia de búsqueda, selección de artículos, clasificación y categorización de la información.

Referencias

Actividad de Construcción y Servicio (A.C.S.). (2022). Política de seguridad de la información. Aprobada por el Consejo de Administración. https://www.acsa.es/es/politica-de-seguridad-de-la-informacion

Amazon Web Services. (2016). Diseño de arquitecturas para la nube: Prácticas recomendadas de AWS. https://docs.aws.amazon.com/es_es/whitepapers/latest/aws-overview/introduction.html

Bhargav, M. (2026). Microservices architecture in cloud computing: A comprehensive analysis of design patterns, scalability, and performance. SSRN. https://doi.org/10.2139/ssrn.6579999

Bou Ghantous, G., & Gill, A. Q. (2021). Evaluating the DevOps reference architecture for multi-cloud IoT-applications. SN Computer Science, 2(2), 123. https://doi.org/10.1007/s42979-021-00519-6

Carrillo, J., Oña, J., & Guaña, J. (2023). La inminente integración de devOps en el futuro de la programación. Revista Ingenio Global Editorial Innova, 2(2). https://doi.org/10.62943/rig.v2n2.2023.65

Chellamalla, M., & Ravi Kiran, S. C. V. S. L. S. (2018). Implementation of DevOps architecture in the project development and deployment with help of tools. International Journal of Scientific Research in Computer Science and Engineering, 6(2), 87–95. https://doi.org/10.26438/ijsrcse/v6i2.8795

Córdova, D., Diaz, S., & Mendoza de los Santos, A. (2026). Zero Trust en la gestión de identidades y accesos en la nube: Una revisión de modelos, ventajas y limitaciones. Ingeniería Investiga, 8(00), 3-7. https://doi.org/10.47796/ing.v8i00.1342

Hidalgo de Benito, C. (2021). Monitorización de un entorno empresarial con Prometheus-Grafana [Trabajo de fin de grado, Universidad Politécnica de Madrid]. Archivo Digital UPM. https://oa.upm.es/68055/1/TFG_CARLOS_HIDALGO_DE_BENITO.pdf

Hornbeek, M. (2021). Zero trust and DevOps: Improving remote access efficiency and security.

National Institute of Standards and Technology. (2022). Secure Software Development Framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST SP 800-218). https://doi.org/10.6028/NIST.SP.800-218

Nurgaliev, I., Karavakis, E., & Aimar, A. (2016). Kibana, Grafana and Zeppelin on monitoring data. CERN openlab Summer Student Report. https://doi.org/10.5281/zenodo.61079

Ogala, J. (2022). A complete guide to DevOps best practices. Department of Computer Science, Faculty of Computing. 20(2), 2-5. https://doi.org/10.5281/zenodo.6376787

Open Source Security Foundation (OpenSSF). (2023). SLSA: Supply-chain Levels for Software Artifacts (versión 1.0). https://slsa.dev

Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. arXiv. https://doi.org/10.48550/arXiv.2103.08266

Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero trust architecture (NIST Special Publication 800-207). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-207

Wilde, N., Eddy, B., Patel, K., Cooper, N., Gamboa, V., Mishra, B., & Shah, K. (2016). Security for DevOps deployment processes: Defenses, risks, research directions. International Journal of Software Engineering & Applications, 7(6), 1–16. https://doi.org/10.5121/IJSEA.2016.7601