Propuesta de arquitectura para la capa de mediación de SIMDE

Gustavo Palacios Solorzano

Proyecto de grado entregado para obtener el título de
Magister en Ingeniería de Software

Dirigida por
MSc. Mario Julian Mora

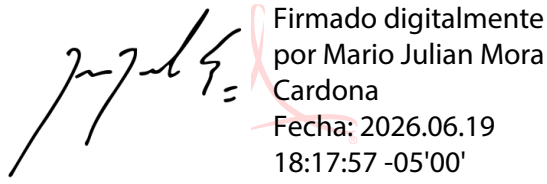
Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
19 de Junio de 2026

Santiago de Cali, 19 de Junio de 2026.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Propuesta de arquitectura para la capa de mediación de SIMDE" realizado por el estudiante de Magister en Ingeniería de Software Gustavo Palacios Solorzano (cod: 8973239), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,



MSc. Mario Julian Mora

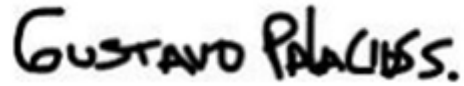
Santiago de Cali, 19 de Junio de 2026.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Propuesta de arquitectura para la capa de mediación de SIMDE" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,



Gustavo Palacios Solorzano
Código: 8973239

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

Propuesta de arquitectura para la capa de mediación de SIMDE

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Sector de Tecnologías de la Información y la Comunicación
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
4. Estudiante: Gustavo Palacios Solorzano
5. Correo electrónico: gups44551@javerianacali.edu.co
6. Dirección y teléfono: cra 17 # 21 - 16 Belalcazar, Telefono 3186765676
7. Director: Mario Julián Mora
8. Vinculación del director: (Planta/Cátedra/Externo) Cátedra
9. Correo electrónico del director: mariomora@javerianacali.edu.co
10. Co-Director (Si aplica):
11. Grupo o empresa que lo avala (Si aplica):
12. Otros grupos o empresas:
13. Palabras clave(al menos 5):Cloud Agnostic, Multinube, Microservicios, Kubernetes, Vendor Lock-in.
14. ODS que aplica al proyecto (Agenda 2030): ODS 9
15. Fecha de inicio:
16. Resumen: La transformación digital de las organizaciones exige arquitecturas de software capaces de garantizar interoperabilidad, seguridad, escalabilidad y capacidad de evolución tecnológica. En la entidad SIMDE se identificaron limitaciones en la arquitectura actual de la capa de mediación, caracterizadas por la integración de sistemas mediante accesos directos a bases de datos y desarrollos específicos para cada conexión, lo que genera un alto nivel de acoplamiento, dificulta la reutilización de servicios, incrementa la complejidad operativa y afecta atributos de calidad fundamentales como la mantenibilidad, confiabilidad y seguridad. Adicionalmente, la dependencia de tecnologías con riesgo de obsolescencia y la ausencia de mecanismos estandarizados de exposición de servicios limitan la capacidad de la organización para responder de manera ágil a nuevas necesidades del negocio. En este contexto, el

presente proyecto tuvo como objetivo diseñar una arquitectura de software para la capa de mediación de SIMDE que permitiera estandarizar y centralizar la exposición de servicios, reducir el acoplamiento entre sistemas y mejorar atributos de calidad como interoperabilidad, mantenibilidad, seguridad, confiabilidad y escalabilidad bajo un enfoque cloud-agnóstico. Para ello, se realizó un análisis de la arquitectura existente, la identificación de requerimientos arquitectónicos y atributos de calidad basados en la norma ISO/IEC 25010, así como la selección de patrones y estilos arquitectónicos orientados al desacoplamiento y la portabilidad tecnológica. Como resultado, se diseñó una arquitectura de referencia basada en principios cloud-native, microservicios y componentes de integración desacoplados, incorporando mecanismos de gestión de APIs, autenticación, administración de secretos, mensajería asíncrona, observabilidad y orquestación de contenedores. Asimismo, se desarrolló un prototipo funcional en un entorno de desarrollo para validar la viabilidad técnica de la propuesta y se realizó su evaluación mediante el método ATAM, obteniendo resultados favorables en escenarios relacionados con disponibilidad y fiabilidad. Las principales lecciones aprendidas evidencian la importancia de adoptar estándares abiertos, mecanismos de centralización de servicios y estrategias de desacoplamiento que permitan reducir la dependencia tecnológica de proveedores específicos y faciliten la evolución sostenible de las plataformas empresariales.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas e instituciones que, de una u otra manera, contribuyeron al desarrollo y culminación de este trabajo de investigación.

En primer lugar, agradezco a Dios por brindarme la fortaleza, la sabiduría y la perseverancia necesarias para superar los desafíos presentados a lo largo de este proceso académico y personal.

A mi familia, especialmente a mis padres, por su amor incondicional, sus enseñanzas y el apoyo constante que me han brindado en cada etapa de mi vida. Su confianza y sacrificio han sido una fuente permanente de inspiración para alcanzar mis metas.

A mi esposa e hijos, por su comprensión, paciencia y apoyo durante las largas jornadas dedicadas al desarrollo de esta investigación. Su compañía y motivación fueron fundamentales para mantener el compromiso y la determinación necesarios para culminar este proyecto.

A mi director de tesis, por su orientación, conocimientos, disposición y acompañamiento académico durante el desarrollo de esta investigación. Sus valiosas observaciones y recomendaciones contribuyeron significativamente al fortalecimiento de este trabajo.

A los docentes de la Maestría, quienes compartieron sus conocimientos y experiencias, aportando a mi formación profesional y académica. Cada una de sus enseñanzas dejó una huella importante en mi proceso de aprendizaje.

A los miembros del jurado y evaluadores de esta investigación, por el tiempo dedicado a la revisión de este trabajo y por sus valiosos comentarios y sugerencias, los cuales contribuyeron a enriquecer y mejorar la calidad de esta tesis.

A mis compañeros de estudio y colegas profesionales, quienes compartieron experiencias, conocimientos y reflexiones que fortalecieron mi visión sobre los temas abordados en esta investigación.

Asimismo, agradezco a los profesionales, expertos y organizaciones que, mediante el intercambio de conocimientos, documentación técnica y experiencias en el ámbito de las arquitecturas tecnológicas y la computación en la nube, aportaron elementos valiosos para el desarrollo de este estudio.

Finalmente, agradezco a todas aquellas personas que, directa o indirectamente, contribuyeron a la realización de este proyecto. La culminación de esta tesis representa no solo un logro académico, sino también una experiencia de crecimiento personal y profesional que reafirma el valor de la disciplina, el esfuerzo y el aprendizaje continuo. Este trabajo es el resultado del apoyo, la confianza y la colaboración de quienes me acompañaron durante este importante camino.

Resumen

La adopción de servicios en la nube ha permitido a las organizaciones mejorar la escalabilidad, disponibilidad y eficiencia de sus sistemas de información. Sin embargo, la dependencia de proveedores específicos genera riesgos asociados al bloqueo tecnológico (vendor lock-in), limitando la portabilidad y flexibilidad de las aplicaciones. Este trabajo propone una arquitectura de referencia cloud agnostic orientada a microservicios, diseñada para facilitar la interoperabilidad y el despliegue en diferentes proveedores de nube. La propuesta integra tecnologías de código abierto como Kubernetes, Kong, HashiCorp Vault, Keycloak, RabbitMQ, Redis y herramientas de observabilidad, permitiendo la gestión segura, escalable y resiliente de aplicaciones empresariales. La investigación se desarrolla bajo un enfoque aplicado, mediante el diseño y validación de la arquitectura en un entorno de pruebas multinube. Los resultados esperados buscan demostrar que la solución propuesta reduce la dependencia tecnológica, mejora la portabilidad de los servicios y facilita la adopción de estrategias multinube e híbridas en las organizaciones.

Palabras clave: Cloud Agnostic, Multinube, Microservicios, Kubernetes, Vendor Lock-in.

Abstract

The adoption of cloud computing services has enabled organizations to improve the scalability, availability, and efficiency of their information systems. However, dependence on specific cloud providers creates risks associated with vendor lock-in, limiting application portability and flexibility. This research proposes a cloud-agnostic reference architecture based on a microservices approach, designed to facilitate interoperability and deployment across different cloud providers. The proposed solution integrates open-source technologies such as Kubernetes, Kong, HashiCorp Vault, Keycloak, RabbitMQ, Redis, and observability tools, enabling secure, scalable, and resilient management of enterprise applications. The study follows an applied research approach through the design and validation of the architecture within a multi-cloud testing environment. The expected results aim to demonstrate that the proposed architecture reduces technological dependency, improves service portability, and facilitates the adoption of multi-cloud and hybrid cloud strategies in organizations. Furthermore, the proposal contributes to establishing architectural guidelines that support cloud-native application development while preserving provider independence and operational flexibility.

Keywords: Cloud Agnostic, Multi-Cloud, Microservices, Kubernetes, Vendor Lock-in.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

Presentación del tema. La transformación digital ha llevado a las organizaciones a replantear sus arquitecturas tecnológicas con el fin de responder de manera ágil, segura y escalable a las demandas del mercado. En este contexto, la computación en la nube y los enfoques *cloud-native* han emergido como pilares fundamentales para la evolución de los sistemas de información, permitiendo optimizar costos, mejorar la disponibilidad y acelerar el desarrollo de productos digitales. No obstante, su adopción también ha introducido desafíos relevantes, especialmente en términos de dependencia tecnológica (*vendor lock-in*), baja portabilidad y limitaciones en la interoperabilidad entre sistemas.

En particular, la entidad SIMDE enfrenta restricciones estructurales en su arquitectura actual, caracterizadas por un alto acoplamiento entre sistemas y la ausencia de mecanismos de estandarización en la exposición de información. Estas condiciones, identificadas durante el diagnóstico de la arquitectura existente, impactan negativamente atributos de calidad como la mantenibilidad, la seguridad, la confiabilidad y la escalabilidad, los cuales constituyen factores esenciales para la evolución y sostenibilidad de los sistemas de software (Bass et al., 2022; ISO, 2011). En consecuencia, resulta necesario diseñar una arquitectura que mejore la interoperabilidad, reduzca la dependencia tecnológica y facilite la integración entre los sistemas institucionales.

Objetivos de la investigación. En este contexto, la presente investigación tiene como objetivo general proponer una arquitectura de software para la capa de mediación de SIMDE que permita estandarizar y centralizar la exposición de servicios, mejorando atributos de calidad como la fiabilidad, la disponibilidad, la seguridad y la mantenibilidad. Para lograrlo, se plantean como objetivos específicos: (i) diseñar una arquitectura de integración desacoplada que facilite la interoperabilidad entre sistemas; (ii) incorporar principios de diseño orientados a atributos de calidad, especialmente fiabilidad y disponibilidad; (iii) definir una arquitectura *cloud-agnostic* que reduzca la dependencia de proveedores tecnológicos; (iv) implementar un prototipo en un entorno de desarrollo controlado; y (v) evaluar la arquitectura mediante técnicas de análisis como ATAM. Estos objetivos permiten dar respuesta a la pregunta de investigación relacionada con el diseño de arquitecturas que habiliten la evolución tecnológica organizacional.

Metodología de la investigación. La investigación adopta un enfoque aplicado, con un alcance descriptivo y un énfasis cualitativo, orientado al diseño y validación de una solución arquitectónica. La metodología se estructura en varias fases: análisis de la arquitectura actual, identificación de brechas, diseño de la arquitectura propuesta, implementación de un prototipo en ambiente de desarrollo y evaluación de la solución con base en atributos de calidad priorizados. Para la recolección

de información se emplea revisión documental y análisis técnico de la arquitectura existente, mientras que la validación se realiza mediante escenarios de evaluación arquitectónica, lo que permite verificar la viabilidad técnica de la propuesta.

Relevancia e impacto. La relevancia de esta investigación radica en su contribución al diseño de arquitecturas de software en entornos organizacionales que buscan evolucionar hacia modelos más flexibles y sostenibles. En particular, el enfoque *cloud-agnostic* propuesto permite mitigar riesgos asociados a la dependencia tecnológica, mejorar la eficiencia operativa y fortalecer la capacidad de innovación. Asimismo, el trabajo aporta a la disciplina de la ingeniería de software al integrar atributos de calidad definidos en estándares como ISO/IEC 25010 dentro del proceso de diseño arquitectónico, generando valor tanto a nivel académico como práctico.

Estructura de la tesis. La presente tesis se organiza de la siguiente manera:

- Capítulo 1: Se presenta el planteamiento y la justificación del problema, objetivo general, objetivos especificos, Delimitacion y alcances, Metodologia de la investigacion
- Capítulo 2: Se presenta Marco Teorico y Estado del Arte.
- Capítulo 3: Se realiza Diagnóstico Arquitectónico Actual, Definición de Requerimientos Arquitectónicos y Atributos de Calidad, Diseño de la Arquitectura Tecnológica de la Solución, implementacion del prototipo y Análisis de costos de la implementación
- Capítulo 4: Se realiza la Evaluación de la Arquitectura Propuesta mediante ATAM
- Capítulo 5: Se presentan las conclusiones, trabajos futuros y Lecciones Aprendidas

1.2. Definición del problema

1.2.1. Planteamiento del Problema

La entidad SIMDE, en su proceso de crecimiento y ampliación del portafolio de servicios digitales, presenta limitaciones estructurales en su arquitectura tecnológica actual, particularmente en la capa de mediación encargada de la exposición e integración de información. En el estado actual, la comunicación entre sistemas se realiza mediante accesos directos a las bases de datos y desarrollos específicos por cada integración, lo que evidencia la ausencia de mecanismos de estandarización, centralización y desacoplamiento.

Esta situación genera un alto nivel de acoplamiento entre aplicaciones, dificultando la interoperabilidad, la reutilización de servicios y la evolución progresiva del sistema. Como consecuencia, se incrementan los tiempos de implementación y la complejidad operativa, dado que las integraciones requieren desarrollos específicos no reutilizables, lo cual impacta negativamente la eficiencia operativa y la capacidad de respuesta ante nuevas demandas del negocio (González et al., 2021).

Adicionalmente, la dependencia de tecnologías en estado de obsolescencia y la exposición directa de datos incrementan los riesgos de fallos operativos y vulnerabilidades de seguridad. De acuerdo con reportes de la industria, las organizaciones con arquitecturas no modernizadas presentan mayores

tasas de incidentes críticos y costos elevados asociados a la gestión de fallas y brechas de seguridad (IBM Security, 2023).

Desde la perspectiva de calidad del software, esta problemática afecta atributos fundamentales definidos en el modelo ISO/IEC 25010, tales como la confiabilidad, la seguridad, la mantenibilidad y la escalabilidad, los cuales son esenciales en entornos empresariales orientados a servicios digitales (ISO, 2011).

En este contexto, se configura una situación indeseable que limita la capacidad de la organización para innovar, escalar sus servicios y adaptarse a entornos tecnológicos dinámicos. Si bien existen enfoques arquitectónicos modernos, como microservicios, arquitecturas orientadas a eventos y soluciones cloud-native, no se dispone de una estrategia claramente definida que permita integrarlos de manera efectiva en una arquitectura de mediación que sea estandarizada, desacoplada y cloud-agnóstica.

Por lo tanto, el problema de investigación radica en el desconocimiento de cómo diseñar una arquitectura de software que permita transformar la capa de mediación actual hacia un modelo que garantice interoperabilidad, portabilidad, seguridad y escalabilidad, sin generar dependencia tecnológica de un proveedor específico.

1.2.2. Formulación del Problema

¿Cómo diseñar una arquitectura para la capa de mediación en la entidad SIMDE que permita estandarizar y centralizar la exposición de servicios, reduciendo el acoplamiento entre sistemas y mejorando atributos de calidad como interoperabilidad, mantenibilidad, seguridad, confiabilidad y escalabilidad, bajo un enfoque cloud-agnóstico?

1.2.3. Sistematización del Problema

- ¿Cuáles son las limitaciones arquitectónicas actuales en la capa de mediación de SIMDE que afectan la interoperabilidad y mantenibilidad de los sistemas?
- ¿Qué atributos de calidad son críticos para garantizar la evolución tecnológica de la organización?
- ¿Qué patrones y estilos arquitectónicos permiten desacoplar y estandarizar la exposición de servicios?
- ¿Cómo diseñar una arquitectura de mediación que minimice la dependencia tecnológica (*vendor lock-in*)?
- ¿Cómo validar que la arquitectura propuesta mejora la fiabilidad y disponibilidad de los servicios?

1.3. Objetivos del proyecto

1.3.1. Objetivo General

Diseñar una arquitectura de software para la capa de mediación en la entidad SIMDE que permita estandarizar y centralizar la exposición de servicios, reduciendo el acoplamiento entre sistemas y mejorando atributos de calidad como la interoperabilidad, mantenibilidad, seguridad, confiabilidad y escalabilidad, bajo un enfoque cloud-agnóstico.

1.3.2. Objetivos Específicos

- Analizar la arquitectura actual de la entidad SIMDE, identificando las limitaciones en la exposición de información, el nivel de acoplamiento entre sistemas y los atributos de calidad afectados.
- Definir los requerimientos arquitectónicos y atributos de calidad necesarios para la solución, alineados con el modelo ISO/IEC 25010.
- Diseñar una arquitectura de mediación que incorpore mecanismos de estandarización, versionamiento y centralización de servicios, orientada al desacoplamiento de los sistemas.
- Diseñar una arquitectura cloud-agnóstica que minimice la dependencia de proveedores tecnológicos y favorezca la portabilidad e interoperabilidad.
- Implementar un prototipo de la arquitectura propuesta en un entorno de desarrollo (DEV) que permita validar su viabilidad técnica.
- Evaluar la arquitectura propuesta mediante escenarios basados en atributos de calidad, utilizando el método ATAM, con énfasis en fiabilidad y disponibilidad.

1.4. Delimitaciones y Alcances

El presente trabajo se desarrolló con el propósito de proponer, diseñar, implementar y evaluar una arquitectura de mediación cloud-agnóstica para la entidad SIMDE. A partir de los objetivos específicos planteados, se definieron los siguientes alcances y delimitaciones:

1.4.1. Alcances

- Se realizó un análisis de la arquitectura actual de SIMDE, identificando los principales problemas relacionados con la exposición de servicios, el alto acoplamiento entre sistemas y los atributos de calidad afectados, tales como mantenibilidad, interoperabilidad y escalabilidad.
- Se definieron los requerimientos arquitectónicos y atributos de calidad, alineados con el modelo ISO/IEC 25010, priorizando características como disponibilidad, fiabilidad, seguridad e interoperabilidad.

- Se diseñó una arquitectura de mediación que incorporó mecanismos de estandarización de interfaces (APIs REST), versionamiento de servicios y centralización del acceso mediante un API Gateway, orientada a reducir el acoplamiento entre sistemas.
- Se propuso una arquitectura cloud-agnóstica basada en principios de portabilidad, desacoplamiento de infraestructura y uso de tecnologías abiertas, evitando dependencias fuertes con proveedores específicos de nube.
- Se implementó un prototipo funcional en un entorno de desarrollo (DEV), el cual permitió validar la viabilidad técnica de la solución propuesta, incluyendo la integración de componentes clave como gestión de APIs, mensajería y seguridad.
- Se evaluó la arquitectura propuesta mediante el método ATAM (Architecture Tradeoff Analysis Method), utilizando escenarios basados en atributos de calidad, con énfasis en disponibilidad y fiabilidad, permitiendo identificar riesgos, sensibilidades y compensaciones arquitectónicas.

1.4.2. Delimitaciones

- El análisis de la arquitectura actual se limitó a los sistemas y componentes más representativos de SIMDE, por lo que no se abordó la totalidad del ecosistema tecnológico de la entidad.
- La definición de requerimientos arquitectónicos no contempló requerimientos funcionales detallados de cada sistema, sino que se enfocó en atributos de calidad (requerimientos no funcionales).
- El diseño de la arquitectura de mediación se desarrolló a nivel lógico y de referencia, sin incluir especificaciones detalladas de implementación para todos los componentes en un entorno productivo.
- La arquitectura cloud-agnóstica propuesta no implicó la migración completa a múltiples proveedores de nube, sino la definición de lineamientos y un prototipo que demostrara la portabilidad.
- La implementación del prototipo se realizó únicamente en un entorno de desarrollo (DEV), por lo cual no se evaluaron aspectos asociados a ambientes productivos como pruebas de carga, costos operativos reales o despliegues a gran escala.
- La evaluación mediante ATAM se centró en escenarios seleccionados de calidad, principalmente disponibilidad y fiabilidad, sin abarcar de manera exhaustiva todos los atributos definidos en el modelo ISO/IEC 25010.
- No se incluyó la implementación de aspectos organizacionales como gobierno de TI, gestión del cambio o adopción institucional, los cuales exceden el alcance técnico de este trabajo.

1.4.3. Límite del Trabajo

El trabajo llegó hasta la validación técnica de la arquitectura propuesta mediante un prototipo funcional en ambiente de desarrollo y su evaluación cualitativa mediante el método ATAM, sin contemplar su despliegue en producción ni su adopción organizacional completa dentro de SIMDE.

1.5. Justificación del Trabajo de Grado

El desarrollo del presente trabajo se justificó en la necesidad de abordar las limitaciones identificadas en la arquitectura tecnológica de la entidad SIMDE, particularmente en lo relacionado con el alto acoplamiento entre sistemas, la baja estandarización en la exposición de servicios y la limitada capacidad de interoperabilidad. Estas condiciones afectaban directamente atributos de calidad como la mantenibilidad, la escalabilidad y la disponibilidad, generando barreras para la evolución tecnológica y la incorporación de nuevos servicios digitales.

La solución propuesta, basada en una arquitectura de mediación y principios cloud-agnósticos, contribuyó a mejorar estas condiciones mediante la introducción de mecanismos de desacoplamiento, estandarización y gobernanza de servicios. Como resultado, al resolverse el problema de investigación, la entidad logró una mayor capacidad para integrar sistemas heterogéneos, reducir la dependencia tecnológica de proveedores específicos y acelerar la entrega de nuevos servicios digitales.

Desde una perspectiva técnica, la adopción de una arquitectura alineada con el modelo ISO/IEC 25010 permitió fortalecer atributos de calidad clave. En particular, se evidenció una mejora en la mantenibilidad mediante la reducción del acoplamiento entre componentes, la cual se reflejó en la disminución del número de dependencias directas entre sistemas y en una mayor facilidad para realizar cambios evolutivos. Asimismo, la incorporación de un API Gateway y mecanismos de versionamiento permitió mejorar la interoperabilidad y la capacidad de evolución de los servicios.

Adicionalmente, durante la implementación del prototipo en ambiente de desarrollo (DEV), se observaron mejoras en indicadores técnicos tales como:

- Reducción del acoplamiento entre sistemas, evidenciado en la eliminación de integraciones punto a punto.
- Disminución del tiempo de integración de nuevos servicios, gracias a interfaces estandarizadas.
- Mejora en la disponibilidad percibida, al incorporar mecanismos de intermediación y desacoplamiento asincrónico.
- Mayor facilidad de despliegue y portabilidad, al utilizar tecnologías independientes del proveedor de nube.

Estos resultados son consistentes con lo planteado en la literatura, donde se establece que arquitecturas desacopladas y basadas en servicios mejoran significativamente la mantenibilidad y escalabilidad de los sistemas (Bass et al., 2012). Asimismo, el uso de arquitecturas cloud-agnósticas

contribuye a reducir el riesgo de *vendor lock-in* y optimizar la gestión de recursos tecnológicos (Zhang et al., 2010).

En términos económicos, la solución permitió identificar oportunidades de reducción de costos asociados al mantenimiento de sistemas altamente acoplados y a la dependencia de tecnologías propietarias. La estandarización de servicios y la reutilización de componentes favorecieron una mayor eficiencia en el desarrollo, lo cual se traduce en una disminución del esfuerzo requerido para la implementación de nuevas funcionalidades.

Desde el punto de vista organizacional, la arquitectura propuesta favoreció la agilidad en el desarrollo y despliegue de servicios, permitiendo a la entidad responder de manera más eficiente a cambios en los requerimientos del entorno. Esto impacta positivamente la competitividad institucional y la capacidad de innovación, aspectos fundamentales en procesos de transformación digital.

En el ámbito social, la mejora en la disponibilidad y calidad de los servicios tecnológicos repercute directamente en los usuarios finales, quienes se benefician de sistemas más confiables, accesibles y eficientes. Esto resulta especialmente relevante en contextos donde los servicios digitales constituyen el principal canal de interacción con la entidad.

En relación con la evaluación arquitectónica, la aplicación del método ATAM permitió validar la arquitectura propuesta mediante escenarios de calidad enfocados en disponibilidad y fiabilidad. A través de este proceso, se identificaron riesgos, puntos de sensibilidad y compensaciones (*trade-offs*), lo cual aportó evidencia cualitativa sobre la viabilidad y robustez de la solución planteada.

Finalmente, este trabajo generó una contribución académica y práctica al documentar el diseño, implementación y evaluación de una arquitectura de mediación cloud-agnóstica en un contexto real. Esto no solo permitió validar la solución propuesta, sino también aportar evidencia sobre cómo este tipo de enfoques pueden mejorar atributos de calidad en sistemas empresariales.

En conclusión, la solución propuesta no solo resolvió las limitaciones identificadas en la arquitectura de SIMDE, sino que también generó beneficios medibles en términos técnicos, económicos y organizacionales, así como aportes relevantes al conocimiento en el área de arquitectura de software.

1.6. Metodología de la investigación

La presente investigación se desarrolla bajo un enfoque aplicado orientado al diseño, implementación y validación de una arquitectura de mediación basada en microservicios, con el propósito de mejorar la integración, seguridad, escalabilidad y observabilidad de los servicios tecnológicos de la organización objeto de estudio. La metodología propuesta permite establecer un proceso sistemático para identificar las necesidades existentes, diseñar una solución tecnológica adecuada y evaluar su efectividad mediante la implementación de un prototipo funcional.

1.6.1. Enfoque de la Investigación

La investigación adopta un enfoque mixto, integrando elementos cualitativos y cuantitativos. El enfoque cualitativo permite analizar la situación actual de la organización, identificar problemáticas

asociadas a la integración de servicios y definir los requerimientos funcionales y no funcionales de la arquitectura propuesta. Por su parte, el enfoque cuantitativo se emplea para medir el desempeño de la solución mediante indicadores relacionados con tiempos de respuesta, disponibilidad, escalabilidad y capacidad de procesamiento.

Según Hernández Sampieri y Mendoza (2018), el enfoque mixto permite una comprensión más amplia del fenómeno estudiado al combinar las fortalezas de los métodos cualitativos y cuantitativos, proporcionando una visión integral del problema de investigación.

1.6.2. Tipo de Investigación

La investigación corresponde a un estudio de tipo aplicado, debido a que busca resolver una problemática específica mediante la utilización de conocimientos científicos y tecnológicos existentes.

Asimismo, presenta características descriptivas, propositivas y experimentales:

- **Descriptiva**: porque identifica y caracteriza la situación actual de la arquitectura tecnológica de la organización.
- **Propositiva**: porque plantea una arquitectura de mediación basada en microservicios para atender las necesidades identificadas.
- **Experimental**: porque la solución propuesta es implementada y sometida a pruebas para evaluar su comportamiento y desempeño.

1.6.3. Diseño de la Investigación

La investigación se desarrolla bajo un diseño no experimental de carácter transversal durante las fases de análisis y diseño, complementado con una fase experimental para la validación de la propuesta tecnológica.

El proceso metodológico se estructura en cuatro fases principales: diagnóstico, diseño, implementación y validación.

1.6.4. Fase I: Diagnóstico de la Arquitectura Actual

Durante esta fase se realiza el análisis de la infraestructura tecnológica existente con el fin de identificar limitaciones, riesgos y oportunidades de mejora.

Las actividades desarrolladas comprenden:

- Levantamiento de información de la infraestructura actual.
- Identificación de sistemas, aplicaciones y servicios existentes.
- Análisis de los mecanismos actuales de integración.
- Identificación de problemas asociados a seguridad, disponibilidad y escalabilidad.

- Elaboración de modelos arquitectónicos AS-IS.
- Evaluación de atributos de calidad basados en la norma ISO/IEC 25010.

Como resultado de esta fase se obtiene una caracterización detallada de la arquitectura existente y un conjunto de requerimientos que servirán como base para el diseño de la solución propuesta.

1.6.5. Fase II: Diseño de la Arquitectura Propuesta

A partir de los hallazgos obtenidos en la fase de diagnóstico, se diseña una arquitectura de mediación basada en microservicios orientada a garantizar la interoperabilidad entre sistemas y la independencia respecto a proveedores específicos de infraestructura.

La arquitectura incorpora los siguientes componentes tecnológicos:

- API Gateway mediante Kong.
- Gestión de identidades y accesos mediante Keycloak.
- Gestión centralizada de secretos mediante HashiCorp Vault.
- Comunicación asíncrona mediante RabbitMQ.
- Caché distribuida mediante Redis.
- Persistencia de datos mediante PostgreSQL.
- Orquestación de contenedores mediante Kubernetes.
- Observabilidad mediante Prometheus y Grafana.

Durante esta fase se elaboran los artefactos arquitectónicos correspondientes, incluyendo:

- Diagramas C4.
- Diagramas de componentes.
- Diagramas de despliegue.
- Diagramas de flujo de autenticación y autorización.
- Diagramas de integración de servicios.

1.6.6. Fase III: Implementación del Prototipo

La validación de la propuesta requiere la construcción de un prototipo funcional que permita demostrar la viabilidad técnica de la arquitectura diseñada.

Las actividades ejecutadas incluyen:

- Construcción de imágenes Docker para cada componente.
- Implementación de microservicios desarrollados bajo arquitectura desacoplada.
- Configuración de Kong como puerta de enlace de servicios.
- Integración de Keycloak para autenticación y autorización.
- Configuración de HashiCorp Vault para la gestión de secretos.
- Implementación de mensajería mediante RabbitMQ.
- Configuración de Redis como mecanismo de caché.
- Configuración de herramientas de monitoreo y observabilidad.

El entorno de implementación se diseña siguiendo principios cloud-agnostic, permitiendo su despliegue en diferentes proveedores de infraestructura sin modificaciones significativas en la arquitectura.

1.6.7. Fase IV: Validación y Evaluación

La evaluación de la arquitectura propuesta se realiza mediante la ejecución de pruebas funcionales y no funcionales orientadas a verificar el cumplimiento de los atributos de calidad establecidos.

Las pruebas contemplan los siguientes aspectos:

- Rendimiento.
- Escalabilidad.
- Disponibilidad.
- Seguridad.
- Interoperabilidad.
- Observabilidad.

Los resultados obtenidos permiten comparar el comportamiento de la solución propuesta frente a la situación inicial identificada durante la fase de diagnóstico.

1.6.8. Población y Unidad de Análisis

La unidad de análisis corresponde a la arquitectura tecnológica utilizada para la integración de servicios empresariales dentro de la organización objeto de estudio.

La población está conformada por los servicios, aplicaciones, componentes de infraestructura y mecanismos de integración que participan en los procesos de intercambio de información institucional.

1.6.9. Técnicas e Instrumentos de Recolección de Información

Para la obtención de información se emplean las siguientes técnicas:

- Observación directa.
- Revisión documental.
- Análisis arquitectónico.
- Pruebas de rendimiento.
- Pruebas funcionales.

Los instrumentos utilizados incluyen:

- Matrices de análisis arquitectónico.
- Guías de observación.
- Registros de monitoreo.
- Reportes generados por Kubernetes.
- Métricas obtenidas mediante Prometheus.
- Tableros de monitoreo de Grafana.

1.6.10. Variables de Evaluación

Con el propósito de determinar la efectividad de la arquitectura propuesta, se consideran las variables presentadas en la Tabla 1.1.

Tabla 1.1: Variables de evaluación de la arquitectura propuesta

Variable	**Indicador**
Rendimiento	Tiempo promedio de respuesta de los servicios
Escalabilidad	Capacidad de crecimiento horizontal bajo carga
Disponibilidad	Porcentaje de servicios operativos
Seguridad	Gestión centralizada de identidades y secretos
Interoperabilidad	Nivel de integración entre servicios
Observabilidad	Disponibilidad de métricas, registros y trazas

1.6.11. Método de Validación

La validación de la propuesta se realiza mediante un caso de estudio, en el cual se implementa la arquitectura diseñada y se ejecutan diferentes escenarios de prueba para verificar el cumplimiento de los requisitos funcionales y no funcionales establecidos.

Los resultados obtenidos permiten determinar el impacto de la solución propuesta sobre los atributos de calidad analizados, especialmente en términos de seguridad, mantenibilidad, disponibilidad, escalabilidad y portabilidad.

Capítulo 2

Marco de referencia

2.1. Marco Teórico

2.1.1. Arquitectura de Software

2.1.1.1. Concepto de Arquitectura de Software

La arquitectura de software constituye la estructura fundamental de un sistema y define los componentes que lo conforman, las relaciones existentes entre ellos y los principios que orientan su diseño y evolución. Según Bass et al. (2022), la arquitectura representa el conjunto de decisiones de diseño con mayor impacto sobre la calidad y el comportamiento de un sistema.

La arquitectura permite gestionar la complejidad inherente a los sistemas modernos mediante la definición de estructuras organizadas que facilitan el desarrollo, mantenimiento y evolución de las aplicaciones. Asimismo, proporciona mecanismos para abordar requerimientos relacionados con seguridad, disponibilidad, escalabilidad, interoperabilidad y mantenibilidad (Bass et al., 2022).

En entornos empresariales, la arquitectura de software se convierte en un elemento estratégico para garantizar que las soluciones tecnológicas respondan de manera eficiente a los objetivos organizacionales y a los cambios constantes del entorno. Una arquitectura adecuadamente diseñada facilita la alineación entre los objetivos de negocio y las capacidades tecnológicas, permitiendo una mayor capacidad de adaptación frente a nuevos requerimientos y cambios en el entorno tecnológico (Richards, 2020).

2.1.1.2. Atributos de Calidad

Los atributos de calidad representan propiedades que describen el comportamiento de un sistema más allá de sus funcionalidades. Estos atributos permiten evaluar la capacidad de una solución tecnológica para satisfacer necesidades relacionadas con rendimiento, seguridad, mantenibilidad y disponibilidad (Bass et al., 2022).

La norma ISO/IEC 25010 establece un modelo de calidad compuesto por características y subcaracterísticas utilizadas para evaluar productos software. Dentro de esta investigación se priorizan los atributos de escalabilidad, seguridad, disponibilidad, interoperabilidad, modificabilidad y mantenibilidad debido a su importancia para sistemas empresariales distribuidos (ISO, 2011).

La escalabilidad se refiere a la capacidad de un sistema para soportar incrementos en la carga de trabajo sin degradar significativamente su desempeño. La seguridad comprende mecanismos destinados a proteger la confidencialidad, integridad y disponibilidad de la información. La interoperabilidad permite la comunicación e integración entre sistemas heterogéneos. La modificabilidad

facilita la incorporación de cambios con un impacto mínimo sobre otros componentes, mientras que la mantenibilidad permite realizar correcciones y mejoras de manera eficiente (Bass et al., 2022; ISO, 2011).

2.1.1.3. Métodos de Evaluación Arquitectónica

La evaluación arquitectónica constituye un proceso sistemático orientado a determinar el impacto de las decisiones de diseño sobre los atributos de calidad de un sistema. Su propósito consiste en identificar riesgos potenciales, validar decisiones arquitectónicas y garantizar que la solución propuesta responda adecuadamente a los requerimientos establecidos.

Entre los métodos más reconocidos se encuentran SAAM (Software Architecture Analysis Method), CBAM (Cost Benefit Analysis Method) y ATAM (Architecture Tradeoff Analysis Method). Estos enfoques permiten analizar arquitecturas desde diferentes perspectivas, considerando aspectos técnicos, económicos y organizacionales.

2.1.1.4. Architecture Tradeoff Analysis Method (ATAM)

ATAM es un método de evaluación arquitectónica desarrollado por el Software Engineering Institute (SEI) con el objetivo de analizar cómo las decisiones de diseño afectan los atributos de calidad de una arquitectura.

El método se fundamenta en la construcción de escenarios de calidad y la identificación de riesgos, puntos de sensibilidad y compromisos entre atributos arquitectónicos. Su aplicación permite comprender cómo determinadas decisiones pueden beneficiar un atributo mientras afectan negativamente otro.

ATAM se estructura en varias fases que incluyen la presentación de la arquitectura, la identificación de drivers arquitectónicos, la construcción del árbol de utilidad, el análisis de escenarios y la identificación de riesgos y tradeoffs.

Debido a su capacidad para evaluar arquitecturas complejas, ATAM es ampliamente utilizado en proyectos empresariales y constituye el método seleccionado para validar la arquitectura propuesta en esta investigación.

2.1.2. Arquitectura Basada en Microservicios

2.1.2.1. Conceptos Fundamentales

La arquitectura de microservicios es un estilo arquitectónico que estructura una aplicación como un conjunto de servicios independientes que implementan capacidades específicas del negocio. Cada servicio puede desarrollarse, desplegarse y escalarse de forma autónoma.

Según Newman (2021), los microservicios promueven la descentralización tecnológica y organizacional, permitiendo que los equipos trabajen de manera independiente sobre diferentes componentes de una misma solución.

Este enfoque reduce el acoplamiento entre módulos y favorece la evolución continua de los sistemas.

2.1.2.2. Ventajas de los Microservicios

Entre los principales beneficios de las arquitecturas basadas en microservicios se encuentran:

- Escalabilidad independiente.
- Mayor disponibilidad.
- Despliegues desacoplados.
- Flexibilidad tecnológica.
- Facilidad de mantenimiento.
- Menor impacto ante fallos.

2.1.2.3. Desafíos de los Microservicios

A pesar de sus ventajas, la adopción de microservicios introduce desafíos relacionados con la gestión de la complejidad distribuida, la observabilidad, la seguridad y la comunicación entre servicios.

Asimismo, requiere mecanismos especializados para el monitoreo, descubrimiento de servicios, gestión de configuraciones y coordinación de despliegues.

2.1.3. Computación en la Nube

2.1.3.1. Cloud Computing

La computación en la nube es un modelo que permite acceder bajo demanda a recursos informáticos compartidos a través de internet.

Según Mell y Grance (2011), este paradigma se caracteriza por el autoservicio bajo demanda, acceso ubicuo a la red, agrupación de recursos, elasticidad rápida y servicio medido.

La computación en la nube constituye la base tecnológica sobre la cual se construyen las arquitecturas modernas de software.

2.1.3.2. Arquitecturas Cloud-Native

Las arquitecturas cloud-native son aquellas diseñadas específicamente para aprovechar las capacidades de la computación en la nube.

Este enfoque integra tecnologías como contenedores, microservicios, automatización y orquestación, permitiendo construir sistemas resilientes, escalables y altamente disponibles.

Las arquitecturas cloud-native facilitan la entrega continua de software y favorecen la adopción de prácticas DevOps y DevSecOps.

2.1.3.3. Arquitecturas Cloud-Agnostic

El concepto cloud-agnostic se refiere a arquitecturas que pueden ejecutarse en diferentes proveedores de infraestructura sin modificaciones significativas.

Este enfoque busca reducir la dependencia tecnológica de un proveedor específico mediante el uso de estándares abiertos y tecnologías portables.

La adopción de herramientas como Kubernetes, Docker, Kong, RabbitMQ y HashiCorp Vault favorece la construcción de plataformas cloud-agnostic.

2.1.3.4. Vendor Lock-In

El vendor lock-in corresponde a la dependencia tecnológica generada cuando una organización utiliza servicios exclusivos de un proveedor de nube.

Esta situación dificulta la migración de aplicaciones hacia otras plataformas y puede incrementar costos operativos y riesgos tecnológicos.

Las arquitecturas cloud-agnostic surgen como una estrategia para minimizar este problema mediante la utilización de tecnologías independientes del proveedor.

2.1.4. Tecnologías para la Implementación de Arquitecturas Cloud-Agnostic

2.1.4.1. Contenedores y Docker

Los contenedores constituyen una tecnología de virtualización ligera que permite empaquetar aplicaciones junto con todas sus dependencias, bibliotecas y configuraciones necesarias para su ejecución. A diferencia de las máquinas virtuales tradicionales, los contenedores comparten el núcleo del sistema operativo anfitrión, lo que reduce significativamente el consumo de recursos y mejora la velocidad de despliegue.

Docker es una plataforma de código abierto ampliamente utilizada para la creación, distribución y ejecución de contenedores. Mediante el uso de imágenes, Docker permite garantizar la consistencia entre ambientes de desarrollo, pruebas y producción, reduciendo problemas asociados a diferencias de configuración entre entornos.

La utilización de contenedores favorece la portabilidad de las aplicaciones y constituye uno de los pilares fundamentales de las arquitecturas cloud-native y cloud-agnostic.

2.1.4.2. Orquestación con Kubernetes

Kubernetes es una plataforma de código abierto diseñada para automatizar el despliegue, administración y escalamiento de aplicaciones contenerizadas. Originalmente desarrollado por Google y actualmente mantenido por la Cloud Native Computing Foundation (CNCF), Kubernetes se ha consolidado como el estándar de facto para la orquestación de contenedores.

Entre sus principales funcionalidades se encuentran el balanceo de carga, el escalamiento automático, la recuperación ante fallos, la gestión de configuraciones y la administración declarativa de la infraestructura.

La utilización de Kubernetes permite construir plataformas altamente disponibles y escalables, facilitando además la portabilidad de aplicaciones entre diferentes proveedores de nube y entornos on-premise. Por esta razón, Kubernetes constituye uno de los principales habilitadores de arquitecturas cloud-agnostic.

2.1.4.3. API Gateway

Un API Gateway es un componente arquitectónico que actúa como punto único de entrada para las solicitudes dirigidas a una arquitectura basada en servicios o microservicios.

Su función principal consiste en abstraer la complejidad interna del sistema, proporcionando capacidades de enrutamiento, autenticación, autorización, monitoreo, balanceo de carga y control de tráfico.

Según Richardson (2018), la implementación de un API Gateway permite centralizar políticas transversales y reducir el acoplamiento entre consumidores y servicios internos.

En la arquitectura propuesta se emplea Kong como API Gateway, permitiendo exponer servicios de manera segura y controlada.

2.1.4.4. Gestión de Identidades y Accesos

La gestión de identidades y accesos (Identity and Access Management, IAM) comprende el conjunto de procesos y tecnologías destinados a autenticar usuarios y controlar el acceso a recursos dentro de un sistema de información.

Las arquitecturas modernas implementan mecanismos basados en estándares como OAuth 2.0 y OpenID Connect, los cuales permiten separar la autenticación de las aplicaciones de negocio y centralizar la administración de usuarios, roles y permisos.

La adopción de plataformas IAM mejora significativamente la seguridad, facilita la implementación de inicio de sesión único (Single Sign-On, SSO) y simplifica la gestión de credenciales en entornos distribuidos.

En esta investigación se utiliza Keycloak como proveedor centralizado de identidad y acceso.

2.1.4.5. Gestión de Secretos

La gestión de secretos consiste en administrar de manera segura información sensible utilizada por aplicaciones y servicios, como contraseñas, claves criptográficas, certificados digitales y tokens de acceso.

En arquitecturas tradicionales es común encontrar credenciales almacenadas directamente en archivos de configuración o incluso dentro del código fuente, lo cual incrementa significativamente los riesgos de seguridad.

Herramientas especializadas como HashiCorp Vault permiten almacenar, distribuir y auditar secretos de forma centralizada, incorporando mecanismos de cifrado, control de acceso y rotación automática de credenciales.

La implementación de una solución de gestión de secretos contribuye a fortalecer la seguridad y reducir la exposición de información crítica.

2.1.4.6. Mensajería Empresarial

La mensajería empresarial constituye un mecanismo de comunicación asíncrona que permite el intercambio de información entre aplicaciones distribuidas mediante mensajes.

Este enfoque favorece el desacoplamiento entre sistemas, mejora la resiliencia y facilita la integración de componentes heterogéneos dentro de una organización.

RabbitMQ es una plataforma de mensajería basada en el protocolo Advanced Message Queuing Protocol (AMQP), ampliamente utilizada para implementar arquitecturas orientadas a eventos. Sus capacidades incluyen enrutamiento de mensajes, persistencia, confirmación de entrega y tolerancia a fallos.

La utilización de mensajería empresarial permite construir sistemas más escalables y flexibles frente a cambios futuros.

2.1.4.7. Caché Distribuido

El caché distribuido es una técnica utilizada para almacenar temporalmente información de acceso frecuente con el fin de reducir la carga sobre los sistemas de persistencia y mejorar los tiempos de respuesta.

Redis es una base de datos en memoria que permite almacenar estructuras de datos de alta velocidad y es ampliamente utilizada como mecanismo de caché en arquitecturas distribuidas.

La implementación de Redis contribuye a mejorar el rendimiento de las aplicaciones, reducir la latencia y optimizar el consumo de recursos computacionales.

Asimismo, Redis puede utilizarse para implementar mecanismos de sesión distribuida, almacenamiento temporal y procesamiento de eventos en tiempo real.

2.1.4.8. Observabilidad

La observabilidad es la capacidad de comprender el estado interno de un sistema a partir de la información generada durante su operación. Este concepto se fundamenta en tres pilares principales: métricas, registros y trazas.

En arquitecturas distribuidas, la observabilidad resulta fundamental para identificar problemas de rendimiento, detectar fallos y analizar el comportamiento de los servicios.

Prometheus es una plataforma especializada en la recolección y almacenamiento de métricas, mientras que Grafana proporciona mecanismos avanzados de visualización mediante paneles interactivos.

La combinación de estas herramientas permite implementar monitoreo en tiempo real, generar alertas y facilitar la toma de decisiones basada en información operativa confiable.

La observabilidad constituye un componente esencial para garantizar la disponibilidad, mantenibilidad y confiabilidad de arquitecturas basadas en microservicios.

2.1.5. DevSecOps e Infraestructura como Código

DevSecOps es una evolución de DevOps que integra prácticas de seguridad durante todo el ciclo de vida del software.

Este enfoque promueve la automatización de controles de seguridad desde el desarrollo hasta la operación de los sistemas.

Complementariamente, la Infraestructura como Código (IaC) permite gestionar recursos tecnológicos mediante archivos declarativos. Herramientas como Terraform facilitan la automatización de despliegues y contribuyen a la reproducibilidad de entornos tecnológicos complejos.

La combinación de DevSecOps e Infraestructura como Código constituye un elemento fundamental para la operación eficiente de arquitecturas cloud-native y cloud-agnostic.

2.2. Estado del Arte

La evolución de las arquitecturas empresariales ha estado impulsada por la necesidad de construir sistemas más flexibles, escalables y resilientes frente a entornos tecnológicos en constante cambio. En este contexto, las arquitecturas basadas en microservicios, los principios cloud-native y las estrategias cloud-agnostic han adquirido una importancia significativa dentro de la comunidad científica y empresarial. Diversas investigaciones han abordado estos enfoques con el propósito de mejorar atributos de calidad como la interoperabilidad, disponibilidad, seguridad y mantenibilidad.

2.2.1. Arquitecturas Basadas en Microservicios

Durante los últimos años, las arquitecturas de microservicios se han consolidado como una alternativa a las arquitecturas monolíticas tradicionales. Este estilo arquitectónico propone la construcción de aplicaciones mediante servicios independientes, altamente cohesionados y débilmente acoplados, donde cada componente implementa una funcionalidad específica del negocio y puede desarrollarse, desplegarse y escalarse de forma autónoma. Diversos estudios reportan mejoras significativas en atributos de calidad como escalabilidad, modularidad, mantenibilidad y capacidad de evolución de los sistemas distribuidos Newman (2021); ZargarAzad and Ashtiani (2023); Pontarolli et al. (2023).

Según Newman Newman (2021), la principal ventaja de los microservicios radica en la posibilidad de dividir aplicaciones complejas en componentes más pequeños e independientes, facilitando el desarrollo continuo, la implementación de cambios y la adopción de diferentes tecnologías según las necesidades de cada servicio. Este enfoque permite además que los equipos de desarrollo trabajen de manera autónoma, reduciendo dependencias organizacionales y acelerando los ciclos de entrega de software.

ZargarAzad y Ashtiani ZargarAzad and Ashtiani (2023) destacan que los microservicios se han convertido en un patrón arquitectónico ampliamente adoptado para aplicaciones cloud-native debido a su capacidad de escalamiento independiente y adaptación dinámica a variaciones en la carga de trabajo. Los autores proponen un mecanismo de autoescalamiento basado en aprendizaje por refuerzo que optimiza la asignación de recursos computacionales manteniendo niveles adecuados

de rendimiento y disponibilidad. Los resultados obtenidos evidencian que las arquitecturas basadas en microservicios permiten aprovechar de manera más eficiente los recursos de infraestructura en comparación con enfoques tradicionales.

De manera similar, Pontarolli et al. Pontarolli et al. (2023) analizaron la aplicación de arquitecturas orientadas a microservicios en entornos industriales asociados a Industria 4.0. Los autores identificaron que la modularización de servicios favorece la interoperabilidad entre sistemas heterogéneos, facilita la descentralización de procesos y mejora la flexibilidad operativa de las organizaciones. Asimismo, concluyen que los microservicios permiten responder de manera más efectiva a los requerimientos de integración presentes en ecosistemas tecnológicos complejos.

La evolución de los sistemas basados en microservicios también ha sido objeto de estudio en investigaciones recientes. Ayas, Leitner y Hebig Ayas et al. (2023) realizaron un estudio empírico sobre procesos de migración desde arquitecturas monolíticas hacia microservicios, identificando que esta transición implica cambios tanto técnicos como organizacionales. Los autores concluyen que la adopción de microservicios contribuye significativamente a mejorar la capacidad de evolución de los sistemas, aunque introduce nuevos desafíos relacionados con la gestión de infraestructura, monitoreo y coordinación entre servicios.

Por otra parte, Taibi et al. Taibi et al. (2023) analizaron la evolución de repositorios de software basados en microservicios y encontraron que gran parte de los cambios realizados durante el ciclo de vida de estos sistemas están relacionados con actividades de operación, automatización y mantenimiento. Este hallazgo evidencia que la implementación de microservicios requiere la adopción de herramientas complementarias orientadas a la observabilidad, automatización de despliegues y gestión de configuraciones.

A pesar de sus múltiples beneficios, la literatura también identifica diversos desafíos asociados a este estilo arquitectónico. Entre ellos se destacan la complejidad operativa, la necesidad de mecanismos avanzados de observabilidad, la gestión de transacciones distribuidas, la seguridad entre servicios y el incremento de la comunicación a través de la red Bogner et al. (2022); Nasab et al. (2023). Estos aspectos demandan la incorporación de componentes especializados que permitan garantizar atributos de calidad como disponibilidad, interoperabilidad, seguridad y mantenibilidad.

En este contexto, las arquitecturas de microservicios suelen complementarse con tecnologías como API Gateway, plataformas de mensajería empresarial, sistemas de gestión de identidades, herramientas de observabilidad y mecanismos de administración de secretos. La integración de estos componentes permite construir soluciones empresariales altamente escalables, resilientes y alineadas con los principios cloud-native y cloud-agnostic.

En síntesis, la evidencia científica demuestra que las arquitecturas basadas en microservicios representan una estrategia efectiva para construir sistemas distribuidos modernos capaces de responder a los desafíos de escalabilidad, flexibilidad e interoperabilidad presentes en las organizaciones actuales. Estos hallazgos justifican la utilización de una arquitectura basada en microservicios como fundamento de la propuesta desarrollada en la presente investigación.

2.2.2. Arquitecturas Cloud-Native

El concepto de arquitectura cloud-native surge como respuesta a la necesidad de aprovechar las capacidades de elasticidad, automatización y resiliencia ofrecidas por los entornos de computación en la nube. Este enfoque arquitectónico promueve el desarrollo y despliegue de aplicaciones diseñadas específicamente para ejecutarse en infraestructuras distribuidas, utilizando tecnologías como contenedores, orquestadores, microservicios, integración continua y automatización operativa Cloud Native Computing Foundation (2024); Ullah et al. (2023).

Según la Cloud Native Computing Foundation (CNCF), una arquitectura cloud-native permite construir y ejecutar aplicaciones escalables en entornos dinámicos tales como nubes públicas, privadas e híbridas. Estas arquitecturas se fundamentan en el uso de contenedores, mallas de servicios, microservicios, infraestructura inmutable y plataformas declarativas, facilitando la automatización de tareas operativas y mejorando la resiliencia de los sistemas Cloud Native Computing Foundation (2024).

Uno de los principales habilitadores de las arquitecturas cloud-native es la contenerización de aplicaciones. Los contenedores permiten empaquetar aplicaciones junto con sus dependencias y configuraciones, garantizando consistencia entre ambientes de desarrollo, pruebas y producción. Esta característica reduce problemas asociados a diferencias de configuración y facilita la portabilidad de las aplicaciones entre distintos entornos de ejecución Soldani et al. (2023).

Complementariamente, Kubernetes se ha consolidado como el estándar de facto para la orquestación de contenedores dentro del ecosistema cloud-native. Diversos estudios destacan que Kubernetes proporciona mecanismos avanzados para la gestión automática del ciclo de vida de las aplicaciones, incluyendo despliegue continuo, escalamiento horizontal, recuperación automática ante fallos, balanceo de carga y administración declarativa de recursos Ullah et al. (2023); Kosinska et al. (2023).

Las investigaciones recientes evidencian que las arquitecturas cloud-native contribuyen significativamente a mejorar atributos de calidad como disponibilidad, escalabilidad y mantenibilidad. Ullah et al. Ullah et al. (2023) señalan que la automatización proporcionada por plataformas de orquestación modernas permite optimizar la utilización de recursos computacionales y reducir tiempos de inactividad. Asimismo, los autores destacan que estas capacidades resultan fundamentales para soportar aplicaciones empresariales que experimentan variaciones dinámicas en la demanda.

Otro aspecto ampliamente estudiado corresponde a la resiliencia operacional de los sistemas cloud-native. Mediante la utilización de mecanismos de recuperación automática, monitoreo continuo y despliegues automatizados, estas arquitecturas permiten minimizar el impacto de fallos y garantizar niveles elevados de disponibilidad. Diversas investigaciones demuestran que la incorporación de prácticas DevOps y DevSecOps dentro de ecosistemas cloud-native contribuye a reducir riesgos operativos y mejorar la calidad del software entregado Di Francesco et al. (2024); Soldani et al. (2023).

La observabilidad constituye igualmente un componente esencial dentro de las arquitecturas cloud-native. Debido a la naturaleza distribuida de los microservicios, resulta necesario implementar mecanismos capaces de recopilar métricas, registros y trazas distribuidas para comprender el

comportamiento interno de las aplicaciones. Kosińska et al. Kosinska et al. (2023) destacan que herramientas como Prometheus, Grafana y OpenTelemetry se han convertido en elementos fundamentales para garantizar la visibilidad operativa de sistemas distribuidos complejos.

No obstante, la literatura también identifica diversos desafíos asociados a la adopción de arquitecturas cloud-native. Entre ellos se encuentran la complejidad de administración de entornos distribuidos, la necesidad de personal especializado, la gestión de seguridad en múltiples capas y la dependencia de herramientas de automatización avanzadas Soldani et al. (2023); Ullah et al. (2023). Estos factores requieren una adecuada planificación arquitectónica y la implementación de mecanismos que permitan controlar la complejidad inherente a este tipo de soluciones.

En síntesis, las arquitecturas cloud-native representan un paradigma fundamental para el desarrollo de sistemas empresariales modernos. La combinación de microservicios, contenedores, orquestación automatizada, observabilidad y prácticas DevSecOps permite construir plataformas altamente escalables, resilientes y adaptables a entornos tecnológicos cambiantes. Estas características justifican la adopción de principios cloud-native como uno de los pilares fundamentales de la arquitectura propuesta en la presente investigación.

2.2.3. Arquitecturas Cloud-Agnostic

La creciente adopción de servicios de computación en la nube ha permitido a las organizaciones incrementar la escalabilidad, disponibilidad y flexibilidad de sus sistemas de información. Sin embargo, la dependencia de servicios propietarios ofrecidos por proveedores específicos de nube puede generar una problemática conocida como *vendor lock-in*, situación en la cual la migración de aplicaciones hacia otros entornos tecnológicos implica elevados costos técnicos, económicos y operativos Petcu (2013); Toosi et al. (2014).

Como respuesta a este desafío surge el enfoque *Cloud-Agnostic*, el cual promueve el diseño de arquitecturas capaces de operar en diferentes plataformas de nube sin requerir modificaciones significativas en su estructura funcional o tecnológica. El objetivo principal de este paradigma consiste en reducir la dependencia hacia proveedores específicos mediante el uso de estándares abiertos, tecnologías portables y mecanismos de abstracción de infraestructura Toosi et al. (2014).

A diferencia de las arquitecturas cloud-native, cuyo propósito principal es maximizar el aprovechamiento de los servicios ofrecidos por una plataforma cloud, las arquitecturas cloud-agnostic priorizan la portabilidad y la interoperabilidad entre diferentes entornos de ejecución. Esto permite que una misma solución pueda desplegarse en proveedores como Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP) o infraestructuras *on-premise* sin requerir cambios sustanciales en la aplicación Pahl (2015).

Diversas investigaciones coinciden en que la contenerización constituye uno de los mecanismos fundamentales para alcanzar la portabilidad entre plataformas cloud. Docker permite encapsular aplicaciones y dependencias dentro de unidades de ejecución estandarizadas, eliminando diferencias entre entornos de infraestructura y facilitando el despliegue consistente en distintos proveedores de nube Pahl (2015); Merkel (2014).

Complementariamente, Kubernetes se ha consolidado como una de las tecnologías más rele-

vantes para la construcción de arquitecturas cloud-agnostic. Su modelo declarativo de gestión de recursos permite abstraer gran parte de las particularidades asociadas a la infraestructura subyacente, proporcionando una capa uniforme de orquestación independiente del proveedor cloud utilizado Burns et al. (2016); Ullah et al. (2023). Gracias a esta capacidad, las aplicaciones contenerizadas pueden ejecutarse en múltiples plataformas manteniendo comportamientos consistentes en términos de despliegue, escalabilidad y administración operativa.

Otro componente ampliamente utilizado dentro de estrategias cloud-agnostic corresponde a la Infraestructura como Código (IaC). Herramientas como Terraform permiten definir y aprovisionar recursos mediante configuraciones declarativas independientes del proveedor de nube. Este enfoque facilita la automatización de despliegues y contribuye a la reproducibilidad de ambientes tecnológicos en diferentes plataformas Morris (2020).

Asimismo, diversos autores destacan la importancia de evitar dependencias directas con servicios propietarios cuando se busca mantener la portabilidad arquitectónica. En este contexto, la adopción de soluciones de código abierto para funciones transversales como autenticación, mensajería, gestión de secretos y observabilidad representa una estrategia efectiva para minimizar el riesgo de dependencia tecnológica Soldani et al. (2023). Tecnologías como Keycloak para gestión de identidades, HashiCorp Vault para administración de secretos, RabbitMQ para mensajería empresarial y Prometheus para monitoreo constituyen ejemplos de herramientas que pueden ejecutarse en diferentes entornos cloud sin restricciones asociadas a un proveedor específico.

No obstante, la literatura también señala que la adopción de arquitecturas cloud-agnostic implica ciertos compromisos arquitectónicos. Al evitar servicios propietarios avanzados, las organizaciones pueden renunciar a funcionalidades optimizadas ofrecidas por determinados proveedores de nube. Como consecuencia, algunas soluciones cloud-agnostic presentan mayores niveles de complejidad operativa o requieren esfuerzos adicionales de administración y mantenimiento Toosi et al. (2014); Pahl (2015).

A pesar de estas limitaciones, múltiples investigaciones concluyen que los beneficios asociados a la reducción del *vendor lock-in*, la flexibilidad tecnológica y la capacidad de migración justifican la adopción de estrategias cloud-agnostic en entornos empresariales que demandan independencia tecnológica y sostenibilidad a largo plazo Petcu (2013); Burns et al. (2016).

En consecuencia, la presente investigación adopta principios cloud-agnostic mediante la utilización de tecnologías abiertas y ampliamente adoptadas en la industria, tales como Kubernetes, Docker, Terraform, Kong, Keycloak, HashiCorp Vault, RabbitMQ, Redis, Prometheus y Grafana. Esta aproximación permite que la arquitectura propuesta pueda desplegarse sobre diferentes proveedores cloud o infraestructuras locales, reduciendo significativamente la dependencia tecnológica y favoreciendo la portabilidad de la solución.

2.2.4. Seguridad en Arquitecturas Distribuidas

La seguridad constituye uno de los principales desafíos en las arquitecturas distribuidas modernas debido a la naturaleza descentralizada de sus componentes y a la multiplicidad de interacciones que se establecen entre servicios, aplicaciones y usuarios. A diferencia de las arquitecturas monolíti-

cas tradicionales, donde los mecanismos de protección suelen concentrarse en un único punto de acceso, las arquitecturas distribuidas requieren estrategias de seguridad capaces de proteger múltiples componentes desplegados en diferentes entornos de ejecución Nasab et al. (2023); Billawa et al. (2022).

La adopción de arquitecturas basadas en microservicios ha incrementado significativamente la superficie de ataque de los sistemas de información. Cada servicio expone interfaces de comunicación que pueden convertirse en potenciales vectores de vulnerabilidad si no se implementan mecanismos adecuados de autenticación, autorización y protección de datos. Como consecuencia, la seguridad debe abordarse como un aspecto transversal que involucre tanto la infraestructura como los componentes de aplicación Nasab et al. (2023).

Nasab et al. Nasab et al. (2023) realizaron un estudio empírico sobre prácticas de seguridad en sistemas basados en microservicios, identificando veintiocho prácticas utilizadas por organizaciones de diferentes sectores. Los resultados evidencian que la autenticación robusta, la autorización basada en roles, el cifrado de comunicaciones y la gestión segura de credenciales constituyen los mecanismos más relevantes para proteger entornos distribuidos.

Dentro de este contexto, los sistemas de Gestión de Identidades y Accesos (Identity and Access Management, IAM) desempeñan un papel fundamental al centralizar los procesos de autenticación y autorización. Estas soluciones permiten garantizar que únicamente usuarios, aplicaciones y servicios autorizados puedan acceder a recursos protegidos. Asimismo, facilitan la implementación de mecanismos de control de acceso basados en roles y políticas de seguridad definidas por la organización Venckauskas et al. (2023).

La literatura especializada destaca el uso de protocolos estandarizados como OAuth 2.0 y OpenID Connect para la protección de aplicaciones distribuidas. OAuth 2.0 proporciona un mecanismo seguro para la delegación de acceso a recursos protegidos, mientras que OpenID Connect extiende sus capacidades incorporando funcionalidades de autenticación e identidad federada Hardt (2012); Sakimura et al. (2014). Estos estándares se han convertido en elementos fundamentales para la implementación de arquitecturas empresariales seguras y escalables.

Otro aspecto crítico corresponde a la protección de secretos y credenciales utilizados por aplicaciones y servicios. En arquitecturas distribuidas, resulta común que múltiples componentes requieran acceso a contraseñas, certificados digitales, claves criptográficas y credenciales de bases de datos. El almacenamiento inseguro de esta información representa uno de los riesgos más frecuentes identificados en entornos cloud y microservicios HashiCorp (2024).

Para mitigar este problema, diversas investigaciones recomiendan la utilización de plataformas especializadas para la gestión centralizada de secretos. Estas soluciones permiten almacenar, rotar y distribuir credenciales de manera segura, reduciendo el riesgo de exposición accidental de información sensible y fortaleciendo la postura de seguridad de la organización HashiCorp (2024).

La protección de las comunicaciones entre servicios constituye igualmente un requisito fundamental dentro de arquitecturas distribuidas. Las mejores prácticas actuales recomiendan la utilización de protocolos seguros como Transport Layer Security (TLS) para garantizar la confidencialidad e integridad de los datos intercambiados entre componentes Rescorla (2018). De igual forma, la implementación de autenticación mutua entre servicios contribuye a prevenir accesos no autorizados

dentro de ecosistemas distribuidos.

En los últimos años, el paradigma Zero Trust ha adquirido una relevancia significativa como modelo de referencia para la protección de sistemas distribuidos. Este enfoque se fundamenta en el principio de que ningún usuario, dispositivo o servicio debe considerarse confiable por defecto, independientemente de su ubicación dentro o fuera de la red corporativa. En consecuencia, cada solicitud de acceso debe ser autenticada, autorizada y validada continuamente antes de permitir el acceso a recursos protegidos Rose et al. (2020).

Asimismo, diversos autores coinciden en que la seguridad debe integrarse desde las primeras etapas del ciclo de vida del software mediante prácticas DevSecOps. Este enfoque promueve la incorporación de controles de seguridad dentro de los procesos de desarrollo, integración continua y despliegue continuo, permitiendo identificar vulnerabilidades de manera temprana y reducir riesgos operativos Myrbakken and Colomo-Palacios (2022).

En síntesis, la evidencia científica demuestra que la seguridad constituye un atributo de calidad fundamental dentro de las arquitecturas distribuidas. La combinación de mecanismos de autenticación centralizada, autorización basada en estándares abiertos, gestión segura de secretos, cifrado de comunicaciones y principios Zero Trust permite construir plataformas resilientes frente a amenazas modernas. Estos elementos representan pilares fundamentales dentro de la arquitectura propuesta en la presente investigación, particularmente mediante la integración de Keycloak para la gestión de identidades y accesos, HashiCorp Vault para la administración de secretos y Kong como punto centralizado para la aplicación de políticas de seguridad.

2.2.5. Integración mediante API Gateway y Mensajería

La integración de aplicaciones constituye uno de los principales desafíos dentro de las arquitecturas empresariales modernas. La creciente adopción de microservicios, sistemas distribuidos y plataformas híbridas ha incrementado la necesidad de implementar mecanismos que permitan la comunicación eficiente, segura y escalable entre múltiples componentes tecnológicos. En este contexto, los patrones basados en API Gateway y sistemas de mensajería empresarial se han consolidado como soluciones ampliamente utilizadas para facilitar la interoperabilidad entre aplicaciones heterogéneas Richards (2020); Hohpe and Woolf (2004).

Dentro de las arquitecturas basadas en microservicios, el patrón API Gateway actúa como un punto único de entrada para las solicitudes provenientes de clientes externos. Su principal objetivo consiste en abstraer la complejidad interna del sistema, proporcionando una interfaz unificada para el acceso a múltiples servicios distribuidos Richardson (2018). Este enfoque permite centralizar funcionalidades transversales como autenticación, autorización, control de tráfico, transformación de mensajes, monitoreo y registro de eventos.

Según Richardson Richardson (2018), la utilización de un API Gateway reduce el acoplamiento entre consumidores y proveedores de servicios, facilitando la evolución independiente de los microservicios y simplificando la gestión de cambios dentro de la arquitectura. Asimismo, este patrón contribuye significativamente a mejorar atributos de calidad como mantenibilidad, interoperabilidad y seguridad.

Diversos estudios destacan que la implementación de un API Gateway resulta especialmente beneficiosa en arquitecturas cloud-native debido a la gran cantidad de servicios que interactúan dentro del ecosistema. En estos entornos, herramientas como Kong, Apigee y Ambassador permiten centralizar políticas de seguridad, aplicar mecanismos de limitación de tráfico (*rate limiting*), balancear carga entre servicios y gestionar certificados digitales de manera uniforme Soldani et al. (2023).

No obstante, la comunicación síncrona basada exclusivamente en APIs puede generar dependencias temporales entre servicios, afectando la resiliencia y escalabilidad de la arquitectura. Para mitigar este problema, numerosas investigaciones recomiendan complementar la integración síncrona con mecanismos de comunicación asíncrona basados en sistemas de mensajería empresarial Hohpe and Woolf (2004).

La mensajería empresarial constituye un paradigma de integración orientado al intercambio de información mediante mensajes desacoplados. Este enfoque permite que aplicaciones y servicios intercambien información sin requerir disponibilidad simultánea de los participantes, favoreciendo la tolerancia a fallos y la escalabilidad del sistema Hohpe and Woolf (2004).

Hohpe y Woolf Hohpe and Woolf (2004) identifican múltiples patrones de integración empresarial que permiten resolver problemas comunes de comunicación entre aplicaciones distribuidas. Entre ellos destacan las colas de mensajes, publicación/suscripción, enrutamiento basado en contenido y procesamiento orientado a eventos, los cuales continúan siendo ampliamente utilizados en arquitecturas modernas.

La adopción de plataformas de mensajería empresarial permite implementar arquitecturas orientadas a eventos (*Event-Driven Architecture*), donde los componentes reaccionan a eventos generados por otros servicios. Este modelo favorece el desacoplamiento funcional entre aplicaciones y facilita la construcción de sistemas altamente escalables y resilientes Kreps (2011).

Entre las tecnologías más utilizadas para implementar este enfoque se encuentran RabbitMQ, Apache Kafka y ActiveMQ. Particularmente, RabbitMQ se ha consolidado como una de las plataformas de mensajería más utilizadas debido a su soporte para protocolos estandarizados como Advanced Message Queuing Protocol (AMQP), su capacidad de enrutamiento flexible y su facilidad de integración con arquitecturas basadas en microservicios VMware (2024).

Diversas investigaciones evidencian que la combinación de mecanismos de integración síncrona y asíncrona permite mejorar significativamente la resiliencia operacional de los sistemas distribuidos. Mientras el API Gateway facilita la comunicación directa entre clientes y servicios, las plataformas de mensajería permiten desacoplar procesos de negocio, reducir dependencias temporales y optimizar la distribución de cargas de trabajo Soldani et al. (2023); Richardson (2018).

Asimismo, la integración basada en eventos contribuye a mejorar atributos de calidad como disponibilidad, escalabilidad e interoperabilidad. Al desacoplar productores y consumidores de información, es posible incorporar nuevos servicios o modificar componentes existentes sin afectar el funcionamiento global del sistema, favoreciendo la evolución continua de la arquitectura Hohpe and Woolf (2004); Kreps (2011).

En síntesis, la evidencia científica demuestra que la combinación de patrones de integración basados en API Gateway y sistemas de mensajería empresarial constituye una estrategia efectiva

para soportar arquitecturas distribuidas modernas. La integración de Kong como API Gateway y RabbitMQ como plataforma de mensajería dentro de la arquitectura propuesta permite fortalecer la interoperabilidad entre componentes, mejorar la resiliencia operacional y facilitar la evolución futura de la solución tecnológica desarrollada en la presente investigación.

2.2.6. Observabilidad en Sistemas Distribuidos

La creciente adopción de arquitecturas distribuidas y basadas en microservicios ha incrementado significativamente la complejidad operativa de los sistemas de información modernos. A diferencia de las arquitecturas monolíticas tradicionales, donde el comportamiento de una aplicación puede analizarse desde un único punto de ejecución, los sistemas distribuidos involucran múltiples servicios que interactúan entre sí a través de redes, protocolos y mecanismos de integración diversos. Esta complejidad ha impulsado el desarrollo de estrategias de observabilidad orientadas a proporcionar visibilidad integral sobre el estado y comportamiento de las aplicaciones Kosinska et al. (2023); Di Francesco et al. (2024).

La observabilidad puede definirse como la capacidad de comprender el estado interno de un sistema a partir de la información generada por sus componentes. Este concepto tiene sus fundamentos en la teoría de control, donde un sistema es considerado observable cuando es posible inferir su estado interno mediante el análisis de sus salidas. En el contexto de la ingeniería de software, la observabilidad permite identificar fallos, analizar comportamientos anómalos y diagnosticar problemas operativos de manera eficiente Kosinska et al. (2023).

Diversos estudios coinciden en que la observabilidad moderna se fundamenta en tres pilares principales: métricas, registros y trazas distribuidas. Las métricas proporcionan información cuantitativa sobre el comportamiento de los servicios, como consumo de recursos, tiempos de respuesta y tasas de error. Los registros (*logs*) permiten documentar eventos relevantes ocurridos durante la ejecución de las aplicaciones, mientras que las trazas distribuidas facilitan el seguimiento de solicitudes a través de múltiples componentes dentro de una arquitectura distribuida Ramaswamy (2024); Kosinska et al. (2023).

Kosińska et al. Kosinska et al. (2023) realizaron una revisión del estado del arte sobre observabilidad en aplicaciones cloud-native, concluyendo que la combinación de métricas, registros y trazas constituye el enfoque más efectivo para comprender el comportamiento de sistemas altamente distribuidos. Los autores destacan que la observabilidad se ha convertido en un requisito indispensable para garantizar la operación confiable de arquitecturas basadas en microservicios.

Por otra parte, Di Francesco et al. Di Francesco et al. (2024) señalan que la adopción de estrategias de observabilidad contribuye significativamente a reducir los tiempos de detección y resolución de incidentes. Los resultados de su estudio evidencian que las organizaciones que implementan mecanismos avanzados de monitoreo logran mejorar la disponibilidad de los servicios y optimizar las actividades de mantenimiento correctivo y preventivo.

Dentro del ecosistema cloud-native, Prometheus se ha consolidado como una de las herramientas más utilizadas para la recopilación y almacenamiento de métricas. Su modelo de adquisición basado en consultas periódicas (*pull-based monitoring*) permite obtener información detallada sobre el

comportamiento de aplicaciones, contenedores e infraestructura. Adicionalmente, proporciona mecanismos para la definición de alertas que facilitan la detección temprana de incidentes operativos Prometheus Authors (2024).

Complementariamente, Grafana se ha convertido en una plataforma ampliamente adoptada para la visualización y análisis de datos de monitoreo. Esta herramienta permite construir paneles interactivos que integran métricas provenientes de múltiples fuentes, facilitando la supervisión centralizada de entornos distribuidos y proporcionando información relevante para la toma de decisiones operativas Grafana Labs (2024).

La literatura reciente también destaca la importancia de las trazas distribuidas para comprender el flujo de ejecución de solicitudes dentro de arquitecturas basadas en microservicios. Tecnologías como OpenTelemetry permiten capturar información detallada sobre las interacciones entre servicios, facilitando la identificación de cuellos de botella, errores de comunicación y problemas de rendimiento OpenTelemetry Community (2024). Estas capacidades resultan especialmente relevantes en entornos donde una única transacción puede involucrar múltiples componentes distribuidos.

Asimismo, diversos estudios indican que la observabilidad constituye un habilitador fundamental para prácticas DevOps y DevSecOps. La disponibilidad de información en tiempo real sobre el estado de las aplicaciones facilita la automatización de procesos operativos, mejora la capacidad de respuesta ante incidentes y fortalece la retroalimentación continua dentro del ciclo de vida del software Di Francesco et al. (2024); Ramaswamy (2024).

No obstante, la implementación de estrategias de observabilidad también presenta desafíos importantes. Entre ellos se encuentran el volumen masivo de datos generados por sistemas distribuidos, la correlación de información proveniente de múltiples fuentes y la necesidad de definir métricas relevantes para los objetivos de negocio. Estos factores requieren una adecuada planificación de la arquitectura de monitoreo y una selección apropiada de herramientas tecnológicas Kosinska et al. (2023).

En síntesis, la evidencia científica demuestra que la observabilidad constituye un componente esencial para garantizar la operación eficiente de arquitecturas distribuidas modernas. La integración de mecanismos de monitoreo, análisis de registros y trazabilidad distribuida permite mejorar la disponibilidad, confiabilidad y mantenibilidad de los sistemas. En consecuencia, la arquitectura propuesta en esta investigación incorpora Prometheus y Grafana como componentes fundamentales para la recopilación, análisis y visualización de información operativa, contribuyendo al cumplimiento de los atributos de calidad definidos para la solución.

2.2.7. Evaluación Arquitectónica mediante ATAM

La arquitectura de software constituye uno de los principales factores que determinan la calidad de un sistema de información. Las decisiones arquitectónicas adoptadas durante las etapas iniciales del desarrollo tienen un impacto directo sobre atributos como rendimiento, seguridad, disponibilidad, interoperabilidad, mantenibilidad y escalabilidad. Por esta razón, resulta fundamental disponer de mecanismos que permitan evaluar la arquitectura antes de su implementación definitiva, identificando riesgos potenciales y validando el cumplimiento de los requisitos de calidad

establecidos para la solución Bass et al. (2021); Clements et al. (2010).

Dentro de los métodos de evaluación arquitectónica existentes, el *Architecture Tradeoff Analysis Method* (ATAM) se ha consolidado como una de las técnicas más utilizadas para analizar arquitecturas de software desde la perspectiva de los atributos de calidad. Este método fue desarrollado por el Software Engineering Institute (SEI) de Carnegie Mellon University con el propósito de identificar riesgos arquitectónicos, puntos de sensibilidad, compensaciones entre atributos de calidad y decisiones críticas de diseño Kazman et al. (2000).

ATAM se fundamenta en el principio de que las decisiones arquitectónicas generan impactos positivos y negativos sobre diferentes atributos de calidad. En consecuencia, el método busca identificar los compromisos (*trade-offs*) existentes entre dichos atributos, permitiendo comprender cómo una decisión orientada a mejorar un aspecto específico puede afectar otros elementos del sistema Bass et al. (2021).

Según Kazman, Klein y Clements Kazman et al. (2000), el proceso de evaluación mediante ATAM se desarrolla a través de una serie de actividades estructuradas que incluyen la presentación de objetivos de negocio, la descripción de la arquitectura propuesta, la identificación de atributos de calidad relevantes, la construcción de escenarios de calidad y el análisis de riesgos arquitectónicos. Este enfoque permite involucrar tanto a arquitectos de software como a interesados del negocio en el proceso de evaluación.

Uno de los conceptos fundamentales dentro de ATAM corresponde a los escenarios de calidad. Un escenario de calidad describe una situación específica en la que un atributo de calidad es puesto a prueba bajo determinadas condiciones operativas. Estos escenarios permiten evaluar de manera objetiva el comportamiento esperado de la arquitectura frente a eventos relacionados con seguridad, rendimiento, disponibilidad, interoperabilidad o mantenibilidad Bass et al. (2021).

La literatura especializada identifica seis elementos principales que conforman un escenario de calidad: fuente del estímulo, estímulo, entorno, artefacto afectado, respuesta esperada y medida de respuesta. Esta estructura facilita la definición de criterios de evaluación medibles y alineados con los objetivos estratégicos de la organización Clements et al. (2010).

Otro concepto relevante dentro de ATAM corresponde a los puntos de sensibilidad (*sensitivity points*), los cuales representan decisiones arquitectónicas cuya modificación puede producir variaciones significativas sobre uno o varios atributos de calidad. Asimismo, el método permite identificar puntos de compensación (*tradeoff points*), donde una decisión arquitectónica beneficia un atributo de calidad mientras afecta negativamente otro Kazman et al. (2000).

Diversas investigaciones han demostrado la efectividad de ATAM para evaluar arquitecturas complejas basadas en microservicios, computación en la nube y sistemas distribuidos. Su aplicación permite detectar riesgos tempranamente, reducir incertidumbres asociadas al diseño arquitectónico y facilitar la toma de decisiones antes de realizar inversiones significativas en infraestructura o desarrollo Capilla et al. (2025).

En arquitecturas modernas basadas en microservicios, la utilización de ATAM resulta particularmente relevante debido a la existencia de múltiples componentes especializados que interactúan entre sí. Elementos como API Gateway, sistemas de gestión de identidades, plataformas de mensajería, mecanismos de observabilidad y servicios de gestión de secretos generan dependencias y

compromisos arquitectónicos que deben ser evaluados de manera sistemática para garantizar el cumplimiento de los atributos de calidad definidos para el sistema Nasab et al. (2023); Soldani et al. (2023).

La presente investigación adopta ATAM como método de evaluación arquitectónica debido a su capacidad para analizar de forma estructurada los atributos de calidad priorizados dentro de la solución propuesta. Particularmente, la evaluación se centra en atributos como seguridad, interoperabilidad, disponibilidad, escalabilidad, mantenibilidad y observabilidad, los cuales constituyen elementos fundamentales para una arquitectura de mediación basada en microservicios y orientada a entornos cloud-agnostic.

Mediante la aplicación de ATAM será posible identificar riesgos arquitectónicos, analizar las decisiones de diseño adoptadas y determinar el impacto de los componentes seleccionados —Kubernetes, Kong, Keycloak, HashiCorp Vault, RabbitMQ, Redis, Prometheus y Grafana— sobre los atributos de calidad establecidos. De esta manera, la evaluación proporcionará evidencia objetiva acerca de la capacidad de la arquitectura propuesta para satisfacer los requerimientos funcionales y no funcionales definidos en la investigación.

En síntesis, ATAM constituye una metodología ampliamente validada para la evaluación temprana de arquitecturas de software. Su enfoque basado en escenarios de calidad y análisis de compromisos arquitectónicos permite valorar de manera sistemática el impacto de las decisiones de diseño sobre los atributos de calidad del sistema. Por esta razón, el método representa una herramienta adecuada para evaluar la arquitectura propuesta en esta investigación y validar su contribución al cumplimiento de los objetivos planteados.

2.2.8. Análisis Comparativo de los Trabajos Relacionados

La revisión de la literatura científica realizada evidencia que las arquitecturas basadas en microservicios, los enfoques cloud-native y las estrategias cloud-agnostic constituyen líneas de investigación activas dentro del ámbito de la arquitectura de software. Diversos autores han propuesto mecanismos orientados a mejorar atributos de calidad como escalabilidad, disponibilidad, interoperabilidad, mantenibilidad y seguridad mediante la utilización de tecnologías modernas de infraestructura y desarrollo.

En el ámbito de los microservicios, estudios como los desarrollados por ZargarAzad y Ashtiani ZargarAzad and Ashtiani (2023), Pontarolli et al. Pontarolli et al. (2023), Ayas et al. Ayas et al. (2023) y Taibi et al. Taibi et al. (2023) destacan las ventajas de este estilo arquitectónico para construir sistemas distribuidos escalables y flexibles. Sin embargo, dichas investigaciones se enfocan principalmente en aspectos relacionados con la modularización, el escalamiento y la evolución de los sistemas, sin abordar de manera integral componentes complementarios asociados a seguridad, observabilidad e interoperabilidad.

Por otra parte, las investigaciones relacionadas con arquitecturas cloud-native evidencian la relevancia de tecnologías como contenedores, Kubernetes y prácticas DevOps para la construcción de plataformas resilientes y altamente disponibles Ullah et al. (2023); Soldani et al. (2023). No obstante, gran parte de estos trabajos se concentran en los beneficios operativos derivados de la

nube y no profundizan en mecanismos orientados a reducir la dependencia tecnológica respecto a proveedores específicos de infraestructura.

En este sentido, los estudios relacionados con arquitecturas cloud-agnostic resaltan la importancia de emplear estándares abiertos y tecnologías portables para minimizar el fenómeno conocido como *vendor lock-in* Petcu (2013); Toosi et al. (2014); Pahl (2015). Aunque estas investigaciones proporcionan lineamientos para mejorar la portabilidad entre plataformas cloud, generalmente analizan componentes específicos de infraestructura sin proponer arquitecturas empresariales completas que integren mecanismos avanzados de seguridad, integración y observabilidad.

Respecto a la seguridad en arquitecturas distribuidas, diversos autores destacan la importancia de implementar mecanismos de autenticación, autorización y gestión de secretos mediante estándares como OAuth 2.0, OpenID Connect y plataformas especializadas de administración de credenciales Nasab et al. (2023); Venckauskas et al. (2023). Sin embargo, la mayoría de los trabajos analizados se centran en dominios específicos de seguridad sin considerar su integración dentro de una arquitectura empresarial orientada a microservicios.

De manera similar, las investigaciones sobre observabilidad resaltan el papel de herramientas como Prometheus, Grafana y OpenTelemetry para mejorar la visibilidad operativa de sistemas distribuidos Kosinska et al. (2023); Di Francesco et al. (2024). A pesar de ello, dichos estudios se enfocan principalmente en estrategias de monitoreo y análisis de rendimiento, sin establecer relaciones directas con otros componentes arquitectónicos como API Gateway, sistemas IAM o plataformas de mensajería.

Finalmente, la literatura relacionada con ATAM demuestra la efectividad de este método para evaluar atributos de calidad y analizar decisiones arquitectónicas en sistemas complejos Kazman et al. (2000); Bass et al. (2021). No obstante, se observa una limitada cantidad de investigaciones que integren simultáneamente arquitecturas cloud-agnostic, componentes de seguridad, mecanismos de observabilidad y plataformas de integración empresarial dentro de un mismo proceso de evaluación arquitectónica.

La Tabla 2.1 presenta una síntesis comparativa de los principales trabajos analizados durante la revisión bibliográfica.

Tabla 2.1: Comparación de trabajos relacionados

Trabajo	Año	Enfoque	Seguridad	Observabilidad	Cloud-Agnostic	ATAM
ZargarAzad y Ashtiani	2023	Autoescalamiento de microservicios	No	No	Parcial	No
Pontarolli et al.	2023	Microservicios para Industria 4.0	No	No	No	No
Ayas et al.	2023	Migración hacia microservicios	No	Parcial	No	No
Nasab et al.	2023	Seguridad en microservicios	Sí	No	No	No
Kosińska et al.	2023	Observabilidad cloud-native	No	Sí	No	No
Di Francesco et al.	2024	Monitoreo y observabilidad	No	Sí	No	No
Kazman et al.	2000	Evaluación arquitectónica ATAM	No	No	No	Sí
Propuesta de investigación	2026	Arquitectura empresarial cloud-agnostic basada en microservicios	Sí	Sí	Sí	Sí

A partir del análisis realizado se identifica una oportunidad de investigación relacionada con la construcción de una arquitectura empresarial que integre de manera conjunta mecanismos de seguridad, interoperabilidad, observabilidad y portabilidad tecnológica mediante tecnologías abiertas y ampliamente adoptadas en la industria. Asimismo, se evidencia la necesidad de evaluar este tipo de arquitecturas utilizando métodos formales que permitan analizar el impacto de las decisiones arquitectónicas sobre los atributos de calidad del sistema.

En consecuencia, la presente investigación propone una arquitectura basada en microservicios orientada a entornos cloud-agnostic, sustentada en tecnologías como Kubernetes, Kong, Keycloak, HashiCorp Vault, RabbitMQ, Redis, Prometheus y Grafana, cuya evaluación será realizada mediante el método ATAM con el propósito de validar el cumplimiento de los atributos de calidad definidos para la solución.

Capítulo 3

Desarrollo del Proyecto

3.1. Diagnóstico Arquitectónico Actual

3.1.1. Introducción

Los sistemas de información en el sector salud requieren altos niveles de calidad en atributos como seguridad, disponibilidad e interoperabilidad. En este contexto, el sistema SIIS es evaluado desde una perspectiva arquitectónica con el fin de identificar limitaciones y oportunidades de mejora.

El análisis se fundamenta en el método ATAM (Architecture Tradeoff Analysis Method), el cual permite evaluar decisiones arquitectónicas en función de atributos de calidad (Kazman et al., 2000).

3.1.2. Descripción del Sistema

El sistema SIIS es una aplicación web utilizada por instituciones de salud para la gestión de procesos clínicos y administrativos. Está construido bajo un modelo cliente-servidor y permite el acceso concurrente de múltiples usuarios.

Su arquitectura corresponde a un modelo monolítico de tres capas, desplegado en infraestructura on-premise.

3.1.3. Stack Tecnológico

El sistema está compuesto por tecnologías tradicionales como PHP para la lógica de negocio, Apache como servidor web y PostgreSQL como sistema gestor de bases de datos (PostgreSQL Global Development Group, 2023; Apache Software Foundation, 2023).

3.1.4. Arquitectura Actual

La arquitectura implementada sigue el modelo de tres capas:

- Capa de presentación: interfaz web accesible mediante navegador.
- Capa de lógica: implementada en PHP.
- Capa de datos: gestionada mediante PostgreSQL.

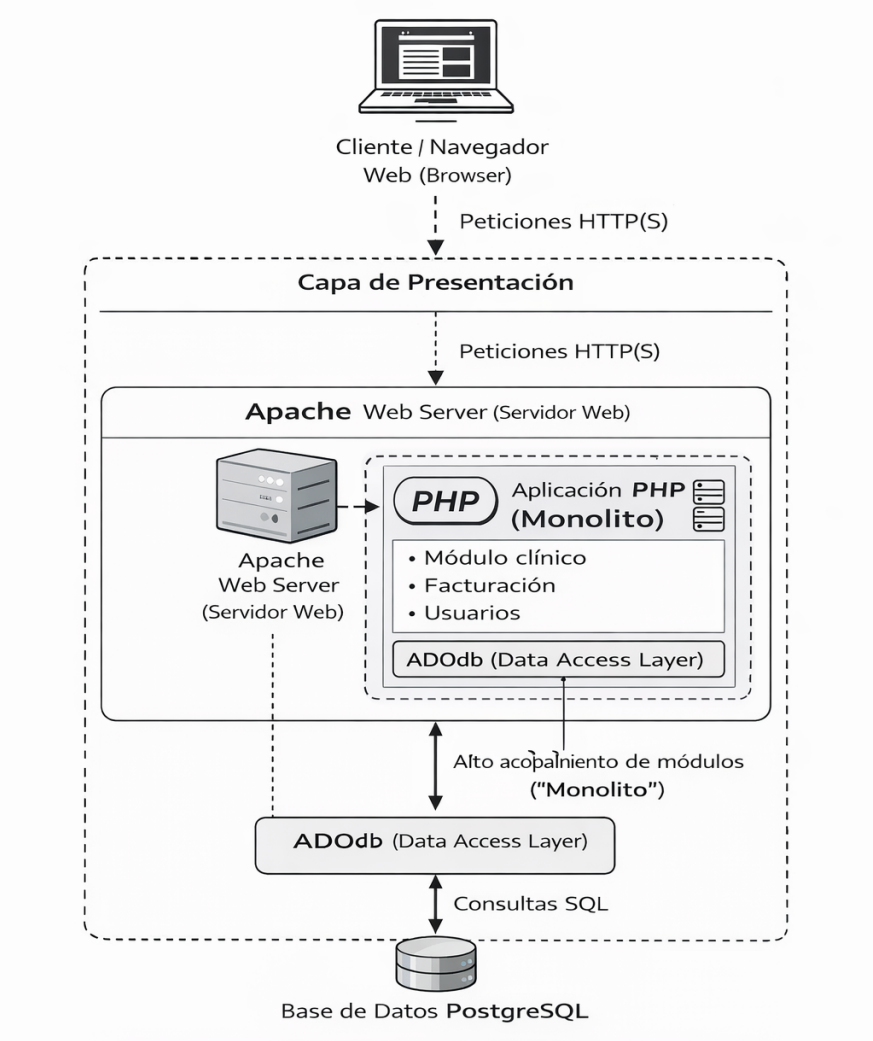


Figura 3.1: Arquitectura Actual

Este enfoque, aunque sencillo, genera un alto nivel de acoplamiento y limita la evolución del sistema (Bass et al., 2012).

3.1.5. Atributos de Calidad

De acuerdo con el estándar ISO/IEC 25010, los atributos de calidad son fundamentales para evaluar sistemas de software (ISO, 2011).

En el sistema analizado se identifican:

- Seguridad: control de acceso y auditoría.
- Rendimiento: uso de pool de conexiones.

- Escalabilidad: limitada a crecimiento vertical.
- Modificabilidad: restringida por acoplamiento.
- Interoperabilidad: baja, debido a la ausencia de APIs.

3.1.6. Evaluación mediante ATAM

El método ATAM permite analizar cómo las decisiones arquitectónicas impactan los atributos de calidad (Clements et al., 2002).

3.1.6.1. Escenarios de Calidad

Seguridad: El sistema utiliza mecanismos de autenticación basados en MD5, lo cual representa un riesgo debido a vulnerabilidades conocidas (OWASP Foundation, 2021).

Rendimiento: La concurrencia se gestiona mediante pool de conexiones, pero el procesamiento centralizado limita la escalabilidad.

Modificabilidad: La arquitectura monolítica dificulta la incorporación de nuevas funcionalidades.

Escalabilidad: El sistema depende de escalamiento vertical, lo cual restringe su crecimiento.

Interoperabilidad: La ausencia de APIs limita la integración con sistemas externos.

3.1.7. Discusión

Los resultados evidencian que la arquitectura actual cumple con los requerimientos operativos básicos, pero presenta limitaciones frente a arquitecturas orientadas a servicios y microservicios (Newman, 2021; Richardson, 2018).

3.1.8. Conclusiones

La arquitectura del sistema presenta debilidades significativas en atributos como escalabilidad, modificabilidad e interoperabilidad. Estas limitaciones justifican la necesidad de una transformación hacia arquitecturas desacopladas y cloud-agnostic, alineadas con definiciones modernas de computación en la nube (Mell and Grance, 2011).

3.2. Definición de Requerimientos Arquitectónicos y Atributos de Calidad

A partir del diagnóstico de la arquitectura actual del sistema, se identifican los requerimientos arquitectónicos necesarios para la solución propuesta. Estos requerimientos se definen con base en atributos de calidad, alineados con el modelo ISO/IEC 25010, el cual establece un marco de referencia para la evaluación de sistemas de software.

3.2.1. Modelo de Calidad ISO/IEC 25010

El estándar ISO/IEC 25010 define características de calidad que permiten evaluar la idoneidad de un sistema software. Entre las principales se encuentran: seguridad, eficiencia de desempeño, compatibilidad, usabilidad, confiabilidad, mantenibilidad y portabilidad.

Para el contexto del sistema SIIS, se priorizan aquellos atributos que impactan directamente la evolución arquitectónica del sistema, especialmente en términos de escalabilidad, integración y seguridad.

3.2.2. Atributos de Calidad Priorizados y Requerimientos Arquitectónicos

3.2.2.1. Escalabilidad

El sistema debe ser capaz de soportar incrementos en la carga de usuarios y volumen de datos sin degradar su desempeño.

Requerimiento arquitectónico: Se requiere una arquitectura distribuida que permita escalamiento horizontal mediante el desacoplamiento de componentes.

Justificación: La arquitectura actual presenta limitaciones al depender de escalamiento vertical.

3.2.2.2. Modificabilidad

El sistema debe permitir la incorporación de nuevas funcionalidades con un impacto mínimo en los componentes existentes.

Requerimiento arquitectónico: Se requiere una arquitectura desacoplada basada en servicios que reduzca la dependencia entre módulos.

Justificación: El alto acoplamiento del sistema monolítico dificulta su evolución.

3.2.2.3. Interoperabilidad

El sistema debe ser capaz de integrarse con sistemas externos del ecosistema de salud.

Requerimiento arquitectónico: Se requiere la implementación de APIs estandarizadas y mecanismos de integración.

Justificación: La arquitectura actual carece de capacidades de interoperabilidad.

3.2.2.4. Seguridad

El sistema debe garantizar la protección de la información sensible.

Requerimiento arquitectónico: Se requiere la implementación de mecanismos de autenticación robusta (OAuth2, JWT), cifrado seguro y control de acceso basado en roles.

Justificación: El uso de algoritmos obsoletos como MD5 representa un riesgo.

3.2.2.5. Disponibilidad

El sistema debe garantizar continuidad operativa.

Requerimiento arquitectónico: Se requiere redundancia, balanceo de carga y tolerancia a fallos.

Justificación: La arquitectura actual presenta puntos únicos de falla.

3.2.2.6. Mantenibilidad

El sistema debe facilitar su mantenimiento y evolución.

Requerimiento arquitectónico: Se requiere modularidad y separación de responsabilidades.

Justificación: La estructura actual dificulta la comprensión del sistema.

3.2.3. Árbol de Utilidad (Utility Tree)

El árbol de utilidad permite organizar y priorizar los atributos de calidad en función de escenarios concretos, facilitando la toma de decisiones arquitectónicas.

3.2.3.1. Escenarios Priorizados

Tabla 3.1: Escenarios del árbol de utilidad

Atributo	Escenario	Importancia	Riesgo
Escalabilidad	Incremento de usuarios concurrentes sin degradación	Alta	Alto
Modificabilidad	Incorporación de nuevos módulos sin impacto	Alta	Alto
Interoperabilidad	Integración con sistemas externos vía APIs	Alta	Alto
Seguridad	Autenticación robusta y control de acceso	Alta	Medio
Disponibilidad	Operación continua ante fallos	Media	Medio
Mantenibilidad	Cambios con bajo impacto en el sistema	Media	Alto

3.2.4. Relación entre Atributos y Decisiones Arquitectónicas

Los atributos de calidad definidos orientan directamente las decisiones arquitectónicas de la solución propuesta. En particular:

- La necesidad de escalabilidad y disponibilidad impulsa el uso de arquitecturas distribuidas.
- La modificabilidad e interoperabilidad justifican la adopción de microservicios y APIs.
- La seguridad requiere la implementación de mecanismos de autenticación y control de acceso.

3.2.5. Conclusión

La definición de los requerimientos arquitectónicos basada en atributos de calidad permite establecer una base sólida para el diseño de la arquitectura objetivo. Este enfoque asegura que la solución propuesta responda de manera efectiva a las limitaciones identificadas en el sistema actual, alineándose con estándares internacionales y buenas prácticas de arquitectura de software.

3.3. Diseño de la Arquitectura Tecnológica de la Solución

3.3.1. Introducción

Con el objetivo de solucionar los problemas de interoperabilidad, desacoplamiento, escalabilidad y seguridad identificados en los sistemas actuales, se propone una arquitectura empresarial basada en microservicios y principios *cloud-native*.

La arquitectura planteada incorpora mecanismos de autenticación centralizada, mensajería asíncrona, gestión segura de secretos, observabilidad, despliegue automatizado y orquestación de contenedores, permitiendo una plataforma moderna, resiliente y alineada con prácticas DevSecOps.

3.3.2. Definicion del stack tecnológico

Una vez definida la arquitectura objetivo, se procedió a seleccionar el stack tecnológico que soportará la implementación de los componentes arquitectónicos propuestos. La selección se realizó mediante un análisis comparativo de diferentes alternativas tecnológicas ampliamente utilizadas en entornos empresariales, considerando criterios como portabilidad, escalabilidad, interoperabilidad, madurez tecnológica, facilidad de integración, soporte comunitario, costos de adopción y alineación con los principios de una arquitectura basada en microservicios y APIs.

La Tabla 3.2 presenta la comparación de las principales tecnologías evaluadas para cada componente arquitectónico, incluyendo el API Gateway, la capa de mediación, el bus de eventos, la plataforma de despliegue de microservicios, el sistema gestor de bases de datos y las herramientas de observabilidad. Para cada alternativa se identificaron sus principales ventajas, desventajas y el nivel de portabilidad, con el propósito de seleccionar aquellas que ofrecen el mejor equilibrio entre funcionalidad, mantenibilidad y capacidad de evolución de la arquitectura.

Los resultados de esta evaluación permitieron identificar un conjunto de tecnologías que favorecen la reducción del acoplamiento entre sistemas, facilitan la interoperabilidad mediante estándares abiertos y proporcionan una base sólida para la escalabilidad y la operación de la solución propuesta.

Tabla 3.2: Comparación de alternativas tecnológicas por componente arquitectónico

Componente	**Opción**	**Ventajas**	**Desventajas**	**Nivel de Portabilidad**
API Gateway	Kong	Extensible, plugins, open source	Requiere configuración avanzada	Alto
	NGINX	Alto rendimiento, ampliamente adoptado	Configuración manual compleja	Alto
	Traefik	Integración nativa con Kubernetes	Menor madurez en entornos legacy	Alto
Capa de Mediación	Apache Camel	Soporta patrones EIP, alta flexibilidad	Curva de aprendizaje	Alto
	Spring Boot	Ecosistema robusto, fácil integración	Requiere diseño adicional	Alto
	Mule ESB	Integración empresarial completa	Licenciamiento y costo	Medio
Bus de Eventos	Apache Kafka	Alta escalabilidad, streaming	Complejidad operativa	Alto
	RabbitMQ	Fácil implementación, múltiples patrones	Menor rendimiento en streaming	Alto
	NATS	Ligero, alta velocidad	Menor adopción empresarial	Alto
Microservicios	Kubernetes	Estándar multi-cloud, escalable	Complejidad de gestión	Alto
	Docker	Portabilidad, facilidad de uso	No orquesta por sí solo	Alto
	OpenShift	Seguridad y gestión empresarial	Dependencia del ecosistema Red Hat	Medio
Base de Datos	PostgreSQL	Robusto, open source	Escalabilidad vertical limitada	Alto
	MongoDB	Flexible, escalable	Consistencia eventual	Alto
	MySQL	Amplia adopción	Menos flexible en escalabilidad	Alto
Observabilidad	Prometheus + Grafana	Estándar cloud-native, flexible	Configuración inicial	Alto
	ELK Stack	Análisis de logs potente	Alto consumo de recursos	Alto
	Jaeger	Trazabilidad distribuida	Requiere integración adicional	Alto

3.3.3. Selección del stack tecnológico

A partir del análisis comparativo de alternativas tecnológicas, se define el siguiente stack como base para la arquitectura de mediación propuesta:

- **API Gateway:** Kong
- **Gestión de identidad:** Keycloak
- **Gestión de secretos:** HashiCorp Vault
- **Capa de mediación:** Spring Boot (microservicios)
- **Bus de eventos:** RabbitMQ
- **Orquestación de microservicios:** Kubernetes
- **Base de datos:** PostgreSQL
- **Observabilidad:** Prometheus + Grafana
- **Cache:** Redis
- **Repositorio de imagenes:** ECR
- **Repositorio de codigo fuente:** Git

3.3.4. Justificación de la selección tecnológica

La selección del stack tecnológico se fundamenta en criterios de portabilidad, escalabilidad, madurez tecnológica y alineación con principios cloud-native, garantizando la independencia del proveedor (*cloud-agnostic*).

En primer lugar, Kong se selecciona como API Gateway debido a su naturaleza open source, su capacidad de extensión mediante plugins y su facilidad de despliegue en entornos Kubernetes, lo que permite implementar mecanismos de seguridad, control de tráfico y gobernanza de APIs de forma desacoplada.

Para la capa de mediación, se adopta Spring Boot bajo un enfoque de microservicios, debido a su amplio ecosistema, facilidad de integración y soporte para arquitecturas distribuidas. Esta elección permite implementar patrones de diseño orientados a servicios, facilitando el desacoplamiento y la mantenibilidad.

El uso de RabbitMQ como broker de mensajería permite habilitar comunicación asíncrona y patrones de integración orientados a eventos, mejorando el desacoplamiento, la resiliencia y la tolerancia a fallos del sistema. RabbitMQ soporta múltiples patrones de mensajería, como publicación/suscripción, colas de trabajo y enrutamiento mediante exchanges, y cuenta con una amplia adopción en entornos empresariales.

Kubernetes se selecciona como plataforma de orquestación debido a su capacidad para gestionar contenedores de manera eficiente, soportar despliegues multi-cloud y facilitar el autoescalamiento, lo que contribuye directamente a los atributos de disponibilidad y escalabilidad.

Como sistema de persistencia, PostgreSQL ofrece robustez, consistencia y compatibilidad con múltiples entornos, siendo una solución open source ampliamente utilizada y portable entre diferentes proveedores de infraestructura.

Finalmente, la observabilidad se implementa mediante Prometheus y Grafana, lo que permite monitorear métricas, visualizar información y realizar trazabilidad distribuida, facilitando la gestión operativa y el análisis de comportamiento del sistema.

En conjunto, este stack tecnológico permite construir una arquitectura desacoplada, escalable y portable, alineada con los atributos de calidad definidos y con los principios de diseño cloud-agnósticos.

3.3.5. Objetivo de la Arquitectura

Diseñar una arquitectura de mediación moderna que permita:

- Desacoplar los sistemas existentes.
- Facilitar la interoperabilidad entre aplicaciones.
- Centralizar la seguridad y autenticación.
- Mejorar la escalabilidad de los servicios.
- Garantizar alta disponibilidad.
- Implementar monitoreo integral.
- Automatizar despliegues mediante CI/CD.
- Asegurar la protección de credenciales y secretos.
- Facilitar la evolución y mantenimiento continuo de la plataforma.

3.3.6. Estilo Arquitectónico

3.3.6.1. Arquitectura de Microservicios

Cada funcionalidad del negocio es implementada como un servicio independiente, desplegado de manera autónoma y desacoplada.

Beneficios

- Escalabilidad independiente.
- Facilidad de mantenimiento.
- Despliegues aislados.
- Alta disponibilidad.
- Flexibilidad tecnológica.
- Reducción del acoplamiento.

3.3.6.2. Arquitectura Orientada a Eventos

Se implementa comunicación asíncrona mediante colas y eventos utilizando RabbitMQ.

Beneficios

- Desacoplamiento temporal.
- Tolerancia a fallos.
- Procesamiento distribuido.
- Integración entre sistemas heterogéneos.
- Optimización del rendimiento.

3.3.6.3. Arquitectura Cloud-Native

La solución es desplegada sobre Kubernetes utilizando contenedores Docker.

Beneficios

- Elasticidad.
- Escalabilidad horizontal.
- Recuperación automática.
- Automatización operativa.
- Portabilidad.

3.3.7. Modelo C4 de la Arquitectura

La arquitectura fue modelada utilizando el enfoque C4, permitiendo representar el sistema desde diferentes niveles de abstracción.

3.3.7.1. Nivel 1 – Context Diagram

El diagrama de contexto representa la interacción entre los usuarios, sistemas externos y la plataforma de microservicios.

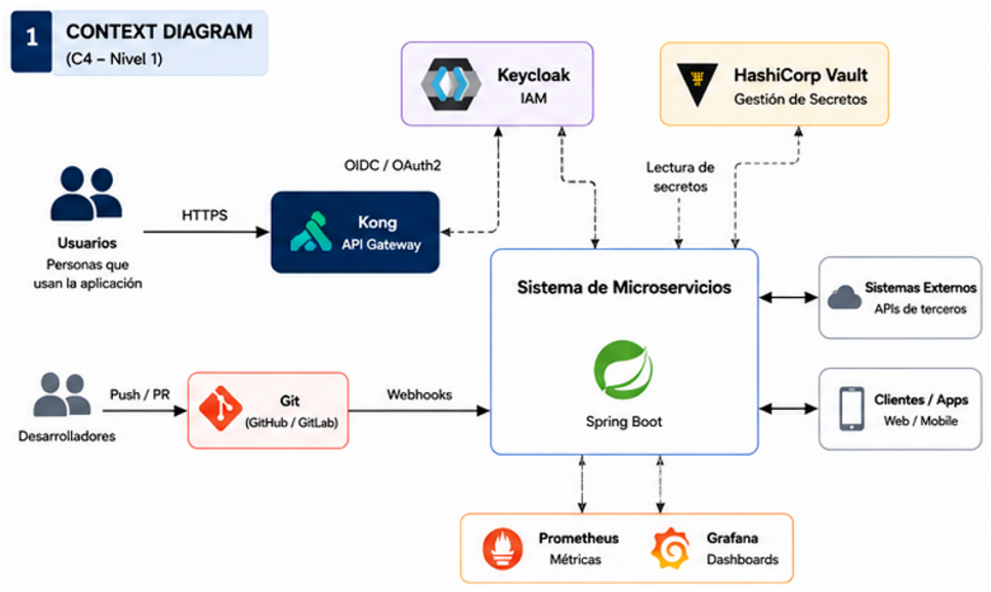


Figura 3.2: Diagrama de Contexto

3.3.7.2. Nivel 2 – Container Diagram

Este nivel representa los principales contenedores y componentes tecnológicos de la solución.

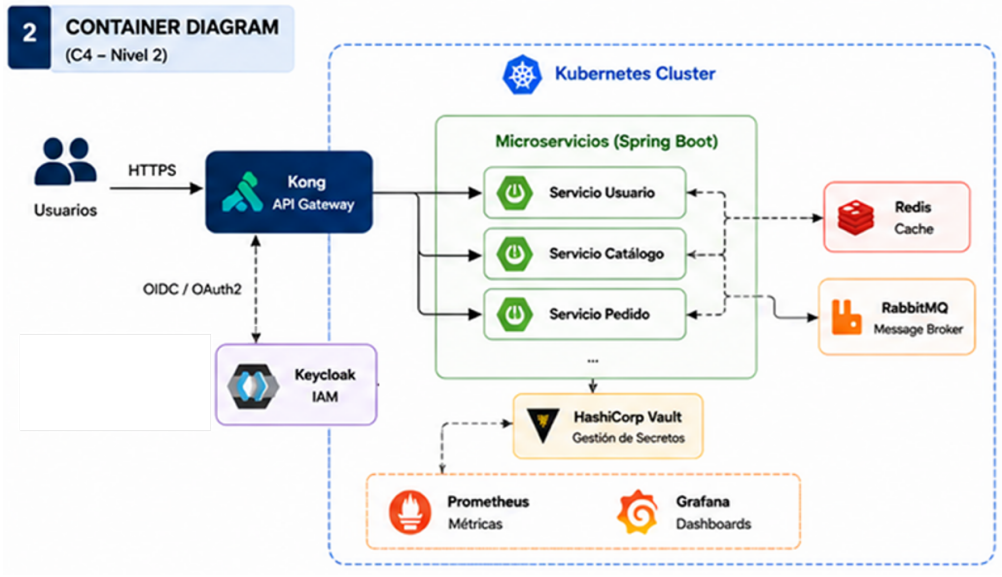


Figura 3.3: Diagrama de Contenedores

3.3.7.3. Nivel 3 – Component Diagram

Cada microservicio implementa componentes internos desacoplados siguiendo principios de arquitectura limpia y hexagonal.

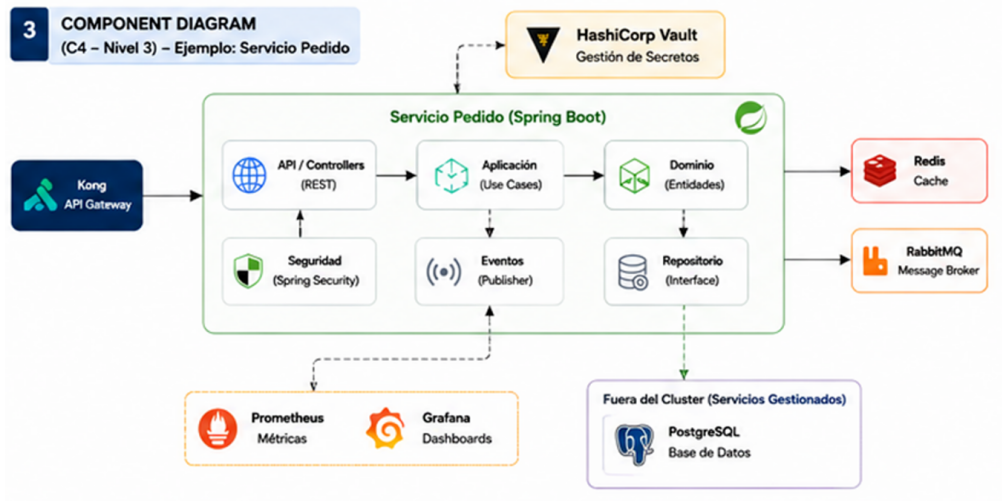


Figura 3.4: Diagrama de Componentes

3.3.7.4. Nivel 4 – Deployment Diagram

El despliegue de la solución se realiza sobre un clúster Kubernetes.

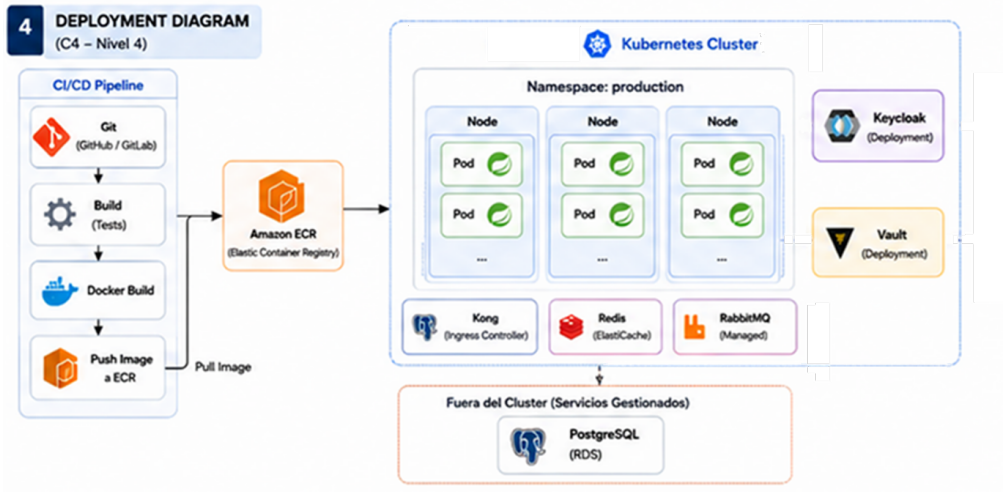


Figura 3.5: Diagrama de Despliegue

La base de datos se implementa fuera del clúster Kubernetes con el fin de garantizar persistencia independiente, respaldo administrado y alta disponibilidad.

3.3.8. Descripción de Componentes Tecnológicos

3.3.8.1. Kong API Gateway

Kong actúa como punto central de entrada a la plataforma.

Funciones

- Enrutamiento de APIs.
- Balanceo de carga.
- Rate limiting.
- Validación de JWT.
- Observabilidad.
- Transformación de solicitudes.

Beneficios

- Centralización del tráfico.
- Gobierno de APIs.

- Seguridad unificada.
- Escalabilidad.

3.3.8.2. Keycloak

Keycloak proporciona autenticación y autorización centralizada.

Características

- Single Sign-On (SSO).
- OAuth2.
- OpenID Connect.
- Gestión de usuarios.
- Roles y permisos.
- Multi-factor authentication.

Flujo de autenticación

1. El usuario solicita acceso.
2. Kong redirecciona a Keycloak.
3. Keycloak autentica al usuario.
4. Se genera un token JWT.
5. Kong valida el token.
6. El microservicio procesa la solicitud.

3.3.8.3. Spring Boot

Los microservicios son desarrollados utilizando Spring Boot.

Características

- APIs REST.
- Arquitectura desacoplada.
- Integración con Kubernetes.
- Seguridad OAuth2.
- Integración con RabbitMQ y Redis.

3.3.8.4. RabbitMQ

RabbitMQ implementa comunicación asíncrona entre servicios.

Casos de uso

- Eventos de negocio.
- Integración de sistemas.
- Procesamiento distribuido.
- Notificaciones.

Beneficios

- Reducción de acoplamiento.
- Tolerancia a fallos.
- Mejor rendimiento.

3.3.8.5. Redis

Redis funciona como motor de caché distribuido.

Casos de uso

- Caché de consultas.
- Gestión de sesiones.
- Tokens temporales.
- Optimización de respuestas.

Beneficios

- Baja latencia.
- Reducción de carga sobre PostgreSQL.
- Mejor tiempo de respuesta.

3.3.8.6. PostgreSQL

PostgreSQL es el motor relacional principal.

Funciones

- Persistencia transaccional.
- Integridad de datos.
- Soporte relacional y JSON.
- Alta disponibilidad.

Justificación Se selecciona PostgreSQL debido a:

- Estabilidad.
- Rendimiento.
- Compatibilidad empresarial.
- Soporte ACID.
- Escalabilidad.

3.3.8.7. HashiCorp Vault

Vault centraliza la gestión de secretos.

Funciones

- Almacenamiento seguro.
- Rotación automática.
- Secrets dinámicos.
- Gestión de certificados.

Beneficios

- Eliminación de secretos hardcodeados.
- Seguridad centralizada.
- Auditoría.

3.3.8.8. Kubernetes

Kubernetes es la plataforma principal de orquestación.

Funciones

- Escalabilidad horizontal.
- Alta disponibilidad.
- Recuperación automática.
- Gestión declarativa.
- Balanceo de carga.

Beneficios

- Resiliencia.
- Elasticidad.
- Automatización.

3.3.8.9. Prometheus y Grafana

Prometheus y Grafana permiten implementar observabilidad integral.

Métricas monitoreadas

- CPU.
- Memoria.
- Latencia.
- Requests HTTP.
- Eventos RabbitMQ.
- Rendimiento PostgreSQL.
- Uso de Redis.

Beneficios

- Monitoreo en tiempo real.
- Detección temprana de fallos.
- Trazabilidad.

3.3.8.10. Git y CI/CD

Git centraliza el código fuente y facilita la integración continua.

Flujo DevOps

1. Desarrollo en Git.
2. Pipeline CI/CD.
3. Construcción Docker.
4. Publicación en Amazon ECR.
5. Despliegue automático en Kubernetes.

Beneficios

- Automatización.
- Reducción de errores.
- Trazabilidad.
- Entrega continua.

3.3.9. Seguridad de la Arquitectura

La solución incorpora seguridad en múltiples capas.

Tabla 3.3: Mecanismos de seguridad implementados

Mecanismo	**Tecnología**
Autenticación	Keycloak
Autorización	OAuth2 / JWT
Gestión de secretos	Vault
Seguridad APIs	Kong
Comunicación segura	TLS
RBAC	Kubernetes

3.3.10. Observabilidad

La arquitectura implementa monitoreo integral mediante Prometheus y Grafana.

Alertas

- Caída de servicios.
- Saturación de recursos.
- Errores HTTP.
- Consumo excesivo.

3.3.11. Alta Disponibilidad y Escalabilidad

Tabla 3.4: Estrategias de alta disponibilidad

Estrategia	**Implementación**
Escalabilidad horizontal	Kubernetes HPA
Balanceo de carga	Kong
Tolerancia a fallos	RabbitMQ
Caché distribuido	Redis
Recuperación automática	Kubernetes
Despliegue continuo	CI/CD

3.3.12. Beneficios de la Arquitectura Propuesta

La arquitectura propuesta ofrece los siguientes beneficios:

- Desacoplamiento de sistemas.
- Mayor seguridad.
- Integración simplificada.
- Escalabilidad horizontal.
- Alta disponibilidad.
- Gobierno centralizado de APIs.
- Observabilidad completa.
- Automatización DevOps.
- Reducción de riesgos operativos.
- Mayor mantenibilidad.

3.3.13. Conclusiones

La arquitectura diseñada permite construir una plataforma moderna, segura y escalable alineada con principios cloud-native y DevSecOps.

La integración de Kubernetes, Kong, Keycloak y Vault proporciona una base sólida para la seguridad y gestión de servicios, mientras que RabbitMQ y Redis optimizan el desacoplamiento y el rendimiento del sistema.

Adicionalmente, Prometheus y Grafana garantizan capacidades avanzadas de monitoreo y observabilidad, permitiendo detectar incidentes de forma temprana y mejorar la operación continua de la plataforma.

Finalmente, el uso de Git, CI/CD y Amazon ECR facilita la automatización de despliegues y la entrega continua de nuevas funcionalidades, fortaleciendo la capacidad evolutiva de la organización.

3.4. Implementación del prototipo

Con el fin de validar la viabilidad de la arquitectura propuesta, se desarrolló un prototipo funcional que implementa los principales componentes definidos durante el diseño arquitectónico. El propósito del prototipo no fue reproducir la totalidad de los procesos de la entidad SIMDE, sino demostrar que la arquitectura planteada permite mejorar la interoperabilidad entre sistemas, reducir el acoplamiento entre aplicaciones y facilitar la exposición de servicios mediante interfaces estandarizadas.

La implementación se realizó siguiendo los principios de una arquitectura basada en microservicios, donde cada servicio encapsula una responsabilidad específica y se comunica con los demás componentes mediante APIs REST y mecanismos de mensajería asíncrona. Esta aproximación favorece la independencia funcional de los servicios, simplifica su mantenimiento y permite su evolución de manera independiente.

Como punto de entrada a la arquitectura se implementó un API Gateway encargado de centralizar el acceso a los servicios, gestionar el enrutamiento de solicitudes y proporcionar capacidades de seguridad y control de acceso. La integración entre los sistemas existentes y los nuevos servicios se realizó mediante una capa de mediación, responsable de transformar mensajes, coordinar procesos de integración y garantizar la interoperabilidad entre aplicaciones con diferentes tecnologías.

Para desacoplar la comunicación entre componentes y soportar el intercambio de eventos, se implementó un bus de mensajería que permite la comunicación asíncrona entre los servicios, reduciendo las dependencias directas y mejorando la escalabilidad de la solución. Asimismo, los microservicios fueron desplegados en contenedores, lo que facilita su portabilidad, el aislamiento de dependencias y la automatización del despliegue en diferentes entornos.

La persistencia de la información se implementó mediante un sistema gestor de bases de datos que garantiza la integridad y disponibilidad de los datos requeridos por los servicios. De forma complementaria, se incorporaron herramientas de monitoreo y observabilidad para recopilar métricas de desempeño, supervisar el estado de los servicios y facilitar la detección de incidentes durante la ejecución del prototipo.

Finalmente, se realizaron pruebas funcionales y de integración con el propósito de verificar el correcto funcionamiento de los componentes implementados, validar la comunicación entre los servicios y comprobar que la arquitectura propuesta satisface los requerimientos de interoperabilidad, mantenibilidad, escalabilidad y disponibilidad definidos durante el diseño. Los resultados obtenidos constituyen la base para la evaluación de la solución presentada en el capítulo siguiente.

3.4.1. Instalación y configuración del clúster Kubernetes

Como base de la arquitectura propuesta se realizó la instalación y configuración de un clúster Kubernetes, encargado de administrar el ciclo de vida de los servicios desplegados dentro del prototipo. Kubernetes fue seleccionado debido a que proporciona mecanismos nativos para la gestión de contenedores, distribución de cargas de trabajo, recuperación automática ante fallos y escalamiento horizontal de aplicaciones.

La configuración inicial del clúster contempló la preparación de los nodos de ejecución, la instalación del motor de contenedores y la configuración de los componentes principales de Kubernetes. Para la administración del clúster se utilizaron herramientas de línea de comandos que permitieron realizar tareas como creación de namespaces, despliegue de aplicaciones mediante archivos declarativos YAML, gestión de servicios internos y validación del estado de los recursos.

Dentro del clúster fueron definidos los espacios lógicos necesarios para organizar los diferentes componentes de la arquitectura, separando los servicios de infraestructura, integración y monitoreo. Esta organización facilita la administración del prototipo, mejora el aislamiento entre componentes y permite una evolución controlada de la solución.

La validación de la instalación se realizó mediante la comprobación del estado de los nodos, servicios y pods desplegados, verificando que la plataforma contara con los recursos necesarios para soportar la ejecución de los componentes posteriores.

3.4.2. Instalación y configuración de Kong + Ingress Controller + Konga

Una vez establecido el entorno Kubernetes, se procedió con la implementación del componente encargado de la gestión y exposición de las interfaces de comunicación. Para esta función se integró Kong como API Gateway, acompañado del Kong Ingress Controller para administrar el tráfico de entrada hacia los servicios desplegados dentro del clúster.

Kong permite centralizar las solicitudes dirigidas hacia los servicios de la arquitectura, proporcionando capacidades como enrutamiento, gestión de autenticación, aplicación de políticas de seguridad y administración de APIs. Su integración con Kubernetes mediante el Ingress Controller permite definir las reglas de exposición utilizando recursos nativos de la plataforma.

La instalación se realizó mediante manifiestos de despliegue y configuraciones declarativas, definiendo los servicios, rutas y políticas necesarias para la comunicación entre clientes externos y los microservicios internos. Adicionalmente, se incorporó Konga como interfaz gráfica de administración, permitiendo visualizar y gestionar de manera simplificada la configuración del API Gateway.

La validación del componente se efectuó mediante pruebas de acceso a los servicios publicados, comprobando el correcto funcionamiento del direccionamiento de solicitudes, la comunicación entre

componentes y la administración de las APIs expuestas.

3.4.3. Instalación y configuración de HashiCorp Vault y Keycloak

Como parte de los mecanismos de seguridad definidos para la arquitectura propuesta, se implementaron HashiCorp Vault y Keycloak como componentes especializados para la gestión segura de credenciales, secretos, identidad y control de acceso. La incorporación de estas herramientas permite fortalecer la seguridad de los servicios desplegados, evitando la exposición de información sensible dentro del código fuente o archivos de configuración y proporcionando un esquema centralizado de autenticación y autorización.

3.4.3.1. Instalación y configuración de HashiCorp Vault

HashiCorp Vault fue implementado como gestor centralizado de secretos, encargado de administrar información sensible utilizada por los diferentes componentes de la arquitectura, tales como credenciales de bases de datos, claves de acceso, tokens y parámetros de configuración confidenciales.

La instalación de Vault se realizó dentro del clúster Kubernetes mediante recursos declarativos, definiendo los componentes necesarios para su ejecución, exposición interna y almacenamiento persistente. La configuración inicial contempló la inicialización del servicio, generación de claves de acceso administrativas, configuración del método de almacenamiento (storage backend) y habilitación de los mecanismos de autenticación requeridos por los servicios consumidores.

Adicionalmente, se definieron políticas de acceso (policies) orientadas a implementar el principio de mínimo privilegio, permitiendo que cada aplicación acceda únicamente a los secretos necesarios para su funcionamiento. Esta configuración reduce la dependencia de variables sensibles almacenadas directamente en los servicios y mejora la trazabilidad sobre el acceso a información crítica.

La validación de Vault se realizó mediante pruebas de creación, almacenamiento y recuperación de secretos, verificando la comunicación segura entre los servicios desplegados y el sistema de gestión de credenciales.

3.4.3.2. Instalación y configuración de Keycloak

Keycloak fue incorporado como proveedor centralizado de identidad (Identity Provider) para gestionar los procesos de autenticación y autorización de los usuarios y aplicaciones que interactúan con la arquitectura propuesta. Su implementación permite aplicar estándares abiertos de seguridad como OAuth 2.0, OpenID Connect y JSON Web Tokens (JWT).

La instalación de Keycloak se realizó dentro del clúster Kubernetes mediante manifiestos de despliegue, configurando los recursos necesarios para su operación, persistencia de datos y exposición mediante el API Gateway. Durante la configuración inicial se definieron los elementos principales de identidad, incluyendo el dominio lógico (realm), clientes (clients), usuarios, roles y permisos asociados.

La integración de Keycloak con Kong permitió establecer un mecanismo de control de acceso para los servicios publicados mediante el API Gateway, validando los tokens de autenticación antes

de permitir el acceso a los recursos protegidos. De esta manera, se centraliza la administración de identidades y se evita la implementación independiente de mecanismos de autenticación en cada microservicio.

Finalmente, se realizaron pruebas de autenticación y autorización para validar la emisión y validación de tokens JWT, el control de acceso basado en roles y la correcta integración entre Keycloak, Kong y los servicios desplegados en Kubernetes.

La incorporación conjunta de Vault y Keycloak proporciona una capa de seguridad transversal para la arquitectura, permitiendo gestionar de forma centralizada los secretos y las identidades, fortaleciendo atributos de calidad como seguridad, mantenibilidad y confiabilidad de la solución propuesta.

3.4.4. Instalación y configuración de RabbitMQ

Como mecanismo de comunicación asíncrona entre componentes de la arquitectura se implementó RabbitMQ como sistema de mensajería basado en el protocolo AMQP. La incorporación de este componente permitió reducir el acoplamiento entre servicios, facilitando la comunicación mediante intercambio de mensajes sin requerir una dependencia directa entre los consumidores y productores de información.

La instalación de RabbitMQ se realizó dentro del clúster Kubernetes mediante recursos de despliegue y servicios internos, definiendo los parámetros necesarios para su operación, almacenamiento persistente y acceso desde las aplicaciones desplegadas.

Durante la configuración se establecieron elementos fundamentales del modelo de mensajería, incluyendo colas, intercambios (exchanges) y reglas de enrutamiento (bindings), permitiendo implementar diferentes patrones de comunicación entre servicios.

La validación del componente incluyó pruebas de publicación y consumo de mensajes, verificando la entrega correcta de eventos, la persistencia de mensajes cuando correspondía y el funcionamiento adecuado de la comunicación desacoplada entre los servicios del prototipo.

3.4.5. Instalación y configuración de Prometheus + Grafana

Con el objetivo de incorporar capacidades de observabilidad al prototipo, se implementó una solución de monitoreo basada en Prometheus y Grafana. Esta integración permitió recopilar métricas operativas de los componentes desplegados y proporcionar mecanismos de visualización para analizar el comportamiento de la plataforma.

Prometheus fue configurado como sistema de recopilación de métricas, encargado de consultar periódicamente los puntos de monitoreo expuestos por los servicios y componentes de infraestructura. Para complementar esta funcionalidad, Grafana fue integrado como herramienta de visualización, permitiendo construir paneles (dashboards) con indicadores relevantes del estado del clúster y las aplicaciones.

La configuración contempló la definición de fuentes de datos, la instalación de agentes de recopilación y la creación de paneles para visualizar métricas como consumo de recursos, disponibilidad de servicios, estado de pods y comportamiento general de la plataforma.

La implementación de estas herramientas permitió evaluar el funcionamiento del prototipo desde una perspectiva operacional, facilitando la identificación de problemas, el análisis del rendimiento y la toma de decisiones relacionadas con la administración de la arquitectura propuesta.

3.4.6. Análisis de costos de la implementación

La implementación del prototipo se realizó utilizando tecnologías basadas principalmente en software libre y componentes de código abierto, lo que permitió reducir los costos asociados a la adquisición de licencias. La selección del stack tecnológico, conformado por Kubernetes, Kong Ingress Controller, RabbitMQ, Prometheus, Grafana, Vault, Keycloak y PostgreSQL, representa una alternativa viable para organizaciones que buscan modernizar sus plataformas tecnológicas sin incurrir en elevados costos iniciales por licenciamiento.

Desde la perspectiva de infraestructura, durante la fase de desarrollo y validación del prototipo se utilizaron recursos locales mediante Docker Desktop y Kubernetes. Por esta razón, los costos directos estuvieron relacionados principalmente con el equipo de desarrollo y el consumo de recursos computacionales disponibles. Este enfoque permitió realizar pruebas de integración, despliegue y comunicación entre servicios sin requerir infraestructura adicional.

Para un escenario productivo, los costos de implementación estarían asociados principalmente a la infraestructura requerida para ejecutar el clúster Kubernetes y los servicios complementarios. Entre los principales componentes de costo se consideran:

Infraestructura de cómputo: recursos necesarios para ejecutar los nodos del clúster Kubernetes, incluyendo capacidad de procesamiento, memoria y almacenamiento. Almacenamiento persistente: recursos requeridos por componentes que mantienen información, como bases de datos, configuraciones y registros históricos. Red y disponibilidad: costos relacionados con balanceadores de carga, comunicación entre servicios y mecanismos de alta disponibilidad. Monitoreo y operación: recursos destinados al almacenamiento de métricas, registros y herramientas de observabilidad. Administración y capacitación: inversión necesaria para la formación del personal técnico encargado de gestionar tecnologías de contenedores, orquestación, seguridad y automatización.

Una ventaja importante de la arquitectura propuesta es que la mayoría de los componentes utilizados no requieren costos de licenciamiento, debido a que están basados en proyectos de código abierto. Esto permite concentrar la inversión principalmente en infraestructura, operación y capacitación técnica, reduciendo la dependencia de soluciones propietarias.

En una implementación basada en servicios administrados de Kubernetes en la nube, como Amazon Elastic Kubernetes Service (EKS) o Azure Kubernetes Service (AKS), los costos dependerán del tamaño del clúster, cantidad de nodos, características de los recursos asignados y servicios complementarios utilizados. Aunque este modelo puede incrementar los costos operativos frente a una implementación local, proporciona beneficios como escalabilidad automática, disponibilidad administrada, reducción de tareas de mantenimiento de infraestructura y mayor facilidad para crecer según la demanda.

En términos generales, la solución propuesta presenta un equilibrio favorable entre inversión y beneficios técnicos. Si bien la adopción de una arquitectura basada en microservicios requiere una

inversión inicial en infraestructura y capacitación, los beneficios obtenidos en mantenibilidad, escalabilidad, interoperabilidad y seguridad permiten justificar su implementación como una estrategia de modernización tecnológica.

La estimación presentada en la Tabla 3.5 considera una arquitectura compuesta por cinco APIs independientes, un servicio Worker para procesamiento asíncrono, RabbitMQ como sistema de mensajería y servicios complementarios de seguridad y observabilidad. Los valores corresponden a una aproximación del costo mensual de operación para ambientes independientes de desarrollo, pruebas y producción utilizando servicios administrados de Kubernetes en proveedores de nube.

Tabla 3.5: Estimación mensual de costos de implementación por ambiente

Ambiente	**Configuración estimada**	**AWS (EKS)**	**Azure (AKS)**
Desarrollo	▪ 1 nodo Kubernetes ▪ 5 APIs (1 réplica) ▪ 1 Worker ▪ RabbitMQ básico ▪ Kong, Vault y Keycloak ▪ PostgreSQL básico ▪ Monitoreo reducido	USD 90–160	USD 80–150
Pruebas	▪ 2 nodos Kubernetes ▪ 5 APIs con réplicas ▪ 1–2 Workers ▪ RabbitMQ persistente ▪ Pruebas de integración ▪ Monitoreo activo	USD 250–450	USD 220–420
Producción	▪ 3 o más nodos Kubernetes ▪ APIs escalables (2–3 réplicas) ▪ Workers escalables ▪ RabbitMQ persistente ▪ PostgreSQL administrado ▪ Backups y monitoreo completo	USD 750–1.600	USD 700–1.500
Total mensual	**Tres ambientes independientes**	**USD 1.090–2.210**	**USD 1.000–2.070**

Capítulo 4

Evaluación de la Arquitectura Propuesta mediante ATAM

4.1. Introducción

Una vez diseñada e implementada la arquitectura de mediación basada en microservicios, es necesario verificar que las decisiones arquitectónicas adoptadas contribuyen efectivamente al cumplimiento de los atributos de calidad definidos durante la fase de requerimientos arquitectónicos.

Para realizar esta validación se emplea el método ATAM (Architecture Tradeoff Analysis Method), desarrollado por el Software Engineering Institute (SEI) de Carnegie Mellon University. Este método permite evaluar arquitecturas de software mediante el análisis de escenarios de calidad, identificando riesgos, puntos de sensibilidad, puntos de compromiso (tradeoffs) y beneficios asociados a las decisiones de diseño arquitectónico Clements et al. (2002).

La aplicación de ATAM permite determinar si la arquitectura propuesta satisface los atributos de calidad priorizados para el sistema y evaluar los impactos que determinadas decisiones arquitectónicas pueden generar sobre otros atributos.

4.2. Objetivo de la Evaluación

Evaluar la arquitectura de mediación basada en microservicios propuesta mediante la aplicación del método ATAM, con el fin de determinar el grado de cumplimiento de los atributos de calidad priorizados y analizar los compromisos arquitectónicos derivados de las decisiones de diseño adoptadas.

4.3. Descripción de la Arquitectura Evaluada

La arquitectura evaluada corresponde a una plataforma cloud-agnostic basada en microservicios desplegada sobre Kubernetes e integrada mediante mecanismos de comunicación síncrona y asíncrona.

La solución está compuesta por los siguientes componentes:

- Kong API Gateway.
- Keycloak.

- HashiCorp Vault.
- RabbitMQ.
- Redis.
- PostgreSQL.
- Kubernetes.
- Prometheus.
- Grafana.
- Microservicios desarrollados con Spring Boot.

La arquitectura fue diseñada bajo principios de desacoplamiento, escalabilidad horizontal, observabilidad y portabilidad entre proveedores de infraestructura.

4.4. Identificación de los Stakeholders

Siguiendo la metodología ATAM, se identifican los actores involucrados en la arquitectura.

Tabla 4.1: Stakeholders identificados

Stakeholder	**Interés**
Usuarios finales	Disponibilidad y desempeño del sistema
Administradores	Gestión y monitoreo de la plataforma
Equipo de desarrollo	Mantenibilidad y facilidad de evolución
Equipo DevOps	Automatización y despliegue continuo
Organización	Seguridad y continuidad operativa
Sistemas externos	Interoperabilidad e integración

4.5. Árbol de Utilidad

El árbol de utilidad representa los atributos de calidad priorizados durante el diseño arquitectónico.

Tabla 4.2: Árbol de utilidad

Atributo	**Escenario**	**Importancia**	**Riesgo**
Seguridad	Acceso únicamente mediante autenticación válida	Alta	Alto
Escalabilidad	Incremento de usuarios concurrentes sin degradación significativa	Alta	Alto
Disponibilidad	Continuidad del servicio ante fallos de componentes	Alta	Medio
Interoperabilidad	Integración con sistemas externos mediante APIs	Alta	Medio
Modificabilidad	Incorporación de nuevas funcionalidades con impacto mínimo	Alta	Alto
Observabilidad	Supervisión en tiempo real de la plataforma	Media	Bajo
Mantenibilidad	Facilidad para corregir errores y realizar mejoras	Media	Medio

4.6. Escenarios de Calidad Evaluados

4.6.1. Escenario S1 - Seguridad

Fuente: Usuario externo.

Estímulo: Intento de acceso a recursos protegidos.

Artefacto: Kong API Gateway y Keycloak.

Respuesta esperada: El sistema valida el token JWT antes de permitir el acceso.

Medida de respuesta: El 100 % de solicitudes sin credenciales válidas deben ser rechazadas.

Resultado: Se evidencia en la imagen 4.1 que se enviaron 100 solicitudes de consumo del api DemoApi donde se evidencia la respues 401 Unautorhorized ya que estas solicitudes tenian una parametrizacion de AUTH se envidencia en la imagen 4.2.

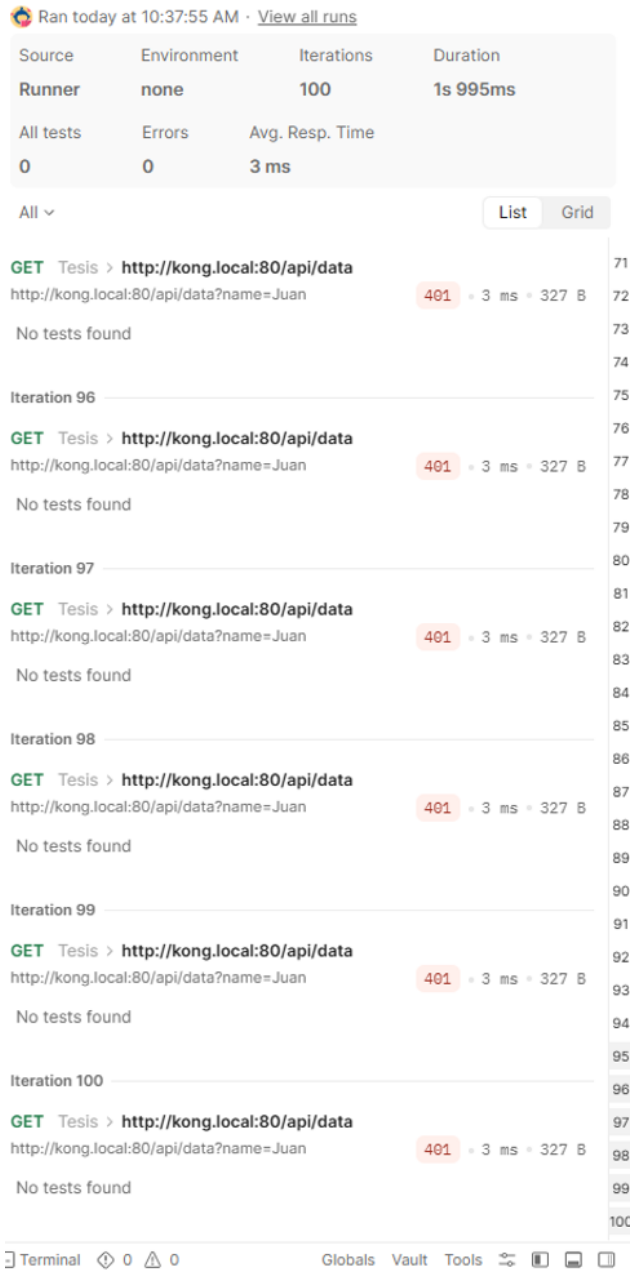


Figura 4.1: Resultado de ejecucion de 100 peticiones

4.6.2. Escenario S2 - Gestión de Secretos

Fuente: Aplicaciones internas.
Estímulo: Solicitud de acceso a credenciales.

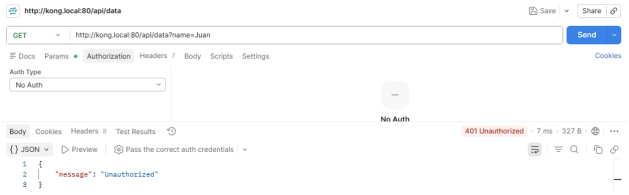


Figura 4.2: No Auth

Artefacto: HashiCorp Vault.

Respuesta esperada: Las credenciales son recuperadas dinámicamente desde Vault.

Medida de respuesta: Ninguna credencial permanece almacenada en el código fuente o Archivos de Despliegue.

Resultado: Se evidencia en la imagen 4.3 que en los archivos de de despliegue no maneja el valor propio sino que maneja variables que mendiante integracion con Vault los cuales se obtiene en tiempo de ejecucion. en la imagen 4.4 podemos evidenciar los valores almacenados en Vault. Dando como resultado Aprobado al escenario S2 Gestión de Secretos.

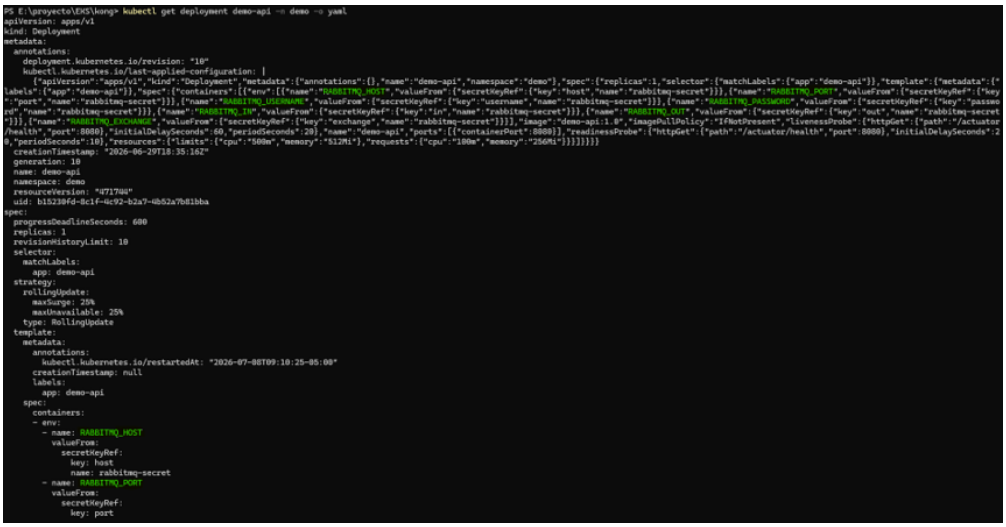


Figura 4.3: Archivo de Despliegue

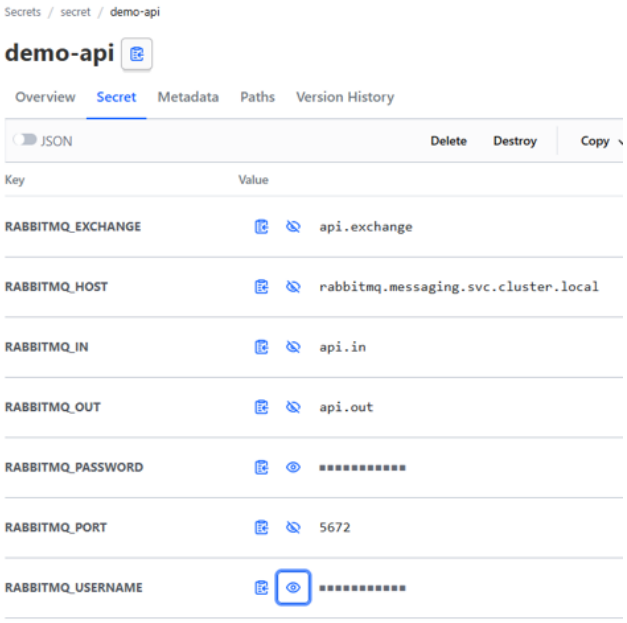


Figura 4.4: Parametrizacion Vault

4.6.3. Escenario E1 - Escalabilidad

Fuente: Usuarios concurrentes.

Estímulo: Incremento del volumen de solicitudes.

Artefacto: Kubernetes y microservicios.

Respuesta esperada: Creación automática de nuevas réplicas.

Medida de respuesta: El tiempo promedio de respuesta se mantiene dentro de los límites definidos.

Resultado: Se evidencia en la imagen 4.5 el estado inical y HPA del pod demo-api e iniciamos a ejecutar desde postman una prueba de carga del servicio enviando aproximadamente 6.347 peticiones en un lapso de 1 minuto lo cual genera automaticamente el estado pending e iniciando el proceso de creacion de 4 nuevos pods que incian en estado ContainerCreating para posteriormente quedar en estado running.

como resultado podemos evidenciar en la imagen 4.5 que en rabbitMQ en la cola api.in estan los 6.347 mensajes enviado y tambien se puede observar en la consola de powershell el consumo de recursos de cada pod.

dado lo anterior se da como satisfactoria la prueba ya que cumple con el escenario de escalabilidad

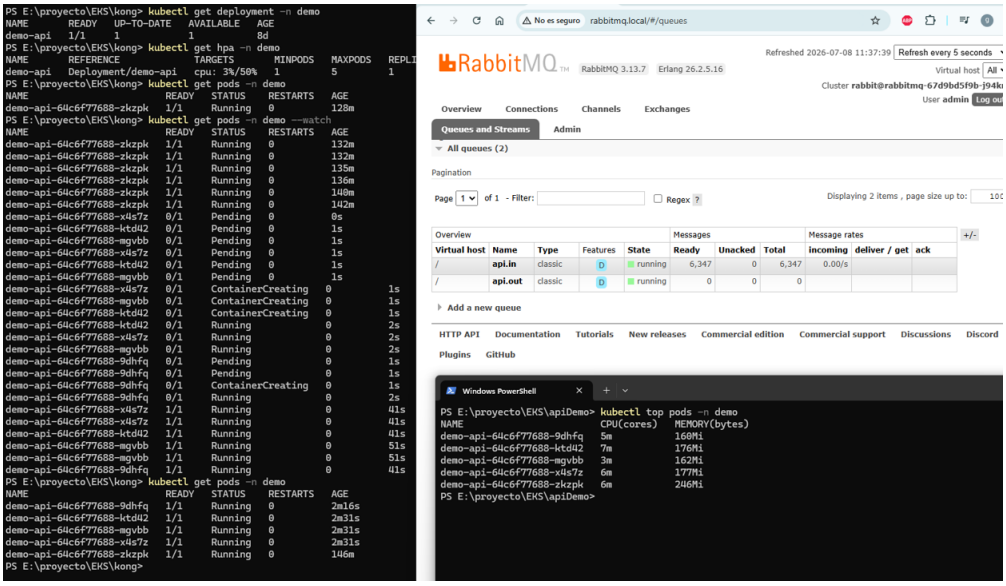


Figura 4.5: Evidencia Escalabilidad

4.6.4. Escenario D1 - Disponibilidad

Fuente: Fallo de un contenedor.

Estímulo: Interrupción inesperada de una instancia.

Artefacto: Kubernetes.

Respuesta esperada: Recreación automática del contenedor.

Medida de respuesta: Recuperación automática sin intervención manual.

Resultado: Se evidencia en la imagen 4.6 el resultado de la ejecuion de la prueba en la cual inicialmente listamos el estado del deployment y del pod demo-api

una vez ejecutado el comando de validacion constante del pod, desde otra consola de powershell realizamos la eliminacion del pod por lo cual se evidencia el estado terminating y automaticamente aparece un nuevo pod en estado ContainerCreating y posteriormente pasa a running, pero queriendo llevar la prueba a un entorno mas realista una vez el pod esta en estado running por medio de la otra consola realizamos la conexion al pod y ejecutamos el comando para eliminar el trabajo principal del pod con el fin de simular una falla interna y podemos observar que el pod queda en estado error y posteriomente pasa a estado running. pero lo que podemos observar es que ahora en Restarts paso de 0 a 1 lo cual nos indica cuantas veces se a reiniado el pod.

nota: si el pod es elliminado y se crea nuevamente este inicia con el Restarts en 0.

Dado lo anterior se evidencia el correcto funcionamiento del escenario por lo cual califica como aprobado.

```
PS E:\proyecto\EKS\apiDemo> kubectl get deployment -n demo
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
demo-api   1/1     1            1           8d
PS E:\proyecto\EKS\apiDemo> kubectl get pods -n demo -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP            NODE
demo-api-64c6f77688-5cllr   1/1     Running   0          13m   10.244.2.13   desktop-worker
PS E:\proyecto\EKS\apiDemo> kubectl get pods -n demo --watch
NAME                        READY   STATUS    RESTARTS   AGE
demo-api-64c6f77688-5cllr   1/1     Running   0          13m
demo-api-64c6f77688-5cllr   1/1     Terminating   0          14m
demo-api-64c6f77688-9qfgw   0/1     Pending       0          0s
demo-api-64c6f77688-9qfgw   0/1     Pending       0          0s
demo-api-64c6f77688-9qfgw   0/1     ContainerCreating   0          0s
demo-api-64c6f77688-5cllr   0/1     Error               0          14m
demo-api-64c6f77688-9qfgw   0/1     Running             0          2s
demo-api-64c6f77688-5cllr   0/1     Error               0          14m
demo-api-64c6f77688-5cllr   0/1     Error               0          14m
demo-api-64c6f77688-9qfgw   1/1     Running             0          40s
demo-api-64c6f77688-9qfgw   1/1     Running             0          40s
demo-api-64c6f77688-9qfgw   0/1     Error               0          2m48s
demo-api-64c6f77688-9qfgw   0/1     Running             1 (2s ago)   2m49s
demo-api-64c6f77688-9qfgw   1/1     Running             1 (33s ago)   3m20s
PS E:\proyecto\EKS\apiDemo> kubectl get pods -n demo
NAME                        READY   STATUS    RESTARTS      AGE
demo-api-64c6f77688-9qfgw   1/1     Running   1 (43s ago)   3m30s
PS E:\proyecto\EKS\apiDemo>
```

Figura 4.6: Evidencia de Disponibilidad

4.6.5. Escenario I1 - Interoperabilidad

Fuente: Sistema externo.

Estímulo: Solicitud de integración mediante API.

Artefacto: Kong API Gateway.

Respuesta esperada: Exposición controlada de servicios mediante APIs REST.

Medida de respuesta: Integración exitosa con sistemas externos.

Resultado: Se evidencia en la imagen 4.7 que modificando la configuracion por medio de konga del routes del api Demo-Api podemos controlar la publicacion externa del cluster. en el caso de la prueba se inactivo el parametro de strip path lo cual genera automaticamnte una afectacion en el consumo del servicio dando como respuesta 404 Not Fount

Dado lo anterior se da como cumplido el escenario de interoperabilidad ya que permine la exposicion controlada de los apis mediante el Kong.

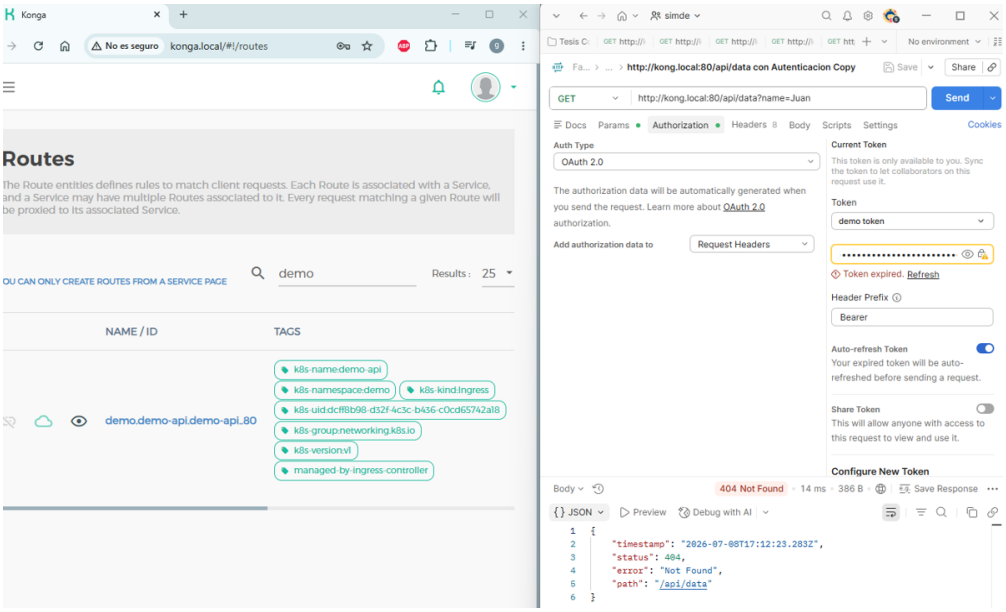


Figura 4.7: Evidencia Interoperabilidad

4.6.6. Escenario M1 - Modificabilidad

Fuente: Equipo de desarrollo.

Estímulo: Implementación de una nueva funcionalidad.

Artefacto: Arquitectura de microservicios.

Respuesta esperada: Incorporación de nuevos servicios sin afectar los existentes.

Medida de respuesta: Cambios localizados en componentes específicos.

Resultado: para esta prueba se desarrollo una nueva funcionalidad que lo que va a realizar el tomar el mensaje de la cola api.in de rabbitMQ lo procesa cambiandolo de estado para posteriormente almacenarlo en base de datos y por medio del metodo de consulta del api-demo vemos que esta en estado FINALIZADO. todo esto de forma automatizada.

En la imagen 4.8 podemos ver el consumo inicial y en la imagen 4.9 podemos observar la respuesta del api-demo con la informacion de la base de datos.

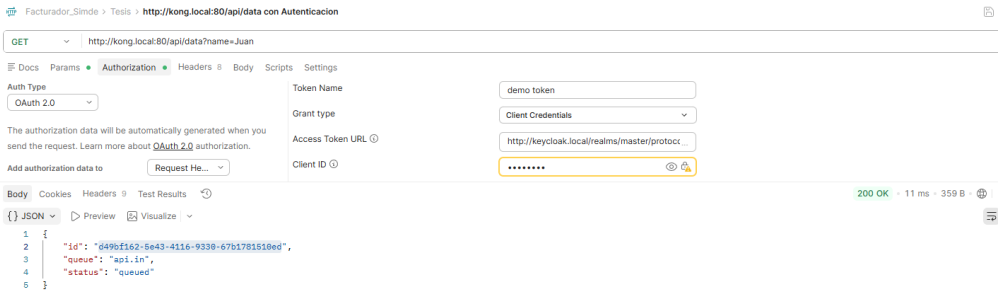


Figura 4.8: Ejecucion Api Demo

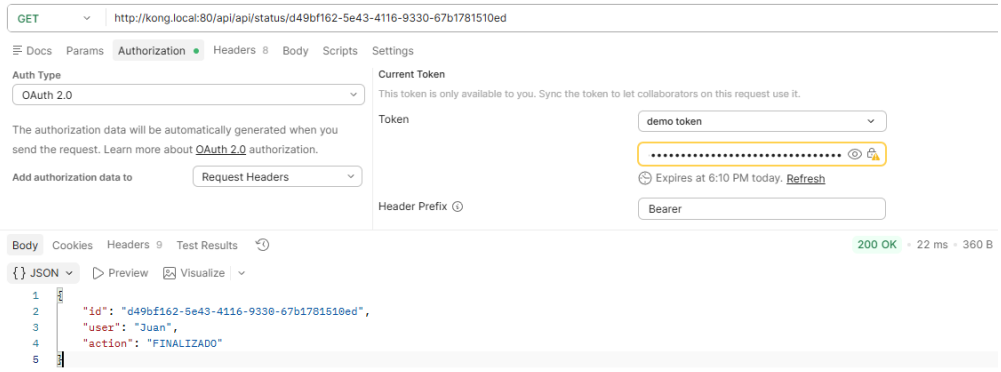


Figura 4.9: Consulta De Estado

Nota. como resultado en rabbitMQ las colas quedan en 0 ya que fueron procesados los mensajes por el worker

El resultado de la prueba es aprobada ya que se realizo integracion de una nueva funcionalidad llamada worker sin impactar o realizar desarrollo alguno sobre el api demo.

4.7. Análisis de Puntos de Sensibilidad

Los puntos de sensibilidad representan decisiones arquitectónicas cuyo cambio impacta significativamente uno o más atributos de calidad.

Tabla 4.3: Puntos de sensibilidad identificados

Decisión Arquitectónica	**Impacto**
Uso de Keycloak	Impacta seguridad y rendimiento
Uso de Kubernetes	Impacta escalabilidad y disponibilidad
Uso de RabbitMQ	Impacta interoperabilidad y rendimiento
Uso de Vault	Impacta seguridad y complejidad operativa
Uso de Prometheus y Grafana	Impacta observabilidad y consumo de recursos

4.8. Identificación de Tradeoffs

4.8.1. Tradeoff T1

Decisión: Implementación de autenticación centralizada mediante Keycloak.
Beneficio:
Incrementa la seguridad y centraliza la gestión de identidades.
Compromiso:
Introduce latencia adicional durante el proceso de autenticación.
Atributos involucrados:
Seguridad versus rendimiento.

4.8.2. Tradeoff T2

Decisión: Incorporación de monitoreo integral.
Beneficio:
Permite observabilidad completa de la plataforma.
Compromiso:
Incrementa el consumo de CPU y memoria.
Atributos involucrados:
Observabilidad versus eficiencia de recursos.

4.8.3. Tradeoff T3

Decisión: Uso de comunicación asíncrona mediante RabbitMQ.
Beneficio:
Reduce el acoplamiento entre sistemas.
Compromiso:
Aumenta la complejidad operativa y de monitoreo.
Atributos involucrados:
Interoperabilidad versus complejidad administrativa.

4.9. Riesgos Identificados y Mitigados

Tabla 4.4: Riesgos arquitectónicos

Riesgo	**Estado Inicial**	**Estado Final**
Credenciales expuestas	Alto	Mitigado mediante Vault
Punto único de falla	Alto	Mitigado mediante Kubernetes
Acoplamiento entre módulos	Alto	Mitigado mediante microservicios
Falta de integración	Alto	Mitigado mediante APIs y RabbitMQ
Ausencia de monitoreo	Alto	Mitigado mediante Prometheus y Grafana

4.10. Resultados de la Evaluación

La aplicación del método ATAM permitió verificar que la arquitectura propuesta satisface los atributos de calidad priorizados durante la fase de diseño.

Los resultados muestran mejoras significativas en:

- Seguridad.

  El sistema implementa un modelo de seguridad basado en OAuth 2.0 y OpenID Connect mediante el uso de Keycloak, el cual actúa como proveedor de identidad centralizado.

  A través de este mecanismo, los usuarios se autentican contra Keycloak y reciben un token de acceso (JWT), el cual es utilizado para autorizar el acceso a los servicios protegidos. Este enfoque permite la separación entre autenticación y autorización, garantizando que las aplicaciones no gestionen credenciales directamente.

  El uso de OAuth 2.0 asegura que el acceso a los recursos esté basado en tokens firmados y con expiración controlada, reduciendo riesgos asociados a sesiones persistentes o credenciales expuestas.

  Desde la perspectiva ATAM, este mecanismo contribuye directamente al atributo de seguridad, ya que permite autenticación centralizada, control de acceso basado en roles (RBAC) y validación de identidad sin exposición de credenciales en los servicios.

- Escalabilidad.

  Para La prueba se realiza consulta de los pods del namespace vault donde se evidencia que existe 1 para vault (el otro prertenece al agente injector), para posteriormente realizar el

incremento de replicas a 3 y se vuelve a consultar donde evidenciamos que incremento a un total de 3 pods para vault.

La evidencia de disponibilidad se observa en el clúster Kubernetes, donde los componentes del API Gateway Kong y su interfaz de administración Konga se encuentran desplegados en múltiples réplicas, garantizando tolerancia a fallos.

Como se muestra en la Figura 4.2, todos los pods presentan estado "Running" y un número de reinicios igual a cero, lo cual indica estabilidad operativa.

La evidencia presentada corresponde al despliegue del sistema de autenticación Keycloak dentro de un clúster de Kubernetes, el cual forma parte de la arquitectura propuesta basada en microservicios.

Se observa que el sistema se encuentra desplegado en el entorno distribuido, con múltiples instancias gestionadas por el orquestador, lo que permite garantizar la continuidad del servicio ante posibles fallos y habilita capacidades de auto-recuperación y escalabilidad horizontal.

Asimismo, el sistema está integrado con una base de datos PostgreSQL para la persistencia de usuarios y configuraciones, asegurando la correcta interoperabilidad entre componentes de la arquitectura. De igual forma, la gestión de credenciales se realiza de manera externa mediante un mecanismo de inyección de secretos, evitando el uso de información sensible directamente en la configuración de la aplicación.

Desde la perspectiva del método ATAM, esta evidencia valida principalmente los atributos de calidad de disponibilidad, interoperabilidad y seguridad, ya que demuestra la operación coordinada entre los distintos componentes del sistema, así como la capacidad de la arquitectura para mantener la funcionalidad del servicio en un entorno distribuido.

- Modificabilidad.

  La siguiente evidencia demuestra la integración entre el gestor de secretos HashiCorp Vault y el clúster de Kubernetes mediante el uso del operador External Secrets.

  Se realizó una actualización del secreto almacenado en Vault correspondiente a las credenciales de base de datos del sistema Keycloak. Esta actualización fue detectada automáticamente por el External Secrets Operator y sincronizada hacia el clúster, generando la actualización del Secret asociado sin necesidad de redeploy de los pods de la aplicación.

  La validación de la propagación se realizó consultando directamente el secreto en Kubernetes, evidenciando que los valores fueron actualizados correctamente tras la modificación en Vault. Asimismo, no se observaron reinicios en los pods de la aplicación durante el proceso, lo que confirma la continuidad del servicio.

  El tiempo de propagación entre la actualización en Vault y su reflejo en Kubernetes fue medido mediante anotación de sincronización forzada, permitiendo estimar la latencia del mecanismo de sincronización automática.

  Desde la perspectiva del análisis ATAM, este escenario valida el atributo de modificabilidad, al permitir cambios en credenciales sin impacto en la disponibilidad del sistema, así como

el atributo de disponibilidad, al no requerir reinicio de componentes para aplicar la nueva configuración.

- Observabilidad.

  La arquitectura implementa un modelo de observabilidad basado en métricas, visualización y logging mediante el uso de Prometheus y Grafana, complementado con los registros generados por los componentes desplegados en Kubernetes y la aplicación Keycloak.

  Prometheus se encarga de la recolección de métricas del clúster, incluyendo el estado de los pods, consumo de recursos y disponibilidad de servicios. Estas métricas son posteriormente visualizadas en Grafana, permitiendo un análisis centralizado del comportamiento del sistema en tiempo real.

  Este enfoque permite identificar fallos, analizar el rendimiento y evaluar el estado operativo del sistema sin intervención directa en los componentes, mejorando la capacidad de diagnóstico y monitoreo.

  Desde la perspectiva ATAM, esta implementación valida el atributo de observabilidad, ya que el sistema proporciona información suficiente para medir, visualizar y analizar su comportamiento interno de manera continua.

Asimismo, se identificaron compromisos arquitectónicos aceptables entre seguridad y rendimiento, así como entre observabilidad y consumo de recursos.

4.11. Conclusiones

La evaluación mediante ATAM permitió validar formalmente las decisiones arquitectónicas adoptadas en la solución propuesta.

Los resultados evidencian que la incorporación de Kubernetes, Kong, Keycloak, RabbitMQ, Redis y HashiCorp Vault contribuyen de manera significativa al cumplimiento de los atributos de calidad definidos en los requerimientos arquitectónicos.

De igual manera, la arquitectura presenta un nivel adecuado de flexibilidad, mantenibilidad y capacidad de evolución, alineándose con principios cloud-native y cloud-agnostic. La utilización de tecnologías abiertas como Kubernetes, Kong, Keycloak, RabbitMQ y HashiCorp Vault permite aprovechar las capacidades de la computación en la nube sin generar dependencia de un proveedor específico de infraestructura. Asimismo, la solución adopta prácticas DevSecOps y arquitecturas modernas basadas en microservicios.

Finalmente, los riesgos identificados en la arquitectura original fueron mitigados mediante mecanismos tecnológicos especializados, permitiendo una plataforma más segura, escalable, interoperable y preparada para futuras necesidades organizacionales.

Capítulo 5

Conclusiones

5.1. Conclusiones

La presente investigación tuvo como propósito diseñar una arquitectura de software para la capa de mediación de la entidad SIMDE que permitiera estandarizar y centralizar la exposición de servicios, reducir el acoplamiento entre sistemas y mejorar atributos de calidad como interoperabilidad, mantenibilidad, seguridad, confiabilidad y escalabilidad bajo un enfoque cloud-agnóstico. Los resultados obtenidos permiten afirmar que los objetivos planteados fueron alcanzados satisfactoriamente.

En primer lugar, el análisis de la arquitectura actual evidenció la existencia de integraciones basadas en accesos directos a bases de datos y desarrollos específicos para cada sistema, situación que generaba altos niveles de acoplamiento y limitaba la evolución tecnológica de la organización. A partir de este diagnóstico se identificaron los atributos de calidad críticos y se definieron los requerimientos arquitectónicos necesarios para soportar una estrategia de modernización alineada con las necesidades institucionales.

Posteriormente, se diseñó una arquitectura de mediación basada en principios cloud-native y cloud-agnostic, incorporando mecanismos de gestión centralizada de APIs, autenticación, administración de secretos, mensajería asíncrona, observabilidad y orquestación de contenedores. Esta propuesta favorece el desacoplamiento de los sistemas, la reutilización de servicios y la reducción de la dependencia tecnológica frente a proveedores específicos de infraestructura.

La implementación de un prototipo funcional permitió validar la viabilidad técnica de la arquitectura propuesta, mientras que la evaluación mediante el método ATAM demostró que las decisiones arquitectónicas adoptadas contribuyen positivamente a escenarios de disponibilidad y fiabilidad, confirmando el cumplimiento de los atributos de calidad definidos durante el proceso de investigación.

Los resultados obtenidos son consistentes con los enfoques arquitectónicos identificados en la literatura especializada sobre microservicios, arquitecturas orientadas a servicios y soluciones cloud-native, confirmando que la adopción de estándares abiertos y componentes desacoplados constituye una estrategia efectiva para mitigar problemas de interoperabilidad y dependencia tecnológica. Asimismo, la investigación aporta una propuesta aplicable a organizaciones que enfrentan desafíos similares de integración y modernización tecnológica.

Desde una perspectiva práctica, la arquitectura propuesta proporciona a SIMDE una hoja de ruta para la evolución de su ecosistema digital, facilitando la incorporación de nuevos servicios, la mejora de la gobernanza tecnológica y la adaptación a futuros requerimientos del negocio.

Entre las principales limitaciones de la investigación se encuentra la validación realizada únicamente en un entorno de desarrollo, por lo que no fue posible evaluar el comportamiento de la arquitectura bajo cargas reales de producción ni medir indicadores operativos a largo plazo. De igual forma, aspectos relacionados con costos de operación multicloud, recuperación ante desastres y automatización avanzada quedaron fuera del alcance definido para este trabajo.

Finalmente, se concluye que la adopción de principios de estandarización, desacoplamiento y portabilidad tecnológica representa un factor clave para la transformación digital sostenible de las organizaciones. Como líneas futuras de investigación se recomienda profundizar en escenarios multicloud, arquitecturas orientadas a eventos a gran escala, automatización mediante prácticas DevSecOps y aplicación de técnicas de inteligencia artificial para la gestión operativa de plataformas distribuidas.

5.2. Trabajos futuros

La arquitectura propuesta constituye una base de referencia para la modernización de la capa de mediación de SIMDE; sin embargo, su evolución abre diversas oportunidades de investigación y desarrollo. Como primera línea de trabajo futuro, se recomienda avanzar hacia una implementación organizacional completa que permita consolidar un ecosistema de servicios empresariales gobernado mediante principios de interoperabilidad, reutilización y desacoplamiento.

Asimismo, resulta pertinente extender la arquitectura hacia escenarios multicloud e híbridos, incorporando mecanismos de federación, balanceo geográfico y movilidad de cargas de trabajo entre proveedores de infraestructura. Esto permitiría profundizar en el análisis de estrategias para minimizar el riesgo de dependencia tecnológica y garantizar la continuidad operativa ante cambios en las condiciones del mercado o de los proveedores.

Otra línea de investigación consiste en la incorporación de arquitecturas orientadas a eventos a gran escala, mediante plataformas de streaming de datos y procesamiento en tiempo real, con el fin de soportar escenarios de integración más complejos y habilitar capacidades avanzadas de analítica operacional.

Se propone igualmente explorar la integración de mecanismos de inteligencia artificial aplicada a las operaciones (AIOps), orientados a la detección temprana de incidentes, análisis predictivo de fallas, optimización automática de recursos y mejora continua de la disponibilidad de los servicios.

Desde la perspectiva de seguridad, se recomienda investigar la incorporación de modelos Zero Trust, gestión avanzada de identidades federadas y mecanismos automatizados de cumplimiento normativo, fortaleciendo la protección de los activos digitales en entornos distribuidos.

A nivel metodológico, futuras investigaciones podrían realizar estudios comparativos entre diferentes arquitecturas cloud-agnósticas, evaluando métricas objetivas relacionadas con costos operativos, portabilidad, rendimiento, resiliencia y complejidad de mantenimiento. Estos estudios permitirían construir modelos de referencia que faciliten la toma de decisiones arquitectónicas en organizaciones con necesidades similares.

Finalmente, se plantea como línea de investigación la construcción de un marco de referencia arquitectónico cloud-agnóstico para entidades del sector público y organizaciones de servicios di-

gitales, integrando patrones, buenas prácticas y mecanismos de gobernanza que sirvan como guía para futuros procesos de transformación digital y modernización tecnológica.

5.3. Lecciones Aprendidas

La modernización arquitectónica requiere comprender que los problemas de integración no se resuelven únicamente mediante la incorporación de nuevas tecnologías, sino a través de una adecuada definición de principios arquitectónicos orientados al desacoplamiento, la reutilización y la estandarización.

La identificación temprana de atributos de calidad permitió orientar las decisiones de diseño hacia objetivos concretos de interoperabilidad, seguridad, mantenibilidad, confiabilidad y escalabilidad, evidenciando la importancia de considerar estos aspectos desde las etapas iniciales de la arquitectura.

El uso de tecnologías de código abierto y estándares abiertos demostró ser una estrategia efectiva para reducir la dependencia tecnológica y aumentar la flexibilidad de las soluciones empresariales frente a cambios futuros en la infraestructura.

La evaluación arquitectónica mediante ATAM evidenció la relevancia de validar las decisiones de diseño antes de una implementación productiva, permitiendo identificar riesgos, sensibilidades y puntos de mejora de manera temprana.

La construcción del prototipo permitió comprobar que una arquitectura cloud-agnóstica puede integrarse de forma efectiva mediante componentes especializados para gestión de APIs, seguridad, observabilidad y mensajería, manteniendo la independencia respecto a proveedores específicos.

Finalmente, se aprendió que la adopción de una arquitectura moderna debe ser entendida como un proceso evolutivo que involucra aspectos tecnológicos, organizacionales y culturales, donde la capacitación de los equipos y la definición de buenas prácticas son factores determinantes para garantizar el éxito de la transformación digital.

5.4. Discusión de costos

Desde una perspectiva económica, la arquitectura propuesta presenta ventajas asociadas al uso de tecnologías de código abierto, reduciendo los costos de licenciamiento asociados a plataformas propietarias. Herramientas como Kubernetes, RabbitMQ, Prometheus, Grafana, Kong Community Edition y PostgreSQL permiten implementar una solución empresarial utilizando componentes ampliamente adoptados en la industria sin requerir inversiones iniciales en licencias de software.

Sin embargo, aunque la reducción de costos de licenciamiento es significativa, la adopción de esta arquitectura implica costos relacionados con infraestructura, operación y capacitación técnica. La administración de un clúster Kubernetes requiere personal con conocimientos especializados en contenedores, redes, seguridad, monitoreo y automatización de despliegues. En ambientes productivos, también deben considerarse costos asociados a recursos computacionales, almacenamiento, respaldo, alta disponibilidad y servicios administrados en la nube.

Para un escenario basado en servicios cloud como Amazon Elastic Kubernetes Service (EKS), los costos dependerán principalmente del tamaño del clúster, cantidad de nodos, capacidad de procesamiento requerida y servicios complementarios utilizados. No obstante, la arquitectura propuesta permite optimizar estos recursos mediante escalamiento dinámico y asignación eficiente de capacidades según la demanda.

En conclusión, la solución propuesta representa una alternativa técnicamente viable y económicamente sostenible para modernizar la arquitectura tecnológica de la organización, siempre que exista una adecuada planificación de infraestructura, capacitación del equipo técnico y una estrategia gradual de adopción.

Bibliografía

(2011). Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models.

Apache Software Foundation (2023). Apache http server documentation.

Ayas, H. M., Leitner, P., and Hebig, R. (2023). An empirical study of the systemic and technical migration towards microservices. *Empirical Software Engineering*, 28(85).

Bass, L., Clements, P., and Kazman, R. (2012). *Software Architecture in Practice*. Addison-Wesley, 3rd edition.

Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley, 4 edition.

Bass, L., Clements, P., and Kazman, R. (2022). *Software Architecture in Practice*. Addison-Wesley, 4 edition.

Billawa, P. et al. (2022). Security of microservice applications: Challenges and best practices. *arXiv*.

Bogner, J. et al. (2022). Designing microservice systems using patterns: An empirical study on quality trade-offs. *arXiv*.

Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57.

Capilla, R., Diaz-Pace, J. A., and Ramirez, Y. (2025). Supporting architecture evaluation for atam scenarios with large language models. *arXiv*.

Clements, P., Kazman, R., and Klein, M. (2002). *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley.

Clements, P., Kazman, R., and Klein, M. (2010). *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley.

Cloud Native Computing Foundation (2024). Cloud native definition v1.0.

Di Francesco, P. et al. (2024). Monitoring and observability for cloud-native microservices: A systematic review. *Journal of Systems and Software*, 209:111925.

González, L. et al. (2021). Microservices architecture: Key concepts and benefits. *Journal of Software Engineering*, 15(2):45–60.

Grafana Labs (2024). Grafana documentation.

Hardt, D. (2012). The oauth 2.0 authorization framework. RFC 6749.

HashiCorp (2024). Vault documentation.

Hohpe, G. and Woolf, B. (2004). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

IBM Security (2023). Cost of a data breach report 2023. Technical report, IBM Security.

Kazman, R., Klein, M., and Clements, P. (2000). Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie Mellon University.

Kosinska, J. et al. (2023). Towards the observability of cloud-native applications: The overview of the state-of-the-art. *IEEE Access*, 11:65018–65041.

Kreps, J. (2011). *Kafka: A Distributed Messaging System for Log Processing*. LinkedIn Engineering.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report SP 800-145, National Institute of Standards and Technology.

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239).

Morris, K. (2020). *Infrastructure as Code*. O'Reilly Media, 2 edition.

Myrbakken, H. and Colomo-Palacios, R. (2022). Devsecops: A multivocal literature review. *Software: Practice and Experience*, 52(10):2218–2247.

Nasab, A. R., Shahin, M., and Liang, P. (2023). An empirical study of security practices for microservices systems. *Journal of Systems and Software*, 198:111563.

Newman, S. (2021). *Building Microservices*. O'Reilly Media, 2 edition.

OpenTelemetry Community (2024). Opentelemetry documentation.

OWASP Foundation (2021). Owasp top 10.

Pahl, C. (2015). Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31.

Petcu, D. (2013). Portability and interoperability between clouds: Challenges and case study. *Journal of Cloud Computing*, 2(1):11.

Pontarolli, R., Bianchi, T., and Cazarini, E. (2023). Microservices-based architecture for industry 4.0 systems: A systematic review. *Applied System Innovation*, 6(2):35.

PostgreSQL Global Development Group (2023). Postgresql documentation.

Prometheus Authors (2024). Prometheus documentation.

Ramaswamy, S. (2024). Observability in microservices: Metrics, traces and logs for devops-driven quality assurance. *International Journal of Advanced Computer Science*.

Rescorla, E. (2018). The transport layer security (tls) protocol version 1.3. RFC 8446.

Richards, M. (2020). *Fundamentals of Software Architecture*. O'Reilly Media.

Richardson, C. (2018). *Microservices Patterns*. Manning Publications.

Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. NIST Special Publication 800-207.

Sakimura, N. et al. (2014). Openid connect core 1.0.

Soldani, J. et al. (2023). The evolution of cloud-native systems: A systematic mapping study. *SN Computer Science*, 4(3).

Taibi, D. et al. (2023). How do microservices evolve? an empirical analysis of changes in open-source microservice repositories. *Journal of Systems and Software*, 204:111788.

Toosi, A. N., Calheiros, R. N., and Buyya, R. (2014). Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys*, 47(1):7.

Ullah, I. et al. (2023). Container orchestration platforms in cloud computing: A systematic literature review. *Journal of Cloud Computing*, 12(1).

Venckauskas, A. et al. (2023). Secure authentication and authorization for microservice architectures. *Applied Sciences*, 13(5).

VMware (2024). Rabbitmq documentation.

ZargarAzad, E. and Ashtiani, M. H. (2023). Auto-scaling of microservices in cloud-native applications: A reinforcement learning approach. *Journal of Grid Computing*, 21(4).

Zhang, Q., Chen, M., Li, L., et al. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.