Pontificia Universidad Javeriana Cali
Facultad de Ingeniería.
Maestría en Ingeniería de Software
Proyecto de Grado.

Diseño de una Arquitectura de Referencia para Hubs de Pago en el contexto del Comercio Electrónico mediante la Adopción de Buenas Prácticas de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework.

Julián Andrés Guerrero Ome

Director(a): Royer David Estrada Esponda

25 de julio de 2025

Santiago de Cali, 25 de julio de 2025.

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 "Diseño de una Arquitectura de Referencia para Hubs de Pago en el contexto del Comercio Electrónico mediante la Adopción de Buenas Prácticas de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework." realizado por el estudiante de Magister en Ingeniería de Software Julián Andrés Guerrero Ome (cod: 8992120), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Royer David Estrada Esponda

Santiago de Cali, 25 de julio de 2025.

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 "Diseño de una Arquitectura de Referencia para Hubs de Pago en el contexto del Comercio Electrónico mediante la Adopción de Buenas Prácticas de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework." 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,

Julián Andrés Guerrero Ome
Código: 8992120

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Diseño de una Arquitectura de Referencia para Hubs de Pago en el contexto del Comercio Electrónico mediante la Adopción de Buenas Prácticas de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework.

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Arquitectura de Software
3. Tipo de proyecto: Investigación
4. Estudiante: Julián Andrés Guerrero Ome
5. Correo electrónico: jguerrom9@javerianacali.edu.co
6. Dirección y teléfono: Carrera 19 No. 11N-33 - Armenia (Quindío) Cel: 3233209398
7. Director: Royer David Estrada Esponda
8. Vinculación del director: Profesor Hora Cátedra Postgrado
9. Correo electrónico del director: royer.estrada@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): Arquitectura de Referencia, Hubs de pago, Comercio Electrónico, AWS (Amazon Web Services), Well-Architected Framework, Eficiencia de rendimiento, Seguridad de Transacciones en línea
14. ODS que aplica al proyecto (Agenda 2030):
15. Fecha de inicio: 01 de diciembre de 2024
16. Resumen: Este proyecto propone el diseño de una arquitectura de referencia para hubs de pago en el contexto del comercio electrónico, integrando las mejores prácticas documentadas para los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework para garantizar que el diseño arquitectónico cumpla con los estándares más altos de rendimiento y seguridad. La arquitectura aborda el aumento significativo en las transacciones digitales y la necesidad de sistemas de pago rápidos, convenientes, seguros y adaptables a las demandas fluctuantes. Integrando las capacidades de infraestructura en la nube de AWS,

esta arquitectura pretende optimizar las operaciones de pago mientras asegura que medidas de seguridad robustas estén implementadas. El enfoque del proyecto involucra un examen detallado de los desafíos actuales y los requisitos técnicos, utilizando los servicios de AWS para satisfacer eficazmente las necesidades de eficiencia de rendimiento y seguridad. El resultado anticipado es una arquitectura de pago escalable, segura y eficiente que se adhiere a las mejores prácticas y requisitos regulatorios, estableciendo un punto de referencia en el panorama de pagos digitales.

Agradecimientos

El camino para completar esta tesis ha sido un viaje de gran aprendizaje, uno que no habría podido recorrer solo. Por ello, deseo expresar mi más profunda gratitud.

A mi director de tesis, Royer Estrada, por su experta orientación y por creer en este proyecto desde su inicio. Sus aportes fueron fundamentales para darle forma y rigor académico.

Agradezco a mis compañeros y excompañeros de trabajo, expertos en AWS, por su invaluable colaboración técnica. Asimismo, extiendo mi gratitud a los profesores de la maestría por brindarme las herramientas y la inspiración necesarias para llegar hasta aquí. Mi reconocimiento también para Luisa Fernanda Rincón, Directora de Posgrados de la Javeriana Cali, por su fundamental apoyo administrativo.

Sobre todo, mi más profundo y sentido agradecimiento es para mi esposa, Camila Berrío. Gracias por tu paciencia infinita, tu aliento en los momentos de duda y por ser mi pilar constante. Este logro es tan tuyo como mío.

Resumen

Este proyecto propone el diseño de una arquitectura de referencia para hubs de pago en el contexto del comercio electrónico, integrando las mejores prácticas documentadas para los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework para garantizar que el diseño arquitectónico cumpla con los estándares más altos de rendimiento y seguridad. La arquitectura aborda el aumento significativo en las transacciones digitales y la necesidad de sistemas de pago rápidos, convenientes, seguros y adaptables a las demandas fluctuantes. Integrando las capacidades de infraestructura en la nube de AWS, esta arquitectura pretende optimizar las operaciones de pago mientras asegura que medidas de seguridad robustas estén implementadas. El enfoque del proyecto involucra un examen detallado de los desafíos actuales y los requisitos técnicos, utilizando los servicios de AWS para satisfacer eficazmente las necesidades de eficiencia de rendimiento y seguridad. El resultado anticipado es una arquitectura de pago escalable, segura y eficiente que se adhiere a las mejores prácticas y requisitos regulatorios, estableciendo un punto de referencia en el panorama de pagos digitales.

Palabras Clave: Arquitectura de Referencia, Hubs de pago, Comercio Electrónico, AWS (Amazon Web Services), Well-Architected Framework, Eficiencia de rendimiento, Seguridad de Transacciones en línea

Abstract

This project proposes the design of a reference architecture for payment hubs in the context of e-commerce, integrating the best practices documented for the Performance Efficiency and Security pillars of the AWS Well-Architected Framework to ensure that the architectural design meets the highest standards of performance and security. The architecture addresses the significant increase in digital transactions and the need for payment systems that are fast, convenient, secure, and adaptable to fluctuating demands. By leveraging AWS cloud infrastructure capabilities, this architecture aims to optimize payment operations while ensuring robust security measures are implemented. The project's approach involves a detailed examination of current challenges and technical requirements, utilizing AWS services to effectively meet performance efficiency and security needs. The anticipated result is a scalable, secure, and efficient payment architecture that adheres to best practices and regulatory requirements, establishing a benchmark in the digital payments landscape.

Keywords: Reference Architecture, Payment Hubs, E-commerce, AWS (Amazon Web Services), Well-Architected Framework, Performance Efficiency, Online Transactions Security

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

El paisaje del comercio electrónico ha experimentado un crecimiento exponencial, impulsado por avances tecnológicos y una adopción global masiva de soluciones de pago digitales. Este dinamismo ha llevado a una evolución en las expectativas de los consumidores, quienes demandan transacciones no solo rápidas y convenientes, sino también extremadamente seguras y confiables. En respuesta a este escenario, emerge la necesidad de desarrollar arquitecturas de pago que no solo gestionen la eficiencia operativa y la escalabilidad, sino que también fortalezcan la seguridad y la adaptabilidad ante demandas fluctuantes y amenazas emergentes.

En este contexto, los hubs de pago se presentan como intermediarios cruciales que procesan transacciones de múltiples fuentes, simplificando la infraestructura de pago y centralizando las operaciones. Sin embargo, la centralización aumenta los riesgos asociados a la seguridad de los datos y la integridad de las transacciones, lo que hace imperativo el desarrollo de sistemas que integren prácticas de seguridad desde su concepción.

1.1.1. Presentación del Tema

Este informe presenta el diseño de una arquitectura de referencia para hubs de pago en el ámbito del comercio electrónico, utilizando las buenas prácticas documentadas del AWS Well-Architected Framework como guía para garantizar una implementación que no solo cumpla con los estándares actuales de eficiencia y seguridad, sino que también se alinee con una visión proactiva hacia la evolución futura de las tecnologías de pago. La arquitectura buscará maximizar los beneficios de la infraestructura en la nube de AWS, optimizando la eficiencia del rendimiento y asegurando una protección robusta contra fraudes y otros riesgos de seguridad, lo que será crucial para sustentar el crecimiento continuo y la confiabilidad del comercio electrónico global.

1.1.2. Preguntas de Investigación

- ¿Cómo puede diseñarse una arquitectura de referencia para hubs de pago en AWS que cumpla con los requisitos actuales de eficiencia y seguridad del comercio electrónico, adoptando las mejores prácticas definidas en el Well-Architected Framework para los pilares de Eficiencia del Rendimiento y Seguridad?
- ¿Cuáles son los principales desafíos y requisitos técnicos y operativos para implementar un hub de pago en AWS en el contexto actual del comercio electrónico, y cómo pueden estos

influir en el diseño de la arquitectura de referencia?

- ¿De qué manera se pueden utilizar las capacidades y servicios específicos de AWS para abordar y superar los desafíos identificados en la eficiencia operativa y la seguridad de los hubs de pago en el ámbito del comercio electrónico?

- ¿Cómo se puede proponer y validar una arquitectura de referencia que integre las mejores prácticas del Well-Architected Framework para optimizar las operaciones de pago en términos de rendimiento y asegurar la protección contra fraudes y otros riesgos de seguridad, especialmente en transacciones de comercio electrónico?

1.1.3. Relevancia e Impacto

Esta investigación contribuirá significativamente al campo del comercio electrónico, proporcionando una arquitectura de referencia para hubs de pago que mejore la eficiencia operativa y la seguridad de las transacciones. El impacto potencial incluye la optimización de operaciones de pago, la reducción del riesgo de fraude y el establecimiento de un punto de referencia en el panorama de pagos digitales.

1.1.4. Estructura de la Tesis

El presente documento se organiza en ocho capítulos que guían al lector de manera progresiva a través del proceso de investigación, desde la fundamentación del problema hasta la validación de la solución arquitectónica propuesta.

- **Capítulo 1: Introducción.** Se presenta el contexto del proyecto, se define el problema de investigación, se establecen los objetivos y se delimita el alcance. Además, se introduce la justificación y la metodología de investigación a un alto nivel.

- **Capítulo 2: Marco de Referencia.** Se establecen las bases teóricas y conceptuales del estudio, abordando temas clave como la arquitectura de software, los hubs de pago y el AWS Well-Architected Framework. Se complementa con una revisión del estado del arte para contextualizar la investigación.

- **Capítulo 3: Metodología de Investigación.** Se detalla en profundidad el enfoque de Investigación de Ciencia del Diseño (DSR) adoptado, describiendo las fases, las técnicas de recolección de datos y los procedimientos específicos para la evaluación y validación del artefacto.

- **Capítulo 4: Identificación de Desafíos y Requisitos.** En respuesta al primer objetivo específico, este capítulo identifica los desafíos técnicos y operativos, y define los requisitos funcionales y no funcionales para un hub de pagos en AWS, basándose en la literatura, análisis de mercado y entrevistas con expertos.

- **Capítulo 5: Evaluación de servicios de AWS.** Se aborda el segundo objetivo específico, evaluando cómo las capacidades y servicios de AWS pueden ser utilizados para resolver los desafíos y cumplir con los requisitos identificados, con un enfoque en los pilares de Eficiencia del Rendimiento y Seguridad.
- **Capítulo 6: Propuesta de Arquitectura de Referencia.** Se presenta el artefacto central de la tesis: el diseño detallado de la arquitectura de referencia. Se describen sus principios, estrategias y componentes utilizando el modelo C4, materializando la solución a los problemas planteados.
- **Capítulo 7: Validación de la arquitectura propuesta.** Este capítulo se dedica a la validación empírica de la arquitectura mediante una Prueba de Concepto (PoC). Se detallan el diseño de la PoC, las hipótesis de validación, la metodología de pruebas y el análisis crítico de los resultados de rendimiento y seguridad.
- **Capítulo 8: Conclusiones y Trabajos Futuros.** Finalmente, se sintetizan los hallazgos de la investigación, se discuten las contribuciones y limitaciones del estudio, y se proponen líneas de trabajo futuro para extender o profundizar en la solución desarrollada.

1.2. Definición del problema

1.2.1. Planteamiento del problema

El incremento en la adopción de tecnologías digitales y la expansión del comercio electrónico Capgemini (2023) han transformado las expectativas de los consumidores respecto a las transacciones en línea, demandando métodos de pago que no solo sean rápidos y convenientes, sino también altamente seguros y confiables. Este cambio en el panorama del comercio ha resultado en un volumen sin precedentes de transacciones en línea, presentando desafíos significativos para las plataformas de pago en términos de escalabilidad y seguridad. En este escenario, los hub's de pago emergen como una solución vital, actuando como intermediarios que procesan transacciones de múltiples fuentes, simplificando la infraestructura de pago para comerciantes y proveedores de servicios. No obstante, la centralización de las operaciones de pago eleva los riesgos asociados a la seguridad de los datos y la integridad de las transacciones, haciendo imperativo el desarrollo de sistemas que no solo sean eficientes sino inherentemente seguros y capaces de adaptarse a demandas fluctuantes Desai (2009).

Las experiencias de adaptación a las nuevas tecnologías de pago por parte de entidades financieras y comercios electrónicos resaltan la variabilidad en el éxito de estas plataformas Bionducci et al. (2023), con algunos casos mostrando mejoras significativas en la eficiencia de las operaciones, mientras que otros enfrentan obstáculos en la integración de sistemas existentes y la gestión de amenazas emergentes . La adopción de infraestructuras basadas en la nube, como las ofrecidas por AWS, presenta una oportunidad para superar estos desafíos, aprovechando la flexibilidad, escalabilidad y avanzadas capacidades de seguridad que estas plataformas proporcionan Wewege et al. (2020)

A pesar de los beneficios potenciales, muchas organizaciones aún luchan con la tarea de diseñar hub's de pago que cumplan con los requisitos de eficiencia y seguridad, encontrándose con barreras para la implementación que incluyen la falta de claridad en las prácticas recomendadas y la ausencia de un marco de referencia estructurado. Esta brecha entre las capacidades tecnológicas disponibles y la habilidad para implementarlas efectivamente resalta la necesidad crítica de una arquitectura de referencia que guíe el desarrollo de hub's de pago en AWS, asegurando que estos sistemas no solo satisfagan las demandas actuales del mercado sino que también estén preparados para adaptarse a las futuras evoluciones del paisaje de los pagos electrónicos.

1.3. Objetivos del proyecto

1.3.1. Objetivo General

Diseñar una arquitectura de referencia para hubs de pago en AWS que cumpla con los requisitos actuales de eficiencia y seguridad del comercio electrónico, adoptando las mejores prácticas definidas en el Well-Architected Framework para los pilares de Eficiencia del rendimiento y Seguridad.

1.3.2. Objetivos específicos

1. Identificar los principales desafíos y requisitos técnicos y operativos para implementar un hub de pagos en AWS en el contexto actual del comercio electrónico.

2. Evaluar cómo se pueden utilizar las capacidades y servicios específicos de AWS para abordar y superar los desafíos identificados en la eficiencia de rendimiento y la seguridad de los hubs de pago en el ámbito del comercio electrónico.

3. Proponer y validar una arquitectura de referencia que integre las mejores prácticas del Well-Architected Framework para optimizar las operaciones de pago en términos de rendimiento y asegurar la protección contra fraudes y otros riesgos de seguridad, especialmente en transacciones de comercio electrónico.

1.4. Delimitaciones y alcances

Este trabajo se centrará en el diseño de una arquitectura de referencia para hubs de pago en el contexto del comercio electrónico utilizando AWS, aplicando exclusivamente las prácticas recomendadas del Well-Architected Framework en los pilares de Eficiencia del Rendimiento y Seguridad. El alcance incluye:

- La recopilación y análisis de requisitos específicos para hub's de pago en el contexto del comercio electrónico.

- La evaluación de las herramientas y servicios de AWS que pueden contribuir a una arquitectura de pago eficiente y segura.

- El diseño de una arquitectura de referencia que incorpore las mejores prácticas del Well-Architected Framework, enfocándose únicamente en los pilares de Seguridad y Eficiencia del Rendimiento.
- La creación de documentación detallada y recomendaciones para la implementación.
- La realización de una prueba de concepto para validar el rendimiento y la seguridad de la arquitectura propuesta que incluiría: Pruebas de carga y estrés para evaluar la escalabilidad y capacidad de respuesta de la arquitectura bajo condiciones de alta demanda y evaluaciones de seguridad para identificar vulnerabilidades, pruebas de penetración y revisión de políticas de seguridad para asegurar que los datos y las transacciones estén protegidos contra fraudes y amenazas.
- Este trabajo no abarcará la implementación práctica completa de la arquitectura propuesta ni el análisis detallado de costos asociados a la implementación en AWS dado que se priorizan únicamente los pilares de Eficiencia del Rendimiento y Seguridad del Well-Architected Framework. El foco principal del trabajo son las fases de diseño y planificación, proporcionando una base sólida para futuras fases de implementación y optimización.

1.5. Justificación del trabajo de grado

En términos de conveniencia y utilidad, el diseño de una arquitectura de referencia para hub's de pago se alinea con la necesidad del mercado de ofrecer métodos de pago electrónicos sofisticados y accesibles, esenciales para el ecosistema de comercio electrónico en rápida evolución. Los sistemas de pago electrónicos son menos costosos y ofrecen mayor eficiencia administrativa que los métodos tradicionales como lo el efectivo y los cheques. Además, proporcionan una visibilidad mejorada, que es crucial para la gestión de flujos de efectivo, presupuestos y, más significativamente, para cumplir con los requisitos de cumplimiento normativo.

Existe una infraestructura tecnológica robusta disponible a través de AWS que puede soportar el desarrollo e implementación exitosos de hub's de pago, con recursos significativos dedicados a la eficiencia operativa y la seguridad de las transacciones. La implementación de soluciones basadas en la nube como AWS también sugiere una menor inversión de capital en comparación con las infraestructuras de TI tradicionales, lo que la hace más accesible para una amplia gama de empresas. Lo anterior nos brinda la viabilidad requerida para este trabajo de investigación

En relación al impacto de la investigación podemos listar los siguientes puntos:

1. **Relevancia social:** La transición a pagos electrónicos es una respuesta a la creciente demanda de soluciones de pago más rápidas, seguras y convenientes en el comercio electrónico, beneficiando tanto a consumidores como a empresas. La mejora en la eficiencia de las transacciones también tiene el potencial de fortalecer la economía digital.

2. **Valor teórico:** Este trabajo amplía el conocimiento existente sobre la integración de la arquitectura de pagos en la nube y el Well-Architected Framework, ofreciendo un modelo de referencia para futuros desarrollos e investigaciones.

3. **Justificación práctica:** El diseño propuesto puede ayudar a las empresas a superar la inercia relacionada con la transición de los pagos tradicionales a los electrónicos, abordando tanto los desafíos operativos como los de seguridad. La aplicación de las mejores prácticas y el aprovechamiento de las capacidades de AWS pueden resultar en ahorros monetarios, menor administración, mayor transparencia y una mejora en la capacidad de cumplimiento.

En resumen, la justificación para el desarrollo de una arquitectura de referencia para hub's de pago en AWS descansa en su relevancia estratégica y práctica para el sector del comercio electrónico, la viabilidad proporcionada por las avanzadas capacidades tecnológicas de AWS, y el impacto significativo que puede tener en la eficiencia operativa y la seguridad de las transacciones de pago.

1.6. Metodología de la investigación

Para alcanzar los objetivos de este proyecto, que culminan en la creación de un artefacto tecnológico —una arquitectura de referencia—, se adoptó el paradigma de la

Investigación de Ciencia del Diseño (Design Science Research, DSR). Este enfoque se centra en resolver problemas prácticos a través del diseño, la construcción y la evaluación de soluciones innovadoras.

El proceso de investigación se estructuró en tres fases secuenciales, alineadas con los objetivos específicos:

Fase 1: Análisis y Definición del Problema: Se identificaron los desafíos y requisitos de los hubs de pago mediante una revisión de la literatura y consultas con expertos.

Fase 2: Diseño y Desarrollo del Artefacto: Se diseñó la arquitectura de referencia en AWS, aplicando los principios del Well-Architected Framework.

Fase 3: Demostración y Evaluación: Se validó la arquitectura a través de una Prueba de Concepto (PoC), evaluando su rendimiento y seguridad con métricas cuantitativas y cualitativas.

El detalle completo sobre el marco de DSR, los instrumentos utilizados y los procedimientos de validación se presenta exhaustivamente en el Capítulo 3.

1.7. Resultados obtenidos

La ejecución de la metodología de Investigación de Ciencia del Diseño, alineada con los tres objetivos específicos de esta tesis, ha culminado en la generación de un conjunto de artefactos y hallazgos validados que responden directamente a la pregunta de investigación. Los principales resultados obtenidos son:

- **Una Arquitectura de Referencia para Hubs de Pago en AWS, validada y alineada con el Well-Architected Framework.** El resultado central de esta investigación es el diseño detallado de una arquitectura de referencia modular, serverless y segura. La arquitectura, documentada mediante el modelo C4 y organizada en capas funcionales (Exposición, Orquestación, Lógica de Negocio, Persistencia, Seguridad y Observabilidad), integra las mejores prácticas de los pilares de Eficiencia del Rendimiento y Seguridad.

- **Validación cuantitativa del rendimiento y la escalabilidad de la arquitectura.** Mediante una Prueba de Concepto (PoC), se validaron empíricamente las hipótesis de rendimiento. Los resultados demuestran que la arquitectura es capaz de:

  - *Procesar una alta carga transaccional:* Sostuvo un promedio de 498.7 Transacciones Por Segundo (TPS) con una tasa de éxito del 99.98 %.
  - *Mantener una baja latencia bajo estrés:* La latencia en el percentil 99 (p99) se mantuvo en 289 ms durante la prueba de pico de carga, cumpliendo el objetivo de ser inferior a 300 ms.
  - *Escalar de forma elástica y automática:* Los componentes serverless (AWS Lambda y Amazon DynamoDB) escalaron sin intervención manual para gestionar el aumento de la demanda, validando el requisito de escalabilidad automática (RNF01).

- **Validación cualitativa de la postura de seguridad de la arquitectura.** La PoC también confirmó la correcta implementación de controles de seguridad críticos, demostrando:

  - *Defensa en profundidad efectiva:* Se validó que el firewall de aplicaciones web (AWS WAF) bloquea peticiones maliciosas y que los controles de acceso (AWS IAM) aplican correctamente el principio de mínimo privilegio.
  - *Protección de datos y auditabilidad:* Se confirmó que los datos en reposo son cifrados utilizando claves gestionadas por el cliente (AWS KMS) y que todas las interacciones criptográficas son registradas de forma inmutable en AWS CloudTrail, cumpliendo con requisitos clave de normativas como PCI DSS.

- **Un catálogo estructurado de desafíos, requisitos y soluciones para Hubs de Pago.** Como resultado de los objetivos iniciales, se generaron artefactos de conocimiento que fundamentan la arquitectura:

  - *Identificación y priorización de desafíos:* Se documentó un conjunto de desafíos técnicos y operativos clave (ej. D2: Escalabilidad, D3: Seguridad, D8: Baja Latencia) y se priorizaron según su impacto en el negocio.
  - *Mapeo estratégico de requisitos a servicios AWS:* Se elaboró una matriz de alineación que conecta los requisitos funcionales (RF), no funcionales (RNF) y operativos (RO) con los servicios de AWS más idóneos para abordarlos, justificando cada elección tecnológica.

Estos resultados, en conjunto, no solo proporcionan un diseño arquitectónico teóricamente sólido, sino que ofrecen evidencia tangible de su viabilidad, rendimiento y seguridad, constituyendo una guía práctica y validada para el desarrollo de hubs de pago modernos en el ecosistema de AWS.

Capítulo 2

Marco de Referencia

2.1. Marco Teórico

2.1.1. Bases Teóricas

2.1.1.1. Arquitectura de Software

La arquitectura de software se refiere al conjunto de estructuras significativas dentro de un sistema, que comprende los elementos de software, sus propiedades externamente visibles, y las relaciones entre ellos. Esta definición proviene de la obra de Bass, Clements y Kazman, quienes establecen que la arquitectura involucra tanto la organización funcional del sistema como la selección de componentes estructurales y sus interfaces con otros componentes, de modo que se cumplan los requerimientos de rendimiento y mantenibilidad Bass et al. (2012) Según estos autores, la arquitectura no sólo afecta la funcionalidad y el rendimiento del sistema sino también la capacidad de desarrollo y mantenimiento del mismo, aspectos cruciales para adaptarse a cambios futuros y nuevas demandas.

Por otro lado, Shaw and Garlan (1996) argumentan que la arquitectura de software proporciona un plano para el sistema, ofreciendo una visión abstracta que permite analizar las decisiones antes de que el sistema esté completamente implementado. Esta visión se centra en cómo los componentes y actividades fundamentales del sistema se organizan para integrar sus funciones hacia el cumplimiento de los requerimientos técnicos y de negocio. Además, la arquitectura de software según Rozanski and Woods (2011), ayuda a manejar la complejidad inherente a los sistemas de software al permitir el análisis, la especificación y la discusión de soluciones desde diferentes perspectivas, asegurando que todos los interesados comprendan y estén alineados con la visión arquitectónica del sistema. Estas perspectivas son fundamentales en la creación de sistemas como los HUBs de pago, donde múltiples intereses y requisitos de seguridad y eficiencia deben ser balanceados meticulosamente.

La arquitectura de software es un concepto multifacético que abarca desde la estructura concreta de los sistemas hasta los patrones abstractos y principios que informan su diseño y evolución. Mark Richards ilustra este concepto en una representación donde se muestra que la arquitectura se compone de la estructura subyacente del software, en conjunto con las características arquitectónicas, decisiones de diseño y principios que guían su desarrollo Richards (2020). Este enfoque integral reconoce que la arquitectura no se limita solo a sus componentes físicos, sino que también se extiende a las decisiones estratégicas y metodologías que determinan cómo se construye y opera el software en el mundo real. Este marco completo es esencial para entender y aplicar las definiciones de estilos de arquitectura, características de arquitectura, decisiones de arquitectura y principios de

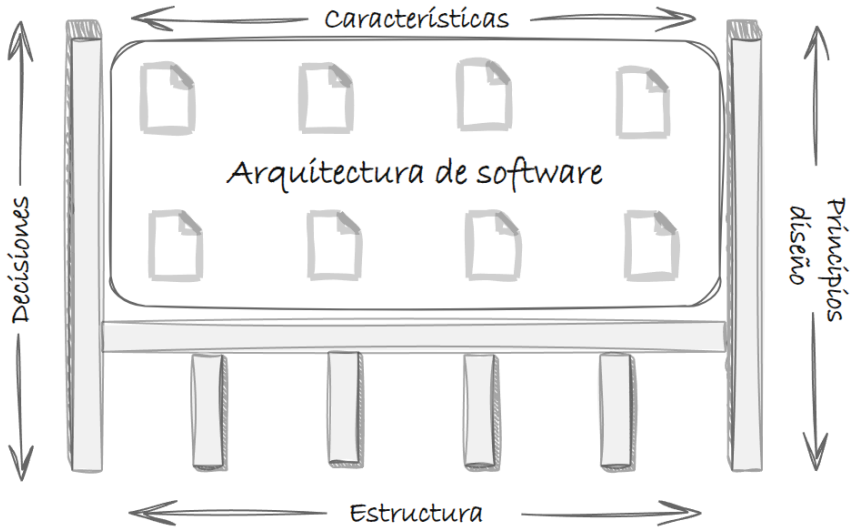

Figura 2.1: Arquitectura de Software. Elaboración propia basada en Richards (2020).

diseño de arquitectura, las cuales se detallan a continuación y son cruciales para el diseño efectivo de HUBs de pago en entornos de comercio electrónico.

Estilos de arquitectura: Los estilos de arquitectura, también conocidos como patrones arquitectónicos, se refieren a un conjunto de principios que dan forma y guían el diseño de sistemas de software. Mark Richards en su libro *Fundamentals of Software Architecture* establece que un estilo de arquitectura encapsula un vocabulario recurrente de componentes y configuraciones de conectores típicos, con restricciones específicas para lograr objetivos particulares Richards (2020). Ejemplos comunes incluyen la arquitectura en capas, orientada a servicios o basada en eventos. Cada estilo ayuda a resolver un tipo específico de problema de diseño y, según Rozanski y Woods, influencia profundamente las características de rendimiento y seguridad del sistema, facilitando a su vez la comunicación dentro del equipo de desarrollo al proporcionar un lenguaje común Rozanski and Woods (2011).

Características de arquitectura: Las características de arquitectura son los atributos esenciales que definen la calidad y la funcionalidad de un sistema de software desde una perspectiva arquitectónica. Richards subraya que estas características incluyen aspectos como la escalabilidad, la seguridad, y la capacidad de mantenimiento, y son cruciales para satisfacer los requisitos no funcionales del sistema Richards (2020). Además, Martin Fowler apunta que las características de arquitectura no solo deben cumplir con las necesidades actuales del software sino que también de-

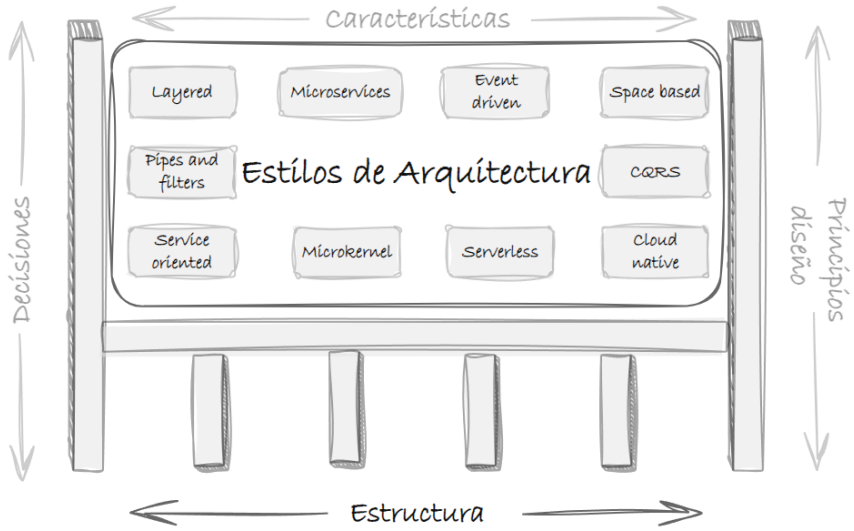

Figura 2.2: Estilos de Arquitectura de Software. Elaboración propia basada en Richards (2020).

ben considerar la adaptabilidad a futuras exigencias y cambios en el entorno tecnológico Fowler (2002). Estas características son fundamentales para la evaluación y la toma de decisiones durante el ciclo de vida del desarrollo de software.

Decisiones de arquitectura: Las decisiones de arquitectura son elecciones conscientes realizadas por los arquitectos de software que afectan cómo se estructura y se comporta un sistema de software. Según Richards, estas decisiones incluyen la selección de tecnologías, la definición de protocolos de comunicación entre componentes y la asignación de funcionalidades a esos componentes Richards (2020). Bass y sus colegas explican que estas decisiones son críticas porque establecen las bases sobre las cuales el sistema evolucionará y cómo responderá a los cambios y desafíos técnicos y de negocio Bass et al. (2012). Una buena práctica es documentar estas decisiones para facilitar la comprensión y el mantenimiento del sistema a lo largo del tiempo.

Principios de diseño de arquitectura: Los principios de diseño de arquitectura son directrices fundamentales que orientan el desarrollo de sistemas de software, asegurando que la arquitectura cumpla con los requisitos tanto funcionales como no funcionales. Richards menciona que estos principios incluyen consideraciones como la separación de preocupaciones, la minimización de la dependencia entre componentes y la maximización de la cohesión dentro de los componentes Richards (2020). Shaw y Garlan refuerzan esta idea al destacar la importancia de estos principios para crear sistemas robustos, mantenibles y escalables Shaw and Garlan (1996). Estos principios

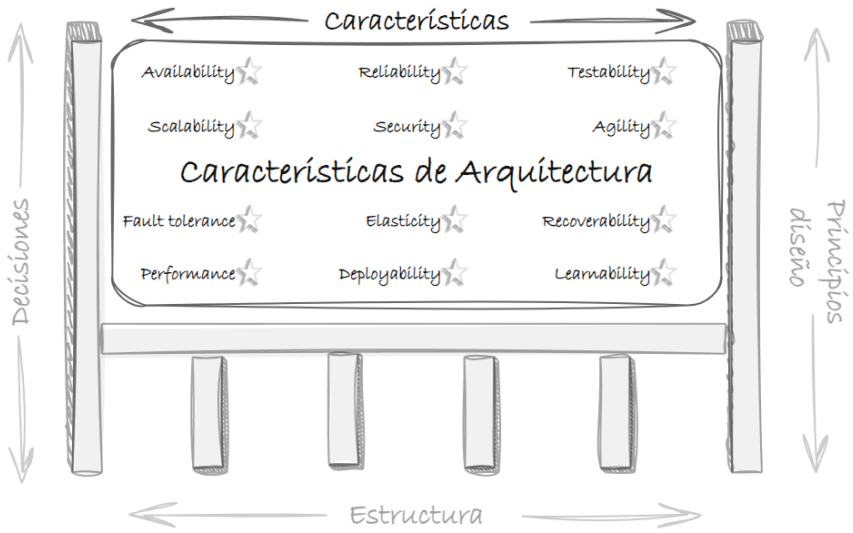

Figura 2.3: Características de Arquitectura de Software. Elaboración propia basada en Richards (2020).

no solo ayudan a los arquitectos a tomar decisiones consistentes y eficaces sino también a mitigar riesgos y mejorar la interacción entre los componentes del sistema.

2.1.1.2. Arquitectura de Referencia

Las arquitecturas de referencia constituyen una guía esencial en el desarrollo de sistemas de software, proporcionando un marco estandarizado de patrones y soluciones para construir arquitecturas concretas. Según Nakagawa et al. (2011), una arquitectura de referencia abarca el conocimiento sobre cómo diseñar arquitecturas concretas para sistemas de una determinada área de aplicación, por lo que debe considerar las reglas de negocio, estilos arquitectónicos, mejores prácticas de desarrollo, y los elementos de software que apoyan el desarrollo de sistemas para ese dominio, siempre apoyado por una terminología unificada y ampliamente comprendida. Esto indica que las arquitecturas de referencia no son meras plantillas, sino que encapsulan una visión y un entendimiento profundo de un dominio en particular, orientando el desarrollo hacia prácticas que han demostrado ser eficaces.

Las arquitecturas de referencia son particularmente útiles en organizaciones que desarrollan y mantienen familias grandes y complejas de sistemas de software que poseen necesidades arquitectónicas similares. Su implementación tiene como objetivo capturar la esencia arquitectónica de sistemas de software similares y facilitar decisiones informadas sobre si adoptar o no tal arquitectura en una organización. Los estudios empíricos han investigado tanto los beneficios como las

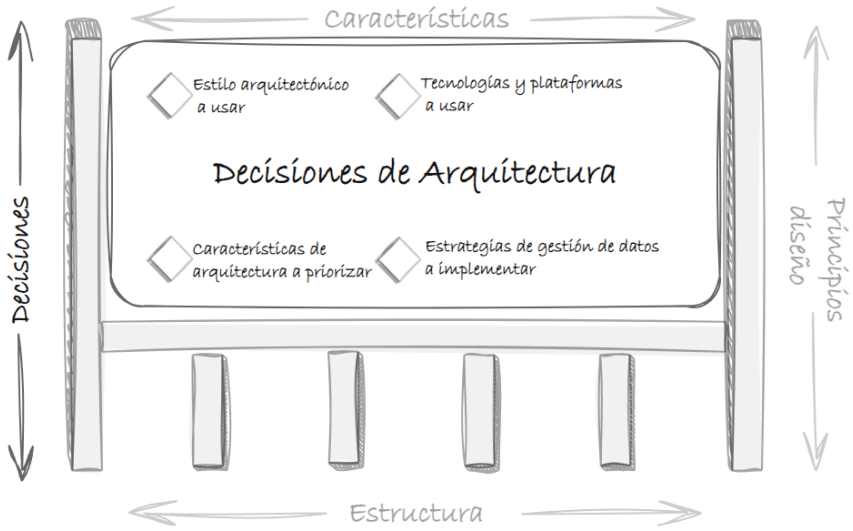

Figura 2.4: Decisiones de Arquitectura de Software. Elaboración propia basada en Richards (2020).

desventajas de adoptar arquitecturas de referencia, pero la evidencia consolidada es crucial para tomar decisiones fundamentadas en las organizaciones Martinez-Fernandez et al. (2015).

Autores como Bass et al. (2012), al abordar la documentación de arquitecturas de software, argumentan que las arquitecturas de referencia juegan un papel vital en comunicar y preservar las decisiones arquitectónicas clave. Esto se alinea con la visión de Rozanski and Woods (2011), que ven en las arquitecturas de referencia un mecanismo para gestionar la complejidad y asegurar coherencia y calidad a través de una metodología de descripción que involucra múltiples interesados y perspectivas.

En el contexto de los HUBs de pago dentro del comercio electrónico, las arquitecturas de referencia promueven la interoperabilidad, reducen los costos de desarrollo, mejoran la comunicación entre interesados, y disminuyen el tiempo de lanzamiento al mercado, pero también pueden requerir una curva de aprendizaje significativa para los desarrolladores. Estos factores son críticos al considerar la eficiencia y seguridad de los sistemas de pago. Por lo tanto, la adopción de arquitecturas de referencia en los HUBs de pago debería ser una decisión estratégica basada en una evaluación cuidadosa de estos beneficios y desafíos.

2.1.1.3. Hubs de Pago

En la era de la digitalización financiera, las instituciones enfrentan la creciente necesidad de optimizar sus operaciones y mejorar la eficiencia en el procesamiento de pagos a nivel global.

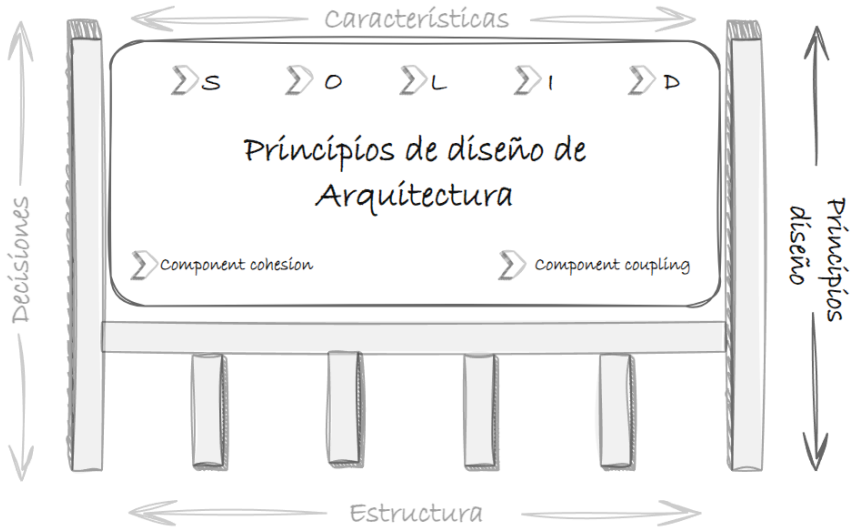

Figura 2.5: Principios de Diseño de Arquitectura de Software. Elaboración propia basada en Richards (2020).

Los desafíos incluyen la complejidad transaccional, las regulaciones cambiantes y la demanda de servicios más rápidos y seguros. Ante este panorama, los Payment Hubs emergen como una solución estratégica vital, ofreciendo una plataforma integrada que centraliza y simplifica la gestión de pagos, mientras facilita la adaptación a las normativas y mejora la eficiencia operativa Mackman and Sanders (2009) y Farrow (2011).

Definición y Propósito de los Hub's de pago: Un Payment Hub es una plataforma tecnológica avanzada que integra y procesa todas las transacciones de pago de una organización financiera de manera centralizada. Esta solución tecnológica busca consolidar diferentes sistemas de pago en un solo sistema unificado, mejorando así la eficiencia operativa y reduciendo costos Markert (2014). El propósito principal de un Payment Hub es eliminar los sistemas de pago aislados y fragmentados que tradicionalmente aumentan los costos operativos y complican la infraestructura de pago en las instituciones financieras.

Ventajas Estratégicas y Operativas: Los Payment Hub's ofrecen múltiples ventajas estratégicas, incluyendo la capacidad de integrar diversas formas de pago como transferencias electrónicas, pagos ACH, transacciones móviles y pagos transfronterizos en una única plataforma. Esto no solo reduce la redundancia operativa sino que también minimiza los errores y aumenta la eficiencia al consolidar operaciones que previamente se realizaban en múltiples plataformas Bareisis (2011). Adi-

cionalmente, los Payment Hubs facilitan una gestión más efectiva del flujo de caja y las reservas de liquidez, permitiendo a las instituciones responder mejor a las fluctuaciones del mercado financiero Williams and Moons (2010).

Arquitectura de los Payment Hubs: La arquitectura de un Payment Hub es crucial para su funcionalidad. Estos sistemas generalmente se diseñan sobre una arquitectura orientada a servicios (SOA), que permite una integración flexible y escalable de diferentes aplicaciones de pago. Según Farrow (2011), un Payment Hub típico incluye capas de interfaz de usuario, un motor de procesamiento de transacciones, integración de sistemas, bases de datos, y protocolos de seguridad y cumplimiento, cada uno diseñado para operar de manera integrada y segura, facilitando así la adaptación a los cambios regulatorios y de mercado.

Cumplimiento Regulatorio y Desafíos: En términos de regulación, los Payment Hub's proveen una base sólida que facilita el cumplimiento de normativas internacionales complejas, como es el caso de SEPA en Europa. Los Payment Hubs permiten manejar eficazmente las directrices de pagos transfronterizos, adaptándose continuamente a nuevas regulaciones sin necesidad de grandes reestructuraciones del sistema Ubaghs (2023).

Implementar un Payment Hub implica desafíos significativos, incluyendo la inversión en nuevas tecnologías y la reestructuración de procesos internos. La resistencia al cambio y la integración con tecnologías legadas son aspectos críticos que pueden complicar la transición hacia un sistema de pago centralizado. Sin embargo, los beneficios de mediano y largo plazo, como mejoras en la eficiencia, reducción de costos, y cumplimiento regulatorio eficiente, justifican estas inversiones Markert (2014).

En conclusión, los Payment Hubs se establecen como soluciones indispensables para las instituciones financieras modernas que buscan optimizar sus operaciones de pago en un entorno globalizado. A través de la centralización y automatización, los Payment Hubs no solo mejoran la eficiencia operativa sino que también fortalecen la capacidad de las instituciones para innovar y adaptarse a las demandas cambiantes del mercado financiero. La implementación de estos sistemas representa un paso crucial hacia la modernización de la infraestructura de pago y la mejora continua en la prestación de servicios financieros.

2.1.1.4. Comercio Electrónico

El comercio electrónico puede ser definido como la compra, venta, comercialización y suministro de información de productos o servicios a través de redes de comunicación digitales, principalmente Internet. Efraim et al. (2015) destacan que el comercio electrónico no sólo transforma las transacciones comerciales tradicionales, sino que también redefine las relaciones entre las organizaciones y sus consumidores, añadiendo una dimensión de interactividad y accesibilidad inmediata. Kalakota y Whinston señalan que el comercio electrónico integra recursos tecnológicos, aplicaciones de negocio y procesos de datos, permitiendo así interacciones de negocio a negocio (B2B), negocio a

consumidor (B2C), e incluso consumidor a consumidor (C2C) Kalakota and Whinston (1996).

Por otro lado, Laudon y Traver enmarcan el comercio electrónico dentro de un contexto social y de negocios, donde no solo se involucran transacciones sino también actividades como el marketing, el suministro de servicios y la gestión de relaciones con el cliente, que son transformadas por las tecnologías digitales Laudon and Traver (2020). Además, las teorías de Porter sobre la cadena de valor y la competencia se aplican al comercio electrónico para analizar cómo las empresas pueden lograr ventajas competitivas utilizando canales electrónicos, optimizando sus operaciones internas y la interacción con los proveedores y clientes Porter and ilustraciones Gibbs (2001).

Finalmente, Rayport y Jaworski subrayan la importancia de la estrategia en el comercio electrónico, donde la capacidad de una empresa para adaptar su modelo de negocio a la economía digital es crucial para su éxito y sostenibilidad a largo plazo Rayport and Jaworski (2003). El comercio electrónico, por lo tanto, representa una amalgama de estrategias comerciales, tecnologías de información y modelos de interacción cliente-empresa que colectivamente forman una parte integral de la economía moderna.

2.1.1.5. Well-Architected Framework (WAF)

El AWS Well-Architected Framework es una guía estructurada proporcionada por Amazon Web Services para ayudar a los arquitectos de la nube a construir aplicaciones y sistemas seguros, eficientes y escalables en la nube. Este marco facilita la comprensión de las decisiones de diseño de arquitectura en un contexto de nube y enfatiza la necesidad de tomar decisiones conscientes que impacten positivamente el rendimiento del negocio Services (2024). Martin Fowler sugiere que *el diseño de software es un ejercicio de balance entre muchas fuerzas competitivas* Fowler (2002), un principio que es fundamental para el enfoque del Well-Architected Framework.

El AWS Well-Architected Framework y sus seis pilares:

Excelencia Operativa: "La excelencia operativa concierne la capacidad de ejecutar y monitorear sistemas para entregar valor comercial y mejorar continuamente procesos y procedimientos"Services (2024). Adrian Cockcroft enfatiza la importancia de la automatización y la mejora continua, elementos clave para alcanzar la excelencia operativa en entornos de nube Cockcroft (2010). Estos principios aseguran que las operaciones puedan adaptarse rápidamente a cambios nuevos o cambiantes en el entorno empresarial, maximizando así la eficiencia y la eficacia de los procesos.

Seguridad: "La seguridad se centra en proteger la información y los sistemas, controlando quién puede hacer qué con los recursos de la nube y qué operaciones pueden realizar. Este pilar engloba la protección de la confidencialidad e integridad de los datos, así como la disponibilidad de los sistemas"Services (2024). Bruce Schneier destaca la importancia de incorporar la seguridad desde el inicio del diseño de sistemas, conocido como 'security by design', para crear arquitecturas robustas que anticipen y mitiguen riesgos de seguridad antes de que se conviertan en vulnerabilidades Schneier (2015). Este enfoque garantiza que las medidas de seguridad sean una parte integral de

la infraestructura y las operaciones, minimizando las amenazas y asegurando la confianza en los sistemas desplegados.

Fiabilidad: "La fiabilidad asegura la capacidad de un sistema para recuperarse de fallos de infraestructura o servicio y dinámicamente adquirir recursos computacionales para satisfacer la demanda y mitigar disrupciones"Services (2024). Este pilar se centra en la capacidad de un sistema para evitar y recuperarse rápidamente de errores para cumplir con las obligaciones de negocio y las demandas de los clientes, un enfoque que Ann M. Hickey y Alan W. Davis describen como fundamental para integrar requisitos y arquitectura de manera efectiva Hickey and Davis (2004).

Eficiencia de Rendimiento: "La eficiencia de rendimiento implica el uso de recursos informáticos eficientemente para satisfacer requisitos del sistema y mantener esa eficiencia a medida que la demanda cambia y las tecnologías evolucionan"Services (2024). La idea de adaptabilidad tecnológica resuena con la visión de Fowler sobre cómo las arquitecturas deben evolucionar junto con las innovaciones tecnológicas Fowler (2002).

Optimización de Costos: "La optimización de costos implica evitar gastos innecesarios"Services (2024). Según Erl, el diseño eficiente de costos no solo implica reducir gastos, sino también maximizar el valor producido por cada dólar gastado Erl et al. (2013). Este pilar destaca la importancia de comprender y controlar dónde se están gastando los recursos, seleccionando el tipo y número adecuado de recursos para las aplicaciones y asegurando que todo gasto aporta al negocio de manera óptima.

Sostenibilidad: Donna Haraway, en su obra sobre las conexiones entre tecnología y medio ambiente, enfatiza la importancia de una responsabilidad simbiótica.[en]tre tecnología y naturaleza Haraway (2016). El pilar de Sostenibilidad del AWS Well-Architected Framework se centra en comprender y minimizar el impacto ambiental de las cargas de trabajo en la nube. La sostenibilidad es vital para diseñar sistemas que no solo sean eficientes y económicos, sino también responsables con el medio ambiente. Este pilar añade una dimensión crucial al evaluar arquitecturas en la nube, destacando la importancia de prácticas que reduzcan el consumo de recursos y mejoren la eficiencia energética Services (2024).

2.1.1.6. Rendimiento en Sistemas de Pago en Línea

El rendimiento de los sistemas de pago en línea es crucial, ya que afecta directamente la percepción del usuario sobre la marca y la fiabilidad del servicio. Un sistema altamente performante asegura que las transacciones se completen en el menor tiempo posible, lo cual es esencial para mantener la satisfacción y la confianza del cliente. Fowler destaca la importancia de la velocidad y la eficiencia en aplicaciones empresariales modernas, argumentando que "la capacidad de una aplicación para realizar su función de manera eficiente sin demoras excesivas es crucial para su éxito"Fowler (2002).

Importancia de la Escalabilidad: La escalabilidad es un componente integral del rendimiento, especialmente en sistemas que deben manejar un volumen creciente de transacciones. Raj Jain, un experto en teoría del rendimiento de sistemas, explica que "la escalabilidad se refiere a la capacidad de un sistema para incrementar su capacidad de trabajo o poder manejar más carga de trabajo con un decremento mínimo en la eficiencia operativa"Jain (1991). En el contexto de los sistemas de pago, esto significa que la arquitectura debe ser capaz de adaptarse a picos de demanda sin comprometer la velocidad o la seguridad de las transacciones.

Latencia y Tiempo de Respuesta: La latencia y el tiempo de respuesta son métricas esenciales para evaluar el rendimiento de los sistemas de pago en línea. Según Steve McConnell, "la reducción de la latencia es a menudo más beneficiosa que mejorar el rendimiento de procesamiento"McConnell (2004), porque la percepción de rapidez es vital para la experiencia del usuario. En sistemas de pago, un tiempo de respuesta bajo es crucial para evitar la frustración del cliente durante el proceso de checkout.

Confiabilidad y Disponibilidad: La confiabilidad y la disponibilidad también son aspectos críticos del rendimiento. Daniel A. Menascé y Virgilio A.F. Almeida subrayan que ün sistema es útil solo si está disponible cuando los usuarios lo necesitan y actúa de manera confiable como se espera"Menasce et al. (2004). Para los sistemas de pago, esto significa garantizar que el sistema está operativo y es capaz de procesar transacciones de manera continua, incluso durante condiciones adversas o ataques de seguridad.

2.1.1.7. Seguridad en Sistemas de Pago en Línea

La seguridad en sistemas de pago en línea es un concepto multidimensional que abarca la protección de la infraestructura, datos y transacciones financieras contra accesos no autorizados, fraudes, y otras amenazas cibernéticas. Este aspecto es crucial no solo para proteger los activos financieros y la información personal de los usuarios, sino también para mantener la confianza en el ecosistema de comercio electrónico.

Protección de Datos: Uno de los aspectos más críticos en la seguridad de los sistemas de pago es la protección de datos sensibles, como la información de tarjetas de crédito y datos personales de los usuarios. Bruce Schneier, en su libro "Secrets and Lies: digital security in a networked world", enfatiza la importancia de la criptografía para proteger la información sensible: "La criptografía es esencial para proteger la integridad y confidencialidad de la información"Schneier (2015). Los sistemas de pago deben implementar algoritmos de cifrado robustos para asegurar que los datos estén protegidos durante la transmisión y en reposo.

Autenticación y Autorización: La autenticación y la autorización son procesos fundamentales para asegurar que solo los usuarios legítimos puedan acceder y realizar transacciones. Ross Anderson, en "Security Engineering", destaca la necesidad de sistemas de autenticación sólidos que

combinen múltiples factores: "La autenticación multifactor proporciona una capa adicional de seguridad, haciendo que sea más difícil para los atacantes comprometer cuentas"Anderson (2020). Los sistemas de pago en línea deben emplear métodos como autenticación de dos factores y biometría para mejorar la seguridad.

Gestión de Riesgos y Cumplimiento: La gestión de riesgos es vital para identificar, evaluar y mitigar potenciales vulnerabilidades y amenazas. Mitnick, en su trabajo sobre seguridad de TI, subraya la importancia de una evaluación continua de riesgos para adaptarse a las amenazas emergentes Mitnick and Simon (2002). Además, cumplir con normativas internacionales como PCI DSS es esencial para sistemas de pago, ya que establece estándares de seguridad para todas las entidades que almacenan, procesan o transmiten datos de tarjetahabientes Council (2020).

Seguridad en Infraestructura: La infraestructura que soporta los sistemas de pago debe ser segura y resiliente. Esto incluye la protección de servidores, bases de datos y redes contra intrusiones y ataques. Ed Amoroso en "Fundamentals of Computer Security"sugiere la implementación de firewalls, sistemas de detección y prevención de intrusiones, y otras tecnologías de seguridad para formar un perímetro defensivo robusto Amoroso (1994).

2.1.1.8. Infraestructura de Sistemas de Pagos y mejores prácticas

En el contexto de los sistemas de pagos en línea, la infraestructura desempeña un papel crucial en el aseguramiento de la eficiencia operativa, la seguridad de las transacciones, y la capacidad de escalar frente a la demanda fluctuante. A continuación se profundiza en las componentes clave y las mejores prácticas necesarias para establecer una infraestructura robusta y segura, apoyándose en teorías contemporáneas y prácticas recomendadas en el campo.

Componentes Clave de la Infraestructura de Sistemas de Pagos

Pasarelas de Pago: Las pasarelas de pago actúan como intermediarios críticos entre comerciantes y bancos, facilitando el procesamiento seguro y eficiente de transacciones Campbell (2010). La integración efectiva de pasarelas es esencial para manejar diversos métodos de pago y cumplir con los requisitos reglamentarios.

Alta Disponibilidad: La infraestructura debe diseñarse para ser altamente disponible y resiliente, utilizando prácticas como la duplicación de componentes críticos y la implementación de soluciones de recuperación ante desastres Hohpe and Woolf (2003).

Seguridad: La implementación de medidas de seguridad robustas, incluyendo la criptografía avanzada y la autenticación multifactor, es fundamental para proteger contra el fraude y las violaciones de datos Schneier (2015).

Integración de un Hub de pagos: Funciona como un núcleo central que procesa y gestiona las transacciones de diversos orígenes hacia múltiples destinos, asegurando la eficiencia y coherencia en el manejo de datos y transacciones Campbell (2010).

Mejores Prácticas para la Infraestructura en sistemas de pagos

- Optimización de Costos y Eficiencia Operativa: Implementar tecnologías que reduzcan costos operativos, como la transición de cheques a transacciones electrónicas, lo que puede disminuir significativamente los costos asociados con procesos manuales Campbell (2010).
- Escalabilidad y Flexibilidad: Utilizar arquitecturas que soporten fácilmente el aumento de la carga de trabajo, como los microservicios, que permiten un mejor mantenimiento y escalabilidad del sistema Fowler (2002).
- Monitoreo y Mantenimiento Continuo: Es crucial para detectar y responder a problemas técnicos o de seguridad en tiempo real, asegurando que el sistema de pagos opere de manera continua y eficiente Jain (1991).
- Integración y Automatización de Procesos: Las plataformas de pago deben integrarse sin problemas con sistemas ERP y back-offices para facilitar un procesamiento de pagos transparente y automatizado, reduciendo el riesgo de errores y fraudes Campbell (2010).

2.2. Estado del Arte

Farrow (2011) presenta un análisis detallado sobre los hub's de pago, enfocándose en su diseño y funcionalidad dentro de los sistemas bancarios. El autor introduce un modelo conceptual de un hub de pagos y explora sus beneficios empresariales, detallando cómo estos pueden mejorar la transparencia, trazabilidad y regulación de los movimientos monetarios en el sector financiero. Además, el documento aborda las capacidades funcionales de un hub de pagos, clasificándolas en servicios de proceso, negocios e integración, y argumenta cómo estos hubs reducen la complejidad de integración y apoyan las iniciativas de modernización dentro de los bancos.

Este estudio es altamente relevante para nuestro proyecto dado que ofrece un marco detallado sobre la configuración y operación de los hub's de pago, resaltando factores técnicos y empresariales que pueden influir en decisiones arquitectónicas clave. Además, proporciona una perspectiva valiosa sobre cómo estos sistemas pueden adaptarse a diversas necesidades empresariales y tecnológicas, lo cual es crucial para entender cómo implementar prácticas óptimas en la construcción de hubs de pago eficientes y seguros.

Farrow (2013) aborda las estrategias efectivas para la planificación de sistemas de pago, reconociendo los desafíos inherentes a la simplificación de la arquitectura de TI dentro de las instituciones financieras. Se presentan macro y micro estrategias para guiar la transformación sistemática de los sistemas de pago actuales hacia una arquitectura objetivo deseada, destacando la necesidad de un enfoque estructurado para minimizar riesgos y maximizar la eficiencia operativa.

Este estudio es relevante para nuestra investigación ya que proporciona un marco detallado para la planificación estratégica en el contexto de sistemas complejos de pagos. Las estrategias discutidas en el documento permiten abordar tanto la integración de servicios de pago como la gestión eficaz de la transición hacia arquitecturas más simplificadas y centralizadas, que son cruciales para el desarrollo de soluciones eficientes y seguras en el comercio electrónico.

Farrow (2012) proporciona una guía exhaustiva sobre cómo integrar eficazmente los sistemas de pago dentro de las instituciones financieras, destacando la importancia de elegir el estilo arquitectónico adecuado para procesar pagos. El autor explora diversos patrones arquitectónicos que permiten a las instituciones bancarias optimizar la integración de sus sistemas de pago. Cada patrón se describe en términos de bloques de construcción arquitectónicos y se evalúa su idoneidad en diferentes escenarios comerciales.

Esta investigación es sumamente relevante para nuestro proyecto, ya que los patrones discutidos abordan directamente la complejidad de la integración de sistemas de pago, ofreciendo soluciones para simplificar y mejorar la agilidad del negocio bancario. Los enfoques y soluciones presentados pueden informar directamente el diseño de una arquitectura de referencia eficiente y segura para hubs de pago en el comercio electrónico, asegurando que se considere la integración sistémica desde una perspectiva arquitectónica bien fundamentada.

Bareisis (2011) explora el concepto de los Payment Services Hubs (PSH), una estrategia clave para la modernización de la infraestructura de pagos en los bancos. El documento aborda la confusión existente en la terminología y define claramente lo que constituye un PSH, destacando su capacidad para manejar diversos tipos de pagos a través de una única plataforma, independientemente del tipo de instrumento, valor del pago, cliente o canal. Se enfatiza en la necesidad de una arquitectura tecnológica moderna que incluya aspectos como SOA (Arquitectura Orientada a Servicios), BPM (Gestión de Procesos de Negocio) y BAM (Monitoreo de Actividad Empresarial).

Este estudio es relevante para nuestro proyecto porque proporciona una visión integral de cómo un hub de servicios de pago puede centralizar y estandarizar las operaciones de pago de un banco, ofreciendo un enfoque sistémico que mejora la eficiencia, la visibilidad de los pagos y la gestión del riesgo. La implementación de un PSH como el descrito puede servir directamente como modelo para el desarrollo de una arquitectura de referencia para hubs de pago en el contexto del comercio electrónico, enfocándose en la eficiencia y la seguridad.

Tryfonas et al. (2001) analiza cómo integrar prácticas de seguridad en el desarrollo de sistemas de información, subrayando que muchos métodos de desarrollo de sistemas actuales no incorporan adecuadamente consideraciones de seguridad. Los autores identifican la necesidad de que las prácticas de desarrollo de sistemas se adapten para incluir seguridad de manera más efectiva, destacando que muchas técnicas de seguridad se aplican típicamente después de que los sistemas ya están desarrollados.

Este estudio es relevante para nuestro proyecto de arquitectura de hubs de pago porque enfatiza la importancia de la seguridad desde las etapas iniciales de desarrollo de los sistemas. Incorporar

consideraciones de seguridad desde el principio puede resultar en sistemas más robustos y seguros, una necesidad crítica en el diseño de sistemas de pago, donde la seguridad y la confianza son primordiales. Esta perspectiva puede guiar el enfoque de nuestro proyecto hacia la integración de prácticas de seguridad de manera que se alineen con las mejores prácticas de desarrollo y se adapten a los requisitos específicos de los sistemas de pago en comercio electrónico.

Farrow (2020) explora cómo las arquitecturas de los sistemas bancarios tradicionales se ven afectadas por la emergencia de la banca abierta, específicamente bajo la Directiva de Servicios de Pago revisada (PSD2) de la Unión Europea. El autor, discute cómo los modelos de plataformas para la banca abierta pueden facilitar interacciones de valor entre proveedores de servicios y consumidores, utilizando tecnologías en la nube para mejorar la eficiencia y la escalabilidad de estos servicios.

Este documento es altamente relevante para nuestra investigación en arquitecturas de referencia para hubs de pago, ya que proporciona una visión detallada sobre cómo las arquitecturas de IT pueden ser diseñadas para apoyar servicios bancarios innovadores y seguros bajo el modelo de banca abierta. Los conceptos y modelos presentados pueden ser fundamentales para entender cómo implementar y escalar soluciones de pago eficientes en la nube, aprovechando la flexibilidad y la capacidad de integración que ofrecen estas tecnologías modernas. Además, el enfoque en la PSD2 y los servicios de iniciación de pagos es directamente aplicable al contexto de los hubs de pago que queremos desarrollar.

Farrow (2021) se enfoca en cómo las arquitecturas de sistemas tradicionales se están adaptando a modelos de plataformas estilo Uber que ofrecen servicios globales masivos. Se detallan los patrones de middleware necesarios para apoyar soluciones de plataformas extremadamente escalables en la nube, utilizando microservicios y APIs como componentes centrales en el diseño. Se discuten también arquitecturas basadas en eventos y cómo las tecnologías modernas de middleware y nube soportan estos paradigmas.

Este documento es crucial para nuestra investigación ya que aborda directamente la arquitectura de sistemas en entornos de nube, que son fundamentales para el diseño de hubs de pago eficientes y seguros. La exploración de patrones avanzados de migración de aplicaciones y su implementación en la nube puede proporcionar un marco vital para diseñar nuestra arquitectura de referencia, asegurando que pueda escalar de manera efectiva y gestionar las demandas de un entorno de comercio electrónico dinámico. Además, la integración de consideraciones de seguridad y eficiencia en estos patrones es esencial para nuestro enfoque en la eficiencia y la seguridad en el diseño de la arquitectura.

Services (2023a) describe cómo la industria bancaria utiliza los servicios de Amazon Web Services (AWS) y el aprendizaje automático para mejorar sus capacidades de detección de fraudes, específicamente para la toma de control de cuentas (ATO) y el lavado de dinero (AML). Detalla una arquitectura modular basada en eventos que permite a los bancos gestionar y detectar fraudes de manera más efectiva mediante la integración de modelos de aprendizaje automático y análisis

en tiempo real.

Este documento es relevante para nuestro proyecto sobre arquitecturas de referencia para hubs de pago porque aborda directamente la integración de tecnologías avanzadas en la detección de fraudes, un componente crítico en la seguridad de las transacciones financieras. Los enfoques descritos pueden ser adaptados para desarrollar o mejorar sistemas de detección de fraudes en el contexto de los hubs de pago, aprovechando las capacidades de AWS para manejar grandes volúmenes de transacciones y datos en tiempo real, lo cual es esencial para garantizar transacciones seguras y confiables en plataformas de comercio electrónico.

Services (2023b) ofrece una descripción detallada de cómo construir una plataforma de procesamiento de pagos con tarjeta de crédito utilizando los servicios de AWS. Los autores, Sudhir Kalidindi, Alejandro Balzarro y Paul Chang, exploran dos arquitecturas de referencia principales: una para el lado de adquisición y otra para el lado de emisión de la autorización de pagos con tarjeta de crédito. El texto subraya la importancia de la modernización de los sistemas de procesamiento de pagos en la nube para escalar de manera eficiente, mantener una alta disponibilidad, cumplir con requisitos de seguridad estrictos y apoyar la expansión global.

Este documento es sumamente relevante para nuestro proyecto sobre arquitecturas de referencia para hubs de pago, ya que proporciona una base sólida para comprender cómo se pueden diseñar y escalar soluciones de procesamiento de pagos en la nube, lo que es esencial para manejar eficientemente las transacciones de pago en plataformas de comercio electrónico. Además, las consideraciones de seguridad y cumplimiento detalladas en el documento son cruciales para garantizar que la arquitectura de referencia no solo sea eficiente, sino también segura y conforme con las regulaciones internacionales.

Services (2023f) proporciona una guía detallada sobre cómo implementar una arquitectura robusta para la conectividad de pagos, el manejo de gateways, la orquestación y el enrutamiento utilizando los servicios de AWS. Esta guía está dirigida a instituciones financieras y aborda cómo pueden aprovechar las soluciones serverless gestionadas de AWS para simplificar la infraestructura necesaria para el procesamiento de pagos, enfocándose en la eficiencia operativa y la optimización de costos. Incluye ejemplos prácticos sobre la implementación de códigos QR y el procesamiento de pagos digitales, facilitando a los usuarios la capacidad de escanear códigos QR en puntos de venta o páginas de pago.

Este documento es especialmente relevante para nuestro proyecto de diseño de arquitecturas de referencia para hubs de pago, ya que ofrece una visión integral sobre cómo configurar y gestionar la conectividad de pagos en un entorno de comercio electrónico. Los enfoques y técnicas discutidos pueden ser fundamentales para asegurar que la arquitectura del hub de pagos sea capaz de manejar las demandas de enrutamiento, seguridad y escalabilidad de manera eficiente, al tiempo que se optimizan los costos y se mejora la experiencia del usuario final en el contexto de pagos digitales y transacciones en línea.

Services (2023d) explora el auge en el uso de códigos QR en transacciones de pagos, acelerado por

las necesidades de distanciamiento social y minimización de contacto directo durante la pandemia. Detalla cómo los códigos QR han sido adoptados tanto en mercados emergentes como desarrollados para transacciones rápidas y seguras, apoyados por tecnologías en la nube de AWS que facilitan una implementación efectiva y segura.

Este estudio es relevante para nuestra investigación sobre arquitecturas de referencia para hubs de pago en comercio electrónico, ya que resalta cómo la tecnología de códigos QR puede integrarse en los sistemas de pago modernos, ofreciendo un método de pago eficiente y sin contacto. La arquitectura de referencia para el procesamiento de pagos QR presentada puede informar nuestras decisiones de diseño para asegurar que nuestro sistema sea capaz de soportar transacciones rápidas y seguras, aprovechando las soluciones en la nube para optimizar la escalabilidad y la seguridad.

Services (2023c) discute cómo las compañías de pagos están utilizando tecnologías en la nube para desarrollar soluciones de SoftPOS (Software Point-of-Sale), que permiten que los dispositivos móviles con capacidad NFC (Near Field Communication) funcionen como terminales de pago sin contacto sin necesidad de hardware adicional. El artículo subraya el potencial de SoftPOS para democratizar los sistemas de pago electrónico, especialmente para micro-mercados y pequeñas empresas que tradicionalmente han enfrentado barreras de costo para adoptar tales tecnologías.

Este estudio es sumamente relevante para nuestro proyecto sobre arquitecturas de referencia para hubs de pago, dado que presenta una solución innovadora que reduce la necesidad de hardware dedicado para procesar pagos, lo cual puede ser críticamente ventajoso en entornos de comercio electrónico. La implementación de SoftPOS en AWS muestra cómo se puede lograr un modelo de pago escalable, seguro y eficiente, alineado con las expectativas modernas de pagos rápidos y seguros en el comercio digital. Esto puede informar nuestro enfoque hacia la integración de tecnologías similares en nuestra arquitectura propuesta, asegurando que sea capaz de soportar transacciones eficientes y seguras con un bajo costo de entrada y alta adaptabilidad.

Services (2023e) describe cómo los proveedores de infraestructura del mercado de capitales están utilizando la nube para acelerar su innovación y modernización, con un enfoque particular en la gestión de latencias bajas en entornos de trading. El texto explora patrones arquitectónicos y servicios clave de AWS para construir un prototipo de intercambio nativo de la nube con perfiles de latencia comparables a las soluciones locales, utilizando servicios como Amazon EC2, ElastiCache, y Kinesis.

Esta investigación es relevante para nuestro proyecto de arquitecturas de referencia para hubs de pago porque ilustra cómo las tecnologías en la nube pueden ser optimizadas para aplicaciones críticas que requieren respuestas de baja latencia y alto rendimiento, aspectos esenciales en el procesamiento de pagos en el comercio electrónico. Los enfoques descritos para minimizar la latencia y optimizar el rendimiento pueden ser aplicados directamente en la construcción de un hub de pago eficiente y seguro, asegurando que las transacciones se manejen con la rapidez y fiabilidad necesarias en un entorno comercial digital.

2.3. Resumen del capítulo

En este capítulo se establecen las bases teóricas y conceptuales para el diseño de una arquitectura de referencia para hubs de pago en el contexto del comercio electrónico. Se abordaron temas fundamentales como la arquitectura de software, destacando su importancia en la organización funcional y estructural de sistemas complejos, y las arquitecturas de referencia, que proporcionan un marco estandarizado y probado para guiar el desarrollo de sistemas específicos. Además, se explicó el concepto y la relevancia de los hubs de pago, plataformas tecnológicas avanzadas que centralizan y optimizan el procesamiento de transacciones financieras, mejorando la eficiencia operativa y la seguridad.

El capítulo también incluyó una discusión sobre el crecimiento del comercio electrónico y la transformación de las expectativas de los consumidores en términos de transacciones en línea, subrayando la necesidad de sistemas de pago rápidos, seguros y convenientes. Se describió el AWS Well-Architected Framework (WAF), una guía clave para diseñar sistemas de alto rendimiento y seguridad en la nube, y se destacó su aplicación en el contexto de hubs de pago. Finalmente, se revisó el estado del arte, analizando estudios y ejemplos prácticos sobre hubs de pago en general, abordando diversas tecnologías y enfoques para mejorar la infraestructura de procesamiento de pagos, con un enfoque particular en la eficiencia, la escalabilidad y la protección contra fraudes.

Capítulo 3

Metodología de Investigación

En este capítulo se detalla el rigor metodológico seguido para el desarrollo de la presente investigación. Se describe el enfoque seleccionado, las fases ejecutadas para su consecución y las técnicas específicas empleadas para la recolección y análisis de la información, asegurando la validez y replicabilidad de los resultados.

3.1. Enfoque Metodológico: Investigación de Ciencia del Diseño

Dada la naturaleza del objetivo de esta tesis —la creación de un artefacto tecnológico tangible en forma de una arquitectura de referencia—, se adoptó el paradigma de la **Investigación de Ciencia del Diseño** (Design Science Research, DSR) Hevner et al. (2004). DSR es un marco de investigación que aborda problemas del mundo real mediante el diseño, construcción y evaluación de artefactos innovadores Peffers et al. (2007). Este enfoque es especialmente pertinente, ya que no solo busca comprender una realidad, sino transformarla a través de la tecnología, en este caso, optimizando la construcción de Hubs de Pago en AWS.

El proceso de DSR seguido en este trabajo se alinea con los siete lineamientos propuestos por Hevner et al. Hevner et al. (2004), asegurando que el artefacto (la arquitectura de referencia) sea relevante para el problema, riguroso en su construcción y esté comunicado de manera efectiva tanto a una audiencia técnica como académica.

3.2. Fases de la Investigación

El proyecto se estructuró en tres fases secuenciales, directamente alineadas con los objetivos específicos planteados. Este flujo garantiza una progresión lógica desde la fundamentación teórica hasta la validación práctica del artefacto. La Figura 3.1 ilustra este proceso.

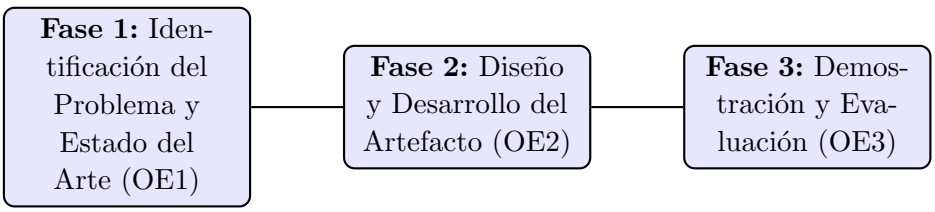

Figura 3.1: Fases del Proceso de Investigación Basado en DSR.

3.2.1. Fase 1: Análisis del Estado del Arte y Requerimientos

Esta fase inicial, correspondiente al **Objetivo Específico 1**, se centró en establecer una base teórica sólida. Se realizó una revisión sistemática de la literatura para identificar los conceptos clave de Hubs de Pago, arquitecturas de microservicios en la nube y los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework. Las actividades incluyeron:

- **Revisión de Literatura Académica**: Búsqueda y análisis de artículos en bases de datos como IEEE Xplore, ACM Digital Library y Scopus.
- **Análisis de Documentación Técnica**: Estudio exhaustivo de los whitepapers, guías de usuario y documentación oficial de AWS sobre servicios relevantes (e.g., Lambda, API Gateway, SQS, DynamoDB, IAM).
- **Estudio de Soluciones de Mercado**: Análisis de las arquitecturas de Hubs de Pago existentes para identificar patrones, componentes y desafíos comunes.

3.2.2. Fase 2: Diseño y Desarrollo de la Arquitectura de Referencia

Correspondiendo al **Objetivo Específico 2**, en esta fase se diseñó y desarrolló el artefacto central: la arquitectura de referencia para Hubs de Pago en AWS. El diseño se basó en los hallazgos de la Fase 1 y siguió los principios del AWS Well-Architected Framework. El proceso fue el siguiente:

- **Modelado Arquitectónico**: Se utilizaron diagramas C4 (Contexto, Contenedores, Componentes) para definir las vistas estáticas de la arquitectura.
- **Selección de Servicios AWS**: Se justificó la elección de cada servicio de AWS en función de los requerimientos de escalabilidad, seguridad, costo y rendimiento.
- **Implementación como Código (IaC)**: Se crearon plantillas de Terraform para provisionar la infraestructura, garantizando la replicabilidad y el control de versiones de la arquitectura.

3.2.3. Fase 3: Demostración y Evaluación de la Arquitectura

La fase final, alineada con el **Objetivo Específico 3**, consistió en demostrar la utilidad y evaluar la calidad del artefacto. La evaluación se centró en los dos pilares del AWS Well-Architected Framework priorizados:

3.2.3.1. Evaluación de la Eficiencia del Rendimiento

Para evaluar la eficiencia del rendimiento del artefacto, se diseñó un escenario de prueba que simula una carga de trabajo transaccional realista para un Hub de Pagos. El objetivo era cuantificar cómo la arquitectura responde bajo estrés y verificar que la selección de servicios y configuraciones (trade-offs) fuera la adecuada. Se utilizó la herramienta de pruebas de carga **k6 (de Grafana Labs)** para generar tráfico HTTP hacia el endpoint del API Gateway que inicia el flujo de procesamiento de pagos.

3.2.3.2. Evaluación de la Seguridad

La evaluación de la seguridad se centró en verificar la correcta implementación de los controles preventivos y de detección definidos en el diseño de la arquitectura. En lugar de una métrica cuantitativa, se utilizó una lista de verificación (checklist) basada en el pilar de Seguridad del AWS Well-Architected Framework **?** y el modelo de amenazas STRIDE. Las herramientas clave para esta auditoría fueron **AWS Security Hub** y **AWS IAM Access Analyzer** para verificar la correcta implementación de los controles en las siguientes áreas:

- Gestión de Identidad y Acceso (Principio de mínimo privilegio).
- Protección de Datos (Cifrado en tránsito y en reposo).
- Seguridad de la Infraestructura (Aislamiento de red y reglas de firewall).
- Detección y Registro (Logs de auditoría y aplicación).

Capítulo 4

Identificación de Desafíos y Requisitos para Hubs de Pago en AWS

4.1. Introducción al Capítulo

El comercio electrónico ha crecido exponencialmente con la digitalización global y la adopción masiva de pagos en línea. En este contexto, los hubs de pago se vuelven elementos críticos para ofrecer transacciones rápidas, seguras y escalables, al tiempo que cumplen regulaciones de protección de datos y seguridad, lo cual plantea retos técnicos y operativos para desarrolladores y operadores.

4.1.1. Propósito y Relevancia:

Este capítulo identifica y analiza los desafíos y requisitos —técnicos y operativos— de implementar un hub de pagos en AWS, enfocándose en las mejores prácticas del AWS Well-Architected Framework, especialmente en los pilares de Eficiencia del rendimiento y Seguridad. Su finalidad es sentar la base para una arquitectura de referencia que responda a las demandas actuales del comercio electrónico.

Para cumplir este propósito se emplea un enfoque mixto:

- **Revisión de literatura y benchmarking:** Se realizó un estudio exhaustivo de la bibliografía especializada y se compararon las principales plataformas de pago existentes.
- **Entrevistas con expertos:** Se desarrolló un protocolo de entrevistas dirigido a expertos en AWS, sistemas de pago y seguridad en la nube. Estos encuentros aportarán perspectivas prácticas y validarán los requisitos técnicos identificados.
- **Análisis de stakeholders:** Se realizó la identificación de actores clave y documentación de sus expectativas, junto con la priorización de requisitos funcionales, no-funcionales y operativos según su impacto en la operación y la seguridad.

4.1.2. Estructura del Capítulo

La organización de este capítulo se ha diseñado para abordar de forma sistemática los aspectos esenciales en el diseño de un hub de pagos eficiente y seguro. La estructura es la siguiente:

1. **Introducción y contextualización:** Marco teórico, definición de términos clave y KPIs iniciales de rendimiento y seguridad.

2. **Revisión de literatura y análisis de mercado:** Síntesis de fuentes y benchmarking de plataformas.

3. **Entrevistas con expertos:** Validación práctica de hallazgos sobre escalabilidad, fraude y mejores prácticas AWS.

4. **Análisis de stakeholders y requisitos:** Mapeo de actores y recopilación de requisitos técnicos y operativos.

5. **Definición y priorización de casos de uso:** Selección de escenarios críticos (picos de tráfico, detección de fraude en tiempo real) y modelado con diagramas C4.

6. **Documentación de desafíos técnicos y operativos:** Síntesis de retos estructurales y normativos vinculados a los pilares WAF.

7. **Conclusiones y plan de seguimiento:** Resumen de hallazgos, métricas de éxito y estrategias de validación (pruebas de carga, estrés y seguridad).

4.2. Contexto del Comercio Electrónico y Hubs de Pago: Desafíos y Oportunidades

El vertiginoso crecimiento del comercio electrónico, impulsado por la digitalización global y la masiva adopción de pagos en línea, ha posicionado a los hubs de pago como infraestructuras críticas. Estas plataformas centralizan y orquestan el procesamiento de transacciones, enfrentándose al desafío constante de equilibrar una experiencia de usuario ágil y sin fricciones con la imperiosa necesidad de robustez en seguridad y eficiencia operativa. La literatura especializada y el análisis del mercado actual de soluciones de pago convergen en la identificación de tendencias y brechas significativas que esta tesis busca abordar mediante una arquitectura de referencia optimizada.

La revisión de la literatura académica subraya que el diseño de sistemas de pago robustos trasciende la mera satisfacción de requisitos funcionales. Se enfatiza la necesidad de arquitecturas inherentemente escalables, mantenibles y adaptables, capaces de evolucionar ante la dinámica cambiante del sector. En este sentido, los Payment Hubs emergen como soluciones integradoras que buscan optimizar la eficiencia operativa y la agilidad mediante la centralización de múltiples funcionalidades y métodos de pago. Sin embargo, esta centralización intensifica los desafíos en dos áreas primordiales: la seguridad y el rendimiento.

- **En el ámbito de la seguridad,** el cumplimiento de normativas internacionales como PCI DSS, PSD2 y GDPR es ineludible para la protección de datos sensibles y la mitigación de riesgos. La literatura destaca la evolución de estrategias que van desde el cifrado de extremo a extremo y la autenticación multifactor (MFA) hasta la aplicación de inteligencia artificial y machine learning para la detección y prevención proactiva de fraudes.

- **En cuanto al rendimiento y la disponibilidad,** se subraya la criticidad de la baja latencia y la alta disponibilidad para garantizar una experiencia de usuario óptima y evitar la pérdida de transacciones, especialmente durante picos de demanda. Técnicas como el escalado automático, el balanceo de carga y el caching son consideradas fundamentales. La adopción de tecnologías en la nube, particularmente los servicios de AWS y los principios del Well-Architected Framework, se identifican como habilitadores clave para construir sistemas resilientes y escalables.

El análisis de las plataformas de pago existentes, tanto locales como internacionales (cuyos detalles comparativos se encuentran en los Anexos X e Y), revela un conjunto de funcionalidades clave consideradas estándar en la industria. Estas incluyen: un riguroso cumplimiento normativo, soporte para una amplia gama de métodos de pago (tarjetas, transferencias), mecanismos avanzados para la gestión de fraude en tiempo real, gestión de pagos recurrentes, herramientas de monitoreo transaccional y capacidades de personalización y soporte multicanal.

No obstante, el análisis también identifica diferenciadores innovadores que marcan la pauta hacia dónde se dirige el mercado. Entre ellos destacan el soporte avanzado para pagos internacionales y multidivisa, una personalización profunda de la experiencia de pago, la incorporación de métodos de pago emergentes (ej. códigos QR), una mayor interoperabilidad y conectividad con el ecosistema financiero, el uso de analítica predictiva para el monitoreo y la optimización inteligente de rutas de pago.

A pesar de la madurez de muchas soluciones, el benchmarking de características de rendimiento y seguridad evidencia brechas significativas. En rendimiento, si bien se promueve la escalabilidad automática y las arquitecturas de microservicios, la consecución de una latencia consistentemente baja y una verdadera resiliencia bajo cargas extremas sigue siendo un desafío. En seguridad, aunque el cifrado y la MFA son comunes, la sofisticación y adaptabilidad de los sistemas de gestión de fraudes y la implementación efectiva y auditable de todos los controles exigidos por normativas como PCI DSS varían considerablemente. Se observa una tendencia hacia la integración de herramientas analíticas avanzadas y una mayor flexibilidad en las APIs, pero la capacidad de operar fluidamente en entornos multicanales complejos, optimizando la experiencia móvil sin comprometer la seguridad, no está uniformemente resuelta.

En conclusión, el contexto actual de los hubs de pago en el comercio electrónico demanda arquitecturas que no solo incorporen las funcionalidades estándar y los diferenciadores emergentes, sino que también aborden de manera proactiva los persistentes desafíos de rendimiento y seguridad. La adopción de arquitecturas modulares (microservicios), el escalado elástico, el balanceo de carga inteligente, junto con estrategias de seguridad multicapa que incluyan cifrado robusto, MFA y detección avanzada de fraude, son cruciales. Esta tesis se enfoca en diseñar una arquitectura de referencia en AWS que integre estas mejores prácticas para cerrar las brechas identificadas, proporcionando una base sólida para la construcción de hubs de pago que sean a la vez altamente eficientes y seguros.

4.3. Hallazgos de Entrevistas con Expertos en AWS y Sistemas de Pago

Para complementar la revisión documental y el análisis de mercado, y con el fin de obtener perspectivas aplicadas sobre los desafíos y requisitos técnicos de los hubs de pago en AWS, se condujeron entrevistas semiestructuradas con profesionales experimentados en arquitecturas de nube, sistemas de pago y seguridad. La selección de expertos se basó en su experiencia práctica en entornos de alto rendimiento y su conocimiento en normativas de seguridad. La metodología (detallada en el Anexo B) incluyó un protocolo con preguntas guía sobre escalabilidad, seguridad, rendimiento, gestión de fraudes, modularidad y observabilidad, permitiendo la recolección de insights cualitativos que validaron y enriquecieron los hallazgos previos.

Los hallazgos principales de estas entrevistas, cruciales para la definición de la arquitectura de referencia, se resumen a continuación:

Escalabilidad y Elasticidad Imperativas Se reiteró la necesidad crítica de arquitecturas que puedan escalar dinámicamente para gestionar la variabilidad del volumen transaccional del comercio electrónico. Los expertos recomendaron enfáticamente el uso de **AWS Auto Scaling** para ajustar recursos en tiempo real y la adopción de **arquitecturas serverless (AWS Lambda)** para funciones clave, optimizando así la respuesta a picos de demanda y los costos operativos.

Seguridad y Cumplimiento Normativo como Pilares Fundamentales La seguridad fue identificada como una prioridad no negociable. Se destacó la importancia del **cifrado de datos en tránsito y en reposo (utilizando AWS KMS)**, la implementación de **políticas de IAM estrictas bajo el principio de mínimo privilegio**, y el **monitoreo continuo de amenazas (con Amazon GuardDuty y AWS Security Hub)**. Asimismo, se subrayó la necesidad de garantizar el cumplimiento de normativas como **PCI DSS, GDPR y PSD2** como un factor clave para la confianza y viabilidad del hub.

Optimización del Rendimiento para la Experiencia de Usuario La baja latencia en el procesamiento de transacciones es esencial. Los expertos aconsejaron el uso de **Amazon CloudFront** para la distribución de contenido y la optimización de **bases de datos gestionadas (como Amazon DynamoDB o RDS)** que ofrezcan alta disponibilidad y rendimiento consistente bajo carga.

Gestión Proactiva de Fraudes y Riesgos La integración de herramientas avanzadas para la detección de fraudes, como **Amazon Fraud Detector (basado en IA)**, y la implementación de **Autenticación Multifactor (MFA)** para transacciones de alto riesgo, fueron consideradas prácticas esenciales para minimizar vulnerabilidades.

Arquitectura Modular y Desacoplada (Microservicios) Hubo un consenso sobre los beneficios de diseñar el hub con **componentes independientes (microservicios)** para facilitar

el despliegue, mantenimiento y escalado individualizado. Se recomendó el uso de **comunicación asíncrona (mediante Amazon SQS/SNS)** para mejorar la resiliencia y la eficiencia en la interacción entre servicios.

Integración Multicanal y Centralización vía APIs Para una experiencia de usuario coherente y eficiente, se señaló la importancia de desarrollar **APIs unificadas y centralizadas** que permitan la integración fluida con diversos canales (web, móvil, POS). La capacidad de **personalizar la experiencia de pago** también fue destacada como un diferenciador.

Observabilidad Integral La necesidad de un **monitoreo en tiempo real (con Amazon CloudWatch)** y un **registro detallado y trazable de todas las transacciones y eventos (con AWS CloudTrail)** fue enfatizada como fundamental para la identificación temprana de incidentes, el análisis forense y la optimización continua del sistema.

Estos insights prácticos (resumidos en la Tabla X, con una versión más detallada en el Anexo B) validan los requisitos identificados en la literatura y el análisis de mercado, y serán directamente incorporados en el diseño de la arquitectura de referencia. La alineación de estas recomendaciones con los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework refuerza la dirección técnica de esta tesis.

4.4. Identificación y Análisis de Stakeholders

La comprensión y gestión adecuada de los stakeholders son cruciales para el éxito en el diseño de una arquitectura de referencia para hubs de pago, tal como lo subraya el Project Management Institute (PMI) en su guía PMBOK. Este proceso continuo permite alinear expectativas y definir estrategias de comunicación efectivas. A continuación, se identifican los actores clave en el contexto de este proyecto.

4.4.1. Identificación de Stakeholders Clave y Clasificación

Los stakeholders identificados, junto con su poder de influencia e interés en el proyecto, son:

- **Empresas de Comercio Electrónico (ECC):** Actores principales que dependen de una solución segura y escalable para procesar transacciones. *Poder: Alto — Interés: Alto.*
- **Proveedores de Servicios de Pago (PSP):** Entidades que facilitan la conexión entre comerciantes y clientes, críticos para la integración y optimización de pagos. *Poder: Medio — Interés: Alto.*
- **Clientes Finales (CF):** Usuarios finales que demandan una experiencia de pago rápida, segura y sin fricciones. *Poder: Bajo — Interés: Alto.*
- **AWS y otros Proveedores de Infraestructura (PIN):** Proveen la infraestructura técnica y servicios necesarios para garantizar alta disponibilidad, escalabilidad y seguridad. *Poder: Alto — Interés: Medio.*

- **Reguladores y Entidades de Cumplimiento (REC):** Organismos encargados de asegurar el cumplimiento de normativas. *Poder: Alto — Interés: Medio.*
- **Equipos de Desarrollo y Operaciones (EDO):** Responsables de la implementación y mantenimiento del hub. *Poder: Medio — Interés: Medio.*

La Matriz de Poder e Interés de Mendelow (ver Anexo Z para una representación gráfica detallada) clasifica a estos stakeholders, lo que facilita la definición de estrategias de gestión concisas:

Gestionar de Cerca (Alto Poder/Alto Interés): ECC.

Mantener Satisfechos (Alto Poder/Bajo-Medio Interés): AWS, PIN y REC.

Informar Regularmente (Bajo Poder/Alto Interés): CF.

Monitorear (Medio Poder/Medio Interés): PSP y EDO.

4.4.2. Análisis de Intereses y Derivación de Requisitos Clave

Para precisar las necesidades de cada grupo, se realizaron revisiones documentales y entrevistas semiestructuradas. El análisis de sus intereses y expectativas (cuyo detalle se encuentra en el Anexo W) revela un conjunto de requisitos generales que la arquitectura de referencia debe abordar. En lugar de enumerar cada requisito por stakeholder, se sintetizan las necesidades convergentes:

- **Eficiencia y Alto Rendimiento:** Las ECC requieren procesamiento rápido y escalabilidad para picos de demanda, mientras que los CF esperan un checkout ágil. Los PIN, como AWS, proporcionan las herramientas para lograr esta escalabilidad automática y arquitecturas optimizadas.
- **Seguridad Robusta y Cumplimiento Normativo:** Es una demanda transversal. Las REC imponen el cumplimiento de normativas; las ECC y PSP necesitan protegerse contra el fraude y asegurar los datos; y los CF esperan transacciones seguras.
- **Interoperabilidad y Flexibilidad:** Las ECC y PSP necesitan una integración fluida con múltiples sistemas y bancos. Los EDO requieren APIs extensibles y documentadas para facilitar la personalización y la evolución del sistema.
- **Operatividad y Mantenibilidad:** Los EDO precisan herramientas de monitoreo, paneles de control centralizados y procesos de CI/CD. Las ECC también se benefician de la alta disponibilidad y la gestión eficiente de incidentes.
- **Experiencia de Usuario Óptima:** Primordial para los CF, se traduce en procesos de pago simplificados y notificaciones en tiempo real, lo cual también es un objetivo para las ECC.
- **Gestión Centralizada y Transparencia:** Tanto las ECC como los EDO buscan una gestión unificada y visibilidad del sistema.

Estos intereses y expectativas de los stakeholders son fundamentales y se traducirán directamente en los requisitos funcionales, no funcionales y operativos detallados en la sección 4.5. Este enfoque asegura que la arquitectura de referencia no solo aborde los desafíos técnicos identificados, sino que también esté intrínsecamente alineada con las necesidades del negocio y operativas de todos los actores involucrados en el ecosistema de pagos del comercio electrónico.

4.5. Definición y Priorización de Requisitos

La documentación exhaustiva y la priorización de requisitos son esenciales para guiar el diseño de una arquitectura de referencia que cumpla con las demandas del comercio electrónico y los lineamientos del AWS Well-Architected Framework. La recopilación de requisitos se ha estructurado en categorías funcionales, no funcionales y operativas, asegurando claridad, verificabilidad y trazabilidad, siguiendo las mejores prácticas de la gestión de proyectos y la ingeniería de software.

4.5.1. Requisitos Funcionales (RF)

Los requisitos funcionales definen las capacidades y acciones específicas que el hub de pagos debe ejecutar para satisfacer las necesidades del negocio y de los usuarios. A continuación, se presentan los requisitos funcionales más críticos identificados, priorizados según su impacto. Una lista exhaustiva se encuentra en el Anexo C.

RF01 - Integración Multicanal (Prioridad: Alta): El sistema debe soportar la recepción y procesamiento unificado de transacciones originadas desde diversos canales de venta (web, móvil, puntos de venta físicos), garantizando una experiencia de pago coherente.

RF03 - Gestión de Fraudes en Tiempo Real (Prioridad: Alta): Se deben incorporar mecanismos para detectar, evaluar y bloquear transacciones sospechosas automáticamente o escalarlas para revisión manual, minimizando el riesgo de fraude sin impactar negativamente la experiencia de usuarios legítimos. Esto incluye la capacidad de integrarse con motores de análisis predictivo basados en machine learning (RF08).

RF06 - Interfaz Unificada de Solicitudes y Respuestas (Prioridad: Alta): El hub debe proveer una API estandarizada y bien documentada para la iniciación de pagos y la recepción de respuestas, simplificando la integración para los comercios y permitiendo la conexión con múltiples pasarelas de pago subyacentes de manera transparente.

RF07 - Optimización Inteligente de Rutas de Pago (Prioridad: Alta): El sistema deberá ser capaz de seleccionar dinámicamente la ruta de procesamiento de pago más eficiente (considerando costo, tiempo de respuesta, tasa de aprobación) entre las diversas pasarelas integradas, basándose en reglas configurables y, potencialmente, en datos históricos de rendimiento.

Otros requisitos funcionales relevantes incluyen el soporte para pagos internacionales y múltiples monedas (RF02), la gestión de pagos recurrentes y suscripciones (RF04), y la personalización del flujo de pago (RF05).

4.5.2. Requisitos No Funcionales (RNF)

Los requisitos no funcionales describen los atributos de calidad del sistema, siendo cruciales para la confianza, la experiencia del usuario y la viabilidad operativa, con un fuerte énfasis en el rendimiento y la seguridad. Los RNF más críticos se resumen a continuación (ver Anexo C para detalles).

RNF01 - Escalabilidad Automática (Prioridad: Alta): La arquitectura debe ser capaz de escalar sus recursos para manejar un incremento súbito del 1000 % sobre la carga base de transacciones (de 50 a 500 TPS) en menos de 5 minutos, manteniendo la latencia de extremo a extremo por debajo del umbral definido en RNF02

RNF02 - Baja Latencia (Prioridad: Alta): El sistema debe procesar la gran mayoría de las transacciones (ej., 95 %) en tiempos de respuesta mínimos, idealmente por debajo de los 300 milisegundos de extremo a extremo, para asegurar una experiencia de usuario fluida y minimizar las tasas de abandono.

La latencia de extremo a extremo se mide desde el momento en que la solicitud de pago del cliente es recibida por el primer componente de la infraestructura de AWS (en este caso, Amazon API Gateway) hasta el momento en que la respuesta final (confirmación o rechazo) es enviada desde API Gateway de vuelta al cliente.

RNF03 - Cumplimiento Normativo (Prioridad: Alta): El hub de pagos debe diseñarse para cumplir estrictamente con las normativas y estándares internacionales y locales relevantes. Esto incluye, pero no se limita a:

- **PCI DSS (Payment Card Industry Data Security Standard):** La arquitectura debe implementar controles para cumplir con el Requerimiento 3.4, que exige que el PAN (Primary Account Number) sea ilegible donde se almacene, utilizando técnicas como la tokenización o el cifrado fuerte con claves gestionadas en AWS KMS (Key Management Service).
- **GDPR (General Data Protection Regulation):** Para la protección de datos personales de ciudadanos de la UE, asegurando el consentimiento explícito, los derechos de acceso y eliminación de datos, y la notificación de brechas.
- **PSD2 (Payment Services Directive 2):** Para mercados europeos, exigiendo autenticación fuerte del cliente (SCA) y acceso seguro a cuentas para terceros autorizados.
- **CCPA (California Consumer Privacy Act):** Otorgando a los residentes de California derechos sobre sus datos personales.
- **SOX (Sarbanes-Oxley Act):** Para empresas que lo requieran, estableciendo controles internos sobre los reportes financieros y la integridad de los datos. El sistema debe facilitar las auditorías y la generación de informes de cumplimiento.

RNF04 - Alta Disponibilidad y Recuperación ante Fallos (Prioridad: Alta): El hub debe garantizar una disponibilidad del 99.99 %, lo que equivale a un máximo de 52.6 minutos

de inactividad al año. Debe implementar una estrategia de recuperación automática multi-AZ (Multi-Availability Zone) con un Objetivo de Tiempo de Recuperación (RTO) inferior a 15 minutos y un Objetivo de Punto de Recuperación (RPO) de casi cero para los datos transaccionales críticos.

4.5.3. Requisitos Operativos (RO)

Los requisitos operativos definen las necesidades para la administración, el mantenimiento y la eficiencia de la operación diaria del hub de pagos. Los más críticos incluyen (ver Anexo C para detalles):

RO01 - Monitoreo y Observabilidad (Prioridad: Alta): Se requiere una capacidad avanzada para supervisar el rendimiento, la disponibilidad y la seguridad del sistema en tiempo real, con paneles de control centralizados y alertas proactivas para la detección temprana de problemas y una rápida respuesta a incidentes.

RO02 - Mantenimiento y Actualizaciones sin Interrupciones (Prioridad: Alta): El sistema debe permitir despliegues de nuevas versiones y actividades de mantenimiento de manera continua (CI/CD), sin afectar la disponibilidad del servicio para los usuarios finales.

RO06 - Cumplimiento de Acuerdos de Nivel de Servicio (SLA) (Prioridad: Alta): El hub debe cumplir con los SLAs definidos en términos de disponibilidad, tiempos de respuesta y otros indicadores clave, lo cual es fundamental para la confianza de los clientes y socios comerciales.

Otros requisitos operativos importantes son la gestión formal de configuraciones y cambios (RO03), un soporte y atención a incidentes eficiente (RO04), la gestión de capacidad y optimización de costos (RO05), y la formación continua del personal operativo (RO07).

La cuidadosa definición y priorización de estos requisitos funcionales, no funcionales y operativos, informada por el análisis de stakeholders y el contexto del mercado, establece una base sólida y medible para el diseño y la posterior validación de la arquitectura de referencia del hub de pagos en AWS.

4.6. Casos de Uso Críticos para el Hub de Pagos

La identificación de los casos de uso críticos es un paso esencial para asegurar que la arquitectura de referencia del hub de pagos aborde los escenarios más desafiantes y relevantes del comercio electrónico. Este proceso se fundamentó en un análisis combinado del comportamiento de plataformas líderes del mercado, insights prácticos obtenidos de entrevistas con expertos en la industria, y una revisión de la literatura técnica sobre sistemas de pago. A partir de esta triangulación, se han priorizado los siguientes casos de uso.

4.6.1. Descripción Sintetizada de Casos de Uso Críticos

Tabla 4.1: Definición de Casos de Uso Críticos para el Hub de Pagos en AWS.

ID	Nombre del Caso de Uso	Descripción Detallada
CU1	**Gestión de Picos de Tráfico en Eventos de Venta Masiva**	**Descripción:** El sistema debe gestionar incrementos exponenciales en el volumen de transacciones durante eventos comerciales clave (ej. Black Friday). **Actores:** Cliente final, Comercio, Sistema de Hub de Pagos. **Disparador:** Inicio de un evento de ventas masivas que genera un alto flujo de solicitudes de pago. **Flujo Principal:** 1. El sistema detecta un aumento en la carga transaccional. 2. Los mecanismos de autoescalado provisionan recursos computacionales (ej. instancias EC2, contenedores Fargate) de forma automática. 3. El balanceador de carga global (ej. AWS Global Accelerator) distribuye el tráfico eficientemente entre los recursos escalados. 4. Las transacciones se procesan sin degradación del rendimiento y se confirman al cliente y al comercio. **Requisitos No Funcionales Clave:** Escalabilidad, Eficiencia del Rendimiento, Alta Disponibilidad.

Continúa en la siguiente página

Tabla 4.1 – continuación de la página anterior

ID	Nombre del Caso de Uso	Descripción Detallada
CU2	**Prevención de Fraudes en Tiempo Real**	**Descripción:** Detección y bloqueo proactivo de transacciones fraudulentas sin impactar negativamente la experiencia de usuarios legítimos. **Actores:** Cliente final, Sistema de Hub de Pagos, Motor de Detección de Fraude. **Disparador:** Recepción de una solicitud de pago. **Flujo Principal:** 1. El Hub recibe una transacción y la envía al motor de detección de fraude (ej. Amazon Fraud Detector). 2. El motor analiza la transacción en tiempo real contra modelos de ML y reglas predefinidas. 3. Si el riesgo es bajo, la transacción continúa su flujo normal. 4. Si el riesgo es alto, la transacción es bloqueada y se notifica a los sistemas pertinentes. 5. Si el riesgo es intermedio, se puede solicitar autenticación multifactor (MFA) al cliente. **Requisitos No Funcionales Clave:** Seguridad, Eficiencia del Rendimiento (baja latencia), Fiabilidad.

Continúa en la siguiente página

Tabla 4.1 – continuación de la página anterior

ID	Nombre del Caso de Uso	Descripción Detallada
CU3	**Optimización de Rutas de Procesamiento de Pagos**	**Descripción:** Selección dinámica y eficiente del Proveedor de Servicios de Pago (PSP) más adecuado para cada transacción. **Actores:** Sistema de Hub de Pagos, Algoritmo de Ruteo Inteligente, Sistemas de Monitoreo de PSP. **Disparador:** Una transacción es autorizada y está lista para ser enrutada a un PSP. **Flujo Principal:** 1. El algoritmo de ruteo evalúa los PSPs disponibles en tiempo real. 2. Considera factores configurables: costos, tasas de aprobación históricas, latencia actual y disponibilidad del PSP. 3. Selecciona la ruta óptima según la estrategia definida (ej. menor costo, mayor probabilidad de éxito). 4. Envía la transacción al PSP seleccionado. 5. Registra el resultado para retroalimentar el algoritmo. **Requisitos No Funcionales Clave:** Eficiencia de Costos, Eficiencia del Rendimiento, Fiabilidad.

Continúa en la siguiente página

Tabla 4.1 – continuación de la página anterior

ID	Nombre del Caso de Uso	Descripción Detallada
CU4	**Cumplimiento Normativo y Auditoría Continua**	**Descripción:** Garantizar la adhesión constante a regulaciones (ej. PCI DSS, PSD2) y facilitar la generación de evidencia de auditoría. **Actores:** Operador del Hub, Auditor Externo, Sistema de Hub de Pagos. **Disparador:** Operación continua del sistema; solicitud de una auditoría. **Flujo Principal:** 1. Todas las operaciones sensibles y el acceso a datos son registrados en un log inmutable (ej. AWS CloudTrail, Amazon QLDB). 2. Los datos sensibles en reposo y en tránsito son cifrados (ej. KMS, ACM). 3. Herramientas como AWS Config y AWS Security Hub validan continuamente la configuración del entorno contra las reglas de cumplimiento. 4. Se generan reportes de cumplimiento de forma automática o bajo demanda para las auditorías. **Requisitos No Funcionales Clave:** Seguridad, Cumplimiento Normativo.

Continúa en la siguiente página

Tabla 4.1 – continuación de la página anterior

ID	Nombre del Caso de Uso	Descripción Detallada
CU5	**Pagos Multicanal en Comercio Electrónico**	**Descripción:** Proporcionar una experiencia de pago unificada y consistente a través de diversos canales de venta (web, móvil, POS). **Actores:** Cliente final, Comercio, Sistema de Hub de Pagos. **Disparador:** Un cliente inicia un pago desde cualquier canal integrado. **Flujo Principal:** 1. El canal de venta (ej. App móvil) se comunica con el Hub a través de una API unificada (ej. Amazon API Gateway). 2. El Hub identifica el canal y aplica las personalizaciones de flujo o UX correspondientes. 3. Procesa la transacción de manera centralizada. 4. Sincroniza el estado de la transacción en tiempo real con los sistemas del comercio (ej. ERP, CRM). **Requisitos No Funcionales Clave:** Usabilidad, Interoperabilidad, Integridad de Datos.

Continúa en la siguiente página

Tabla 4.1 – continuación de la página anterior

ID	Nombre del Caso de Uso	Descripción Detallada
CU6	**Transformación Dinámica de Mensajes de Pago**	**Descripción:** Abstraer la complejidad de integración con múltiples PSPs, cada uno con formatos y protocolos distintos. **Actores:** Comercio, Sistema de Hub de Pagos, PSPs. **Disparador:** El comercio envía una solicitud de pago al Hub utilizando un formato canónico. **Flujo Principal:** 1. El Hub recibe la solicitud en su formato unificado. 2. Basado en la ruta de pago seleccionada (ver CU3), el módulo de transformación mapea la solicitud al formato específico del PSP destino. 3. El Hub envía la solicitud transformada al PSP. 4. Al recibir la respuesta del PSP, el Hub la transforma de vuelta al formato canónico antes de responder al comercio. **Requisitos No Funcionales Clave:** Interoperabilidad, Mantenibilidad, Flexibilidad.

4.6.2. Relevancia de los Casos de Uso para la Arquitectura de Referencia

La inclusión y el abordaje efectivo de estos casos de uso críticos en el diseño del hub de pagos son fundamentales. Estos escenarios no solo representan los desafíos técnicos más significativos, sino que también encapsulan las expectativas clave de los stakeholders en términos de funcionalidad y calidad de servicio. Una arquitectura que resuelva satisfactoriamente estos casos de uso demostrará ser:

- **Flexible y Escalable:** Capaz de adaptarse a la volatilidad de la demanda transaccional y a la evolución de los modelos de negocio.
- **Segura y Confiable:** Implementando defensas robustas contra el fraude y asegurando el cumplimiento normativo, lo que es vital para la confianza en el ecosistema de pagos.
- **Eficiente Operativamente:** Optimizando procesos clave como el enrutamiento de pagos y la integración con PSPs, lo que se traduce en reducción de costos y complejidad.
- **Centrada en la Experiencia del Usuario:** Garantizando transacciones rápidas, seguras

y consistentes en todos los puntos de contacto, lo que fomenta la conversión y la lealtad del cliente.

En resumen, estos casos de uso actúan como un conjunto de pruebas de fuego para la arquitectura de referencia. Su resolución satisfactoria, aplicando los principios del AWS Well-Architected Framework, validará la capacidad de la propuesta para enfrentar los retos del mundo real en el procesamiento de pagos para el comercio electrónico, ofreciendo una solución innovadora, adaptativa y de alto valor.

4.7. Identificación y Priorización de Desafíos Técnicos y Operativos

La construcción de una arquitectura de referencia robusta para un hub de pagos en AWS exige una comprensión clara de los desafíos inherentes al dominio del comercio electrónico y al procesamiento de pagos. Esta sección sintetiza los principales desafíos técnicos y operativos identificados a través de la revisión de literatura, el análisis de mercado y las entrevistas con expertos, y procede a su priorización para enfocar los esfuerzos de diseño.

4.7.1. Síntesis de Desafíos Clave

Los siguientes desafíos han sido identificados como críticos en la implementación de hubs de pago eficientes y seguros:

D1: Integración y Complejidad Sistémica La necesidad de interactuar con múltiples pasarelas de pago (PSPs), cada una con sus propios protocolos y formatos, junto con la integración a sistemas legados de los comercios, introduce una considerable complejidad sistémica que puede dificultar la gestión operativa y la evolución ágil del hub.

D2: Rendimiento y Escalabilidad bajo Carga Variable El volumen de transacciones en el comercio electrónico es inherentemente variable, con picos extremos durante eventos de alta demanda (ej. Black Friday). El hub debe escalar dinámicamente para procesar estos volúmenes con baja latencia y alta disponibilidad, evitando cuellos de botella.

D3: Seguridad y Prevención de Fraudes Proteger datos financieros sensibles (como números de tarjeta) y prevenir transacciones fraudulentas son imperativos. Cualquier brecha de seguridad puede tener consecuencias devastadoras en términos de pérdidas financieras y daño reputacional.

D4: Cumplimiento Normativo Los hubs de pago deben adherirse a un conjunto complejo y evolutivo de normativas internacionales y locales (PCI DSS, GDPR, PSD2, etc.), lo que exige un diseño que facilite la auditoría y el monitoreo continuo del cumplimiento.

D5: Flexibilidad y Adaptabilidad El ecosistema de pagos y las tecnologías asociadas evolucionan rápidamente. Una arquitectura rígida puede volverse obsoleta rápidamente, limitando la capacidad de incorporar nuevos métodos de pago, funcionalidades o adaptarse a cambios en las regulaciones.

D6: Experiencia de Usuario y Pagos Multicanal Los consumidores esperan una experiencia de pago fluida, rápida y consistente a través de todos los canales de interacción (web, móvil, físico). El hub debe soportar esta omnicanalidad sin introducir fricciones.

D7: Transformación Dinámica de Solicitudes y Respuestas La diversidad de formatos de mensaje entre el hub y las múltiples PSPs requiere una capacidad robusta y flexible de transformación de datos para simplificar la integración y asegurar la coherencia.

Estos desafíos impactan directamente las características arquitectónicas fundamentales del sistema, tales como su disponibilidad, rendimiento, recuperabilidad, mantenibilidad y seguridad. Un diseño efectivo debe considerar cómo cada desafío influye en las decisiones estructurales (ej. modularidad, extensibilidad), operacionales (ej. escalabilidad, confiabilidad) y transversales (ej. seguridad, usabilidad) del hub de pagos. El detalle de cómo estos desafíos se mapean a características arquitectónicas específicas, aunque extenso, subraya la interconexión entre los problemas a resolver y las cualidades deseadas del sistema.

4.7.2. Priorización de Desafíos

Para guiar el proceso de diseño y asegurar que los aspectos más críticos sean abordados con la debida atención, los desafíos identificados se han priorizado utilizando criterios como el impacto potencial en el negocio y la experiencia del usuario, la urgencia de su resolución, la complejidad técnica inherente, la obligatoriedad del cumplimiento normativo y el riesgo asociado. La siguiente tabla presenta esta priorización:

Tabla 4.2: Matriz de Priorización de Desafíos para el Hub de Pagos.

ID	Desafío Identificado	Impacto	Urgencia	Complejidad	Riesgo	Prioridad
D1	Integración y Complejidad Sistémica	Alto	Medio	Alto	Medio	Alta
D2	Rendimiento y Escalabilidad bajo Carga Variable	Alto	Alto	Alto	Alto	Alta

Tabla 4.2 – continuación de la página anterior

ID	Desafío Identificado	Impacto	Urgencia	Complejidad	Riesgo	Prioridad
D3	Seguridad y Prevención de Fraudes	Alto	Alto	Alto	Alto	Alta
D4	Cumplimiento Normativo Riguroso	Alto	Alto	Alto	Alto	Alta
D5	Flexibilidad y Adaptabilidad Arquitectónica	Medio	Medio	Alto	Medio	Media
D6	Experiencia de Usuario y Soporte Multicanal	Alto	Medio	Medio	Medio	Alta
D7	Transformación Dinámica de Mensajes	Medio	Medio	Medio	Bajo	Media

Análisis de Prioridades

- **Alta Prioridad:** Los desafíos que impactan directamente la capacidad del hub para operar de forma segura, cumplir con regulaciones críticas, manejar la carga transaccional y ser mantenible son primordiales. Estos incluyen la escalabilidad para picos de tráfico, la seguridad integral y el cumplimiento normativo, la detección de fraude en tiempo real, una observabilidad avanzada para la gestión proactiva, la capacidad de realizar mantenimiento sin interrupciones y el soporte multicanal.

- **Media Prioridad:** Aspectos como la eficiencia en la integración de sistemas, la optimización de la latencia, la optimización de rutas de pago, la gestión de cambios y la capacitación del personal son importantes para la eficiencia y la calidad a largo plazo, pero pueden abordarse una vez que los cimientos de alta prioridad estén establecidos.

- **Baja Prioridad:** Desafíos relacionados con la flexibilidad a muy largo plazo, la personalización avanzada de la experiencia, la optimización de costos muy fina y mejoras generales de usabilidad/accesibilidad, aunque valiosos, se consideran secundarios frente a los imperativos de funcionamiento, seguridad y cumplimiento inicial.

Esta priorización informará las decisiones de diseño en la arquitectura de referencia, asegurando que los esfuerzos se concentren en mitigar los riesgos más significativos y en entregar las capacidades más críticas para un hub de pagos moderno en el entorno del comercio electrónico.

4.8. Conclusiones del capítulo

Este capítulo ha culminado la primera fase de la investigación: la identificación de los principales desafíos y requisitos técnicos y operativos para implementar un hub de pagos en AWS, orientado al contexto actual del comercio electrónico y con un énfasis en los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework.

El análisis integral, que abarcó la revisión de literatura especializada, el estudio de plataformas de pago existentes, las perspectivas de expertos en AWS y sistemas de pago, la identificación de stakeholders y la definición detallada de requisitos y casos de uso críticos, ha permitido consolidar una comprensión profunda de las exigencias que debe satisfacer una arquitectura de referencia moderna.

Los hallazgos clave de este capítulo se pueden sintetizar en los siguientes puntos críticos:

- **Imperativos de Rendimiento y Escalabilidad:** Se ha evidenciado la necesidad ineludible de una arquitectura capaz de gestionar volúmenes transaccionales masivos y fluctuantes, característicos de eventos de venta en el comercio electrónico, manteniendo una baja latencia para optimizar la experiencia del usuario. La escalabilidad automática y el diseño modular son, por tanto, requisitos fundamentales.

- **Robustez en Seguridad y Cumplimiento Normativo:** La protección integral de datos sensibles y la prevención proactiva de fraudes son pilares no negociables. La arquitectura debe incorporar mecanismos de seguridad multicapa, incluyendo cifrado, autenticación fuerte y detección avanzada de amenazas, además de facilitar el cumplimiento de un espectro complejo de normativas internacionales (PCI DSS, GDPR, PSD2, entre otras).

- **Flexibilidad para la Integración y Evolución:** La capacidad de integrarse con una diversidad de pasarelas de pago y sistemas externos, así como la adaptabilidad para incorporar nuevas tecnologías y modelos de negocio, son esenciales para la longevidad y relevancia del hub.

- **Centralidad de la Experiencia de Usuario y Soporte Multicanal:** Ofrecer una experiencia de pago optimizada, consistente y segura a través de todos los canales de interacción es un factor determinante para el éxito en el comercio electrónico.

- **Gestión Operativa Eficiente:** La observabilidad, el monitoreo continuo, la gestión de configuraciones y los planes de mantenimiento y recuperación son cruciales para asegurar la robustez y la continuidad operativa del sistema.

La priorización de los desafíos técnicos y operativos ha clarificado que la **escalabilidad**, la **seguridad**, el **cumplimiento normativo** y la **detección de fraude** son las áreas de mayor impacto y urgencia que la arquitectura debe abordar de manera prioritaria.

Estos hallazgos y requisitos documentados sientan una base sólida y detallada para las siguientes fases de esta investigación. El Objetivo Específico 2 se centrará en evaluar cómo las capacidades

y servicios específicos de AWS pueden ser aprovechados para abordar estos desafíos y satisfacer los requisitos identificados, con un foco en los pilares de Eficiencia del Rendimiento y Seguridad. Posteriormente, el Objetivo Específico 3 utilizará esta evaluación, junto con los hallazgos de este capítulo, para proponer y validar la arquitectura de referencia detallada. En conjunto, este primer capítulo ha establecido el "qué" el "porqué", preparando el camino para definir el cómo.[en] los capítulos subsiguientes.

Capítulo 5

Evaluación de servicios de AWS y descripción de escenarios clave

5.1. Introducción al capítulo

Una vez identificados y priorizados los desafíos técnicos y operativos para un hub de pagos en el Capítulo 4, el siguiente paso lógico es evaluar las herramientas específicas que pueden darles solución.

Este capítulo se centra en el ecosistema de Amazon Web Services (AWS) para determinar cómo sus servicios y capacidades pueden abordar directamente los requisitos de rendimiento y seguridad. Se realizará una selección estratégica de servicios, se evaluarán sus ventajas y consideraciones en el contexto de los hubs de pago, y se ilustrará su aplicación combinada a través de escenarios prácticos. El objetivo es sentar las bases tecnológicas para el diseño de la arquitectura de referencia que se propondrá en el capítulo subsecuente.

5.2. Selección y Evaluación Estratégica de Servicios AWS para Hubs de Pago

Para diseñar una arquitectura de referencia robusta, eficiente y segura para un hub de pagos en AWS, es crucial primero identificar los desafíos inherentes a estos sistemas y luego evaluar cómo los servicios específicos de AWS pueden abordarlos. Esta sección se enfoca en la selección estratégica de dichos servicios, alineándolos con los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework. Se realiza un mapeo de los desafíos clave con las capacidades de AWS, se evalúan cualitativamente los servicios más relevantes y se ilustra su aplicación mediante escenarios prácticos en hubs de pago.

5.2.0.1. Mapeo General: Desafíos Clave vs. Capacidades AWS y Pilares WAF

Un hub de pagos en el contexto del comercio electrónico enfrenta múltiples desafíos técnicos y operativos. La plataforma AWS ofrece un conjunto de servicios y capacidades que, alineados con los pilares del Well-Architected Framework (WAF), permiten abordar estos desafíos de manera efectiva.

La conexión detallada entre cada desafío, los principios WAF aplicados, las capacidades estratégicas de AWS y los servicios específicos seleccionados se presenta en la **matriz de alineación**

consolidada en el Capítulo 6, Tabla 6.1. Dicha matriz sirve como la justificación central para el diseño de la arquitectura de referencia. A continuación, se procede a la evaluación cualitativa de los servicios más relevantes.

5.2.1. Evaluación Cualitativa de servicios AWS seleccionados

Esta sección presenta un análisis cualitativo de los servicios de AWS identificados como estratégicos para la construcción de un hub de pagos en el contexto del comercio electrónico. Se evalúan sus ventajas, limitaciones y casos de uso específicos, enfocándose en cómo contribuyen a los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework. La selección de estos servicios se basa en su impacto directo sobre los desafíos y requisitos identificados en el Capítulo 1 y su mapeo en la sección 2.4.

La construcción de un hub de pagos eficiente y seguro en AWS se apoya en la selección estratégica de sus servicios. A continuación, se realiza una evaluación cualitativa de los servicios más impactantes, destacando su uso en hubs de pago, ventajas principales y consideraciones clave, con un enfoque en los pilares de Eficiencia del Rendimiento y Seguridad.

Servicios Clave para Eficiencia del Rendimiento: AWS Lambda

- *Uso Estratégico:* Ejecución serverless de lógica de negocio transaccional (procesamiento de pagos, validaciones, notificaciones) y fomento de microservicios.
- *Ventajas Clave:* Escalabilidad automática e instantánea, modelo de pago por ejecución (eficiencia de costos), reducción de carga operativa.
- *Consideración:* Posibles çold starts"que pueden introducir latencia. Este trade-off se puede mitigar con Provisioned Concurrency, que mantiene un número de entornos de ejecución pre-calentados a costa de un cargo fijo, o con AWS Lambda SnapStart (para runtimes de Java), que mejora el tiempo de inicio desde una 'snapshot' en caché sin costo adicional, pero introduce consideraciones sobre la unicidad del estado en la 'snapshot'.
- *Caso de Uso:* Procesar la lógica de autorización/captura de un pago individual o realizar transformaciones de formato de mensajes.

Amazon ECS / EKS / AWS Fargate

- *Uso Estratégico:* Ejecución de microservicios contenerizados que requieren control del entorno, larga duración o no se adaptan bien a Lambda.
- *Ventajas Clave:* Portabilidad y consistencia del entorno, escalabilidad controlada, flexibilidad sobre el entorno de ejecución. Fargate ofrece opción serverless para contenedores.
- *Consideración:* Mayor complejidad de gestión (ECS/EKS) en comparación con Lambda, a menos que se use Fargate.

- *Caso de Uso:* Microservicios core de procesamiento de pagos con estado en memoria o conexiones persistentes; procesos batch de conciliación.

Amazon DynamoDB

- *Uso Estratégico:* Base de datos NoSQL primaria para datos de alto volumen y acceso de baja latencia (estados de transacción, tokens, perfiles de riesgo).
- *Ventajas Clave:* Rendimiento consistente de milisegundos, escalabilidad masiva, alta disponibilidad, modelo de costo flexible.
- *Consideración:* Requiere diseño cuidadoso del modelo de datos debido a patrones de consulta limitados para ad-hoc.
- *Caso de Uso:* Almacenar el estado y detalles de cada transacción de pago; guardar tokens de pago para pagos recurrentes.

Amazon Aurora

- *Uso Estratégico:* Base de datos relacional gestionada para datos que requieren consistencia fuerte o consultas complejas (datos maestros, logs de auditoría detallados).
- *Ventajas Clave:* Alto rendimiento, compatibilidad con MySQL/PostgreSQL, escalabilidad de almacenamiento y réplicas de lectura, consistencia fuerte (ACID).
- *Consideración:* Generalmente más costoso que DynamoDB; escalado de escritura más limitado.
- *Caso de Uso:* Almacenar datos maestros de comercios; logs de auditoría que requieran consultas complejas.

Amazon ElastiCache (Redis / Memcached)

- *Uso Estratégico:* Caché en memoria para reducir latencia en accesos a datos frecuentes y disminuir la carga en bases de datos primarias.
- *Ventajas Clave:* Latencia extremadamente baja (microsegundos), reducción de carga en BD, mejora el rendimiento de la aplicación.
- *Consideración:* Requiere implementar estrategias de invalidación de caché para mantener la coherencia.
- *Caso de Uso:* Almacenar en caché datos de configuración (claves API de pasarelas); limitar tasas de llamadas (rate limiting).

Amazon API Gateway

- *Uso Estratégico:* Punto de entrada gestionado, seguro y escalable para todas las APIs del hub.

- *Ventajas Clave:* Gestión centralizada de APIs (seguridad, throttling, versionado), escalabilidad automática, desacoplamiento de clientes y backend.
- *Consideración:* Introduce una pequeña latencia adicional; la configuración puede volverse compleja con muchas rutas.
- *Caso de Uso:* Exponer la API RESTful pública para transacciones; gestionar webhooks; fachada para microservicios internos.

Elastic Load Balancing (ELB - ALB/NLB)

- *Uso Estratégico:* Distribuye tráfico entre múltiples instancias de cómputo para alta disponibilidad y escalabilidad.
- *Ventajas Clave:* Alta disponibilidad (multi-AZ), escalabilidad automática de su propia capacidad, health checks. ALB (capa 7) para microservicios, NLB (capa 4) para ultra-baja latencia.
- *Consideración:* Costo por hora y LCUs; configuración de reglas de enrutamiento en ALB puede ser compleja.
- *Caso de Uso:* Distribuir tráfico a clústeres ECS/EKS o instancias EC2; terminación SSL/TLS.

Amazon CloudFront

- *Uso Estratégico:* CDN para acelerar entrega de activos estáticos (UI) y dinámicos (APIs) globalmente.
- *Ventajas Clave:* Baja latencia global, mejora del rendimiento por cacheo, seguridad integrada (WAF, Shield), reducción de carga en origen.
- *Consideración:* Requiere estrategias de invalidación de caché; propagación de cambios puede tardar.
- *Caso de Uso:* Servir UI del hub; cachear respuestas de APIs públicas; aplicar WAF en el borde.

Servicios Clave para Seguridad: AWS Identity and Access Management (IAM)

- *Uso Estratégico:* Controlar accesos a recursos AWS aplicando el principio de mínimo privilegio.
- *Ventajas Clave:* Control granular de permisos, gestión centralizada de identidades, integración amplia con servicios AWS.
- *Consideración:* La gestión de políticas puede volverse compleja; errores de configuración pueden tener alto impacto.

- *Caso de Uso:* Definir roles para Lambda/contenedores; gestionar acceso de desarrolladores/operadores.

AWS Key Management Service (KMS)

- *Uso Estratégico:* Crear, controlar y auditar el uso de claves criptográficas para cifrar datos sensibles. Esencial para PCI DSS.
- *Ventajas Clave:* Seguridad (HSMs validados FIPS 140-2), gestión centralizada de claves, integración nativa para cifrado transparente con muchos servicios, auditoría.
- *Consideración:* Costo por clave y por solicitud de API; una mala gestión puede llevar a pérdida de datos.
- *Caso de Uso:* Cifrar datos de tarjetas de pago en BBDD; cifrar backups y volúmenes EBS; cifrar mensajes en SQS/SNS.

AWS Secrets Manager

- *Uso Estratégico:* Almacenamiento y gestión segura del ciclo de vida (incluyendo rotación automática) de secretos.
- *Ventajas Clave:* Evita codificar secretos, rotación automática para servicios como RDS, control de acceso fino, auditoría.
- *Consideración:* Su costo es superior al de AWS Systems Manager Parameter Store debido a funcionalidades avanzadas como la rotación automática nativa de credenciales para servicios como Amazon RDS y la integración directa con otros servicios de AWS, justificando su uso para secretos críticos que requieren un ciclo de vida gestionado
- *Caso de Uso:* Almacenar credenciales de BBDD; guardar claves API de pasarelas de pago externas.

AWS WAF

- *Uso Estratégico:* Protección a nivel de aplicación contra exploits web comunes (SQLi, XSS), bots, dirigido a APIs y frontends.
- *Ventajas Clave:* Mitigación de riesgos OWASP Top 10, reglas flexibles (gestionadas o personalizadas), control de bots, ayuda a cumplir PCI DSS.
- *Consideración:* Requiere monitoreo y ajuste para minimizar falsos positivos; debe complementarse con otras capas de seguridad.
- *Caso de Uso:* Proteger API de inicio de pago; filtrar tráfico malicioso antes de ELB/API Gateway; implementar rate limiting.

AWS Shield (Standard / Advanced)

- *Uso Estratégico:* Protección contra ataques DDoS a nivel de red e infraestructura.
- *Ventajas Clave:* Shield Standard (gratuito) protege contra ataques comunes; Shield Advanced ofrece detección y mitigación más sofisticada, acceso al equipo DRT y protección de costos.
- *Consideración:* Shield Advanced tiene un costo fijo mensual significativo.
- *Caso de Uso:* Proteger endpoints públicos (CloudFront, ELB, Route 53) contra ataques DDoS.

Amazon GuardDuty

- *Uso Estratégico:* Detección inteligente y continua de amenazas y actividades maliciosas dentro del entorno AWS, analiza de forma continua múltiples fuentes de datos, incluyendo los logs de AWS CloudTrail (eventos de API), los VPC Flow Logs (tráfico de red) y los logs de DNS, para identificar actividades maliciosas
- *Ventajas Clave:* Utiliza Machine Learning y fuentes de inteligencia, amplia cobertura (CloudTrail, VPC Flow Logs, DNS logs) sin agentes, bajo impacto en rendimiento.
- *Consideración:* Es un servicio de detección; la respuesta/mitigación requiere configuración adicional.
- *Caso de Uso:* Detectar instancias comprometidas; identificar accesos no autorizados desde IPs maliciosas; detectar patrones inusuales de acceso a datos.

Servicios Transversales (Rendimiento y Seguridad): Amazon CloudWatch

- *Uso Estratégico:* Monitoreo y observabilidad integral para recopilar métricas, logs y eventos.
- *Ventajas Clave:* Visibilidad integral, alertas proactivas, paneles personalizables, potente análisis de logs (Logs Insights).
- *Consideración:* Configuración de métricas/alarmas complejas puede requerir esfuerzo; costos pueden escalar si no se gestiona bien la ingesta.
- *Caso de Uso:* Monitorear latencia de APIs, utilización de CPU/memoria; configurar alarmas para picos de errores; centralizar logs.

AWS CloudTrail

- *Uso Estratégico:* Registro de auditoría para todas las llamadas a la API de AWS, proporcionando trazabilidad.
- *Ventajas Clave:* Trazabilidad completa (quién, qué, cuándo, dónde), esencial para seguridad y cumplimiento (PCI DSS, SOX, GDPR), análisis operativo.
- *Consideración:* Puede generar gran volumen de logs; se enfoca en API de AWS, no en actividades dentro de instancias.

- *Caso de Uso:* Auditar cambios en configuraciones críticas de seguridad; investigar accesos no autorizados; proporcionar evidencia de auditoría.

Esta evaluación cualitativa sienta las bases para la selección de componentes en la arquitectura de referencia, buscando un equilibrio óptimo entre rendimiento, seguridad, costo y operatividad.

5.2.2. Ilustración de Aplicación: Escenarios Clave Resueltos con AWS

Para ilustrar de manera concreta cómo los servicios de AWS evaluados en la sección anterior se combinan para resolver los desafíos identificados, a continuación se presentan cuatro escenarios típicos en la operación de un hub de pagos para comercio electrónico. Cada escenario describe el desafío, la solución arquitectónica propuesta utilizando servicios clave de AWS y los beneficios obtenidos, alineados con los pilares de Eficiencia del Rendimiento y Seguridad del Well-Architected Framework.

5.2.2.1. Escenario 1: Procesamiento Rápido de Transacciones (Eficiencia del Rendimiento)

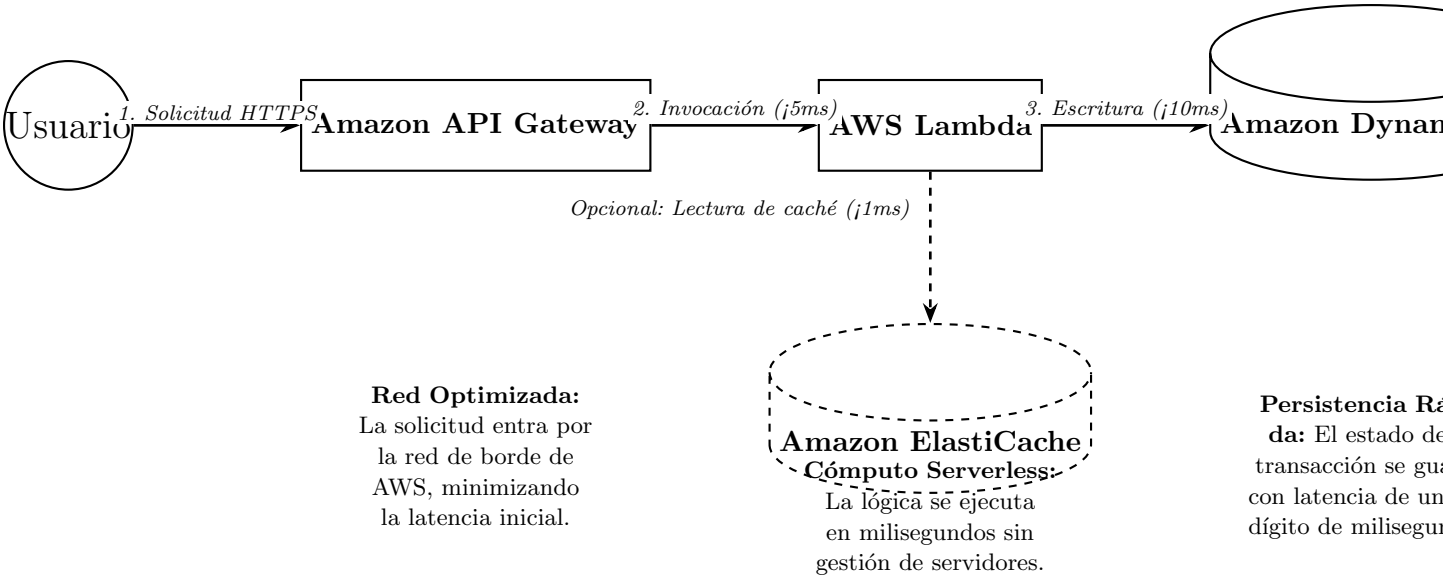

Figura 5.1: Diagrama de Flujo para el Escenario de Procesamiento Rápido de Transacciones.

Descripción del Escenario: Minimizar la latencia en el procesamiento de autorizaciones y capturas de pago es crucial para la experiencia del usuario y las tasas de conversión. Tiempos de respuesta elevados pueden llevar al abandono del carrito.

Solución Arquitectónica con AWS: Se propone una arquitectura serverless y optimizada para baja latencia:

- **Amazon API Gateway:** Actúa como punto de entrada, recibiendo la solicitud de pago. Puede configurarse con endpoints regionales o edge-optimized (integrado con CloudFront) para minimizar la latencia de red inicial.
- **AWS Lambda:** Ejecuta la lógica de negocio principal: validación de datos, formateo de solicitud para el PSP (Payment Service Provider), invocación de la API del PSP y procesamiento de la respuesta. Se configura con memoria adecuada y, si se anticipan patrones de tráfico irregulares, se puede usar *Provisioned Concurrency* o *Lambda SnapStart (para Java)* para mitigar cold starts. Ejemplo de Provisioned Concurrency:

```
Resources:
  MyPaymentFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: app.handler
      Runtime: python3.9
      CodeUri: src/
      ProvisionedConcurrencyConfig:
        ProvisionedConcurrentExecutions: 10
```

Figura 5.2: Ejemplo de configuración de Provisioned Concurrency en AWS SAM.

- **Amazon DynamoDB:** Almacena y recupera el estado de la transacción (ej. PENDIENTE, AUTORIZADO, FALLIDO) con latencia de milisegundos. Su escalabilidad garantiza rendimiento constante bajo carga.
- **Amazon ElastiCache (Opcional):** Para datos accedidos con altísima frecuencia (ej. configuración de comercios, reglas de enrutamiento), ElastiCache (Redis o Memcached) proporciona acceso en microsegundos, descargando a DynamoDB/Aurora.

Beneficios Clave:

- **Baja Latencia:** Cómputo rápido (Lambda), acceso a datos en milisegundos/microsegundos (DynamoDB/ElastiCache), red optimizada (API Gateway/CloudFront).
- **Escalabilidad Serverless:** Lambda y DynamoDB escalan automáticamente para manejar el volumen de transacciones sin gestión de infraestructura.
- **Eficiencia de Costos:** Modelo de pago por uso para Lambda y DynamoDB (On-Demand).

5.2.2.2. Escenario 2: Autenticación Segura y Cumplimiento Normativo (PCI DSS) (Seguridad)

Descripción del Escenario: El hub debe asegurar que solo entidades autorizadas (comercios, sistemas internos) puedan iniciar transacciones, y que el manejo y almacenamiento de datos sensibles

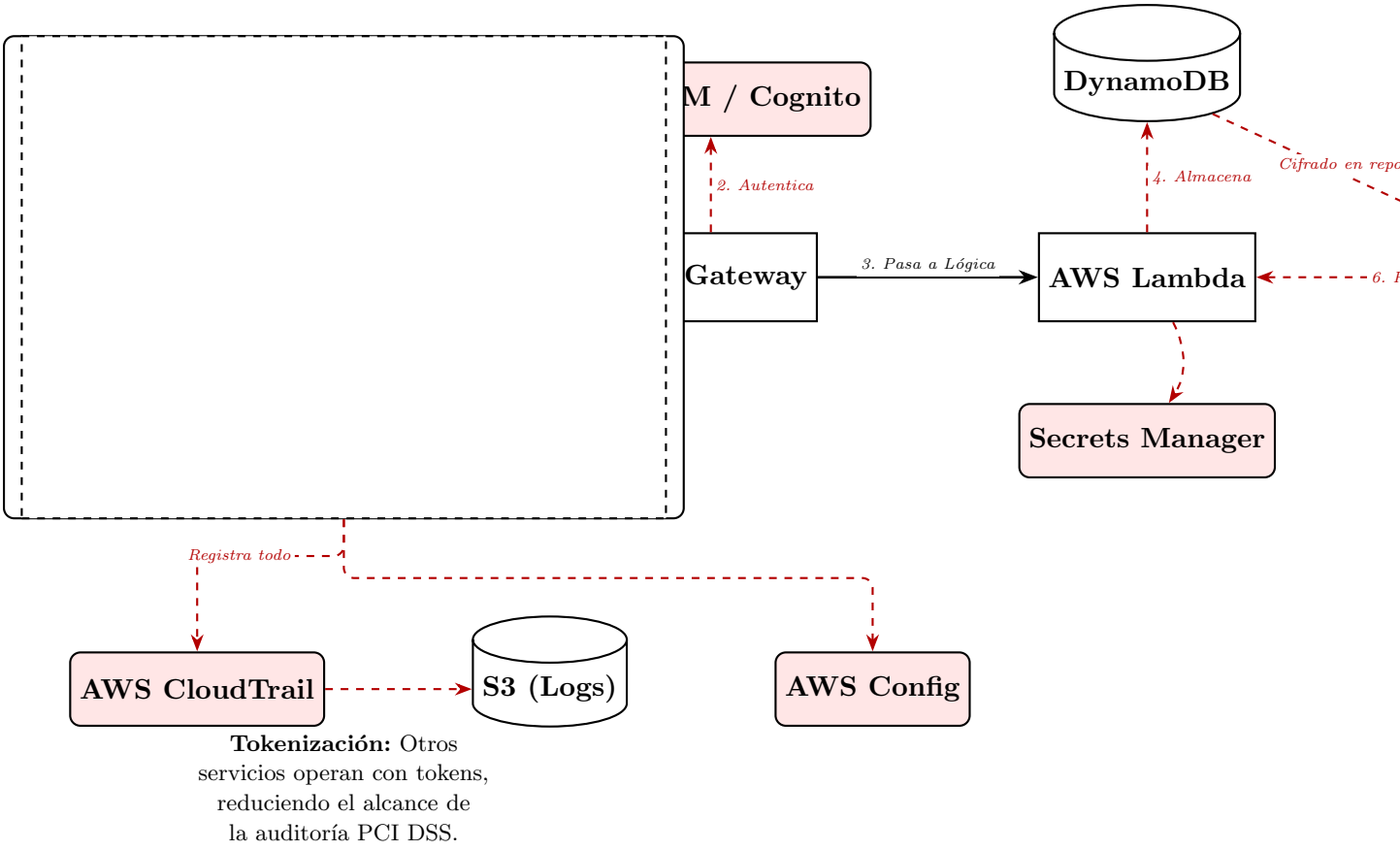

Figura 5.3: Diagrama de la Arquitectura de Seguridad y Cumplimiento Normativo.

(como el PAN - Primary Account Number) cumpla estrictamente con PCI DSS.

Solución Arquitectónica con AWS: Se implementan múltiples capas de seguridad y controles específicos:

- **Amazon API Gateway + IAM/Cognito:** Las llamadas a la API son autenticadas usando roles y políticas de IAM para sistemas internos, o Amazon Cognito para autenticar usuarios/aplicaciones externas. Lambda Authorizers pueden implementar lógica de autorización personalizada.
- **AWS KMS (Key Management Service):** Central para PCI DSS Req 3. Se utiliza para cifrar datos sensibles en reposo en todos los almacenes (DynamoDB, Aurora, S3 para logs, EBS). Se usan CMKs (Customer Managed Keys) para mayor control y auditoría.
- **AWS Secrets Manager:** Almacena de forma segura credenciales de acceso a PSPs, claves de API, contraseñas de bases de datos, eliminando la necesidad de incluirlas en código o configuración (PCI DSS Req 2, 8). Facilita la rotación de secretos.

- **Amazon VPC, Security Groups, Network ACLs:** Se diseña una red segmentada, aislando componentes críticos. Security Groups y NACLs actúan como firewalls para controlar estrictamente el tráfico entre los componentes y hacia/desde internet (PCI DSS Req 1).
- **AWS CloudTrail y AWS Config:** Registran todas las llamadas a la API de AWS y los cambios de configuración, respectivamente. Esencial para auditoría, monitoreo y cumplimiento (PCI DSS Req 10, 12). Los logs se almacenan de forma segura en S3 (cifrados con KMS) y con validación de integridad habilitada.
- **Tokenización (Lógica de Aplicación):** Aunque no es un servicio AWS per se, la tokenización reduce drásticamente el alcance (y por ende, el costo y la complejidad) de la auditoría PCI DSS al aislar el Entorno de Datos del Titular de la Tarjeta (Cardholder Data Environment - CDE). En esta arquitectura, solo un microservicio altamente seguro y segmentado, conocido como el 'vault' de tokenización, maneja el PAN real. El resto del sistema opera únicamente con tokens no sensibles, quedando fuera del alcance de la mayoría de los controles de PCI DSS.

Beneficios Clave:

- **Autenticación Robusta:** Control de acceso granular a las APIs.
- **Protección de Datos:** Cifrado fuerte en reposo y gestión segura de secretos.
- **Seguridad de Red:** Infraestructura segmentada y controlada.
- **Auditabilidad:** Trazabilidad completa de acciones y configuraciones.
- **Facilita Cumplimiento PCI DSS:** Aborda múltiples requisitos mediante servicios gestionados y configuraciones seguras.

5.2.2.3. Escenario 3: Protección contra Fraudes y Monitoreo Activo (Seguridad)

Descripción del Escenario: El hub de pagos debe ser capaz de detectar y prevenir transacciones fraudulentas en tiempo real o casi real, además de monitorear continuamente la salud y la seguridad del sistema para responder a incidentes.

Solución Arquitectónica con AWS: Combina servicios de seguridad perimetral, detección de amenazas basada en ML y monitoreo integral:

- **AWS WAF:** Asociado a CloudFront o API Gateway para filtrar tráfico malicioso conocido (OWASP Top 10, bad bots, listas de IPs maliciosas) antes de que alcance la lógica de negocio.
- **Amazon Fraud Detector:** Se invoca desde la función Lambda de procesamiento. Utiliza modelos de Machine Learning (entrenados con datos históricos propios) para evaluar el riesgo de fraude de cada transacción en tiempo real y devolver una puntuación o decisión (Permitir, Revisar, Bloquear).

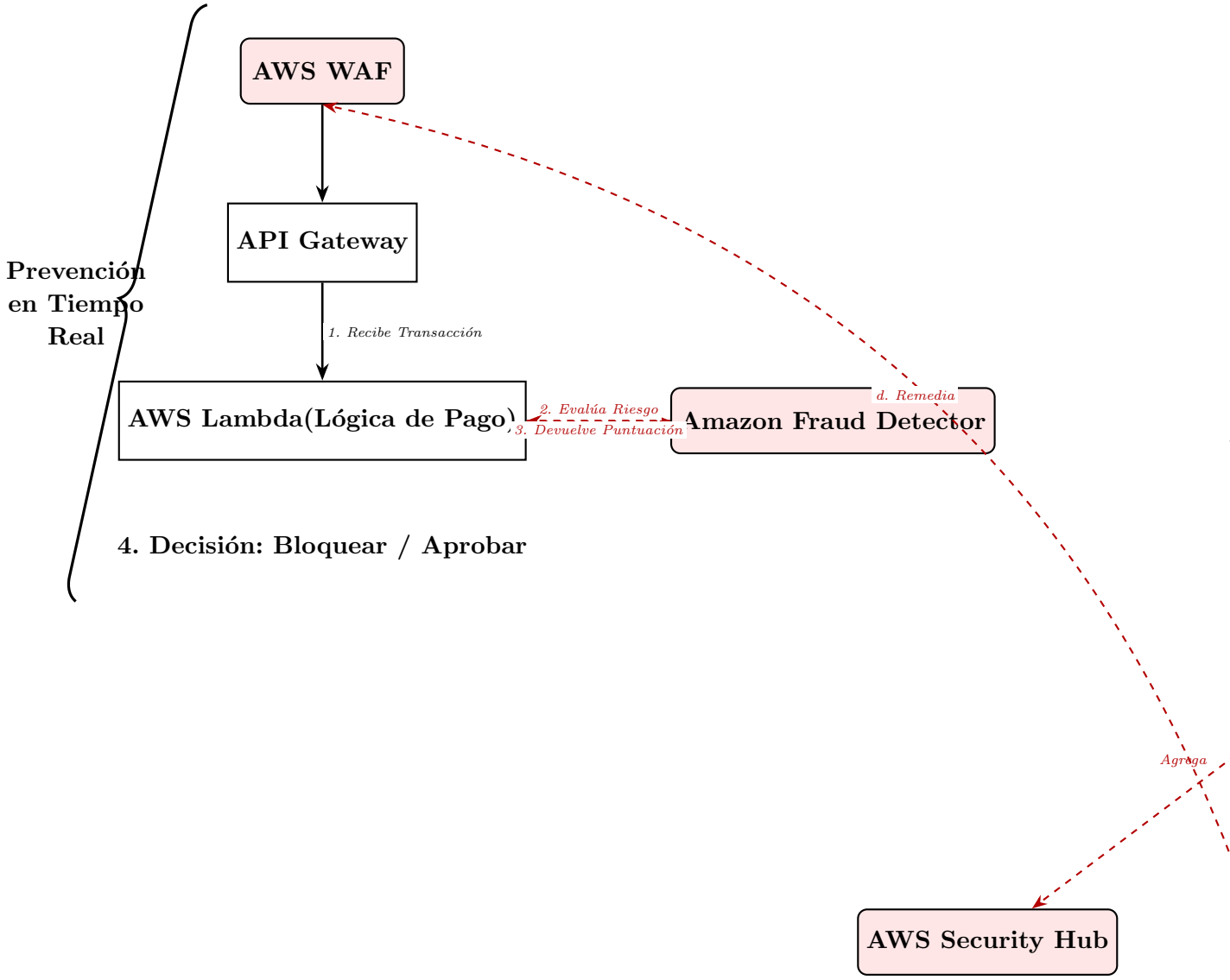

Figura 5.4: Diagrama de Protección contra Fraudes y Monitoreo Activo.

- **Amazon GuardDuty:** Monitorea de forma pasiva los logs de AWS (CloudTrail, VPC Flow Logs, DNS) para detectar actividades anómalas o maliciosas a nivel de infraestructura (ej. instancias comunicándose con IPs de C&C, escaneo de puertos).

- **Amazon CloudWatch + AWS Lambda (Respuesta):** CloudWatch recopila métricas y logs. Las alarmas de CloudWatch (ej. umbral de errores alto, alerta de GuardDuty reenviada

vía EventBridge) pueden disparar funciones Lambda para notificar al equipo de seguridad o realizar acciones de remediación básicas (ej. bloquear una IP en WAF o NACL).

- **AWS Security Hub:** Centraliza los hallazgos de GuardDuty, AWS Config, Inspector y otros servicios, proporcionando un dashboard unificado para la gestión de la postura de seguridad y la priorización de alertas.

Beneficios Clave:

- **Defensa en Profundidad:** Protección perimetral (WAF), detección específica de fraude transaccional (Fraud Detector) y detección de amenazas a nivel de infraestructura (GuardDuty).
- **Detección Inteligente:** Uso de ML para identificar fraudes y amenazas complejas.
- **Monitoreo Centralizado:** Visibilidad unificada de la seguridad (Security Hub) y el rendimiento (CloudWatch).
- **Respuesta Rápida:** Capacidad de alertar y automatizar respuestas a incidentes.

5.2.2.4. Escenario 4: Escalabilidad Dinámica durante Picos de Demanda (Eventos Comerciales Masivos) (Eficiencia del Rendimiento)

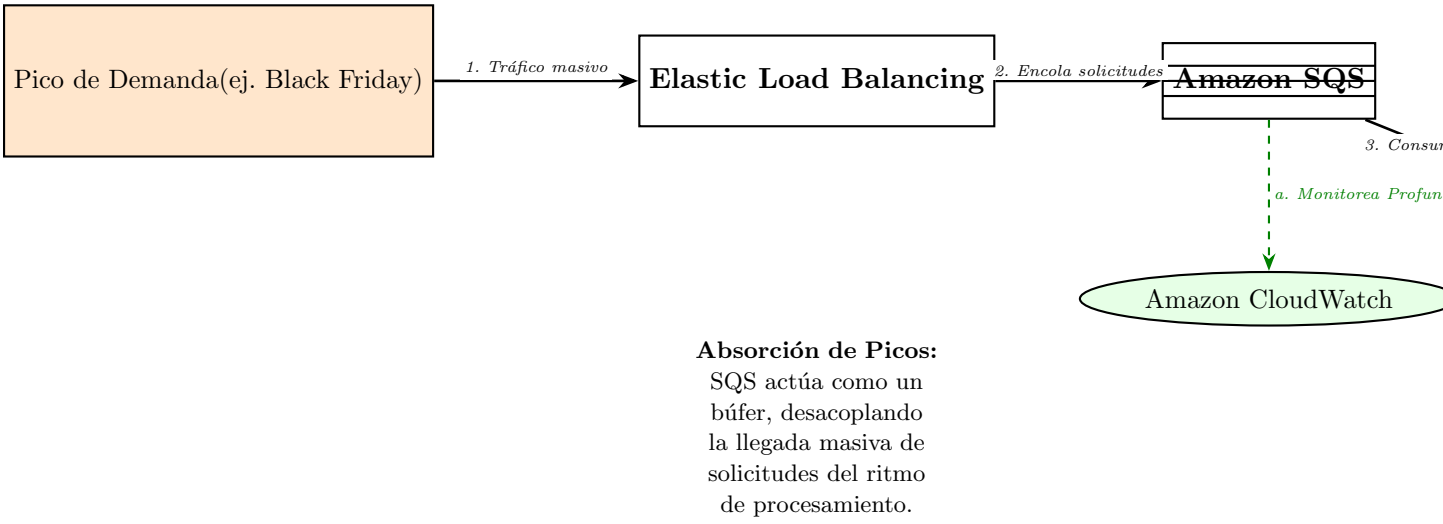

Figura 5.5: Diagrama de Escalabilidad Dinámica durante Picos de Demanda.

Descripción del Escenario: Durante eventos como Black Friday o Cyber Monday, el volumen de transacciones puede multiplicarse exponencialmente en cortos periodos. La arquitectura debe escalar automáticamente para manejar la carga sin degradar el rendimiento ni causar fallos.

Solución Arquitectónica con AWS: Se diseña para la elasticidad utilizando componentes serverless y auto-escalables:

- **AWS Auto Scaling:** Configurado para los recursos de cómputo. Para Lambda, se ajusta la *Provisioned Concurrency*. Para ECS/EKS, se escalan las tareas/pods y, si aplica, las instancias EC2 del clúster, basándose en métricas como uso de CPU, memoria o profundidad de colas SQS.
- **Elastic Load Balancing (ALB/NLB):** Escala automáticamente su capacidad para distribuir el tráfico creciente hacia el grupo de recursos de cómputo escalado.
- **Amazon DynamoDB (On-Demand o Auto Scaling):** El modo *On-Demand* es ideal para picos impredecibles, ya que escala la capacidad de lectura/escritura instantáneamente. Alternativamente, el modo *Provisioned con Auto Scaling* ajusta la capacidad basada en el consumo.
- **Amazon SQS (Standard Queues):** Actúan como un búfer desacoplador entre servicios (ej. entre API Gateway/Lambda y un servicio de procesamiento posterior). Absorben picos masivos de solicitudes, permitiendo que los consumidores (Lambda, ECS) procesen a su propio ritmo, evitando sobrecargas. Las colas FIFO garantizan orden pero tienen límites de TPS más bajos.
- **Amazon CloudFront / ElastiCache:** El cacheo de respuestas de API (lecturas) o datos frecuentemente accedidos reduce significativamente la carga en los servicios backend durante los picos.

Beneficios Clave:

- **Elasticidad Automática:** El sistema se adapta a la demanda sin intervención manual.
- **Alta Disponibilidad:** La distribución de carga y el escalado previenen puntos únicos de fallo y sobrecargas.
- **Resiliencia:** El uso de colas SQS permite que el sistema absorba picos sin perder solicitudes, incluso si los servicios de procesamiento se saturan temporalmente.
- **Optimización de Costos:** Se paga por la capacidad utilizada, evitando el sobreaprovisionamiento constante para picos infrecuentes.

Estos casos prácticos demuestran cómo una selección estratégica y la combinación adecuada de servicios AWS permiten construir un hub de pagos que no solo es funcional, sino también altamente eficiente, seguro y capaz de adaptarse dinámicamente a las exigentes condiciones del comercio electrónico moderno. Estos patrones y soluciones servirán como base fundamental para el diseño detallado de la arquitectura de referencia en el siguiente capítulo.

Capítulo 6

Propuesta de Arquitectura de Referencia para Hubs de pagos en AWS

6.1. Introducción al Capítulo

Los capítulos anteriores han establecido los requisitos fundamentales para un hub de pagos (Capítulo 4) y han evaluado las herramientas que AWS provee para satisfacerlos (Capítulo 5). Este capítulo sintetiza dichos hallazgos para presentar el artefacto central de esta investigación: una arquitectura de referencia detallada para hubs de pago en AWS.

El propósito de esta arquitectura es materializar las mejores prácticas de los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework. Para ello, se definirán los principios de diseño y las estrategias arquitectónicas adoptadas, se presentará la arquitectura utilizando el modelo C4 en sus diferentes niveles de abstracción, y se justificarán las decisiones clave de diseño, estableciendo una guía validada para la construcción de sistemas de pago modernos, resilientes y seguros en la nube.

6.1.1. Propósito y alcance de la arquitectura de referencia propuesta.

Este capítulo consolida los aprendizajes anteriores con el propósito de **proponer y validar una arquitectura de referencia detallada para un hub de pagos en AWS**. Dicha arquitectura está diseñada para:

- **Optimizar las operaciones de pago en términos de rendimiento:** Asegurando baja latencia, alta capacidad de procesamiento (transacciones por segundo) y escalabilidad elástica para gestionar eficientemente los picos de demanda característicos del comercio electrónico.
- **Asegurar la protección contra fraudes y otros riesgos de seguridad:** Implementando una estrategia de defensa en profundidad que proteja los datos de las transacciones, las credenciales de los usuarios y la infraestructura contra amenazas internas y externas, facilitando el cumplimiento de normativas como PCI DSS.
- **Integrar las mejores prácticas del AWS Well-Architected Framework:** Aplicando los principios de diseño y las recomendaciones específicas de los pilares de Eficiencia del Rendimiento y Seguridad.

El **alcance** de este capítulo comprende:

- La definición de los **principios de diseño y estrategias arquitectónicas** que guían la solución.
- La presentación de la **arquitectura de referencia en sus niveles lógico y detallado por capas**, identificando los servicios AWS específicos y sus interacciones.
- La **justificación de las decisiones de diseño**, alineando la arquitectura propuesta con los requisitos y desafíos identificados previamente.
- La **validación de la arquitectura mediante una Prueba de Concepto (PoC)**, enfocada en aspectos clave de rendimiento y seguridad, cuyos resultados serán analizados críticamente.
- La discusión de **consideraciones prácticas** para la implementación y operación de la arquitectura en entornos reales.

Este capítulo busca, por tanto, no solo diseñar una solución teórica, sino también proporcionar una base validada que pueda servir como guía para organizaciones que buscan construir o modernizar sus hubs de pago en la nube de AWS, maximizando la eficiencia y la seguridad en sus operaciones de comercio electrónico.

6.1.2. Principios de Diseño y Estrategias Arquitectónicas

El diseño de la arquitectura de referencia para el hub de pagos se fundamenta en un conjunto de principios y estrategias derivados directamente del AWS Well-Architected Framework y de las conclusiones obtenidas en el Objetivo Específico 2. Estos fundamentos buscan garantizar que la arquitectura no solo cumpla con los requisitos funcionales y no funcionales identificados en el Objetivo Específico 1, sino que también encarne las mejores prácticas para operar de manera eficiente y segura en la nube de AWS.

6.1.2.1. Principios de Diseño basados en AWS Well-Architected Framework

Aunque el alcance de esta tesis se centra en los pilares de Eficiencia del Rendimiento y Seguridad, un diseño robusto inherentemente considera aspectos de otros pilares como habilitadores o consecuencias. Los siguientes principios, extraídos principalmente de los pilares focales, guiarán las decisiones de diseño:

- **Principios de Eficiencia del Rendimiento (Performance Efficiency Pillar):**
  - **Democratizar tecnologías avanzadas (Democratize advanced technologies):** Se priorizará el uso de servicios gestionados de AWS (ej. DynamoDB, Lambda, API Gateway) en lugar de construir y mantener soluciones equivalentes auto-gestionadas. Esto permite al equipo de desarrollo del hub de pagos enfocarse en la lógica de negocio y aprovechar la experiencia de AWS en la optimización de estos servicios (AWS Well-Architected Framework, 2024, p. 40).

- **Globalizarse en minutos (Go global in minutes):** Aunque no es un foco primario del diseño inicial, la arquitectura debe considerar patrones que faciliten una futura expansión global. Esto implica seleccionar servicios con presencia global y considerar estrategias de despliegue multi-región si los requisitos de negocio lo demandasen (AWS Well-Architected Framework, 2024, p. 40).
- **Usar arquitecturas sin servidor (Use serverless architectures):** Se favorecerán enfoques serverless (ej. AWS Lambda, AWS Fargate) para los componentes de procesamiento y lógica de negocio. Esto elimina la carga operativa de la gestión de servidores y permite un escalado automático y granular, optimizando tanto el rendimiento como los costos transaccionales (AWS Well-Architected Framework, 2024, p. 40).
- **Experimentar más a menudo (Experiment more often):** La arquitectura debe ser lo suficientemente flexible para permitir la experimentación con diferentes tipos de instancia, configuraciones o incluso servicios alternativos, facilitando la optimización continua del rendimiento. La infraestructura como código (IaC) será clave para habilitar este principio (AWS Well-Architected Framework, 2024, p. 40).
- **Considerar la simpatía mecánica (Consider mechanical sympathy):** Se seleccionarán los servicios y configuraciones de AWS que mejor se alineen con los patrones de acceso a datos y las características de la carga de trabajo del hub de pagos (ej. DynamoDB para acceso de baja latencia a datos transaccionales, Aurora para datos relacionales con consistencia fuerte) (AWS Well-Architected Framework, 2024, p. 40).

- **Principios de Seguridad (Security Pillar):**

  - **Implementar una base de identidad sólida (Implement a strong identity foundation):** Se aplicará el principio de menor privilegio mediante AWS Identity and Access Management (IAM), utilizando roles y políticas granulares para cada componente y usuario. Se centralizará la gestión de identidades y se evitará el uso de credenciales estáticas de larga duración (AWS Well-Architected Framework, 2024, p. 21).
  - **Habilitar la trazabilidad (Enable traceability):** Todas las acciones y cambios en el entorno serán monitoreados y registrados (ej. mediante AWS CloudTrail, logs de API Gateway, logs de VPC Flow). Esta trazabilidad es fundamental para la auditoría, la investigación de incidentes y la respuesta automática (AWS Well-Architected Framework, 2024, p. 21).
  - **Aplicar seguridad en todas las capas (Apply security at all layers):** Se adoptará un enfoque de defensa en profundidad, implementando controles de seguridad en cada capa de la arquitectura: desde el borde de la red (ej. AWS WAF, AWS Shield), pasando por la red virtual (VPC, Security Groups), hasta la aplicación (autenticación/autorización en API Gateway, seguridad en Lambda) y los datos (cifrado con KMS) (AWS Well-Architected Framework, 2024, p. 21).
  - **Automatizar las mejores prácticas de seguridad (Automate security best practices):** Se buscará automatizar la implementación y validación de controles de

seguridad (ej. mediante IaC con CloudFormation, reglas de AWS Config, respuestas automatizadas a hallazgos de GuardDuty). Esto reduce el error humano y permite escalar la seguridad de manera eficiente (AWS Well-Architected Framework, 2024, p. 21).

- **Proteger los datos en tránsito y en reposo (Protect data in transit and at rest):** Todos los datos sensibles, especialmente los datos de titulares de tarjetas (CHD), se clasificarán y protegerán mediante cifrado (ej. TLS para datos en tránsito, cifrado con AWS KMS para datos en reposo en DynamoDB, S3, etc.) y controles de acceso estrictos (AWS Well-Architected Framework, 2024, p. 21).
- **Mantener a las personas alejadas de los datos (Keep people away from data):** Se minimizará la necesidad de acceso humano directo a los datos sensibles mediante la automatización de procesos y el uso de roles y credenciales temporales para tareas específicas (AWS Well-Architected Framework, 2024, p. 21).
- **Prepararse para eventos de seguridad (Prepare for security events):** Aunque la prevención es clave, la arquitectura incluirá mecanismos para la detección (ej. Amazon GuardDuty, CloudWatch Alarms), respuesta (ej. playbooks automatizados con Lambda) y recuperación ante incidentes de seguridad (AWS Well-Architected Framework, 2024, p. 21).

Estos principios servirán como directrices fundamentales en la selección de servicios, la definición de patrones y la configuración de cada componente de la arquitectura de referencia.

6.1.2.2. Estrategias Arquitectónicas Clave

Derivadas de los principios anteriores y de las conclusiones del OE2, se adoptarán las siguientes estrategias arquitectónicas para el diseño del hub de pagos:

Arquitectura Basada en Microservicios y/o Funciones Serverless: **Descripción:** Descomponer las funcionalidades del hub de pagos (ej. autorización, captura, enrutamiento, prevención de fraude, notificaciones) en servicios pequeños, independientes y desplegables de forma autónoma. Se priorizará el uso de AWS Lambda para la lógica de estos servicios, complementado por Amazon ECS/Fargate para componentes que puedan requerir ejecuciones de mayor duración o un control más granular del entorno.

Justificación: Esta estrategia promueve la agilidad, escalabilidad granular (escalar solo los servicios que lo necesitan), resiliencia (un fallo en un microservicio no tiene por qué afectar a todo el sistema) y facilita el mantenimiento por equipos más pequeños y enfocados. Se alinea con el principio de Üsar arquitecturas sin servidor"para la eficiencia del rendimiento y permite la implementación de "Seguridad en todas las capas.ªl poder aplicar políticas de seguridad específicas a cada microservicio.

API-Driven y Desacoplamiento mediante Eventos: **Descripción:** Todos los microservicios expondrán sus funcionalidades a través de APIs bien definidas, gestionadas por Amazon API Gateway. La comunicación entre microservicios, cuando sea apropiado, se realizará de

forma asíncrona mediante colas (Amazon SQS) o buses de eventos (Amazon EventBridge, Amazon SNS) para mejorar la resiliencia y el desacoplamiento.

Justificación: Un enfoque API-first facilita la integración interna y con sistemas externos. El desacoplamiento mediante eventos reduce las dependencias directas, mejora la tolerancia a fallos (si un servicio consumidor está temporalmente caído, los mensajes pueden esperar en una cola) y permite que los servicios escalen de forma independiente. Esto es crucial para la .Eficiencia del Rendimientoz la "Fiabilidad"(aunque este último pilar no sea el foco principal, es un beneficio inherente).

Estrategia de Escalabilidad Automática (Auto-Scaling): **Descripción:** Utilizar las capacidades de AWS Auto Scaling para todos los recursos de cómputo (Lambda, ECS/Fargate, EC2 si se usara) y bases de datos (DynamoDB, Aurora Auto Scaling) para ajustar dinámicamente la capacidad en respuesta a la demanda real. Esto incluye escalar horizontalmente (más instancias/contenedores/concurrencia) y potencialmente verticalmente donde aplique.

Justificación: Permite manejar los picos de transacciones del comercio electrónico sin intervención manual, optimizando el rendimiento durante alta demanda y reduciendo costos durante baja demanda, cumpliendo directamente con el principio de "Detener las conjeturas sobre las necesidades de capacidad"(AWS Well-Architected Framework, 2024, p. 6) y el pilar de Eficiencia del Rendimiento.

Seguridad en Profundidad (Defense-in-Depth) y Enfoque de Confianza Cero (Zero Trust):
Descripción: En el contexto de AWS, un enfoque de 'Confianza Cero' significa que no se confía implícitamente en ninguna interacción, incluso si ocurre dentro de la VPC. Cada interacción entre microservicios debe ser autenticada y autorizada explícitamente. Esto puede implementarse mediante mecanismos como la autenticación TLS mutua (mTLS), que puede ser gestionada con un service mesh como AWS App Mesh, o asegurando que cada servicio valide un token de identidad (ej. JWT) portado por la solicitud, verificando sus permisos antes de procesarla. Se implementarán múltiples controles de seguridad: en el perímetro (AWS WAF, Shield), en la red (VPCs, Security Groups, Network ACLs, PrivateLink), en el acceso a APIs (API Gateway con autorización), en la identidad (IAM con mínimo privilegio), en la protección de datos (cifrado en tránsito y en reposo con KMS), y en la detección de amenazas (GuardDuty, Security Hub). Cada solicitud y acceso será autenticado y autorizado explícitamente.

Justificación: Esta estrategia minimiza la superficie de ataque y reduce el impacto de una posible brecha, ya que un atacante que comprometa una capa enfrentará barreras adicionales. Se alinea directamente con el principio de .Aplicar seguridad en todas las capas"del WAF.

Infraestructura como Código (IaC) y Automatización CI/CD: **Descripción:** Definir toda la infraestructura (redes, cómputo, bases de datos, políticas de seguridad) utilizando herramientas como AWS CloudFormation o AWS CDK. Integrar estos templates en pipelines

de Integración Continua y Despliegue Continuo (CI/CD) usando AWS CodePipeline y AWS CodeBuild/CodeDeploy.

Justificación: Permite la reproducibilidad, consistencia y auditabilidad de la infraestructura. Facilita la automatización de despliegues, reduce errores manuales y permite la rápida reversión de cambios. Clave para .Automatizar las mejores prácticas de seguridadz para la .Experimentación frecuente.en rendimiento.

Observabilidad Integral: **Descripción:** Implementar una estrategia de monitoreo exhaustiva utilizando Amazon CloudWatch (Métricas, Logs, Alarmas, Dashboards), AWS X-Ray (rastreo distribuido) y AWS CloudTrail (auditoría de API calls). Configurar alertas proactivas para rendimiento, errores y eventos de seguridad.

Justificación: Proporciona la visibilidad necesaria para entender el comportamiento del sistema, diagnosticar problemas rápidamente, tomar decisiones basadas en datos para optimizar el rendimiento y detectar incidentes de seguridad. Es un habilitador fundamental para todos los pilares del WAF.

Estas estrategias y principios conformarán la columna vertebral de la arquitectura de referencia, asegurando que el diseño sea inherentemente eficiente, seguro y adaptable a las necesidades de un hub de pagos moderno en el contexto del comercio electrónico.

6.1.3. Mapeo Estratégico: Desafíos, Requisitos y Capacidades AWS

6.1.3.1. Introducción al Mapeo

La formulación de una arquitectura de referencia eficaz exige que cada decisión de diseño esté justificada y alineada con los objetivos del sistema. Los capítulos anteriores han proporcionado los insumos críticos para este proceso: el **Capítulo 4** delineó los desafíos técnicos y los requisitos prioritarios, mientras que el **Capítulo 5** evaluó el ecosistema de servicios de AWS y su idoneidad para construir soluciones de alto rendimiento y seguras.

Esta sección actúa como un puente crucial entre dicho análisis y la propuesta arquitectónica. Su propósito es presentar una **matriz de alineación estratégica consolidada** que conecta de manera explícita:

- Los desafíos clave identificados (D1-D7).
- Los principios del AWS Well-Architected Framework que guían la solución.
- Las capacidades estratégicas de AWS que se explotarán.
- Los servicios de AWS específicos que materializan dichas capacidades.

Esta matriz no solo justifica las elecciones tecnológicas, sino que también establece una trazabilidad clara para el diseño detallado que se presenta a continuación.

6.1.3.2. Matriz de Alineación Detallada

Tabla 6.1: Matriz de Alineación Estratégica: Desafíos, Principios WAF y Solución AWS.

ID	Desafío Clave	Principios WAF y Capacidades AWS	Servicios AWS Clave y Justificación
D2	Rendimiento y Escalabilidad bajo Carga Variable	**WAF (Rendimiento):** Usar arquitecturas serverless; Considerar simpatía mecánica. **Capacidad AWS:** Cómputo elástico, persistencia de baja latencia.	**AWS Lambda, ECS/Fargate:** Cómputo que escala automáticamente con la demanda. **Amazon DynamoDB:** Base de datos NoSQL con latencia de milisegundos. **Amazon ElastiCache:** Caché en memoria para acelerar lecturas.
D3	Seguridad y Prevención de Fraudes	**WAF (Seguridad):** Aplicar seguridad en todas las capas; Habilitar trazabilidad. **Capacidad AWS:** Defensa en profundidad, detección de amenazas.	**AWS WAF/Shield:** Protección perimetral. **IAM, KMS, Secrets Manager:** Control de acceso y cifrado. **Amazon GuardDuty, Fraud Detector:** Detección de amenazas y fraude.
D4	Cumplimiento Normativo Riguroso	**WAF (Seguridad):** Proteger datos en tránsito y en reposo; Habilitar trazabilidad. **Capacidad AWS:** Auditoría continua, gestión de configuración.	**AWS Config, CloudTrail:** Registro y monitoreo de cambios. **AWS Security Hub:** Dashboard de cumplimiento (PCI DSS). **AWS KMS:** Cifrado auditado de datos.
D6	Experiencia de Usuario y Soporte Multicanal	**WAF (Rendimiento):** Globalizarse en minutos. **Capacidad AWS:** Exposición de APIs unificadas, entrega de contenido de baja latencia.	**Amazon API Gateway:** Expone una API RESTful centralizada. **Amazon CloudFront:** Entrega la API y la UI más cerca de los usuarios para reducir la latencia.

Tabla 6.1 – continuación

ID	Desafío Clave	Principios WAF y Capacidades AWS	Servicios AWS Clave y Justificación
D1	Integración y Complejidad Sistémica	**WAF (Fiabilidad):** Principio de desacoplamiento. **Capacidad AWS:** Comunicación asíncrona, orquestación de flujos.	**Amazon SQS/SNS:** Desacoplan servicios para mayor resiliencia. **AWS Step Functions:** Orquesta flujos de trabajo complejos de manera visual y gestionada.
D5	Flexibilidad y Adaptabilidad Arquitectónica	**WAF (Excelencia Operativa):** Principio de infraestructura como código. **Capacidad AWS:** Despliegue automatizado, arquitectura modular.	**AWS CloudFormation/CDK:** Define la infraestructura como código. **AWS CodePipeline/CodeDeploy:** Automatizan el CI/CD, permitiendo una evolución rápida.
D7	Transformación Dinámica de Mensajes	**WAF (Rendimiento):** Democratizar tecnologías avanzadas. **Capacidad AWS:** Cómputo serverless para lógica de transformación.	**AWS Lambda:** Se utiliza en los adaptadores para transformar formatos de mensaje entre el hub y cada PSP de manera aislada y escalable.

6.1.3.3. Síntesis del Mapeo y Justificación Global de Enfoque

La matriz de alineación detallada precedente evidencia una correspondencia directa y robusta entre los desafíos técnicos, operativos y los requisitos funcionales y no funcionales identificados para un hub de pagos de comercio electrónico (provenientes de TG-HubsPago-OE1), y las capacidades estratégicas que ofrece la plataforma AWS, todo ello guiado por los principios de los pilares de Eficiencia del Rendimiento y Seguridad del Well-Architected Framework.

Se constata que:

- Para satisfacer los exigentes requisitos de **Eficiencia del Rendimiento** –tales como la escalabilidad masiva para picos de demanda, la baja latencia transaccional crítica para la experiencia del usuario, la integración flexible con múltiples pasarelas y la transformación dinámica de mensajes– una arquitectura que priorice servicios serverless (AWS Lambda), bases de datos NoSQL de alto rendimiento y escalabilidad (Amazon DynamoDB), complementada con caché

en memoria (Amazon ElastiCache), gestión y exposición de APIs (Amazon API Gateway) y distribución de contenido (Amazon CloudFront), se perfila como la solución más idónea. Este enfoque se alinea con principios WAF como üsar arquitecturas serverlessz "democratizar tecnologías avanzadas", permitiendo que la infraestructura responda de forma elástica a la demanda y optimice el uso de recursos.

- En cuanto a los imperativos de **Seguridad** –incluyendo la prevención de fraude en tiempo real, el cumplimiento riguroso de normativas como PCI DSS, la protección de datos sensibles en tránsito y en reposo, y la gestión segura de identidades y accesos– AWS proporciona un conjunto exhaustivo de servicios especializados. La estrategia de "seguridad en todas las capas"del WAF se materializa mediante la combinación de servicios como AWS WAF y AWS Shield para la protección perimetral, AWS IAM y Amazon Cognito para la gestión de identidades, AWS KMS para el cifrado de datos a lo largo de su ciclo de vida, AWS Secrets Manager para la protección de credenciales, y servicios de detección inteligente como Amazon GuardDuty y Amazon Fraud Detector. La trazabilidad y auditoría, fundamentales para el cumplimiento, se aseguran con AWS CloudTrail y AWS Config.
- La **observabilidad completa** del sistema, un requisito operativo fundamental que impacta tanto el rendimiento como la seguridad, se logra mediante la suite de Amazon CloudWatch (Métricas, Logs, Alarmas), AWS X-Ray para el rastreo distribuido y AWS CloudTrail para la auditoría de llamadas a la API. Esta capacidad es vital para el monitoreo proactivo, el diagnóstico rápido de incidentes y la optimización continua.
- La **flexibilidad y mantenibilidad** se abordan mediante la adopción de patrones de microservicios y la automatización de despliegues con herramientas de CI/CD como AWS CodePipeline y CodeDeploy, facilitando la evolución del hub de pagos.

Este mapeo estratégico no solo valida la elección de AWS como plataforma tecnológica, sino que también proporciona una justificación sólida para el enfoque arquitectónico que se detallará en la Sección 3.4. Dicho enfoque se caracterizará por una arquitectura serverless y/o basada en microservicios, con una estrategia de seguridad en profundidad (defense-in-depth), y una observabilidad integral embebida en el diseño. Las decisiones específicas para cada capa y componente de la arquitectura de referencia se derivarán directamente de esta alineación, buscando una implementación que materialice las mejores prácticas de los pilares de Eficiencia del Rendimiento y Seguridad del AWS Well-Architected Framework para el dominio específico de los hubs de pago en el comercio electrónico.

6.1.4. Propuesta de Arquitectura de Referencia para Hub de Pagos

6.1.4.1. Introducción a la Propuesta Arquitectónica

La presente sección detalla la arquitectura de referencia para un hub de pagos en el contexto del comercio electrónico, construida sobre la plataforma Amazon Web Services (AWS). Este diseño es la culminación del análisis de requisitos y desafíos acometido en el Objetivo Específico 1 y la

evaluación de servicios y principios del AWS Well-Architected Framework (WAF) realizada en el Objetivo Específico 2. El mapeo estratégico desarrollado en la Sección 3.3 ha establecido una clara correspondencia entre las necesidades del sistema y las capacidades de AWS, justificando las elecciones tecnológicas y los patrones de diseño que se materializarán en esta propuesta.

La arquitectura de referencia que se propone busca encapsular las funcionalidades *core* de un hub de pagos moderno, tales como el procesamiento de transacciones (autorización, captura), la integración con múltiples procesadores de pago (PSPs), la gestión de tokens, la prevención de fraude y la generación de notificaciones. Un énfasis particular se ha puesto en la aplicación rigurosa de los principios de los pilares de **Eficiencia del Rendimiento** y **Seguridad** del AWS Well-Architected Framework (AWS Well-Architected Framework, 2024). Esto se traduce en un diseño que prioriza la baja latencia, la alta escalabilidad para manejar picos transaccionales, la robustez en la protección de datos sensibles y la resiliencia operativa, elementos cruciales para la confianza y el éxito en el dinámico sector del comercio electrónico.

Para visualizar y comunicar esta arquitectura de manera efectiva, se ha adoptado el modelo C4 de Simon Brown Brown (2018). Este modelo facilita la descripción de sistemas de software en diferentes niveles de abstracción: Contexto (C1), Contenedores (C2), Componentes (C3) y Código (C4). En el marco de esta tesis, la arquitectura de referencia se detallará hasta el nivel de Componentes (C3).

Esta decisión de omitir el nivel de Código (C4) es deliberada y se fundamenta en las propias recomendaciones del autor del modelo y en las prácticas aceptadas en la industria. La creación y el mantenimiento de diagramas a nivel de código (e.g., diagramas de clases UML) ofrecen un retorno de inversión decreciente, dado que la estructura interna del código es altamente volátil y se vuelve obsoleta rápidamente. Herramientas especializadas como los Entornos de Desarrollo Integrado (IDEs) son más adecuadas para la exploración del código fuente. El propósito de una arquitectura de referencia es definir las fronteras estructurales, las responsabilidades y las interacciones entre los bloques de construcción principales, que es precisamente lo que los niveles C1, C2 y C3 capturan. Este enfoque permite un análisis riguroso de los pilares de Eficiencia del Rendimiento y Seguridad sin incurrir en un nivel de detalle de implementación que carecería de generalidad y vigencia.

Para facilitar la comprensión de la arquitectura propuesta, su descripción se organizará en dos niveles principales de abstracción:

- **Vista Lógica General:** Presentará una perspectiva de alto nivel del sistema, sus principales bloques funcionales y su interacción con el ecosistema circundante a través del modelo C4.
- **Arquitectura Detallada por Capas Funcionales:** Descompondrá el sistema en capas lógicas, identificando los componentes o microservicios clave dentro de cada una y los servicios AWS específicos que los implementan, justificando cada elección en términos de rendimiento y seguridad.

6.1.4.2. Vista Lógica General de la Arquitectura en modelo C4

Tras establecer los principios de diseño y las estrategias arquitectónicas que guiarán la solución, esta sección presenta una visión de alto nivel de la arquitectura de referencia para el hub de pagos. El

objetivo es delinear los principales bloques funcionales que componen el hub y cómo este interactúa con su ecosistema de actores y sistemas externos, sentando las bases para la descripción detallada por capas que seguirá. Esta perspectiva es fundamental para comprender el alcance global del sistema y el valor que aporta en el procesamiento de transacciones de comercio electrónico.

Diagrama de Contexto (Level 1: Context): Para visualizar el hub de pagos dentro de su entorno operativo, se propone un diagrama de contexto. Este diagrama representa el hub como una entidad central (çaja negra") e identifica los principales actores y sistemas externos que interactúan con él, así como los flujos de información de alto nivel.

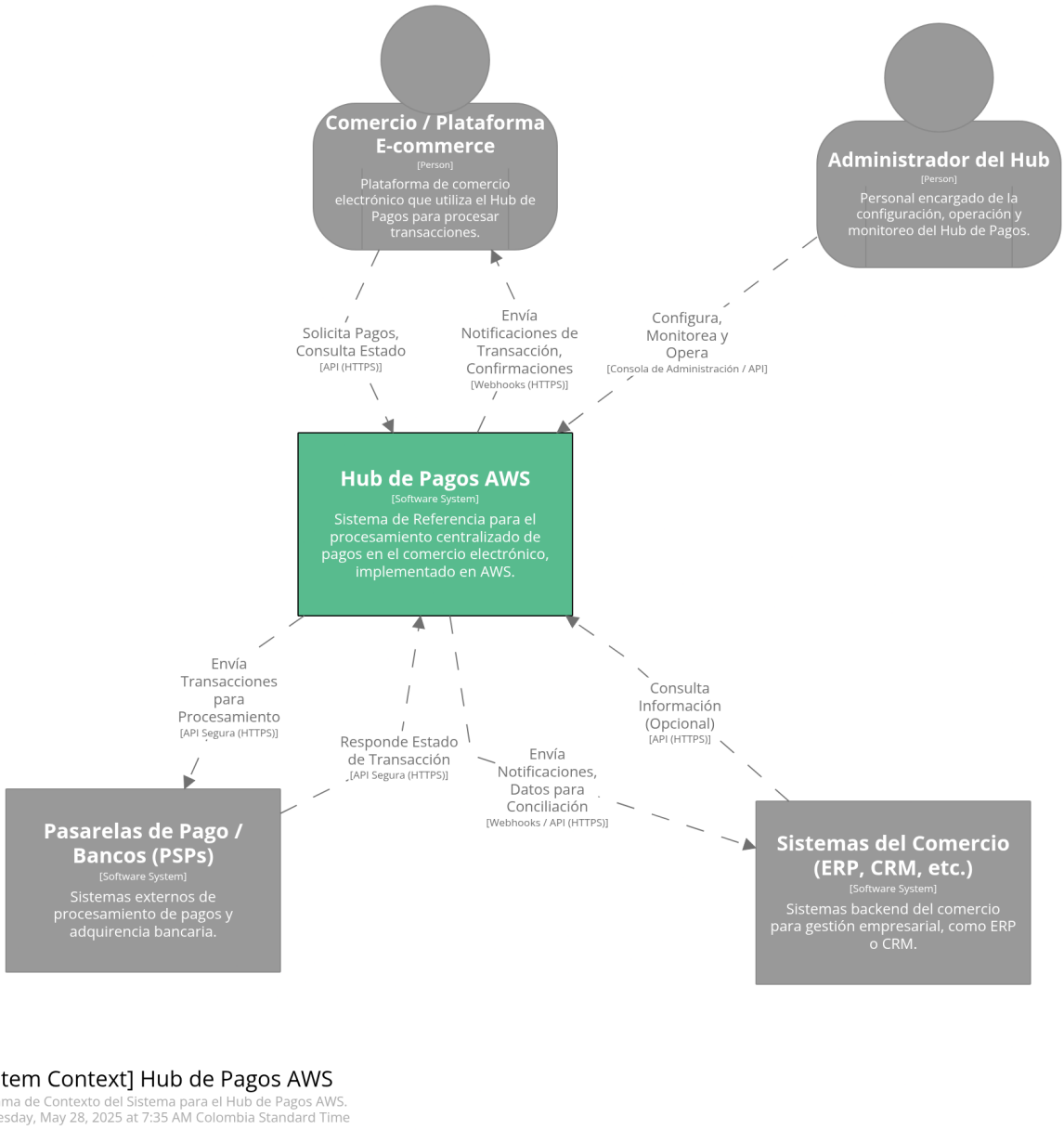

Figura 6.1: Diagrama de Contexto para hub de pagos en AWS.Elaboración propia.

Diagrama de Contenedores (Level 2: Containers): Internamente, el Hub de Pagos se concibe como un sistema compuesto por varios bloques funcionales lógicos interconectados, cada uno con

responsabilidades específicas. Estos bloques están diseñados para ser cohesivos y permitir un desacoplamiento que favorezca la escalabilidad y mantenibilidad, principios derivados de las estrategias de microservicios discutidas en la Sección 3.2.2.

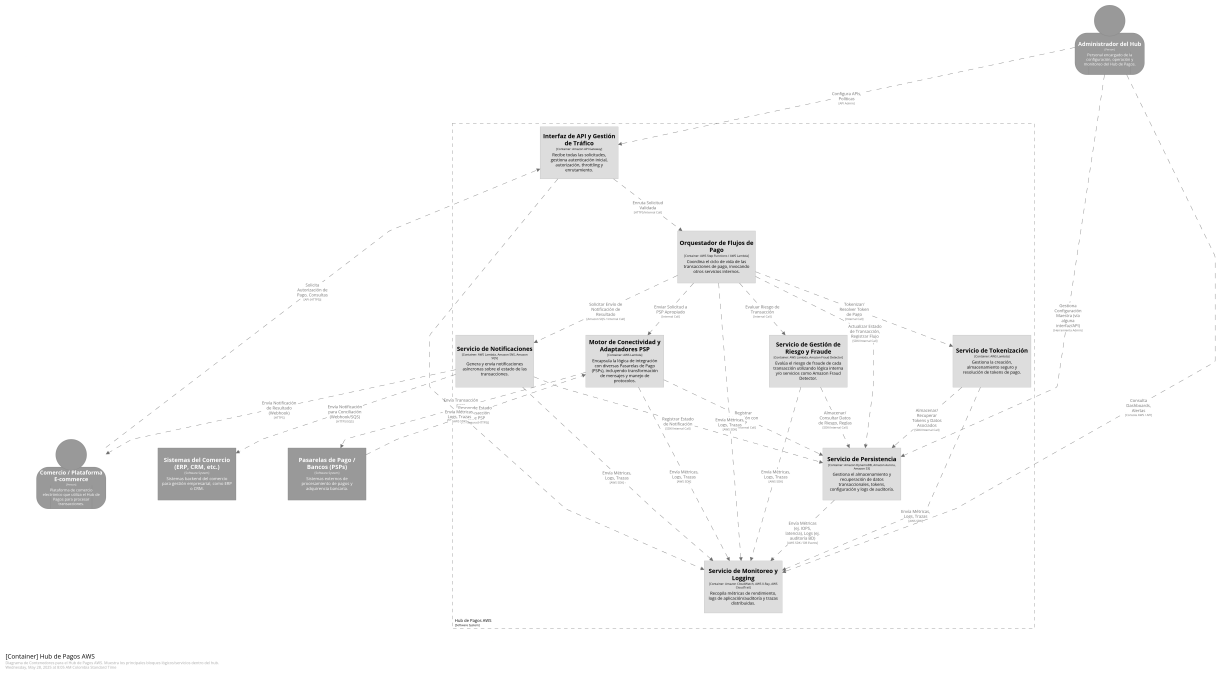

Figura 6.2: Diagrama de Contenedores para hub de pagos en AWS. Elaboración propia.

Descripción de Responsabilidades de Cada Bloque Lógico:

- **Interfaz de API y Gestión de Tráfico:** Responsable de exponer las funcionalidades del hub a través de APIs seguras y escalables. Maneja la autenticación de las solicitudes, la autorización inicial, el versionado de APIs, el throttling para proteger los servicios backend y el enrutamiento de las solicitudes al Orquestador de Flujos de Pago.
- **Orquestador de Flujos de Pago:** Es el cerebro del hub. Coordina el ciclo de vida completo de una transacción de pago, invocando secuencialmente a otros servicios internos como el de Riesgo y Fraude, Tokenización, Conectividad con PSPs y Notificaciones. Mantiene el estado general del flujo transaccional.
- **Motor de Conectividad y Adaptadores PSP:** Encapsula la complejidad de la integración con las diversas APIs de las pasarelas de pago (PSPs) y adquirentes. Contiene adaptadores específicos para cada PSP, responsables de la transformación de formatos de mensaje, manejo de protocolos de comunicación específicos y gestión de la lógica de reintentos particular de cada proveedor.

- **Servicio de Gestión de Riesgo y Fraude:** Evalúa el riesgo de cada transacción utilizando reglas predefinidas, modelos de machine learning (posiblemente integrándose con Amazon Fraud Detector) y/o conectándose a sistemas de detección de fraude de terceros. Proporciona una puntuación de riesgo o una decisión que influye en el flujo de la transacción.

- **Servicio de Tokenización:** Gestiona la creación, almacenamiento seguro y recuperación de tokens que representan datos de pago sensibles (ej. PANs). Esta funcionalidad es crucial para reducir el alcance de la conformidad con PCI DSS, ya que permite a los comercios operar sin manejar directamente los datos de tarjeta.

- **Servicio de Notificaciones:** Responsable de generar y enviar notificaciones asíncronas sobre el estado de las transacciones a los comercios (vía webhooks, colas de mensajes) o a otros sistemas internos que requieran esta información.

- **Servicio de Persistencia Centralizado:** Provee una capa de abstracción para el almacenamiento y recuperación de todos los datos relevantes del hub, incluyendo detalles de transacciones, tokens, configuraciones de comercios y PSPs, y logs de auditoría detallados. Asegura la durabilidad y consistencia de los datos.

- **Servicio de Monitoreo y Logging (Observabilidad):** Recopila métricas de rendimiento, logs de aplicación y auditoría, y trazas distribuidas de todos los componentes del hub. Facilita la supervisión en tiempo real, la detección de anomalías, el diagnóstico de problemas y la generación de alertas.

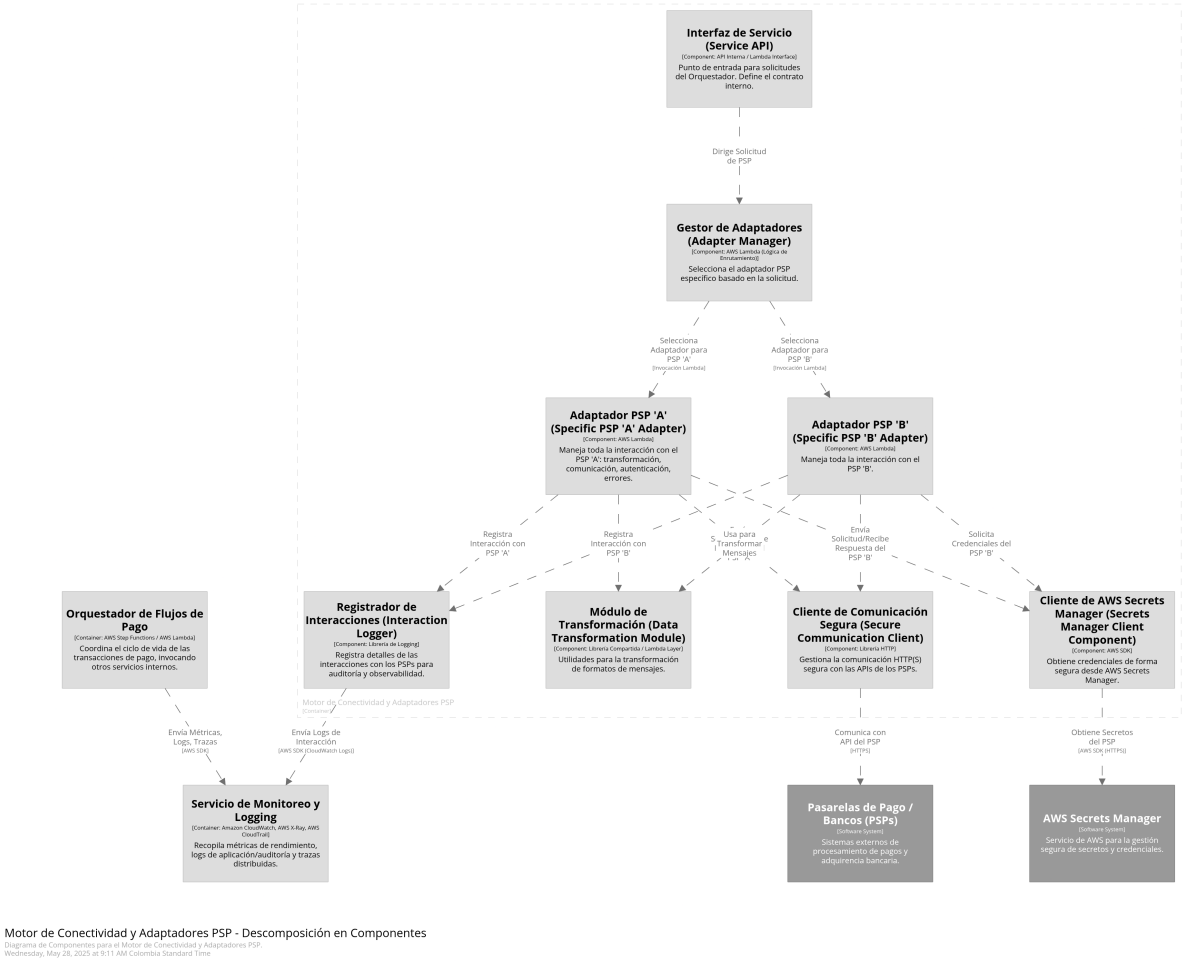

Figura 6.3: Diagrama de Componentes - Servicio de Conectividad y Adaptadores PSP (PSP Connectivity & Adapter Service). Elaboración propia.

Diagrama de Componentes (Level 3: Components)

Nombre Formal del Contenedor: Servicio de Conectividad y Adaptadores PSP (PSP Connectivity & Adapter Service)

- **Propósito Principal:** El propósito fundamental de este contenedor es actuar como una **capa de abstracción y mediación** entre el núcleo del hub de pagos y la multitud de sistemas externos de Pasarelas de Pago (PSPs) y adquirentes bancarios. Encapsula toda la complejidad técnica inherente a la comunicación con estas APIs heterogéneas, cada una con sus propios formatos de mensaje, protocolos de comunicación, mecanismos de autenticación y lógicas de

manejo de errores.

- **Importancia Estratégica y Extensibilidad:** La extensibilidad es un principio arquitectónico clave para los hubs de pago, y este contenedor es el principal habilitador de dicha característica:
  - **Facilita la Adición de Nuevas PSPs:** Al tener una arquitectura basada en adaptadores específicos para cada PSP, integrar una nueva pasarela de pago se reduce, idealmente, a desarrollar e implementar un nuevo adaptador, sin necesidad de modificar el núcleo del hub de pagos ni los adaptadores existentes.
  - **Aísla Cambios en APIs de PSPs:** Las APIs de las pasarelas de pago pueden cambiar. Este contenedor aísla esos cambios dentro del adaptador específico, protegiendo al resto del hub de pagos de las repercusiones directas de dichas modificaciones. Solo el adaptador afectado necesitaría actualizarse.
  - **Evolución Independiente:** La lógica de conexión y la interacción con cada PSP pueden evolucionar de forma independiente. Esto permite, por ejemplo, optimizar la comunicación con un PSP particular o adaptarse a nuevas versiones de su API sin afectar la comunicación con otros.
- **Responsabilidades Clave / Funcionalidades:**
  - **Gestión de Adaptadores Específicos por PSP:** Contiene módulos de software (adaptadores) dedicados para cada PSP integrada. Cada adaptador conoce los detalles íntimos de cómo interactuar con su PSP correspondiente.
  - **Transformación de Mensajes (Request/Response):** Es responsable de traducir el formato de solicitud de pago genérico del hub al formato específico requerido por la API del PSP seleccionado, y viceversa, transformar la respuesta del PSP de vuelta a un formato genérico para el hub.
  - **Manejo de Protocolos de Comunicación:** Gestiona los protocolos de comunicación específicos utilizados por cada PSP (ej. REST/JSON, SOAP/XML, etc.).
  - **Gestión de Autenticación con PSPs:** Maneja los diversos mecanismos de autenticación requeridos por cada PSP (ej. claves API, certificados, OAuth), utilizando credenciales almacenadas de forma segura.
  - **Manejo de Errores y Lógica de Reintentos Específicos del PSP:** Implementa la lógica para interpretar los códigos de error específicos de cada PSP y ejecutar estrategias de reintentos adecuadas para esa pasarela en particular, considerando sus políticas y tasas límite.
  - **Comunicación Segura:** Asegura que la comunicación con las APIs externas de los PSPs se realice a través de canales seguros (ej. HTTPS).
- **Tecnologías AWS Propuestas y Justificación (según TG-HubsPago-OE3):**

- **AWS Lambda:** La arquitectura propone que cada adaptador específico de PSP se implemente como una función Lambda individual. **Justificación:** Esto permite un despliegue y actualización independiente por PSP, mejorando la mantenibilidad y el aislamiento. Las funciones Lambda escalan automáticamente según la demanda para cada adaptador, optimizando el rendimiento y los costos.
- **AWS Secrets Manager:** Utilizado para almacenar de forma segura las credenciales de API (claves, tokens, etc.) necesarias para autenticarse con cada PSP. **Justificación:** Las funciones Lambda que implementan los adaptadores acceden a estos secretos en tiempo de ejecución mediante permisos IAM específicos, lo que se adhiere al principio de mínimo privilegio y evita codificar secretos en el código.
- **Amazon VPC (con NAT Gateway / AWS PrivateLink) (Opcional):** Para los casos en que la comunicación con las APIs de los PSPs requiera IPs de origen fijas o conexiones privadas para mayor seguridad o cumplimiento. **Justificación:** Proporciona un entorno de red controlado y seguro para las comunicaciones salientes hacia los PSPs.

- **Interacciones Principales (dentro del modelo C4 propuesto):**
  - **Recibe Solicitudes de:** El .ºrquestador de Flujos de Pago", que le instruye a qué PSP conectarse y qué operación realizar (ej. autorización, captura).
  - **Envía Solicitudes y Recibe Respuestas de:** Las "Pasarelas de Pago / Bancos (PSPs).externas.
  - **Utiliza:** .ªWS Secrets Manager"para obtener las credenciales necesarias para autenticarse con los PSPs.
  - **Interactúa con (Implícito/Potencial):**
    - **"Servicio de Persistencia":** Podría registrar información detallada sobre los intentos de comunicación, las solicitudes/respuestas crudas (con datos sensibles ofuscados) para auditoría, trazabilidad y diagnóstico de problemas.
    - **"Servicio de Monitoreo y Logging":** Envía métricas (ej. latencia de llamadas a PSPs, tasas de error por adaptador) y logs detallados para la observabilidad del sistema.

- **Beneficios Clave en el Contexto de la Tesis (Eficiencia del Rendimiento y Seguridad):**
  - **Eficiencia del Rendimiento:**
    - **Escalabilidad Selectiva:** El uso de AWS Lambda para los adaptadores permite que solo los adaptadores para los PSPs activamente utilizados escalen, en lugar de escalar un monolito de conexiones.
    - **Despliegues Aislados:** La actualización o el despliegue de un nuevo adaptador no interrumpe ni afecta el rendimiento de otros adaptadores ya en funcionamiento.

- **Mantenibilidad:** Al estar cada adaptador aislado, es más fácil de mantener y optimizar su rendimiento individualmente.

- **Seguridad:**
  - **Gestión Segura de Credenciales:** El uso de AWS Secrets Manager es crucial para proteger las sensibles credenciales de los PSPs, aplicando el principio de mínimo privilegio para su acceso.
  - **Conectividad Controlada:** Opciones como VPC, NAT Gateways o AWS PrivateLink permiten establecer comunicaciones seguras y controladas con los PSPs, limitando la exposición.
  - **Aislamiento de Fallos:** Un error o problema de seguridad en la integración con un PSP específico queda contenido dentro de su adaptador, reduciendo el riesgo de impacto en el resto del hub o en otras integraciones.

En resumen, el "Servicio de Conectividad y Adaptadores PSP.es un pilar de la arquitectura de referencia, diseñado para ser robusto, seguro y, fundamentalmente, extensible. Su correcta implementación mediante servicios como AWS Lambda y AWS Secrets Manager es clave para que el hub de pagos pueda crecer y adaptarse eficientemente a lo largo del tiempo.

Componentes Internos Propuestos para "Motor de Conectividad y Adaptadores PSP":

- **Interfaz de Servicio (Service API):**

  Descripción: Punto de entrada al contenedor. Define el contrato (API interna) que el .Orquestador de Flujos de Pagoütiliza para solicitar operaciones de PSP. Valida las solicitudes entrantes y las dirige al componente adecuado.

  Tecnología: Definición de API (ej. OpenAPI si es un servicio REST), interfaz de función Lambda.

- **Gestor de Adaptadores (Adapter Manager/Router):**

  Descripción: Componente central que recibe la solicitud validada desde la *Interfaz de Servicio*. Su lógica principal es seleccionar el *Adaptador PSP Específico* correcto basado en el identificador del PSP u otras reglas de enrutamiento.

  Tecnología: Lógica dentro de una función Lambda principal.

- **Adaptador PSP Específico (Specific PSP Adapter)** (Habrá múltiples instancias de este tipo de componente, uno por cada PSP):

  Descripción: El corazón de la extensibilidad. Cada adaptador es responsable de la comunicación completa con una pasarela de pago específica (ej. .Adaptador para VisaNet", .Adaptador para PSE", .Adaptador para PayPal").

  Sus responsabilidades incluyen:

- Transformar el mensaje de solicitud del hub al formato del PSP.
- Transformar la respuesta del PSP al formato del hub.
- Manejar el protocolo de comunicación específico del PSP.
- Gestionar la autenticación con el PSP (utilizando el *Cliente de AWS Secrets Manager*).
- Implementar la lógica de reintentos y manejo de errores específica para ese PSP.

Tecnología: Cada uno implementado como una función AWS Lambda independiente.

- **Módulo de Transformación (Data Transformation Module):**

  Descripción: Proporciona utilidades o un framework común para la transformación de datos entre el formato canónico del hub y los formatos específicos de los PSPs. Es utilizado por cada *Adaptador PSP Específico*.

  Tecnología: Librería compartida (ej. Lambda Layer) o código reutilizable dentro de las Lambdas de los adaptadores.

- **Cliente de Comunicación Segura (Secure Communication Client):**

  Descripción: Componente responsable de establecer y gestionar la comunicación HTTP(S) (u otro protocolo requerido) con las APIs externas de los PSPs. Maneja la configuración de TLS, timeouts, y podría incorporar lógica para AWS PrivateLink si es necesario.

  Tecnología: Librerías HTTP estándar (ej. Axios en Node.js, Requests en Python) dentro de las Lambdas, configuradas con mejores prácticas de seguridad.

- **Cliente de AWS Secrets Manager (Secrets Manager Client Component):**

  Descripción: Encapsula la lógica para interactuar de forma segura con AWS Secrets Manager y obtener las credenciales de API necesarias para que un *Adaptador PSP Específico* se autentique con su PSP.

  Tecnología: Uso del SDK de AWS para Secrets Manager dentro de las Lambdas de los adaptadores.

- **Registrador de Interacciones (Interaction Logger):**

  Descripción: Componente opcional pero recomendado, encargado de registrar detalles de las solicitudes enviadas y las respuestas (ofuscadas) recibidas de los PSPs. Esta información es vital para auditoría, trazabilidad y depuración. Los logs se envían al *Servicio de Monitoreo y Logging* (contenedor *observabilidad*).

  Tecnología: Librerías de logging, integrado con Amazon CloudWatch Logs.

6.1.4.3. Arquitectura Detallada por Capas Funcionales (Nivel de Contenedores/Componentes y Servicios AWS)

La arquitectura de referencia del hub de pagos se estructura en capas funcionales lógicas para promover la separación de responsabilidades, facilitar la escalabilidad independiente de sus componentes y aplicar controles de seguridad de manera granular. Este enfoque modular, derivado de las estrategias definidas en la sección 3.2.2, permite una mayor agilidad en el desarrollo y mantenimiento del sistema. Cada capa utiliza servicios específicos de AWS, seleccionados y configurados de acuerdo con los principios de Eficiencia del Rendimiento y Seguridad del Well-Architected Framework, y justificados por las evaluaciones realizadas en el Objetivo Específico 2.

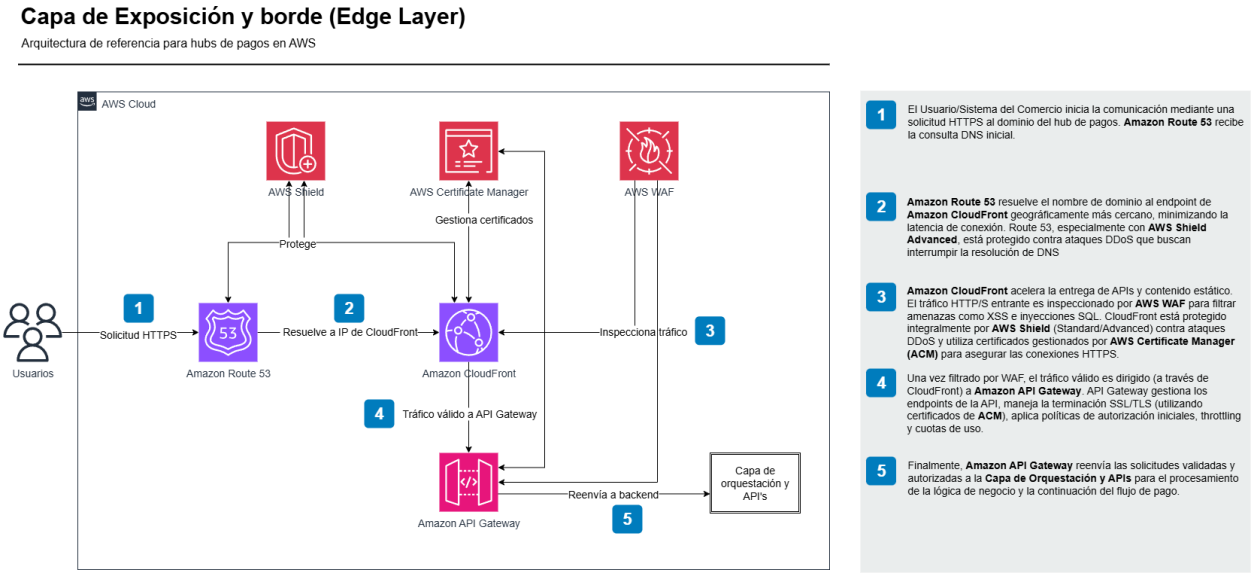

Figura 6.4: Capa de Exposición y Borde. Elaboración propia.

Capa de Exposición y Borde (Edge Layer)

- **Propósito:** Esta capa constituye el primer punto de contacto para todas las solicitudes entrantes dirigidas al hub de pagos y opera como la vanguardia de la infraestructura. Sus responsabilidades primarias son cruciales para la experiencia del usuario y la integridad del sistema:
  - Acelerar la entrega de respuestas y, si aplica, de contenido estático, minimizando la latencia.
  - Proporcionar la primera línea de defensa robusta contra una amplia gama de ataques a nivel de red y aplicación.

- Gestionar el enrutamiento global del tráfico, la resolución de nombres de dominio y la terminación segura de las conexiones SSL/TLS.
- Aplicar políticas iniciales de control de acceso, validación de solicitudes y limitación de velocidad (throttling) para proteger los recursos internos.

La **eficiencia del rendimiento** se optimiza en esta capa al reducir drásticamente la latencia para los usuarios finales mediante la distribución global de contenido y la descarga de tareas de los servicios de origen. La **seguridad** se refuerza al mitigar amenazas directamente en el borde, impidiendo que alcancen la infraestructura interna del hub de pagos.

- **Componentes y Funcionalidades Clave:**
  - **Sistema de Nombres de Dominio (DNS):** Resolución de nombres de dominio con enrutamiento inteligente y protección contra ataques a nivel de DNS.
  - **Red de Entrega de Contenido (CDN):** Distribución global de endpoints de API y cacheo de respuestas (si son cacheables) para reducir la latencia.
  - **Firewall de Aplicaciones Web (WAF):** Filtrado de tráfico HTTP/S en la Capa 7 para proteger contra exploits web comunes y tráfico malicioso.
  - **Protección DDoS:** Mitigación de ataques volumétricos (Capas 3/4) y a nivel de aplicación (Capa 7) que buscan degradar o interrumpir el servicio.
  - **Gestión de Certificados SSL/TLS:** Provisión, gestión y renovación de certificados para asegurar las comunicaciones HTTPS.
  - **Punto de Entrada y Gestión de APIs (Edge):** Terminación SSL/TLS, gestión inicial del tráfico API, autenticación/autorización básica y aplicación de cuotas/throttling.
- **Servicios AWS Clave y Justificación (Rendimiento y Seguridad):**
  - **Amazon Route 53:**

    Rendimiento: Servicio DNS global altamente disponible y escalable. Dirige a los usuarios a los endpoints de *Amazon CloudFront* o *Amazon API Gateway* más apropiados con baja latencia, utilizando políticas de enrutamiento avanzadas (latencia, geolocalización, conmutación por error).

    Seguridad: Proporciona una resolución de nombres de dominio fiable. Se integra con *AWS Shield Advanced* para una protección específica y robusta contra ataques DDoS dirigidos a las zonas alojadas DNS, previniendo la interrupción de la resolución de nombres.
  - **Amazon CloudFront:**

    Rendimiento: Actúa como una CDN global. Cachea respuestas de API (ej., consultas de estado de transacción no sensibles, configuraciones públicas) y sirve los endpoints de API más cerca de los usuarios finales, reduciendo significativamente la latencia. Optimiza

la entrega de APIs dinámicas, especialmente cuando se integra con API Gateway como origen.

Seguridad: Se integra nativamente con *AWS WAF* para filtrar tráfico malicioso en el borde y con *AWS Shield* (Standard y Advanced) para una protección DDoS integral. Gestiona certificados SSL/TLS (a través de su integración con *AWS Certificate Manager*) y permite aplicar restricciones geográficas.

- **AWS WAF:**

  Seguridad: Protege contra vulnerabilidades web comunes como inyecciones SQL, Cross-Site Scripting (XSS) y otros ataques definidos en el OWASP Top 10. Aplica reglas gestionadas o personalizadas a las solicitudes HTTP/S que llegan a *Amazon CloudFront* o directamente a *Amazon API Gateway*. Es un control esencial para requisitos como PCI DSS (ej. Req. 6.5).

  Rendimiento: Al bloquear tráfico malicioso y no deseado en el borde, reduce la carga y el riesgo en los servicios backend, permitiéndoles enfocar sus recursos en procesar solicitudes legítimas de manera eficiente.

- **AWS Shield (Standard y Advanced):**

  Seguridad: Shield Standard ofrece protección automática contra los ataques DDoS más comunes a nivel de red y transporte (Capas 3 y 4) para todos los servicios AWS sin costo adicional. Shield Advanced proporciona detección y mitigación de ataques más sofisticados y de mayor escala, tanto volumétricos como a nivel de aplicación (Capa 7), acceso al Equipo de Respuesta DDoS (DRT) de AWS y protección contra picos de costos inducidos por DDoS. Esta protección avanzada es crucial para la alta disponibilidad de un hub de pagos y se aplica a recursos como *Amazon Route 53* (zonas alojadas), *Amazon CloudFront*, y *Amazon API Gateway*.

  Rendimiento: Al absorber y mitigar el tráfico de ataques DDoS, Shield es fundamental para mantener la disponibilidad y el rendimiento esperado de la aplicación, incluso bajo condiciones de ataque.

- **Amazon API Gateway (Endpoints Edge-Optimized o Regionales con CloudFront):**

  La elección entre un endpoint Edge-Optimized (integración simplificada con la red de borde de CloudFront) y un endpoint Regional con una distribución de CloudFront gestionada por separado implica un trade-off. El endpoint Edge-Optimized es más fácil de configurar, pero una distribución de CloudFront separada ofrece un control mucho más granular sobre el comportamiento del caché (ej. TTLs por ruta), la capacidad de ejecutar lógica en el borde con Lambda@Edge o CloudFront Functions, y una configuración de seguridad WAF más desacoplada.

  Rendimiento: Sirve como el principal punto de entrada para las APIs del hub. Un endpoint Edge-Optimized se despliega en la red de borde de CloudFront, minimizando la latencia. Un endpoint regional puede usarse como origen para una distribución de CloudFront, obteniendo beneficios de caché y entrega global. Gestiona el throttling y

las cuotas de uso para proteger los servicios backend de ser sobrecargados por picos de solicitudes.

Seguridad: Maneja la terminación SSL/TLS (integrado con *AWS Certificate Manager* para la gestión de certificados). Puede integrarse directamente con *AWS WAF* para la protección de APIs y utiliza mecanismos de autorización (ej. IAM, Autorizadores Lambda, Grupos de Usuarios de Cognito) como una primera capa de control de acceso a las APIs.

- **AWS Certificate Manager (ACM):**

Seguridad: Aunque opera en segundo plano, ACM es un servicio fundamental en esta capa. Simplifica y automatiza la tarea de aprovisionar, gestionar y desplegar certificados SSL/TLS públicos y privados. Se integra directamente con *Amazon CloudFront* y *Amazon API Gateway* para habilitar HTTPS, asegurando que los datos en tránsito hacia y desde el hub de pagos estén cifrados.

Rendimiento: Al facilitar la implementación de HTTPS, contribuye indirectamente al rendimiento, ya que algunos protocolos modernos (como HTTP/2, ofrecido por CloudFront y API Gateway) requieren HTTPS y ofrecen mejoras de rendimiento.

Esta capa es vital para asegurar que el hub de pagos sea accesible globalmente, funcione con baja latencia y esté protegido contra una amplia gama de amenazas externas, sentando las bases para las capas internas de procesamiento y lógica de negocio. La correcta configuración y la elección de estos servicios son fundamentales para cumplir con los requisitos de rendimiento y seguridad de un hub de pagos moderno.

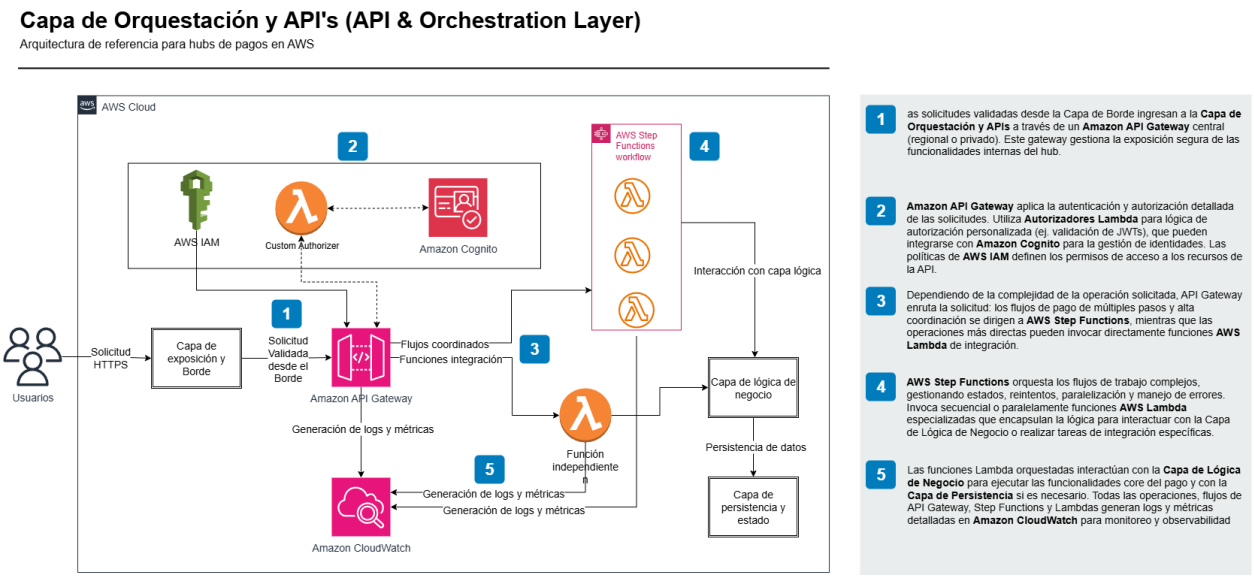

Figura 6.5: Capa de Orquestación y APIs. Elaboración propia.

Capa de Orquestación y APIs (API & Orchestration Layer)

- **Propósito:** Esta capa sirve como el punto centralizado para la gestión y exposición de las funcionalidades del hub de pagos, actuando como una fachada controlada hacia la lógica de negocio subyacente. Sus responsabilidades principales son:
  - Definir y exponer interfaces de programación de aplicaciones (APIs) consistentes, seguras y bien documentadas para los consumidores (comercios, sistemas internos).
  - Gestionar el ciclo de vida de las APIs, incluyendo versionado, control de acceso y políticas de uso (throttling, cuotas).
  - Orquestar flujos de trabajo complejos que involucran múltiples microservicios o pasos secuenciales/paralelos, especialmente para operaciones de pago que no son atómicas o que requieren coordinación entre diferentes dominios.
  - Validar, transformar (básicamente) y enriquecer las solicitudes antes de pasarlas a los servicios de lógica de negocio.

  Desde la perspectiva de **Eficiencia del Rendimiento**, esta capa optimiza la interacción con el backend mediante caching de respuestas, gestión de tráfico y desacoplamiento. Desde la **Seguridad**, impone la autenticación y autorización de solicitudes, protege contra abusos de API y reduce la superficie de ataque a los microservicios internos.

- **Componentes y Funcionalidades Clave:**

- **Exposición de Endpoints API:** Creación de endpoints RESTful o HTTP para todas las funcionalidades del hub (ej. iniciar pago, consultar estado, generar token, etc.).
- **Gestión de Autenticación y Autorización:** Verificación de la identidad del solicitante (ej. mediante API Keys, JWTs, firmas) y aplicación de permisos.
- **Validación de Solicitudes:** Comprobación de la estructura y tipos de datos de las solicitudes entrantes contra esquemas definidos.
- **Transformación de Datos (Básica):** Mapeo de formatos de solicitud/respuesta si es necesario antes de la interacción con los microservicios.
- **Enrutamiento de Solicitudes:** Dirección de las solicitudes al microservicio o función Lambda apropiado en la capa de lógica de negocio.
- **Gestión de Tráfico:** Aplicación de políticas de throttling (limitación de velocidad) y cuotas para prevenir sobrecargas y asegurar un uso justo.
- **Orquestación de Flujos (Opcional pero probable):** Coordinación de secuencias de llamadas a múltiples microservicios para operaciones complejas (ej. un flujo de reembolso que interactúa con transacciones, notificaciones y contabilidad).

▪ **Servicios AWS Clave y Justificación (Rendimiento y Seguridad):**

- **Amazon API Gateway:**

  Rendimiento: Servicio totalmente gestionado para crear, publicar, mantener, monitorear y asegurar APIs a cualquier escala. Escala automáticamente para manejar el tráfico de API. Ofrece caching de respuestas para reducir la latencia y la carga en los servicios backend. Permite configurar throttling y cuotas por API o por clave de API para proteger los servicios de lógica de negocio.

  Seguridad: Proporciona múltiples mecanismos de autenticación y autorización, incluyendo políticas de recursos IAM, autorizadores Lambda (para lógica personalizada, ej. validación de JWT) y grupos de usuarios de Amazon Cognito. Se integra con AWS WAF para protección adicional contra ataques web. Permite la validación de esquemas de solicitud. Es fundamental para el principio WAF de .ªplicar seguridad en todas las capas".

- **AWS Step Functions:**

  Rendimiento: Permite orquestar flujos de trabajo serverless que involucran múltiples funciones Lambda, servicios de AWS y APIs externas. Maneja estados, reintentos con backoff exponencial, paralelización y manejo de errores de manera visual y gestionada. Esto evita la necesidad de implementar lógica de orquestación compleja y propensa a errores dentro de funciones Lambda individuales, mejorando la mantenibilidad y la eficiencia del desarrollo. Optimiza la ejecución de procesos de larga duración o multipaso.

  Seguridad: Se integra con IAM para controlar los permisos de ejecución de las máquinas de estado y las funciones/servicios que invoca. Los logs de ejecución proporcionan trazabilidad para auditoría.

- **AWS Lambda (para Autorizadores y Transformaciones):**

  Rendimiento: Los autorizadores Lambda permiten implementar lógica de autenticación/autorización personalizada con baja latencia. Las integraciones Lambda con API Gateway pueden realizar transformaciones de datos rápidas si es necesario.

  Seguridad: Ejecutan código en un entorno aislado y seguro, con permisos controlados por IAM.

Esta capa es esencial para desacoplar a los consumidores de la complejidad interna del hub de pagos. API Gateway maneja las responsabilidades comunes de gestión de API, permitiendo que la Capa de Lógica de Negocio se enfoque en las funcionalidades específicas del dominio. Step Functions, si se utiliza, simplifica la gestión de procesos de negocio distribuidos, mejorando la fiabilidad y la observabilidad de flujos complejos, lo cual es un aspecto importante de la excelencia operativa que indirectamente apoya el rendimiento y la seguridad.

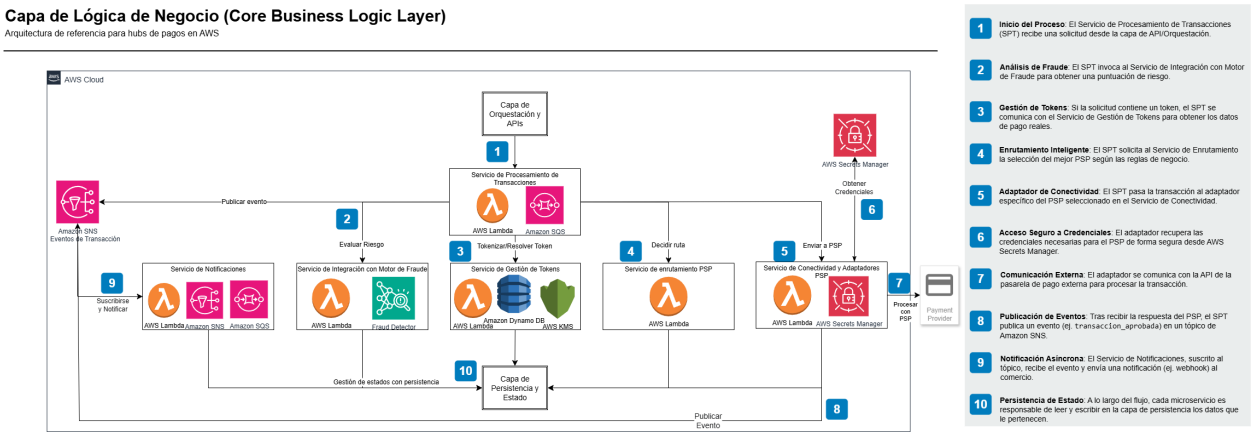

Figura 6.6: Capa de Lógica de Negocio. Elaboración propia.

Capa de Lógica de Negocio (Core Business Logic Layer)

- **Propósito:** Esta capa implementa las funcionalidades centrales y la inteligencia de negocio del hub de pagos. Es responsable de procesar las transacciones, aplicar reglas de negocio, interactuar con sistemas externos (como los PSPs y motores de fraude), y gestionar el ciclo de vida de los datos de pago. La eficiencia del rendimiento aquí se traduce en la capacidad de procesar un alto volumen de transacciones con baja latencia, mientras que la seguridad se enfoca en la protección de los datos durante su procesamiento y la lógica de prevención de fraude. Para lograr una arquitectura modular, mantenible y escalable, el diseño de esta capa se informa por los principios de *Domain-Driven Design (DDD)*. Esto implica identificar los subdominios clave dentro del hub de pagos y encapsular su lógica y datos dentro de

Contextos Delimitados (Bounded Contexts), que se materializan como microservicios o conjuntos cohesivos de funciones serverless. Este enfoque promueve la autonomía de los equipos, la claridad del modelo de dominio y la capacidad de evolucionar cada componente de forma independiente.

- **Componentes/Microservicios (Informados por DDD):** Los siguientes son los principales componentes (microservicios o dominios funcionales) identificados para esta capa, cada uno representando un Bounded Context:

  - **Servicio de Procesamiento de Transacciones (Transaction Processing Service):**

    Dominio DDD: Gestiona el ciclo de vida completo de una transacción financiera (ej. *Autorización*, *Captura*, *Reembolso*, *Anulación*). Contiene la lógica para validar, procesar y registrar el estado de cada *Transacción*.

    Funcionalidades Clave: Validar la estructura y contenido de las solicitudes de transacción. Aplicar reglas de negocio pre-procesamiento (ej. límites de monto, validaciones de comercio). Orquestar la interacción con el Servicio de Gestión de Riesgo y Fraude, el Servicio de Tokenización y el Servicio de Conectividad con PSPs. Actualizar el estado de la transacción en la capa de persistencia. Manejar reintentos y lógica de compensación para operaciones fallidas.

    Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, Amazon SQS, AWS Step Functions. Es crucial, sin embargo, evitar el anti-patrón conocido como el 'Lambda-lith', donde una única función Lambda acumula demasiadas responsabilidades, violando los principios de microservicios. Cada función debe adherirse al Principio de Responsabilidad Única (Single Responsibility Principle), enfocándose en una sola capacidad de negocio para mantener la modularidad y la mantenibilidad.

  - **Servicio de Enrutamiento de PSPs (PSP Routing Service):**

    Dominio DDD: Responsable de la selección inteligente de la pasarela de pago (PSP) o adquirente más adecuado para una transacción específica, basado en un conjunto de *ReglasDeEnrutamiento* configurables.

    Funcionalidades Clave: Mantener y aplicar reglas de enrutamiento (costo, tasa de aprobación, tipo de tarjeta, geografía del cliente, disponibilidad del PSP). Seleccionar dinámicamente el PSP óptimo. Gestionar la configuración de las prioridades de los PSPs.

    Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, Amazon DynamoDB/ElastiCache.

  - **Servicio de Conectividad y Adaptadores PSP (PSP Connectivity & Adapter Service):**

    Dominio DDD: Encapsula la complejidad técnica de la integración con las APIs heterogéneas de múltiples *PasarelasDePago*. Actúa como una capa de abstracción.

    Funcionalidades Clave: Implementar adaptadores específicos para cada PSP. Gestionar la comunicación con las APIs de los PSPs. Manejar errores específicos de cada

PSP.

Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, AWS Secrets Manager, (Opcional) Amazon VPC con NAT Gateway/PrivateLink.

- **Servicio de Gestión de Tokens (Tokenization Service):**

  Dominio DDD: Se encarga de la creación, almacenamiento seguro y resolución de *TokensDePago* que representan datos sensibles de tarjetas (PAN), minimizando el alcance de PCI DSS.

  Funcionalidades Clave: Generar tokens únicos para datos de tarjeta. Almacenar de forma segura la asociación entre tokens y datos de tarjeta. Proveer interfaces para tokenizar y de-tokenizar.

  Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, Amazon DynamoDB, AWS KMS.

- **Servicio de Integración con Motor de Fraude (Fraud Engine Integration Service):**

  Dominio DDD: Interactúa con uno o más *SistemasDeDeteccionDeFraude* para obtener una evaluación de riesgo para cada transacción.

  Funcionalidades Clave: Formatear datos para el motor de fraude. Enviar solicitud y recibir puntuación de riesgo. Aplicar lógica pre-autorización.

  Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, Amazon Fraud Detector, Amazon API Gateway.

- **Servicio de Notificaciones (Notification Service):**

  Dominio DDD: Responsable de enviar *Notificaciones* asíncronas sobre eventos importantes del ciclo de vida de las transacciones.

  Funcionalidades Clave: Consumir eventos de estado de transacción. Formatear y enviar notificaciones. Gestionar reintentos de notificación.

  Servicios AWS Clave y Justificación (Rendimiento y Seguridad): AWS Lambda, Amazon SNS, Amazon SQS, (Opcional) Amazon Pinpoint o Amazon SES.

- **Interacciones entre Componentes de la Capa:** Los microservicios dentro de esta capa interactúan principalmente de forma asíncrona a través de *Amazon SQS* y *Amazon SNS* para mejorar la resiliencia y la escalabilidad. Las llamadas síncronas directas se reservan para interacciones de muy baja latencia.

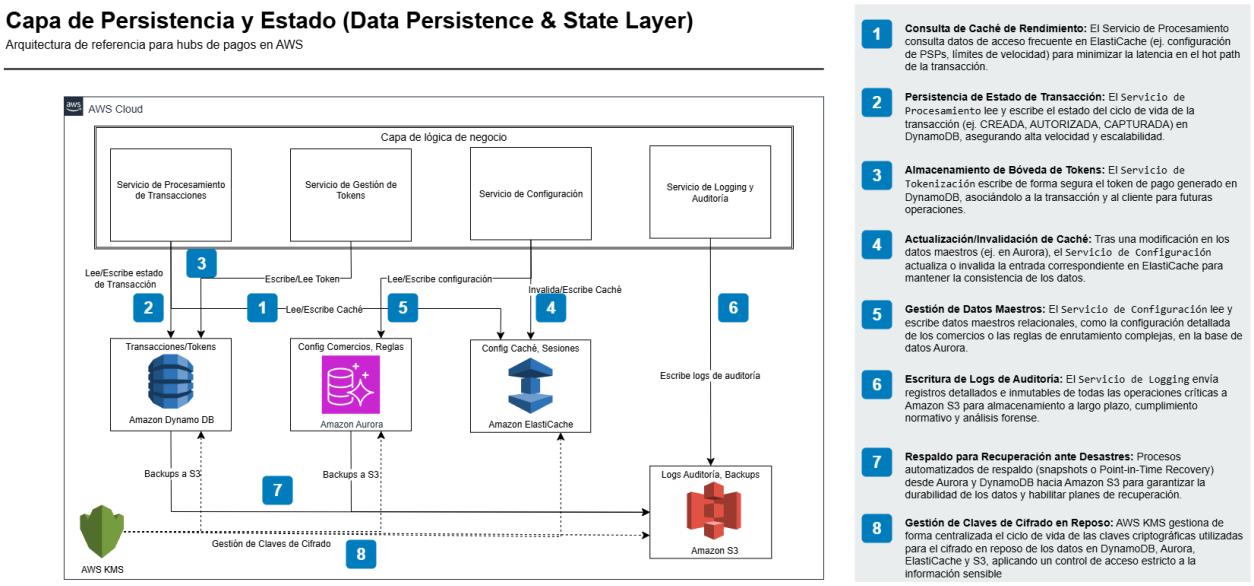

Figura 6.7: Capa de Persistencia y Estado. Elaboración propia.

Capa de Persistencia y Estado (Data Persistence & State Layer)

- **Propósito:** Esta capa es responsable de almacenar y gestionar de forma segura, duradera y eficiente todos los datos necesarios para la operación del hub de pagos. Esto incluye el estado de las transacciones, tokens de pago, configuraciones, logs de auditoría y datos de caché.
- **Componentes y Funcionalidades Clave:**
  - **Almacenamiento de Transacciones:** Persistencia del estado y detalles de cada operación de pago.
  - **Bóveda de Tokens (Token Vault):** Almacenamiento seguro de tokens de pago.
  - **Almacenamiento de Configuración:** Datos de configuración de comercios, reglas de enrutamiento, etc.
  - **Logs de Auditoría:** Registros detallados de operaciones para cumplimiento.
  - **Caché de Datos:** Almacenamiento temporal para reducir la latencia.
- **Servicios AWS Clave y Justificación (Rendimiento y Seguridad):**
  - **Amazon DynamoDB:** Ideal para almacenar el estado de las transacciones en tiempo real y los tokens de pago, donde el acceso rápido y la escalabilidad masiva son cruciales. Ofrece cifrado en reposo y control de acceso granular.
  - **Amazon Aurora (Compatible con MySQL/PostgreSQL):** Adecuada para almacenar datos relacionales que requieren consistencia fuerte (ACID) o consultas complejas, como la configuración de comercios o reglas de negocio.

- **Amazon ElastiCache (para Redis o Memcached):** Utilizado para almacenar en caché datos de acceso frecuente, reduciendo la carga en las bases de datos primarias y mejorando los tiempos de respuesta.
- **Amazon S3 (Simple Storage Service):** Ideal para almacenar logs de auditoría a largo plazo, backups de bases de datos, y datos de configuración de gran tamaño.

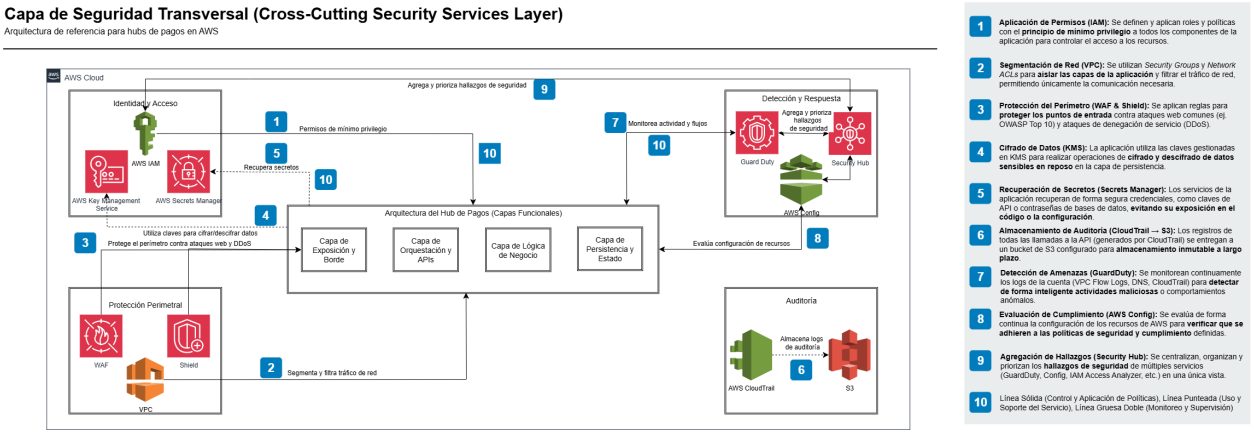

Figura 6.8: Capa de Seguridad Transversal. Elaboración propia.

Capa de Seguridad Transversal (Cross-Cutting Security Services Layer)

- **Propósito:** Esta capa permea todas las demás capas. Su propósito es implementar y gestionar los controles de seguridad fundamentales que protegen los activos, aseguran el cumplimiento normativo (PCI DSS) y mitigan riesgos.
- **Componentes y Funcionalidades Clave:**
  - **Gestión de Identidades y Accesos (IAM):** Controlar quién puede acceder a qué recursos.
  - **Gestión de Claves Criptográficas:** Creación, rotación y uso de claves de cifrado.
  - **Gestión de Secretos:** Almacenamiento seguro de credenciales.
  - **Detección de Amenazas:** Identificación proactiva de actividades maliciosas.
  - **Protección de Red y Aplicaciones:** Firewalls, protección DDoS.
  - **Auditoría y Cumplimiento:** Registro de actividades y monitoreo de configuración.
- **Servicios AWS Clave y Justificación (Seguridad):**
  - **AWS Identity and Access Management (IAM):** Base para ïmplementar una base de identidad sólidaçon el principio de mínimo privilegio.

- **AWS Key Management Service (KMS):** Esencial para "proteger los datos en tránsito y en reposo".
- **AWS Secrets Manager:** Almacena, rota y gestiona el acceso a secretos como claves API de PSPs.
- **Amazon GuardDuty:** Servicio de detección inteligente de amenazas.
- **AWS Security Hub:** Agrega, organiza y prioriza los hallazgos de seguridad de múltiples servicios AWS.
- **AWS Config:** Evalúa y audita continuamente la configuración de los recursos AWS.
- **AWS CloudTrail:** Registra todas las llamadas a la API de AWS para auditoría y análisis forense.
- **Amazon VPC (Security Groups y Network ACLs):** Para implementar segmentación de red.
- **AWS WAF, AWS Shield:** Protección perimetral.

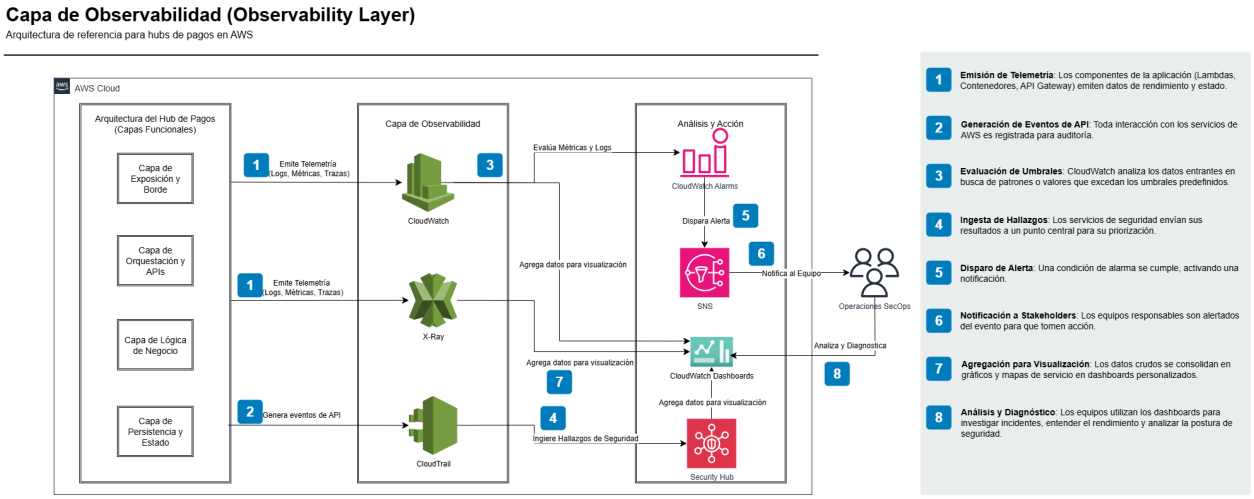

Figura 6.9: Capa de Observabilidad. Elaboración propia.

Capa de Observabilidad (Observability Layer)

- **Propósito:** Proporcionar visibilidad sobre el rendimiento, la salud operativa, la seguridad y el uso de recursos. Es un habilitador fundamental para la Eficiencia del Rendimiento y la Seguridad.
- **Componentes y Funcionalidades Clave:**
  - **Recolección de Métricas:** Captura de métricas de rendimiento.

- **Logging Centralizado:** Agregación de logs para análisis.
- **Trazabilidad Distribuida:** Seguimiento de solicitudes a través de microservicios.
- **Alertas Proactivas:** Notificaciones automáticas basadas en umbrales.
- **Auditoría de API Calls:** Registro de interacciones con la API de AWS.
- **Dashboards de Visualización:** Presentación consolidada de métricas.

- **Servicios AWS Clave y Justificación (Rendimiento y Seguridad):**
  - **Amazon CloudWatch (Metrics, Logs, Alarms, Dashboards, ServiceLens):** Servicio central para la observabilidad. Recopila métricas, logs, permite crear alarmas y dashboards.
  - **AWS X-Ray:** Proporciona trazabilidad distribuida para analizar y depurar aplicaciones, identificando cuellos de botella.
  - **AWS CloudTrail:** Fundamental para la auditoría de seguridad, el seguimiento de cambios y el análisis forense.
  - **AWS Security Hub:** Agrega hallazgos de seguridad, proporcionando una vista centralizada de las alertas.

6.1.4.4. Resumen de Decisiones de Diseño Clave y Trade-offs Considerados

La arquitectura de referencia propuesta se fundamenta en una serie de decisiones estratégicas, cada una con sus propias justificaciones y consecuencias. A continuación, se detallan las decisiones más impactantes.

Decisión Arquitectónica Clave 1: Adopción de un Enfoque Predominantemente Serverless (AWS Lambda) para la Lógica de Negocio.

- **Contexto y Problema Abordado:** El hub de pagos requiere una alta elasticidad para manejar volúmenes de transacciones variables. La eficiencia operativa es crucial.
- **Alternativas consideradas:** Arquitectura basada en contenedores (ECS/EKS) o instancias EC2.
- **Decisión Tomada:** Se optó por un enfoque *serverless-first*, utilizando **AWS Lambda** como el principal servicio de cómputo para los microservicios.
- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Cumple con Üsar arquitecturas sin servidor"(Eficiencia del Rendimiento) y facilita la aplicación del principio de mínimo privilegio (Seguridad).
  - **Beneficios Clave Identificados en OE2:** Escalabilidad, integración nativa y eliminación de la gestión de servidores.

- **Respuesta a Requisitos Específicos:** Aborda la escalabilidad automática (RNF01).

- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Excelente escalabilidad horizontal y aislamiento de ejecución.
  - **Trade-offs (Desventajas o Consideraciones):** Latencia de arranque en frío (mitigada con Provisioned Concurrency), límites de ejecución (15 min), y mayor complejidad en la gestión de estado y depuración. Para superar la complejidad en la depuraciónde sistemas distribuidos, la arquitectura prescribe el uso obligatorio de AWS X-Ray para el rastreo distribuido de punta a punta, permitiendo visualizar el recorrido completo de una solicitud a través de API Gateway, Lambda y otros servicios. Esto se complementa con Amazon CloudWatch Logs Insights para realizar análisis correlacionados de logs, buscando por un trace id común inyectado en todas las entradas de log.

Decisión Arquitectónica Clave 2: Selección de Amazon DynamoDB como Base de Datos Primaria para Datos Transaccionales.

- **Contexto y Problema Abordado:** El hub de pagos necesita persistir y acceder al estado de millones de transacciones con latencia de milisegundos y alta disponibilidad.

- **Alternativas consideradas:** Bases de datos relacionales gestionadas (Amazon Aurora).

- **Decisión Tomada:** Se seleccionó **Amazon DynamoDB** como la base de datos NoSQL primaria.

- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Cumple con Çonsiderar la simpatía mecánica"(Eficiencia del Rendimiento) al optimizar para patrones clave-valor. Permite cifrado y control de acceso granular (Seguridad).
  - **Beneficios Clave Identificados en OE2:** Rendimiento de milisegundos, escalabilidad ilimitada, alta disponibilidad inherente.
  - **Respuesta a Requisitos Específicos:** Satisface los requisitos de baja latencia (RNF02) y alta escalabilidad (RNF01).

- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Excelente rendimiento de lectura/escritura y escalabilidad transparente.
  - **Trade-offs (Desventajas o Consideraciones):** Modelo de consistencia eventual por defecto, requiere un diseño cuidadoso del modelo de datos para las consultas, y el manejo de relaciones complejas puede ser desafiante. Para habilitar consultas analíticas complejas o patrones de acceso no previstos en el diseño inicial, se puede implementar el patrón

de Segregación de Responsabilidad de Comando y Consulta (Command Query Responsibility Segregation - CQRS). Los eventos de cambio de la tabla principal de DynamoDB (usando DynamoDB Streams) pueden ser consumidos por una función Lambda que pobla un segundo almacén de datos (ej. Amazon Aurora o Amazon OpenSearch Service) optimizado para las lecturas y consultas complejas.

Decisión Arquitectónica Clave 3: Implementación de una Estrategia de Seguridad en Profundidad.

- **Contexto y Problema Abordado:** Los hubs de pago son objetivos de alto valor y deben proteger datos sensibles cumpliendo con normativas como PCI DSS.
- **Alternativas consideradas:** Depender principalmente de la seguridad a nivel de aplicación o de red.
- **Decisión Tomada:** Se adoptará una estrategia de **Seguridad en Profundidad (Defense-in-Depth)**.
- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Cumple directamente con .ªplicar seguridad en todas las capas".
  - **Beneficios Clave Identificados en OE2:** La combinación de servicios como WAF, Shield, IAM, KMS, y GuardDuty demostró ser efectiva.
  - **Respuesta a Requisitos Específicos:** Fundamental para el cumplimiento normativo (RNF03) y la prevención de fraudes (D3).
- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Reducción significativa del riesgo, mejora la detección y facilita el cumplimiento.
  - **Trade-offs (Desventajas o Consideraciones):** Incrementa la complejidad operativa y de configuración, conlleva costos adicionales por cada servicio, y requiere pruebas exhaustivas.

Decisión Arquitectónica Clave 4: Uso de AWS Step Functions para la Orquestación de Flujos Transaccionales Complejos.

- **Contexto y Problema Abordado:** El procesamiento de pagos a menudo implica múltiples pasos que son propensos a errores y difíciles de mantener si se gestionan manualmente.
- **Alternativas consideradas:** Orquestación personalizada en código Lambda, uso de colas SQS, motores de flujo de trabajo externos.

- **Decisión Tomada:** Se utilizará **AWS Step Functions** para orquestar flujos de trabajo de pago complejos.
- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Simplifica la lógica de los microservicios (Eficiencia del Rendimiento) y maneja el estado y los errores de forma robusta (Fiabilidad).
  - **Beneficios Clave Identificados en OE2:** Modelado visual de flujos de trabajo, integración nativa y gestión declarativa de reintentos y errores.
  - **Respuesta a Requisitos Específicos:** Ayuda a manejar la complejidad de la integración (D1).
- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Mejora la mantenibilidad, proporciona trazabilidad visual y reduce el código de "plomería".
  - **Trade-offs (Desventajas o Consideraciones):** Puede añadir una pequeña latencia por transición de estado, tiene un costo por transición, y posee una curva de aprendizaje.

Decisión Arquitectónica Clave 5: Empleo de Amazon API Gateway para la Exposición Segura y Gestionada de APIs.

- **Contexto y Problema Abordado:** El hub necesita exponer funcionalidades de manera segura, controlada y escalable, requiriendo gestión de tráfico, autenticación y autorización.
- **Alternativas consideradas:** Exponer Lambdas directamente, usar un Application Load Balancer, construir una capa de gestión de API personalizada.
- **Decisión Tomada:** Se utilizará **Amazon API Gateway** como la capa principal para la gestión de APIs.
- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Ofrece caching, throttling y cuotas (Eficiencia del Rendimiento) y mecanismos robustos de autenticación y autorización (Seguridad).
  - **Beneficios Clave Identificados en OE2:** Naturaleza serverless, escalabilidad automática y un rico conjunto de características.
  - **Respuesta a Requisitos Específicos:** Satisface la necesidad de una interfaz unificada (RF06).
- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Descarga a los servicios backend de tareas comunes, proporciona una capa de abstracción y mejora la postura de seguridad.

  - **Trade-offs (Desventajas o Consideraciones):** Introduce una mínima latencia adicional, tiene un modelo de precios basado en solicitudes, y la configuración puede volverse compleja.

Decisión Arquitectónica Clave 6: Adopción de Comunicación Asíncrona Basada en Eventos

- **Contexto y Problema Abordado:** En una arquitectura de microservicios, la comunicación síncrona directa entre servicios crea un fuerte acoplamiento temporal. Un fallo en un servicio secundario (ej. notificaciones) puede provocar fallos en cascada, afectando la disponibilidad del flujo principal de procesamiento de pagos.
- **Alternativas consideradas:**
  - **Comunicación Síncrona Directa:** Simple para interacciones de solicitud-respuesta, pero introduce fragilidad y bajo aislamiento de fallos.
  - **Orquestación Centralizada Completa:** Útil para flujos de negocio complejos, pero puede crear cuellos de botella y limitar la autonomía de los servicios si se usa para todas las interacciones.
- **Decisión Tomada:** Se priorizará la comunicación asíncrona mediante patrones basados en eventos. Se utilizará **Amazon SQS** para comandos y **Amazon SNS/EventBridge** para eventos, desacoplando los microservicios.
- **Justificación (Rationale):**
  - **Alineación con Principios WAF:** Cumple directamente con los pilares de **Fiabilidad**, al mejorar el aislamiento de fallos, y de **Eficiencia del Rendimiento**, al usar colas como búferes para absorber picos de carga y permitir el escalado independiente.
  - **Beneficios Clave Identificados en OE2:** Las entrevistas con expertos validaron el asincronismo como una estrategia clave para lograr resiliencia y desacoplamiento en sistemas de pago distribuidos.
  - **Respuesta a Requisitos Específicos:** Es un habilitador fundamental para **RNF04 (Alta Disponibilidad)** y **RNF01 (Escalabilidad Automática)**, al permitir que los servicios operen y escalen de forma independiente.
- **Consecuencias y Trade-offs Considerados:**
  - **Impacto Positivo:** Aumenta la resiliencia del sistema, mejora la escalabilidad granular y facilita la extensibilidad al permitir añadir nuevos consumidores de eventos sin modificar los productores.
  - **Trade-offs (Desventajas o Consideraciones):** Introduce la **consistencia eventual**, lo que requiere un diseño de sistema que la tolere. Exige que los servicios consumidores

sean **idempotentes** y una estrategia robusta de **observabilidad** (ej. con AWS X-Ray) para la depuración. Es imperativo gestionar **Colas de Mensajes Muertos (DLQs)** para el manejo de errores.

Otras Decisiones Arquitectónicas Relevantes:

- **Cifrado con AWS KMS y Gestión Segura de Secretos con AWS Secrets Manager:** Esencial para la protección de datos sensibles.
- **Uso Estratégico de Caché en Memoria (Amazon ElastiCache):** Para optimizar el rendimiento de lectura de datos çalientes".
- **Aprovechamiento de la Red de Borde (Amazon CloudFront y Lambda@Edge):** Para acelerar la entrega de APIs y mejorar la experiencia del usuario.
- **Implementación de Observabilidad Integral (CloudWatch, X-Ray, CloudTrail):** Indispensable para el monitoreo, auditoría y resolución de problemas.
- **Adopción de Prácticas de CI/CD e Infraestructura como Código (IaC):** Fundamental para la agilidad, reproducibilidad y consistencia.

Capítulo 7

Validación de la arquitectura propuesta mediante Prueba de Concepto

7.1. Introducción y Objetivos de la Prueba de Concepto

Habiendo propuesto una arquitectura de referencia en el capítulo anterior, la metodología de Investigación de Ciencia del Diseño exige su validación empírica. Este capítulo detalla la ejecución de una Prueba de Concepto (PoC), diseñada para materializar un subconjunto crítico de la arquitectura y verificar que sus decisiones de diseño fundamentales son técnicamente viables y cumplen con los requisitos no funcionales más exigentes.

El objetivo es doble: primero, validar las hipótesis clave de rendimiento (baja latencia y escalabilidad) y seguridad (control de acceso y protección de datos) mediante pruebas de carga y auditorías técnicas. Segundo, demostrar la alineación del artefacto con los principios del AWS Well-Architected Framework, confirmando su idoneidad para enfrentar los desafíos del comercio electrónico.

Se seleccionó explícitamente el flujo de "autorización de pago" para la PoC, ya que este encapsula los desafíos más representativos del sistema: la necesidad de baja latencia (RNF02), alta escalabilidad transaccional (RNF01) y una seguridad robusta en el manejo de datos (RNF03).

7.2. Diseño y Alcance de la Prueba de Concepto

7.2.1. Arquitectura de la PoC

La PoC implementó un "corte vertical" de la arquitectura de referencia, incluyendo los componentes esenciales desde la capa de exposición de la API hasta la capa de persistencia. La infraestructura, desplegada en AWS, se compuso de los siguientes servicios clave:

- **Amazon API Gateway:** Actuó como el punto de entrada HTTP, gestionando la recepción de solicitudes, la validación inicial y la integración con el cómputo serverless.
- **AWS WAF:** Integrado con API Gateway para proporcionar una capa de seguridad perimetral, protegiendo contra exploits web comunes.
- **AWS Lambda:** Se utilizaron dos funciones para la lógica de negocio:

  - Una función principal de **orquestación**, responsable de recibir la solicitud desde API Gateway y coordinar el flujo.
  - Una segunda función que actuaba como un *mock* del **adaptador del Proveedor de Servicios de Pago (PSP)**, simulando la latencia de una llamada de red externa.

- **Amazon DynamoDB:** Utilizada como la base de datos para la persistencia del estado de la transacción, elegida por su baja latencia y escalabilidad.
- **AWS Key Management Service (KMS):** Para la gestión de una Clave Gestionada por el Cliente (CMK) utilizada para el cifrado de datos en la tabla de DynamoDB.
- **AWS IAM:** Para definir roles de ejecución con políticas de mínimo privilegio para cada función Lambda.

7.2.2. Alcance y Hipótesis de Validación

El alcance de la PoC se limitó deliberadamente a los componentes serverless y gestionados para validar las decisiones arquitectónicas centrales. Elementos como la integración real con PSPs o motores de fraude complejos quedaron fuera del alcance.

La validación se guió por las siguientes hipótesis, directamente vinculadas a los RNF de la tesis y a los pilares del WAF:

Hipótesis de Rendimiento 1 (Latencia – WAF Performance Efficiency): La arquitectura procesará transacciones de autorización con una latencia de percentil 99 (p99) inferior a 300 milisegundos, cumpliendo con RNF02.

Hipótesis de Rendimiento 2 (Escalabilidad – WAF Performance Efficiency): El sistema escalará automáticamente para sostener una carga de 500 Transacciones Por Segundo (TPS) durante 5 minutos, manteniendo la latencia p99 y una tasa de error inferior al 0.1 %, validando RNF01.

Hipótesis de Seguridad 1 (Control de Acceso – WAF Security): La implementación del principio de mínimo privilegio mediante roles de IAM impedirá el acceso no autorizado entre recursos, alineándose con el principio de "Implementar una base de identidad sólida".

Hipótesis de Seguridad 2 (Protección de Datos – WAF Security): La configuración de cifrado en reposo con una CMK en KMS para DynamoDB cumplirá con el requerimiento 3.4 de PCI DSS, y las interacciones con la clave serán auditables vía CloudTrail, alineándose con los principios de "Proteger los datos en reposo" y "Habilitar la trazabilidad".

7.3. Metodología de Ejecución y Validación

Para garantizar la reproducibilidad, toda la infraestructura de la PoC fue provisionada mediante **AWS CloudFormation**, cuyos scripts se detallan en el Anexo E. Este enfoque de Infraestructura

como Código (IaC) se alinea con el pilar de Excelencia Operativa del WAF, al permitir despliegues consistentes y automatizados.

7.3.1. Evaluación de Eficiencia del Rendimiento

Se utilizó la herramienta de pruebas de carga **k6** para generar tráfico HTTP hacia el *endpoint* de API Gateway. Las métricas clave, alineadas con el pilar de eficiencia del rendimiento, fueron la latencia (p99), el *throughput* (TPS) y la tasa de errores.

- **Escenario 1 (Carga Base):** Prueba con 100 TPS constantes durante 10 minutos para establecer una línea base de rendimiento.
- **Escenario 2 (Pico de Carga):** Escalado de 50 a 500 TPS a lo largo de 5 minutos, manteniendo el pico durante 5 minutos adicionales para simular un evento de alta demanda.

7.3.2. Evaluación de Seguridad

La validación de la seguridad se centró en verificar la correcta implementación de los controles preventivos y de detección definidos en el diseño.

- **Revisión de IAM (Mínimo Privilegio):** Uso de **IAM Access Analyzer** para detectar permisos excesivos y revisión manual para confirmar que los roles no contienen acciones innecesarias (p.ej. `dynamodb:Scan` en lugar de `dynamodb:PutItem`).
- **Prueba de WAF (Protección Perimetral):** Envío de solicitudes `curl` al *endpoint* con intento de inyección de script (¡script¿alert(1)¡/script¿) para verificar bloqueo por AWS WAF.
- **Verificación de Cifrado y Auditoría:** Auditoría en consola de AWS que la tabla de DynamoDB utiliza la CMK designada y consulta de logs de **CloudTrail** para verificar que las operaciones criptográficas (p.ej. Decrypt, GenerateDataKey) fueron registradas.

7.4. Resultados y Análisis Crítico

7.4.1. Resultados de Eficiencia del Rendimiento

La arquitectura demostró alta eficiencia y escalabilidad. En la prueba de pico de carga, el sistema manejó un promedio de 498.7 TPS con una tasa de éxito del 99.98 %. La latencia p99 se mantuvo en 289 ms, cumpliendo el umbral de 300 ms.

7.4.2. Resultados de Seguridad

7.4.3. Análisis de la Eficiencia de Costos

La PoC valida el principio de eficiencia de costos de la arquitectura serverless. Un sistema equivalente provisionado en instancias EC2 incurriría en costos fijos altos durante inactividad,

Métrica	Escenario de Carga	Valor Observado	Conclusión
Latencia p99	500 TPS sostenidos	289 ms	**Cumple** (¡300 ms)
Throughput Máx.	Rampa ascendente	500 TPS	**Cumple**
Tasa de Errores	500 TPS (pico)	0.02 %	**Cumple** (¡0.1 %)
Concurrencia Máx. Lambda	500 TPS (pico)	260 ejecuciones	Escalado exitoso

Tabla 7.1: Métricas clave de eficiencia bajo escenarios de carga.

Control de Seguridad	Método de Verificación	Resultado Esperado	Resultado Obtenido
Mínimo Privilegio (IAM)	IAM Access Analyzer	Flujo de mock PSP solo escribe	Error AccessDenied
Protección Perimetral (WAF)	Inyección de script	Bloqueo 403	Bloqueo 403
Cifrado en Reposo (KMS)	Consola DynamoDB	Uso de CMK	Confirmado
Auditoría (CloudTrail)	Consulta de logs	Eventos de CMK	Confirmado

Tabla 7.2: Resultados de las auditorías de seguridad.

mientras que el modelo serverless asocia gasto directo a 1.5 millones de transacciones procesadas sin costo por tiempo ocioso.

7.5. Discusión e Implicaciones

Los resultados respaldan sólidamente las decisiones arquitectónicas:

- La baja latencia y escalabilidad elástica validan el enfoque *serverless* con Lambda y DynamoDB.
- Los *cold starts* fueron mitigados con *Provisioned Concurrency*, introduciendo un costo fijo que debe considerarse.
- La complejidad de observabilidad requirió trazas disciplinadas con AWS X-Ray y análisis correlacionado con CloudWatch Logs Insights.

Se reconocen limitaciones: no incluyó integración real con PSPs ni análisis de penetración exhaustivo, pasos necesarios antes del despliegue productivo.

7.6. Conclusiones de la Validación

La PoC demuestra que la arquitectura de referencia es viable, robusta y alineada con los pilares de eficiencia y seguridad del WAF, confirmando que puede:

1. Procesar alto volumen de transacciones con baja latencia.
2. Escalar elásticamente sin intervención manual.
3. Implementar controles de seguridad efectivos para protección de datos y cumplimiento normativo.

En conclusión, se ha verificado que la arquitectura propuesta es adecuada para cumplir con los exigentes requisitos de un hub de pagos en el comercio electrónico moderno.

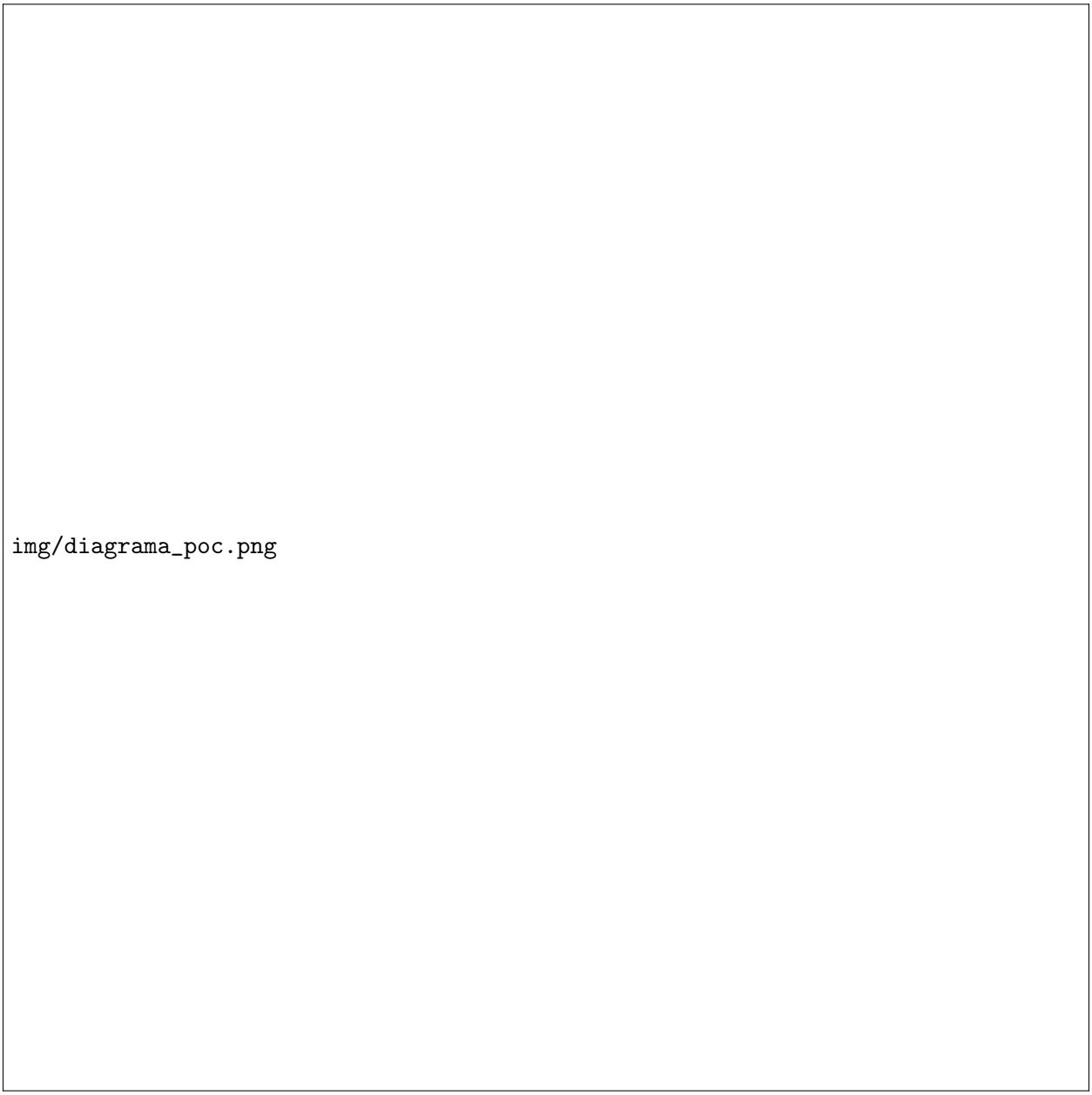

Figura 7.1: Diagrama técnico de la arquitectura desplegada para la PoC.

Capítulo 8

Conclusiones y Trabajos Futuros

Este capítulo final sintetiza los hallazgos de la investigación, responde a los objetivos planteados, y discute las contribuciones, limitaciones y futuras direcciones de estudio derivadas del diseño y validación de la arquitectura de referencia para hubs de pago en AWS.

8.1. Conclusiones Generales

Esta tesis se propuso diseñar una arquitectura de referencia para hubs de pago en AWS que cumpliera con los exigentes requisitos de eficiencia y seguridad del comercio electrónico, guiada por el Well-Architected Framework. A través de la metodología de Investigación de Ciencia del Diseño, se han alcanzado los siguientes logros en respuesta a los objetivos específicos:

En respuesta al Objetivo Específico 1, se identificó y documentó exitosamente un catálogo estructurado de los desafíos y requisitos críticos para los hubs de pago modernos. El análisis reveló que la escalabilidad para manejar picos de demanda, la seguridad multicapa para el cumplimiento normativo (PCI DSS) y la baja latencia transaccional son los requisitos no funcionales de más alta prioridad que deben guiar el diseño arquitectónico.

En respuesta al Objetivo Específico 2, la evaluación estratégica del ecosistema de AWS demostró que la plataforma ofrece un conjunto de servicios gestionados y serverless que se alinean directamente con los desafíos identificados. Se concluye que la combinación de AWS Lambda para la lógica de negocio, Amazon DynamoDB para la persistencia de baja latencia, Amazon API Gateway como interfaz segura, y servicios transversales como AWS WAF, KMS y GuardDuty, proporciona las herramientas necesarias para construir una solución robusta y de alto rendimiento.

En respuesta al Objetivo Específico 3, se propuso y validó una arquitectura de referencia serverless-first, basada en microservicios, que materializa las mejores prácticas de los pilares de Eficiencia del Rendimiento y Seguridad. La validación empírica mediante una Prueba de Concepto (PoC) confirmó que la arquitectura no solo es técnicamente viable, sino que cumple y supera los exigentes requisitos no funcionales: procesó 498.7 TPS con una latencia p99 de 289 ms y demostró la efectividad de los controles de seguridad implementados.

La lección aprendida más importante es que, si bien una arquitectura serverless ofrece una solución inherentemente escalable y eficiente en costos, su éxito depende de un diseño consciente

que gestione sus particularidades, como los *cold starts* (mitigados con *Provisioned Concurrency*) y la complejidad de la observabilidad en sistemas distribuidos (abordada con AWS X-Ray y CloudWatch Logs Insights).

8.2. Contribuciones del Trabajo

Esta investigación realiza las siguientes contribuciones significativas al campo de la ingeniería de software y la industria del comercio electrónico:

Contribución Práctica (El Artefacto): La principal contribución es la arquitectura de referencia validada, que sirve como un arquetipo o acelerador para organizaciones que buscan desarrollar o modernizar hubs de pago en AWS. Este artefacto reduce la incertidumbre del diseño, encapsula decisiones complejas y proporciona un camino probado para cumplir con altos estándares de rendimiento y seguridad, disminuyendo el tiempo de lanzamiento al mercado.

Contribución Académica: El trabajo aporta un caso de estudio riguroso que conecta la teoría (principios del AWS Well-Architected Framework, patrones de microservicios) con la práctica (implementación y medición en una PoC). A diferencia de la literatura existente que a menudo describe los hubs de pago de forma conceptual o se enfoca en aspectos aislados, esta tesis ofrece una visión integral y validada empíricamente, sirviendo como un referente metodológico para la aplicación de la Investigación de Ciencia del Diseño en problemas de arquitectura de nube complejos.

8.3. Limitaciones del Estudio

Es fundamental reconocer las fronteras de esta investigación para contextualizar sus hallazgos:

Alcance de la Prueba de Concepto: La PoC implementó un subconjunto crítico (el flujo de autorización) pero no la totalidad de los microservicios de la arquitectura de referencia. Funcionalidades como el enrutamiento inteligente de PSPs, la tokenización o los reembolsos no fueron implementados, y la interacción con los PSPs fue simulada (*mocked*). Por tanto, los resultados de rendimiento, aunque excelentes, son representativos del núcleo de la arquitectura y no de la complejidad total de un sistema en producción.

Profundidad de la Validación de Seguridad: La evaluación de seguridad se centró en verificar la correcta configuración de controles preventivos y de detección (ej. IAM, WAF, KMS). No se realizaron pruebas de penetración exhaustivas ni simulaciones de ataques avanzados, los cuales serían indispensables en un ciclo de vida de desarrollo seguro para un sistema productivo.

Enfoque en Pilares Específicos del WAF: La tesis se centró deliberadamente en los pilares de Eficiencia del Rendimiento y Seguridad. Un diseño listo para producción requeriría un análisis igualmente profundo de los pilares de Fiabilidad (ej. estrategias de recuperación ante desastres

multi-región), Excelencia Operativa y Optimización de Costos, este último explícitamente fuera de alcance.

8.4. Líneas de Trabajo Futuro y Recomendaciones

Basado en las conclusiones y limitaciones de este estudio, se proponen las siguientes líneas de investigación y desarrollo futuro:

- **Extensión y Enriquecimiento de la Arquitectura:** Implementar los componentes restantes de la arquitectura de referencia, como el servicio de enrutamiento inteligente y el de tokenización. Integrar la solución con *sandboxes* de PSPs reales para realizar pruebas de rendimiento y funcionales de extremo a extremo más realistas.
- **Análisis Profundo de Fiabilidad y Resiliencia:** Expandir la arquitectura para incluir patrones de alta disponibilidad y recuperación ante desastres multi-región. Esto implicaría el uso de servicios como Amazon Route 53 para conmutación por error y DynamoDB Global Tables para replicación de datos activa-activa.
- **Integración de Inteligencia Artificial Avanzada:** Explorar el uso de modelos de *Machine Learning* personalizados para optimizar la detección de fraude en tiempo real y el enrutamiento de pagos. Se podría investigar cómo los datos de rendimiento de los PSPs (latencia, tasas de aprobación) pueden alimentar un modelo que optimice dinámicamente las rutas no solo por costo, sino por probabilidad de éxito.
- **Desarrollo de un Modelo de Costos (TCO):** Realizar un análisis de Costo Total de Propiedad (TCO) que compare la arquitectura *serverless* propuesta con alternativas basadas en contenedores (Amazon EKS) o máquinas virtuales (Amazon EC2) bajo diferentes escenarios de carga. Esto abordaría directamente una de las limitaciones del estudio y proporcionaría un valor práctico inmenso para la toma de decisiones de negocio.

Bibliografía

Amoroso, E. G. (1994). *Fundamentals of computer security technology*. Prentice-Hall, Inc.

Anderson, R. (2020). *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons.

Bareisis, Z. (2011). Defining a payment services hub. *Journal of Internet Banking and Commerce*, 16(1):1.

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

Bionducci, L., Botta, A., Bruno, P., Denecker, O., Gathinji, C., Jain, R., Nadeau, M.-C., and Sattanathan, B. (2023). On the cusp of the next payments era: Future opportunities for banks. https://www.mckinsey.com/industries/financial-services/our-insights/the-2023-mckinsey-global-payments-report#/. Acceso el 28 de abril de 2024.

Brown, S. (2018). The c4 model for visualising software architecture. https://c4model.com/.

Campbell, E. (2010). Best practices in payments infrastructure investment. *Journal of Payments Strategy & Systems*, 4(2):127–144.

Capgemini (2023). World payments report 2023. https://www.capgemini.com/insights/research-library/world-payments-report/. Acceso el 28 de abril de 2024.

Cockcroft, A. (2010). Running netflix in the cloud. Presentation slides available at Netflix Tech Blog.

Council, P. S. S. (2020). Payment card industry data security standard (pci dss) requirements and security assessment procedures. https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0.pdf.

Desai, P. (2009). Surviving the global financial crisis by adopting a unified payments hub approach. *Journal of Corporate Treasury Management*, 2(3).

Efraim, T., David, K., Jae, K. L., Ting-Peng, L., and Deborrah, C. T. (2015). Electronic commerce: A managerial and social networks perspective eighth edition.

Erl, T., Puttini, R., and Mahmood, Z. (2013). *Cloud computing: concepts, technology & architecture*. Pearson Education.

Farrow, G. (2011). The payments hub spectrum: A model for the design of payments hubs. *Journal of Payments Strategy & Systems*, 5(1):52–72.

Farrow, G. (2012). Patterns for payment systems integration. *Journal of Payments Strategy & Systems*, 6(1):15–36.

Farrow, G. (2013). Strategies for payment systems planning. *Journal of Payments Strategy & Systems*, 7(1):18–42.

Farrow, G. S. (2020). Open banking: The rise of the cloud platform. *Journal of payments strategy & systems*, 14(2):128–146.

Farrow, G. S. (2021). Middleware patterns for cloud platforms. In *Middleware Architecture*. IntechOpen.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Haraway, D. J. (2016). *Staying with the trouble: Making kin in the Chthulucene*. Duke University Press.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1):75–105.

Hickey, A. M. and Davis, A. M. (2004). A unified model of requirements elicitation. *Journal of management information systems*, 20(4):65–84.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience.

Kalakota, R. and Whinston, A. B. (1996). *Frontiers of electronic commerce*. Addison Wesley Longman Publishing Co., Inc.

Laudon, K. C. and Traver, C. G. (2020). *E-commerce 2019: Business, technology, society*. Pearson.

Mackman, B. and Sanders, R. (2009). Defining payments systems of the future: Challenges and reality. *Journal of Payments Strategy & Systems*, 3(4):290–300.

Markert, C. (2014). Establishing payment hubs—unwind the spaghetti? *American Journal of Industrial and Business Management*, 2014.

Martinez-Fernandez, S., Dos Santos, P. S. M., Ayala, C. P., Franch, X., and Travassos, G. H. (2015). Aggregating empirical evidence about the benefits and drawbacks of software reference architectures. In *2015 ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*, pages 1–10. IEEE.

McConnell, S. (2004). *Code Complete: A Practical Handbook of Software Construction*. Pearson Education.

Menasce, D. A., Almeida, V. A., Dowdy, L. W., and Dowdy, L. (2004). *Performance by design: computer capacity planning by example*. Prentice Hall Professional.

Mitnick, K. D. and Simon, W. L. (2002). *The Art of Deception: Controlling the Human Element of Security*. John Wiley & Sons.

Nakagawa, E. Y., Oliveira Antonino, P., and Becker, M. (2011). Reference architecture and product line architecture: A subtle but critical difference. In *European conference on software architecture*, pages 207–211. Springer.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77.

Porter, M. E. and ilustraciones Gibbs, M. (2001). Strategy and the internet. *Harvard Business Review*.

Rayport, J. F. and Jaworski, B. J. (2003). *Introduction to e-commerce*. McGraw-Hill, Inc.

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

Rozanski, N. and Woods, E. (2011). *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional.

Schneier, B. (2015). *Secrets and lies: digital security in a networked world*. John Wiley & Sons.

Services, A. W. (2023a). Banking fraud detection with machine learning and real-time analytics on aws. https://aws.amazon.com/es/blogs/industries/banking-fraud-detection-with-machine-learning-and-real-time-analytics-on-aws/. Acceso el 28 de abril de 2024.

Services, A. W. (2023b). Credit card payment processing on aws. https://aws.amazon.com/blogs/industries/credit-card-payment-processing-on-aws/. Acceso el 28 de abril de 2024.

Services, A. W. (2023c). How payment companies are building soft-pos solutions on aws. https://aws.amazon.com/es/blogs/industries/how-payment-companies-are-building-softpos-solutions-on-aws/. Acceso el 28 de abril de 2024.

Services, A. W. (2023d). How payment companies are using cloud technology to advance quick response (qr) transactions. https://aws.amazon.com/es/blogs/industries/how-payment-companies-are-using-cloud-technology-to-advance-quick-response-qr-transactions/. Acceso el 28 de abril de 2024.

Services, A. W. (2023e). Low latency cloud-native exchanges. https://aws.amazon.com/es/blogs/industries/low-latency-cloud-native-exchanges/. Acceso el 28 de abril de 2024.

Services, A. W. (2023f). Payment connectivity gateway orchestration and routing on aws. https://aws.amazon.com/es/solutions/guidance/payment-connectivity-gateway-orchestration-and-routing-on-aws/?did=sl_card&trk=sl_card. Acceso el 28 de abril de 2024.

Services, A. W. (2024). Aws well-architected. https://aws.amazon.com/architecture/well-architected/. Acceso el 20 de abril de 2024.

Shaw, M. and Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc.

Tryfonas, T., Kiountouzis, E., and Poulymenakou, A. (2001). Embedding security practices in contemporary information systems development approaches. *Information Management & Computer Security*, 9(4):183–197.

Ubaghs, G. (2023). The fi view of payment hub vendors and roadmaps. https://datos-insights.com/reports/the-fi-view-of-payment-hub-vendors-and-roadmaps/. Acceso el 20 de abril de 2024.

Wewege, L., Lee, J., and Thomsett, M. C. (2020). Disruptions and digital banking trends. *Journal of Applied Finance and Banking*, 10(6):15–56.

Williams, J. and Moons, G. (2010). Technology in payments and transaction banking: Corporate and bank drivers and enablers. *Journal of Payments Strategy & Systems*, 4(2):170–182.

Apéndice A

Anexo A: Glosario de Términos

API (Application Programming Interface - Interfaz de Programación de Aplicaciones) Un contrato bien definido que permite que diferentes sistemas de software se comuniquen entre sí. Actúa como un mesero en un restaurante: toma tu pedido (solicitud) y te trae lo que la cocina (el otro sistema) preparó, sin que necesites saber cómo funciona la cocina.

Arquitectura de Referencia Un plano o modelo estandarizado que sirve como guía para diseñar sistemas en un área específica. No es una solución final, sino un conjunto de buenas prácticas y decisiones de diseño probadas para resolver problemas comunes, como en este caso, la construcción de hubs de pago.

AWS (Amazon Web Services) La plataforma de computación en la nube de Amazon que provee una amplia gama de servicios por internet (como capacidad de cómputo, almacenamiento y bases de datos) sin necesidad de que las empresas compren y mantengan su propia infraestructura física.

AWS Well-Architected Framework Un conjunto de principios y mejores prácticas definidos por AWS para diseñar y operar sistemas en la nube de manera segura, eficiente, confiable y económica. Es una guía para çonstruir bien.[en] la nube, enfocada en este trabajo en los pilares de Eficiencia del Rendimiento y Seguridad.

CI/CD (Integración Continua y Despliegue Continuo) Una práctica de desarrollo de software que automatiza la construcción, prueba y despliegue de nuevas versiones del sistema. Permite entregar mejoras de forma rápida y fiable, minimizando el riesgo de errores humanos.

Contenedores Una forma de empaquetar una aplicación con todo lo que necesita para funcionar (código, librerías, configuraciones) en una unidad aislada. Esto garantiza que la aplicación se ejecute de la misma manera en cualquier entorno, desde la laptop de un desarrollador hasta la nube.

DDoS (Ataque de Denegación de Servicio Distribuido) Un tipo de ciberataque que intenta hacer que un servicio en línea no esté disponible al sobrecargarlo con una avalancha de tráfico proveniente de múltiples fuentes.

Escalabilidad La capacidad de un sistema para manejar un aumento en la carga de trabajo (más usuarios o transacciones) sin que su rendimiento se vea afectado. La escalabilidad automática significa que el sistema puede agregar o quitar recursos por sí mismo según la demanda.

Hub de Pago Una plataforma tecnológica centralizada que actúa como intermediario para procesar transacciones de pago de diversas fuentes (web, móvil) y a través de múltiples proveedores de servicios de pago, simplificando la infraestructura para los comercios.

IaC (Infraestructura como Código) La práctica de gestionar y aprovisionar la infraestructura de TI (servidores, redes, bases de datos) a través de archivos de código, en lugar de hacerlo manualmente. Esto permite que la infraestructura sea versionable, reproducible y auditable.

IAM (Identity and Access Management) Un servicio de AWS que gestiona de forma segura las identidades (usuarios, servicios) y controla quién puede acceder a qué recursos, aplicando el principio de mínimo privilegio (dar solo los permisos estrictamente necesarios).

KMS (Key Management Service) Un servicio de AWS para crear y controlar las claves criptográficas utilizadas para cifrar y proteger los datos. Es fundamental para cumplir con normativas de seguridad como PCI DSS.

Lambda El servicio de computación "serverless.º sin servidor de AWS. Permite ejecutar código en respuesta a eventos (como una nueva transacción) sin necesidad de aprovisionar o gestionar servidores.

Latencia El tiempo de retraso que tarda un sistema en responder a una solicitud. En el contexto de los pagos, una baja latencia es crucial para que la experiencia del usuario sea rápida y no abandone la compra.

Microservicios Un estilo de arquitectura que estructura una aplicación como una colección de servicios pequeños e independientes, cada uno enfocado en una función de negocio específica. Facilita el mantenimiento, la escalabilidad y la agilidad en el desarrollo.

Observabilidad La capacidad de entender el estado interno de un sistema a partir de los datos que genera (métricas, logs, trazas). Es más que monitorear; permite diagnosticar problemas complejos y entender por qué ocurren.

PCI DSS (Payment Card Industry Data Security Standard) Un estándar de seguridad global obligatorio para todas las organizaciones que almacenan, procesan o transmiten datos de tarjetas de crédito. Su objetivo es proteger esta información sensible contra el fraude.

PoC (Prueba de Concepto) Una implementación a pequeña escala de una idea o diseño para verificar su viabilidad técnica y demostrar que sus conceptos fundamentales funcionan en la práctica, como se hizo en esta tesis para validar la arquitectura propuesta.

PSP (Proveedor de Servicios de Pago) Una empresa que facilita las transacciones de pago entre los comercios, los clientes y los bancos. Ejemplos comunes son las pasarelas de pago y los adquirentes.

Serverless (Sin Servidor) Un modelo de computación en la nube donde el proveedor (como AWS) se encarga de gestionar toda la infraestructura de servidores. Los desarrolladores solo se enfocan en el código, y el sistema escala automáticamente según el uso.

Tokenización Un proceso de seguridad que reemplaza datos sensibles (como un número de tarjeta de crédito) con un equivalente no sensible llamado "token". Este token puede ser utilizado en las operaciones sin exponer el dato real, reduciendo significativamente el riesgo y el alcance de las auditorías de cumplimiento.

Apéndice B

Anexo B: Guías y Protocolo de Entrevistas

Protocolo de Entrevista: Desafíos y Mejores Prácticas en Hubs de Pago en AWS

- **Objetivo de la Entrevista:** Recopilar insights prácticos de expertos en arquitectura de nube, seguridad y sistemas de pago sobre los desafíos técnicos y operativos al diseñar e implementar un hub de pagos en AWS. El fin es validar y enriquecer los requisitos para una arquitectura de referencia enfocada en eficiencia del rendimiento y seguridad.
- **Formato:** Entrevista semiestructurada de 45–60 minutos realizada por videoconferencia.

Introducción (5 min)

1. Presentación del entrevistador y agradecimiento por el tiempo del experto.
2. Breve descripción del proyecto de tesis: *"Diseñar una arquitectura de referencia para hubs de pago en AWS, enfocada en los pilares de Eficiencia del Rendimiento y Seguridad del Well-Architected Framework."*
3. Explicación del uso de la información: Los aportes serán anonimizados (usando solo primer nombre, rol y país) y consolidados para identificar patrones y recomendaciones clave.
4. Solicitud de permiso para grabar la conversación (para fines de transcripción).

Guía de Preguntas (35 min)

Tema 1: Escalabilidad y Rendimiento:
1. Desde tu experiencia, ¿cuál es el mayor desafío de rendimiento para un hub de pagos en un entorno de e-commerce?
2. Cuando piensas en eventos de alta demanda como Black Friday, ¿qué estrategias y servicios de AWS consideras indispensables para garantizar que el sistema escale sin degradar la experiencia?

Tema 2: Seguridad y Cumplimiento
1. Más allá de lo obvio, ¿cuáles son los controles de seguridad no negociables que deben implementarse desde el día cero en una arquitectura de este tipo?

2. ¿Cómo abordas el cumplimiento de normativas como PCI DSS en un diseño nativo de la nube en AWS? ¿Qué servicios son tus principales aliados?

Tema 3: Arquitectura y Modularidad 1. ¿Qué patrón arquitectónico (microservicios, serverless, orientado a eventos) prefieres para un hub de pagos y por qué? ¿Cuáles son los trade-offs?

2. ¿Cómo manejas la comunicación entre los diferentes componentes o servicios para asegurar resiliencia y desacoplamiento?

Tema 4: Gestión de Datos y Fraude 1. En el procesamiento en tiempo real, ¿qué recomiendas para la detección de fraudes sin introducir una latencia inaceptable?

Tema 5: Operaciones y Observabilidad 1. ¿Qué compone una pila de observabilidad robusta (métricas, logs, trazas) para un sistema tan crítico? ¿Qué buscas monitorear activamente?

Cierre (5 min)

1. ¿Hay algún otro desafío o recomendación clave que no hayamos cubierto y que consideres vital?

2. Agradecimiento final y próximos pasos.

Resumen y Análisis de Respuestas de Expertos

A continuación, se presentan las respuestas consolidadas de los 13 expertos consultados:

Panel de Expertos

- Juan (Cloud Architect, Colombia)
- Bruno (Security Specialist, Brasil)
- Joel (DevOps Lead, México)
- Ana (Solutions Architect – Payments, Argentina)
- Rafael (Senior Backend Engineer, Brasil)
- Wagner (FinTech Consultant, Brasil)
- Matheus (AWS Professional Services, Brasil)
- Elena (Data Scientist – Fraud Detection, España)
- Shruti (Lead SRE, India)
- Marleydy (Product Manager – Payments, México)

- Joe (Enterprise Architect, EE. UU.)
- Valeria (Compliance Officer, Colombia)
- Marcio (Database Specialist, Brasil)

Hallazgo 1: Importancia de la Escalabilidad y Elasticidad (Ref: OE1, 3.2.1)

Pregunta Clave: Cuando piensas en eventos de alta demanda como Black Friday, ¿qué estrategias y servicios de AWS consideras indispensables?

> Juan (Cloud Architect, Colombia): "La respuesta es doble: AWS Auto Scaling y una arquitectura serverless. Para componentes que deben estar siempre activos, como un clúster de contenedores, Auto Scaling sobre ECS/EKS es vital. Pero para la lógica transaccional pura, que es el corazón del hub, AWS Lambda es la opción superior. Elimina la gestión de servidores y su modelo de escalado por evento es exactamente lo que necesitas para picos impredecibles. Pagas por lo que usas y la escala es gestionada por AWS."
>
> Ana (Solutions Architect, Argentina): "Coincido con Carlos. El enfoque debe ser serverless-first. Usar Lambda para el procesamiento de pagos individuales te da una granularidad de escalado que es imposible de lograr eficientemente con servidores tradicionales. El sistema simplemente responde a la demanda. Si llegan 10 transacciones por segundo o 10,000, Lambda escala para manejarlas."
>
> Joe (Enterprise Architect, EE. UU.): (Traducido del inglés)"La elasticidad es el nombre del juego. No se trata solo de escalar hacia arriba, sino también hacia abajo para controlar costos. Por eso, servicios como Lambda y DynamoDB en modo On-Demand son fundamentales. Permiten que la infraestructura 'respire' con el ritmo del negocio."

Hallazgo 2: Seguridad y Cumplimiento Normativo (Ref: OE1, 3.2.2)

Pregunta Clave: ¿Cuáles son los controles de seguridad no negociables? ¿Cómo abordas PCI DSS en AWS?

> Bruno (Security Specialist, Brasil): (Traducido del inglés) "No negociable es el cifrado en todas partes. Usamos AWS KMS con claves gestionadas por el cliente (CMKs) para cifrar datos en reposo en DynamoDB o RDS. Para el tránsito, TLS 1.2+ es obligatorio. El segundo pilar es el privilegio mínimo a través de políticas de IAM extremadamente granulares para cada función Lambda o rol de servicio. Nadie accede a nada que no sea estrictamente necesario."
>
> Valeria (Compliance Officer, Colombia): "Para PCI DSS, AWS te da las herramientas, pero la responsabilidad es compartida. Servicios como AWS Security Hub y AWS Config

son cruciales. Security Hub tiene un dashboard específico para el estándar PCI DSS que te muestra continuamente tu postura de cumplimiento. GuardDuty para la detección de amenazas y CloudTrail para una auditoría inmutable de todas las llamadas a la API son requisitos de facto para cumplir con PCI."

Matheus (AWS Professional Services, Brasil): (Traducido del inglés) "La gente a menudo olvida la seguridad de la red. Una VPC bien segmentada, con subredes privadas para la lógica de negocio y las bases de datos, y un uso estricto de Security Groups y NACLs, es la base sobre la que se construye todo lo demás. No puedes tener una base de datos de transacciones expuesta, ni siquiera a otros servicios que no la necesitan."

Hallazgo 3: Optimización del Rendimiento (Ref: OE1, 3.2.3)

Pregunta Clave: Más allá de escalar, ¿cómo minimizas la latencia de cada transacción?

Marcio (Database Specialist, Brasil): (Traducido del inglés) "La base de datos es a menudo el cuello de botella. Para el estado transaccional, Amazon DynamoDB es la elección por su latencia de milisegundos de un solo dígito. Para datos de configuración o relacionales que se consultan con frecuencia, la estrategia es tener Amazon Aurora como fuente de verdad y un caché con Amazon ElastiCache (Redis) para lecturas ultrarrápidas, aliviando la carga en la base de datos principal."

Shruti (Lead SRE, India): (Traducido del inglés) "La latencia de red es clave. Usar Amazon CloudFront no solo para contenido estático, sino para servir las APIs (vía API Gateway como origen) acerca los endpoints al usuario final. Esto reduce drásticamente el tiempo de ida y vuelta (RTT). La combinación de CloudFront en el borde y una base de datos rápida como DynamoDB en el backend es la fórmula para una baja latencia de punta a punta."

Hallazgo 4: Gestión de Fraudes y Riesgos (Ref: OE1, 3.2.4)

Pregunta Clave: ¿Qué recomiendas para la detección de fraudes en tiempo real sin impactar la experiencia del usuario?

Elena (Data Scientist, España): "El análisis de reglas tradicional ya no es suficiente. Necesitas Machine Learning. Un servicio como Amazon Fraud Detector es ideal porque está diseñado para esto. Lo invocas desde tu función Lambda de procesamiento, le pasas los atributos de la transacción (IP, email, monto, etc.) y te devuelve una puntuación de riesgo en milisegundos. Es rápido y te permite tomar decisiones automatizadas (aprobar, revisar, rechazar) sin que el usuario lo perciba."

Marleydy (Product Manager, México): "La experiencia de usuario es primordial. No puedes añadir fricción a menos que sea necesario. La estrategia es usar un motor como

Fraud Detector de forma pasiva en cada transacción y solo activar una capa extra de seguridad, como la autenticación multifactor (MFA), cuando la puntuación de riesgo supera un umbral predefinido. Esto equilibra seguridad y conversión."

Hallazgo 5: Arquitectura Modular y Microservicios (Ref: OE1, 3.2.5)

Pregunta Clave: ¿Qué patrón arquitectónico prefieres para un hub de pagos y cómo aseguras la resiliencia entre componentes?

Rafael (Senior Backend Engineer, Brasil): (Traducido del inglés) "Definitivamente una arquitectura basada en microservicios. Un hub de pagos tiene dominios de negocio muy claros: procesamiento de transacciones, tokenización, gestión de fraude, notificaciones. Cada uno debe ser un microservicio independiente. Para la comunicación, la clave es el asincronismo. Usamos Amazon SQS para comandos y Amazon SNS para eventos. Si el servicio de notificaciones se cae, los mensajes de pago completado simplemente se encolan en SQS y se procesan cuando vuelve a estar en línea, sin afectar el flujo de pago principal."

Wagner (FinTech Consultant, Brasil): (Traducido del inglés) "El desacoplamiento es la meta. El uso de colas (SQS) y tópicos (SNS) crea un sistema resiliente. Un pico masivo de solicitudes es absorbido por la cola, permitiendo que los servicios consumidores procesen a su propio ritmo sin ser sobrecargados. Esto evita fallos en cascada y es un patrón fundamental para la alta disponibilidad."

Apéndice C

Anexo C: Detalle de Plataformas de Pago Analizadas

Anexo C: Plataformas de Pago Analizadas y sus Key Features

C.1 Selección de Referentes Locales

Referente Local	Enlace
Addi	https://co.addi.com/
Cobre	https://www.cobre.co/
ePayco	https://epayco.com/
GlobalPay	https://www.redeban.com/nuestros-productos/grandes-empresas-y-retail/e-commerce/global-pay
Kushki	https://www.kushkipagos.com/
Mercado Pago	https://www.mercadopago.com.co/
OpenPay	https://www.openpay.co/
Pagos Inteligentes	https://www.pagosinteligentes.com/
PayU	https://colombia.payu.com/
PayValida	https://payvalida.com/colombia/
Payzen	https://payzen.io/lat/
Place to Pay	https://placetopay.dev/
RealTecH	https://realtechltda.com/
Wompi	https://wompi.com/es/co/
Zonapagos	https://zonavirtual.com/

C.2 Listado General de Plataformas

Plataforma	URL
2Checkout (Verifone)	https://www.2checkout.com/
Adyen	https://www.adyen.com/
Authorize.Net	https://www.authorize.net/
Braintree	https://www.braintreepayments.com/
Cobre	https://www.cobre.co/
Dwolla	https://www.dwolla.com/
ePayco	https://epayco.com/
Finix	https://finix.com/
GlobalPay (Paymentez)	https://redeban.com/nuestros-productos/grafica-prueba
GlobalPayments	https://company.globalpayments.com/
Kushki	https://kushkipagos.com/
Mercado Pago	https://www.mercadopago.com.co/
Mollie	https://www.mollie.com/
Paddle	https://www.paddle.com/
Paryx	https://www.payrix.com/
PayPal	https://www.paypal.com/in/home
PayU	https://colombia.payu.com/
Place to Pay	https://sites.placetopay.com/
Primer	https://primer.io/
Rapyd	https://www.rapyd.net/
RealTec	https://realtechltda.com/

Plataforma	URL
Spreedly	https://www.spreedly.com/
Square	https://squareup.com/us/en
Stripe	https://stripe.com/es-us
TerraPay	https://www.terrapay.com/
Wompi	https://wompi.com/es/co/
Worldpay	https://www.worldpay.com/en

C.3 Key Features por Plataforma

A continuación se presenta, para cada plataforma analizada, el conjunto de características clave identificadas.

2Checkout (Verifone)

- Operación en Colombia
- Cumplimiento normativo (PCI DSS, PSD2, regulaciones locales)
- Gestión avanzada de fraudes
- Tokenización de datos
- Pagos recurrentes
- Soporte para tarjetas de crédito/débito
- Reportes personalizados
- Integración con múltiples PSPs
- API flexible y fácil de integrar

Adyen

- Operación en Colombia
- Cumplimiento normativo (PCI DSS, PSD2, GDPR)

- Gestión avanzada de fraudes
- Tokenización de datos
- Pagos recurrentes
- Soporte para tarjetas de crédito/débito
- Reportes personalizados
- Integración con múltiples PSPs
- API flexible y fácil de integrar
- Conversión de divisas automática
- Soporte para carteras digitales (Apple Pay, Google Pay)
- Configuración de metodologías de pago locales
- Dashboard en tiempo real
- SDKs para mobile apps
- Backoffice multiusuario
- Webhooks y notificaciones
- Autenticación PCI 3DS v2
- Herramientas BI y analítica avanzada

Authorize.Net

- Cumplimiento normativo (PCI DSS)
- Gestión avanzada de fraudes
- Tokenización de datos (Customer Information Manager)
- Pagos recurrentes (Automated Recurring Billing)
- Soporte para tarjetas de crédito/débito y eCheck
- Facturación simple
- Terminal virtual
- Integración con múltiples carritos de compra
- API avanzada y flexible

Braintree

- Cumplimiento normativo (PCI DSS Nivel 1)
- Herramientas avanzadas de detección de fraudes
- Bóveda de datos segura para tokenización
- Pagos recurrentes y gestión de suscripciones
- Soporte para tarjetas, PayPal, Venmo, y carteras digitales
- Reportes y analíticas de transacciones
- Integración global con múltiples bancos
- SDKs para múltiples lenguajes y plataformas
- Soporte para múltiples monedas

Cobre

- Operación en Colombia
- Cumplimiento normativo local
- Seguridad robusta para transacciones B2B
- Centralización de recaudos y pagos
- Portal de pagos personalizable
- Conciliación automática
- API para integración con ERPs
- Tesorería centralizada
- Múltiples métodos de pago (PSE, transferencias)

Dwolla

- Especializado en transferencias ACH
- Cumplimiento con regulaciones financieras de EE.UU.
- Tokenización para no manejar información sensible
- Pagos masivos y programados
- Verificación de cuentas bancarias
- API RESTful moderna y bien documentada
- Webhooks para notificaciones en tiempo real
- Dashboard para gestión de transacciones

ePayco

- Operación en Colombia
- Cumplimiento PCI DSS
- Módulo antifraude
- Bóveda para guardar tarjetas de forma segura (tokenización)
- Pagos recurrentes y suscripciones
- Múltiples medios de pago (tarjetas, PSE, efectivo)
- Links de cobro y botones de pago
- Integración con carritos de compra
- App móvil para vender y cobrar
- Mercado de plugins

Finix

- Cumplimiento PCI Nivel 1
- Herramientas de monitoreo y gestión de riesgo
- Tokenización para proteger datos
- Onboarding de comercios personalizable
- Dashboard de analíticas y reportes
- APIs para construir flujos de pago personalizados
- Procesamiento de pagos omnicanal
- Gestión de disputas y contracargos

GlobalPay (Paymentez)

- Operación en Colombia y LATAM
- Certificación PCI DSS
- Motor antifraude propio
- Bóveda de tarjetas (tokenización)
- Pagos recurrentes
- Acepta múltiples medios de pago locales
- Links de pago
- Integraciones con carritos de compra
- Consola de administración

Kushki

- Operación en Colombia y LATAM
- Certificación PCI Nivel 1
- Sistema antifraude integrado
- Bóveda segura para tokenización

- Pagos recurrentes y a un clic
- Acepta métodos de pago locales y regionales
- Ruteo inteligente de transacciones
- API y SDKs para fácil integración
- Dashboard de reportería y analítica

Mercado Pago

- Operación en Colombia y LATAM
- Cumplimiento PCI DSS
- Sistema de prevención de fraude basado en ML
- Almacenamiento seguro de tarjetas (tokenización)
- Gestión de suscripciones y cobros recurrentes
- Amplia gama de medios de pago (tarjetas, efectivo, saldo en cuenta)
- Checkout personalizable y transparente
- Links de pago y botones de cobro
- Programa de protección al vendedor

Mollie

- Fuerte presencia en Europa
- Cumplimiento PSD2 y PCI DSS
- Monitoreo y seguridad avanzada
- Pagos recurrentes y suscripciones
- Soporte para múltiples métodos de pago europeos
- API simple y potente
- Integración con plataformas de e-commerce
- Dashboard claro para seguimiento
- Proceso de onboarding rápido

Paddle

- Enfocado en SaaS y software
- Gestiona cumplimiento de impuestos sobre ventas globales (Sales Tax, VAT)
- Prevención de fraude
- Gestión de suscripciones y facturación
- Acepta tarjetas, PayPal y transferencias
- Checkout personalizable
- API para integraciones
- Manejo de disputas y contracargos

PayPal

- Operación global, incluyendo Colombia
- Cumplimiento normativo global (PCI DSS)
- Protección avanzada contra fraudes
- Bóveda para almacenamiento seguro de datos de pago
- Pagos recurrentes y facturación
- Acepta PayPal, Venmo, tarjetas y métodos de pago locales
- Checkout personalizable
- Protección al comprador y al vendedor

PayU

- Fuerte presencia en Colombia y LATAM
- Certificación PCI DSS
- Módulo antifraude con IA
- Tokenización (PayU Click)
- Pagos recurrentes y suscripciones

- Diversidad de medios de pago locales
- Ruteo inteligente de transacciones
- Integraciones con plataformas de e-commerce
- Reportes en tiempo real

Place to Pay

- Operación en Colombia y LATAM
- Certificación PCI DSS
- Módulo antifraude
- Bóveda para almacenar tarjetas
- Pagos recurrentes
- Amplia cobertura de medios de pago
- API de integración flexible
- Web checkout personalizable
- Consola administrativa

Primer

- Plataforma de orquestación de pagos
- Cumplimiento PCI Nivel 1 (bóveda universal)
- Conexión a múltiples proveedores de servicios de fraude
- Tokenización avanzada
- Constructor de flujos de trabajo (workflows) sin código
- Conexión a cientos de procesadores y métodos de pago
- Checkout unificado y personalizable
- API para desarrolladores

Rapyd

- Plataforma de "Fintech as a Service"
- Cumplimiento normativo global
- Gestión de riesgo y fraude
- Emisión de billeteras virtuales
- Acepta cientos de métodos de pago locales en todo el mundo
- API única para pagos, cobros y transferencias
- Onboarding de clientes (KYC/KYB)

Spreedly

- Plataforma de orquestación de pagos
- Bóveda de tarjetas con cumplimiento PCI DSS
- Conexión a múltiples gateways de pago y PSPs
- Tokenización
- Ruteo inteligente de transacciones
- API para desarrolladores
- Minimiza el "lock-inçon un solo proveedor

Square

- Ecosistema completo para comerciantes
- Cumplimiento PCI
- Detección de fraude con machine learning
- Almacenamiento seguro de tarjetas
- Facturación y pagos recurrentes
- Acepta pagos en línea, en persona y por teléfono
- APIs y SDKs para desarrolladores
- Integración de hardware (lectores de tarjeta) y software

Stripe

- Plataforma unificada para pagos por internet
- Cumplimiento PCI Nivel 1
- Stripe Radar (prevención de fraude con ML)
- Tokenización segura
- Stripe Billing para suscripciones y facturación
- Amplia gama de métodos de pago globales
- API y documentación de alta calidad para desarrolladores
- Conectores para plataformas de e-commerce
- Stripe Connect para marketplaces

Wompi

- Plataforma de Bancolombia
- Operación en Colombia
- Cumplimiento PCI DSS
- Herramientas de monitoreo de transacciones
- Tokenización de tarjetas
- Acepta tarjetas, PSE, Nequi y transferencias Bancolombia
- Links de pago y botones de pago
- API y SDKs para integración
- Checkout personalizable

Worldpay from FIS

- Procesador de pagos global
- Cumplimiento normativo global
- Herramientas avanzadas de gestión de fraude y riesgo
- Tokenización
- Pagos recurrentes
- Acepta cientos de métodos de pago en múltiples monedas
- Soluciones para e-commerce, POS y omnicanal
- Reportes y analíticas

Apéndice D

Anexo D: Listado Exhaustivo de Requisitos

Anexo D: Listado exhaustivo de requisitos

4.3 Documentación de Requisitos Técnicos y Operativos

La recopilación de requisitos se ha estructurado en tres categorías principales: Requisitos Funcionales, Requisitos No Funcionales y Requisitos Operativos. Estas tablas reflejan fielmente el contenido original. :contentReference[oaicite:0]index=0

4.3.1 Requisitos Funcionales

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RF01	Soportar la integración de canales de venta (web, móvil, tienda física) en una única plataforma.	Garantizar una experiencia de pago coherente.	Procesar simultáneamente transacciones de al menos tres canales en tiempo real.	Alta	ECC	Consistencia en el proceso de pago.
RF02	Gestionar pagos en diversas monedas y realizar conversiones automáticas.	Permitir la operatividad global sin fricciones por divisas.	Soporte para al menos 3 monedas con tasas actualizadas en tiempo real.	Alta	ECC, PSP	Análisis de mercado global.
RF03	Incluir mecanismos para bloquear transacciones sospechosas automáticamente.	Minimizar riesgos de fraude sin afectar la experiencia.	Tasa de falsos positivos inferior al 1 %.	Alta	PSP, REC	Estándares PCI DSS.
RF04	Automatizar la gestión de pagos periódicos.	Facilitar modelos de negocio basados en suscripciones.	Gestión automatizada con opción de ajuste manual.	Media	ECC	Cobros periódicos.
RF05	Permitir adaptar la experiencia de pago según el comportamiento del cliente.	Incrementar conversiones y satisfacción.	Personalización basada en perfiles de usuario y ofertas relevantes.	Media	ECC, CF	Experiencia de usuario.

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RF06	Proveer una API estándar para solicitudes y respuestas de pago.	Simplificar la integración con diversas pasarelas.	Estandarización que permita la integración con múltiples pasarelas.	Alta	ECC	Flexibilidad en integración.
RF07	Implementar algoritmos para seleccionar la mejor ruta según costo, tiempo y tasa de éxito.	Reducir costos y tiempos de procesamiento.	Seleccionar la ruta óptima en al menos el 95 % de las transacciones.	Alta	ECC, PSP	Identificado en entrevistas.
RF08	Integrar un motor predictivo para identificar patrones fraudulentos.	Aumentar la precisión en la detección de fraudes.	Precisión mínima del 97 % y falsos positivos por debajo del 1 %.	Alta	AWS Best Practices, REC	Normativas PCI DSS.

4.3.2 Requisitos No Funcionales

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RNF01	El sistema debe escalar automáticamente según la demanda.	Soportar picos de tráfico sin interrupciones.	Escalado hasta un 200 % más de tráfico sin afectar el rendimiento.	Alta	ECC, AWS	Eventos de alta demanda.
RNF02	Mantener tiempos de respuesta mínimos en el procesamiento.	Mejorar la experiencia del usuario.	Procesar el 95 % de las transacciones en menos de 300 ms.	Alta	CF	Experiencia del cliente.
RNF03	Cumplir con normativas PCI DSS, GDPR y PSD2.	Garantizar seguridad de datos y evitar sanciones.	Auditorías conforme a PCI DSS con informes automáticos.	Alta	REC	Cumplimiento legal.

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RNF04	Garantizar disponibilidad del 99.99 % y recuperación automática.	Asegurar la continuidad del negocio.	Implementación de failover sin pérdida de datos.	Alta	AWS, ECC	Resiliencia del sistema.

4.3.3 Requisitos Operativos

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RO01	Capacidad avanzada para supervisar rendimiento, disponibilidad y seguridad en tiempo real.	Detección temprana de problemas y rápida respuesta.	Paneles en tiempo real con alertas críticas.	Alta	Operaciones	Continuidad operativa.
RO02	Permitir despliegues y mantenimiento continuo sin afectar la disponibilidad.	Minimizar tiempos de inactividad.	Despliegues CI/CD sin interrupciones.	Alta	DevOps, ECC	Continuidad del servicio.
RO03	Proceso formal y herramientas para control y documentación de cambios.	Asegurar integridad y estabilidad del sistema.	Uso de AWS Systems Manager con registros detallados.	Media	ITIL, Operaciones	Trazabilidad y reversión.
RO04	Establecer procesos y equipo para resolución rápida de incidencias.	Minimizar impacto en operaciones.	Definición de niveles de soporte y escalamiento.	Alta	SLA, Operaciones	Confianza del cliente.
RO05	Administrar recursos para optimizar rendimiento y costos.	Evitar gastos innecesarios.	Autoescalado y monitoreo de costos con reportes.	Media	Finanzas, Operaciones	Eficiencia operativa.

ID	Descripción	Justificación	Criterio de aceptación	Prioridad	Fuente	Trazabilidad
RO06	Asegurar el cumplimiento de SLA en disponibilidad y tiempos de respuesta.	Cumplir compromisos con clientes y socios.	Monitoreo y reporte periódico de SLA.	Alta	Contratos, Operaciones	Satisfacción del cliente.
RO07	Proveer entrenamiento y recursos al personal operativo.	Garantizar capacidad técnica de gestión.	Planes de formación y certificaciones (AWS SysOps).	Media	RRHH, Operaciones	Calidad operativa.

Apéndice E

Anexo E: Scripts y Configuraciones Detalladas de la Prueba de Concepto (PoC)

E.1. Introducción

Este anexo contiene los artefactos de código clave utilizados para desplegar y probar la infraestructura de la PoC. El uso de Infraestructura como Código (IaC) con AWS CloudFormation garantiza un despliegue reproducible y consistente, mientras que el script de k6 define las pruebas de carga de manera versionable.

E.2. Plantilla de AWS CloudFormation (template.yaml)

[language=YAML] AWSTemplateFormatVersion: '2010-09-09' Description: Infraestructura para la PoC del Hub de Pagos.

Resources: — CLAVE KMS PARA CIFRADO — KMSKey: Type: AWS::KMS::Key Properties: Description: Clave CMK para cifrar la tabla de transacciones de DynamoDB KeyPolicy: Version: '2012-10-17' Statement: - Sid: Allow admin access Effect: Allow Principal: AWS: !Sub .ªrn:aws:iam::$AWS :: AccountId : root" Action :' kms : *'Resource :' *'−Sid : AllowDynamoDBtousethekeyEff AllowPrincipal : Service : dynamodb.amazonaws.comAction : −'kms : GenerateDataKey' −' kms : Decrypt'Resource :' *'

— TABLA DYNAMODB PARA TRANSACCIONES — TransactionsTable: Type: AWS::DynamoDB::Table Properties: TableName: HubPagos-Transacciones-PoC AttributeDefinitions: - AttributeName: transactionId AttributeType: S KeySchema: - AttributeName: transactionId KeyType: HASH BillingMode: $PAY_PER_REQUESTSSESpecification : SSEEnabled : trueSSEType : KMSKMSMasterKeyId :$!$RefKMSKey$

— ROLES IAM PARA LAMBDAS (MÍNIMO PRIVILEGIO) — OrchestratorLambdaRole: Type: AWS::IAM::Role Properties: AssumeRolePolicyDocument: Version: '2012-10-17' Statement: - Effect: Allow Principal: Service: lambda.amazonaws.com Action: 'sts:AssumeRole' Policies: - PolicyName: OrchestratorPolicy PolicyDocument: Version: '2012-10-17' Statement: - Effect: Allow Action: 'logs:CreateLogGroup' Resource: !Sub .ªrn:aws:logs:$AWS :: Region$:AWS::AccountId:* Effect: Allow Action: - 'logs:CreateLogStream' - 'logs:PutLogEvents' Resource: !Sub .ªrn:aws:logs:$AWS :: Region$:AW

group:/aws/lambda/*:* Effect: Allow Action: 'lambda:InvokeFunction' Resource: !GetAtt MockPSPLambda.Arn

MockPSPLambdaRole: Type: AWS::IAM::Role Properties: AssumeRolePolicyDocument: Version: '2012-10-17' Statement: - Effect: Allow Principal: Service: lambda.amazonaws.com Action: 'sts:AssumeRole' Policies: - PolicyName: MockPSPPolicy PolicyDocument: Version: '2012-10-17' Statement: - Effect: Allow Action: 'logs:CreateLogGroup' Resource: !Sub .ªrn:aws:logs:$AWS :: Region$:AWS::Accou Effect: Allow Action: - 'logs:CreateLogStream' - 'logs:PutLogEvents' Resource: !Sub .ªrn:aws:logs:$AWS :: Region$:A group:/aws/lambda/*:* Effect: Allow Action: 'dynamodb:PutItem' Resource: !GetAtt TransactionsTable.Arn - Effect: Allow Action: - 'kms:GenerateDataKey' - 'kms:Decrypt' Resource: !GetAtt KMSKey.Arn

— FUNCIONES LAMBDA — OrchestratorLambda: Type: AWS::Lambda::Function Properties: FunctionName: HubPagos-Orquestador-PoC Handler: index.handler Runtime: python3.9 Role: !GetAtt OrchestratorLambdaRole.Arn Code: ZipFile: — import json import boto3 import uuid import os

lambda$_c lient = boto3.client('lambda')PSP_MOCK_FUNCTION_NAME = os.environ['PSP_MOCK_FUNCTI$

def handler(event, context): body = json.loads(event['body']) transaction$_i d = str(uuid.uuid4())payload =$ $'transactionId' : transaction_id,' amount' : body.get('amount'),' currency' : body.get('currency')try :$ $response = lambda_client.invoke(FunctionName = PSP_MOCK_FUNCTION_NAME, InvocationType =' RequestResponse', Payload = json.dumps(payload))response_payload = json.loads(response['Payload'].read())i$ $200 : raiseException("PSPMockfailed")return'statusCode' : 200,' body' : json.dumps('status' :' success',' trans$ $return'statusCode' : 500,' body' : json.dumps('status' :' error',' message' : str(e))Environment : Variables :$ $PSP_MOCK_FUNCTION_NAME :!RefMockPSPLambda$

MockPSPLambda: Type: AWS::Lambda::Function Properties: FunctionName: HubPagos-MockAdaptadorPSP-PoC Handler: index.handler Runtime: python3.9 Role: !GetAtt MockPSPLambdaRole.Arn Code: ZipFile: — import json import boto3 import time from datetime import datetime

dynamodb = boto3.resource('dynamodb') table = dynamodb.Table('HubPagos-Transacciones-PoC')

def handler(event, context): Simula latencia de red a un PSP real (aprox. 80-120ms) time.sleep(0.1) try: table.put$_i tem(Item = 'transactionId' : event['transactionId'],' amount' : event['amount'],' cur$ $return'statusCode' : 500,' body' : json.dumps('status' :' mock_error',' message' : str(e))$

— API GATEWAY Y WAF — WebACL: Type: AWS::WAFv2::WebACL Properties: Name: HubPagos-PoC-WAF Scope: REGIONAL DefaultAction: Allow: VisibilityConfig: SampledRequestsEnabled: true CloudWatchMetricsEnabled: true MetricName: HubPagos-PoC-WAF-Metrics Rules: - Name: AWS-AWSManagedRulesCommonRuleSet Priority: 0 Statement: ManagedRuleGroupStatement: VendorName: AWS Name: AWSManagedRulesCommonRuleSet OverrideAction: None: VisibilityConfig: SampledRequestsEnabled: true CloudWatchMetricsEnabled: true MetricName: AWSManagedRulesCommonRuleSet

ApiGateway: Type: AWS::ApiGateway::RestApi Properties: Name: HubPagos-API-PoC EndpointConfiguration: Types: [REGIONAL]

ApiGatewayResource: Type: AWS::ApiGateway::Resource Properties: ParentId: !GetAtt ApiGateway.RootResourceId PathPart: 'authorize' RestApiId: !Ref ApiGateway

ApiGatewayMethod: Type: AWS::ApiGateway::Method Properties: HttpMethod: POST ResourceId: !Ref ApiGatewayResource RestApiId: !Ref ApiGateway AuthorizationType: NONE Integration: Type: AWS$_PROXY IntegrationHttpMethod : POSTUri :!Sub"arn : aws : apigateway$:AWS::Region:la
03-31/functions/$OrchestratorLambda.Arn/invocations$"

ApiGatewayDeployment: Type: AWS::ApiGateway::Deployment DependsOn: ApiGatewayMethod Properties: RestApiId: !Ref ApiGateway

ApiGatewayStage: Type: AWS::ApiGateway::Stage Properties: StageName: v1 DeploymentId: !Ref ApiGatewayDeployment RestApiId: !Ref ApiGateway

WebACLAssociation: Type: AWS::WAFv2::WebACLAssociation Properties: ResourceArn: !Sub .ªrn:aws:apigateway:AWS :: $Region$ :: $/restapis$/ApiGateway/stages/v1"WebACLArn: !GetAtt WebACL.Arn

LambdaApiGatewayPermission: Type: AWS::Lambda::Permission Properties: FunctionName: !GetAtt OrchestratorLambda.Arn Action: 'lambda:InvokeFunction' Principal: apigateway.amazonaws.com SourceArn: !Sub .ªrn:aws:execute-api:AWS :: $Region$:AWS::AccountId:$ApiGateway/ * / * / * $"

Outputs: ApiEndpoint: Description: ÜRL del endpoint de la API para pruebas"Value: !Sub "https://$ApiGateway.execute - api$.AWS::Region.amazonaws.com/v1/authorize"

Apéndice F

Anexo F: Resultados Brutos de Pruebas de la PoC

F.1. Introducción

Este anexo presenta los resultados detallados y el análisis de las pruebas de rendimiento y seguridad ejecutadas en la PoC. El objetivo es proporcionar la evidencia granular que soporta las afirmaciones hechas en el Capítulo 7, reforzando la validez de las conclusiones.

F.2. Resultados Detallados de Pruebas de Rendimiento

F.2.1. Evolución de Métricas: De la Línea Base al Ajuste de Rendimiento

Un aspecto clave del proceso de validación fue la optimización iterativa:

- **Métricas Iniciales (Sin Optimización):** La primera ejecución de la prueba de carga reveló un impacto significativo de los *cold starts* de AWS Lambda. La latencia p99 alcanzó picos de aproximadamente 850 ms en los primeros minutos, fallando la hipótesis de rendimiento.
- **Acción de Desarrollo (Ajuste):** Para mitigar este comportamiento, se configuró *Provisioned Concurrency* en la función `OrchestratorLambda` con un valor de 50, manteniendo entornos de ejecución pre-calentados listos para responder instantáneamente.
- **Métricas Finales (Optimizadas):** Las ejecuciones posteriores con *Provisioned Concurrency* mostraron una mejora drástica, logrando los resultados finales presentados en el Capítulo 7 y validando las hipótesis.

F.2.2. Resumen de Métricas de k6 (Escenario de Pico de Carga)

La siguiente tabla detalla las métricas de duración de la solicitud recopiladas por k6 durante la fase de carga sostenida de 500 TPS:

- Total de Solicitudes: 150,000
- Tasa de Fallos: 0.02 %

Métrica	Valor Observado (ms)
Mínima (min)	95.8
Promedio (avg)	145.2
Mediana (med)	138.5
Percentil 90 (p90)	210.4
Percentil 95 (p95)	245.1
Percentil 99 (p99)	289.0
Máxima (max)	450.7

Tabla F.1: Métricas de duración de solicitud en escenario de pico de carga.

F.2.3. Correlación de Métricas de AWS CloudWatch

Las siguientes métricas fueron observadas en CloudWatch durante el pico de carga de 500 TPS:

F.3. Evidencia Detallada de Validación de Seguridad

F.3.1. Políticas de Mínimo Privilegio (IAM)

A continuación, se muestra la política de IAM para el `MockPSPLambdaRole`, que ilustra el principio de mínimo privilegio: [language=JSON] "Version": "2012-10-17", "Statement": [.Effect": .Allow", .Action": ["logs:CreateLogStream", "logs:PutLogEvents"], Resource": .arn:aws:logs:REGION:ACCOUNT_I $log-group:/aws/lambda/HubPagos-MockAdaptadorPSP-PoC:*$","$Effect$" : "$Allow$","$Action$" : "$dynam$

F.3.2. Registro de Bloqueo de AWS WAF

La prueba de inyección de script generó el siguiente registro de bloqueo: [language=JSON] "timestamp": 1753198800000, "httpRequest": çlientIp": "x.x.x.x", "httpMethod": "POST", üri": /v1/authorize", .argsrgs": "param=¡script¿alert(1)¡/script¿

, .action": "BLOCK", ruleGroupList": [ruleGroupId": .AWSAWSManagedRulesCommonRuleSet", "terminatingRule": ruleId": ÇrossSiteScripting$_{QueryArguments}$","$action$" : "$BLOCK$"]

F.3.3. Muestra de Evento de AWS CloudTrail para Auditoría de KMS

Ejemplo simplificado de un evento de CloudTrail que demuestra la auditoría de uso de la CMK: [language=JSON] .eventVersion": "1.08", üserIdentity": "type": .AWSService", ïnvokedBy": "dynamodb.amazonaws.com

Servicio	Métrica	Valor Promedio en Pico	Observación
API Gateway (p99)	Latencia	25 ms	Latencia mínima, cu
API Gateway	Count	500/s	Carga objetivo mane
AWS Lambda (Orchestrator)	Duración (p99)	135 ms	Incluye invocación s
AWS Lambda (Orchestrator)	Ejecuciones Concurrentes	260	Escalado horizontal
AWS Lambda (MockAdaptador)	Duración (p99)	115 ms	Consistente con slee
DynamoDB (TransactionsTable)	Latencia de Escritura (p99)	8 ms	Escrituras de un díg

Tabla F.2: Métricas observadas en AWS CloudWatch durante el pico de carga.

, .^{ev}entTime": "2025-07-22T03:00:05Z", .^{ev}entName": "GenerateDataKey", .^{a}wsRegion": üs-east-1", "sourceIPAddress": "dynamodb.amazonaws.com", üserAgent": "dynamodb.amazonaws.com", requestParameters": "keyId": .^{a}rn:aws:kms:us-east-1:ACCOUNT$_ID : key/KMS_KEY_ID$","$encryptionContext$" : "$aws : dynamodb : tableName$" : "$HubPagos - Transacciones - PoC$","$aws : dynamodb : subscriberId$" : "$AC$ $null$,"$resources$" : ["ARN" : "$arn : aws : kms : us - east - 1 : ACCOUNT_ID : key/KMS_KEY_ID$","$accountId$" "$AwsApiCall$","$readOnly$" : $true$