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

Diseño de una Arquitectura Resiliente de Referencia en la Nube de AWS. Aplicando el Well-Architected Framework para la Excelencia Operativa, Fiabilidad y Rendimiento

José Antonio Moreno Popayan

Director(a): Wilson Calvo Alvarez

03 de Agosto de 2025

Santiago de Cali, 03 de Agosto 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 Resiliente de Referencia en la Nube de AWS. Aplicando el Well-Architected Framework para la Excelencia Operativa, Fiabilidad y Rendimiento" realizado por el estudiante de Magister en Ingeniería de Software José Antonio Moreno Popayan (cod: 8988016), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,

Wilson Calvo Alvarez
Firmado digitalmente por Wilson Calvo Alvarez
Fecha: 2025.08.04 10:36:00 -05'00'

Wilson Calvo Alvarez

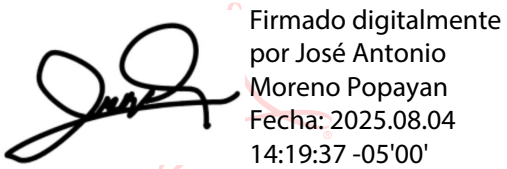
Santiago de Cali, 03 de Agosto de 2025.

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 Resiliente de Referencia en la Nube de AWS. Aplicando el Well-Architected Framework para la Excelencia Operativa, Fiabilidad y Rendimiento" 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,



José Antonio Moreno Popayan
Código: 8988016

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Diseño de una Arquitectura Resiliente en la Nube de AWS Aplicando el Well-Architected Framework para la Excelencia Operativa, Fiabilidad y Rendimiento

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Arquitectura de software en la nube
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Innovación
4. Estudiante: José Antonio Moreno Popayan
5. Correo electrónico: josemoreno99@javerianacali.edu.co
6. Dirección y teléfono: Kra 113 N° 28-71
7. Director: Wilson Calvo Alvarez
8. Vinculación del director: Externo
9. Correo electrónico del director: wilson.calvoa@pragma.com.co
10. Co-Director (Si aplica): N/A
11. Grupo o empresa que lo avala (Si aplica): N/A
12. Otros grupos o empresas: N/A
13. Palabras clave(al menos 5): Nube, Well-Architected Framework, Arquitectura, Excelencia operativa, fiabilidad, rendimiento
14. ODS que aplica al proyecto (Agenda 2030): N/A
15. Fecha de inicio: 01/09/2024

Agradecimientos

Quiero expresar el más sincero agradecimiento a todas las personas que, de una u otra forma, me acompañaron, apoyaron y motivaron durante el desarrollo de este proyecto de grado.

En primer lugar, tengo que expresar un sincero agradecimiento a mi familia, especialmente a mis padres, por su apoyo incondicional, confianza en mí y por ser mi sostén en los momentos de dificultad. Su apoyo emocional ha sido fundamental para alcanzar esta meta.

Un agradecimiento especial a mi director, Wilson Calvo Álvarez, MSc. por la dirección, apoyo, retroalimentación constructiva y su experiencia. De igual forma quiero agradecer a la directora de la Maestría en ingeniería de software, Luisa Rincón, Ph.D. por el apoyo brindado y retroalimentación constructiva brindada desde su experiencia profesional. También me gustaría agradecer al Comité de Evaluación por su tiempo, sus observaciones y por enriquecer con sus aportes la calidad final de este proyecto.

Gracias a mis compañeros estudiantes de maestría y colegas por sus ideas compartidas, asistencia mutua y debates técnicos, que han permitido ampliar mi perspectiva. A mis amigos más cercanos, gracias por las conversaciones, por escucharme en los momentos de frustración y por celebrar cada pequeño logro conmigo.

Extiendo mi gratitud a la Pontificia Universidad Javeriana Cali por brindarme las herramientas y el entorno académico ideal para la investigación académica donde se realizó este trabajo. Los recursos bibliográficos y tecnológicos que hicieron posible este proyecto.

Finalmente, este proyecto significa no solo los resultados del trabajo académico, sino también el cierre de un período en mi vida profesional y personal. Me llevo no solo conocimientos técnicos, sino también un crecimiento integral que ha sido posible gracias a todas las personas mencionadas y a las experiencias vividas en este proceso.

Gracias a todos.

Resumen

Este proyecto de grado tiene como propósito el diseño de una arquitectura resiliente de referencia en la nube de Amazon Web Services (AWS), aplicando los principios del Well-Architected Framework, con énfasis en los pilares de excelencia operativa, fiabilidad y rendimiento.

En la era en la que la mayoría de las organizaciones dependen de los servicios digitales, la resiliencia es una cualidad esencial para ayudar a una organización a sobrevivir a fallos, ataques o simplemente sobrecargas. Existen mejores prácticas y patrones de diseño de AWS, pero no hay arquitecturas de referencia que se centren puntualmente en la resiliencia, lo que deja a los equipos técnicos a la deriva sobre lo que se necesita para tener un entorno resiliente.

Con el fin de abordar esta problemática, se propuso el diseño de arquitectura basada en servicios nativos de AWS, integrando atributos de calidad como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación. La propuesta fue diseñada con enfoque modular y multirregional, utilizando componentes como Lambda, SQS, DynamoDB, CloudFront, WAF y Route 53, los cuales se organizaron en capas funcionales y fueron validados mediante una prueba de concepto.

Las pruebas contenían diferentes escenarios controlados como fallos regionales, errores en el procesamiento, verificación de idempotencia y conmutación por error automática, demostrando que la arquitectura propuesta es capaz de seguir funcionando, incluso sin interacción manual, dentro de los límites de RTO y RPO establecidos. Entre las lecciones aprendidas se destaca que la resiliencia debe ser diseñada como una 'preocupación transversal' desde el inicio del sistema; el desacoplamiento y la automatización son clave para responder a los fallos; y el propio WAF proporciona una base útil, pero requiere una mayor adaptación para lograr una cobertura completa de diseños resilientes. También se evidenció el valor de realizar pruebas orientadas a escenarios de fallo, incluso en entornos controlados, como mecanismo para validar el diseño.

En resumen, el trabajo presentado aquí proporciona una guía práctica y replicable para arquitectos de nube e ingenieros de infraestructura, y constituye un avance técnico y académico en la creación formalizada de soluciones resilientes en la nube pública.

Palabras Clave: Resiliencia, Arquitectura en la nube, Amazon Web Services (AWS), Well-Architected Framework, Excelencia operativa

Abstract

This thesis project aims to design a resilient reference architecture in the Amazon Web Services (AWS) cloud, applying the principles of the Well-Architected Framework with an emphasis on the pillars of operational excellence, reliability, and performance.

In an era where most organizations depend on digital services, resilience is an essential quality to help an organization withstand failures, attacks, or simply overloads. While AWS offers best practices and design patterns, there are no reference architectures focused purely on resilience, leaving technical teams without clear guidance on what is required to achieve a resilient environment.

To address this challenge, the project proposed an architecture based on native AWS services, integrating key attributes such as reliability, availability, security, robustness, and quick recovery. The proposed solution was designed with a modular and multi-region approach, using components such as Lambda, SQS, DynamoDB, CloudFront, WAF, and Route 53, organized into functional layers and validated through a proof of concept.

The tests included controlled scenarios such as regional failures, processing errors, idempotency verification, and automatic failover, demonstrating that the proposed architecture can continue operating—without manual intervention—within the established RTO and RPO limits.

Among the lessons learned, it was evident that resilience must be designed as a cross-cutting concern from the beginning of the system; decoupling and automation are key to responding to failures; and while the Well-Architected Framework provides a useful foundation, it requires further adaptation to achieve full coverage of resilient designs. The value of conducting failure scenario-based testing, even in controlled environments, was also highlighted as a mechanism to validate the design.

In summary, the work presented here offers a practical and replicable guide for cloud architects and infrastructure engineers, representing both a technical and academic advancement in the formalized creation of resilient solutions in the public cloud.

Keywords: Resilience, Cloud Architecture, Amazon Web Services (AWS), Well-Architected Framework, Operational Excellence

Índice general

Agradecimientos **7**

1. Introducción **1**
1.1. Definición del problema . . . 2
1.1.1. Planteamiento del problema . . . 2
1.1.2. Formulación del problema . . . 4
1.2. Objetivos del proyecto . . . 4
1.2.1. Objetivo General . . . 4
1.2.2. Objetivos específicos . . . 5
1.3. Delimitaciones y alcances . . . 5
1.4. Justificación del trabajo de grado . . . 6
1.5. Criterios de selección del entorno de nube de AWS y marco de diseño WAF . . . 7
1.6. Metodología de la investigación . . . 7
1.7. Resultados obtenidos . . . 9

2. Marco de referencia **11**
2.1. Marco Teórico . . . 11
2.1.1. Bases Teóricas . . . 11
2.2. Estado del Arte . . . 14
2.3. Resumen del capítulo . . . 20

3. Desarrollo del Proyecto **21**
3.1. Atributos de calidad clave para la resiliencia . . . 21
3.2. Diseño . . . 23
3.3. Resumen del capítulo . . . 35

4. Evaluación **37**
4.1. Diseño de la evaluación . . . 37
4.2. Resultados de la evaluación . . . 40
4.3. Consideraciones sobre costos . . . 57
4.4. Resumen del capítulo . . . 57

5. Conclusiones **59**
5.1. Conclusiones . . . 59
5.2. Trabajos futuros . . . 60
5.3. Lecciones aprendidas . . . 60

Bibliografía **63**

Índice de figuras

2.1. Principios para Mejorar la Resiliencia de las Aplicaciones en la Nube 13
2.2. Patrón - Multi-AZ 15
2.3. Arquitectura segura, altamente disponible en AWS 16
2.4. Patrón distribución del portafolio de aplicaciones 17
2.5. Patrón multi-Region active-active 18
2.6. Copia de seguridad y restauración 19
2.7. Arquitecturas basadas en "células" 20

3.1. Diagrama de contexto – Entrega de contenido web 26
3.2. Diagrama de contexto – Procesamiento backend asincrónico con failover multirregional 27
3.3. Arquitectura del frontend resiliente en AWS con CloudFront y WAF 28
3.4. Arquitectura del backend multirregional con procesamiento desacoplado en AWS . . 29
3.5. Diagrama de capas lógico-funcional 30

4.1. Stack o pilas región Principal (us-east-1) 40
4.2. Stack o pilas región Secundaria (us-west-1) 40
4.3. Distribución de CloudFront 41
4.4. Resultados de la prueba en artillery 42
4.5. Panel general de artillery 42
4.6. Estadísticas del WAF 43
4.7. Panel general del WAF 43
4.8. Panel general del CloudWatch-Synthetics 44
4.9. Sitios disponibles 45
4.10. Stack o pilas región Principal (us-east-1) 45
4.11. Stack o pilas región Secundaria (us-west-1) 46
4.12. Colas SQS 46
4.13. Registros en CloudWatch intento N°1 47
4.14. Registros en CloudWatch intento N°2 47
4.15. Registros en tabla de proceso Dynamo (us-east-1) 48
4.16. Registros en tabla de idempotencia Dynamo (us-east-1) 48
4.17. Registros en tabla de proceso Dynamo (us-west-1) 49
4.18. Registros en tabla de idempotencia Dynamo (us-west-1) 49
4.19. Mensajes en transito en SQS (AWS) 50
4.20. Mensajes en transito en la SQS (cmd) 50
4.21. Mensajes en disponibles en la DLQ (AWS) 50
4.22. Registros en CloudWatch intentos SQS 51
4.23. Mensajes en transito en la SQS (cmd) 51

4.24. EndPoint status (us-east-1) 52
4.25. EndPoint status (us-west-1) 52
4.26. HealtCheck Route 53 52
4.27. Zonas hospedadas 53
4.28. Status en ambas regiones 53
4.29. Excepción en EndPoint status 54
4.30. Consumo del servicio de status 54
4.31. Comprobación de estado Route 53 55
4.32. Consumo del servicio de status 55

Índice de tablas

3.1. Requisitos funcionales 24
3.2. Requisitos no Funcionales 25
3.3. Relación entre patrones arquitectónicos, atributos de calidad y pilares del WAF . . . 33
3.4. Servicios implementados vs atributos de calidad 34

4.1. Trazabilidad entre pruebas, atributos de calidad para resiliencia y resultados observados 56

Capítulo 1

Introducción

En la actualidad las empresas se encuentran en un punto donde la transformación digital y la acelerada migración de aplicaciones hacia entornos en la nube se ha convertido en una práctica común. Este cambio responde a la necesidad de mejorar la Excelencia Operativa, Fiabilidad y Rendimiento de la mano de proveedores de nube líderes en el mercado, entre los que se encuentra Amazon Web Services (AWS), el cual se ha posicionado como una de las plataformas más utilizadas, gracias a la diversidad de soluciones que ofrece.

Sin embargo, el simple uso de la nube no significa que una aplicación esté diseñada para tolerar fallos o interrupciones. Es en este punto donde la resiliencia, entendida como la capacidad que tiene un sistema para repararse a sí mismo y continuar realizando su trabajo en presencia de fallos, se vuelve un aspecto fundamental en el diseño de arquitecturas modernas.

Aunque AWS proporciona a los arquitectos de software un conjunto de lineamientos y buenas prácticas a través de su Well-Architected Framework, este marco no presenta de forma explícita una arquitectura de referencia que sirva como base para construir soluciones con enfoque resilientes. Esta ausencia representa una amplia brecha para las organizaciones que buscan implementar arquitecturas sólidas frente a fallos (especialmente en entornos críticos donde la alta disponibilidad, confiabilidad, seguridad y rendimiento son importantes).

En este contexto, el objetivo principal de este trabajo de grado fue proponer una arquitectura de referencia enfocada en la resiliencia, utilizando los conceptos y principios establecidos en el Well-Architected Framework, con énfasis en los pilares de excelencia operativa, fiabilidad y rendimiento. La idea no solo era sugerir una arquitectura técnica adecuada, sino proporcionar una plantilla que pudiera usarse como guía en desarrollos futuros.

La elección de este tema obedece a su relevancia en el contexto actual y a su valor tanto académico como práctico. Desde el punto de vista académico, es una contribución interesante investigar arquitecturas en la nube resilientes; y desde la perspectiva práctico, es una herramienta necesaria para quienes buscan la manera de construir sistemas preparados para operar de forma confiable ante cualquier eventualidad.

El objetivo general de esta investigación fue diseñar una arquitectura de software resiliente de referencia para AWS, incorporando prácticas recomendadas del Well-Architected Framework y

atributos de calidad como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación. Para lograrlo, se plantearon los siguientes objetivos específicos:

- Definir los requisitos y características fundamentales de una arquitectura resiliente en la nube, incluyendo la identificación de subatributos clave como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación, que deben ser considerados en el diseño.
- Diseñar la estructura y componentes de la arquitectura de referencia, integrando los principios del Well-Architected Framework de AWS, y adaptándolos para satisfacer los requisitos de resiliencia previamente definidos.
- Evaluar la arquitectura de referencia propuesta mediante su aplicación en una prueba de concepto, para garantizar la fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación.

El enfoque metodológico adoptado fue de tipo aplicado, apoyado en una adaptación del modelo en cascada (Waterfall), dado su carácter secuencial y estructurado. Se inició con la revisión teórica de atributos de calidad relevantes para la resiliencia, seguida de un análisis de los pilares del Well-Architected Framework. A partir de estos insumos, se diseñó la arquitectura de referencia y, finalmente, se construyó una prueba de concepto que permitió validar su funcionamiento frente a escenarios críticos.

Este trabajo aporta al campo de la ingeniería de software y la computación en la nube al proponer un diseño práctico de una arquitectura resiliente, con base en principios ofrecidos por el proveedor de nube AWS. Su impacto potencial radica en brindar una arquitectura de referencia para organizaciones que requieren construir sistemas tolerantes a fallos, especialmente en sectores donde la disponibilidad y la continuidad del servicio son fundamentales.

El documento se encuentra estructurado en cinco capítulos. En el primer capítulo se presenta la introducción general del proyecto, sus objetivos, justificación y metodología. El segundo capítulo desarrolla el marco teórico y contextual que sustenta la investigación. En el tercer capítulo se presenta ya el diseño de la arquitectura propuesta. El cuarto capítulo contiene la evaluación de la arquitectura planteada. Finalmente, el quinto capítulo presenta las conclusiones, recomendaciones para trabajos futuros y lecciones aprendidas.

1.1. Definición del problema

1.1.1. Planteamiento del problema

En la actualidad donde cada vez más las empresas son dependientes de la tecnología, la resiliencia de los sistemas juega un papel importante para asegurar la continuidad del negocio y la satisfacción del cliente. La resiliencia cuenta con varias definiciones, sin embargo, Según Mezzalira et al. (2022b) la resiliencia es “la capacidad de un sistema de recuperarse cuando está estresado por la carga (más

solicitudes de servicio), ataques (ya sea accidental a través de un error o deliberado a través de la intención) y falla de cualquier componente en los componentes de las cargas de trabajo".

Adicionalmente se presenta una definición por parte de Hornsby (2019a) "Los sistemas resilientes abrazan la idea de que los fallos son normales, y que está perfectamente bien ejecutar sistemas en lo que llamamos *modo de fallo parcial*". Finalmente, Guamán (2023) la define como "La resiliencia en software se refiere a la capacidad de un sistema para manejar y recuperarse de fallos y errores, garantizando una operación continua y una calidad de servicio adecuada. En otras palabras, es la habilidad del sistema para mantenerse en funcionamiento, incluso cuando las cosas no van según lo planeado".

A partir de las definiciones antes mencionadas se puede sintetizar en que *La resiliencia en arquitectura de software es la capacidad de un sistema para resistir y recuperarse de fallos, manteniendo su funcionamiento y calidad de servicio incluso en condiciones adversas.*

Según Pan et al. (2023a) la resiliencia en una arquitectura de software incluye seis subatributos, los cuales son:

- La **Fiabilidad**: significa la capacidad del sistema para mantener las funciones del sistema después de ser atacado (Hosseini et al., 2014).
- La **Disponibilidad**: es la proporción de tiempo en que un sistema está en un estado funcional (Cai et al., 2018).
- La **Seguridad**: es la capacidad del sistema para resistir ataques (Zhang et al., 2021).
- La **Robustez**: describe la capacidad del sistema para soportar choques de riesgo y perturbaciones antes de que sea dañado (Zhu et al., 2019).
- La **Rapidez de recuperación**: se define como la velocidad con la que un sistema puede recuperarse de un estado dañado a un estado normal (Zhu et al., 2019).

Ahora que se tiene una definición y características más consolidadas en cuanto a que debe tener una arquitectura resiliente, se presenta la necesidad de diseñar una arquitectura de software de referencia, la cual es definida según SAP (2024) "La arquitectura o modelo de referencia proporciona un vocabulario común, diseños reutilizables y las mejores prácticas de la industria que se utilizan como una restricción para arquitecturas más concretas. Por lo general, la arquitectura de referencia incluye principios de arquitectura comunes, patrones, bloques de construcción y estándares". Esta arquitectura de referencia servirá como base para futuros diseños en el ámbito de la resiliencia, facilitando la implementación de sistemas que puedan resistir y recuperarse de fallos, asegurando la continuidad operativa y la calidad del servicio.

En la actualidad compañías como AWS presentan un abanico de patrones y estilos como es el caso del WAF o el Well-Architected Framework que permite tener una guía al implementar arquitecturas en nube, esto ayudando a los arquitectos a validar qué atributos de calidad deben tener en cuenta al momento de plasmar sus ideas.

Sin embargo, aunque existen patrones y estilos como lo son Patrón multi-AZ, Patrón multi-Region active-active, Arquitecturas basadas en "células", los cuales en el apartado de estado del arte se presentarán con mayor detalle, en conjunto con un análisis de los puntos de mejora de cada uno, hay una notable carencia de guías específicamente orientadas a la resiliencia, lo que deja a las organizaciones sin una base o sin una referencia al momento de implementar una aplicación que cuente con una arquitectura orientada a la resiliencia.

Adicionalmente cabe mencionar que el Well-Architected Framework de AWS proporciona un marco valioso para la creación de arquitecturas en la nube, pero no aborda de manera explícita la necesidad de arquitecturas resilientes de referencia, aun así, los pilares de excelencia operativa, fiabilidad y eficacia del rendimiento, tienen relación con los atributos de calidad de la resiliencia atendiendo aspectos como la fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación.

Este trabajo de grado propone llenar este vacío diseñando una arquitectura resiliente de referencia que considere las prácticas recomendadas por los pilares de excelencia operativa, fiabilidad, eficacia del rendimiento del Well-Architected Framework, atendiendo los atributos de calidad que pertenecen a una arquitectura resiliente como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación con el objetivo que las arquitecturas futuras que tomen esta arquitectura de referencia contemplen los pilares de la excelencia operativa, la fiabilidad y el rendimiento.

1.1.2. Formulación del problema

¿Qué patrones de diseño y mejores prácticas deben ser incorporados en una arquitectura de referencia que abarque los atributos de calidad como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación para implementación de arquitecturas en la nube de AWS?

¿Cómo pueden los pilares de excelencia operativa, fiabilidad y eficacia del rendimiento del Well-Architected Framework ser utilizados para diseñar una arquitectura resiliente que abarque los atributos de calidad como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación?

¿Qué criterios y métricas deben utilizarse para evaluar la efectividad y desempeño de la arquitectura de software resiliente de referencia en términos de fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Diseñar una arquitectura resiliente de referencia que integre los atributos de calidad como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación, teniendo en cuenta los pilares de excelencia operativa, fiabilidad y rendimiento del Well-Architected Framework.

1.2.2. Objetivos específicos

Definir los requisitos y características fundamentales de una arquitectura resiliente en la nube, incluyendo la identificación de subatributos clave como fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación, que deben ser considerados en el diseño.

Diseñar la estructura y componentes de la arquitectura de referencia, integrando los principios del Well-Architected Framework de AWS, y adaptándolos para satisfacer los requisitos de resiliencia previamente definidos.

Evaluar la arquitectura de referencia propuesta mediante su aplicación en una prueba de concepto, para garantizar la fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación.

1.3. Delimitaciones y alcances

El objetivo de este trabajo de grado fue diseñar una arquitectura de referencia resiliente en la nube de AWS basada en las mejores prácticas del Well-Architected Framework, enfocándose en los pilares de excelencia operativa, fiabilidad y rendimiento. Para definir mejor el alcance de este trabajo, se ha considerado cada uno de los objetivos específicos propuestos y discutido en más detalle lo que se ha logrado.

En relación con la identificación de los requisitos técnicos y atributos de calidad esenciales de una arquitectura resiliente, donde el trabajo se centró en los cinco atributos de calidad centrales de la resiliencia: fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación. Esta elección fue el resultado de la lectura de literatura especializada y casos prácticos. Otras cualidades como mantenibilidad, escalabilidad o capacidad de evolución no fueron cubiertas extensamente, a pesar de ser reconocidas como relevantes para la resiliencia.

El análisis del Well-Architected Framework se llevó a cabo solo en tres pilares: excelencia operativa, fiabilidad y rendimiento. Hay seis pilares en el modelo completo, pero los tres restantes (seguridad, eficiencia de costos y sostenibilidad) no se enfocaron teniendo en cuenta que no abordan específicamente el objetivo de agregar resiliencia desde el punto de vista de un arquitecto.

La arquitectura de referencia fue diseñada con una estructura base construida sobre servicios nativos de AWS y mejores prácticas respaldadas por la documentación oficial de AWS. La propuesta se basó en la combinación de dos zonas de disponibilidad, respaldo, monitoreo y automatización. Sin embargo, características avanzadas como estrategias híbridas con otros proveedores, contenedores en EKS o autorrecuperación basada en IA, no se incluyeron porque requerirían más tiempo y trabajo de desarrollo en el proceso.

Finalmente, en términos de validación mediante prueba de concepto, se desarrolló un entorno controlado que permitió evaluar la funcionalidad básica de la arquitectura frente a eventos simulados de fallo. Esta validación no contempló pruebas de carga extensivas, análisis de rendimiento bajo alta concurrencia, ni evaluaciones en ambientes productivos, debido a restricciones de infraestructura y alcance del proyecto académico.

Alcances

- Se logró diseñar una arquitectura resiliente de referencia aplicando principios del Well-Architected Framework de AWS.
- Se desarrolló una prueba de concepto funcional, capaz de demostrar el comportamiento esperado frente a fallos simulados.

Delimitaciones

- El estudio se limitó al proveedor AWS, sin considerar soluciones multicloud o híbridas.
- La arquitectura fue validada en un entorno de laboratorio, sin pruebas en producción ni cargas reales.
- No se abordaron pilares como eficiencia de costos o sostenibilidad dentro del análisis del Well-Architected Framework.
- No se incluyeron mecanismos avanzados de autoescalado predictivo, balanceo global o automatización por eventos complejos.

1.4. Justificación del trabajo de grado

Este trabajo se orientó a abordar un problema muy común en las empresas hoy en día, la ausencia de arquitecturas de software que estén diseñadas explícitamente para la resiliencia dentro de un entorno en la nube. En un escenario donde los sistemas de TI son críticos para mantener las operaciones y la satisfacción de los usuarios finales, las interrupciones del servicio que ocurren debido a fallos técnicos, ataques o sobrecarga pueden generar impactos económicos, reputacionales y operativos significativos para las empresas.

La propuesta de una arquitectura de referencia resiliente en AWS ayuda a mitigar esas condiciones a los que los sistemas se ven sometidos, al tener una guía para que los equipos técnicos se apoyen al diseñar sistemas capaces de resistir y recuperarse ante eventos imprevistos, lo que ayuda a hacer el sistema resiliente. Al seguir principios del Well-Architected Framework, esta arquitectura proporciona una solución simple y aplicable que puede ser utilizada fácilmente, algo valioso en áreas sensibles como finanzas, salud, logística y servicios digitales.

Si el problema abordado no hubiera sido enfrentado, muchas empresas continuarían desarrollando arquitecturas en la nube por ensayo y error o empíricamente, teniendo en cuenta que no hay una guía sobre cómo incluir características como fiabilidad, disponibilidad, seguridad y rendimiento. Esto las haría más vulnerables a riesgos relacionados con operaciones, pérdida de datos, pérdida de servicios clave o sanciones regulatorias en entornos críticos AWS (2023); Hornsby (2019b).

Desde una perspectiva monetaria, diseñar arquitecturas resilientes ayuda a reducir el costo de la no disponibilidad al disminuir la cantidad total de tiempo que una aplicación está inactiva y reducir el número de interrupciones que requieren que corramos de un lado a otro. Por ejemplo, (Gartner Inc., 2023) afirma un costo de USD 5,600 por minuto de inactividad en algunos sectores.

Además, la solución propuesta se considera una contribución académica y técnica relevante, dado que un diseño estructurado y documentado de una arquitectura de referencia tolerante a fallos no se presenta a menudo en la documentación técnica de AWS, lo cual es reconocido en (Mezzalira et al., 2022a) y (Pan et al., 2023b). En consecuencia, el trabajo contribuye tanto a la comunidad profesional como académica de la ingeniería de software en la nube al avanzar en las mejores prácticas y la resiliencia a fallos en un entorno distribuido.

1.5. Criterios de selección del entorno de nube de AWS y marco de diseño WAF

La selección de Amazon Web Services (AWS) como plataforma tecnológica se fundamentó en su madurez en el mercado, su cobertura global, y la amplitud de un portafolio de servicios resilientes, respaldado por documentación técnica robusta, adicionalmente por la madurez del conocimiento en este proveedor de nube para la implementación de la prueba de concepto. AWS ofrece de forma nativa herramientas como replicación automática, balanceo de carga, failover y monitoreo centralizado, elementos clave para construir soluciones capaces de resistir fallos sin comprometer su operación.

Del mismo modo, se optó por utilizar el Well-Architected Framework (WAF) como referencia metodológica, debido a que proporciona principios claros y ampliamente aceptados por la industria para evaluar y mejorar arquitecturas en la nube. Si bien el WAF no fue diseñado exclusivamente con un enfoque en resiliencia, su estructura basada en seis pilares permite orientar decisiones técnicas con criterios sólidos. En el contexto de este trabajo, la atención se enfocó en tres de esos pilares: fiabilidad, excelencia operativa y rendimiento, por su relación directa con la capacidad de un sistema para mantenerse disponible, eficiente y recuperarse ante eventualidades.

1.6. Metodología de la investigación

Con el fin de resolver el problema planteado y cumplir con los objetivos establecidos, se aplicó una adaptación de la metodología Waterfall (Cascada). Esta elección se basó en la necesidad de que

el proyecto estuviera organizado en pasos claros, para tener una transición ordenada de una idea a la validación de la arquitectura propuesta. El procedimiento de investigación fue el siguiente:

- **Recolección y análisis de información**: Al principio, se realizó una investigación documental y bibliográfica analizando tres ejes:

  1. Atributos de calidad relacionados con la resiliencia en arquitecturas de software.
  2. El marco de mejores prácticas de AWS llamado Well-Architected Framework (WAF).
  3. Patrones de diseño indicados por AWS para desarrollar soluciones tolerantes a fallos.

  Estos datos se obtuvieron de artículos científicos, blogs técnicos oficiales, documentación de AWS y literatura especializada.

- **Estructuración de los requisitos y las características clave**: Según el estudio anterior, se seleccionaron los atributos de calidad directamente medibles de la resiliencia como fiabilidad, disponibilidad, seguridad, robustez y velocidad. Cada uno de estos atributos de calidad se relacionó con requisitos técnicos, así como con criterios de evaluación, permitiendo el establecimiento de una base clara para el diseño posterior. Dicha estructuración se presentará como una tabla de trazabilidad entre atributos de calidad, pilares de WAF y patrones relacionados.

- **Diseño de la arquitectura de referencia**: Se desarrolló una arquitectura de referencia en la nube de AWS. Esta arquitectura utilizó varios recursos nativos de AWS como Route 53, AWS Lambda, Dynamo, S3, WAF, CloudWatch, IAM, entre otros, con principios de diseño distribuidos y desacoplados. Se consideraron al menos tres patrones arquitectónicos importantes: multi-AZ, respaldo multi-región y diseño basado en células. Los patrones se seleccionaron en función de su contribución a los atributos de calidad.

- **Construcción y validación de la prueba de concepto**: Se realizó una prueba de concepto funcional, modelando las capacidades de un sistema en la nube de AWS. Esta prueba permitió validar la reacción del sistema ante interrupciones controladas (por ejemplo, caída de zonas de disponibilidad, latencia de respuesta o indisponibilidad del servicio). La validación provino de registros, métricas de CloudWatch que producimos y observando directamente cómo el sistema pudo seguir funcionando y recuperarse cuando ocurrieron problemas.

- **Interpretación e informe de resultados**: Los resultados obtenidos durante la ejecución de la prueba de concepto fueron organizados y analizados en función de los cinco atributos de calidad clave. Se utilizó una matriz de validación para relacionar los resultados observados con los objetivos del proyecto. Este análisis permitió identificar fortalezas del diseño, posibles limitaciones técnicas y oportunidades de mejora.

1.7. Resultados obtenidos

Los principales resultados obtenidos en el desarrollo del proyecto fueron el diseño de una arquitectura de referencia resiliente en la nube de AWS y la implementación de una prueba de concepto funcional (POC).
La arquitectura propuesta integró componentes nativos de AWS y patrones reconocidos como Multi-AZ, respaldo multi-región y segmentación por células, permitiendo una estructura distribuida, escalable y tolerante a fallos. Este diseño fue orientado por los pilares del Well-Architected Framework y los atributos de calidad clave para resiliencia definidos durante la investigación.
Como complemento, se construyó una prueba de concepto que permitió validar el comportamiento de la arquitectura frente a escenarios simulados de interrupción y falla. Esta validación evidenció la capacidad del sistema para mantener la continuidad operativa, recuperarse automáticamente ante eventos adversos y preservar la integridad del servicio, confirmando así la efectividad del diseño propuesto.

Capítulo 2

Marco de referencia

2.1. Marco Teórico

2.1.1. Bases Teóricas

Computación en la nube

La informática o computación en la nube es la distribución de recursos de TI bajo demanda disponibilizados a través de Internet mediante un esquema de pago por uso. En lugar de comprar, poseer y mantener servidores y centros de datos físicos, puede acceder a servicios tecnológicos, como capacidad informática, almacenamiento y bases de datos, en función de sus necesidades a través de un proveedor de la nube como Amazon Web Services (AWS, 2023b).

Modelos de informática en la nube

- **Infraestructura como servicio (IaaS)**

  La infraestructura como servicio (IaaS) contiene los componentes básicos de la TI en la nube y por lo general proporciona acceso a funciones de red, a computadoras (virtuales o en hardware dedicado) y a espacio de almacenamiento de datos. Los proveedores de infraestructura como servicio pueden ayudarlo con el más alto nivel de flexibilidad y control de administración sobre sus recursos de TI y es lo más similar a los recursos de TI existentes con los que muchos departamentos de TI y desarrolladores están familiarizados hoy en día (AWS-Cloud, 2023).

- **Plataforma como servicio (PaaS)**

  Los proveedores de plataformas como servicio (PaaS) eliminan la necesidad de que las organizaciones gestionen la infraestructura subyacente (normalmente hardware y sistemas operativos) y esta integración le permite centrarse en la implementación y gestión de sus aplicaciones. Esto contribuye a mejorar el nivel de eficiencia, teniendo en cuenta que no debe preocuparse por el aprovisionamiento de recursos, la planificación de la capacidad, el mantenimiento del software, la implementación de parches ni ninguna de las demás arduas tareas que conlleva la ejecución de su aplicación (AWS-Cloud, 2023).

- **Software como servicio (SaaS)**

  Los proveedores de software como servicio (SaaS) le proporcionan aplicaciones de software que el proveedor ejecuta y administra. En la mayoría de los casos, las personas que hablan de software como servicio se refieren a aplicaciones de usuario final de terceros. Con una oferta de SaaS, no tiene que pensar en cómo se mantiene el servicio ni en cómo se administra la infraestructura subyacente. Sólo debe preocuparse por cómo utilizar ese sistema de software concreto (AWS-Cloud, 2023).

Migración a la nube

La migración a la nube es el proceso en el cual se trasladan activos digitales como datos, aplicaciones y recursos de TI a la nube. Tradicionalmente, las organizaciones ejecutaban sus aplicaciones y servicios de TI en una infraestructura de TI autogestionada que se mantenía en un centro de datos local. Algunas organizaciones pueden tener miles de bases de datos, aplicaciones y software de sistema funcionando en el sitio. (Amazon, 2023).

Resiliencia Informática

La resiliencia informática es la capacidad de un sistema para recuperarse de un fallo y conservar la confiabilidad del servicio cuando este las presenta, su objetivo es asegurar que todas las operaciones comerciales estén protegidas, para que así una amenaza o incumplimiento no afecte todo el negocio. Esta resiliencia informática permite contar con varios enfoques que posibilitan mantener una entrega continua de las operaciones mientras hay una interrupción, todos estos enfoques en conjunto resguardan el ciclo de vida de los procesos necesarios para planificar, detectar, responder, recuperarse y mejorar después de una limitación asociada con un fallo o un ataque informático (Martínez et al., 2023).

Resiliencia en Términos de Otros Atributos de Calidad

- **Adaptabilidad** Según Firesmith (2019) un sistema que puede adaptarse a sí mismo a condiciones cambiantes podría responder y recuperarse de eventos y condiciones adversas, así como de cualquier daño que puedan causar.

- **Disponibilidad** Según Firesmith (2019) un sistema resiliente necesita mantener la disponibilidad de sus capacidades/servicios críticos durante condiciones adversas y a pesar de la ocurrencia de eventos adversos.

- **Mantenibilidad** Según Firesmith (2019) un sistema se considera más resistente si es capaz de repararse a sí mismo en lugar de depender de un mantenedor humano.

- **Rendimiento** Según Firesmith (2019) los altos niveles de resiliencia en términos de detección disminuyen el rendimiento en términos de capacidad de procesamiento y tiempo de respuesta.

- **Fiabilidad** Según Firesmith (2019) un sistema resistente necesita mantener la confiabilidad de sus capacidades/servicios críticos durante condiciones adversas y a pesar de la ocurrencia de eventos adversos.

- **Reparabilidad** Según Firesmith (2019) un sistema que puede repararse a sí mismo, puede recuperarse mejor después de un evento adverso.

Principios de la Resiliencia en la Nube

> La capacidad de recuperación no es un estado binario. Nadie puede pretender tener una capacidad de recuperación absoluta, ni usted ni ningún proveedor de nubes. Las nubes deberían ser tan o incluso más resilientes que la infraestructura local, pero sólo si el equipo de Infraestructura y operación las utiliza de forma resiliente (Gartner-Inc., 2023).

Los analistas de Gartner recomiendan a los responsables de Infraestructura y operación que se centren en nueve principios clave para mejorar la resistencia de la nube (ver Figura 2.1).

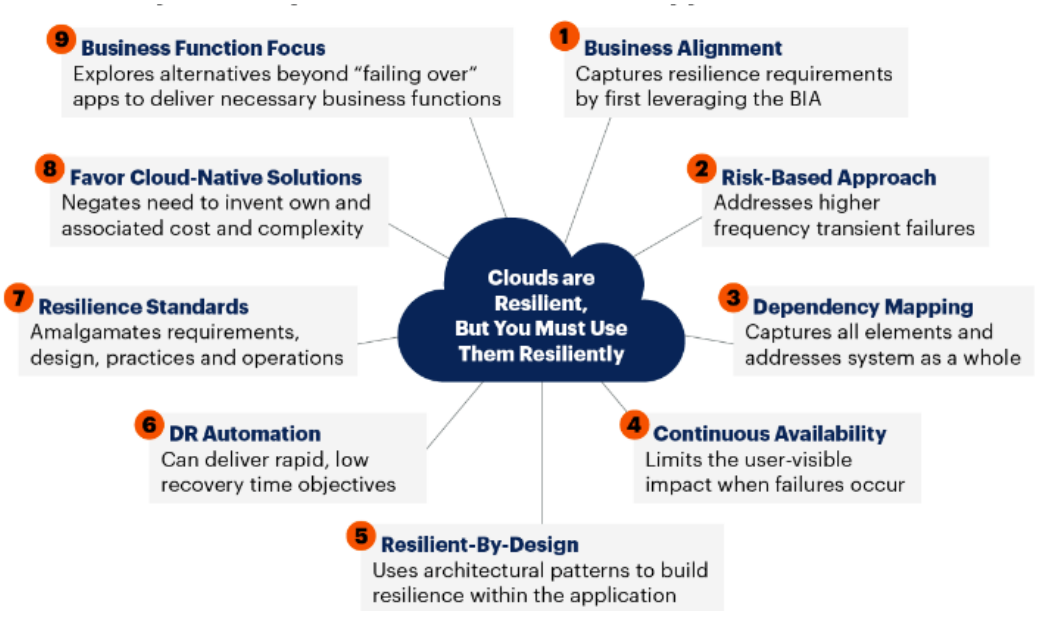


Figura 2.1: Principios para Mejorar la Resiliencia de las Aplicaciones en la Nube
Fuente: Gartner highlights 9 principles to improve cloud resilience

Well-Architected Framework

AWS Well-Architected ayuda a los arquitectos de la nube a crear una infraestructura segura, de alto rendimiento, resistente y eficiente para una variedad de aplicaciones y cargas de trabajo. Este marco, creado en torno a seis pilares (excelencia operativa, seguridad, fiabilidad, eficiencia de rendimiento, optimización de costos y sostenibilidad), ofrece un enfoque coherente para que los clientes y los socios evalúen las arquitecturas e implementen diseños escalables (AWS, 2023a).

- El **pilar de la excelencia operativa** se concentra en ejecutar y monitorear los sistemas y en mejorar constantemente los procesos y los procedimientos. Entre los temas clave se incluyen la automatización de cambios, la respuesta a eventos y la definición de estándares para administrar las operaciones diarias (AWS, 2023a).

- El **pilar de fiabilidad** se centra en las cargas de trabajo que realizan las funciones previstas y en cómo recuperarse rápidamente de los errores para cumplir con las demandas. Entre los temas clave se incluyen el diseño de sistemas distribuidos, la planificación de la recuperación y cómo adaptarse a los requisitos cambiantes (AWS, 2023a).

- El **pilar de eficacia del rendimiento** se centra en la asignación estructurada y simplificada de TI y en los recursos informáticos. Entre los temas clave se incluyen la selección de los tipos y tamaños de recursos optimizados para los requisitos de la carga de trabajo, la supervisión del rendimiento y el mantenimiento de la del rendimiento a medida que evolucionan las necesidades de la empresa (AWS, 2023a).

2.2. Estado del Arte

En la actualidad y la constante evolución de la computación en la nube ha transformado la forma en que las organizaciones diseñan, implementan y gestionan sus aplicaciones. En este contexto dinámico y en constante cambio, el estado del arte en el diseño de arquitecturas resilientes en la nube de Amazon Web Services (AWS) representa un proceso de investigación de prácticas y avances tecnológicos que definen el panorama actual de la resiliencia en entornos cloud, a continuación se destacan las siguientes prácticas propuestas por AWS.

Patrón multi-AZ

Según Nandwani et al. (2023) este es un patrón de arquitectura basado en la nube que introduce Zonas de Disponibilidad (AZs, por sus siglas en inglés) en su arquitectura para aumentar la resiliencia de su sistema. El patrón utiliza una arquitectura Multi-AZ donde las aplicaciones operan en múltiples AZs dentro de una sola Región de AWS. (ver Figura 2.2) .

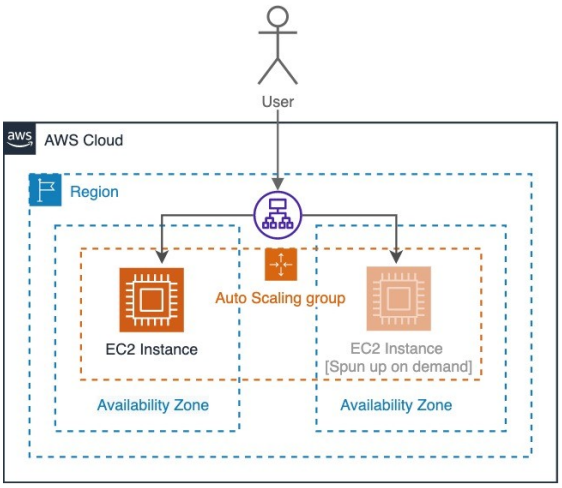


Figura 2.2: Patrón - Multi-AZ
Fuente: Entender los patrones de resiliencia y las consideraciones claves para arquitecturar eficientemente en la nube

Posibles puntos de Mejora: Este patrón propuesto por AWS apunta a brindar una alta disponibilidad de la aplicación dentro de una sola región, partiendo del enfoque hacia la disponibilidad se puede extender su estrategia a un enfoque Multi-Región, lo que permitiría resiliencia frente a fallos regionales completos. También, se podría contemplar la integración de mecanismos de automatización y orquestación que permitan una recuperación ante fallos, teniendo en cuenta que se deben optimizar latencias entre regiones, que permitan garantizar la continuidad operativa de la aplicación.

Construyendo una arquitectura segura, altamente disponible en AWS

Según Martin et al. (2022) MPM Software utiliza AWS WAF como firewall de aplicaciones para proteger su suite de soluciones contra ataques web y bots que puedan afectar la disponibilidad de la plataforma. En la región de Irlanda, la arquitectura está asegurada por una Virtual Private Cloud (VPC), brindándoles control total sobre la conectividad y seguridad de la aplicación. Además, la configuración de alta disponibilidad entre diferentes zonas y la orquestación del grupo de Auto Escalado son elementos clave. Para las bases de datos, utilizan el servicio RDS de AWS, que proporciona alta disponibilidad y una política de copias de seguridad en S3, permitiendo la recuperación de datos desde hace 5 minutos hasta 31 días atrás. (ver Figura 2.3) .

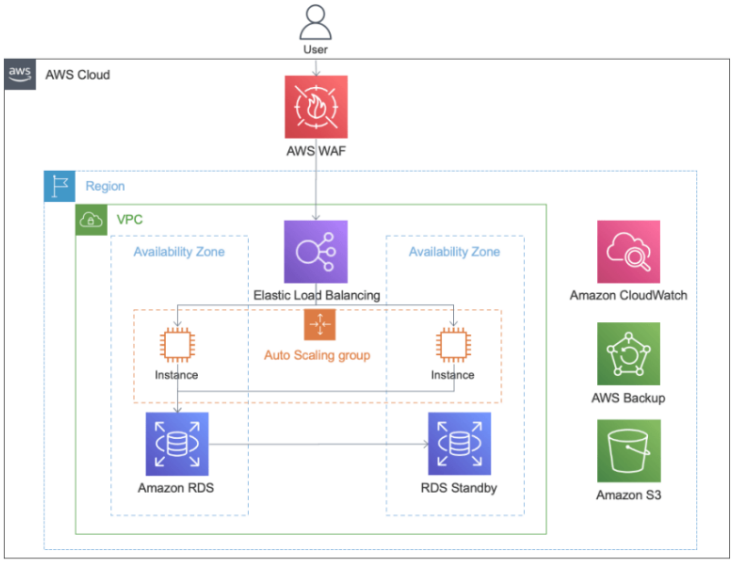


Figura 2.3: Arquitectura segura, altamente disponible en AWS
Fuente: Construyendo una arquitectura segura, altamente disponible en AWS

Posibles puntos de Mejora: El enfoque que implementa MPM en su Software es sólido, pero podría mejorarse al incorporar medidas de seguridad adicionales que se adapten dinámicamente a las amenazas emergentes, adicionalmente se podría apuntar a la resiliencia mediante un enfoque Multi-Región para recuperación ante desastres, mejorar el autoescalado con técnicas que permitan optimizar la disponibilidad y costos. Además, se pude implementar la recuperación de datos mediante replicación continua y restauración casi instantánea, minimizando así el tiempo de inactividad y la pérdida de datos.

Patrón distribución del portafolio de aplicaciones

Según Nandwani et al. (2023) este patrón utiliza un patrón Multi-Región para aumentar la resiliencia funcional, como se muestra en la (Figura 2.4). Distribuye diferentes aplicaciones en múltiples Regiones. Por ejemplo *Example Corp* proporciona servicios bancarios, como verificaciones de saldo, a consumidores en múltiples canales digitales. Estos servicios están disponibles para los consumidores a través de una aplicación móvil, un centro de contacto y aplicaciones basadas en web. Cada canal digital se implementa en una Región separada, lo que mitiga una interrupción del servicio regional.

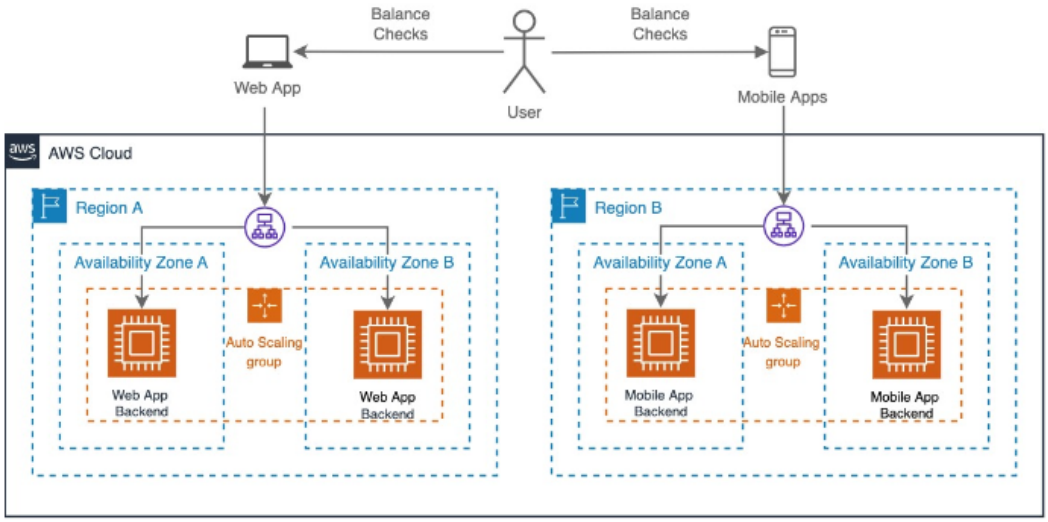


Figura 2.4: Patrón distribución del portafolio de aplicaciones
Fuente: Entender los patrones de resiliencia y las consideraciones claves para arquitecturar eficientemente en la nube

Posibles puntos de Mejora: El patrón distribución del portafolio de aplicaciones, puede ser una buena solución en cuanto a disponibilidad, pero podría acarear con altos costos, teniendo en cuenta que se tendrían regiones disponibles para uno de los aplicativos de la compañía y se puede presentar el caso de que uno de estos no requiera la totalidad de recursos destinados. En ese caso, el diseño debería implementar un enfoque híbrido de distribución regional, el cual constaría de una región primaria donde se centralizan la mayoría de las operaciones, mientras que las otras regiones actúan como nodos secundarios, activándose sólo en caso de fallo en la región primaria. Este diseño reduce la complejidad operativa y los costos asociados con la replicación y sincronización de datos en tiempo real en todas las regiones.

Patrón multi-Region active-active

Según Nandwani et al. (2023) las aplicaciones centrales de banca y gestión de relaciones no toleran interrupciones. Utilizan este patrón para desplegar estas aplicaciones porque tiene un Tiempo de Recuperación Objetivo (RTO) en tiempo real y un Objetivo de Punto de Recuperación (RPO) de pérdida de datos cercana a cero. Ejecutan su carga de trabajo simultáneamente en múltiples regiones, lo que les permite atender el tráfico desde todas las regiones simultáneamente. Este patrón no sólo mitiga las interrupciones regionales, sino que también cumple con sus requisitos de tolerancia cero (ver Figura 2.5).

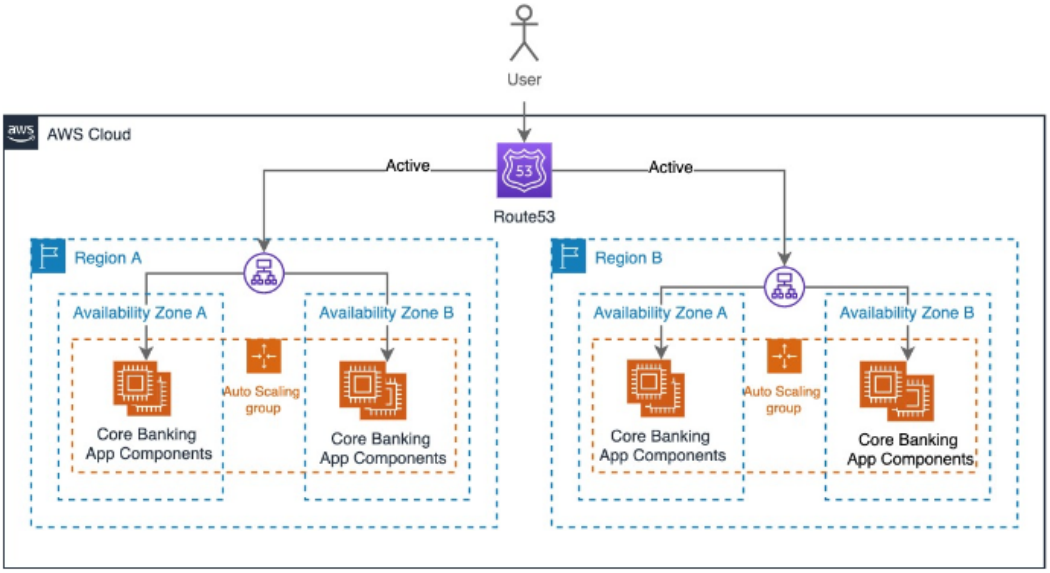


Figura 2.5: Patrón multi-Region active-active
Fuente: Entender los patrones de resiliencia y las consideraciones claves para arquitecturar eficientemente en la nube

Posibles puntos de Mejora: Este patrón Multi-Región active-active es altamente efectivo para aplicaciones críticas que requieren tolerancia cero a fallos, pero su alta redundancia y complejidad operativa pueden generar costos elevados. Una posible mejora sería implementar la activación dinámica de regiones secundarias, adicionalmente optimizar la sincronización de datos para equilibrar consistencia, eficiencia de recursos y aplicar una escalabilidad selectiva, es decir, implementar políticas de escalabilidad que ajusten automáticamente la cantidad de recursos desplegados en cada región en función de la demanda en tiempo real.

Copia de seguridad y restauración

Según Eliot (2021) esta estrategia muestra copias de seguridad de varios recursos de datos de AWS. Las copias de seguridad se crean en la misma Región que su fuente y también se copian en otra Región. Esto le brinda la protección más efectiva contra desastres de cualquier alcance de impacto. Para la conmutación por error entre regiones, además de la recuperación de datos de la copia de seguridad, también debe poder restaurar su infraestructura en la Región de recuperación. (ver Figura 2.6).

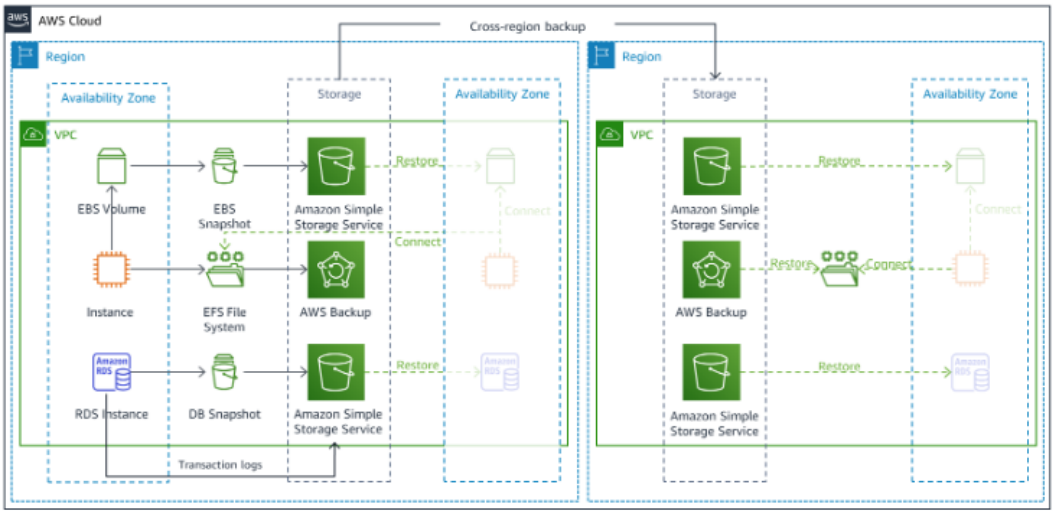


Figura 2.6: Copia de seguridad y restauración
Fuente: Arquitectura de Recuperación de Desastres (DR) en AWS, Parte I: Estrategias para la Recuperación en la Nube

Posibles puntos de Mejora: Este patrón propuesto por AWS busca directamente la protección contra desastres, Resiliencia y Continuidad del Servicio. Aunque se pueden considerar posibles puntos de mejora, como lo pueden ser temas de costos y gestión de recursos teniendo en cuenta que mantener copias de seguridad en múltiples regiones los costos se elevan, adicionalmente el proceso de restauración en una nueva región puede llegar a ser compleja y se requiere tener muy detallado los procesos.

Arquitecturas basadas en "células"

Según Villabona (2022) el patrón de célula hace referencia a una colección de componentes agrupados que forman una unidad funcional completa dentro de una aplicación. Cada célula es autónoma y capaz de realizar todas las funciones necesarias para satisfacer las necesidades de negocio asignadas a su carga de trabajo, es decir, sin depender de otra célula. Este patrón permite que cada célula sea desplegada, administrada y monitoreada de forma independiente, lo que ofrece flexibilidad y resiliencia en la operación (ver Figura 2.7).

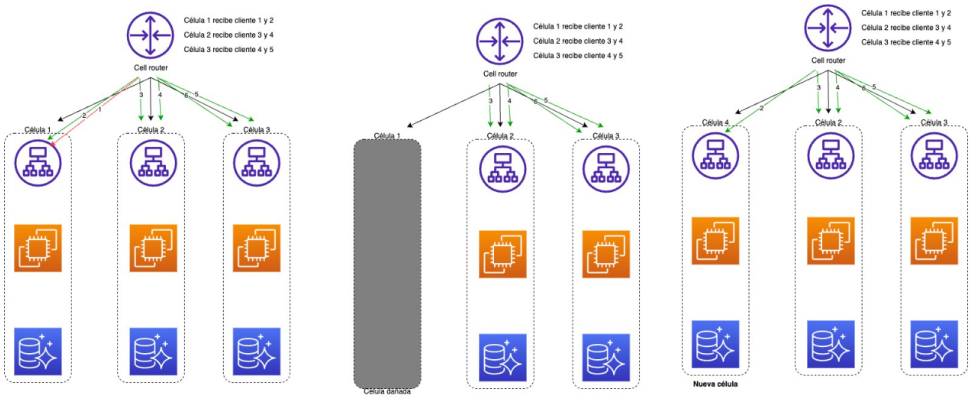


Figura 2.7: Arquitecturas basadas en "células"
Fuente: Creación de cargas de trabajo resilientes usando arquitecturas basadas en "células"

Posibles puntos de Mejora: Este patrón podría mejorarse al integrar capacidades de auto recuperación y escalabilidad automática dentro de cada célula para que se pueda ajustar dinámicamente a fallos o variaciones en la carga, adicionalmente se puede implementar un sistema de monitoreo centralizado que proporcione una visión global del rendimiento y estado de todo el sistema, que permita facilitar la toma de decisiones o una respuesta coordinada ante incidentes.

2.3. Resumen del capítulo

En este capítulo se presentaron los fundamentos conceptuales y técnicos detrás del diseño y la construcción de arquitecturas resilientes en la nube. Inicialmente, se presentó una revisión del concepto de resiliencia en sistemas de software, donde se destacaron sus atributos de calidad: fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación. A partir de fuentes académicas y técnicas, se definió cómo estos atributos de calidad permiten mantener la continuidad operativa frente a fallos, ataques o condiciones adversas.

Posteriormente, se analizó el Well-Architected Framework (WAF) de AWS, con un enfoque en los pilares de excelencia operativa, fiabilidad y rendimiento, los cuales se relacionan directamente con los objetivos de resiliencia. Estos pilares se vieron como una 'base metodológica' para ayudar a guiar las decisiones arquitectónicas a lo largo del proyecto.

Finalmente, en la sección de estado del arte, se examinaron algunos de los patrones arquitectónicos sugeridos por AWS, como el patrón Multi-AZ, Multi-Región active-active, distribución del portafolio de aplicaciones, copia de seguridad multi-región y arquitecturas basadas en células. Se señalaron sus pros y contras para cada patrón, facilitando la extracción de mejores prácticas y áreas de mejora. Este análisis fue clave para apoyar la propuesta hacia una arquitectura de referencia que aproveche sus beneficios, abordando al mismo tiempo sus debilidades mediante el uso de escalabilidad dinámica y monitoreo centralizado. De esta manera, el capítulo proporcionó las bases conceptuales y técnicas necesarias para el diseño y validación.

Capítulo 3

Desarrollo del Proyecto

3.1. Atributos de calidad clave para la resiliencia

En el entorno de computación en la nube, la resiliencia se refiere a la capacidad del sistema para operar correctamente o recuperarse de fallos, interrupciones o eventos inesperados. A continuación, se ampliarán la definición de los atributos de calidad y subatributos mínimos que una arquitectura de nube resiliente debe cumplir y van de la mano con los establecidos por las prácticas presentadas en el Marco de Buenas Prácticas de AWS. Es mediante la aplicación de estos atributos de calidad que se puede evaluar si una solución es capaz de mantener sus niveles de servicio (SLA), salvaguardar la información y recuperarse de forma rápida, incluso en la situación de desastres.

1. **Fiabilidad**: Este atributo se refiere a la capacidad del sistema para funcionar de forma consistente y correcta bajo condiciones operacionales definidas, durante un periodo determinado. Una arquitectura confiable debe soportar fallos parciales, errores inesperados o condiciones anómalas sin comprometer la operación general del sistema (iso, 2023)

    - **Subatributos**
      Tolerancia a fallos Esta es la capacidad del sistema para seguir operando correctamente incluso cuando uno o más de sus componentes fallan. Esto se logra mediante redundancia, aislamiento de fallos y mecanismos automáticos de recuperación (iso, 2023).
      Recuperación automática Esta es la capacidad del sistema para detectar fallos y restaurar automáticamente el estado funcional sin interacción humana. Este comportamiento se basa en health checks, políticas de restart y eventos automatizados (iso, 2023).
      Redundancia activa Esta es la Implantación de múltiples instancias o componentes funcionales que operan en paralelo para eliminar puntos únicos de fallo. No es sólo respaldo, sino operación continúa distribuida (iso, 2023).

2. **Disponibilidad**: Este atributo representa la capacidad del sistema para estar operativo y accesible cuando los usuarios o procesos lo requieran. Se mide típicamente como un porcentaje (SLA) y está estrechamente vinculada con la capacidad de respuesta ante fallas regionales, zonales o lógicas (NIST, 2020).

    - **Subatributos**
      Alta disponibilidad Diseño arquitectónico orientado a mantener el servicio accesible el mayor porcentaje del tiempo posible, utilizando componentes distribuidos, autogestionados y balanceados (NIST, 2020).

Conmutación por error Mecanismo que detecta fallos en un componente primario y redirige automáticamente la carga a una instancia secundaria saludable. Puede ser activo-activo o activo-pasivo (NIST, 2020).
Distribución geográfica Despliegue del sistema en múltiples regiones o zonas de disponibilidad para minimizar el impacto de interrupciones locales o catastróficas (NIST, 2020).

3. **Seguridad**: Este atributo en el contexto de resiliencia implica la capacidad del sistema para resistir accesos no autorizados, manipulaciones maliciosas y fugas de información, mientras mantiene su operatividad (NIST, 2020).

   - **Subatributos**
     Autenticación y autorización Verificación de identidades y control de acceso basado en roles y políticas. Previene accesos no autorizados y restringe las operaciones de cada entidad en el sistema (NIST, 2020).
     Encripción en tránsito y reposo Protección de los datos tanto mientras se transmiten por la red (TLS/SSL), como cuando se almacenan (AES-256, KMS). Mitiga el riesgo de exposición accidental o acceso malicioso (NIST, 2020).
     Auditoría y trazabilidad Registro detallado de operaciones, accesos, cambios de configuración y comportamiento del sistema, con fines de seguridad, cumplimiento y respuesta ante incidentes (NIST, 2020).

4. **Robustez**: Este atributo describe la capacidad del sistema para operar de manera aceptable ante condiciones adversas, como entradas erróneas, degradación de servicios o cargas inesperadas. Se considera un atributo esencial en entornos dinámicos como la nube, donde los errores no se pueden evitar, pero sí controlar y aislar (iso, 2023).

   - **Subatributos**
     Desacoplamiento lógico Separación de componentes mediante colas, mensajes o eventos que evitan dependencias rígidas entre servicios. Mejora tolerancia a fallos parciales y simplifica el escalado (iso, 2023).
     Manejo de errores estructurado Implementación de políticas de reintento, circuit breakers, dead-letter queues (DLQ) y manejo explícito de excepciones. Evita errores silenciosos o reintentos infinitos (iso, 2023).
     Autoescalado horizontal Ajuste dinámico de la capacidad de cómputo agregando o eliminando instancias para adaptarse a la demanda. Mantiene el rendimiento y evita caídas por sobrecarga (iso, 2023).

5. **Rapidez de recuperación**: Este atributo mide la capacidad del sistema para restaurar su funcionamiento normal en un periodo aceptable tras una interrupción, minimizando la pérdida de datos y la indisponibilidad de servicios. Implica tanto la automatización de recuperación como el diseño para el fallo (NIST, 2020).

- **Subatributos**
  Recovery Time Objective (RTO) Tiempo máximo aceptable entre la ocurrencia de una falla y la restauración del servicio. Influye directamente en la experiencia del usuario y continuidad del negocio (iso, 2023).
  Recovery Point Objective (RPO) Cantidad máxima de datos que el sistema puede perder tras una interrupción. Se mide en tiempo y determina la frecuencia de replicación o backups. (iso, 2023).
  Automatización de restauración Capacidad de desplegar de forma automática recursos, datos y configuraciones en entornos nuevos o alternativos sin intervención manual (iso, 2023).

3.2. Diseño

Para llevar a cabo la definición de una arquitectura resiliente de referencia en la nube se debe tener una serie de requisitos que permitan tomar decisiones sobre los aspectos técnicos, estructurales y operativos en la fase de diseño del sistema. Estas restricciones son las condiciones mínimas que la arquitectura debe satisfacer para mantener una continuidad en circunstancias desfavorables y que la pérdida de datos o el impacto en el rendimiento se reduzcan.

1. **Descripción de la propuesta**: La propuesta de este trabajo consistió en diseñar una arquitectura resiliente de referencia en la nube de Amazon Web Services (AWS) que sirviera como guía para crear sistemas capaces de seguir funcionando bajo todo tipo de condiciones adversas. La arquitectura integró directrices del Well-Architected Framework y enfatizó sus estándares de operación, fiabilidad y rendimiento. El propósito de esta propuesta fue presentar una solución técnica que permita facilitar la implementación práctica de resiliencia en entornos en la nube, abordando una necesidad existente: la falta de una arquitectura de referencia específica para ayudar a diseñar sistemas que puedan seguir funcionando ante posibles fallos, especialmente en aplicaciones críticas o de alta disponibilidad. La arquitectura está diseñada para sacar provecho de patrones resilientes a través de procesos modulares y escalables. Adoptando tecnologías como Multi-AZ, arquitecturas activas-activas y arquitecturas por células.

   Los usuarios objetivo de esta propuesta son principalmente arquitectos de software, ingenieros de plataformas, equipos de DevOps y profesionales de infraestructura en la nube que son responsables de diseñar o implementar soluciones en la nube pública en AWS. Además, también puede ser útil como recurso material para investigadores y estudiantes involucrados en temas como la resiliencia, la planificación de continuidad del negocio o arquitecturas de sistemas distribuidos.

   La solución propuesta ayudará a mejorar la situación actual en las arquitecturas al ser una guía estructurada y reutilizable, que reduce la dependencia de enfoques empíricos o aislados. También contribuye con la disminución de los riesgos de indisponibilidad, sobrecostos por sobredimensionamiento de recursos y retrasos en la recuperación ante fallos. En conclusión,

esta propuesta de arquitectura representa una solución práctica que busca fortalecer la resiliencia operativa, optimizar el uso de recursos en la nube y permitir responder de manera más eficiente a incidentes críticos.

2. **Requisitos**:

Requisitos funcionales:

Requisito funcional	**Descripción**
Multi-región (Soporte para replicación y failover)	La solución debe estar preparada para operar en más de una región AWS en escenarios de recuperación ante desastres.
Procesamiento asíncrono desacoplado	Los servicios deben estar desacoplados mediante colas o buses de eventos, permitiendo tolerancia a fallos y recuperación progresiva.
Failover automatizado	Deben existir mecanismos que detecten fallos y redirijan automáticamente el tráfico o las operaciones a una zona o región secundaria sin intervención manual.
Replicación de datos y estado	La arquitectura debe implementar replicación automática (síncrona o asíncrona) de datos críticos entre zonas o regiones para preservar la continuidad operativa.
Observabilidad centralizada	El sistema debe contar con monitoreo, alarmas y trazabilidad completa del comportamiento y del estado de cada componente, idealmente de forma unificada.
Control de acceso granular	La solución debe integrar mecanismos de autenticación y autorización por roles, minimizando privilegios y permitiendo auditoría por operación.

Tabla 3.1: Requisitos funcionales

Requisitos no funcionales:

Requisito no funcional	Valor de referencia	Justificación técnica
SLA (Availability)	$\geq 99.99\,\%$	Implica ≤ 5 minutos de inactividad al mes. Requiere alta disponibilidad y redundancia en múltiples zonas/regiones.
RTO (Recovery Time Objective)	≤ 5 minutos	Tiempo máximo tolerado para restaurar el sistema. Soportado con automatización de recuperación y failover.
RPO (Recovery Point Objective)	≤ 1 minuto	Tiempo máximo de datos que se pueden perder. Requiere replicación continua o versionado.
Escalabilidad automática	Obligatoria	El sistema debe ajustar su capacidad sin intervención humana para manejar variaciones de carga.
Monitoreo y alerta proactiva	≤ 1 minuto de detección	Detección automática de fallos antes que el usuario los perciba, usando alarmas y monitoreo sintético.
Trazabilidad completa	$100\,\%$ de operaciones trazadas	Toda operación sensible debe registrarse para cumplir con criterios de auditoría, cumplimiento y análisis post mortem.

Tabla 3.2: Requisitos no Funcionales

3. **Diseño de la propuesta**: Para el apartado del diseño se llevará a cabo la presentación de dos tipos de diagramas, entre los que se encuentran los diagramas de proceso y diagrama detallado de arquitectura en AWS lo cual permitirá realizar a alto nivel la validación de los flujos que corresponden en cada proceso, adicionalmente cabe destacar que esta propuesta de arquitectura se llevo a cabo en dos capas, una de ellas correspondiente a la entrega de contenido estático y la otra capa pertenece al nivel lógico de la aplicación.

   Diagrama de contexto – Flujo de entrega de contenido en el frontend

   La Figura 3.1 representa el flujo de alto nivel correspondiente al proceso de entrega de contenido en el frontend de la solución. Este diagrama de contexto describe cómo una solicitud realizada por el usuario viaja a través de los distintos componentes de la capa de presentación, la cual se encuentra orientada a ofrecer disponibilidad, velocidad de entrega y seguridad perimetral.

   Este proceso inicia con el usuario solicitando acceso a un recurso, el cual es resuelto por el sistema de nombres de dominio (DNS). Esta resolución direcciona la solicitud hacia la Red de Distribución de Contenido (CDN), encargada de entregar recursos estáticos de manera

eficiente desde ubicaciones cercanas al cliente final. La CDN intercambia datos con el cortafuegos de aplicaciones web, el cual filtra las solicitudes maliciosas antes de que lleguen a la infraestructura interna. Una vez es validada la solicitud, el contenido puede ser entregado directamente desde la CDN desde la caché, en caso de haber expirado la cache la petición es reenviada al servidor de origen. Este servidor obtiene los recursos desde la capa de almacenamiento. El flujo antes mencionado permite garantizar una respuesta rápida, segura y tolerante a picos de tráfico o fallos en regiones específicas.

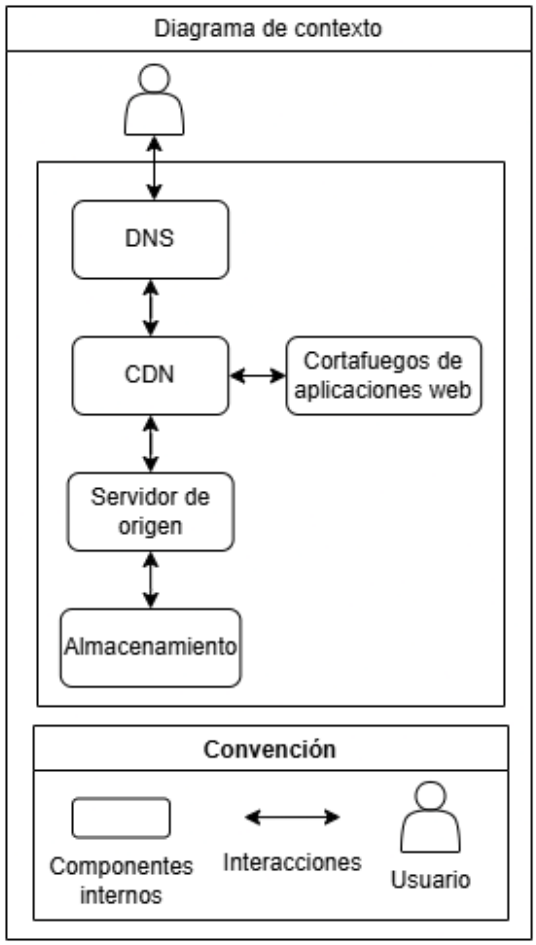


Figura 3.1: Diagrama de contexto – Entrega de contenido web
Fuente: Elaboración propia

Diagrama de contexto – Procesamiento backend asincrónico y multirregional

La Figura 3.2 representa el flujo del procesamiento de eventos dentro en el backend de la arquitectura propuesta, con énfasis en el desacoplamiento, la resiliencia y la tolerancia a fallos a través de múltiples regiones. Este diagrama de contexto permite visualizar cómo se

gestiona una solicitud desde su ingreso hasta el almacenamiento del resultado, siguiendo un enfoque asincrónico y distribuido.

El proceso inicia con la recepción de una solicitud, la cual es encolada en una infraestructura de mensajería que permite desacoplar cliente o sistema externo del módulo de procesamiento. Esta cola actúa como búfer temporal que permite mantener disponible el mensaje a procesar y así la continuidad del flujo, incluso si el módulo de procesamiento no está disponible de forma inmediata. Posteriormente, la solicitud es procesada por un componente lógico interno encargado de ejecutar las reglas de negocio. Una vez finalizada la ejecución, el resultado del proceso es almacenado de forma persistente. Este mismo flujo se encuentra replicado en una región secundaria, a la cual se redirige la solicitud automáticamente en caso de que la región primaria no se encuentre disponible para procesar. Esta replicación garantiza la continuidad del servicio y la protección ante desastres, manteniendo la integridad operativa sin intervención manual.

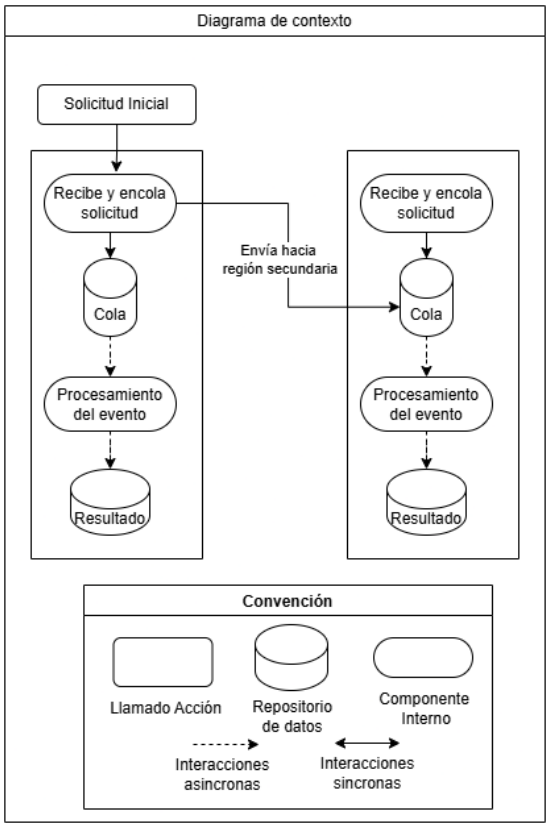


Figura 3.2: Diagrama de contexto – Procesamiento backend asincrónico con failover multirregional
Fuente: Elaboración propia

Diseño arquitectónico del frontend en AWS

La Figura 3.3 presenta la propuesta de diseño detallado de la capa de presentación implementado en la nube de AWS, orientado a la entrega de contenido estático con alta disponibilidad y protección a nivel perimetral. El tráfico de los usuarios es dirigido a través de Amazon Route 53, configurado con políticas de enrutamiento basadas en geolocalización y con monitoreo de estado habilitado, lo cual permitió identificar fallas en regiones y redirigir el tráfico de forma automática.

Posteriormente las solicitudes pasan a través de Amazon CloudFront, la cual actúa como red de distribución de contenido (CDN), que permitiendo la entrega global de archivos con baja latencia. El tráfico que viaja través de este flujo es protegido mediante AWS WAF, configurado para detectar y bloquear amenazas comunes como ataques de inyección o denegación de servicio a nivel de aplicación. Cabe mencionar que el contenido es servido desde Amazon S3, desplegado en una configuración multi-región. La observabilidad del sistema se encuentra reforzada mediante Amazon CloudWatch Synthetics, las cuales permiten emular interacciones de usuarios y monitorear endpoints críticos, junto con CloudWatch Alarms, que emitieron alertas basadas en fallos, latencia o códigos HTTP.

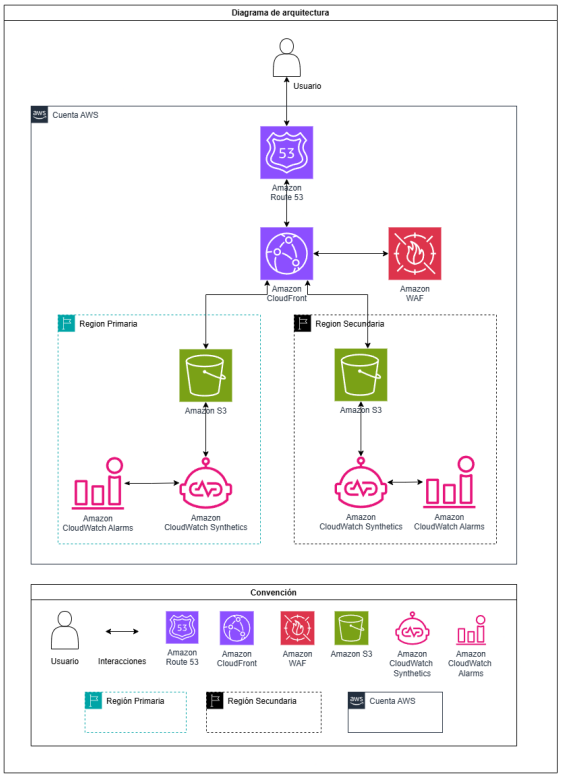


Figura 3.3: Arquitectura del frontend resiliente en AWS con CloudFront y WAF
Fuente: Elaboración propia

Diseño arquitectónico del backend en AWS

La Figura 4.12 presenta el diseño técnico detallado del backend en la nube de AWS, construido sobre servicios serverless y orientado a la gestión de eventos asincrónicos. La arquitectura se distribuye entre una región primaria y una región secundaria, ambas contenidas dentro de una misma cuenta de AWS, y preparadas para activarse en caso de falla detectada.

El proceso inicia desde la captura de solicitudes a través de Amazon API Gateway, que actúa como interfaz de entrada expuesta al cliente. Este servicio integró directamente con Amazon SNS, que se encarga de publicar los eventos recibidos hacia colas Amazon SQS, desacoplando el flujo y asegurando la durabilidad de los mensajes. Los mensajes almacenados en SQS son procesados por funciones AWS Lambda, las cuales ejecutan la lógica de negocio. Los resultados de dichos procesos son almacenados en tablas Amazon DynamoDB, distribuidas por función (por ejemplo, "idempotencia" y "proceso"). Se implementó replicación asincrónica entre las tablas de DynamoDB en ambas regiones, con el fin de garantizar la continuidad operativa sin pérdida de datos. En caso de que la región primaria presente algún tipo de falla el servicio de Amazon Route 53, en conjunto con mecanismos de health check, redirige automáticamente el tráfico a la región secundaria. Esta arquitectura multirregional, orientada al desacoplamiento, permite cumplir con objetivos clave de recuperación (RTO y RPO) y mejorar la resiliencia del sistema.

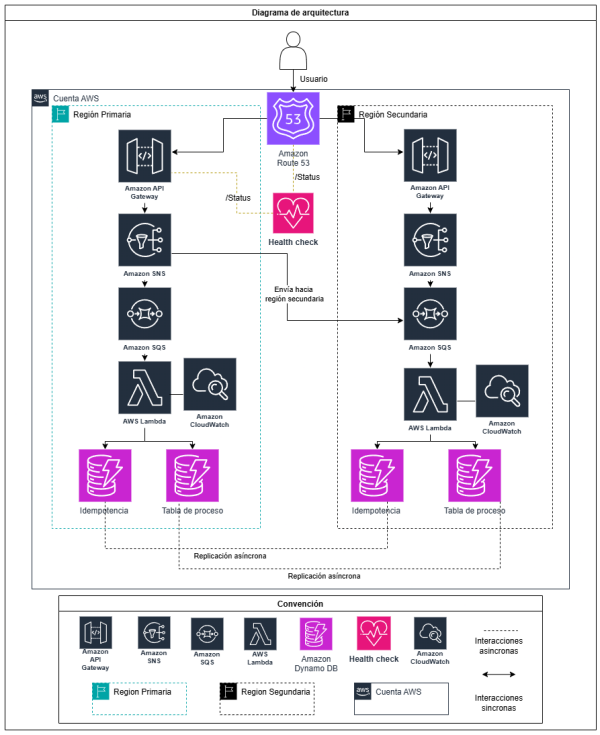


Figura 3.4: Arquitectura del backend multirregional con procesamiento desacoplado en AWS
Fuente: Elaboración propia

4. **Implementación de la propuesta**: La implementación de la arquitectura resiliente en la nube propuesta se llevó a cabo utilizando servicios nativos de Amazon Web Services (AWS), siguiendo lo planteado en cada uno de los diagramas de contexto. Esta implementación permitió materializar los principios del Well-Architected Framework, aplicando buenas prácticas de diseño orientadas a la resiliencia, la disponibilidad, el desacoplamiento y la observabilidad.

   La infraestructura se desplegó en dos regiones de AWS, configuradas bajo un enfoque activo-pasivo, con activación dinámica de la región secundaria ante fallos en la primaria. A través de mecanismos de monitoreo y failover automático, el sistema fue preparado para responder de forma proactiva a incidentes, garantizando continuidad operativa.

   Capas lógico-funcional
   La implementación se organizó según el Diagrma de capas presentado en la Figura 3.5, lo cual permitió distribuir responsabilidades funcionales de manera clara y escalable.

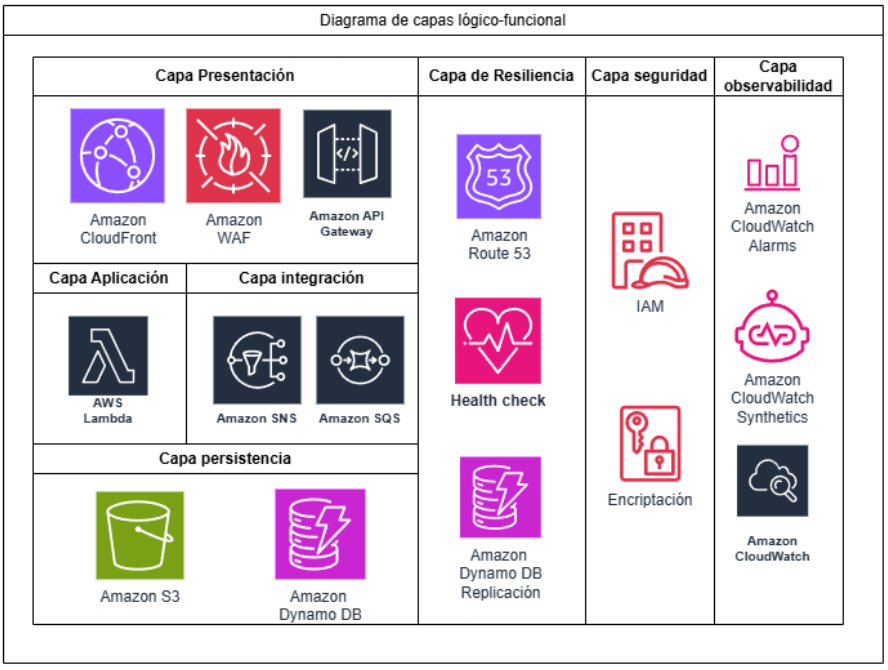


Figura 3.5: Diagrama de capas lógico-funcional
Fuente: Elaboración propia

- **Capa de presentación** En esta capa se configuró Amazon CloudFront como red de distribución de contenido (CDN), habilitando múltiples edge locations para entregar contenido estático almacenado en Amazon S3 con baja latencia. Para reforzar la seguridad, se integró AWS WAF con reglas personalizadas que protegieron la capa de borde contra amenazas comunes (SQLi, XSS, DoS).
- **Capa de aplicación** En esta capa se implementaron funciones AWS Lambda, que per-

miten procesar eventos de manera nativa en serverless, escalable y aislada. Las funciones Lambda fueron activadas a partir de mensajes encolados, permitiendo un diseño desacoplado y tolerante a fallos.

- **Capa de integración** En esta capa se implementaron Amazon SNS para distribuir eventos y Amazon SQS como mecanismo de almacenamiento temporal y desacoplamiento entre componentes. Esto permitió garantizar la entrega de mensajes, incluso cuando los consumidores no estaban disponibles inmediatamente.
- **Capa de persistencia** En esta capa los datos procesados son almacenados en Amazon DynamoDB, configurado con replicación entre regiones mediante Global Tables. Esto permite mantener sincronía entre las tablas de la región primaria y secundaria, asegurando consistencia eventual y alta disponibilidad. Para contenido estático, se utilizaron buckets de Amazon S3 con replicación cruzada activada.
- **Capa de resiliencia** En esta capa se encuentra configurado el servicio de Amazon Route 53 con políticas de enrutamiento Healt Check, integradas a checks automatizados. Esto permitió que, ante una falla en los endpoints de la región primaria, el tráfico fuera redirigido hacia la región secundaria de manera automática.
- **Capa de seguridad** En esta capa se llevó a cabo la implementación de políticas de acceso mediante AWS IAM, asegurando la autenticación y autorización basadas en roles. Además, se aplicó cifrado en reposo en todos los recursos (S3, DynamoDB, SQS).
- **Capa de observabilidad** En esta capa la observabilidad se logró mediante la integración de Amazon CloudWatch, configurando logs, métricas personalizadas y dashboards. También se empleó CloudWatch Alarms para detectar umbrales críticos, y CloudWatch Synthetics para simular interacciones de usuario y verificar la disponibilidad de servicios.

Patrones Implementados

- **Patrón Multi-Región Activo-Pasivo**

  Objetivo:

  Permitir la disponibilidad continua del sistema ante fallos a nivel regional, permitiendo una recuperación automática, rápida y sin pérdida de datos significativa, cumpliendo con los objetivos de RTO y RPO definidos en el diseño.

  Justificación de selección:

  Este patron fue elegido como pilar fundamental de resiliencia en la propues de arquitectura a nivel de la infraestructura global; teniendo en cuenta que representa una de las mejores práctica para lograr la continuidad operativa ante desastres con el objetivo de no tener periodos de inactividad muy largos por errores en una única región.

- **Patrón Serverless Multi-Región**

  Objetivo:

  Permitir la ejecución de flujos de negocio críticos mediante una arquitectura nativamente resiliente, sin tener que provisionar servidores, asegurando una respuesta automática a

fallos regionales o funcionales, así como una alta tolerancia a las duplicaciones, caídas parciales o los reintentos.

Justificación de selección:

Este patrón fue elegido por las capacidades de resolver la resiliencia frente a los diferentes desafíos en la capa de procesamiento lógico y de eventos. Al ser un patrón que no necesita instancias o servidores para activarse, el riesgo operativo se ve disminuido y podemos permitir un escalado automático de procesos frente a nuevos incrementos de la carga de trabajo sin que se degrade el rendimiento.

- **Patrón de Capas Funcionales**

  Objetivo:

  Permitir mantener una claridad arquitectónica e independencia entre módulos, permitiendo la asignación de únicas responsabilidades a cada módulo de la aplicación, visibilidad transversal y capacidad de evolución tecnológica del sistema sin afectar demás funcionalidades.

  Justificación de selección:

  Este patrón fue elegido para proporcionar una estructura lógica y escalabilidad organizacional a la arquitectura de la referencia. Este patrón no sólo da sentido a los aspectos técnicos que cubre, sino que también permite facilitar el trabajo en grupo, la gestión de los cambios e ir entregando porciones de la arquitectura por equipos especializados. Su utilidad se encuentra, además, en su capacidad de aplicarse en diversos contextos, como la de no renunciar a la gobernanza.

Patrones arquitectónicos vs Atributos de calidad vs Pilares del WAF

Cada uno de los patrones antes mencionados no sólo aportan en solventar el problema técnico de resiliencia en la nube, sino que también se alinean de forma directa con cada uno de los pilares del AWS Well-Architected Framework priorizados en este proyecto: fiabilidad, rendimiento y excelencia operativa. A continuación, se muestra cómo estos patrones no son aislados, sino componentes fundamentales de una arquitectura de referencia resiliente.

Patrón arquitectónico	Atributos de calidad	Pilar WAF
Multi-región activo-pasivo	- Fiabilidad - Disponibilidad - Rapidez de recuperación	- Fiabilidad - Excelencia operativa
Arquitectura serverless multi-región	- Fiabilidad - Disponibilidad - Robustez - Rapidez de recuperación - Seguridad	- Fiabilidad - Rendimiento - Excelencia operativa
Patrón de capas funcionales	- Robustez - Seguridad - Mantenibilidad	- Excelencia operativa - Rendimiento

Tabla 3.3: Relación entre patrones arquitectónicos, atributos de calidad y pilares del WAF

Criterios de selección de servicios AWS

La selección de los servicios de Amazon Web Services (AWS) se realizó siguiendo criterios técnicos orientados a garantizar resiliencia, escalabilidad y simplicidad operativa. Se priorizaron servicios administrados por el proveedor de nube y desacoplados, con capacidad de recuperación automática y bajo acoplamiento entre componentes. Esta elección buscó reducir la complejidad de operación, mejorar el tiempo de respuesta ante fallos y facilitar la replicación regional.

A continuación, se listan cada uno de los criterios utilizados para la selección de los servicios:

- **Escalabilidad automática** Prioridad por servicios que crecen o disminuyen su capacidad en función de la demanda.
- **Desacoplamiento funcional** Prioridad por servicios independientes que puedan fallar o reiniciarse sin afectar al sistema completo.
- **Alta disponibilidad y replicación** Prioridad por servicios que soportan replicación entre zonas o regiones y garantizan continuidad.

- **Procesamiento asíncrono y resiliente** Prioridad por servicios que cuenten con la capacidad de recibir, almacenar y procesar eventos sin perder información ni depender del orden de llegada.
- **Failover y balanceo automático** Prioridad por servicios que soporten reruteo de tráfico y activación de entornos secundarios sin intervención manual.
- **Minimización de la complejidad operativa** Prioridad por servicios administrados para evitar la necesidad de gestionar infraestructura manualmente.
- **Integración nativa con herramientas de monitoreo y trazabilidad** Prioridad por servicios compatibles con CloudWatch, X-Ray y otras herramientas de observabilidad.

Servicios implementados vs atributos de calidad

A continuación, se puede visualizar el cómo corresponden cada uno de los servicios con los atributos de calidad implementados en la propuesta de diseño de arquitectura de referencia, cabe destacar que estos comparten la principal característica de ser servicios Cloud Native o nativos de nube que permiten apuntar a procesos desacoplados en nube.

Componente AWS	**Fiabilidad**	**Disponibilidad**	**Seguridad**	**Robustez**	**Rapidez de recuperación**
Amazon route 53	✓	✓			✓
Amazon cloudFront	✓	✓	✓	✓	
Amazon api Gateway	✓	✓	✓	✓	
AWS lambda	✓	✓	✓	✓	✓
Amazon sqs	✓	✓		✓	✓
Amazon sns	✓	✓		✓	✓
Amazon dynamodb	✓	✓	✓	✓	✓
Amazon s3	✓	✓	✓	✓	✓
AWS waf			✓		
AWS iam			✓		
Cloudwatch / Logs	✓		✓	✓	✓
Cloudwatch synthetics	✓	✓			✓

Tabla 3.4: Servicios implementados vs atributos de calidad

3.3. Resumen del capítulo

En este capítulo se llevó a cabo el análisis y desarrollo necesario para dar cumplimiento a los objetivos específicos del proyecto, partiendo así con el análisis de atributos de calidad directamente relacionados con la resiliencia, para esto se expandió un poco más lo ya documentado en el marco teórico, con un especial enfoque en la búsqueda de los subatributos de calidad que permitieran capturar con mayor precisión los requisitos funcionales y los no funcionales.

En este capítulo también se podrán encontrar listados los requisitos funcionales y los no funcionales, gracias a estos requisitos se pudo llevar a cabo el planteamiento de la propuesta de diseño para la arquitectura de nube, para presentar esta propuesta se llevo acabo el planteamiento de dos tipos de diagramas, el primero fue de contexto el cual permite presentar a alto nivel el flujo que lleva cada una de las capas del sistema, así como también se pueden evidenciar los diagramas detallados de la nube de AWS, en los que se pueden ya ver los servicios que se implementaron con propósito de materializar la propuesta de diseño presentada anteriormente.

Una vez presentado el diseño también se sintetizaron en otro diagrama las tecnologías utilizadas en la propuesta, esto con el fin de identificar cada uno de los servicios que pertenecen a estas capas, así como también su responsabilidad dentro del diseño de la arquitectura resiliente de referencia en la nube de AWS. Finalmente se documentaron los patrones que fueron implementados en este diseño, así como también el porqué fueron estos elegidos y esto es también sustentado con matrices donde se identifica la relación que tiene cada atributo, con los patrones y adicionalmente con las practicas del WAF.

Capítulo 4

Evaluación

4.1. Diseño de la evaluación

Para llevar a cabo el proceso de validación de la propuesta de arquitectura desarrollada, se realizó un grupo de pruebas controladas que permitieron simular el funcionamiento real, sobrecarga y fallos. Estas pruebas fueron creadas para demostrar que se han cumplido los atributos de calidad establecidos para el proyecto como los son la fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación y para comprobar que la solución respondía adecuadamente frente a los escenarios definidos en los objetivos específicos.

La evaluación se dividió en dos partes; las pruebas sobre el frontend, donde se probó la capa de entrega de contenido estático, y pruebas sobre el backend, donde se validó la lógica de negocio distribuida, la resiliencia en el procesamiento de eventos y la continuidad operativa multirregional.

1. **Pruebas en el Frontend**

   Para esta capa de la arquitectura se diseñaron dos tipos de pruebas clave:

   - **Prueba de rendimiento o carga con Artillery**

     Esta prueba se realiza con el fin de medir la capacidad de respuesta del sistema ante distintos niveles de carga, analizando métricas como tiempos de respuesta (P95 y P99) estabilidad del endpoint bajo diferentes cargas de tráfico y picos.

     Configuración de la prueba

     El archivo de configuración usado en Artillery incluyó tres fases: calentamiento, incremento progresivo de carga, y un pico, para observar el comportamiento tanto bajo tráfico sostenido como ante incrementos bruscos.

```
target: https://pocproyecto.click
phases:
  - duration: 60       # 60 segundos
    arrivalRate: 1     # 1 usuario por segundo
    rampTo: 2
    name: Warm up phase
  - duration: 60
    arrivalRate: 2
    rampTo: 4
    name: Ramp up load
```

```
  - duration: 30
    arrivalRate: 10
    rampTo: 30
    name: Spike pase
```

Listing 4.1: Configuración de prueba con Artillery

- **Simulación de fallo en región principal (S3 Multi-AZ)**

  Esta prueba se realizó con el fin de confirmar la configuración de conmutación por error en Amazon Route 53. El origen fue restringido intencionalmente en la primera región de disponibilidad, por lo que el sistema redirigió el tráfico a la siguiente región disponible. Esta prueba buscó validar la disponibilidad continua y la conmutación dinámica entre regiones.

  Configuración de la prueba

  En el apartado de permisos de acceso del servicio de bucket S3 se Modificó el archivo que declara los permisos a los recursos de este servicio, así que en este caso se realizó la denegación de la siguiente forma, simulando que el cloudfront no pueda recuperar el contenido estático.

```
{
  "Effect": "Deny",
  "Principal": {
    "Service": "cloudfront.amazonaws.com"
  },
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::<bucket-name>/*",
  "Condition": {
    "StringEquals": {
      "AWS:SourceArn": "arn:aws:cloudfront::<AWS_ACCOUNT_ID>:
          distribution/<DISTRIBUTION_ID>"
    }
  }
}
```

Listing 4.2: Política S3 para denegar acceso desde CloudFront

2. **Pruebas en el Backend** Para esta capa, se diseñaron tres pruebas específicas orientadas a validar la operación distribuida, el control de duplicación, y la recuperación automática:

   - **Prueba de idempotencia**

     Para esta prueba se realizó una solicitud con un `request_token` como identificador de token y la misma solicitud se repitió intencionadamente. La verificación confirmó que el sistema manejó la solicitud sólo una vez, y por lo tanto confirmó el control de eliminación requerido en sistemas distribuidos.

Configuración de la prueba

A continuación se muestra el payload o cuerpo de la petición con el JSON el cual contiene parámetros de un proceso de creación de una cuenta en un servicio, donde cabe destacar la importancia del atributo `request_token` el cual permite validar si esta solicitud ya fue procesada.

```
{
    "request_token": "384b17fb-0437-45fc-b242-7d678b60ae9c",
    "user_id": "JMP123",
    "customer_first_name": "jose123",
    "customer_last_name": "Moreno",
    "account_type": "usuario",
    "comment": "Comentario",
    "beneficiaries": [
        {
            "name": "Pedro",
            "percentage": 50
        },
        {
            "name": "Pablo",
            "percentage": 50
        }
    ],
    "suitability": {
        "liquidity": "Bajo",
        "time_horizon": "20 years",
        "risk_tolerance": "Alto"
    },
    "instructions": {
        "dividends": "Reinvertit"
    }
}
```

Listing 4.3: Ejemplo de payload enviado para prueba de idempotencia

- **Prueba de reintentos con colas (SQS)**

  Para esta prueba se realizó una interrupción simulada del consumidor de la cola, lo que ayudó a ver cómo se comportaba el sistema en caso de una falla temporal. Después de que se reanudó el consumidor, se demostró que los mensajes se reprocesaron sin pérdida, lo que significa que la configuración de reintento y la tolerancia a fallos estaban funcionando.

- **Prueba de salto de región (failover con Route 53)**

  Para esta prueba se simularon fallos en los puntos finales de la región primaria, activando el mecanismo de salud establecido en Route 53. El tráfico se redirigió automáticamente a la región alternativa, lo que demostró que el sistema es de naturaleza autorreparable.

Todo el proceso de pruebas ha sido rastreado y respaldado por evidencia tomada de servicios como CloudWatch, WAF y Artillery, con el fin de obtener métricas reales y confirmar que el comportamiento técnico de la arquitectura fue el esperado en la peor situación.

4.2. Resultados de la evaluación

Pruebas en el FrontEnd

Antes de iniciar con el proceso de análisis de los resultados se llevará a cabo la presentación de los recursos utilizados para el proceso de evaluación en la capa del FrontEnd, donde se destacan las dos regiones utilizadas, así como las pilas o stacks que se encuentran disponibles en cada una de ellas.

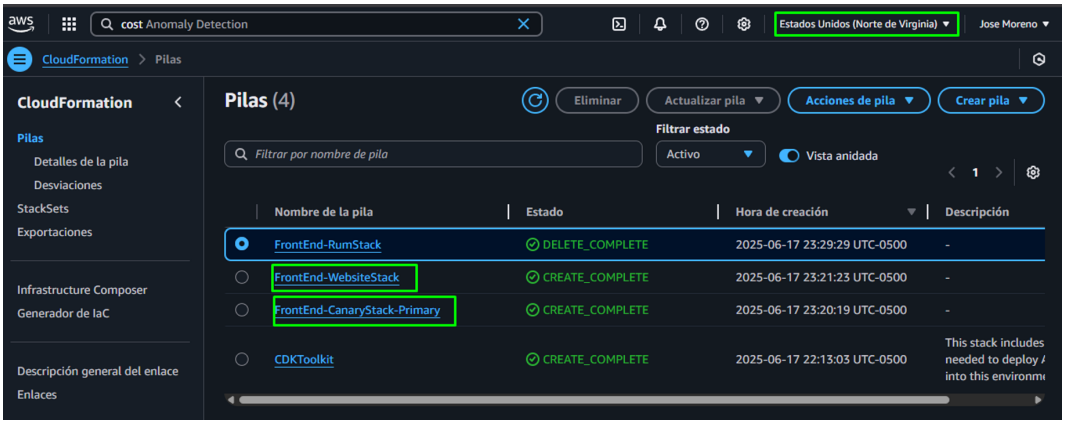


Figura 4.1: Stack o pilas región Principal (us-east-1)
Fuente: Elaboración propia

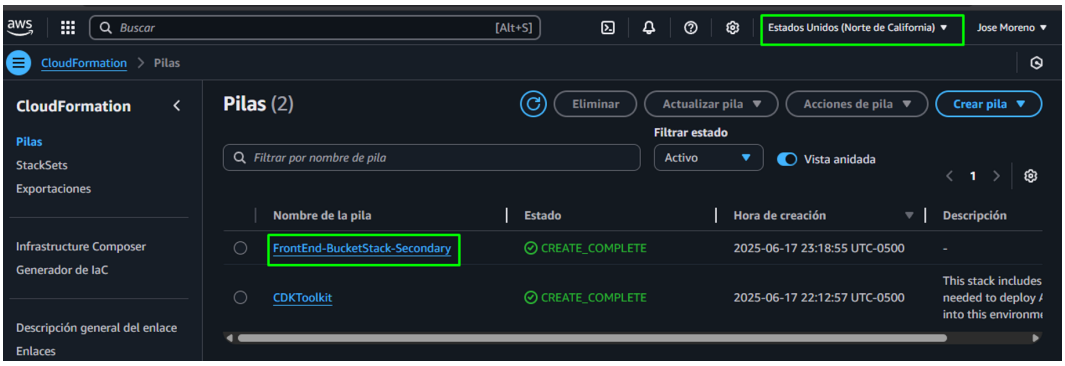


Figura 4.2: Stack o pilas región Secundaria (us-west-1)
Fuente: Elaboración propia

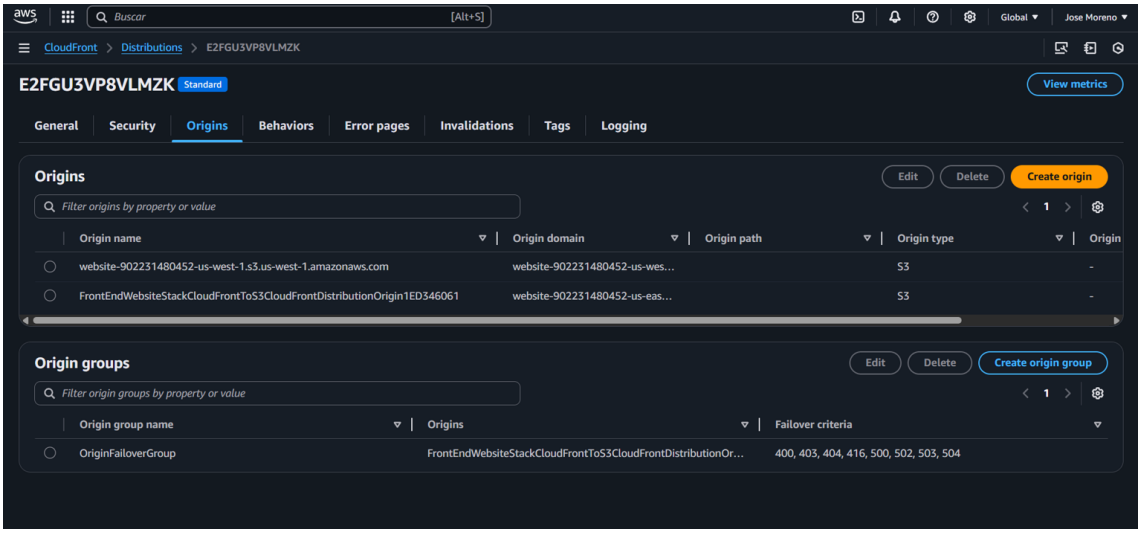


Figura 4.3: Distribución de CloudFront
Fuente: Elaboración propia

- **Prueba de rendimiento o carga con Artillery**

  Escenario evaluado:

  - Total de solicitudes: 174.000
  - Duración aproximada de la prueba: 15 minutos
  - Endpoint evaluado: /
  - Tipos de respuesta: HTTP 200 (OK) y HTTP 403 (Acceso denegado)

  Validaciones implementadas:

  - 200 iteraciones de solicitudes GET al endpoint raíz (/)
  - Umbrales de validación (ensure) definidos:
    - http.response_time.p99 $\leq$ 100 ms
    - http.response_time.p95 $\leq$ 75 ms
  - Medición de satisfacción del usuario con índice Apdex, usando un umbral de respuesta de 100 ms

  A continuación, podemos ver los resultados de la prueba ejecutado con artillery el cual nos permite visualizar que el proyecto cumple con las validaciones implementadas.

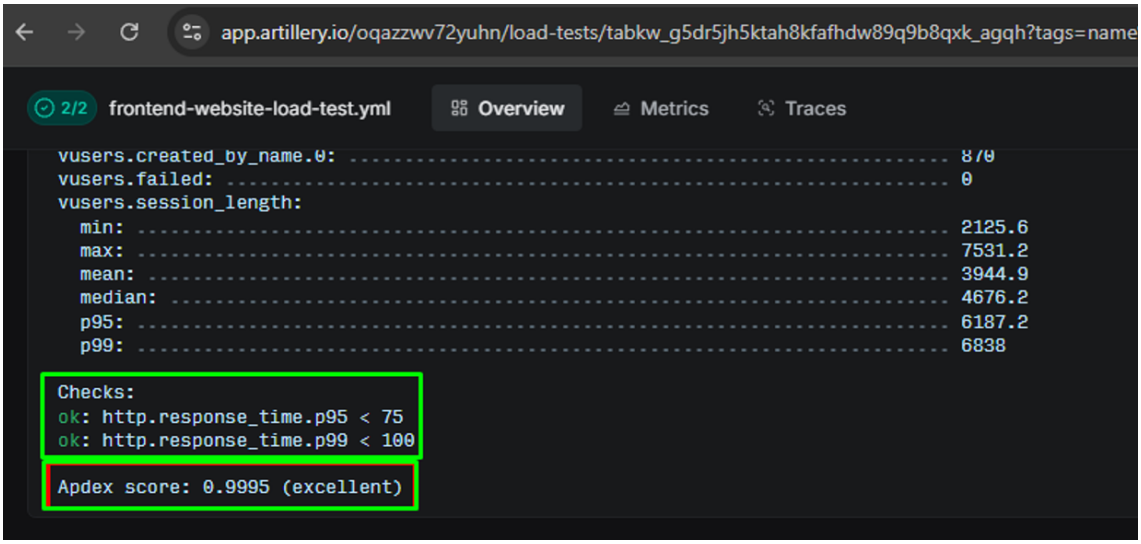


Figura 4.4: Resultados de la prueba en artillery
Fuente: Elaboración propia

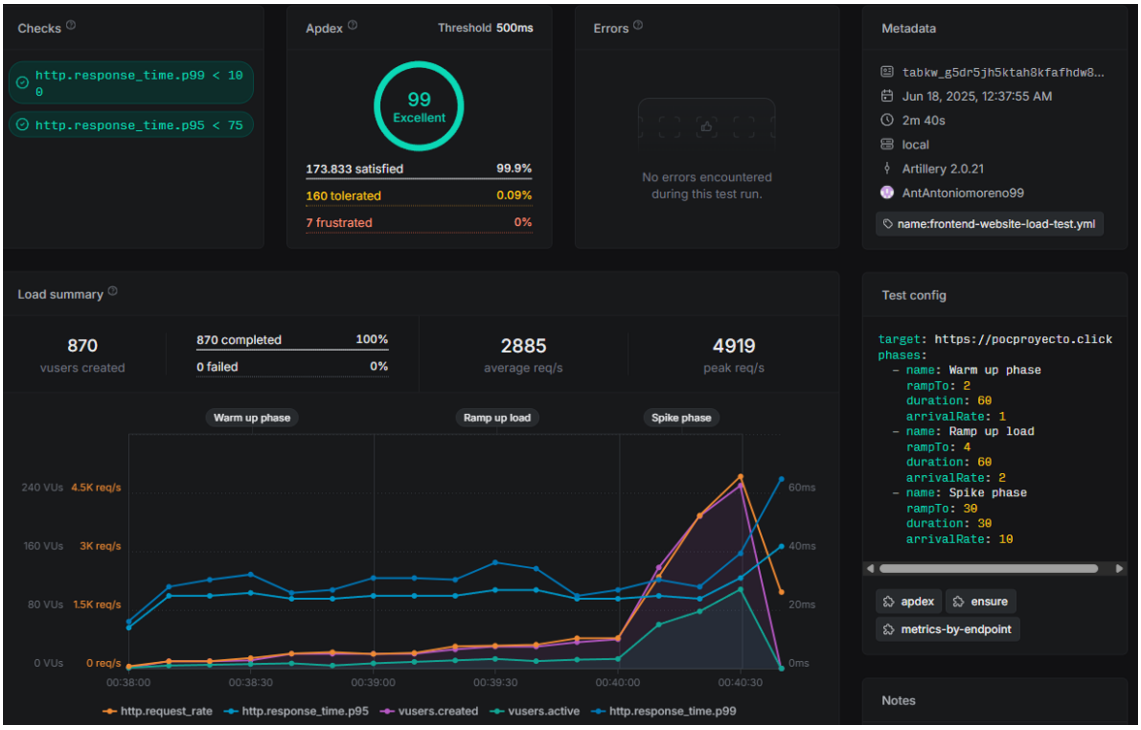


Figura 4.5: Panel general de artillery
Fuente: Elaboración propia

Ahora verificando con los datos del WAF en AWS, se puede validar la cantidad de solicitudes que fueron bloqueadas ver Figura 4.6 por la regla implementada por este servicio, esto con el fin de mantener el buen funcionamiento del servicio.

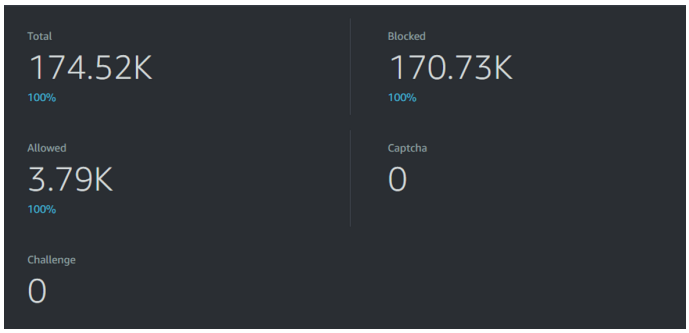


Figura 4.6: Estadísticas del WAF
Fuente: Elaboración propia

Se puede evidenciar que hay muchas peticiones que fueron bloqueadas porque el servicio de WAF las clasifica como bot's o posibles bot's al tener el mimo origen Figura 4.7.

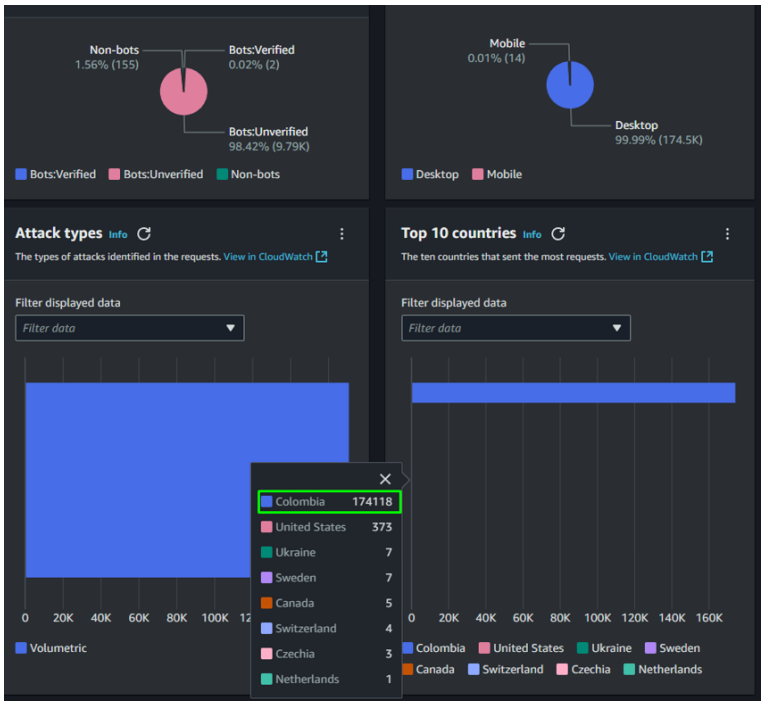


Figura 4.7: Panel general del WAF
Fuente: Elaboración propia

- **Simulación de fallo en región principal (S3 Multi-AZ)**

  Para esta prueba, como se mencionó en el diseño de la prueba se realizó la simulación de un fallo del servidor de origen en la región principal, es decir, se modifico el permiso de acceso del CloudFront, para validar que no se pudiera entregar el contenido de esta primera región, para poder visualizar esa parte podemos encontrar el registro en CloudWatch-Synthetics Figura 4.8, en esta podemos evidenciar que la entrega de contenido en esta región estuvo abajo o sin respuesta.

Figura 4.8: Panel general del CloudWatch-Synthetics
Fuente: Elaboración propia

Sitios disponibles

A continuación, podremos evidenciar los dos contenidos estáticos que son entregadas por cada una de las regiones, en este caso en la Figura 4.9 en el lado izquierdo se encuentra el contenido estático alojado en la región us-east-1, la cual es la principal, pero una vez simulamos el error de acceso al S3 donde se encuentra dicho contenido, podemos ver que inmediatamente el CloudFront empieza a servir el contenido alojado en la región secundaria la cual está en us-west-1 y es la que se puede evidenciar al lado derecho de la figura.

Figura 4.9: Sitios disponibles
Fuente: Elaboración propia

Pruebas en el BackEnd

Antes de iniciar con el proceso de análisis de los resultados se llevará a cabo la presentación de los recursos utilizados para el proceso de evaluación en la capa del BackEnd, donde se destacan las dos regiones utilizadas, así como las pilas o stacks que se encuentran disponibles en cada una de ellas.

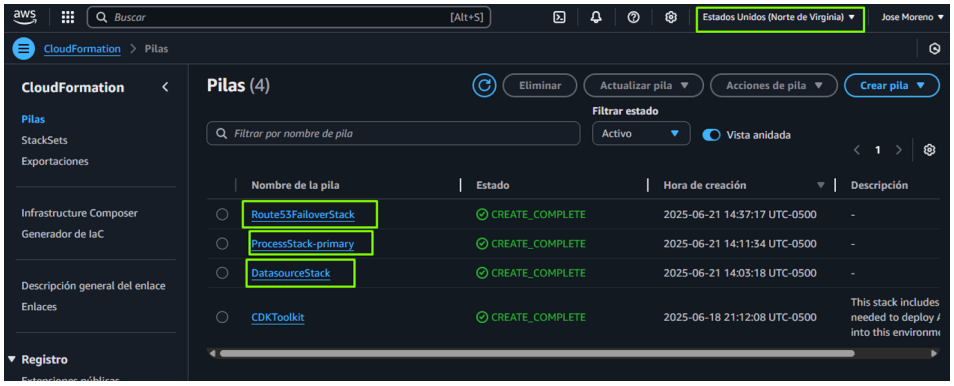


Figura 4.10: Stack o pilas región Principal (us-east-1)
Fuente: Elaboración propia

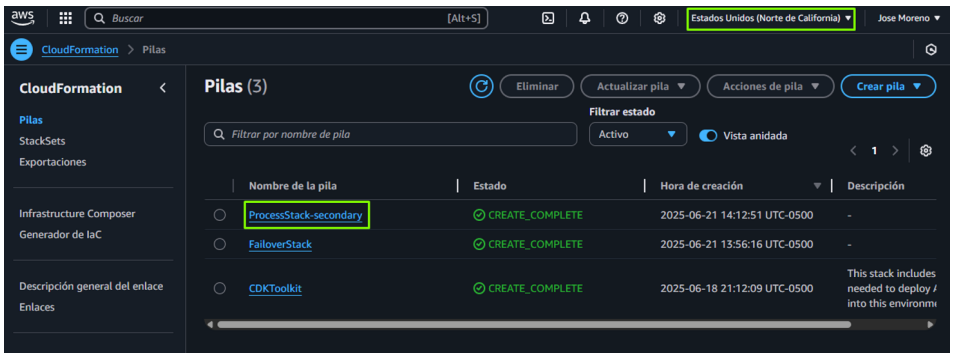


Figura 4.11: Stack o pilas región Secundaria (us-west-1)
Fuente: Elaboración propia

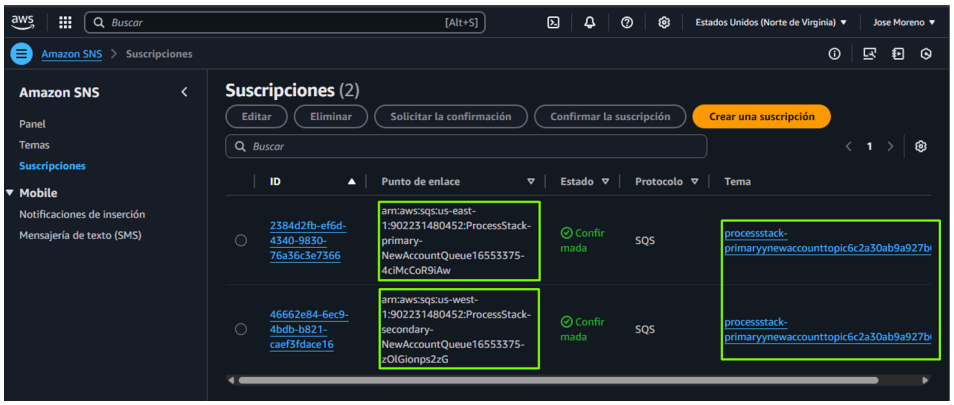


Figura 4.12: Colas SQS
Fuente: Elaboración propia

- **Prueba de idempotencia**

  Para esta prueba se realizó él envió de una solicitud de tipo POST el cual en su cuerpo contenía `request_token` único como identificador de la petición. Posteriormente, se envió la misma solicitud intencionadamente utilizando exactamente el mismo token. Esto con el objetivo de verificar que el sistema detectara la duplicación y ejecutara la lógica de negocio una sola vez, asegurando así que no se generaran duplicaciones en el procesamiento.

  A continuación, podemos ver los logs en CloudWatch Figura 4.13 donde se puede evidenciar que esta primera petición es notificada a través de los servicios de colas y a su vez es procesada por la lambda la cual contiene la lógica de negocio y posteriormente realiza el registro en la base de datos.

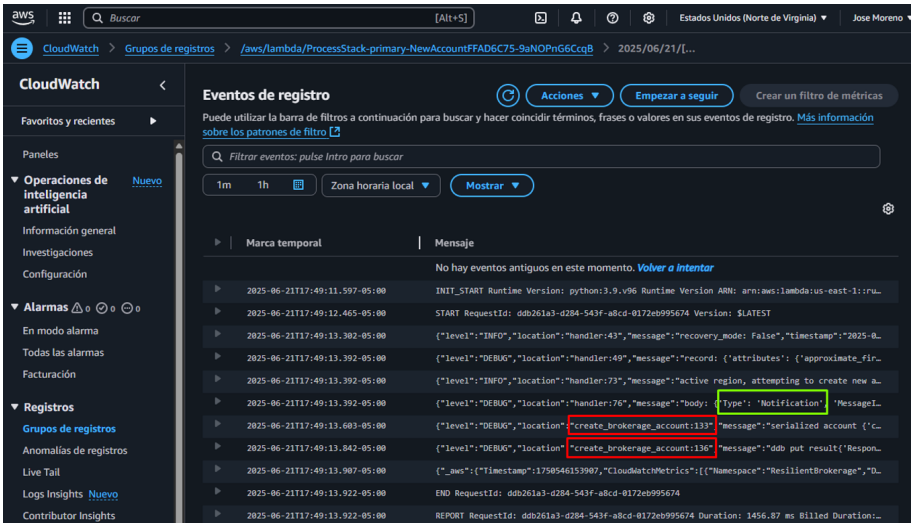


Figura 4.13: Registros en CloudWatch intento N°1
Fuente: Elaboración propia

Luego al realizar esta petición por segunda vez con el mismo `request_token` podemos validar en los logs Figura 4.14 de que este es notificado a través de las colas de mensajería, pero no se evidencian que se creen los registros, es decir, que no son procesados por la lambda

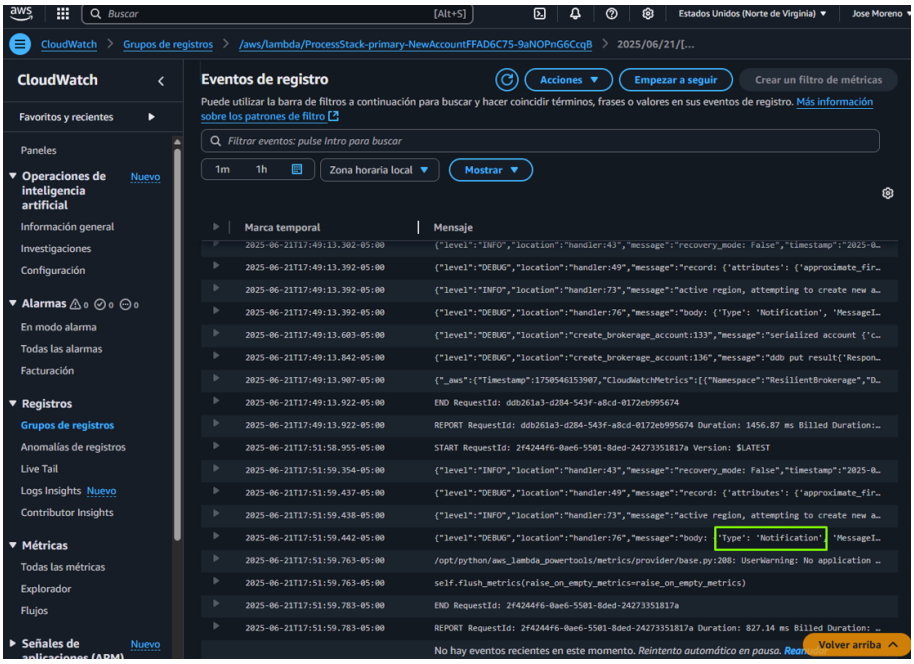


Figura 4.14: Registros en CloudWatch intento N°2
Fuente: Elaboración propia

A continuación , podremos validar en cada una de las tablas de proceso Figura 4.15 que sólo se encuentra un registro dado que la implementación de la idempotencia, adicionalmente también podemos ver el registro en la tabla de idempotencia Figura 4.16 que se lleva a cabo el seguimiento de los `request_token`, lo que permite que sea procesado una única vez un registro, también cabe destacar que en la Figura 4.17 y Figura 4.18 se encontrarán la replicación de estos registros y esto se debe a la configuración de las tablas de dynamo que permiten la replicación automática manteniendo así la consistencia de los datos.

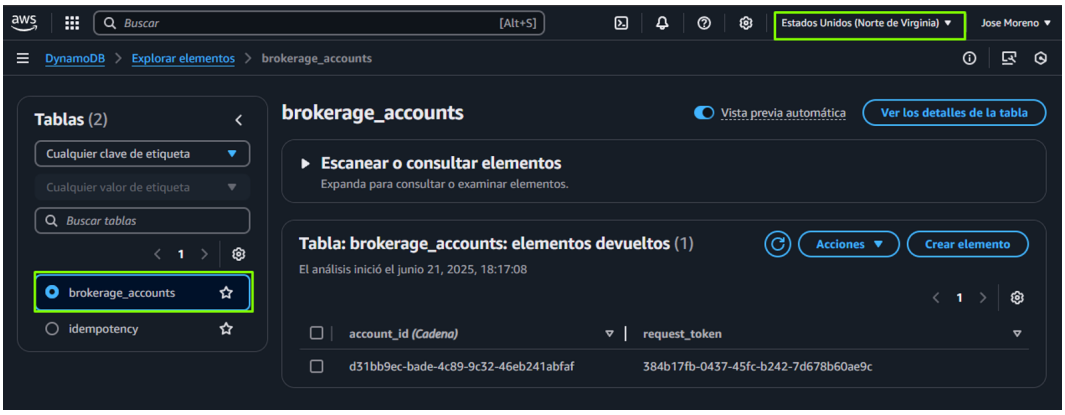


Figura 4.15: Registros en tabla de proceso Dynamo (us-east-1)
Fuente: Elaboración propia

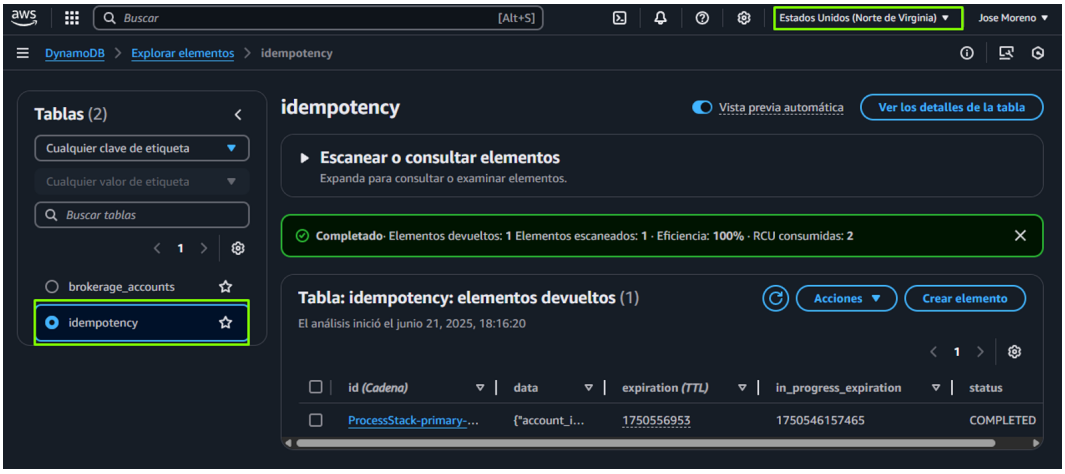


Figura 4.16: Registros en tabla de idempotencia Dynamo (us-east-1)
Fuente: Elaboración propia

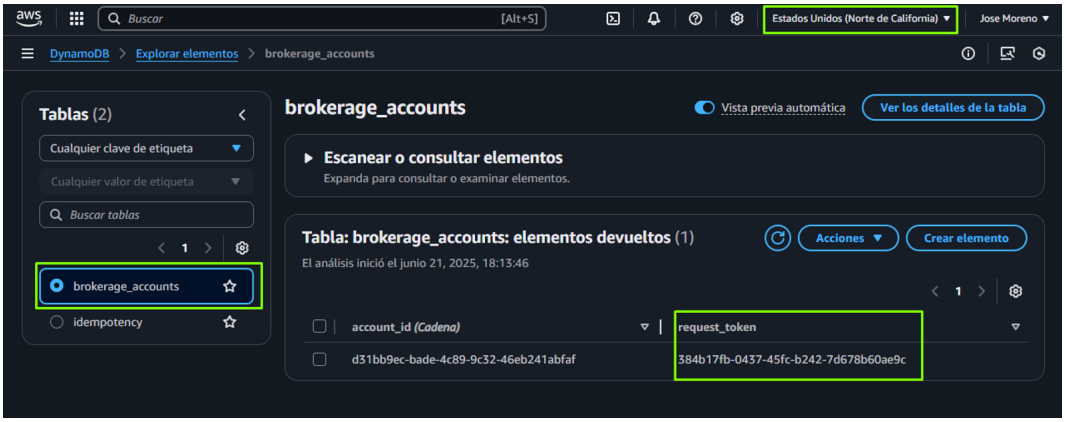


Figura 4.17: Registros en tabla de proceso Dynamo (us-west-1)
Fuente: Elaboración propia

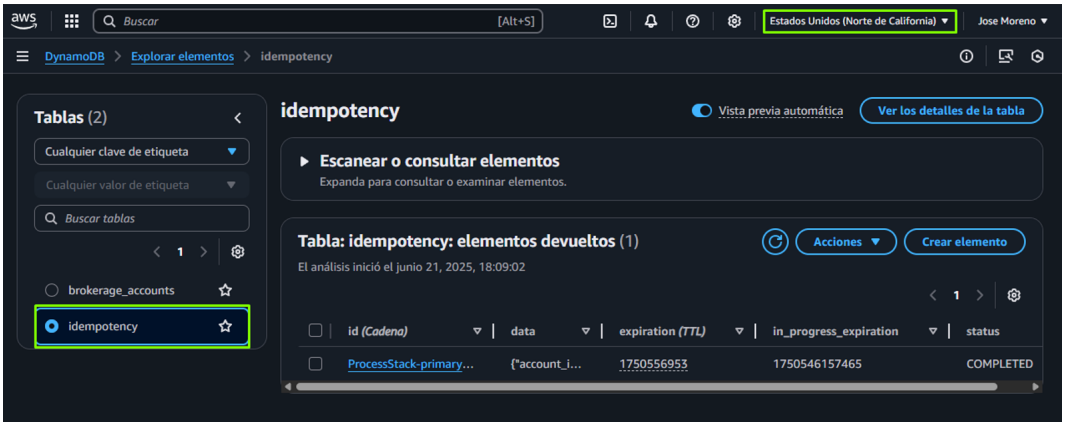


Figura 4.18: Registros en tabla de idempotencia Dynamo (us-west-1)
Fuente: Elaboración propia

- **Prueba de reintentos con colas (SQS)**

  Para llevar acabo esta prueba lo que se realizó es inyectar un error o una excepción en el código fuente de la región principal el cual se ejecuta en la lambda, la cual juega el papel de receptor del mensaje enviado por la cola, es así como podemos validar que el mensaje al no ser entregado permanece en la cola a la espera de que el receptor esté de nuevo en línea y así poder entregar el mensaje manteniendo así la confiabilidad del sistema evitando la pérdida de mensajes o procesos por culpa de un error presentado en la lógica de negocio.

Este punto se puede evidenciar en la Figura 4.19 que se realizan 4 peticiones las cuales se cargan a las colas, pero también se puede evidenciar en la Figura 4.20 que estas permanecen disponibles para entregar por más tiempo en las colas Figura 4.21 dado que la lambda receptora está presentado un inconveniente como se evidencia en los primeros registros de CloudWatch Figura 4.22, los cuales son excepciones generadas por el receptor. Luego de que el proceso dentro de la lambda es corregido y desplegado de nuevo se vuelve y se consulta la SQS Figura 4.23 y se encuentra de que estos han sido ya entregados y posteriormente procesados.

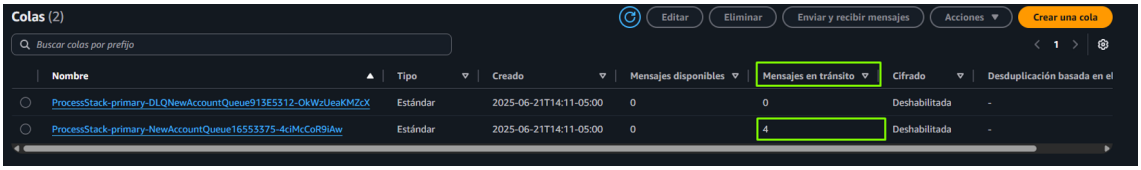


Figura 4.19: Mensajes en transito en SQS (AWS)
Fuente: Elaboración propia

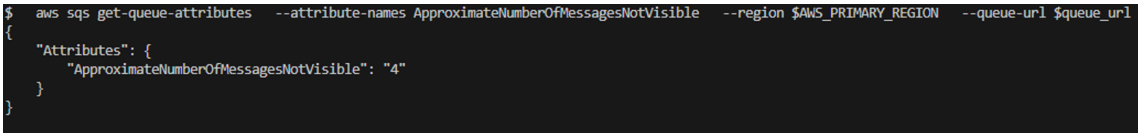


Figura 4.20: Mensajes en transito en la SQS (cmd)
Fuente: Elaboración propia

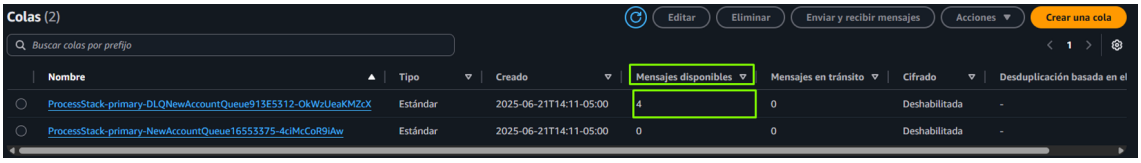


Figura 4.21: Mensajes en disponibles en la DLQ (AWS)
Fuente: Elaboración propia

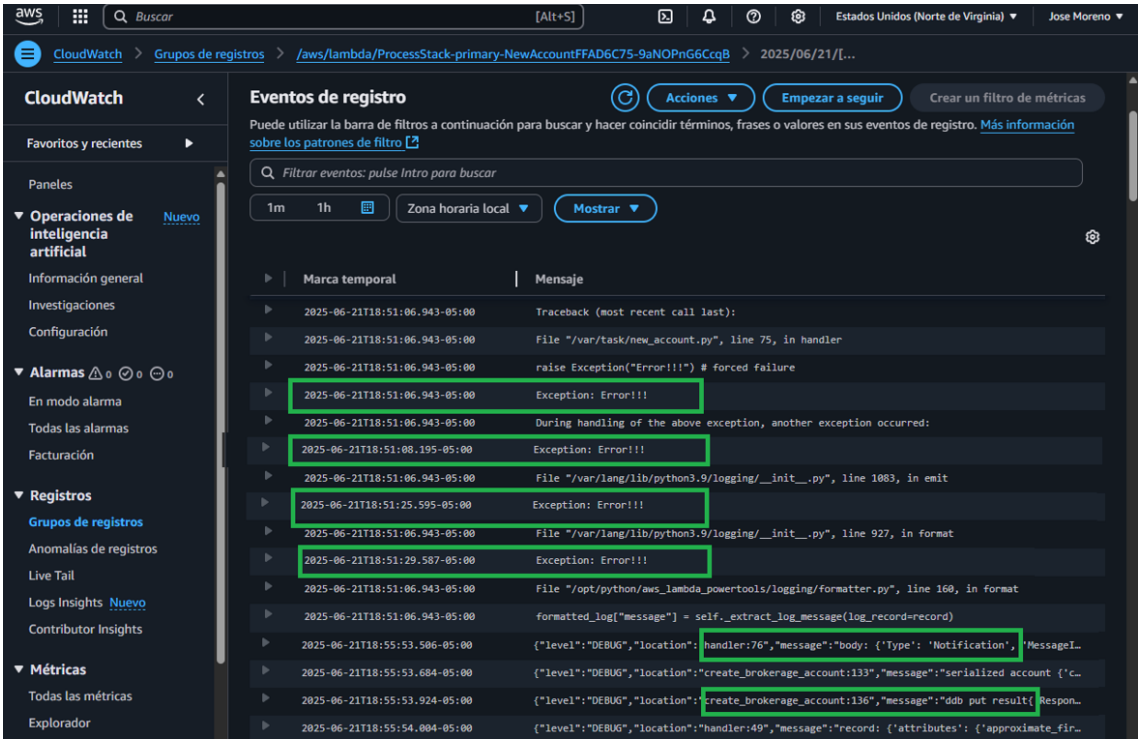


Figura 4.22: Registros en CloudWatch intentos SQS
Fuente: Elaboración propia

```
$ aws sqs get-queue-attributes --attribute-names ApproximateNumberOfMessagesNotVisible --region $AWS_PRIMARY_REGION --queue-url $queue_url
{
    "Attributes": {
        "ApproximateNumberOfMessagesNotVisible": "0"
    }
}
```

Figura 4.23: Mensajes en transito en la SQS (cmd)
Fuente: Elaboración propia

- **Prueba de salto de región (failover con Route 53)**

  Para llevar a cabo esta prueba se realizó la implementación de un EndPoint de estado denominado `/status` en ambas regiones Figura 4.24 y Figura 4.25, el cual le permite validar al servicio de Route 53 el estado de la aplicación que se encuentra desplegada en esa región como se puede ver en la Figura 4.26.

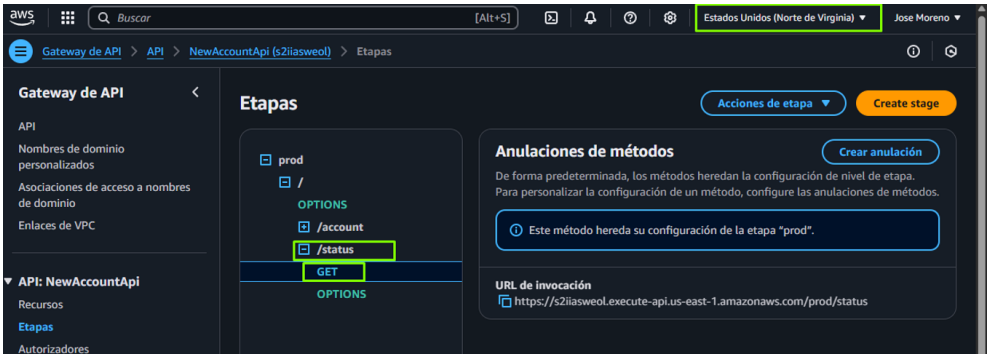


Figura 4.24: EndPoint status (us-east-1)
Fuente: Elaboración propia

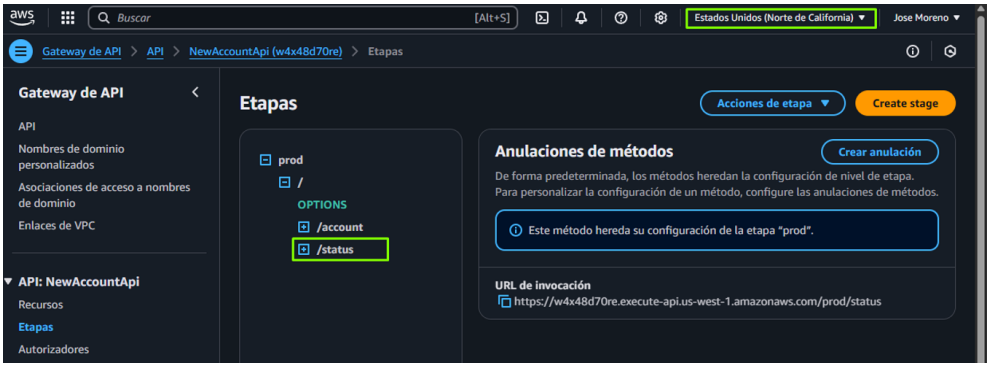


Figura 4.25: EndPoint status (us-west-1)
Fuente: Elaboración propia

Figura 4.26: HealtCheck Route 53
Fuente: Elaboración propia

Ahora se validará que el servicio de Route 53 tenga configuradas las dos ApiGateway disponibles Figura 4.27, con los EndPoint de `/status` y adicionalmente que estos dos se encuentren arriba o funcionales Figura 4.28 para que pueda este tomar la decisión de redirigir las peticiones de la región principal para la región secundaria.

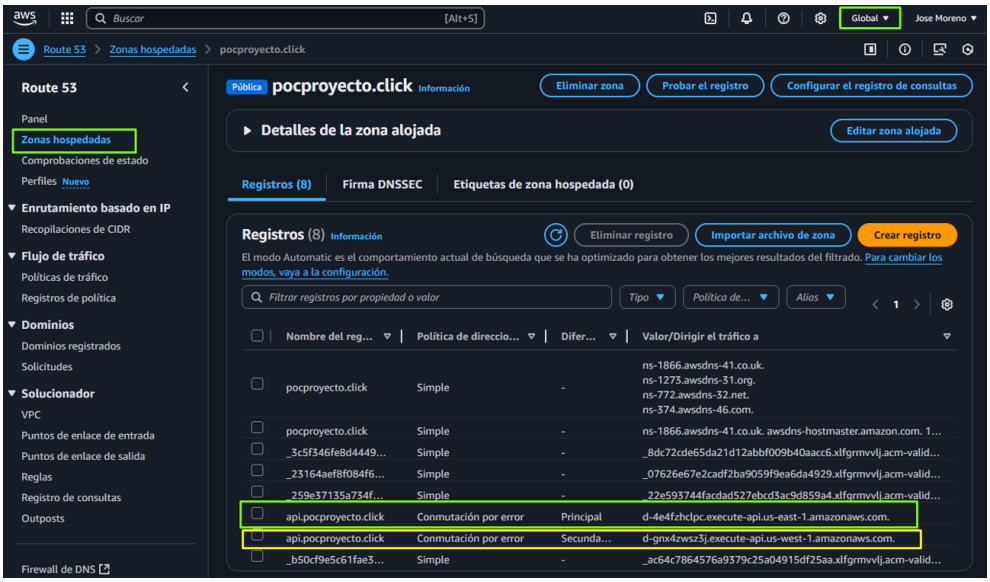


Figura 4.27: Zonas hospedadas
Fuente: Elaboración propia

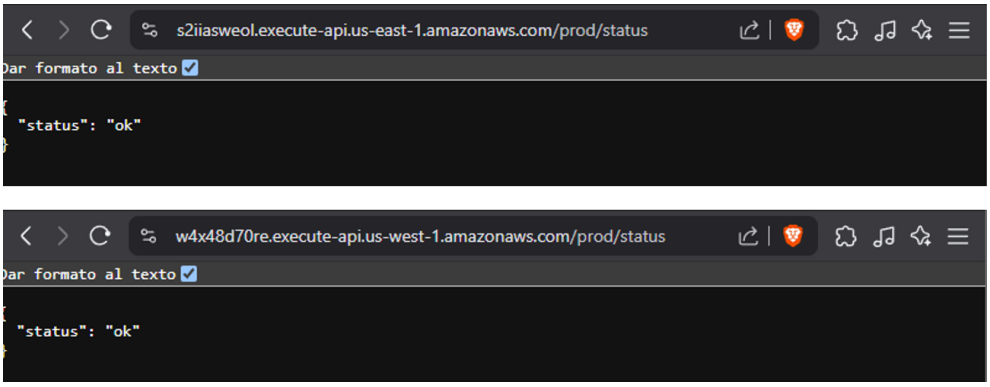


Figura 4.28: Status en ambas regiones
Fuente: Elaboración propia

Ahora se procederá a realizar la inyección de una excepción en el EndPoint de `/status` Figura 4.29 simulando de que esta región está presentando fallas y así el route 53 pueda pasar el flujo a la región secundaria, posteriormente validaremos consumiendo el servicio y se encontrará que este está fallando Figura 4.30.

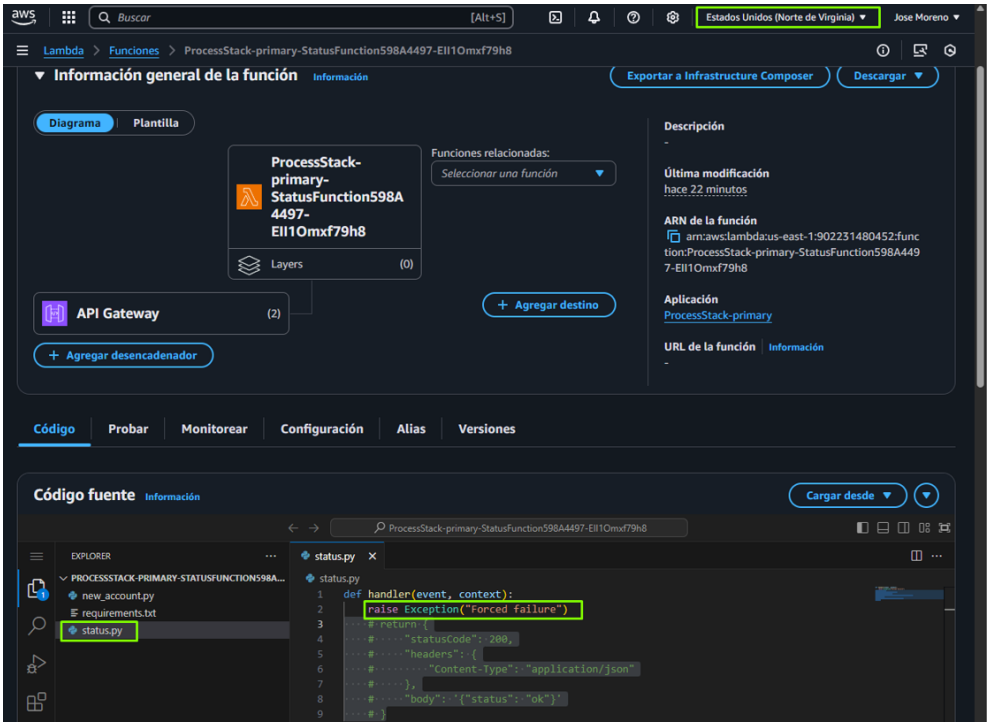


Figura 4.29: Excepción en EndPoint status
Fuente: Elaboración propia

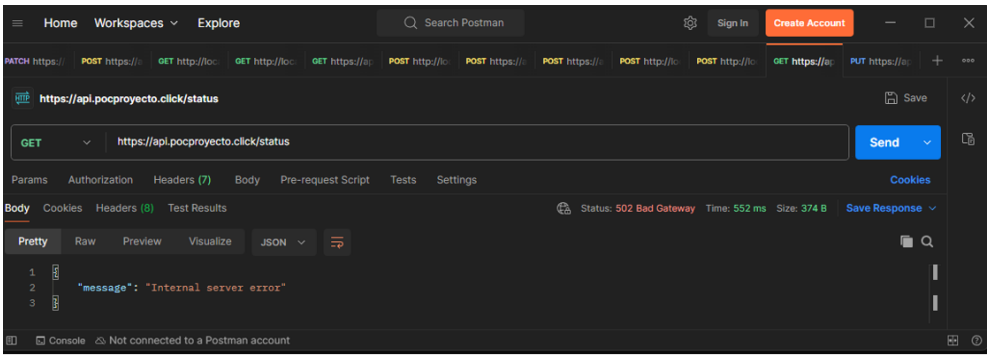


Figura 4.30: Consumo del servicio de status
Fuente: Elaboración propia

En el apartado de comprobación de estado en el servicio de route 53 se podrá evidenciar el detalle del servicio que está en mal estado, es decir, se encuentra abajo Figura 4.31; pero también cabe destacar que, este servicio realiza la comprobación cada 10 segundos con un umbral de error de 2 lo que significa que va hacer dos chequeos desde 3 regiones diferentes para saber si este responde en alguna de las regiones, lo cual sumaria un total de 1 minuto mientras realiza la comprobación de todas las regiones, una vez todas estas regiones reportan que esta fallando el route 53 realiza el redireccionamiento de todo el tráfico de la región principal hacia la región secundaria, permitiendo que el servicio vuelva a su estado normal Figura 4.32

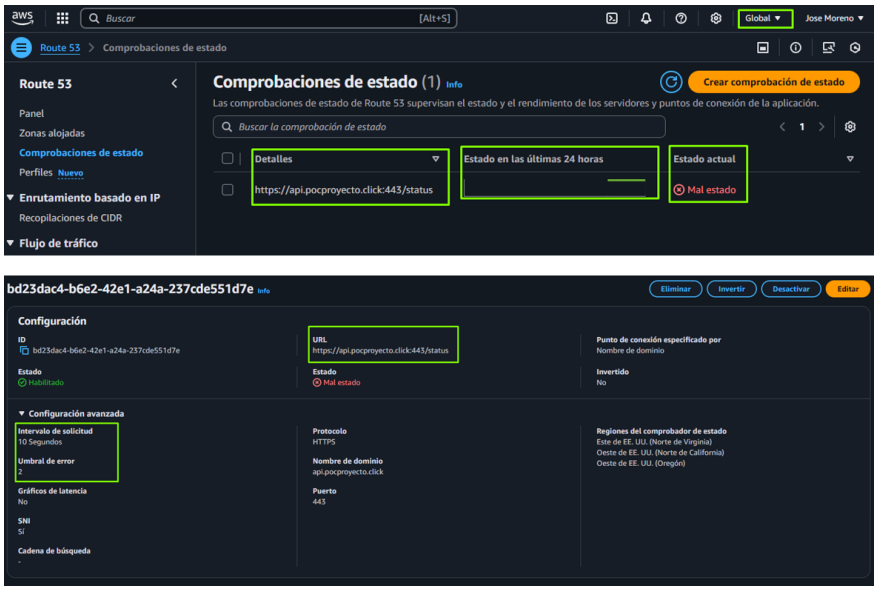


Figura 4.31: Comprobación de estado Route 53
Fuente: Elaboración propia

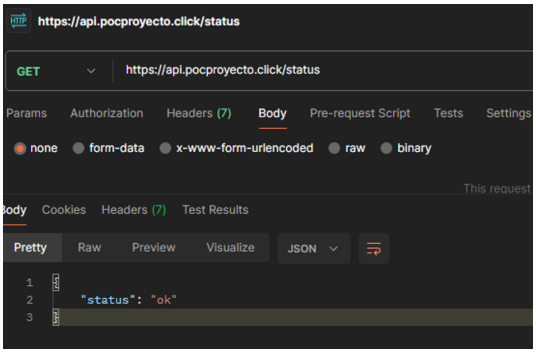


Figura 4.32: Consumo del servicio de status
Fuente: Elaboración propia

Con el fin de organizar y ver fácilmente la relación entre las pruebas ejecutadas, los atributos de calidad para la resiliencia evaluados y sus resultados, se construyó una tabla de trazabilidad. Esta tabla proporciona los medios para mostrar cómo se definieron los escenarios de prueba para validar una o más preocupaciones arquitectónicas clave, y proporciona un resumen de las observaciones realizadas sobre estas durante las pruebas. La trazabilidad proporciona coherencia entre la perspectiva de evaluación definida y los datos recopilados, permitiendo que los resultados se interpreten de una manera rigurosamente estructurada que esté completamente alineada con los objetivos del proyecto.

Prueba realizada	**Atributo de calidad evaluado**	**Resultado observado**
Prueba de carga con Artillery sobre el frontend	- Fiabilidad - Rendimiento - Capacidad de respuesta	Sistema mantuvo tiempos dentro de umbrales (P95 $\leq$ 75ms, P99 $\leq$ 100ms) incluso ante picos de tráfico.
Simulación de falla en región primaria (denegación en S3)	- Disponibilidad - Rapidez de recuperación	Route 53 redirigió automáticamente el tráfico a la región secundaria, sin pérdida de servicio.
Validación de idempotencia en el backend	- Fiabilidad - Robustez	El sistema procesó una sola vez solicitudes con el mismo `request_token`, evitando duplicación.
Simulación de falla temporal en el consumidor (SQS)	- Robustez - Rapidez de recuperación	Eventos encolados fueron procesados correctamente tras la recuperación. Se validó el mecanismo de reintentos.
Fallo forzado de región primaria en backend (Route 53)	- Disponibilidad - Fiabilidad - Rapidez de recuperación	Se ejecutó el failover hacia la región secundaria y se mantuvo la operación sin intervención manual.

Tabla 4.1: Trazabilidad entre pruebas, atributos de calidad para resiliencia y resultados observados

4.3. Consideraciones sobre costos

Aunque en este proyecto no se incluye un análisis detallado de costos, es importante reconocer que la arquitectura propuesta al estar construida sobre servicios administrados por el proveedor de nube y que cuenta con configuraciones en múltiples regiones representa un balance claro entre resiliencia esperada y los costos operativos que consigo esta trae. La duplicación de componentes para asegurar la tolerancia a fallos, así como la replicación de datos entre regiones, puede elevar los gastos mensuales, especialmente en almacenamiento y transferencia de datos (como ocurre con servicios como S3 y DynamoDB).

Sin embargo, en escenarios donde la continuidad del servicio es crítica, es decir, que se necesita que el servicio este funcional en todo momento, como en sectores financieros, de salud o infraestructura, este tipo de inversión es justificable. Además, al apoyarse en servicios serverless y bajo demanda, como AWS Lambda o API Gateway; la arquitectura evita el sobredimensionamiento innecesario y puede adaptarse dinámicamente a la carga real, lo que contribuye a mantener la eficiencia económica a largo plazo.

4.4. Resumen del capítulo

En este capítulo se presentó el diseño e implementación de las pruebas realizada mediante la implementación de una prueba de concepto, con el enfoque de evaluar y así demostrar la validez de la arquitectura resiliente en la nube propuesta anteriormente.

Primero, se describió cómo se estructuró la evaluación mediante escenarios prácticos, orientados a simular condiciones reales de operación y fallos controlados. Cada prueba fue alineada con atributos de calidad específicos para la resiliencia definidos previamente en el proyecto, como fiabilidad, disponibilidad, robustez, rapidez de recuperación y seguridad.

Luego, se mostraron los resultados de las pruebas para el frontend y el backend. Todo, desde pruebas de carga automatizadas (Artillery), hasta simulación de fallos multi-regional, verificación de idempotencia de sistemas distribuidos, hasta la confirmación de que los reintentos funcionan para las colas. En todos los escenarios, el sistema mostró un comportamiento estable y continuó operando sin intervención manual.

Los resultados de esta fase establecieron que la arquitectura descrita anteriormente cumple con los objetivos de resiliencia establecidos por el proyecto y puede ser utilizada como una arquitectura de referencia para su uso en entornos comerciales, que tienen requisitos de alta disponibilidad y continuidad operativa.

Capítulo 5

Conclusiones

5.1. Conclusiones

El presente proyecto de grado tuvo como finalidad diseñar una arquitectura resiliente de referencia en la nube de AWS, considerando los principios del Well-Architected Framework (WAF) con un enfoque en los principios de excelencia operativa, fiabilidad y rendimiento. Durante el desarrollo del trabajo, se cumplieron con cada uno de los objetivos planteados. Partiendo desde la identificación y análisis de los principales atributos de calidad para la resiliencia en la nube, como lo son fiabilidad, disponibilidad, seguridad, robustez y rapidez de recuperación, los cuales sirvieron como base para la definición de los requisitos técnicos y funcionales que guiaron el diseño y el desarrollo de la arquitectura.

El diseño propuesto fue implementado mediante una prueba de concepto que integró una serie de servicios nativos de AWS, en conjunto con un grupo de patrones arquitectónicos como Multi-Región activo-pasivo, procesamiento desacoplado y segmentación por capas. Esta implementación fue validada mediante pruebas que simularon escenarios reales de fallo, lo que permitió cumplir y confirmar que la arquitectura mantiene la continuidad operativa sin ninguna intervención manual, con tiempos reducidos y datos dentro de los límites de tiempo y pérdida.

Los resultados obtenidos se encuentran alineados con la literatura existente, particularmente con las investigaciones y planteamientos de autores como (Mezzalira et al., 2022a), (Pan et al., 2023b), (Firesmith, 2019) y (Nandwani et al., 2023), entre otros, que abordan la resiliencia como una propiedad compuesta de múltiples atributos de calidad técnicos. Así mismo, el proyecto confirma que los pilares del Well-Architected Framework (WAF) aunque no diseñados exclusivamente para resiliencia pueden utilizarse estratégicamente para fortalecer sistemas distribuidos. Esta propuesta, por tanto, contribuye al conocimiento técnico y académico consolidando una estructura reutilizable y que se encuentra alineada con las buenas prácticas de AWS, respondiendo a una necesidad real encontrada en la industria.

A nivel práctico, esta arquitectura puede ser utilizada por profesionales encargados de diseñar soluciones críticas o tolerantes a desastres en la nube, reduciendo el uso de prácticas empíricas, minimizando riesgos de inactividad y mejorando la capacidad de recuperación ante fallos. Sin embargo, el trabajo también presenta limitaciones, a considerar, teniendo en cuenta que sólo funciona para AWS, sin incluir escenarios multicloud o híbridos. Además, la validación se realizó en un entorno de laboratorio implementando una prueba de concepto, sin pruebas de carga a gran escala o condiciones de un entorno de producción. Asimismo, no se incorporaron soluciones más avanzadas como autoescalado predictivo o recuperación mediante inteligencia artificial.

Para trabajos futuros de investigación, se plantea extender el enfoque propuesto hacia una arquitectura multicloud, validar su comportamiento en condiciones productivas mucho más exigentes, e incorporar mecanismos de recuperación apoyados por la IA. También resulta relevante explorar arquitecturas resilientes en contextos de microservicios o contenedores.

En resumen, este trabajo logró construir y validar una arquitectura resiliente de referencia en la nube de AWS, que ofrece una referencia práctica y replicable para la construcción de soluciones críticas basadas en la nube. Las conclusiones reiteran que el diseño proactivo, el desacoplamiento funcional y el monitoreo continuo son pilares esenciales al servicio de la resiliencia operativa de los sistemas modernos.

5.2. Trabajos futuros

A partir de lo desarrollado en este proyecto, se identificaron algunas áreas adicionales en las que es posible realizar un trabajo más detallado para madurar el diseño y la validación de arquitecturas resilientes en la nube.

En primer lugar, un trabajo futuro prometedor sería extender la propuesta a entornos multicloud y nube híbrida, comparando la resiliencia de los diferentes proveedores y definiendo patrones de interoperabilidad que mantengan los atributos de calidad propuestos, incluso fuera del ecosistema de AWS.

Además, sería interesante incluir mecanismos o estrategias avanzadas de recuperación automática, como el uso de mecanismos predictivos basados en IA (Inteligencia Artificial) o ML (Aprendizaje Automático), que permitan anticiparse a fallos y reaccionar en tiempo real con mayor precisión. También sería importante, la integración de arquitecturas basadas en contenedores con autoescalado inteligente, balanceo de carga global.

Por último, también sería importante el desarrollo de herramientas de verificación de resiliencia automatizadas, incorporando pruebas continuas como (chaos engineering, monitoreo sintético y trazabilidad distribuida) en el ciclo de vida del desarrollo. Estas herramientas permitirían institucionalizar la resiliencia como una práctica constante y no como una validación puntual.

5.3. Lecciones aprendidas

En el apartado de lecciones aprendidas se puede destacar que se obtuvo aprendizajes significativos desde el punto de vista teórico y práctico los cuales facilitaron el proceso de investigación, diseño y validación de la propuesta de arquitectura resiliente en la nube de AWS.

En primer lugar, se observó que la resiliencia en sistemas y aplicaciones distribuidas no se puede considerar simplemente desde un solo componente o tecnología, sino que debe especificarse como una propiedad transversal desde un punto temprano del diseño y desarrollo de un sistema, teniendo en cuenta que el termino de resiliencia integra múltiples capas de protección como redundancia geográfica, desacoplamiento asíncrono, mecanismos de idempotencia y monitoreo proactivo; los cuales son esenciales para construir soluciones verdaderamente tolerantes a fallos.

Otro aprendizaje importante fue comprender que los pilares del Well-Architected Framework, aunque no están dirigidos únicamente a la resiliencia, ofrecen una base sólida sobre la cual pueden alinearse los atributos de calidad necesarios para garantizar continuidad operativa. Esta alineación metodológica permitió estructurar la arquitectura con base teórica, pero también involucrando las mejores prácticas consolidadas por AWS.

Desde el punto de vista técnico, se reafirmó lo valioso que es adoptar servicios nativos de la nube desacoplados como Lambda, SQS y DynamoDB con replicación multirregional haciendo posible la construcción de soluciones flexibles, escalables y más fáciles de recuperar. Así mismo, se evidenció la importancia de la automatización en la recuperación ante fallos, como elemento central para minimizar tiempos de respuesta (RTO) y pérdida de datos (RPO).

En cuanto al proceso de evaluación, se enfatizó el aprendizaje de cómo diseñar e implementar pruebas de resiliencia controladas con fallos simulados, reintentos, conmutaciones por error automáticas y verificaciones de idempotencia. Estas pruebas demostraron que, incluso sin infraestructura compleja, es posible validar de forma estructurada los atributos de calidad resilientes de una arquitectura.

Bibliografía

(2023). ISO/IEC 25010:2023 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Disponible en: https://www.iso25000.com/index.php/normas-iso-25000/iso-25010.

Amazon (2023). Qué es la migración a la nube. *https://aws.amazon.com/es/what-is/cloud-migration*.

AWS (2023). AWS Well-Architected.

AWS (2023a). Aws well-architected. *https://aws.amazon.com/es/architecture/well-architected*.

AWS (2023b). Qué es la computación en la nube. *https://aws.amazon.com/es/what-is-cloud-computing*.

AWS-Cloud (2023). Tipos de computación en la nube. *https://aws.amazon.com/es/types-of-cloud-computing*.

Cai, B., Xie, M., Liu, Y., Liu, Y., and Feng, Q. (2018). Availability-based engineering resilience metric and its corresponding evaluation methodology. *Reliability Engineering System Safety*, 172:216–224.

Eliot, S. (2021). Arquitectura de recuperación de desastres (dr) en aws, parte i: Estrategias para la recuperación en la nube.

Firesmith, D. (2019). System resilience part 2: How system resilience relates to other quality attributes. Carnegie Mellon University, Software Engineering Institute's Insights (blog). Accessed: 2024-Mar-15.

Gartner Inc. (2023). Gartner highlights 9 principles to improve cloud resilience. Disponible en línea.

Gartner-Inc. (2023). Gartner highlights 9 principles to improve cloud resilience.

Guamán, F. (2023). Arquitecturas resilientes: Diseñando sistemas para el fracaso.

Hornsby, A. (2019a). Patterns for resilient architecture — part 1.

Hornsby, A. (2019b). Patterns for resilient architecture — part 1. *https://adhorn.medium.com/patterns-for-resilient-architecture-part-1-d3b60cd8d2b6*. Disponible en línea.

Hosseini, S., Yodo, N., and Wang, P. (2014). Resilience modeling and quantification for design of complex engineered systems using bayesian networks. In *Proceedings of the ASME Design Engineering Technical Conference*, volume 2.

Martin, C., Morillo, F., and de Esteban Belzuz, M. (2022). Construyendo una arquitectura segura, altamente disponible en aws.

Martínez, N. A. V., Valencia, C. M. Y., and Valencia, B. D. C. (2023). Resiliencia en la informática. *RECIMUNDO*, 7:79–86.

Mezzalira, L., Hyatt, L., Denti, V., and Jaupaj, Z. (2022a). Let's architect! creating resilient architecture. *https://aws.amazon.com/es/blogs/architecture/lets-architect-creating-resilient-architecture*. Disponible en línea.

Mezzalira, L., Hyatt, L., Denti, V., and Jaupaj, Z. (2022b). Let's architect! creating resilient architecture.

Nandwani, H., Taylor, L., and McClure, B. (2023). Entender los patrones de resiliencia y las consideraciones claves para arquitecturar eficientemente en la nube. *https://aws.amazon.com/es/blogs/aws-spanish/entender-los-patrones-de-resiliencia-y-las-consideraciones-claves-para-arquitecturar-eficientemente-en-la-nube*.

NIST (2020). Security and Privacy Controls for Information Systems and Organizations. Technical Report Special Publication 800-53 Revision 5, NIST. Disponible en: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf.

Pan, J., Liu, Z., Li, D., Wang, L., and Li, B. (2023a). An empirical study of software architecture resilience evaluation methods. *The Journal of Systems Software*, pages 1–2.

Pan, J., Liu, Z., Li, D., Wang, L., and Li, B. (2023b). An empirical study of software architecture resilience evaluation methods. *The Journal of Systems Software*, pages 1–2.

SAP (2024). The definitive guide to reference architecture. *https://www.leanix.net/en/wiki/ea/reference-architecturewhat-does-ra-mean*.

Villabona, C. G. (2022). Creación de cargas de trabajo resilientes usando arquitecturas basadas en "células".

Zhang, M., Wang, L., Jajodia, S., and Singhal, A. (2021). Network attack surface: lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks. *IEEE Transactions on Dependable and Secure Computing*, 18(1):310–324.

Zhu, S., Guo, L., Cui, Y., Xiao, R., Yu, Z., Jin, Y., Fu, R., Zhang, J., Xu, T., Chen, J., et al. (2019). Quality suitability modeling of volatile oil in chinese materia medica–based on maximum entropy and independent weight coefficient method: Case studies of atractylodes lancea, angelica sinensis, curcuma longa and atractylodes macrocephala. *Industrial Crops and Products*, 142:111807.