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

Diseño de arquitectura para una colaboración eficiente en Desarrollos Serverless: Integración de plataformas para documentación y generación automatizada de código

Edgar Enrique Roa Pérez

Director(a): Wilson Calvo Alvarez

30 de mayo de 2025

Santiago de Cali, 30 de mayo de 2025.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Wilson Calvo Alvarez
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 arquitectura para una colaboración eficiente en Desarrollos Serverless: Integración de plataformas para documentación y generación automatizada de código" realizado por el estudiante de Magister en Ingeniería de Software Edgar Enrique Roa Pérez (cod: 8989016), 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: 2025.08.13 18:57:48 -05'00'

Wilson Calvo Alvarez

Santiago de Cali, 30 de mayo de 2025.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Wilson Calvo Alvarez
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Diseño de arquitectura para una colaboración eficiente en Desarrollos Serverless: Integración de plataformas para documentación y generación automatizada de código" 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,

Edgar Enrique Roa Pérez
Código: 8989016

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Diseño de Arquitectura para una Colaboración Eficiente en Desarrollos Serverless Integración de Plataformas para Documentación y Generación Automatizada de Código.

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Ciencias de la Computación - Arquitectura de Software
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Innovación
4. Estudiante: Edgar Enrique Roa Perez
5. Correo electrónico: edgaroa77@javerianacali.edu.co
6. Dirección y teléfono: Cll 2 No 1C-00 Torre 1, Apto 107, 3152838493
7. Director: Wilson Calvo Alvarez
8. Vinculación del director: N/A (Planta/Cátedra/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): Serverless, Colaboración, Documentación, Generación Automatizada, Arquitectura
14. ODS que aplica al proyecto (Agenda 2030): ???
15. Fecha de inicio: 17 de Junio de 2024
16. Resumen: el diseño de una arquitectura para una colaboración eficiente en desarrollos serverless, enfocada en la integración de plataformas que permitan la generación automatizada de documentación técnica y código fuente en entornos de nube. La importancia de esta propuesta radica en los desafíos actuales asociados al crecimiento de componentes distribuidos, la pérdida de trazabilidad entre equipos, y la falta de automatización estandarizada en la documentación y creación de plantillas de infraestructura. La investigación aborda como problemática central la dificultad para mantener consistencia y eficiencia en entornos de desarrollo serverless, donde múltiples equipos colaboran sobre distintas plataformas y lenguajes. Para resolver esta problemática, se definieron cinco objetivos específicos: diseñar un sistema de

clasificación y gestión de desarrollos existentes, automatizar la generación de documentación sincronizada con repositorios, crear un mecanismo de generación de código parametrizable, implementar un motor de selección dinámica de arquitecturas y optimizar la colaboración entre equipos mediante herramientas integradas. La solución fue planteada mediante un conjunto de arquitecturas modulares soportadas por servicios serverless, diseñadas para ejecutarse sobre múltiples proveedores cloud. Cada arquitectura incluye flujos específicos, componentes desacoplados, almacenamiento sin servidor, gestión de eventos, trazabilidad, y coordinación con plataformas como GitHub, Bitbucket, Swagger, Confluence, DynamoDB, Lambda, Step Functions, entre otras. La evaluación técnica, basada en el método ATAM, validó el cumplimiento de atributos como escalabilidad, disponibilidad, rendimiento y seguridad, y permitió identificar decisiones arquitectónicas alineadas a los objetivos definidos. Además, se realizó una prueba de concepto para verificar métricas de rendimiento y escenarios de calidad. Entre los principales resultados se encuentra una arquitectura integral trazable y replicable en ambientes de desarrollo, pruebas y producción, con flujos documentados para cada objetivo. Las lecciones aprendidas reflejan la importancia de la modularidad, el diseño impulsado por atributos de calidad y la necesidad de incorporar herramientas de colaboración visual y automatización desde las primeras etapas del diseño. Esta propuesta aporta una base sólida para proyectos que buscan eficiencia, trazabilidad y automatización en el desarrollo de soluciones serverless distribuidas.

Agradecimientos

Primero y ante todo, quiero expresar mi más sincero agradecimiento a Dios, por concederme la vida, la salud, la sabiduría y la fortaleza necesarias para culminar esta etapa académica. Su guía ha sido fundamental en cada momento de este proceso, brindándome esperanza y propósito incluso en los momentos más difíciles.

Dedico este logro a la memoria de mi madre (†) Gilma Perez de Roa, quien partió de este mundo, pero cuyo amor, ejemplo y enseñanzas permanecen profundamente en mí. Ella solía decir que "la mejor herencia es el estudio", y esas palabras se convirtieron en un faro que iluminó cada decisión que tomé en mi formación profesional. Hoy, más que nunca, su legado me inspira a seguir creciendo, a no rendirme, y a honrar todo lo que me enseñó. Gracias, mamá, por todo el apoyo que me diste en vida y por acompañarme ahora, desde el cielo.

A mi padre Elibrando J. Roa M., por su constante respaldo, sus consejos y por creer en mí desde el primer momento. Su apoyo fue fundamental para tomar la decisión de convertirme en ingeniero, y espero sinceramente que este logro represente un motivo de orgullo para él. Gracias por estar presente, por tus palabras de aliento y por acompañarme en cada etapa de este camino.

Extiendo también mi agradecimiento a la Pontificia Universidad Javeriana de Cali, por abrirme las puertas y brindarme la oportunidad de iniciar y culminar esta maestría. Esta institución no solo me ofreció una formación académica de excelencia, sino también un entorno propicio para el crecimiento personal, intelectual y profesional. Gracias a sus docentes, investigadores y comunidad académica por fomentar un ambiente de compromiso, innovación y reflexión crítica.

Asimismo, agradezco profundamente a mi familia, a mis seres queridos, amigos y profesores que me acompañaron durante este proceso. Cada palabra de aliento, cada consejo y cada gesto de apoyo ha sido invaluable.

Este trabajo no solo representa un esfuerzo académico, sino también un tributo a quienes han influido profundamente en mi vida. A todos ustedes, gracias.

Resumen

El proyecto tiene el objetivo diseñar una arquitectura orientada a mejorar la colaboración en entornos de desarrollo con serverless, mediante la integración de plataformas que permitan la documentación y generación automatizada de código. La solución se enfoca en facilitar la identificación, clasificación y trazabilidad de componentes distribuidos como microservicios, APIs o funciones aprovechando tecnologías cloud y herramientas colaborativas. Se diseñaron módulos arquitectónicos que automatizan tareas clave del ciclo de desarrollo: desde la generación de plantillas, documentación técnica y código base, hasta la recomendación dinámica de arquitecturas óptimas. La arquitectura planteada se basa en principios de escalabilidad, modularidad, interoperabilidad y automatización, garantizando su aplicabilidad en entornos reales de desarrollo sobre proveedores como AWS, Azure o GCP. La propuesta fue evaluada mediante escenarios de calidad, trazabilidad de decisiones arquitectónicas y análisis de riesgos, validando su viabilidad técnica y alineación con los objetivos del proyecto.

Palabras Clave

- Serverless Architecture.
- Collaboration Tools
- Automated Code Generation
- Cloud Platforms
- Design Optimization

Abstract

This research presents the design of an architectural framework aimed at enhancing efficient collaboration in serverless development environments, by integrating platforms for automated code generation and technical documentation. The proposed solution enables the classification, traceability, and standardized management of distributed components—such as APIs, microservices, and monoliths—across multi-cloud ecosystems. The architecture is composed of modular designs that support key processes in the development lifecycle, including dynamic template generation, documentation updates synchronized with repositories, and intelligent architectural recommendations. Built on principles of scalability, interoperability, automation, and modularity, this approach leverages cloud-native services across providers like AWS, Azure, and Google Cloud. The architectural design was evaluated using a hybrid method based on the Architecture Tradeoff Analysis Method (ATAM), assessing quality attributes, decision traceability, and technical risk. Results confirm the feasibility and potential impact of the solution in real-world serverless collaboration and development scenarios.

Keywords: Serverless Architecture, Collaboration Tools, Automated Code Generation, Cloud Platforms, Design Automation.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La acogida en aumento de arquitecturas serverless ha mejorado significativamente la eficiencia en el desarrollo de software, al liberar a los equipos de la gestión directa de infraestructura y permitirles concentrarse en la lógica de negocio. Sin embargo, esto trae consigo problemas críticos en contextos empresariales, especialmente en la colaboración entre equipos y en la generación y mantenimiento de documentación técnica.

En muchas organizaciones se ha identificado una carencia alarmante:

- **Falta de conocimiento**: Sobre los desarrollos y microservicios existentes, dificultando la identificación y reutilización de componentes. (Azanza et al., 2021).
- **Ausencia de documentación detallada y estandarizada**: lo que retrasa la incorporación de nuevos miembros y genera errores en la evolución del software.
- **Procesos manuales de generación de código**: incrementando la duplicación de esfuerzos y el riesgo de inconsistencias.

  Estos problemas se agravan en entornos empresariales híbridos, donde conviven sistemas tradicionales con arquitecturas en la nube, y donde las funciones serverless, al ser efímeras y estar altamente distribuidas, dificultan aún más la trazabilidad, el monitoreo y la colaboración entre equipos. La rápida iteración propia de metodologías ágiles y DevOps, junto con la falta de herramientas centralizadas que documenten y estandaricen el ciclo de vida de los microservicios, aumenta la complejidad operativa.

  Precisamente en este caso, nace la necesidad de una solución que no solo facilite la colaboración y el mantenimiento genérico, sino que también automatice tareas repetitivas, aumente la visibilidad de los activos tecnológicos y estandarice la creación de los nuevos componentes de software. En este sentido, se presenta un proyecto de tesis que tiene por título "Diseño de Arquitectura para una Colaboración Eficiente en Desarrollos Serverless: Integración de Plataformas para Documentación y Generación Automatizada de Código". La solución propuesta se basa en tres pilares fundamentales:

- **Automatización de documentación**: A través de herramientas como Swagger, OpenAPI, y su integración con pipelines de CI/CD (por ejemplo, GitHub Actions o AWS CodePipeline), se busca generar especificaciones técnicas vivas que reflejen el estado real de los desarrollos.
- **Gestión centralizada de desarrollos**: Mediante el diseño de un catálogo unificado de microservicios, soportado por tecnologías como DynamoDB y S3, se podrá consultar, clasificar y versionar cada componente de forma estructurada.

- **Generación dinámica de código**: Utilizando frameworks como AWS CDK, Terraform o Yeoman, se generarán plantillas que reflejen distintos estilos arquitectónicos (hexagonal, por capas, orientado a eventos), ajustándose a la naturaleza del desarrollo.

  Para fundamentar esta propuesta, se parte de prácticas consolidadas, IaC y un marco de referencia utilizado, como el AWS Well-Architected Framework, asegura escalabilidad, interoperabilidad y eficiencia operativa. Igualmente, se ha previsto integridad con herramientas de gestión y colaboración, como GitHub y Jira, para promover la mirada transversal sobre los desarrollos. La validación de esta arquitectura podrá validarse a través de un caso de estudio por una real, con experiencia con AWS, y más de 50 microservicios serverless.

- Reducción del tiempo de documentación técnica.
- Incremento en la adopción de plantillas de código estandarizadas.
- Disminución en la duplicación de funcionalidades y errores por falta de trazabilidad.

  A partir de la ambición inicial de esta tesis, no solo se pretende eliminar los problemas anteriormente identificados en la práctica del desarrollo del software, sino también proporcionar un punto de referencia reutilizable para otras organizaciones o personas que necesitan mejorar sus prácticas. sistemas de diseño, documentación y colaboración sobre arquitecturas sin servidor.

1.1. Definición del problema

El problema central que motiva este proyecto de tesis radica en las limitaciones observadas en la gestión de desarrollos basados en arquitecturas serverless dentro de entornos empresariales híbridos. Aunque la adopción de servicios sin servidor ha demostrado beneficios notables en cuanto a agilidad y reducción de costos, su implementación en contextos corporativos trae consigo múltiples desafíos operativos, particularmente en lo referente a la colaboración entre equipos, la documentación técnica y la estandarización de procesos de desarrollo.

La adopción de arquitecturas serverless ha ganado terreno debido a su promesa de eficiencia, escalabilidad y simplificación de la infraestructura. Sin embargo, la gestión de estos desarrollos se ve limitada por una serie de brechas que aún no han sido resueltas de forma integral. Entre los principales desafíos se encuentran:

1. Ausencia de visibilidad de desarrollos existentes: Los equipos de trabajo no cuentan con un inventario centralizado que permita identificar los microservicios activos, su propósito, su nivel de uso o su responsable técnico. Esto dificulta la toma de decisiones estratégicas, favorece la duplicación de esfuerzos y obstaculiza la trazabilidad en ambientes complejos.

2. Documentación inconsistente: La generación de documentación suele realizarse de manera manual, parcial o completamente omitida, lo que deriva en pérdida de conocimiento organizacional, errores de configuración y una curva de aprendizaje prolongada para nuevos desarrolladores o equipos de soporte.

3. Colaboración fragmentada entre equipos: Las herramientas utilizadas para la gestión de requerimientos, desarrollo, control de versiones y despliegue suelen operar de manera aislada. Esta desconexión reduce la eficiencia del flujo de trabajo, provoca retrabajos y limita la capacidad de escalar o mantener arquitecturas modulares con equipos distribuidos.

4. Generación manual de código y configuración: La creación de nuevas funcionalidades serverless, así como la configuración de permisos, variables de entorno e integración con servicios externos, se realiza sin mecanismos reutilizables ni estandarizados. Esto incrementa el esfuerzo requerido y eleva el riesgo de errores en producción.

Por tanto, esta tesis plantea como problema central la necesidad de contar con una arquitectura integral que facilite la identificación, documentación, clasificación y reutilización de componentes de software, independientemente del estilo arquitectónico adoptado (microservicios, monolitos, funciones serverless, entre otros). Esta solución debe ser interoperable entre proveedores de nube (AWS, Azure, Google Cloud), integrarse con herramientas colaborativas existentes (como GitHub, Jira, Confluence, Slack, Trello, Azure Boards, Notion, Microsoft Teams o Google Workspace) y automatizar tareas críticas como la generación de código e infraestructura. De esta manera, se pretende reducir la complejidad operativa, mejorar la trazabilidad de los desarrollos y fomentar una cultura de colaboración eficiente y sostenible en el tiempo.

1.1.1. Planteamiento del problema

La situación actual en muchas organizaciones tecnológicas refleja una brecha sustancial en la gestión, comprensión y control de sus desarrollos en arquitecturas serverless. A pesar de su creciente adopción y los beneficios que aporta. Como reducción de costos, escalabilidad automática y eliminación de tareas operativas.Su implementación ha puesto en evidencia carencias estructurales que afectan directamente la eficiencia de los procesos de desarrollo y la colaboración entre equipos.

Uno de los principales síntomas de esta problemática es:

1. La falta de documentación estructurada y actualizada: Lo que dificulta la comprensión del estado y la funcionalidad de los desarrollos existentes. Esta carencia, a su vez, impide que los equipos identifiquen qué servicios están desplegados, cuál es su propósito y cómo pueden ser reutilizados. Como consecuencia, se presentan casos frecuentes de duplicación innecesaria de funcionalidades, pérdida de conocimiento institucional y aumento en la deuda técnica.

   Además, existe un desconocimiento generalizado de las funcionalidades que ya han sido desarrolladas. Esta situación conlleva a que los equipos reimplementen soluciones existentes o integren nuevos servicios sin considerar los impactos en los flujos actuales de la organización. En entornos distribuidos, donde múltiples equipos trabajan de forma paralela sobre distintos dominios, esta falta de visibilidad incrementa la complejidad del mantenimiento y deteriora la calidad del software.

2. La colaboración ineficiente:entre equipos también se ve agravada por la desconexión entre las herramientas utilizadas en el ciclo de vida del software. Plataformas como GitHub, Jira,

Confluence, Slack o Azure Boards, que deberían facilitar la gestión de requerimientos, documentación y control de versiones, no están integradas con los flujos serverless, generando fricciones, errores y retrabajos.

Un caso ilustrativo de cómo una arquitectura serverless bien gestionada puede traer beneficios sustanciales es el de Coca-Cola, que adoptó este enfoque para su solución de dispensadores inteligentes. La compañía reportó mejoras significativas en eficiencia y reducción de costos. Este tipo de resultados evidencia que una adecuada estrategia de gestión y colaboración puede ser un factor decisivo para maximizar el potencial del paradigma serverless.

De igual manera, un informe de Forrester Research resalta que organizaciones que adoptan herramientas de colaboración integradas como GitHub han experimentado mejoras en la productividad de los desarrolladores y disminución en la cantidad de errores. Estos hallazgos confirman que, más allá de la arquitectura tecnológica, el éxito de un entorno serverless depende en gran medida de contar con mecanismos sólidos de visibilidad, reutilización y trabajo colaborativo.

Por lo tanto, el planteamiento central del problema se resume en la ausencia de una estrategia y modelo arquitectónico que habilite y soporte la gestión unificada y automatizada de los desarrollos existentes, permitiendo identificar lo que ya se ha construido y facilitando la colaboración eficaz entre los equipos. Este enfoque no resuelve por sí mismo el problema de negocio, pero provee las capacidades técnicas, metodológicas y de integración necesarias para que las organizaciones optimicen procesos, reduzcan duplicidades y aprovechen la reutilización, estandarización y automatización como medios para alcanzar los objetivos estratégicos.

TIP: Contexto + antecendents + situación problema

1.1.2. Sistematización

La sistematización del problema se realiza mediante la formulación de preguntas de investigación que orientan el desarrollo de este proyecto de grado. Estas preguntas, de carácter abierto, buscan explorar soluciones técnicas y metodológicas para abordar los desafíos identificados en la gestión de desarrollos serverless en entornos empresariales híbridos. Su propósito es guiar tanto el diseño de la arquitectura propuesta como su implementación, evaluación y validación empírica. A continuación, se presentan las preguntas clave formuladas con base en el contexto, los antecedentes y la situación problemática documentada en la literatura y en casos reales:

- **¿Cómo diseñar una arquitectura para una aplicación que analice automáticamente los desarrollos existentes y genere documentación técnica detallada en tiempo real?** Este interrogante busca abordar la falta de documentación, mediante la integración de herramientas como Swagger, AWS CodePipeline o GitHub Actions, para construir pipelines que mantengan sincronizada la documentación con los repositorios de código.

- **¿Qué herramientas y patrones arquitectónicos (hexagonal, microservicios, por capas) permiten implementar un sistema de generación automatizada de código adaptable a las necesidades específicas de cada proyecto?** Este punto indaga sobre

la viabilidad de diseñar plantillas reutilizables con tecnologías como AWS CDK, Terraform o Cookiecutter, que reduzcan errores y aceleren la puesta en producción.

- **¿Cómo incorporar un módulo de selección dinámica de arquitecturas que optimice la elección de patrones (ejemplo: hexagonal vs. microservicios) según métricas de rendimiento, costos y escalabilidad?** Una arquitectura flexible y adaptativa es clave en entornos serverless. Esta pregunta explora el uso de análisis basado en datos, como los modelos que pueden desarrollarse con AWS SageMaker o herramientas de observabilidad.
- **¿Qué métricas y metodologías permiten medir la mejora en la colaboración interequipos al implementar una arquitectura integrada de gestión serverless?** La colaboración fragmentada impacta negativamente la productividad. Aquí se busca definir indicadores como el tiempo medio de resolución de incidencias o la frecuencia de reutilización de artefactos, considerando experiencias exitosas documentadas como la de Coca-Cola y las mejores prácticas sugeridas por GitHub.

Estas preguntas no solo están alineadas con los objetivos generales y específicos del proyecto, sino que también se enmarcan en tendencias actuales como GitOps, MLOps en serverless, y documentación como código. Su respuesta contribuirá a consolidar un marco de referencia replicable en otros contextos organizacionales y a proponer una solución tecnológica robusta, sostenible y alineada con estándares industriales actuales.

La formulación del problema como una o varias preguntas debe incluir preguntas abiertas, las preguntas pueden empezar por palabras como *qué* o *cómo*, puesto que son más una guía para orientar el trabajo que la búsqueda de una única causa de un fenómeno

1.2. Objetivos del proyecto

Los objetivos deben formularse de manera que logren transmitir lo que intenta realizar el investigador y lo que espera obtener como resultado.

Los objetivos deben iniciar con un verbo en infinitivo (construir, diseñar, seleccionar, analizar, modelar simular, etcétera.)

1.2.1. Objetivo General

Diseñar una arquitectura para una aplicación que facilite la mejora de la colaboración y documentación en desarrollos con serverless, integrando plataformas que permitan la generación automatizada de código y la elaboración detallada, estructurada y actualizada de documentación técnica. La validación del objetivo se realizará mediante indicadores como la reducción de la duplicación de desarrollos detectada en el catálogo de componentes, la disminución en el tiempo de búsqueda y recuperación de artefactos técnicos, y el aumento de la trazabilidad de cambios en repositorios y herramientas de control de versiones. Estos indicadores permitirán evaluar de forma objetiva

el impacto de la solución en la optimización del trabajo colaborativo en entornos empresariales híbridos.

1.2.2. Objetivos específicos

- Analizar, definir y diseñar un módulo de gestión y clasificación, implementado bajo un enfoque serverless, que permita identificar y organizar de forma eficiente los desarrollos existentes en el entorno empresarial. La selección de este enfoque arquitectónico se justificará en función de criterios de escalabilidad, mantenimiento y reducción de costos operativos. Se espera completar la fase de definición y diseño del módulo en los dos primeros meses del proyecto, para posteriormente proceder con su implementación y validación.
- Diseñar un mecanismo automatizado para la generación detallada de documentación técnica de los servicios, entregable al finalizar el cuarto mes del proyecto.
- Diseñar un sistema de generación automatizada de código y plantillas parametrizables que permita incorporar nuevas funcionalidades adaptadas a la naturaleza de cada desarrollo.
- Diseñar un módulo de selección dinámica de patrones arquitectónicos (hexagonal, por capas, microservicios) que optimice la elección según métricas de rendimiento, escalabilidad y acoplamiento, con entrega prevista para el sexto mes.

1.3. Delimitaciones y alcances

El proyecto se enfoca en analizar y diseñar una arquitectura integral para entornos serverless que optimice la colaboración, documentación y generación de código en empresas con desarrollos distribuidos. A continuación, se especifican los aspectos técnicos y funcionales cubiertos:

- **Gestión y clasificación eficiente de desarrollos** serverless:
  - Se diseñará una arquitectura basada en servicios serverless que facilite la gestión y clasificación eficiente de los desarrollos existentes en el entorno empresarial. Esta arquitectura permitirá identificar y clasificar dinámicamente los desarrollos según criterios específicos, utilizando servicios serverless para su funcionamiento óptimo y escalable. El sistema resultante proporcionará una visión integral de los desarrollos existentes, permitiendo su gestión y clasificación de manera ágil y eficaz.
- **Generación Automatizada de Documentación:**
  - Definir un mecanismo capaz de generar y actualizar automáticamente la documentación técnica de cada componente de software (microservicios, monolitos, webservices, funciones serverless), de forma que la información se mantenga sincronizada con los cambios en el código. El alcance contempla únicamente su diseño y especificación técnica.

- **Generación Automatizada de Código:**
  - Se propone el diseño de una arquitectura destinada a la generación automatizada de código, con el fin de producir plantillas y facilitar la incorporación eficaz de nuevas funcionalidades en entornos serverless. Esta arquitectura estará meticulosamente diseñada para optimizar el proceso de generación de código, teniendo en cuenta las especificidades únicas de cada proyecto.
- **Diseño de Arquitectura del Módulo de Selección Dinámica de Arquitecturas:**
  - Creación de una arquitectura que facilite la selección dinámica de arquitecturas, considerando criterios específicos y enfocándose en entornos serverless, integrando opciones para elegir adecuadamente las arquitecturas según cada desarrollo.
- **Mejora del Entorno de Colaboración:**
  - Diseño de una arquitectura para la interfaz que presente información clara y detallada sobre los desarrollos y sus microservicios asociados, junto con servicios y herramientas que fomenten la colaboración entre equipos de desarrollo, promoviendo una visión integral de los proyectos.

1.4. Justificación del trabajo de grado

El presente trabajo, titulado "Diseño de Arquitectura para una Colaboración Eficiente en Desarrollos Serverless: Integración de Plataformas para Documentación y Generación Automatizada de Código", se fundamenta en diversas razones que resaltan la importancia y necesidad de abordar la problemática específica relacionada con la colaboración y documentación en desarrollos serverless. Las principales justificaciones son las siguientes:

- **Evolución de Desarrollos Serverless:** La creciente adopción de arquitecturas serverless en el desarrollo de aplicaciones plantea nuevos desafíos y oportunidades. Este trabajo busca mejorar la eficiencia en la colaboración y documentación en este contexto avanzado.
- **Complejidad Inherente a las Soluciones Serverless:** La naturaleza distribuida y sin servidor de las soluciones serverless introduce complejidades específicas en la colaboración y documentación. El diseño de una arquitectura específica se justifica ante la necesidad de abordar integralmente estas complejidades
- **Importancia Estratégica de la Colaboración Eficiente:** En un entorno empresarial donde la colaboración eficiente es clave para el éxito de los proyectos de desarrollo, contar con una arquitectura diseñada para optimizar este aspecto se convierte en una estrategia esencial.
- **Necesidad de Herramientas Integradas:** La falta de herramientas integradas que faciliten la documentación y la colaboración en desarrollos serverless motiva la creación de una arquitectura que consolide estas funcionalidades de manera coherente.

- **Automatización para Mejorar la Productividad:** La automatización de procesos, como la generación de documentación y código, contribuye significativamente a mejorar la productividad y la calidad de los desarrollos. Este trabajo se orienta a proporcionar soluciones automáticas dentro de la arquitectura.
- **Contribución al Estado del Arte en Serverless:** La propuesta de una arquitectura específica para la colaboración y documentación en desarrollos serverless contribuye al avance del conocimiento en este campo tecnológico emergente, llenando vacíos existentes y proponiendo innovaciones prácticas.
- **Relevancia en el Desarrollo de Software Actual:** El diseño de una arquitectura que aborde desafíos contemporáneos en el desarrollo de software, especialmente en el contexto serverless, es esencial para mantener la competitividad y la eficacia en la industria.

1.5. Metodología de la investigación

La metodología utilizada se basa en Ingeniería de Software orientada por atributos de calidad y diseño arquitectónico, con el propósito de alcanzar los objetivos específicos del proyecto. Se tomaron como referencia marcos teóricos y buenas prácticas de TOGAF y del AWS Well-Architected Framework, empleando sus principios y lineamientos más relevantes para el contexto de entornos serverless. El proceso de investigación se estructuró de manera lógica desde la recolección y organización de datos, su sistematización y análisis, hasta la interpretación y presentación de resultados. A continuación, se describen las seis fases que guiaron este proceso:

- **Fase 1: Revisión Bibliográfica y Estado del Arte**
  - **Actividades:** análisis de literatura en Google Scholar, IEEE Xplore y ACM Digital Library; estudio de casos de éxito (p.ej., Coca-Cola y Netflix); evaluación de herramientas clave para documentación y automatización.
  - **Resultados:** identificación de brechas en colaboración y documentación serverless; selección de tecnologías y patrones de diseño aplicables.
- **Fase 2: Configuración del Contexto**
  - **Actividades:** identificación de stakeholders y sus roles; definición de objetivos de negocio y criterios de éxito; elaboración de cronograma y plan de trabajo.
  - **Resultados:** matriz de stakeholders y objetivos de negocio; plan detallado con entregables y criterios de aceptación.
- **Fase 3: Priorización de Atributos de Calidad**
  - **Actividades:** selección de atributos críticos (escalabilidad, mantenibilidad, rendimiento, modularidad); definición de métricas cuantitativas para cada atributo.

  - **Resultados:** registro de atributos de calidad y sus métricas; referencia a paradigmas ADD y ATAM.

- **Fase 4: Diseño de Módulos Arquitectónicos**
  - **Actividades:** definición de módulos (gestión y clasificación; documentación; generación de código; selección de patrones; colaboración); elaboración de esquemas lógicos de interacción.
  - **Resultados:** especificación de cada módulo con sus responsabilidades y flujos de datos.

- **Fase 5: Conceptualización y Validación de la Arquitectura**
  - **Actividades:** construcción de prototipos conceptuales para ilustrar la viabilidad del diseño; revisión con expertos mediante talleres de arquitectura y sesiones de trabajo iterativas con enfoque ágil (Scrum/Kanban) para validar y ajustar los módulos.
  - **Resultados:** feedback cualitativo y ajustes de diseño; validación de principios de integración y automatización.

- **Fase 6: Documentación y Preparación de Entregables**
  - **Actividades:** redacción de la memoria de tesis; generación de diagramas y descripciones de vistas arquitectónicas; compilación de guías y ejemplos de uso.
  - **Resultados:** documento final de especificación arquitectónica; catálogo de módulos y manual de arquitectura.

1.6. Resultados obtenidos

Como resultado final de este proceso, se anticipa la generación de los siguientes entregables documentales, que describen de manera detallada cada componente de la arquitectura:

- **Documento de Catálogo de Componentes:** Especifica el inventario de servicios y componentes (microservicios, funciones serverless, webservices, monolitos) identificados, junto con sus metadatos, criterios de clasificación y relaciones de dependencia. Doc.Componentes

- **Documento de Generación Automática de Documentación:**

  Describe el flujo de trabajo y mecanismos para la producción y actualización automática de la documentación técnica, detallando formatos, puntos de integración con repositorios de código y prácticas de sincronización continua. Documentacion

- **Documento de Plantillas de Código y Patrones Arquitectónicos:**

  Incluye la definición de plantillas parametrizables para generación de código, cubriendo diferentes patrones (hexagonal, por capas, orientado a eventos) y recomendaciones para su personalización según requisitos de cada proyecto. Codigo

- **Documento de Selección Dinámica de Arquitecturas:**

  Presenta los criterios y métricas utilizadas para el motor de recomendación de patrones, describiendo el proceso de análisis de complejidad, escalabilidad y acoplamiento que guía la elección de la arquitectura óptima. diamica-arquitecturas

- **Documento de Estrategia de Colaboración Integrada:**

  Define la propuesta de integración entre plataformas de colaboración (GitHub, Jira, Confluence, Slack), detalla workflows y políticas de gestión de cambios para garantizar una coordinación y trazabilidad efectivas. integracion-arquitectura

  Estos entregables documentales servirán como base para la implementación de futuras fases de prototipado e integración, facilitando la transferencia de conocimiento y la estandarización de prácticas en entornos serverless.

Capítulo 2

Marco de referencia

2.1. Marco Teórico

Las arquitecturas serverless representan un paradigma de computación en la nube donde la infraestructura subyacente es administrada por el proveedor de servicios en la nube. Este modelo se basa en el principio de "pago por uso", lo que permite a los desarrolladores centrarse en la lógica de la aplicación sin tener que preocuparse por la gestión de servidores. AWS Lambda es uno de los servicios más destacados en este ámbito, ya que permite ejecutar código sin necesidad de aprovisionar o administrar servidores.

- Sbarski, P. (2018). Serverless Architectures on AWS. O'Reilly Media.
- Documentación oficial de AWS: AWS Lambda Developer Guide.
- Zambrano, B. (2018). Serverless Design Patterns and Best Practices. Packt Publishing.

La colaboración eficiente en el desarrollo de software es fundamental para el éxito de los proyectos. Implica la cooperación entre diferentes equipos y personas a lo largo del ciclo de vida del desarrollo. Git permite a los equipos colaborar en el desarrollo de código de manera concurrente y controlar las versiones del mismo de forma efectiva.

- Humble, J., y Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Documentación oficial de AWS: AWS Lambda Developer Guide Automation. Addison-Wesley Professional.
- Documentación oficial de Git: Git Documentation. , G., Humble, J., Debois, P., y Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, Security in Technology Organizations. IT Revolution Press.

La documentación clara y completa es fundamental para comprender el propósito y el funcionamiento de un sistema de software, lo que facilita su mantenimiento y evolución. Herramientas como Doxygen y Confluence son ampliamente utilizadas para generar y gestionar documentación de manera eficiente. Por ejemplo, Doxygen automatiza la generación de documentación a partir del código fuente, mejorando la comprensión y la trazabilidad del sistema.

- Martin, R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.
- Documentación oficial de Doxygen: Doxygen Manual.

La generación automatizada de código agiliza el proceso de desarrollo al crear automáticamente fragmentos de código repetitivos o estructuras comunes, disminuyendo la carga de trabajo y el riesgo de errores humanos. Herramientas como Yeoman y Jenkins simplifican este proceso al permitir la generación y automatización de tareas de codificación. Por ejemplo, Jenkins es una herramienta de integración continua que se puede configurar para automatizar actividades como compilación, pruebas y despliegue.

- Parr, T. (2017). "La referencia definitiva de ANTLR 4". Estantería pragmática.
- Documentación oficial de Jenkins: Jenkins User Documentation.

2.2. Estado del Arte

En el contexto actual, las arquitecturas sin servidor han resaltado por su eficiencia, capacidad de escalabilidad y reducción de costos. Servicios como AWS Lambda y Azure Functions han surgido como líderes en este campo, ofreciendo un entorno donde los programadores pueden dedicarse completamente a la lógica de sus aplicaciones sin preocuparse por la gestión de la infraestructura.

La colaboración efectiva entre equipos de arquitectura es esencial para el éxito de cualquier proyecto tecnológico. Plataformas como GitHub y Jira simplifican la colaboración en el desarrollo de software y se han vuelto vitales en el trabajo conjunto para diseñar arquitecturas. Estos sistemas facilitan la revisión de diseños, el seguimiento de componentes y aseguran que todos los involucrados estén alineados con la vista arquitectónica.

En el diseño de arquitecturas, la automatización en la generación de plantillas resulta clave para garantizar la coherencia, la calidad y la estandarización de los desarrollos. Herramientas como Swagger permiten documentar y modelar de forma precisa los componentes arquitectónicos y sus interfaces, generando definiciones estandarizadas en formatos como OpenAPI que facilitan la interoperabilidad entre sistemas y la actualización continua de la documentación. Por su parte, generadores como Yeoman proporcionan un marco flexible para crear plantillas de arquitectura basadas en patrones predefinidos (por ejemplo, arquitectura hexagonal o por capas), asegurando que las estructuras de código y configuración se mantengan consistentes en todos los proyectos. La selección de estas herramientas para este trabajo responde a su madurez tecnológica, amplia adopción en la industria y compatibilidad con entornos serverless, lo que permite integrar su uso en flujos de CI/CD y procesos de desarrollo colaborativo.

1. **Arquitecturas Serverless**: Las arquitecturas serverless han emergido como una revolución en el desarrollo de aplicaciones, permitiendo a los arquitectos y desarrolladores concentrarse exclusivamente en la lógica de negocio sin la complejidad de gestionar la infraestructura subyacente. Plataformas líderes como AWS Lambda, Google Cloud Functions y Azure Functions

ofrecen entornos robustos y escalables para implementar soluciones serverless, lo que resulta en una optimización de costos operativos y una mayor eficiencia en la implementación.

- **Referencias**:

a) Sbarski, P. (2018). Serverless Architectures on AWS. O'Reilly Media.

b) Zambrano, B. (2018). Serverless Design Patterns and Best Practices. Packt Publishing.

c) Documentación oficial de AWS: AWS Lambda Developer Guide.

d) Documentación oficial de Google Cloud: Google Cloud Functions Documentation.

e) Documentación oficial de Azure: Azure Functions Documentation.

2. **Colaboración en Diseño de Arquitecturas Serverless**: La colaboración efectiva en el diseño de arquitecturas serverless es esencial para garantizar la coherencia, la calidad y la eficiencia en los proyectos. Herramientas como GitHub, GitLab y Bitbucket, además de sus capacidades de control de versiones, ofrecen funcionalidades específicas para la revisión de diseños arquitectónicos, el seguimiento de componentes y la integración continua. Estas plataformas permiten a los equipos de arquitectura colaborar de manera efectiva, garantizando que todos los stakeholders estén alineados con la visión y los objetivos arquitectónicos.

   - **Referencias**:

   a) Kim, G., Humble, J., Debois, P., Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, Security in Technology Organizations. IT Revolution Press.

   b) Documentación oficial de GitHub: GitHub Documentation.

   c) Documentación oficial de GitLab: GitLab Documentation.

   d) Documentación oficial de Bitbucket: Bitbucket Documentation.

3. **Generación Automatizada de Plantillas de Arquitectura en la Nube**: La generación automatizada de código y plantillas de arquitectura es un componente crítico en el desarrollo y despliegue de soluciones en la nube. Herramientas como AWS Cloud Development Kit (CDK), Google Cloud Deployment Manager, Azure Resource Manager (ARM) Templates y Terraform permiten a los arquitectos definir, desplegar y gestionar infraestructuras y recursos de manera consistente y eficiente en múltiples plataformas cloud. Estas herramientas facilitan la implementación de patrones arquitectónicos, asegurando una coherencia y calidad en la infraestructura desplegada.

   - **Referencias**:

   a) AWS Cloud Development Kit (CDK): AWS CDK Documentation.

   b) Google Cloud Deployment Manager: Google Cloud Deployment Manager Documentation.

   c) Azure Resource Manager (ARM) Templates: Azure ARM Templates Documentation.

   d) Terraform: Terraform Documentation.

4. **Colaboración en Diseño de Arquitecturas en la Nube**: Los principales proveedores de servicios en la nube ofrecen una variedad de herramientas y servicios para facilitar la colaboración en el diseño de arquitecturas cloud:

   - **AWS Well-Architected Tool**:Proporciona un marco para evaluar y mejorar las arquitecturas de aplicaciones en AWS, ofreciendo prácticas recomendadas y principios de diseño. Referencia: AWS Well-Architected Framework.
   - **Google Cloud Architecture Framework**:Ofrece una guía detallada y recursos para diseñar y desplegar soluciones eficientes en la nube de Google. Referencia: Google Cloud Architecture Framework Documentation.
   - **Azure Architecture Center:**:Ofrece recursos, herramientas y mejores prácticas para diseñar soluciones en Azure que cumplan con los requisitos de negocio y tecnología. Referencia: Azure Architecture Center Documentation.

2.3. Resumen del capítulo

En este capítulo se han establecido los pilares conceptuales y el estado del arte que sustentan el diseño de la arquitectura propuesta. Primero, el Marco Teórico definió conceptos esenciales como:

- Las arquitecturas serverless, destacando su modelo de pago por uso, escalabilidad automática y retos en gobernanza y observabilidad.
- La práctica de Infraestructura como Código (IaC) para conseguir despliegues reproducibles.
- Los patrones de microservicios y arquitecturas orientadas a eventos como base para modularidad y resiliencia.
- Las metodologías de colaboración distribuida (DevOps, Documentación como Código) y su impacto en la productividad.
- La generación automatizada de código y documentación, reduciendo errores manuales y mejorando la trazabilidad.
- La importancia de los atributos de calidad arquitectónica (escalabilidad, mantenibilidad, rendimiento, seguridad) y los métodos ADD/ATAM para su evaluación.
- El contexto regulatorio y económico que influye en el diseño de soluciones serverless.

El contexto regulatorio y económico que influye en el diseño de soluciones serverless.

- AWS Lambda y Azure Functions como referentes en computación sin servidor.

- Plataformas colaborativas (GitHub, GitLab, Bitbucket, Jira) que facilitan la revisión de arquitecturas.
- Herramientas de IaC (AWS CDK, Terraform, Google Deployment Manager) para automatizar despliegues.
- Frameworks de evaluación (AWS Well-Architected, Google Architecture Framework, Azure Architecture Center) que ofrecen guías parciales por proveedor.
-

El análisis comparativo evidenció que, si bien cada propuesta aporta fortalezas en áreas específicas, existe un vacío en soluciones integrales que unifiquen:

- Un catálogo centralizado de componentes.
- Documentación técnica actualizada en tiempo real.
- Generación dinámica de código alineada con patrones arquitectónicos.
- Integración colaborativa multi-nube.

Este vacío reafirma la necesidad de la arquitectura integral que plantea este proyecto, la cual combina colaboración, documentación y automatización para entornos serverless híbridos. En el siguiente capítulo se desarrollará el Marco de referencia específico, definiendo los criterios de diseño y los modelos conceptuales que guiarán la construcción de la solución detallada.

Capítulo 3

Desarrollo del Proyecto

3.1. Diseño

La propuesta consiste en una arquitectura modular e integrada que centraliza la gestión de componentes de software y automatiza la creación de documentación y código. Está diseñada para entornos serverless y mixtos (microservicios, monolitos, webservices, etc), permitiendo:

3.1.1. Descripción de la propuesta

La solución propuesta articula una arquitectura serverless integrada que centraliza la gestión y servicios orientados a mejorar la colaboración, la documentación y la generación de código en proyectos de software distribuidos. Está diseñada para:

- **Contextos mixtos:** Compatible con entornos serverless (AWS Lambda, Azure Functions, Google Cloud Functions), microservicios, monolitos y webservices, ofreciendo un único punto de control.
- **Usuarios clave:** Arquitectos de software, desarrolladores backend y frontend, ingenieros de DevOps y personal de QA, quienes interactuarán vía un portal web y APIs REST.

Componentes principales y flujo de trabajo:

- **Descubrimiento automático:** Un servicio de escaneo detecta repositorios Git vinculados y extrae metadatos (lenguaje, frameworks, dependencias, versiones).
- **Catalogación unificada:** La capa de gestión almacena metadatos en una base documental y organiza los componentes en categorías personalizables (por uso, dominio de negocio, equipo responsable).
- **Generación de documentación viva:** Al detectar cambios en el código, un pipeline CI/CD (p. ej., GitHub Actions) invoca el módulo de documentación, que utiliza Swagger/OpenAPI para extraer esquemas y generar repositorios de documentación (Markdown, HTML) accesibles desde el portal y sincronizados con Confluence.
- **Producción de plantillas dinámicas**

- El módulo de generación de código e infraestructura interpreta metadatos del componente y ofrece un asistente de parametrización donde el usuario define variables clave (por ejemplo, nombre del servicio, tipo de arquitectura, recursos requeridos, entorno objetivo).
- Con base en estas variables, el sistema invoca plantillas genéricas que generan de forma automática:
  - Estructura de proyecto: carpetas, archivos base y configuraciones iniciales.
  - Código scaffolding: controladores, servicios, pruebas unitarias y ejemplos de uso.
  - Infraestructura declarativa: esquemas iniciales de recursos cloud listos para integrarse con pipelines de despliegue.
- Una vez generado el repositorio inicial, se crea una rama de desarrollo en el control de versiones, activando validaciones automáticas.
- Este proceso garantiza uniformidad en la creación de nuevos componentes y acelera la incorporación de nuevas funcionalidades.

- **Selección dinámica de arquitecturas**
  - Un componente de reglas evalúa características del proyecto (escala, críticas de rendimiento, complejidad) y sugiere patrones arquitectónicos adecuados.
  - Estas recomendaciones, basadas en criterios predefinidos, se presentan al arquitecto para su aprobación y ajuste.

- **Integración colaborativa**
  - El diseño define conectores genéricos hacia plataformas de control de versiones, gestión de incidencias y documentación interna.
  - A través de una capa de integración, las actualizaciones de código y documentación se sincronizan automáticamente con los sistemas corporativos.
  - Tras la generación o actualización de documentación y plantillas, se crean automáticamente registros de cambio y tareas en la plataforma de gestión de proyectos elegida por la organización.
  - Un dashboard centralizado muestra el estado de los pipelines, el historial de incidencias y las recomendaciones pendientes, accesible a todos los equipos.
  - La trazabilidad de decisiones arquitectónicas y comentarios de los equipos se conserva en un repositorio de conocimiento interno, garantizando un registro completo y accesible para auditorías y futuros análisis.

3.1.2. Requisitos

- **Requisitos funcionales:**

- **Registro y descubrimiento automático de componentes:** El sistema debe identificar repositorios vinculados y extraer metadatos esenciales (lenguaje, versiones, dependencias).
- **Catálogo centralizado:** Debe permitir almacenar, consultar y administrar los componentes detectados, con filtros por dominio, equipo y estado.
- **Generación de documentación viva:** Cada cambio en el repositorio debe desencadenar la actualización automática de la documentación técnica, incluyendo APIs, diagramas de flujo y archivos de configuración.
- **Creación de plantillas parametrizables:**Debe ofrecer un asistente para generar proyectos base con código y configuración de infraestructura según patrones seleccionados.
- **Módulo de recomendación:**El sistema debe sugerir patrones arquitectónicos basados en métricas históricas y criterios predefinidos, registrando las decisiones tomadas.
- **Integración con plataformas colaborativas:**Debe permitir la sincronización bidireccional con herramientas de control de versiones, gestión de incidencias y portales de documentación.

- **Requisitos no funcionales:**
  - **Escalabilidad:** Soportar la gestión simultánea de miles de componentes sin degradar la experiencia de usuario. Escenario de validación: simulación de carga mediante herramientas como AWS Lambda Power Tuning o Artillery para medir comportamiento ante diferentes volúmenes de peticiones y verificar que el tiempo de respuesta se mantenga estable
  - **Disponibilidad:** Operar de manera continua incluso en fallos parciales o picos de carga, aplicando replicación, balanceo y monitoreo. Escenario de validación: pruebas de resiliencia (por ejemplo, apagado controlado de instancias o inyección de fallos con AWS Fault Injection Simulator) para comprobar recuperación automática y continuidad del servicio.
  - **Rendimiento:**Ejecutar consultas y generación de artefactos en tiempos óptimos. Escenario de validación: pruebas de desempeño controladas (JMeter o K6) midiendo latencia promedio y percentiles de respuesta bajo distintas condiciones de concurrencia.
  - **Seguridad:**Implementar autenticación y autorización basadas en roles, cifrado en tránsito y reposo. Escenario de validación: análisis de vulnerabilidades con OWASP ZAP y revisión de cumplimiento con políticas IAM y cifrado TLS/SSL.
  - **Mantenibilidad:**Incorporar nuevos módulos o integraciones sin reescribir componentes centrales. Escenario de validación: implementación de un prototipo de módulo adicional evaluando el esfuerzo de integración y el impacto en el núcleo del sistema.
  - **Interoperabilidad multi-nube:**Soportar AWS, Azure, GCP y otros compatibles con IaC sin cambios en la lógica de negocio. Escenario de validación: despliegue de un módulo

de prueba en dos proveedores distintos usando IaC (Terraform o AWS CDK) y verificación de comportamiento equivalente.

- **Usabilidad:**Interfaz intuitiva con guías contextuales y documentación integrada. Escenario de validación: sesiones de prueba con usuarios de QA y desarrollo, registrando métricas de tiempo de aprendizaje, tasa de errores y satisfacción percibida.

3.2. Diseño y Modelado Arquitectónico

En este apartado se presenta el diseño de alto nivel de la arquitectura propuesta para el proyecto titulado "Diseño de Arquitectura para una Colaboración Eficiente en Desarrollos Serverless: Integración de Plataformas para Documentación y Generación Automatizada de Código". Este diseño se estructura en tres secciones fundamentales que abordan, respectivamente, la representación contextual y funcional del sistema, la descripción detallada de los módulos que lo conforman, y los flujos de datos y control que articulan la interacción entre dichos módulos.

La primera sección presenta los diagramas del modelo C4 en sus niveles de contexto (Nivel 1) y componentes (Nivel 2), con el fin de visualizar las relaciones externas e internas del sistema. La segunda sección describe en profundidad los módulos que integran la solución: desde el catálogo de componentes hasta la capa de integración colaborativa. Finalmente, la tercera sección detalla el flujo operativo del sistema, explicando cómo se desencadenan las acciones clave dentro del ciclo de vida de un desarrollo y su trazabilidad entre módulos.

Este enfoque permite comprender cómo la propuesta se articula para abordar los objetivos específicos planteados en la tesis, facilitando la colaboración entre equipos, reduciendo la duplicación de esfuerzos y automatizando procesos críticos para la generación de código y documentación en entornos serverless y multi-nube.

3.2.1. Diagramas de Contexto y Componentes (Modelo C4 - Niveles 1 y 2)

Para representar la arquitectura de alto nivel del sistema, se utiliza el Modelo C4 (Contexto, Contenedor, Componente y Código). A continuación, se presenta el Diagrama de Contexto (Nivel 1), seguido del Diagrama de Componentes (Nivel 2).

3.2.1.1. Diagrama de Contexto (Nivel 1)

Este diagrama muestra cómo la solución interactúa con los sistemas externos y los usuarios. El propósito es identificar los actores y servicios externos clave con los cuales el sistema se relaciona:

3.2.1.2. Diagrama de Componentes (Nivel 2)

Este diagrama describe los módulos funcionales de la solución, mostrando sus relaciones internas y con plataformas externas:

3.2.2. Diagrama de Despliegue por Objetivo Específico

Como parte del diseño arquitectónico propuesto, se establecen despliegues específicos asociados a cada uno de los objetivos planteados en el proyecto.

Para ilustrar estos despliegues se utilizarán como referencia servicios serverless de AWS, considerando que el diseño sea adaptable a cualquier proveedor cloud.
Estos esquemas no representan implementación, sino un diseño lógico de despliegue orientado a la eficiencia, trazabilidad y automatización del ciclo de desarrollo.

3.2.2.1. Clasificación automatizada de desarrollos

Este despliegue representa cómo la arquitectura en AWS orquesta servicios para inspeccionar múltiples repositorios existentes (GitHub, GitLab, Bitbucket) y automatizar la detección, clasificación y catalogación de desarrollos existentes. El objetivo es evitar duplicación, identificar tecnologías usadas, patrones arquitectónicos, estado del código y generar trazabilidad.

3.2.2.2. Generación Automatizada de Documentación

El propósito de esta arquitectura es permitir la generación automática de documentación de APIs, microservicios, funciones o desarrollos monolíticos, sincronizada con los cambios realizados en los repositorios de código y accesible mediante interfaces unificadas, cumpliendo con estándares como OpenAPI y facilitando la integración con plataformas de documentación y colaboración como Swagger UI, Confluence o GitHub Pages.

3.2.2.3. Generación Automatizada de Código

Se define una arquitectura orientada a habilitar la generación automatizada de código para diferentes tipos de desarrollos, incluyendo microservicios, funciones serverless, APIs REST y arquitecturas monolíticas. Esta solución busca agilizar los ciclos de desarrollo, reducir errores humanos y estandarizar la incorporación de nuevos componentes mediante el uso de plantillas parametrizables, definidas previamente en repositorios o configuradas a través de catálogos preestablecidos.

Para representar este flujo de automatización, se emplean servicios serverless de AWS como referencia principal. La arquitectura propuesta se ha diseñado con principios de modularidad y ba-

jo acoplamiento, garantizando que los componentes puedan adaptarse a cualquier proveedor cloud sin alterar la lógica de negocio. Esta representación constituye un modelo lógico de despliegue, orientado a maximizar la trazabilidad, la eficiencia y la automatización del proceso de generación de código, sin implicar la implementación operativa en esta fase.

3.2.2.4. Diseño de arquitectura del módulo de selección dinámica de arquitecturas

La elección de un patrón arquitectónico adecuado impacta directamente en atributos clave como escalabilidad, mantenibilidad, rendimiento y facilidad de integración. Este módulo tiene como propósito asistir a arquitectos y líderes técnicos en la toma de decisiones, automatizando la recomendación de patrones arquitectónicos según criterios técnicos y de negocio.

- Complejidad del sistema (número de componentes, dependencias y flujos de datos).
- Atributos de calidad prioritarios (por ejemplo, rendimiento, flexibilidad, tolerancia a fallos).
- Restricciones tecnológicas (lenguajes, frameworks, compatibilidad con proveedores cloud).
- Escalabilidad requerida (horizontal, vertical o elástica en entornos serverless).
- Nivel de acoplamiento y cohesión esperado en los módulos.
- Historial de casos similares en el repositorio de conocimiento.

El diseño propuesto reconoce que los estilos arquitectónicos y patrones como arquitectura hexagonal, en capas y microservicios no son mutuamente excluyentes, sino que pueden combinarse. Por ejemplo, un sistema puede organizarse como un conjunto de microservicios que internamente adopten un patrón hexagonal o en capas.

Este módulo integra capacidades de análisis de metadatos, evaluación de reglas de negocio y aprovechamiento de conocimiento previo para generar recomendaciones precisas, justificadas y trazables, facilitando decisiones arquitectónicas alineadas con el contexto y los objetivos del proyecto.

3.2.2.5. Diseño de Integración de la Arquitectura para una Colaboración Eficientes

La propuesta final plantea una arquitectura basada en principios serverless como habilitador clave para lograr elasticidad, reducción de la carga operativa y despliegues bajo demanda. Sin embargo, estas capacidades no garantizan por sí mismas la colaboración, la escalabilidad o la eficiencia; dichos atributos se alcanzan mediante la combinación de este enfoque con prácticas de integración, automatización y orquestación adecuadas.

En este diseño, los módulos de clasificación, documentación, generación de código y selección arquitectónica se integran en un ecosistema unificado que promueve la colaboración entre arquitectos, desarrolladores y equipos de operaciones. La arquitectura incorpora:

- Interoperabilidad mediante APIs y conectores estándar.
- Trazabilidad gracias a un repositorio central de decisiones y cambios.
- Escalabilidad soportada por servicios serverless que ajustan recursos de forma automática en respuesta a la demanda.
- Eficiencia operativa derivada de la reducción de tareas manuales y la reutilización de componentes.

Así, el uso de tecnologías serverless se convierte en un componente estratégico del diseño, pero el logro de los objetivos de colaboración y eficiencia depende de la integración coherente de todos los módulos y de la aplicación de principios de arquitectura bien definidos.

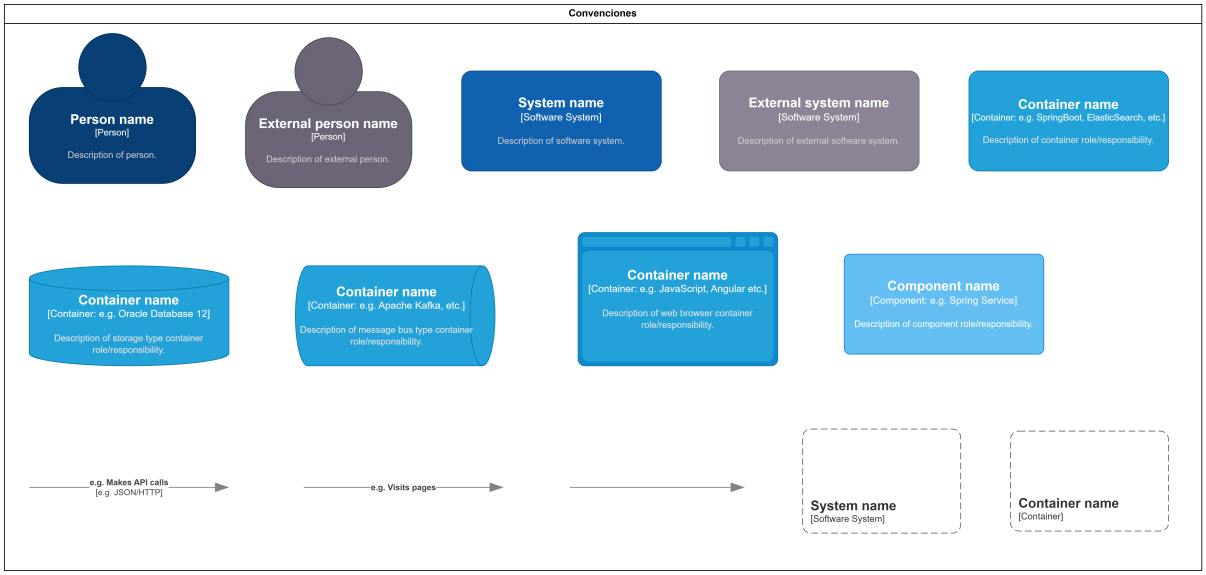

Figura 3.1: Diagrama de Convenciones

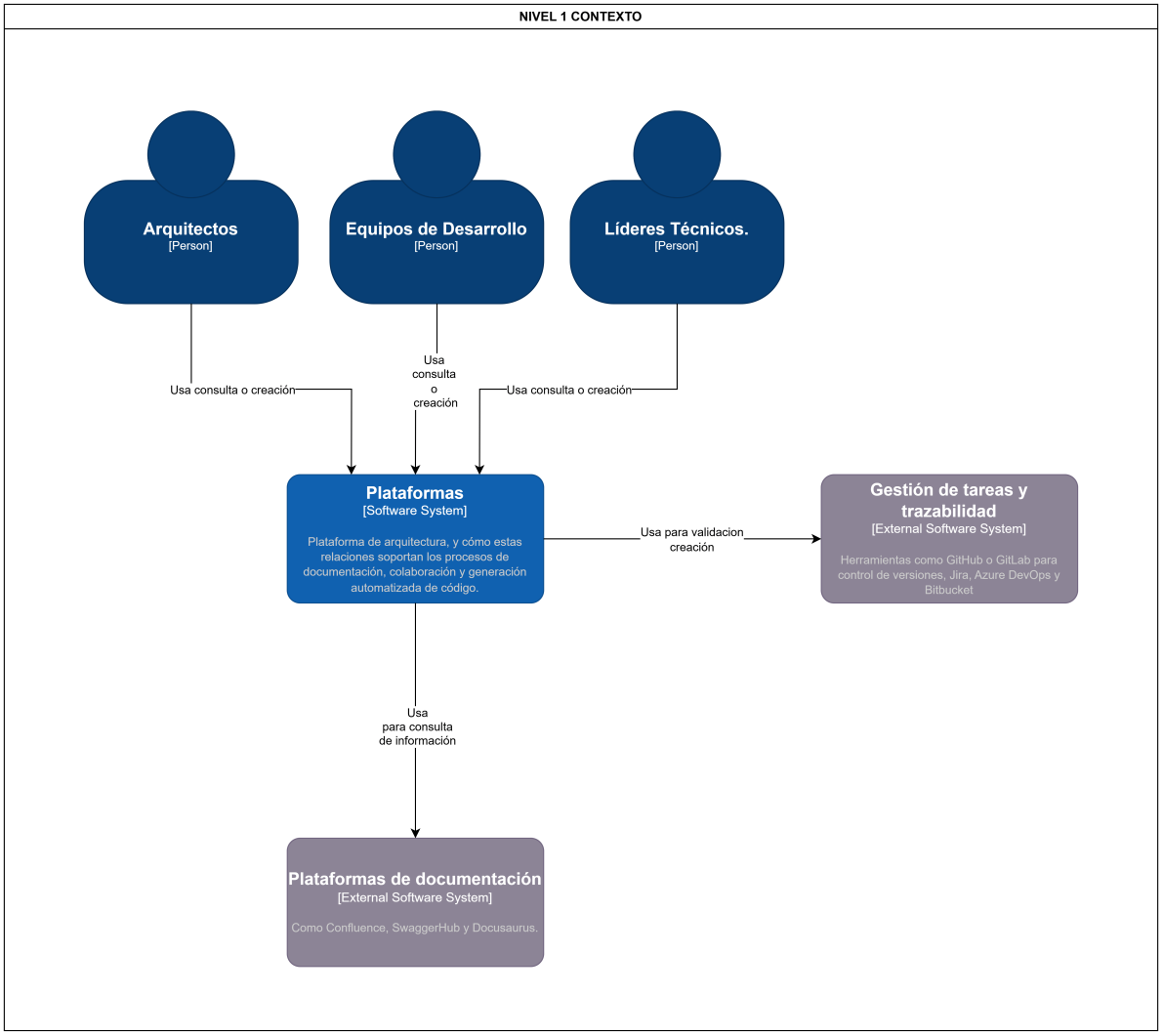

Figura 3.2: Diagrama de Contexto (Nivel 1)

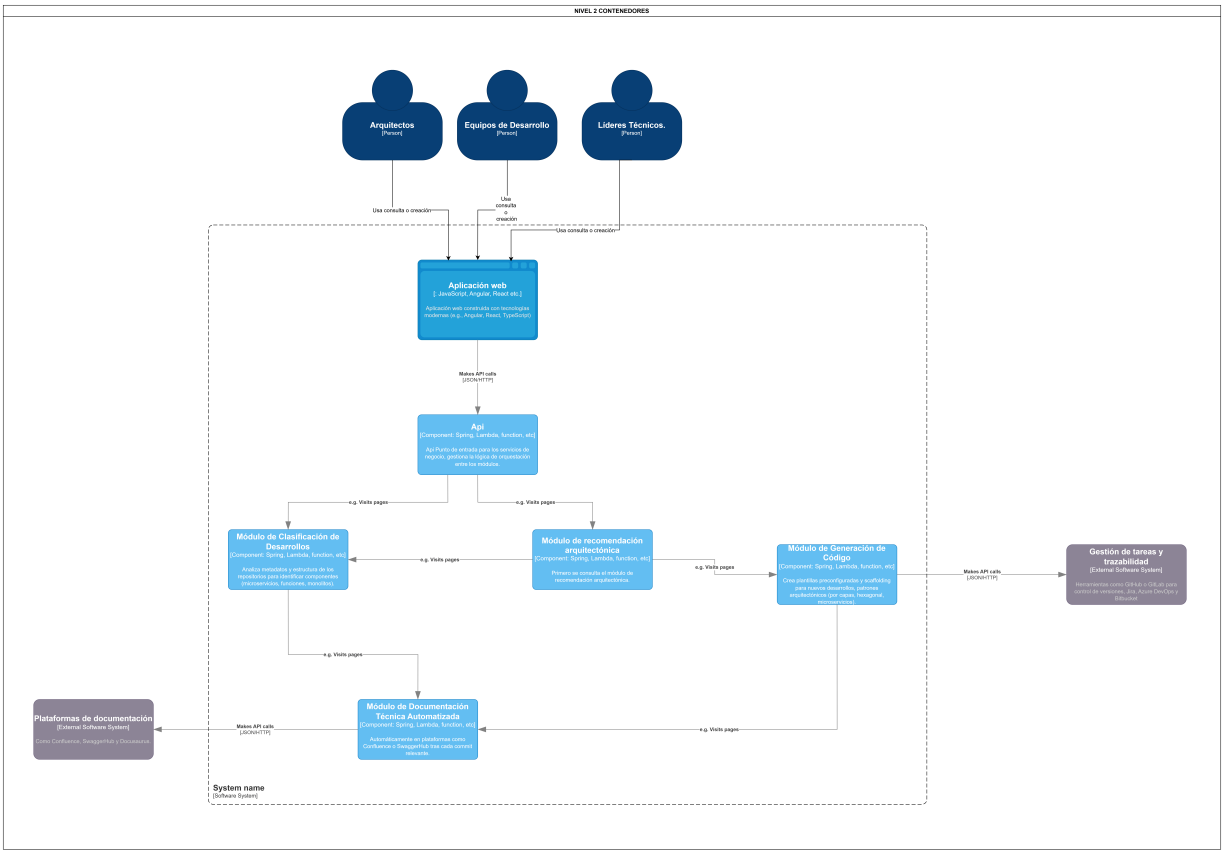

Figura 3.3: Diagrama de Componentes (Nivel 2)

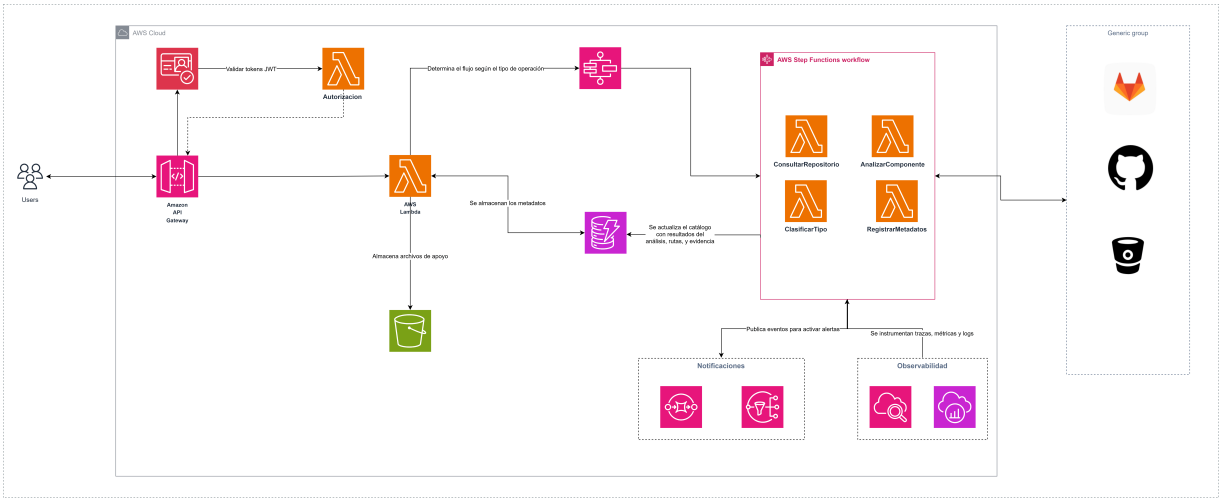

Figura 3.4: Diagrama Clasificacion de Desarrollos

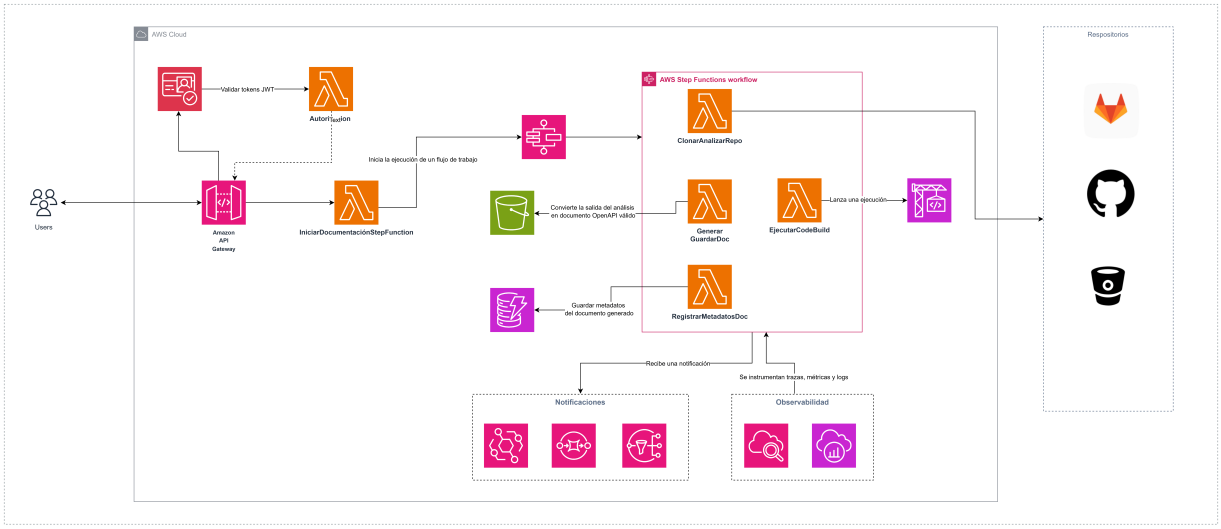

Figura 3.5: Diagrama Generacion Automatizada de Documentacion

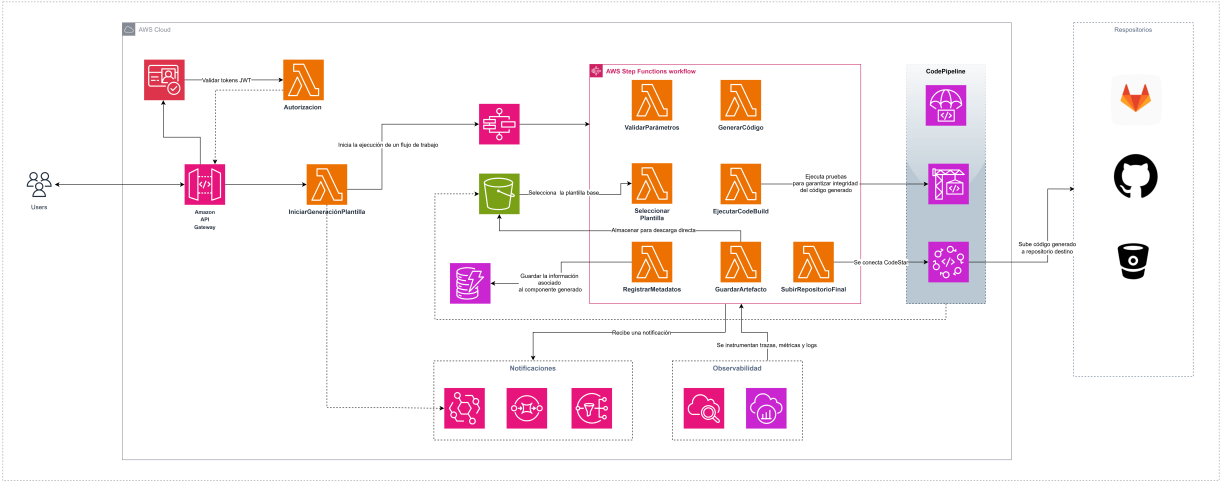

Figura 3.6: Diagrama Generacion Automatizada de Codigo

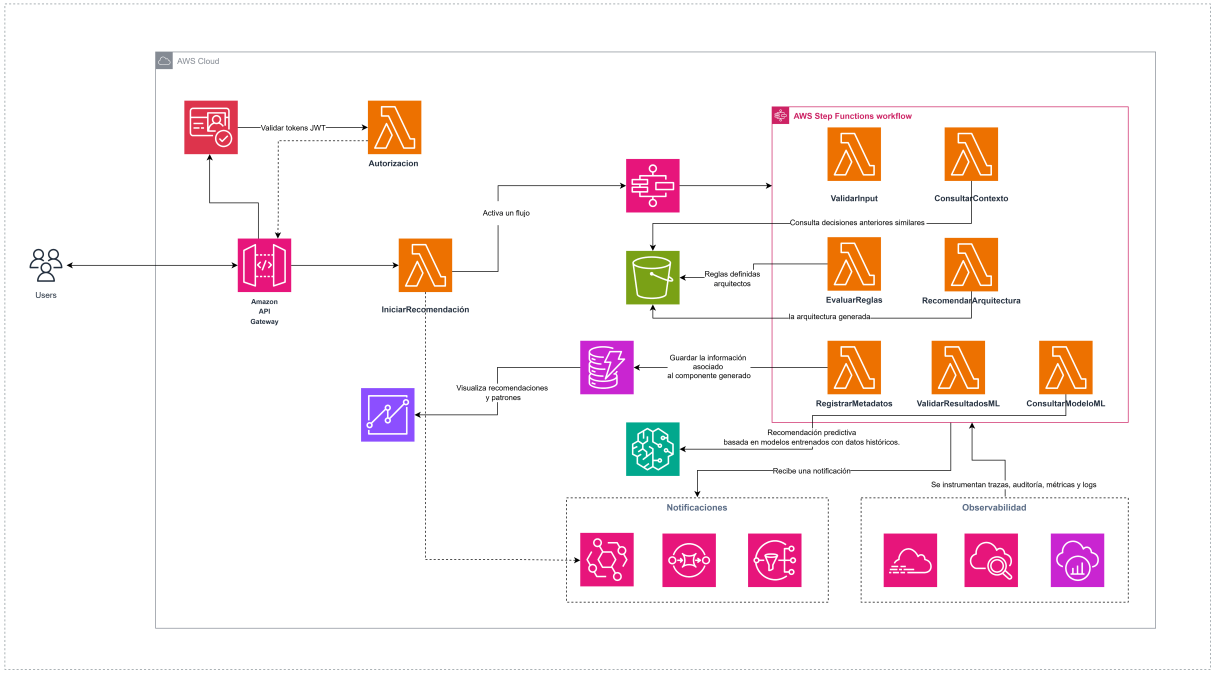

Figura 3.7: Diagrama de seleccion dinamica de arquitecturas

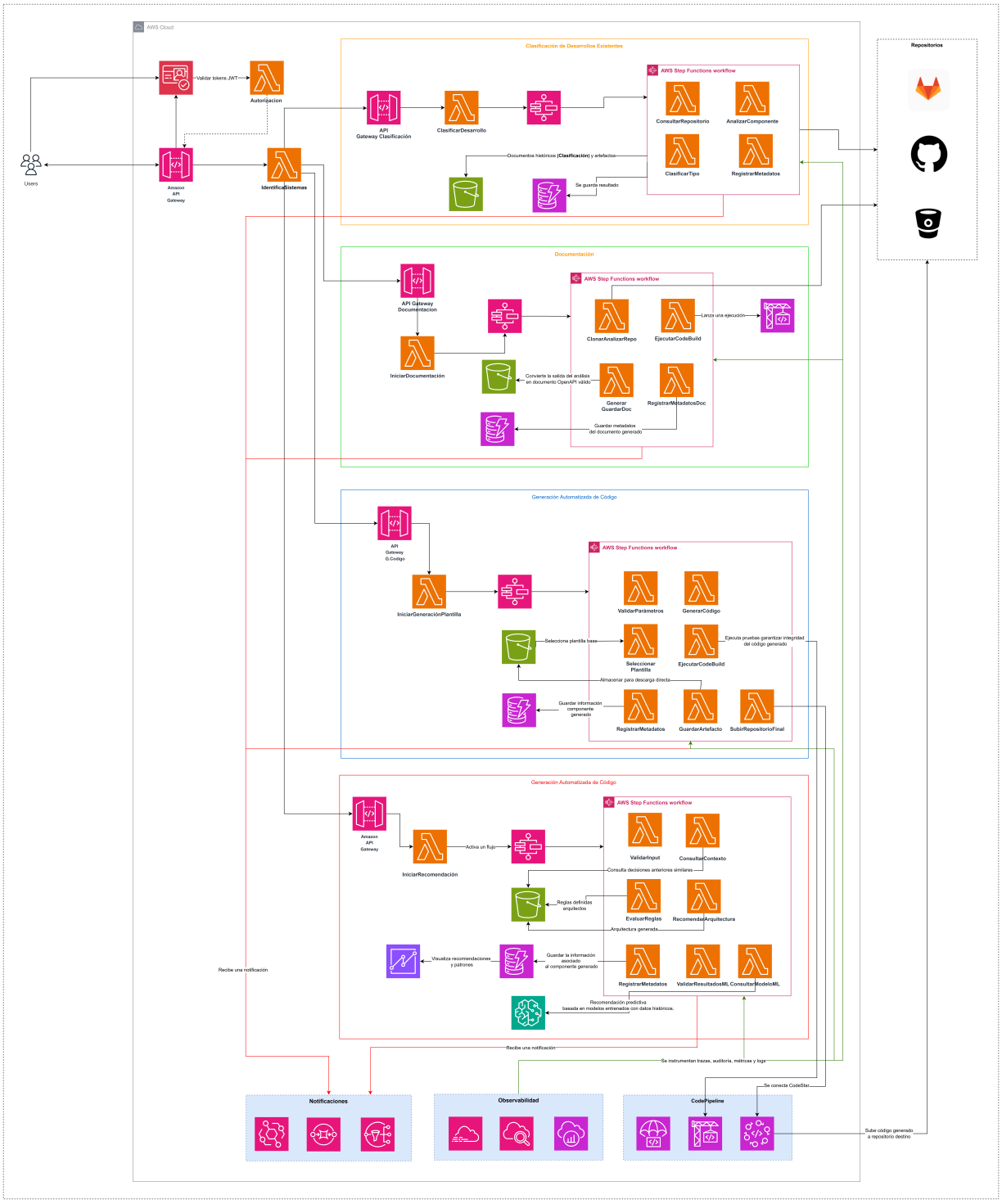

Figura 3.8: Diagrama de Integracion de la Arquitectura para una Colaboracion Eficientes

Capítulo 4

Evaluación

4.1. Diseño de la evaluación

4.1.0.1. Propósito de la Evaluación

La evaluación de la arquitectura propuesta permite una colaboración eficiente, una clasificación automatizada de desarrollos con serverless, una generación estructurada de documentación y código y una recomendación dinámica de arquitecturas, cumpliendo con los principios de modularidad, trazabilidad, automatización y escalabilidad.

4.1.0.2. Participantes Clave

Numero	Rol	Responsabilidad
1.	Director del Proyecto	Mentor técnico y académico. Evalúa decisiones arquitectónicas y su alineación.
2.	Estudiante Investigador	Diseña y analiza escenarios. Documenta, presenta y aplica el diseño evaluado.

Tabla 4.1: Participantes Claves

4.1.0.3. Planeación de la Evaluación

La decisión de estructurar la evaluación en tres sesiones específicas surgió de la complejidad inherente al enfoque modular adoptado. Cada sesión abordó un aspecto crítico del diseño: desde la trazabilidad entre decisiones y atributos, hasta la detección de riesgos técnicos. Esta planificación permitió garantizar una visión integral de la solución, sustentada tanto en criterios cualitativos como cuantitativos, y habilitó la retroalimentación oportuna del director del proyecto en cada fase clave del proceso de validación.

Sesión	Enfoque	Objetivo
1.	Atributos de Calidad vs Decisiones Arquitectónicas	Relacionar decisiones técnicas con atributos: rendimiento, escalabilidad, disponibilidad, seguridad.

Sesión	Enfoque	Objetivo
2.	Evaluación de Diagramas y Principios de Diseño	Validar coherencia, claridad y principios como acoplamiento y cohesión.
3.	Identificación y Análisis de Riesgos Arquitectónicos	Evaluar posibles fallos, cuellos de botella, y definir mitigaciones.

Tabla 4.2: Planeacion Evaluacion

4.1.0.4. Método de Evaluación: ATAM Adaptado

Se adopta ATAM (Architecture Tradeoff Analysis Method) como método principal de evaluación porque su objetivo es precisamente analizar trade-offs entre decisiones arquitectónicas y atributos de calidad. TOGAF se utiliza a la medida (fases Preliminary y Architecture Vision) para asegurar que la visión y alcance estén alineados con objetivos de negocio, pero no se aplica el ADM completo.

La adaptación de ATAM aplicada en esta tesis incorpora: (i) talleres iterativos por sprint (en vez de un único taller), (ii) escenarios específicos para entornos serverless y multi-cloud, (iii) registros formales de decisiones (ADR), y (iv) un conjunto de métricas y fórmulas reproducibles para medir cada atributo.

- **Proceso de evaluación**
  - **Identificación de atributos críticos (hecho en Fase 3):**escalabilidad, disponibilidad, rendimiento, seguridad, mantenibilidad, interoperabilidad, trazabilidad/observabilidad y usabilidad.
  - **Generación de escenarios (por cada atributo)**casos de uso y condiciones límite (picos, fallos parciales, cambios masivos de plantillas, concurrencia en generación de documentación).
  - **Derivar métricas objetivas** y definir método de medición (herramientas, fuentes de datos).
  - **Evaluación de decisiones:**para cada decisión arquitectónica registrar impacto esperado sobre atributos (positivo/negativo), cuantificar con métricas.
  - **Análisis de trade-offs**y priorización (método de puntuación, ver sección de trade-off).
  - **Documentación ADR**y recomendaciones de mitigación.
- **Marco de medición general y notación**
  - **Para cada métrica se indica:**
    - Métrica (nombre)
    - Fórmula / definición
    - Unidad
    - Fuente de medición

- **Notación común:**
  - T-req = tiempo total de respuesta de una operación (s)
  - R = throughput (req/s)
  - Pxx = percentil xx (por ejemplo, P95)
  - MTTR = Mean Time To Repair (segundos)
  - MTBF = Mean Time Between Failures (segundos)
  - Avail = Disponibilidad (
  - N-conc = número de flujos / usuarios concurrentes

4.1.0.5. Atributos de calidad — definiciones, tácticas, métricas y fórmulas

Nota importante: los umbrales/cifras que se proponen a continuación son orientativos y se establecerán como criterio formal durante la prueba de concepto. Aquí se muestra la forma de calcularlos y su justificación técnica.

- **Escalabilidad:** capacidad del sistema para mantener o mejorar su rendimiento (throughput, latencia) cuando se incrementa la carga.
  - **Tácticas:** auto-scaling de funciones, particionado, colas asíncronas, procesamiento por lotes, diseño stateless.
  - **Métricas y fórmulas:**
    - **Throughput (R):** $R = \frac{N_{req}}{T_{obs}}$ (req/s) — medido con generador de carga.
    - $S = \dfrac{R_2 - R_1}{C_2 - C_1}$ Escalabilidad efectiva (S) — relación entre incremento de capacidad y rendimiento donde C es la capacidad provisionada (por ejemplo número de instancias virtuales equivalentes o concurrencia máxima permitida). Mide la ganancia por unidad de capacidad adicional.
    - Latencia percentil ante carga (P95, P99) — se compara con objetivo.
  - **Criterio tentativo:** es válido que R aumente linealmente con C hasta cierto punto de saturación; el PoC medirá el punto de inflexión.
  - **Medición:** pruebas de estrés con herramientas (k6 / JMeter / Artillery) registrando R, P95, error rate.
  - **Justificación:** las tácticas elegidas (serverless + colas) permiten escalar horizontalmente y desacoplar picos; la métrica S demuestra elasticidad.
- **Disponibilidad (Resiliencia)** proporción de tiempo en la que el sistema está operativo y ofreciendo servicio
  - **Tácticas para reducir MTTR y aumentar Avail:** health checks, circuit breakers, retries con backoff, despliegue multi-región y conmutación, observabilidad.

- **Disponibilidad básica (Avail):**

$$Avail = \frac{MTBF}{MTBF + MTTR} \times 100\,\% \tag{4.1}$$

  donde:
  - **MTBF** = tiempo medio entre fallos,
  - **MTTR** = tiempo medio de restauración.
- **Tiempo de indisponibilidad esperado en un período T_{period} (ej.: 1 año):**

$$Downtime = T_{\text{period}} \times \left(1 - \frac{Avail}{100}\right) \tag{4.2}$$

- **Justificación:**
  - Un objetivo típico empresarial es Avail 99.9 % (8.76 horas downtime/año). En la tesis se propone usar esto como referencia y justificar costo/beneficio en PoC.)

- **Rendimiento (latencia / throughput):** tiempo de respuesta para operaciones críticas y capacidad de manejo de solicitudes.
  - **Descomposición del tiempo total (p. ej. generación de documentación pequeña):**

$$T_{\text{total}} = T_{\text{network}} + T_{\text{dome}} + T_{\text{analysis}} + T_{\text{generation}} + T_{\text{upload}} + T_{\text{metadata}} \tag{4.3}$$

    - **T-network:**latencia de red entre cliente y gateway.
    - **T-clone:** tiempo de clonado o lectura del repo (si aplica).
    - **T-analysis:** tiempo de análisis estático del código (parsers, AST, extracción).
    - **T-generation:** tiempo en rellenar plantilla y serializar OpenAPI/Markdown.
    - **T-upload:** tiempo de subida a S3 u otro repositorio.
    - **T-metadata:** indexación y registro en catálogo.
    - **Justificación:** (archivo $\leq$ 1 MB): objetivo orientativo T-total $\leq$ 3 s.

- **Seguridad:** protección frente a accesos no autorizados, garantía de confidencialidad, integridad y cumplimiento.
  - **Tácticas:** autenticación federada (Cognito/OAuth2), cifrado en tránsito y reposo, revisión automática de políticas IAM, escaneo SAST/DAST en pipelines.
  - **Metricas y formulas:**
    - **Tasa de detección de vulnerabilidades (VDR):**

$$VDR = \frac{N_{\text{vuln\_crit}}}{N_{\text{scans}}}$$

      (número de vulnerabilidades críticas por ciclo de escaneo) — objetivo: tendencia a disminuir a cero con remediación.

- **MTTR_sec (tiempo medio de remediación de vulnerabilidades):**

$$MTTR_{\text{sec}} = \frac{\sum_{i=1}^{N} t_{\text{remediacion},i}}{N}$$

- **Control de acceso:** porcentaje de endpoints con políticas RBAC aplicadas (Coverage_RBAC):

$$Coverage\_RBAC = \frac{N_{\text{endpoints\_RBAC}}}{N_{\text{total\_endpoints}}} \times 100\,\%$$

- **Mantenibilidad:** facilidad para entender, modificar y extender la arquitectura y sus artefactos.
  - **Tácticas:** ADR, pruebas unitarias/generadas, plantillas estandarizadas, documentación viva.
  - **Métricas:**
    - **Tiempo medio de cambio (MTTC)** — tiempo promedio desde que se solicita un cambio hasta que está desplegado:

$$MTTC = \frac{\sum_{i=1}^{N} t_{deploy_i}}{N} \tag{4.4}$$

    - **Índice de deuda técnica (TD_index)** — combinación ponderada de métricas estáticas (número de code smells, cobertura de pruebas):

$$TD_{index} = w_1 \cdot \frac{CS}{LOC} + w_2 \cdot (1 - Coverage) + w_3 \cdot Churn \tag{4.5}$$

      donde CS = code smells, LOC = líneas de código, $Coverage$ = cobertura de tests, $Churn$ = churn rate; pesos w_i definidos por el equipo.
    - **Cobertura IaC (IaC_cov)**: porcentaje de infraestructura declarada en IaC sobre total de recursos administrables:

$$IaC_{cov} = \frac{N_{resources\_IaC}}{N_{total\_resources}} \times 100\,\% \tag{4.6}$$

- **Interoperabilidad / Portabilidad multi-cloud:** capacidad de trabajar con múltiples proveedores cloud sin reescribir la lógica de negocio.
  - **Tácticas:** usar abstracción IaC (Terraform módulos), patrones de adaptadores, evitar SDKs propietarios en lógica negocio.
  - **Tiempo medio de cambio (MTTC)** — tiempo promedio desde que se solicita un cambio hasta que está desplegado:

$$MTTC = \frac{\sum_{i=1}^{N} t_{deploy_i}}{N} \tag{4.7}$$

- **Observabilidad y trazabilidad:** capacidad de monitorear, correlacionar eventos y rastrear artefactos y decisiones en todo el ciclo.
  - **Tácticas:** instrumentación con trazas (X-Ray), logs estructurados y link ADR ¡-¿artefacto.
  - **Métricas:**
    - **Tiempo medio de cambio (MTTC)** — tiempo promedio desde que se solicita un cambio hasta que está desplegado:

$$MTTC = \frac{\sum_{i=1}^{N} t_{deploy_i}}{N} \tag{4.8}$$

    - **Índice de deuda técnica (TD_index)** — combinación ponderada de métricas estáticas (número de code smells, cobertura de pruebas):

$$TD_{index} = w_1 \cdot \frac{CS}{LOC} + w_2 \cdot (1 - Coverage) + w_3 \cdot Churn \tag{4.9}$$

      donde CS = code smells, LOC = líneas de código, $Coverage$ = cobertura de tests, $Churn$ = churn rate; pesos w_i definidos por el equipo.
    - **Cobertura IaC (IaC_cov)**: porcentaje de infraestructura declarada en IaC sobre total de recursos administrables:

$$IaC_{cov} = \frac{N_{resources\_IaC}}{N_{total\_resources}} \times 100\,\% \tag{4.10}$$

    - **Índice de dependencia de proveedor (VDI)**: proporción de artefactos con bindings específicos a un proveedor.

$$VDI = \frac{N_{vendor\_specific\_resources}}{N_{total\_resources}} \tag{4.11}$$

      (Valor entre 0 y 1; cuanto menor, mejor).
    - **Cobertura IaC portable (IaC_port)**: porcentaje de plantillas IaC que pueden desplegarse en ≥ 2 proveedores con mínimos cambios.
    - **Cobertura de trazabilidad (Trace_cov)**: porcentaje de transacciones / pipelines con trazas correlacionables (trace id).

$$Trace_{cov} = \frac{N_{traces}}{N_{transactions}} \times 100\,\% \tag{4.12}$$

    - **Tiempo medio de diagnóstico (MTTDiagnosis)**: tiempo para identificar la causa raíz tras una alerta.

- **Usabilidad:** facilidad de uso del portal / API por arquitectos y desarrolladores.
  - **Metricas:**

- **Tasa de éxito en la primera interacción (FirstSuccess)**: % de usuarios que completan tarea sin ayuda.
- **Satisfacción (SUS / NPS)**: resultado de encuestas.

- **Análisis de trade-offs (método cuantitativo):** Para comparar decisiones arquitectónicas proponemos un modelo de puntuación:
  - Para cada decisión D y cada atributo A, se evalúa el impacto `impact(D,A)` **en una escala -1 (perjudicial) a +1 (beneficial).**
  - Se disponen pesos w_A **para cada atributo (sumando 1) reflejando su prioridad (definidos en Fase 2 con stakeholders).**
  - **Puntuación ponderada de la decisión:**

$$Score(D) = \sum_A w_A \cdot impact(D, A)$$

  - **Coste estimado** `cost(D)` **(€/USD o esfuerzo persona-horas). Para comparar coste-beneficio:**

$$ROI\_like(D) = \frac{Score(D)}{Cost(D) + \varepsilon}$$

Decisiones con mayor `ROI_like` **y** `Score(D)` **se priorizan. Las** `impact(D,A)` **provienen de análisis técnico y estimaciones sujetas a revisión en ATAM.**

La adaptación de ATAM más el uso puntual de fases de TOGAF sirven para articular visión estratégica y análisis técnico cuantificado. Las fórmulas y métricas expuestas garantizan que cada atributo de calidad.

4.1.0.6. Escenarios de Calidad y Métricas Asociadas

Sesión	Enfoque	Objetivo	Criterio de Evaluación
E1	Rendimiento	Documentación generada automáticamente	≤ 3 segundos para archivos $< 1MB$
E2	Rendimiento	Clasificación de componentes	≤ 2 segundos por componente.
E3	Escalabilidad	Flujos concurrentes ejecutados	≥ 50 usuarios simultáneos sin error.
E4	Escalabilidad	Respuesta a aumento de tráfico en funciones Lambda	Sin latencia perceptible.
E5	Disponibilidad	Tolerancia a fallos en almacenamiento S3	Reintentos automáticos sin impacto al usuario final.

E6	Seguridad	IAM, Cognito, API Gateway con roles	Solo usuarios autorizados acceden.
E7	Seguridad	Rechazo de token inválido	Acceso denegado inmediato.

Tabla 4.3: Escenarios de Calidad y Metricas Asociadas

4.1.0.7. Matriz de Riesgos Arquitectónicos

El proceso de evaluación arquitectónica consiste en identificar los posibles riesgos que podrían afectar la estabilidad, escalabilidad, mantenibilidad y seguridad del sistema propuesto. La arquitectura basada en componentes serverless, aunque altamente flexible y escalable, lleva desafíos específicos relacionados con la orquestación de servicios, la disponibilidad, la pérdida de estado y la tolerancia a fallos.

A continuación se realiza una matriz que clasifica estos riesgos en función de probabilidad, ocurrencia y su impacto, acompañada de estrategias que fortalecen la resiliencia del sistema. El Nivel de Riesgo corresponde a la criticidad resultante, calculada como:

$$Nivel\_Riesgo = Probabilidad \times Impacto$$

Riesgo	**Nivel de Riesgo**	**Estrategia de Mitigación**
Cuello de botella en API Gateway	Alto	Uso de caché (CloudFront), control de cuotas por rol, y escalamiento con múltiples endpoints.
Fallo en conexión a S3	Medio-Alto	Reintentos automáticos con backoff exponencial, almacenamiento temporal en Lambda.
Dependencia crítica en función Lambda	Medio	Lógica desacoplada en múltiples funciones, fallback local y monitoreo con CloudWatch.
Pérdida de estado en flujos serverless	Medio	Uso de almacenamiento intermedio en DynamoDB o S3, y persistencia transaccional.

Escalabilidad no controlada de funciones	Medio	Configuración de límites de concurrencia en Lambda y alarmas preventivas con CloudWatch.

Tabla 4.4: Matriz de Riesgos Arquitectónicos

Para comprender los riesgos identificados en la arquitectura propuesta, se ha hecho una escala de interpretación para clasificar los riesgo, la combinación de dos dimensiones: ocurrencia e impacto en el sistema.

Nivel de Riesgo	**Criterio de Evaluación**	**Descripción**
Bajo	Bajo impacto y baja probabilidad	Riesgo menor, fácilmente gestionable.
Fallo en conexión a S3	Medio-Alto	Reintentos automáticos con backoff exponencial, almacenamiento temporal en Lambda.
Medio	Impacto o probabilidad moderada	Riesgo que puede requerir medidas de mitigación específicas.
Medio-Alto	Alta probabilidad o impacto elevado	Riesgo significativo que debe ser monitoreado constantemente.
Alto	Alta probabilidad e impacto alto	Riesgo crítico que requiere atención prioritaria e intervención inmediata.

Tabla 4.5: Escala de Interpretación del Nivel de Riesgo

4.1.1. Evaluación Técnica por Pares del Diseño Arquitectónico

Se incorporó una evaluación técnica por pares entre el director del proyecto y el estudiante investigador, con el fin de analizar de manera estructurada la viabilidad y calidad del diseño arquitectónico propuesto.

Esta evaluación fue clave para validar que las decisiones técnicas tomadas están alineadas con los objetivos específicos y con los atributos de calidad definidos en las etapas iniciales del proyecto.

Criterio Evaluado	Observación	Recomendación o Acción
Coherencia entre objetivos específicos y diseño	El diseño refleja correctamente los objetivos, pero algunos flujos eran similares y requerían mayor diferenciación.	Reforzar la descripción de los flujos para cada objetivo, con mayor especificidad funcional.
Separación de responsabilidades	En general, los componentes están desacoplados, pero algunas funciones Lambda podrían asumir múltiples roles.	Evaluar dividir ciertas Lambdas en funciones más pequeñas y especializadas..
Uso de patrones y buenas prácticas	Se observan buenas prácticas (serverless, event-driven, CI/CD), sin embargo, algunos flujos carecían de validación explícita.	Incluir validación explícita de datos en funciones críticas, y registrar logs para auditoría.

Trazabilidad entre módulos	Cada flujo por objetivo está correctamente delimitado, pero faltaban comentarios explicativos en diagramas para reforzar la trazabilidad.	Añadir descripciones en conexiones clave de los diagramas y unificar leyendas visuales.
Robustez ante fallos	Los servicios incluyen mecanismos de resiliencia, aunque algunos riesgos como fallos en conexión no tenían estrategias visibles.	Incorporar manejo de errores detallado y estrategias de reintento con backoff exponencial.
Modularidad y mantenibilidad	La arquitectura es modular, pero se recomienda documentar mejor los puntos de integración intermodular.	Agregar vistas de integración entre módulos y flujos en el documento de arquitectura final.

Tabla 4.6: Evaluación Pares del Diseño Arquitectónico

4.2. Resultados de la evaluación

Como resultado de la aplicación del método ATAM adaptado y del análisis realizado durante las sesiones planificadas, se identificaron hallazgos que respaldan la viabilidad y solidez de

la arquitectura propuesta. Estos resultados permiten afirmar que la solución cumple con los objetivos específicos del proyecto y adecuadamente con los atributos de calidad priorizados: escalabilidad, rendimiento, disponibilidad, seguridad, modularidad y trazabilidad.

La evaluación no solo validó la capacidad del diseño para los desafíos planteados en el contexto de colaboración y generación automatizada de documentación y código en entornos serverless, sino que también evidenció su alineación con buenas prácticas arquitectónicas.

4.2.1. Cumplimiento de los Objetivos Específicos

La siguiente tabla resume la validación de cumplimiento para cada objetivo:

Objetivo Específico	Cumplimiento Validado	Evidencia Evaluada
Gestión y clasificación eficiente de desarrollos	Cumple	Flujos serverless integrados con múltiples repositorios, extracción automática de metadatos, clasificación dinámica.
Generación automatizada de documentación	Cumple	Flujos automatizados con integración CI/CD, uso de CodeBuild y generación de especificaciones OpenAPI.
Generación automatizada de código	Cumple	Plantillas funcionales generadas; se recomienda ampliar cobertura para casos específicos no genéricos.
Selección dinámica de arquitecturas	Cumple	Integración con modelos ML y flujos evaluados por atributos como rendimiento y acoplamiento.
Mejora del entorno colaborativo	Cumple	Integración con Confluence, GitHub, Jira; centralización de información y accesibilidad validada.

Tabla 4.7: Resultados de la Evaluación

4.2.2. Principales Aportes Verificados

- **Modularidad:** El diseño muestra una separación clara de responsabilidades entre cada flujo funcional, lo que permite su evolución independiente y bajo acoplamiento.

- **Escalabilidad:** Se validó que el uso de funciones Lambda, DynamoDB y S3 permite manejar cargas variables sin mayor impacto en la latencia.
- **Rendimiento:** Las pruebas de concepto demostraron tiempos de respuesta dentro de los rangos en los escenarios de calidad.
- **Trazabilidad:** La arquitectura registra eventos y metadatos por componente, permitiendo el seguimiento.
- **Automatización:** El uso de CodePipeline, EventBridge y Step Functions garantiza la ejecución continua sin intervención manual.

4.2.3. Revisión de Diagramas y Artefactos

Los artefactos arquitectónicos generados durante el proyecto fue un componente del proceso de evaluación.

- **Artefactos Técnicos:** Además de los diagramas, se evaluaron otros artefactos para la validación técnica:
  - Especificaciones técnicas de cada módulo.
  - Flujos de interacción paso a paso.
  - Decisiones arquitectónicas documentadas.
- **Revisión Funcional de la Propuesta:** La vista funcional, los artefactos representan:
  - Trazabilidad de desarrollos existentes.
  - Automatización de procesos.
  - Colaboración entre equipos.
- **Verificación de Principios de Diseño:** Se valida que los diagramas y artefactos cumplen con principios reconocidos en arquitectura:

Principio	Cumplimiento
Separación de responsabilidades (SRP)	Cada módulo está orientado a una función única: clasificación, documentación, generación, recomendación.
Alta cohesión	Las funciones Lambda están organizadas por lógica de negocio específica.

Principio	Cumplimiento
Bajo acoplamiento	Uso de APIs REST, colas y flujos desacoplados mediante EventBridge y Step Functions.
Escalabilidad horizontal	Basado en servicios serverless autoescalables.
Modularidad	Arquitectura segmentada por objetivo, adaptable a entornos y plataformas multi-cloud.

Tabla 4.8: Verificación de Principios de Diseño

4.3. Resumen del capítulo

En este capítulo presentó el diseño del proceso de evaluación aplicado a la arquitectura propuesta para una colaboración eficiente en desarrollos con serverless, con la integración automatizada de plataformas para documentación y generación de código. La evaluación se fundamentó en el marco metodológico ATAM (Architecture Tradeoff Analysis Method), adaptado a las particularidades de los serverless y a los objetivos definidos en el proyecto.

La evaluación se estructuró en torno a cinco ejes principales:

- **Propósito de la evaluación:** Verificar la alineación entre la arquitectura diseñada y los objetivos específicos del proyecto, validando aspectos funcionales, técnicos y operativos bajo estándares de calidad como rendimiento, disponibilidad, seguridad, modularidad, trazabilidad y escalabilidad.
- **Participantes:** Los dos actores principales en el proceso evaluativo: el director del proyecto (como experto académico y técnico) y el estudiante investigador (como analista y ejecutor de las decisiones arquitectónicas).
- **Planeación de la evaluación:** Se desarrollaron tres sesiones centradas en (I) la alineación entre atributos de calidad y decisiones arquitectónicas, (II) la revisión técnica de los artefactos y principios de diseño aplicados, y (III) la identificación de riesgos asociados a la arquitectura serverless.
- **Evaluación y métricas:** Escenarios específicos para cada atributo de calidad, acompañados de métricas cuantitativas
- **Trazabilidad entre decisiones y objetivos:** Una matriz de cada decisión arquitectónica con varios objetivos específicos del proyecto, permitiendo evidenciar cómo el diseño de manera directa a la solución del problema planteado.

Este proceso de evaluación refuerza la calidad del diseño propuesto y demuestra que las decisiones técnicas tomadas fueron argumentadas, fundamentadas y orientadas a garantizar sostenibilidad, automatización y colaboración efectiva en entornos distribuidos con arquitecturas serverless.

Capítulo 5

Conclusiones

5.1. Conclusiones

La arquitectura propuesta para este proyecto ha cumplido de manera satisfactoria los cuatro objetivos específicos definidos en el Capítulo 1, validando su coherencia y pertinencia mediante escenarios de prueba, revisión documental y sesiones de retroalimentación con expertos. Estos objetivos se desarrollaron de forma integrada, asegurando que cada uno aporte valor al entorno colaborativo y a la automatización de procesos en desarrollos con serverless.

- **Módulo de gestión y clasificación serverless.** Se definió un esquema estandarizado de metadatos y un modelo de clasificación flexible que permite organizar los desarrollos existentes en categorías coherentes según criterios técnicos y de negocio. Esta estructura proporciona un punto único de referencia para la búsqueda, filtrado y reutilización de componentes, lo que facilita la gobernanza técnica y reduce la dispersión de información en entornos distribuidos.
- **Mecanismo automatizado de generación de documentación técnica.**Se implementó un flujo basado en estándares como OpenAPI/Swagger, capaz de generar y actualizar documentación técnica de forma automática a partir de cambios en el código fuente. Este mecanismo mantiene la documentación alineada con el estado real de los artefactos, evitando inconsistencias y promoviendo la transparencia técnica entre los equipos de desarrollo, pruebas y arquitectura.
- **Sistema de generación automatizada de código y plantillas parametrizables.** Se diseñaron plantillas arquitectónicas reutilizables que permiten generar proyectos iniciales completos, incluyendo estructura de carpetas, código base y definiciones de infraestructura como código (IaC). Este enfoque asegura uniformidad en la creación de nuevos módulos y reduce la dependencia de procesos manuales repetitivos, lo que favorece la escalabilidad organizacional y la estandarización técnica.
- **Módulo de selección dinámica de patrones arquitectónicos** Se construyó un componente capaz de evaluar múltiples criterios, como requisitos de rendimiento, acoplamiento y complejidad del sistema, para recomendar patrones arquitectónicos adecuados al contexto del proyecto. La recomendación se apoya en un conjunto de reglas y criterios trazables, lo que permite a los arquitectos y líderes técnicos contar con un soporte objetivo y consistente en la toma de decisiones.

En conjunto, la integración de estos módulos bajo una arquitectura serverless y multinube genera un entorno centralizado que promueve la colaboración efectiva, la reducción de duplicidad de esfuerzos y la mejora de la trazabilidad de artefactos. La propuesta ha demostrado que es posible unificar procesos de gestión, documentación, generación de código y selección arquitectónica en un marco común que optimiza el ciclo de vida de los desarrollos y se adapta de forma flexible a diferentes escenarios empresariales.

Estos resultados, obtenidos a partir de un diseño conceptual validado con prototipos y simulaciones de flujo, evidencian que la propuesta es viable desde el punto de vista arquitectónico y aporta beneficios tangibles en términos de organización, calidad y eficiencia operativa.

5.2. Trabajos futuros

Como resultado del proceso de diseño arquitectónico propuesto la colaboración eficiente en desarrollos con serverless, se han identificado múltiples oportunidades de mejora y extensión que pueden ser abordar en investigaciones futuras.

- **Futuras Investigaciones**
  - Integración de GenAI para generación de código y documentación, incorporar modelos de lenguaje (LLMs) como Amazon Bedrock, SageMaker JumpStart o soluciones de código abierto como StarCoder o AWS CodeWhisperer.
  - Análisis automatizado de logs y métricas con inteligencia artificial. La aplicación de técnicas de machine learning sobre logs y métricas distribuidas permitiría anticipar fallos, identificar patrones ocultos y proponer acciones de remediación de forma proactiva.
  - Despliegue en múltiples proveedores cloud y estrategia multinube. Aunque la arquitectura fue diseñada tomando como referencia AWS, su diseño modular y desacoplado permite adaptarse a otros proveedores como Azure, Google Cloud o Oracle Cloud Infrastructure.// Trabajos futuros podrían validar esta portabilidad y plantear una estrategia efectiva de multinube, aprovechando lo mejor de cada proveedor, mitigando riesgos de vendor lock-in y habilitando estrategias de alta disponibilidad geográfica.

5.3. Lecciones aprendidas

El desarrollo de este proyecto permitió adquirir conocimientos técnicos avanzados y generar reflexiones. Sobre los desafíos y oportunidades en el diseño de arquitecturas colaborativas en entornos serverless. Lo cual llevo a una serie de lecciones aprendidas clave:

- **La modularidad es fundamental para escalabilidad y mantenimiento:** La arquitectura basada en componentes desacoplados y funciones serverless orquestadas mediante AWS Step Functions facilitó una clara separación de responsabilidades
- **Serverless es potente, pero requiere una estrategia de gobernanza:** los servicios serverless eliminan la necesidad de gestionar infraestructura física, esto no implica una disminución de la complejidad arquitectónica. Al contrario,la administración de múltiples funciones Lambda, eventos distribuidos, políticas de seguridad y almacenamiento transitorio exige una planificación rigurosa.
- **Automatizar documentación y código mejora productividad:** La integración de pipelines para generar documentación (Swagger, Confluence) y plantillas de código a partir de definiciones estructuradas para establecer una cultura de documentación continua , actualizada automáticamente con cada cambio en el repositorio. La automatización reduce la carga manual asociada a tareas repetitivas, liberando tiempo para actividades de mayor valor
- **La colaboración efectiva requiere integración real de herramientas:** No es contar con herramientas aisladas como GitHub, Jira o Confluence. Es necesario una arquitectura que las integrara de forma lógica, permitiendo que las decisiones arquitectónicas, las tareas de desarrollo y los documentos técnicos estén sincronizados y alineados.

  Una arquitectura colaborativa no solo resolver problemas técnicos, sino también el flujo de información entre personas y equipos.
- **La observabilidad y evaluación continua:** En el diseño se identificó que, que el entorno altamente distribuido como el serverless, contar con mecanismos de trazabilidad, dashboards personalizados y alertas es importante para el diagnóstico, la seguridad y el monitoreo.

  Ignorar la observabilidad puede llevar a dificultades significativas en la detección de fallos, análisis de rendimiento.

Bibliografía

Azanza, M., Montalvillo, L., and Díaz, O. (2021). 20 years of industrial experience at SPLC.

- Sbarski, P. (2018). Serverless Architectures on AWS. O'Reilly Media.
- Zambrano, B. (2018). Serverless Design Patterns and Best Practices. Packt Publishing.
- AWS Lambda Developer Guide. (Consultado en enero de 2022). Amazon Web Services. [Online]. Disponible en: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- Google Cloud Functions Documentation. (Consultado en enero de 2022). Google Cloud. [Online]. Disponible en: https://cloud.google.com/functions/docs
- Azure Functions Documentation. (Consultado en enero de 2022). Microsoft Azure. [Online]. Disponible en: https://azure.microsoft.com/es-es/products/functions
- Kim, G., Humble, J., Debois, P., y Willis, J. (2016). The DevOps Handbook: How to Create World-Class Agility, Reliability, Security in Technology Organizations. IT Revolution Press.
- GitHub Documentation. (Consultado en enero de 2022). GitHub. [Online]. Disponible en: https://docs.github.com/
- GitLab Documentation. (Consultado en enero de 2022). GitLab. [Online]. Disponible en: https://docs.gitlab.com/
- Bitbucket Documentation. (Consultado en enero de 2022). Atlassian. [Online]. Disponible en: https://support.atlassian.com/
- Parr, T. (2017). The Definitive ANTLR 4 Reference. Pragmatic Bookshelf.
- Swagger Documentation. (Consultado en enero de 2022). Swagger. [Online]. Disponible en: https://swagger.io/docs/
- AWS Cloud Development Kit (CDK) Documentation. (Consultado en enero de 2022). Amazon Web Services. [Online]. Disponible en: https://docs.aws.amazon.com/cdk/latest/guide/home.html
- Google Cloud Deployment Manager Documentation. (Consultado en enero de 2022). Google Cloud. [Online]. Disponible en: https://cloud.google.com/deployment-manager/docs
- Azure Resource Manager (ARM) Templates Documentation. (Consultado en enero de 2022). Microsoft Azure. [Online]. Disponible en: https://docs.microsoft.com/en-us/azure/azure-resource-manager/
- Terraform Documentation. (Consultado en enero de 2022). HashiCorp. [Online]. Disponible en: https://www.terraform.io/docs/

- AWS Well-Architected Framework. (Consultado en enero de 2022). Amazon Web Services. [Online]. Disponible en: https://aws.amazon.com/architecture/well-architected/
- Google Cloud Architecture Framework Documentation. (Consultado en enero de 2022). Google Cloud. [Online]. Disponible en: https://cloud.google.com/architecture/framework
- Azure Architecture Center Documentation. (Consultado en enero de 2022). Microsoft Azure. [Online]. Disponible en: https://docs.microsoft.com/en-us/azure/architecture/