Sistema de Evaluación de Costos en AWS (SECC-AWS)

Luisa Fernanda Rojas Rodriguez

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

Dirigida por
Juan Camilo Parra Martínez

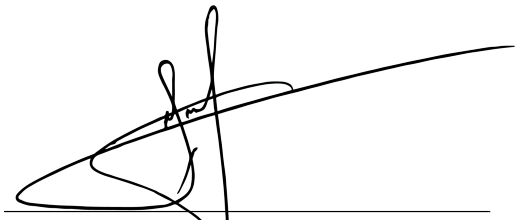
Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
19 de Junio de 2026

Santiago de Cali, 19 de Junio de 2026.

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

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Sistema de Evaluación de Costos en AWS (SECC-AWS)" realizado por el estudiante de Magister en Ingeniería de Software Luisa Fernanda Rojas Rodriguez (cod: 8984585), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,



Juan Camilo Parra Martínez

Santiago de Cali, 19 de Junio de 2026.

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

Cordial Saludo.

Me permito presentar a su consideración el proyecto de grado titulado "Sistema de Evaluación de Costos en AWS (SECC-AWS)" 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,

Luisa F Rojas Rodriguez

Luisa Fernanda Rojas Rodriguez
Código: 8984585

Ficha Resumen
Anteproyecto de Trabajo de Grado

Posible Título: Sistema de Evaluación de Costos en AWS (SECC-AWS)

1. Área de trabajo: Gestión de costos en computación en la nube
2. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
3. Estudiante: Luisa Fernanda Rojas Rodríguez
4. Correo electrónico: luisarojas@javerianacali.edu.co
5. Dirección y teléfono: Diagonal 14 No 25-01, Cel: 3507774957
6. Director: Juan Camilo Parra Martínez
7. Vinculación del director: HC
8. Correo electrónico del director: juan.parra@avaldigitallabs.com
9. Co-Director (Si aplica): NA
10. Grupo o empresa que lo avala (Si aplica): NA
11. Otros grupos o empresas: NA
12. ODS que aplica al proyecto (Agenda 2030):
    - **ODS 4: Educación de calidad.** Garantizar una educación inclusiva, equitativa y de calidad, y promover oportunidades de aprendizaje permanente para todos.
    - **ODS 12: Producción y consumo responsables.** Garantizar modalidades de consumo y producción sostenibles.
13. Palabras clave (al menos 5): computación en la nube, estimación de costos, inteligencia artificial, AWS, arquitectura serverless.
14. Fecha de inicio: 19 de febrero de 2026
15. Duración estimada (en meses): cuatro meses

Resumen: El crecimiento de la computación en la nube ha transformado la manera en que las organizaciones adquieren y gestionan infraestructura tecnológica bajo modelos de pago por uso; sin embargo, esta flexibilidad introduce complejidad en la estimación preliminar del impacto económico de las arquitecturas antes de su despliegue, especialmente en entornos como Amazon Web Services (AWS), donde la estructura de costos depende de múltiples variables técnicas y patrones de consumo. Aunque el Pilar de Optimización de Costos del AWS Well-Architected Framework establece lineamientos para integrar la gestión financiera desde el diseño, persiste una brecha en su aplicación práctica en etapas tempranas, cuando aún no existen datos reales de consumo y las decisiones deben tomarse con base en supuestos técnicos. En este contexto, el presente proyecto desarrolló y validó un sistema de proyección preliminar de costos denominado SECC-AWS, implementado como una aplicación web que permite evaluar arquitecturas y escenarios técnicos suministrados por el usuario mediante la selección estructurada de parámetros relacionados con arquitectura, carga, datos, cómputo y otros aspectos operativos, con el fin de generar estimaciones iniciales del impacto económico en horizontes temporales definidos antes de la implementación de cargas de trabajo. El sistema integra interfaces de usuario, un backend de procesamiento y un agente de inteligencia artificial utilizado como mecanismo de apoyo informativo para generar orientaciones contextualizadas, sin realizar simulaciones de facturación en tiempo real ni sustituir procesos formales de análisis financiero. La metodología adoptada fue de carácter ágil e incremental, apoyada en Scrumban, e incluyó una validación comparativa frente a expertos de referencia mediante métricas de precisión, consistencia, tiempo de análisis y utilidad percibida. Como resultado, se obtuvo un prototipo funcional que aporta una visión estructurada del impacto económico potencial en fases tempranas de diseño arquitectónico y que puede servir como herramienta de apoyo en procesos iniciales de evaluación de infraestructuras en la nube de AWS.

Agradecimientos

Agradezco a todas las personas que de una u otra manera hicieron parte de este proyecto.

Primero que todo, le doy gracias a Dios por sostenerme a lo largo de este proceso, que aunque no comenzó con mucha claridad, me demostró de mil maneras que todo es posible mientras exista voluntad.

A mi madre, por escucharme y prestarme atención aunque no siempre entienda el tema del que le hablo, por alentarme, cuidarme y no dejarme rendir en ningún momento.

A mis perritos Scoby y Scrapy, por su compañía incondicional en cada noche de trabajo.

A Jose Alejandro Benitez Aragon, por su valiosa orientación, que me devolvió la fe en el proyecto en el momento justo.

A Juan Camilo Parra Martínez, director de este proyecto, por no desistir, y por la claridad y puntualidad de sus aportes a lo largo de todo el proceso.

Al profesor Mario Julián Mora Cardona y a la directora del programa, Luisa Fernanda Rincón Pérez, por la oportunidad brindada y por creer en mí a lo largo de este proceso.

A la Pontificia Universidad Javeriana Cali, por la oportunidad y por todo lo vivido durante este proceso.

A todas las demás personas que de alguna manera dejaron su huella en este camino, gracias.

Resumen

El crecimiento de la computación en la nube introduce complejidad en la estimación preliminar del impacto económico de los escenarios de arquitectura en AWS, donde los costos dependen de múltiples variables técnicas y patrones de consumo. Aunque el pilar de Optimización de Costos del AWS Well-Architected Framework establece lineamientos para integrar la gestión financiera desde el diseño, persiste una brecha en etapas tempranas, cuando las decisiones deben tomarse con base en supuestos técnicos sin datos reales de consumo. En este contexto, el presente proyecto desarrolla y valida el Sistema de Evaluación de Costos en la Nube para AWS (SECC-AWS), una aplicación web serverless que estima el impacto económico de escenarios de arquitectura descritos por el usuario, a partir de parámetros técnicos complementados mediante inferencia determinista automática según el estilo arquitectónico seleccionado. El sistema integra un agente de inteligencia artificial basado en Strands Agents con Amazon Bedrock, que identifica los servicios AWS necesarios, consulta sus precios oficiales en tiempo real mediante el protocolo Model Context Protocol (MCP) y calcula el costo mensual de cada servicio con el Intérprete de Código de Bedrock AgentCore. El resultado es un informe ejecutivo descargable en PDF con desglose de costos, evaluación frente al presupuesto, nivel de riesgo, modelo de pricing recomendado y evaluación Well-Architected. El prototipo opera sobre una arquitectura serverless sin persistencia de datos, validada comparativamente frente a cuatro expertos mediante métricas de precisión, consistencia, tiempo de análisis y utilidad percibida.

Palabras Clave: computación en la nube, estimación de costos, inteligencia artificial, AWS, arquitectura serverless.

Abstract

The growth of cloud computing introduces complexity in the preliminary estimation of the economic impact of architecture scenarios in AWS, where costs depend on multiple technical variables and consumption patterns. Although the Cost Optimization pillar of the AWS Well-Architected Framework establishes guidelines for integrating financial management from the design stage, a gap persists in early stages, when decisions must be made based on technical assumptions without real consumption data. In this context, the present project develops and validates the Cloud Cost Evaluation System for AWS (SECC-AWS), a serverless web application that estimates the economic impact of architecture scenarios described by the user, based on technical parameters complemented through automatic deterministic inference according to the selected architectural style. The system integrates an artificial intelligence agent based on Strands Agents with Amazon Bedrock, which identifies the required AWS services, queries their official prices in real time through the Model Context Protocol (MCP), and calculates the monthly cost of each service using the Bedrock AgentCore Code Interpreter. The result is an executive report, downloadable as a PDF, that includes a cost breakdown, a budget evaluation, a risk level assessment, a recommended pricing model, and a Well-Architected evaluation. The prototype operates on a serverless architecture without data persistence, and was validated comparatively against four experts using metrics of precision, consistency, analysis time, and perceived usefulness.

Keywords: cloud computing, cost estimation, artificial intelligence, AWS, serverless architecture.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Presentación del tema

La computación en la nube ha transformado la manera en que las organizaciones adquieren y gestionan infraestructura tecnológica, introduciendo modelos de pago por uso que, si bien ofrecen flexibilidad y escalabilidad, generan complejidad en la estimación preliminar del impacto económico antes del despliegue de cargas de trabajo. En este contexto, la evaluación previa de costos en infraestructuras como Amazon Web Services (AWS) se convierte en un elemento clave para una adopción informada y sostenible.

En respuesta a esta necesidad, el presente trabajo desarrolla y valida SECC-AWS (Sistema de Evaluación de Costos en la Nube para AWS), una aplicación web serverless que integra un agente de inteligencia artificial para generar estimaciones preliminares de costos a partir de parámetros de arquitectura definidos por el usuario, con precios oficiales obtenidos en tiempo real desde la AWS Price List API.

1.2. Definición del problema

1.2.1. Planteamiento del problema

En el contexto actual de transformación digital, las organizaciones que adoptan servicios de computación en la nube enfrentan el desafío de diseñar arquitecturas que no solo cumplan requisitos de desempeño, seguridad y disponibilidad, sino que también sean económicamente sostenibles. El AWS Well-Architected Framework, particularmente su Pilar de Optimización de Costos, establece que la gestión financiera debe integrarse desde las etapas iniciales del diseño arquitectónico, dado que decisiones como la selección de servicios, el dimensionamiento de recursos, la arquitectura de red y los modelos de adquisición de capacidad tienen un impacto financiero directo sobre las cargas de trabajo (Amazon Web Services, 2024).

No obstante, la evidencia de mercado muestra que su aplicación práctica continúa siendo un desafío. Gartner Inc. (2025) identifica falencias recurrentes en los procesos de adopción de la nube, entre ellas migraciones mal concebidas con retornos de inversión poco realistas, ausencia de experiencia especializada y prácticas ineficientes en la adquisición y gestión de servicios. De manera complementaria, Gartner Inc. (2024) proyecta un crecimiento sostenido del gasto mundial en servicios de nube pública, lo que intensifica la necesidad de mecanismos claros para la comprensión y control financiero previo al despliegue.

En este escenario, las decisiones arquitectónicas deben adoptarse en fases donde aún no existen datos reales de consumo, lo que implica que la estimación del impacto económico depende de proyecciones basadas en supuestos técnicos. Aunque el Pilar de Optimización de Costos establece principios claros para integrar la gestión financiera desde el diseño, estos lineamientos no proporcionan mecanismos específicos para generar estimaciones cuantificadas antes del despliegue. En consecuencia, persiste una brecha entre los principios estratégicos y su aplicación operativa en etapas tempranas de adopción.

1.2.2. Sistematización

Con base en lo anterior, surge la siguiente pregunta de investigación:

> ¿Cómo diseñar un sistema que permita evaluar arquitecturas de referencia y generar una proyección preliminar de costos en infraestructuras AWS, integrando un agente de inteligencia artificial como mecanismo de apoyo informativo previo al despliegue de cargas de trabajo?

1.3. Justificación del Proyecto:

La gestión financiera en entornos de computación en la nube se ha convertido en un aspecto crítico dentro de los procesos de transformación digital. A medida que un número creciente de organizaciones migra sus cargas de trabajo hacia AWS, la limitada comprensión de los modelos de consumo, facturación y control del gasto representa un riesgo significativo para la sostenibilidad económica y la planificación informada de la adopción tecnológica.

Desde la perspectiva técnica, los costos en entornos cloud no dependen de un único componente, sino de la combinación de múltiples recursos, configuraciones y patrones de uso, lo que hace que la evaluación del impacto económico no sea evidente antes de la implementación. Desde la perspectiva económica, el desafío radica en la dificultad de anticipar el comportamiento financiero de una infraestructura cuando las decisiones técnicas se toman sin una referencia clara sobre su impacto en el gasto. Desde la perspectiva académica, el desarrollo de SECC-AWS aporta a la construcción de conocimiento aplicado al articular los aspectos técnicos de la infraestructura cloud con su dimensión económica.

La solución está dirigida a profesionales y equipos de tecnología que participan en la planeación de arquitecturas en AWS, así como a estudiantes o perfiles técnicos que buscan evaluar costos antes del despliegue, sin sustituir los procesos formales de análisis financiero ni las soluciones empresariales especializadas. En un entorno donde la adopción de servicios cloud continúa en expansión, disponer de mecanismos estructurados para la evaluación previa de costos se convierte en un elemento clave para una implementación más informada.

1.4. Delimitaciones y alcances

El proyecto comprendió el diseño, construcción y evaluación de un prototipo de aplicación web denominado SECC-AWS, orientado a evaluar arquitecturas de referencia suministradas por el usuario con el fin de generar una estimación preliminar de costos proyectados en un horizonte temporal seleccionado (mensual, trimestral o anual), previo al despliegue de cargas de trabajo en AWS.

En relación con cada objetivo específico, el alcance se delimitó de la siguiente manera:

- El análisis de escenarios representativos se limitó a cinco estilos arquitectónicos de referencia: monolítico, microservicios, serverless, event-driven e híbrido. No se abordaron configuraciones personalizadas fuera de estos estilos ni la totalidad de servicios disponibles en AWS.
- El diseño del modelo funcional comprendió la definición de dos bloques de parámetros de entrada — contexto de evaluación y datos de arquitectura —, un conjunto de parámetros inferidos automáticamente por el sistema según el estilo arquitectónico seleccionado, y los contratos de la interfaz pública e interna. No se incluyó el diseño de mecanismos de gobernanza organizacional ni de integración con sistemas externos de gestión financiera.
- El desarrollo del prototipo funcional se ejecutó sobre una arquitectura serverless en AWS, implementando el frontend como sitio estático en Amazon S3, el backend mediante funciones AWS Lambda y Amazon API Gateway, y el agente de inteligencia artificial mediante Strands Agents con Amazon Bedrock. El prototipo no fue desplegado en un entorno productivo ni contempló manejo de datos personales identificables.
- La estrategia de validación comparativa se aplicó sobre los cinco escenarios definidos, utilizando cuatro métricas objetivas: precisión de recomendaciones, consistencia en estimaciones, tiempo de análisis y utilidad percibida. La validación no incluyó pruebas de carga, pruebas de seguridad ni evaluación de desempeño bajo condiciones de alto tráfico.

El sistema se limitó a la generación de estimaciones y recomendaciones conforme a los parámetros definidos por el usuario, sin realizar simulaciones de facturación en tiempo real, optimización automática de costos ni sustituir soluciones empresariales especializadas o procesos formales de análisis financiero. El agente de inteligencia artificial operó como un mecanismo de consulta informativa y no como un sistema autónomo de toma de decisiones. Se asumió que los usuarios poseen conocimientos básicos en tecnología de la información, aunque no necesariamente experiencia previa en AWS o en evaluación de costos en la nube.

1.5. Objetivos de la investigación

El objetivo general del proyecto es desarrollar y validar el sistema de proyección de costos SECC-AWS, orientado a la estimación preliminar del gasto y a la identificación de mecanismos de control en infraestructuras AWS, mediante la integración de un agente de inteligencia artificial y la

comparación de sus resultados con evaluaciones realizadas por expertos de referencia, con el fin de analizar su utilidad práctica como herramienta de apoyo previo al despliegue de cargas de trabajo en la nube.

Este objetivo general se desglosa en cuatro objetivos específicos:

1. Analizar escenarios representativos de implementación de cargas de trabajo en AWS, identificando arquitecturas de referencia y casos de estudio relevantes que permitan estructurar adecuadamente la proyección de costos.

2. Diseñar la arquitectura y el modelo funcional del sistema SECC-AWS, definiendo sus componentes, módulos, flujos de interacción y el esquema de integración con el agente de inteligencia artificial para la generación de estimaciones y orientaciones sobre la evaluación y control del gasto en AWS.

3. Desarrollar el prototipo funcional de la aplicación web para la proyección de costos en infraestructuras AWS, implementando su arquitectura frontend y backend e integrando el agente de inteligencia artificial para la generación de estimaciones preliminares.

4. Diseñar e implementar una estrategia de validación comparativa entre las estimaciones generadas por el sistema y las evaluaciones realizadas por expertos de referencia, estableciendo métricas objetivas para el análisis de su desempeño.

1.6. Metodología de la investigación

El desarrollo del sistema SECC-AWS se abordó mediante una metodología ágil de tipo incremental, basada en un enfoque híbrido Scrumban, el cual combina principios de Scrum y Kanban. Este enfoque permitió estructurar el trabajo en iteraciones cortas con entregables funcionales, al tiempo que se gestionó el flujo de actividades de manera flexible y visual, resultando adecuado para un proyecto académico de desarrollo de software orientado a la construcción de un prototipo funcional.

- **Enfoque de desarrollo:** Desde la perspectiva de Scrum, el proyecto se organizó en ciclos iterativos de corta duración, definidos como sprints semanales, los cuales permitieron la entrega progresiva de incrementos funcionales del sistema. Cada iteración facilitó la validación de resultados parciales, el refinamiento continuo de los requisitos y el ajuste del desarrollo en función de los objetivos establecidos. De manera complementaria, los principios de Kanban se emplearon como mecanismo de gestión del trabajo, permitiendo la visualización de las tareas en un tablero organizado por estados, lo que favoreció el seguimiento del progreso, la priorización de actividades y el control del flujo de trabajo.

  La adopción del enfoque Scrumban se justificó por la naturaleza evolutiva del proyecto, en el cual los requisitos se ajustaron a medida que se avanzó en el análisis conceptual y en la construcción del prototipo. Este enfoque resultó apropiado para un proyecto desarrollado

de manera individual y con tiempo limitado, ya que permitió mantener un equilibrio entre planificación estructurada y flexibilidad operativa.

- **Fases del desarrollo:** Desde el punto de vista técnico, la metodología se estructuró en fases ejecutadas de forma iterativa a lo largo de los ciclos de trabajo:

  Elicitación y análisis de requisitos: recopilación y análisis de los requisitos funcionales y no funcionales del sistema, identificando variables técnicas y económicas relevantes para la evaluación preliminar de costos en AWS.

  Documentación y priorización de requisitos: organización y priorización de los requisitos identificados, definiendo los incrementos funcionales a desarrollar en cada iteración.

  Diseño y desarrollo incremental: construcción progresiva de los módulos del sistema, integrando interfaces de usuario, lógica de negocio y el agente de inteligencia artificial para la generación de recomendaciones.

  Pruebas funcionales internas: verificación del correcto funcionamiento del prototipo en términos de coherencia lógica, generación de informes y consistencia de resultados.

  Documentación y cierre: consolidación de la documentación técnica y académica del proyecto.

- **Estrategia de evaluación y validación académica:** Con el fin de fortalecer el rigor científico del proyecto, se implementó una estrategia de validación comparativa entre las recomendaciones generadas por el sistema SECC-AWS y las evaluaciones realizadas por expertos de referencia en infraestructuras AWS.

  Los cinco escenarios representativos de implementación de cargas de trabajo definidos fueron utilizados durante el desarrollo y las pruebas funcionales internas del sistema. Para la validación con los expertos de referencia, se adoptó un enfoque de libre elección, permitiendo que cada evaluador eligiera el escenario de su preferencia y registrara los resultados obtenidos mediante un conjunto estructurado de métricas objetivas:

  - **Precisión de recomendaciones:** porcentaje de coincidencia entre las recomendaciones generadas por el sistema y el criterio del experto.
  - **Consistencia en estimaciones:** diferencia porcentual entre los valores de estimación de costos generados por el sistema y los calculados mediante la Calculadora de Precios de AWS.
  - **Tiempo de análisis:** tiempo requerido por el sistema para generar una evaluación preliminar completa.
  - **Utilidad percibida:** valoración cualitativa del sistema por parte del experto mediante una escala estructurada de apreciación.

Para cada evaluación, el experto registró los resultados obtenidos y realizó un análisis con base en su criterio, identificando niveles de coincidencia, diferencias significativas y oportunidades de mejora del sistema.

1.7. Relevancia e impacto

La relevancia de este trabajo se articula en tres dimensiones. Desde la perspectiva técnica, aborda la complejidad de la estimación de costos en entornos cloud, donde el gasto depende de la combinación de múltiples recursos, configuraciones y patrones de uso cuya interacción no es evidente antes del despliegue. Desde la perspectiva económica, ofrece un mecanismo estructurado para anticipar el comportamiento financiero de una infraestructura, reduciendo la incertidumbre que enfrentan los equipos técnicos al tomar decisiones arquitectónicas sin una referencia clara sobre su impacto en el gasto. Desde la perspectiva académica, el trabajo contribuye a la construcción de conocimiento aplicado en ingeniería de software y computación en la nube, articulando modelado arquitectónico y evaluación económica preliminar en un marco integrador que supera la fragmentación identificada en la literatura existente.

El sistema está dirigido a profesionales y equipos de tecnología que participan en la planeación de arquitecturas en AWS, así como a estudiantes o perfiles técnicos que buscan evaluar costos antes del despliegue, sin sustituir los procesos formales de análisis financiero ni las soluciones empresariales especializadas.

1.8. Resultados obtenidos

El desarrollo del proyecto permitió alcanzar los siguientes resultados principales:

- Se analizaron y definieron cinco escenarios representativos de implementación de cargas de trabajo en AWS, basados en los cinco estilos arquitectónicos más representativos de los sistemas cloud modernos: monolítico, microservicios, serverless, event-driven e híbrido. Cada escenario cuenta con sus respectivos parámetros de entrada y parámetros inferidos automáticamente por el sistema según el estilo arquitectónico seleccionado.
- Se diseñó e implementó la arquitectura serverless del sistema SECC-AWS, compuesta por un frontend estático en Amazon S3, un backend con funciones AWS Lambda expuestas mediante Amazon API Gateway, un agente de inteligencia artificial orquestado con Strands Agents y Amazon Bedrock, un servidor MCP que consulta precios oficiales en tiempo real mediante la AWS Price List API, y el Intérprete de Código de Bedrock AgentCore para el cálculo preciso de costos.
- Se desarrolló el prototipo funcional de la aplicación web, que permite al usuario describir su arquitectura, generar una estimación de costos con precios reales de AWS y descargar el informe ejecutivo en formato PDF, integrando recomendaciones de optimización alineadas con el Pilar de Optimización de Costos del AWS Well-Architected Framework.
- Se diseñó e implementó una estrategia de validación comparativa mediante un formato de evaluación estructurado para expertos en AWS, definiendo cuatro métricas objetivas que permiten contrastar las estimaciones del sistema con el criterio experto y la Calculadora de Precios de AWS.

1.9. Estructura de la tesis

El documento se organiza en cinco capítulos. El **Capítulo 1** presenta la introducción, incluyendo el planteamiento del problema, los objetivos, la metodología y la justificación del trabajo. El **Capítulo 2** expone el marco teórico y los antecedentes, abarcando los fundamentos de computación en la nube, gestión financiera cloud y FinOps, sistemas de apoyo a decisiones basados en inteligencia artificial, y la síntesis crítica de los enfoques existentes que evidencia la brecha que motiva esta propuesta. El **Capítulo 3** describe el diseño e implementación del sistema SECC-AWS, detallando su arquitectura serverless, los contratos de la interfaz pública e interna, el diseño del agente de inteligencia artificial con Strands Agents y Amazon Bedrock, la integración mediante el protocolo MCP, el Intérprete de Código de Bedrock AgentCore, los requisitos funcionales del prototipo y el diseño del prompt del agente. El **Capítulo 4** presenta la estrategia de evaluación y los resultados de la validación comparativa frente al experto de referencia, analizando el desempeño del sistema con base en las métricas definidas. Finalmente, el **Capítulo 5** expone las conclusiones del trabajo, las limitaciones identificadas y las líneas de trabajo futuro.

Capítulo 2

Marco de referencia

En este capítulo se presentan las bases teóricas y los trabajos previos que sirven como fundamento para la realización del proyecto. El capítulo se organiza en dos secciones principales: el marco teórico, que desarrolla los conceptos clave necesarios para comprender el problema abordado y la solución propuesta, y el estado del arte, que revisa y analiza los trabajos previos relacionados con la estimación temprana de costos en entornos cloud, evidenciando la brecha que motiva la presente investigación.

2.1. Marco teórico

2.1.1. Definición de computación en la nube y modelo económico

La computación en la nube representa un modelo de provisión de servicios tecnológicos que transforma la forma tradicional de adquisición y gestión de infraestructura de TI. Según (Mell and Grance, 2011) , se define como un modelo que permite el acceso ubicuo, conveniente y bajo demanda a un conjunto compartido de recursos informáticos configurables como redes, servidores, almacenamiento y aplicaciones que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de gestión. Este paradigma se sustenta en cinco características esenciales: autoservicio bajo demanda, acceso amplio a la red, agrupación de recursos, elasticidad rápida y servicio medido (Mell and Grance, 2011).

Desde una perspectiva económica, la computación en la nube no solo introduce un cambio tecnológico, sino una transformación en la estructura financiera de las organizaciones. Mientras que los modelos tradicionales de infraestructura requerían inversiones iniciales significativas en activos físicos clasificadas como Capital Expenditures (CapEx), el modelo cloud se fundamenta en el consumo bajo demanda y la facturación basada en uso, características asociadas a Operational Expenditures (OpEx). Este enfoque se sustenta en el principio de utility pricing, donde los recursos se comparten, se asignan dinámicamente y se cobran según el nivel real de utilización (Weinman, 2012). De acuerdo con esta perspectiva, el valor económico del cloud no radica únicamente en convertir CapEx en OpEx, sino en la flexibilidad que ofrece para ajustar el gasto en función de la demanda efectiva, reduciendo compromisos de largo plazo y permitiendo mayor adaptabilidad financiera. Esta naturaleza dinámica del gasto constituye el fundamento sobre el cual emergen prácticas de gestión financiera específicas para entornos cloud.

2.1.2. Arquitectura en AWS y su impacto en la estructura de costos

En entornos de Amazon Web Services (AWS), las decisiones arquitectónicas determinan no solo el comportamiento técnico de una solución, sino también su dinámica económica. Debido al modelo

de pago por consumo y a la naturaleza elástica de los recursos, la configuración de servicios, el patrón de escalabilidad y el modelo de adquisición seleccionado influyen directamente en el costo total de operación. Como señalan (Wittig and Wittig, 2023), la computación en la nube introduce un esquema basado en uso medido, donde el dimensionamiento adecuado y la selección del modelo de precios son factores clave para evitar sobrecostos asociados a capacidad infrautilizada o mal estimada.

2.1.2.1. Patrones arquitecturales y eficiencia económica

Los patrones arquitectónicos adoptados en AWS influyen en la forma en que se consumen y distribuyen los recursos. De acuerdo con (Dua and Aggarwal, 2019), diseños basados en desacoplamiento, modularidad y escalamiento selectivo permiten ajustar la asignación de recursos según la demanda específica de cada componente. En contraste, arquitecturas menos flexibles pueden generar escalamiento global innecesario, incrementando el consumo sin un aumento proporcional en valor de negocio. Estas diferencias estructurales evidencian que la arquitectura condiciona el comportamiento económico de la infraestructura.

2.1.2.2. Modelos de precios y decisiones bajo incertidumbre

AWS ofrece distintos modelos de adquisición de capacidad, como esquemas bajo demanda y opciones con compromiso temporal. La selección entre estos modelos depende del grado de previsibilidad de la carga de trabajo. Según (Wittig and Wittig, 2023), la optimización económica requiere alinear el perfil de uso esperado con el modelo de adquisición correspondiente, integrando estas decisiones desde la fase de diseño. Esta elección implica enfrentar incertidumbre respecto al comportamiento futuro de la demanda, lo cual introduce un componente estratégico en la planificación de costos. Esta interdependencia entre arquitectura, modelos de adquisición y comportamiento esperado de la demanda configura un escenario de decisión complejo en etapas previas al despliegue, donde el impacto económico debe estimarse a partir de supuestos técnicos y escenarios de uso.

2.1.3. Gestión financiera en la nube y FinOps

La adopción del modelo económico basado en consumo bajo demanda introduce nuevos desafíos en la administración del gasto tecnológico. En este contexto surge el enfoque denominado FinOps (Financial Operations), definido como una disciplina de gestión financiera para la nube que promueve la colaboración entre equipos de ingeniería, finanzas y negocio con el fin de maximizar el valor obtenido del gasto en servicios cloud (Storment and Fuller, 2020).

FinOps se fundamenta en la premisa de que el gasto en la nube es dinámico, distribuido y altamente variable, lo que requiere procesos continuos de medición, análisis y optimización. A diferencia de los modelos tradicionales de control presupuestal centrados en inversiones de capital, FinOps propone un enfoque iterativo basado en visibilidad, responsabilidad compartida y toma de decisiones informada por datos (Storment and Fuller, 2020).

Entre los principios fundamentales del enfoque se encuentran la necesidad de transparencia en el consumo, la responsabilidad distribuida del gasto donde los equipos técnicos asumen corresponsabilidad sobre el impacto financiero de sus decisiones y la alineación del uso de recursos con métricas de valor de negocio. Este modelo reconoce que, en entornos cloud, la gestión financiera no puede recaer exclusivamente en el área de finanzas, sino que debe integrarse dentro de los procesos técnicos y operativos de la organización.

En la literatura reciente, la gestión financiera en la nube ha comenzado a integrarse con enfoques avanzados de optimización basados en inteligencia artificial, ampliando el alcance tradicional de FinOps hacia modelos predictivos y multiobjetivo. Investigaciones contemporáneas han propuesto marcos unificados que combinan predicción de costos mediante aprendizaje automático, asignación dinámica de recursos mediante aprendizaje por refuerzo y mecanismos de detección de anomalías para mejorar la resiliencia operativa en entornos multi-cloud Bhaskaran et al. (2025). Estos enfoques evidencian que la optimización económica no puede abordarse de manera aislada, sino que requiere integrar variables de desempeño, riesgo y comportamiento dinámico de la carga de trabajo dentro de un mismo sistema decisional.

Más recientemente, el enfoque denominado *Predictive Green FinOps* amplía esta perspectiva al introducir funciones objetivo ponderadas que optimizan simultáneamente costo, huella de carbono y confiabilidad, incorporando predicciones de intensidad de carbono y riesgo de interrupción en instancias spot Jakkaraju and Jayaprakasan (2026). En conjunto, estos desarrollos consolidan la tendencia hacia sistemas inteligentes de apoyo a decisiones capaces de formalizar matemáticamente la complejidad económica y técnica de la infraestructura cloud, proporcionando fundamentos teóricos adicionales para la integración de modelos predictivos en la optimización financiera de arquitecturas en la nube.

2.1.4. Sistemas de apoyo a decisiones basados en inteligencia artificial

La inteligencia artificial puede entenderse como el estudio y diseño de agentes racionales, es decir, entidades que perciben su entorno y actúan sobre él con el propósito de maximizar el cumplimiento de determinados objetivos (Russell and Norvig, 2021). Desde esta perspectiva, la racionalidad implica seleccionar acciones que optimicen resultados bajo condiciones de incertidumbre y restricciones del entorno.

En el ámbito de los sistemas de apoyo a decisiones, estos principios proporcionan marcos formales para representar conocimiento, evaluar alternativas y estructurar procesos de elección fundamentados en criterios explícitos. En escenarios donde múltiples variables técnicas y económicas interactúan, tales modelos permiten reducir la incertidumbre asociada a la complejidad del problema mediante procedimientos analíticos sistemáticos.

El desarrollo reciente de la inteligencia artificial ha estado marcado por el avance del aprendizaje automático, y en particular del aprendizaje profundo (Deep Learning), basado en modelos compuestos por múltiples capas de representación que permiten aprender patrones complejos directamente a partir de los datos (Goodfellow et al., 2016). A diferencia de los enfoques tradicionales basados en reglas explícitas, estos modelos pueden extraer características relevantes de manera automática,

mejorando su capacidad de generalización en entornos con alta dimensionalidad y variabilidad.

La evolución del aprendizaje profundo dio lugar a arquitecturas más eficientes para el procesamiento de secuencias, destacándose el modelo Transformer propuesto por Vaswani et al. (2017). Este enfoque introduce el mecanismo de atención (attention mechanism), que permite modelar dependencias entre elementos de una secuencia sin recurrir a estructuras recurrentes tradicionales, mejorando la paralelización y el manejo de relaciones contextuales de largo alcance. Estas características sentaron las bases para el desarrollo posterior de modelos de lenguaje de gran escala.

A partir de esta arquitectura emergieron los denominados modelos fundacionales (Foundation Models), definidos como modelos de gran escala entrenados sobre amplios volúmenes de datos que pueden adaptarse a múltiples tareas mediante ajustes posteriores (Bommasani et al., 2021). Su capacidad de generalización y adaptación ha ampliado el alcance de los sistemas inteligentes en tareas de análisis y generación de información en lenguaje natural, lo cual resulta pertinente en contextos donde la toma de decisiones requiere interpretar documentación técnica extensa y heterogénea.

En entornos de computación en la nube, caracterizados por configuraciones dinámicas de servicios, variabilidad en la demanda y múltiples restricciones técnicas y económicas, la optimización del uso de recursos puede interpretarse como un problema de toma de decisiones bajo incertidumbre. Desde la perspectiva de los agentes racionales, la selección de acciones orientadas a maximizar una medida de desempeño constituye el principio central para abordar este tipo de escenarios (Russell and Norvig, 2021). En este contexto, los modelos basados en aprendizaje profundo y arquitecturas fundacionales proporcionan herramientas para analizar información compleja y estructurar procesos de decisión que contribuyan a una utilización más eficiente de los recursos disponibles.

Desde esta perspectiva, los modelos de aprendizaje profundo y las arquitecturas fundacionales permiten la construcción de sistemas de recomendación capaces de generar sugerencias contextualizadas a partir del análisis de múltiples variables técnicas y económicas. En entornos cloud, estos sistemas pueden estructurar escenarios de configuración y ofrecer orientaciones preliminares fundamentadas en criterios explícitos, contribuyendo a la reducción de incertidumbre en etapas previas al despliegue.

En el contexto específico de la computación en la nube, los sistemas de recomendación orientados a la optimización de costos se enfocan en problemas como el dimensionamiento adecuado de instancias (rightsizing), la selección de modelos de adquisición bajo distintos niveles de previsibilidad de demanda y la programación eficiente de cargas de trabajo (Wittig and Wittig, 2023). Estudios recientes han demostrado que la integración de predicción de costos y asignación dinámica de recursos mediante aprendizaje automático permite generar recomendaciones cuantificables orientadas a la eficiencia económica (Bhaskaran et al., 2025).

Estos sistemas integran información arquitectónica, métricas de uso histórico y restricciones económicas para apoyar decisiones técnicas con impacto financiero directo. En entornos caracterizados por alta variabilidad e incertidumbre, la incorporación de mecanismos de recomendación basados en datos permite estructurar decisiones racionales bajo restricciones dinámicas (Russell and Norvig, 2021).

2.1.4.1. Agentes de inteligencia artificial y frameworks de orquestación

Un agente de inteligencia artificial es un sistema que percibe su entorno mediante sensores y actúa sobre él mediante efectores, con el objetivo de maximizar una medida de desempeño (Russell and Norvig, 2021). En el contexto de aplicaciones basadas en modelos de lenguaje, los agentes combinan capacidades de razonamiento con el uso de herramientas externas para resolver tareas complejas de manera autónoma. El framework Strands Agents, desarrollado por AWS, implementa este paradigma mediante un bucle autónomo de razonamiento que permite al agente identificar qué herramientas invocar, en qué orden y con qué parámetros, sin requerir la construcción manual de flujos de orquestación en el código (Amazon Web Services, 2025b). Este enfoque delega en el modelo de lenguaje la lógica de coordinación, reduciendo la complejidad de implementación y facilitando la integración de herramientas externas mediante protocolos estandarizados.

2.1.4.2. Protocolo Model Context Protocol (MCP)

El Model Context Protocol (MCP) es un estándar abierto que define cómo los modelos de lenguaje se comunican con herramientas y fuentes de datos externas (Anthropic, 2024). MCP establece un contrato bien definido basado en JSON-RPC 2.0 sobre transporte StreamableHTTP, que permite a los agentes invocar herramientas de manera estandarizada e interoperable. En el contexto del sistema SECC-AWS, el protocolo MCP fue implementado mediante FastMCP conforme a la especificación MCP 2025-03-26, posibilitando la comunicación nativa entre el agente Strands y el Servidor MCP AWS que expone las herramientas de consulta de precios oficiales.

2.1.4.3. Intérprete de Código de Bedrock AgentCore

El Intérprete de Código de Bedrock AgentCore es un servicio de AWS que permite ejecutar código Python en un entorno aislado y seguro como herramienta invocable por un agente de inteligencia artificial (Amazon Web Services, 2025a). A diferencia de la estimación directa por el modelo de lenguaje, el Intérprete de Código garantiza cálculos precisos y deterministas al ejecutar scripts con variables de precio unitario y uso estimado, eliminando el riesgo de errores de redondeo o alucinaciones numéricas del modelo. En el sistema SECC-AWS, esta herramienta fue integrada como `execute_cost_calculation`, permitiendo al agente Strands calcular el costo mensual de cada servicio AWS con precisión matemática a partir de los precios oficiales obtenidos de la AWS Price List API.

2.1.5. Evaluación de sistemas inteligentes

La evaluación constituye un componente esencial en el desarrollo de sistemas basados en inteligencia artificial, ya que permite determinar en qué medida un agente cumple con los objetivos para los cuales fue diseñado. Desde la perspectiva de los agentes racionales, el desempeño se mide en función de su capacidad para maximizar una medida de rendimiento previamente definida en relación con el entorno en el que opera (Russell and Norvig, 2021). En consecuencia, la definición explícita de criterios de evaluación resulta fundamental para establecer la efectividad del sistema.

En modelos basados en aprendizaje automático, la evaluación suele realizarse mediante métricas cuantitativas que comparan el comportamiento del modelo frente a datos de referencia. Entre las métricas más utilizadas se encuentran la exactitud (accuracy), la precisión (precision), la exhaustividad (recall) y la medida F1, las cuales permiten analizar el desempeño en tareas de clasificación o predicción (Goodfellow et al., 2016). No obstante, como señalan Provost and Fawcett (2013), la selección de métricas debe considerar el contexto del problema y los costos asociados a errores de distinto tipo, dado que diferentes métricas pueden conducir a conclusiones divergentes sobre la calidad del modelo.

En sistemas de recomendación, la evaluación puede incorporar métricas adicionales como el error absoluto medio (MAE), la raíz del error cuadrático medio (RMSE) y medidas de relevancia como Precision@K o Recall@K, que permiten estimar la calidad de las recomendaciones generadas (Ricci et al., 2015). La elección de estas métricas depende del tipo de sistema y de los objetivos funcionales definidos.

Finalmente, la evaluación experimental implica validar el sistema en conjuntos de datos independientes a los utilizados durante el entrenamiento, con el fin de estimar su capacidad de generalización y evitar fenómenos como el sobreajuste (overfitting). Este procedimiento suele incluir la separación explícita entre datos de entrenamiento y prueba, así como técnicas de validación cruzada que permiten obtener estimaciones más robustas del desempeño del modelo. De esta manera, se busca garantizar que el comportamiento observado refleje la capacidad real del sistema para enfrentar nuevas entradas y no únicamente su adaptación a datos específicos.

2.2. Estado del Arte

2.2.1. Modelos de estimación temprana de costos en entornos cloud

La estimación temprana de costos en entornos de computación en la nube ha sido abordada mediante enfoques basados en modelado estructurado y representaciones formales del sistema. Diversos estudios han propuesto mecanismos para anticipar el impacto económico de decisiones arquitectónicas antes del despliegue operativo, con el objetivo de reducir la incertidumbre financiera asociada al modelo de pago por uso.

Aoshima y Yoshida proponen un método de estimación en fase pre-diseño basado en grafos dirigidos acíclicos (DAG), permitiendo modelar dependencias entre componentes del sistema y derivar estimaciones de costos bajo esquemas de facturación cloud Aoshima and Yoshida (2020, 2022). Este enfoque enfatiza la sistematización del análisis estructural como mecanismo para estimar costos antes de la implementación, destacando la importancia de incorporar consideraciones económicas desde etapas conceptuales.

De manera complementaria, Raouf et al. introducen un modelo basado en Data Flow Diagrams (DFD) enriquecidos con anotaciones estructuradas, mediante el cual las interacciones funcionales del sistema se traducen en estimaciones de consumo de recursos cloud Raouf et al. (2026). La validación empírica presentada por los autores demuestra alta precisión comparativa frente a herramientas automatizadas de cálculo de infraestructura, evidenciando la viabilidad de integrar

modelado conceptual y análisis económico preliminar.

Por su parte, He et al. desarrollan un modelo UML Activity extendido (AeModel) orientado a entornos AWS, incorporando algoritmos automáticos de extracción de información desde modelos de actividad para generar estimaciones operativas He et al. (2012). Este enfoque vincula explícitamente el modelado funcional con restricciones de desempeño y costo, fortaleciendo la relación entre diseño arquitectónico y proyección financiera.

No obstante, aunque estos trabajos representan avances significativos en la incorporación del costo como variable temprana de diseño, su orientación se mantiene predominantemente técnica, centrada en la cuantificación del consumo de recursos, con menor énfasis en la articulación organizacional o en la integración con marcos estructurados de gestión financiera cloud a nivel empresarial.

2.2.2. Modelado formal para decisiones de despliegue y escalamiento

Además de la estimación conceptual de costos, la literatura ha explorado el uso de lenguajes formales y enfoques model-driven para evaluar decisiones de despliegue antes de la implementación productiva.

Johnsen et al. introducen el lenguaje ABS como mecanismo para modelar estrategias de escalamiento y balanceo de carga en fases tempranas de diseño Johnsen et al. (2017). Mediante modelos ejecutables, su propuesta permite comparar alternativas de despliegue considerando simultáneamente calidad de servicio y costo, trasladando el análisis desde experimentación posterior al desarrollo hacia simulaciones analíticas previas.

En el ámbito de Model-Driven Web Engineering, Preciado et al. proponen estimaciones de costo en fase de diseño mediante la definición sistemática de variables de infraestructura y su validación experimental Preciado et al. (2018); Martin et al. (2017). Estos enfoques permiten anticipar costos de producción antes del despliegue definitivo, reduciendo riesgos asociados a decisiones arquitectónicas ineficientes.

Si bien estos estudios aportan rigurosidad formal en la evaluación de estrategias técnicas, su foco permanece centrado en la optimización arquitectónica y el comportamiento del sistema, sin extender el análisis hacia marcos organizacionales de control financiero continuo o estructuras de gobernanza del gasto cloud.

2.2.3. Enfoques tecno-económicos integrados

Desde una perspectiva más amplia, Filiopoulou et al. proponen un enfoque tecno-económico basado en modelado SysML para estimar indicadores financieros como TCO, ROI y NPV en despliegues cloud Filiopoulou et al. (2015). Este trabajo representa un avance relevante al integrar arquitectura de sistemas y evaluación económica dentro de un mismo marco analítico.

La incorporación de métricas financieras formales permite ampliar la discusión más allá del consumo técnico de recursos, orientando el análisis hacia decisiones de inversión. Sin embargo, aunque el modelo incorpora evaluación económica estructurada, su orientación se centra principalmente en

escenarios de inversión específicos y no en la construcción de mecanismos organizacionales continuos de gestión y gobernanza financiera cloud.

2.2.4. Síntesis crítica y brecha identificada

El análisis de la literatura evidencia avances sustanciales en tres dimensiones principales:

1. Modelado temprano de costos cloud a partir de representaciones estructurales del sistema.
2. Simulación formal de estrategias de despliegue y escalamiento antes de la implementación.
3. Integración tecno-económica para evaluar inversiones específicas en entornos cloud.

No obstante, estos enfoques permanecen fragmentados y mayoritariamente orientados a la optimización técnica o al análisis financiero puntual. Se identifica una ausencia de modelos integrados que articulen simultáneamente:

- Estimación conceptual temprana del impacto económico.
- Evaluación formal de escenarios arquitectónicos.
- Mecanismos estructurados de apoyo informativo para la toma de decisiones.
- Una visión coherente con prácticas de gestión financiera y gobernanza cloud.

En consecuencia, persiste una brecha en la literatura respecto a propuestas que integren modelado estructural, análisis económico y mecanismos de apoyo informativo en un marco coherente orientado a la evaluación preliminar del costo cloud en entornos empresariales.

2.2.5. Transición hacia la propuesta

A diferencia de los enfoques previamente descritos, que abordan de manera aislada la estimación técnica, la simulación arquitectónica o el análisis financiero puntual, la presente investigación propone un modelo integrador orientado a la evaluación preliminar del impacto económico, que articula estas dimensiones dentro de un marco unificado enfocado en la proyección temprana de costos en infraestructuras AWS.

La propuesta desarrollada en este trabajo combina:

- Representaciones estructurales tempranas del sistema.
- Evaluación formal de escenarios de despliegue.
- Análisis económico estructurado.
- Un componente explícito de apoyo informativo a la evaluación preliminar.

Este enfoque busca superar la fragmentación identificada en la literatura, ofreciendo un modelo orientado no únicamente a la optimización técnica del consumo de recursos, sino a la construcción de un mecanismo de apoyo informativo para la evaluación preliminar del costo cloud en contextos empresariales.

De esta manera, la contribución del presente estudio radica en la integración metodológica de modelado conceptual y evaluación económica preliminar, proporcionando un marco analítico que fortalece la evaluación informada desde fases tempranas del diseño arquitectónico.

2.3. Resumen del capítulo

Este capítulo presentó las bases teóricas y los trabajos previos que sustentan el desarrollo del sistema SECC-AWS. En el marco teórico se desarrollaron los conceptos fundamentales de computación en la nube y su modelo económico, la arquitectura en AWS y su impacto en los costos, la gestión financiera cloud y FinOps, los sistemas de apoyo a decisiones basados en inteligencia artificial, incluyendo el framework Strands Agents, el protocolo MCP y el Intérprete de Código de Bedrock AgentCore, así como los fundamentos de evaluación de sistemas inteligentes. En el estado del arte se revisaron los principales enfoques de estimación temprana de costos en entornos cloud, identificando la brecha que persiste en la literatura respecto a propuestas que integren modelado estructural, consulta de precios oficiales en tiempo real, cálculo preciso mediante ejecución de código y mecanismos de apoyo informativo en un marco coherente orientado a la evaluación preliminar del costo cloud. El siguiente capítulo describe el diseño e implementación del sistema SECC-AWS como respuesta a esta brecha identificada.

Capítulo 3

Diseño e Implementación del Sistema SECC-AWS

Este capítulo describe el diseño e implementación del prototipo SECC-AWS, una aplicación web serverless que integra un agente de inteligencia artificial para generar estimaciones preliminares de costos en AWS a partir de parámetros de arquitectura definidos por el usuario. El sistema evalúa cinco estilos arquitectónicos de referencia: monolítico, microservicios, serverless, event-driven e híbrido, y produce un informe ejecutivo con costos calculados a partir de precios oficiales obtenidos en tiempo real desde la AWS Price List API.

El desarrollo se gestionó mediante una metodología Scrumban con sprints semanales, haciendo seguimiento de las tareas a través de un tablero Kanban en Notion.

3.1. Diseño

3.1.1. Análisis y delimitación de escenarios representativos de consumo en AWS

Con el fin de abordar la complejidad en la estimación de costos en entornos cloud, particularmente en AWS, se definió un enfoque basado en escenarios representativos, fundamentado en el análisis de estilos y patrones de arquitectura de software. A partir de este análisis, se identificó y depuró un conjunto de parámetros de entrada clasificados en dos grupos: aquellos proporcionados explícitamente por el usuario y aquellos inferidos automáticamente por el sistema según el estilo arquitectónico seleccionado.

Parámetros de entrada del usuario. El modelo de entrada organiza los parámetros en dos bloques: el *contexto de evaluación*, que describe las condiciones generales del escenario, y los *datos de arquitectura*, que caracterizan técnicamente el sistema a evaluar. La Tabla 3.1 presenta el conjunto final de parámetros definidos.

Tabla 3.1: Parámetros de entrada del sistema por bloque

Bloque	Parámetro	Descripción	Valores posibles
Contexto de evaluación	descripcion	Descripción libre de la arquitectura	Texto libre
	estilo_arquitectura	Estilo arquitectónico base	monolitica, microservicios, serverless, event_driven, hibrida
	horizonte_tiempo	Periodo de estimación	mensual, trimestral, anual
	presupuesto	Presupuesto disponible en USD	Número
	ambiente	Ambiente de despliegue	desarrollo, staging, produccion
	ubicacion_usuarios	Ubicación geográfica de los usuarios	latinoamerica, estados_unidos, europa, global
	ia_tipo	Tipo de integración de inteligencia artificial	ninguna, apis_externas, propia
	plazo_compromiso	Plazo de compromiso de adquisición de capacidad	sin_compromiso, 1_anio, 3_anios
Arquitectura	patron_despliegue	Patrón de despliegue	VMs, contenedores, funciones, mixto
	usuarios_concurrentes	Cantidad de usuarios simultáneos	Texto libre (Ej: 500 usuarios)
	tipo_base_datos	Tipo de base de datos	relacional, nosql, mixta, ninguna
	volumen_datos_inicial	Volumen inicial de datos	Texto libre (Ej: 50 GB)
	intensidad_procesamiento	Intensidad de cómputo requerida	ligera, media, alta
	cumplimiento	Estándar de cumplimiento requerido	ninguno, GDPR, HIPAA, PCI-DSS
	transferencia_mensual	Transferencia de datos mensual	Texto libre (Ej: 100 GB)
	sla_objetivo	Nivel de disponibilidad objetivo	>99 %, >99.9 %, >99.95 %, >99.99 %
	almacenamiento_archivos	Almacenamiento de archivos requerido	Texto libre (Ej: 80 GB)

Parámetros inferidos por el sistema. A partir del estilo arquitectónico seleccionado, el sistema infiere automáticamente un conjunto de parámetros técnicos complementarios. Este mecanismo reduce la carga de entrada del usuario y aprovecha los patrones conocidos de cada estilo arquitectónico. La Tabla 3.2 presenta la aplicabilidad de cada parámetro inferido por escenario.

Tabla 3.2: Aplicabilidad de parámetros inferidos por escenario

Parámetro inferido	Monolito	Microservicios	Serverless	Event-driven	Híbrido
Multi-AZ	—	✓	—	✓	✓
Backups	✓	✓	—	✓	✓
Auto-scaling	—	✓	✓	✓	✓
CDN (CloudFront)	—	✓	✓	—	✓
Monitoreo y alertas	✓	✓	✓	✓	✓
Expone API pública	—	✓	✓	—	✓
Red privada	—	✓	—	✓	✓
Salida a internet	—	✓	—	✓	✓
Tipo de aplicación	web	api_web	api	backend	web_api_backend

Escenarios definidos. A partir del parámetro `estilo_arquitectura`, se establecieron cinco escenarios base de consumo en AWS, uno por cada estilo arquitectónico representativo de los sistemas cloud modernos. La Tabla 3.3 presenta el resumen de los escenarios definidos.

Tabla 3.3: Resumen de escenarios representativos

Escenario	Estilo	Usuarios concurrentes	Base de datos
1	Monolítico	100–1K	Relacional
2	Microservicios	1K–10K	Mixta
3	Serverless	Variable (100–10K)	NoSQL
4	Event-driven	Variable	NoSQL
5	Híbrido	1K–10K	Mixta

3.1.2. Escenario 1: Arquitectura Monolítica

Contexto de evaluación

- **Descripción:** Aplicación web de comercio electrónico para venta de productos en Latinoamérica
- **Estilo de arquitectura:** monolitica

- **Horizonte de tiempo:** mensual
- **Presupuesto:** 500 USD
- **Ambiente:** produccion
- **Ubicación de usuarios:** latinoamerica
- **Tipo de IA:** apis_externas
- **Plazo de compromiso:** sin_compromiso

Datos proporcionados por el usuario

- **Patrón de despliegue:** VMs
- **Usuarios concurrentes:** 500 usuarios
- **Tipo de base de datos:** relacional
- **Volumen de datos inicial:** 50 GB
- **Intensidad de procesamiento:** ligera
- **Cumplimiento requerido:** ninguno
- **Transferencia de datos mensual:** 100 GB
- **SLA objetivo:** >99 %
- **Almacenamiento de archivos:** 80 GB

Parámetros inferidos por el modelo

- **Multi-AZ:** No
- **Backups:** Sí
- **Auto-scaling:** No
- **CDN:** No
- **Monitoreo y alertas:** Sí
- **Expone API pública:** No
- **Componentes en red privada:** No
- **Salida a internet:** No
- **Tipo de aplicación:** web

3.1.3. Escenario 2: Arquitectura de Microservicios

Contexto de evaluación

- **Descripción:** Plataforma SaaS de gestión empresarial con múltiples módulos independientes
- **Estilo de arquitectura:** microservicios
- **Horizonte de tiempo:** trimestral
- **Presupuesto:** 5000 USD
- **Ambiente:** produccion
- **Ubicación de usuarios:** global
- **Tipo de IA:** propia
- **Plazo de compromiso:** 3_anios

Datos proporcionados por el usuario

- **Patrón de despliegue:** contenedores
- **Usuarios concurrentes:** 5000 usuarios
- **Tipo de base de datos:** mixta
- **Volumen de datos inicial:** 500 GB
- **Intensidad de procesamiento:** media
- **Cumplimiento requerido:** GDPR
- **Transferencia de datos mensual:** 2 TB
- **SLA objetivo:** >99.9 %
- **Almacenamiento de archivos:** 1 TB

Parámetros inferidos por el modelo

- **Multi-AZ:** Sí
- **Backups:** Sí
- **Auto-scaling:** Sí
- **CDN:** Sí

- **Monitoreo y alertas:** Sí
- **Expone API pública:** Sí
- **Componentes en red privada:** Sí
- **Salida a internet:** Sí
- **Tipo de aplicación:** api_web

3.1.4. Escenario 3: Arquitectura Serverless

Contexto de evaluación

- **Descripción:** API de procesamiento de documentos bajo demanda con funciones independientes
- **Estilo de arquitectura:** serverless
- **Horizonte de tiempo:** mensual
- **Presupuesto:** 1000 USD
- **Ambiente:** produccion
- **Ubicación de usuarios:** global
- **Tipo de IA:** apis_externas
- **Plazo de compromiso:** sin_compromiso

Datos proporcionados por el usuario

- **Patrón de despliegue:** funciones
- **Usuarios concurrentes:** 300 usuarios
- **Tipo de base de datos:** nosql
- **Volumen de datos inicial:** 20 GB
- **Intensidad de procesamiento:** ligera
- **Cumplimiento requerido:** ninguno
- **Transferencia de datos mensual:** 50 GB
- **SLA objetivo:** >99.9 %
- **Almacenamiento de archivos:** 50 GB

Parámetros inferidos por el modelo

- **Multi-AZ:** No (gestionado por AWS)
- **Backups:** No (gestionado por AWS)
- **Auto-scaling:** Sí (automático)
- **CDN:** Sí
- **Monitoreo y alertas:** Sí
- **Expone API pública:** Sí
- **Componentes en red privada:** No
- **Salida a internet:** No
- **Tipo de aplicación:** api

3.1.5. Escenario 4: Arquitectura Event-Driven

Contexto de evaluación

- **Descripción:** Sistema de procesamiento de pagos en tiempo real con notificaciones asíncronas
- **Estilo de arquitectura:** event_driven
- **Horizonte de tiempo:** anual
- **Presupuesto:** 3000 USD
- **Ambiente:** produccion
- **Ubicación de usuarios:** europa
- **Tipo de IA:** propia
- **Plazo de compromiso:** 1_anio

Datos proporcionados por el usuario

- **Patrón de despliegue:** funciones
- **Usuarios concurrentes:** 2000 usuarios
- **Tipo de base de datos:** nosql
- **Volumen de datos inicial:** 500 GB

- **Intensidad de procesamiento:** media
- **Cumplimiento requerido:** GDPR
- **Transferencia de datos mensual:** 1 TB
- **SLA objetivo:** >99.9 %
- **Almacenamiento de archivos:** 500 GB

Parámetros inferidos por el modelo

- **Multi-AZ:** Sí
- **Backups:** Sí
- **Auto-scaling:** Sí
- **CDN:** No
- **Monitoreo y alertas:** Sí
- **Expone API pública:** No
- **Componentes en red privada:** Sí
- **Salida a internet:** Sí
- **Tipo de aplicación:** backend

3.1.6. Escenario 5: Arquitectura Híbrida

Contexto de evaluación

- **Descripción:** Plataforma de análisis de datos que combina procesamiento batch y servicios en tiempo real
- **Estilo de arquitectura:** hibrida
- **Horizonte de tiempo:** trimestral
- **Presupuesto:** 8000 USD
- **Ambiente:** staging
- **Ubicación de usuarios:** global
- **Tipo de IA:** propia
- **Plazo de compromiso:** 3_anios

Datos proporcionados por el usuario

- **Patrón de despliegue:** mixto
- **Usuarios concurrentes:** 3000 usuarios
- **Tipo de base de datos:** mixta
- **Volumen de datos inicial:** 2 TB
- **Intensidad de procesamiento:** media
- **Cumplimiento requerido:** GDPR
- **Transferencia de datos mensual:** 2 TB
- **SLA objetivo:** >99.9 %
- **Almacenamiento de archivos:** 2 TB

Parámetros inferidos por el modelo

- **Multi-AZ:** Sí
- **Backups:** Sí
- **Auto-scaling:** Sí
- **CDN:** Sí
- **Monitoreo y alertas:** Sí
- **Expone API pública:** Sí
- **Componentes en red privada:** Sí
- **Salida a internet:** Sí
- **Tipo de aplicación:** web_api_backend

3.1.7. Diseño de la interfaz de usuario

El diseño de la interfaz de usuario partió de un proceso de bocetado manual que permitió definir la estructura y el flujo de navegación del sistema antes de iniciar la implementación. Este enfoque, coherente con la metodología Scrumban adoptada, facilitó la validación conceptual de la experiencia de usuario en una etapa temprana del desarrollo, sin incurrir en el costo de implementar pantallas que pudieran requerir cambios estructurales.

La Figura 3.1 presenta el boceto inicial de la pantalla de inicio, que concibió dos zonas: una sección de bienvenida con el llamado a la acción principal y una sección de formulario con campos de información básica del proyecto.

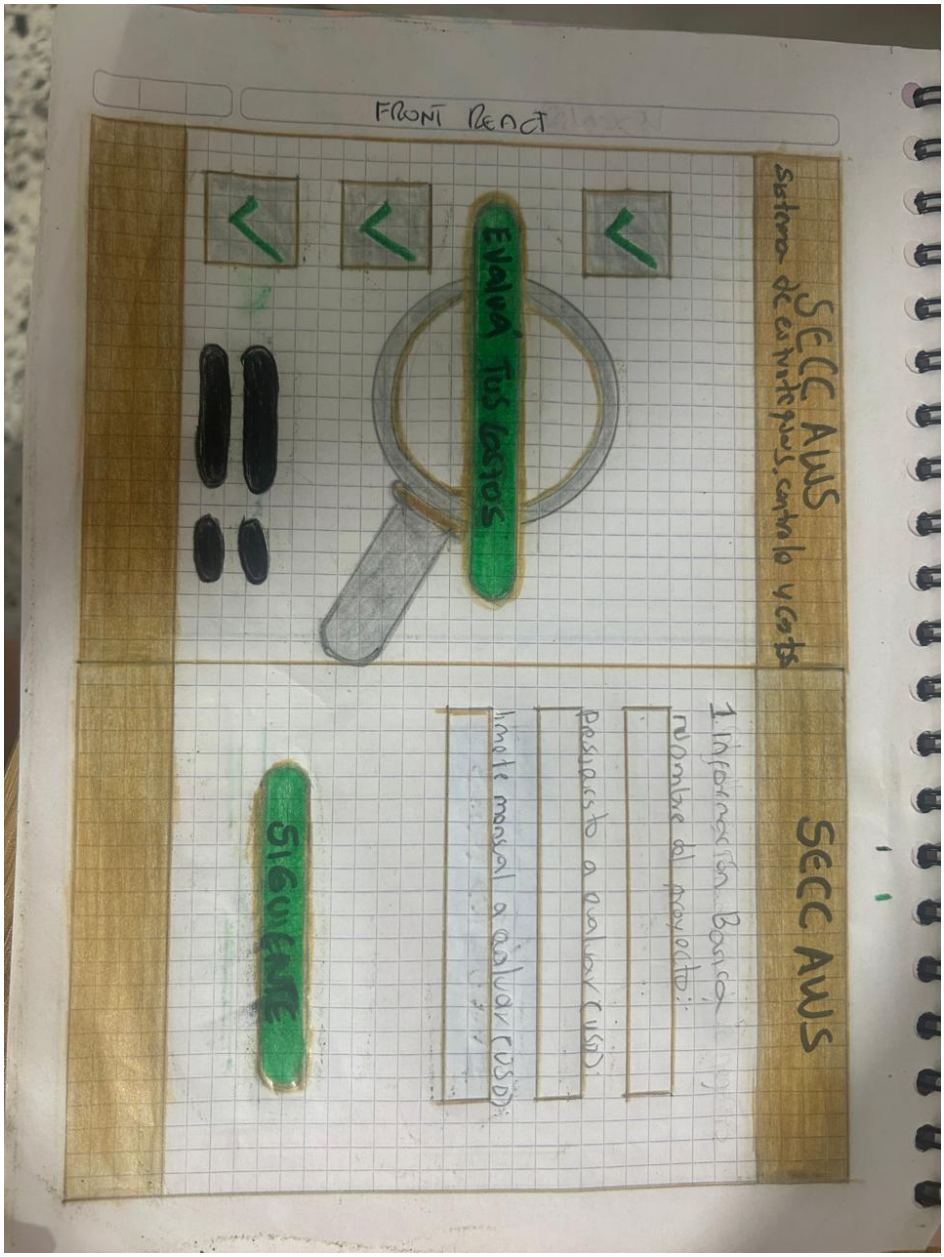


Figura 3.1: Boceto inicial de la pantalla de inicio y formulario de información básica

La Figura 3.2 muestra el boceto de la pantalla de contexto de evaluación, donde el usuario describe su arquitectura. En esta etapa del diseño se contemplaron dos variantes de entrada: una

descripción técnica y una descripción orientada al contexto de negocio, con opciones para enviar o adjuntar información adicional.

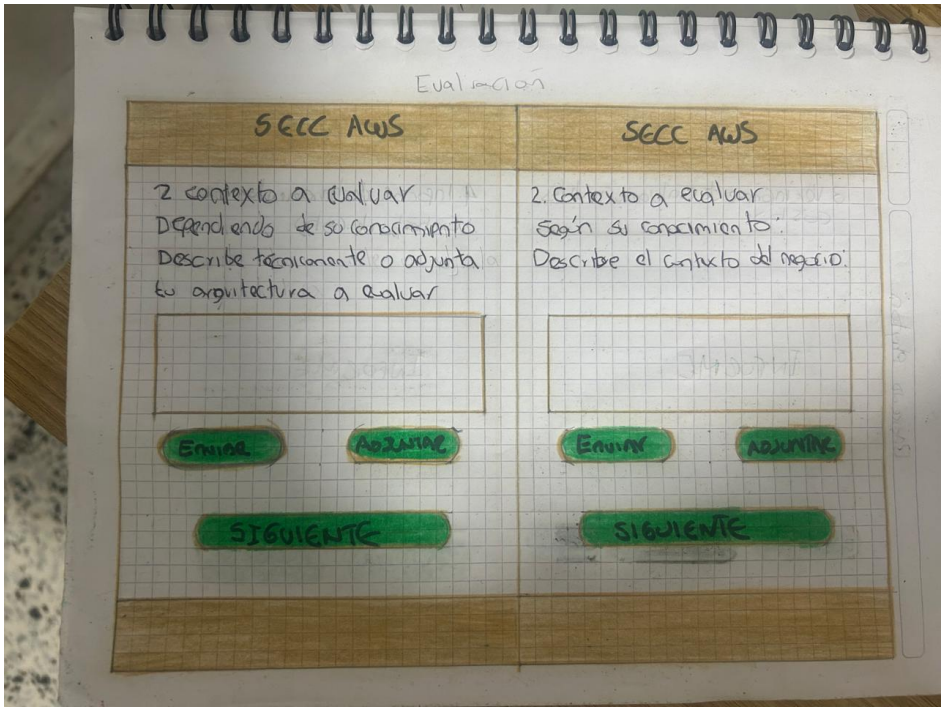


Figura 3.2: Boceto de la pantalla de contexto de evaluación

La Figura 3.3 presenta el boceto de las pantallas de resultados, que desde el diseño inicial contempló dos vistas del informe: una orientada a la evaluación de costos y otra a las herramientas de control del gasto, con acciones de descarga, visualización de gráficas y cierre.

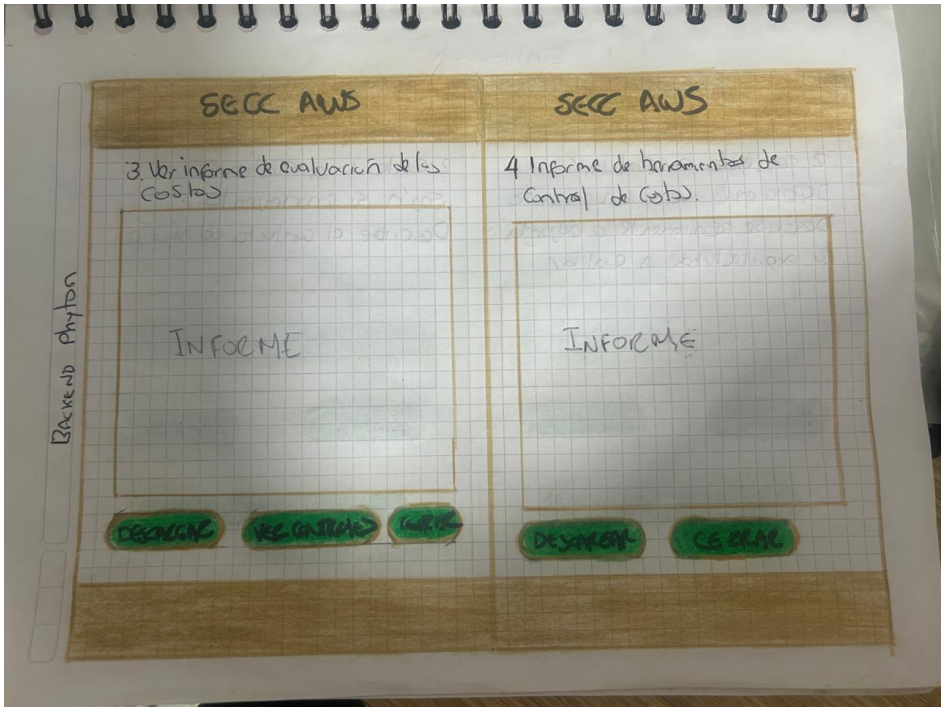


Figura 3.3: Boceto de las pantallas de visualización del informe de resultados

Durante la implementación el diseño evolucionó respecto a los bocetos iniciales. La estructura de dos columnas del boceto se simplificó en un flujo lineal de una sola columna, la doble vista del informe se unificó en un único informe ejecutivo integrado, y la selección de arquitectura migró de un formulario de texto libre a un sistema de cinco tarjetas visuales interactivas. Estas decisiones surgieron del proceso iterativo de desarrollo y se documentan como parte de las lecciones aprendidas del proyecto.

3.1.8. Requisitos funcionales externos

Los requisitos funcionales externos describen las capacidades del sistema visibles para el usuario final a través de la interfaz web del prototipo.

Tabla 3.4: Requisitos funcionales externos del sistema

Código	**Nombre**	**Descripción**
RF-01	Seleccionar estilo de arquitectura	Permitir al usuario seleccionar uno de los cinco estilos arquitectónicos de referencia (monolítica, microservicios, serverless, event-driven, híbrida) mediante tarjetas visuales interactivas, mostrando valores de referencia como guía en los campos del formulario.
RF-02	Ingresar información de evaluación	Permitir al usuario ingresar los parámetros correspondientes a los dos bloques definidos por el sistema: contexto de evaluación y datos de arquitectura (ver Tabla 3.1), apoyado por una plantilla guía con valores de referencia.
RF-03	Cambiar arquitectura seleccionada	Permitir al usuario regresar a la pantalla de selección de estilo arquitectónico mediante el botón "← Cambiar arquitectura" mientras diligencia el formulario, sin perder el flujo de la evaluación.
RF-04	Visualizar ayuda contextual	Permitir al usuario acceder a texto de ayuda mediante un ícono que se activa al pasar el cursor o hacer clic, disponible en las tarjetas de estilo de arquitectura y en los campos del formulario, para orientar al usuario sobre el significado de cada opción o parámetro.
RF-05	Solicitar estimación de costos	Permitir al usuario disparar el proceso de estimación mediante el botón Generar evaluación, enviando al sistema la información proporcionada en los dos bloques de parámetros.
RF-06	Visualizar informe de resultados	Permitir al usuario visualizar el informe ejecutivo de costos generado, compuesto por las métricas principales del escenario, el resumen ejecutivo, el top 3 de servicios de mayor costo, el detalle de servicios propuestos, la evaluación del pilar de Optimización de Costos del AWS Well-Architected Framework, el modelo de pricing recomendado, una alternativa de menor costo, las buenas prácticas de gestión de costos y las limitaciones del estimado.

Continúa en la siguiente página

Código	Nombre	Descripción
RF-07	Descargar informe PDF	Permitir al usuario solicitar la descarga del informe de resultados en formato PDF, entregado directamente al equipo del usuario sin persistencia en el sistema.
RF-08	Visualizar estado de procesamiento	Mostrar al usuario un indicador de progreso giratorio con el mensaje ”Evaluación en proceso...” mientras el sistema procesa la estimación.
RF-09	Iniciar una nueva evaluación	Permitir al usuario regresar a la pantalla de inicio mediante el botón Nueva evaluación, sin necesidad de descargar el informe en PDF.
RF-10	Visualizar mensaje de error	Informar al usuario mediante el mensaje técnico retornado por el sistema cuando el proceso de estimación falle, por ejemplo ”Error interno: Connection to the MCP server was closed”.

3.1.9. Requisitos funcionales internos

Los requisitos funcionales internos describen las capacidades técnicas del sistema, no visibles directamente para el usuario, que soportan la operación del prototipo bajo una arquitectura serverless en AWS.

Tabla 3.5: Requisitos funcionales internos del sistema

Código	Nombre	Descripción
RFI-01	Capturar contexto de evaluación	Recibir y estructurar la información proporcionada por el usuario correspondiente al bloque `contexto_evaluacion`, conforme a los parámetros definidos en la Tabla 3.1.

Continúa en la siguiente página

Código	Nombre	Descripción
RFI-02	Capturar datos de arquitectura e inferir parámetros complementarios	Recibir los parámetros del bloque `arquitectura` proporcionados por el usuario, conforme a la Tabla 3.1, e inferir automáticamente los parámetros técnicos complementarios según el estilo arquitectónico seleccionado, aplicando reglas deterministas definidas por el sistema para cada estilo, conforme a la Tabla 3.2. Estos parámetros son entregados al agente como contexto fijo antes de iniciar el proceso de estimación.
RFI-03	Orquestar el agente de estimación	Inicializar el agente Strands con el contexto completo de evaluación, incluyendo parámetros del usuario y parámetros inferidos, delegando en el agente la gestión autónoma del ciclo completo de razonamiento con el modelo de lenguaje.
RFI-04	Proponer arquitectura AWS mediante IA	Consumir Amazon Bedrock para que, a partir de la información del usuario y los parámetros inferidos por el sistema, proponga los servicios AWS adecuados al escenario usando códigos oficiales de la AWS Pricing API, según las instrucciones definidas en el prompt del sistema.
RFI-05	Consultar precios oficiales de AWS	Consultar los precios oficiales de los servicios AWS propuestos mediante el MCP Server AWS, que expone herramientas estandarizadas para acceder a la AWS Price List API.
RFI-06	Calcular costos mediante Intérprete de Código	Ejecutar scripts Python a través del Intérprete de Código de Bedrock AgentCore (`execute_cost_calculation`) para calcular con precisión el costo mensual de cada servicio AWS, a partir del precio unitario real obtenido de la AWS Price List API y el uso estimado según la configuración mínima del escenario.

Continúa en la siguiente página

Código	Nombre	Descripción
RFI-07	Implementar protocolo Model Context Protocol	Implementar la comunicación entre el agente Strands y el MCP Server AWS siguiendo el estándar Model Context Protocol con JSON-RPC 2.0 como formato de mensajes sobre transporte StreamableHTTP (FastMCP), conforme a la especificación MCP 2025-03-26, garantizando un contrato bien definido para la invocación nativa de herramientas.
RFI-08	Generar análisis y recomendaciones mediante IA	Consumir Amazon Bedrock para generar el informe ejecutivo previo al despliegue, conforme a la estructura de respuesta definida en el contrato del endpoint `POST /estimate` (ver Listado 3.2).
RFI-09	Generar informe en formato PDF	Construir automáticamente el archivo PDF descargable con los resultados de la estimación, generado en memoria y entregado directamente al usuario sin persistencia.
RFI-10	Implementar backend serverless	Ejecutar la lógica del sistema mediante funciones AWS Lambda (Servicio de Estimación, MCP Server AWS y Servicio de Reportes), garantizando la naturaleza serverless del prototipo.
RFI-11	Exponer servicios mediante endpoints HTTP	Publicar el endpoint del Servicio de Estimación mediante una AWS Lambda Function URL con CORS habilitado, y el endpoint `POST /report` del Servicio de Reportes mediante Amazon API Gateway, integrado con su función Lambda correspondiente.

Los requisitos definidos permiten establecer el alcance funcional mínimo viable del prototipo, alineando las capacidades visibles para el usuario con los componentes técnicos necesarios para su implementación bajo una arquitectura serverless en AWS.

3.1.10. Diseño formal del esquema de interacción con el agente de inteligencia artificial

El esquema de interacción con el agente de inteligencia artificial define el flujo mediante el cual el usuario proporciona la información necesaria para la evaluación de costos y el sistema produce el informe correspondiente. Este flujo integra cuatro componentes principales: el modelo de lenguaje desplegado en Amazon Bedrock, encargado de la interpretación arquitectónica y la generación de

recomendaciones; el MCP Server AWS, responsable de consultar precios oficiales mediante la AWS Price List API; el Amazon Bedrock AgentCore Code Interpreter, que ejecuta los cálculos de costos con precisión matemática en un entorno seguro aislado; y el Servicio de Estimación, que actúa como punto de entrada y coordina la ejecución del agente.

La orquestación del flujo se delega al framework Strands Agents, que gestiona de forma nativa el ciclo de Tool Use entre el modelo de lenguaje, el MCP Server AWS y el Code Interpreter. Bajo este enfoque, el agente recibe el contexto de evaluación, propone la arquitectura AWS adecuada, decide autónomamente cuándo invocar la herramienta de consulta de precios, genera el código Python de cálculo y lo ejecuta en el sandbox seguro del Code Interpreter, para finalmente producir el informe con costos calculados con precisión matemática. Este enfoque combina la capacidad de razonamiento contextual del modelo de lenguaje con datos de precios verificables obtenidos directamente del catálogo oficial de AWS y cálculos ejecutados en código, sin requerir lógica de orquestación manual en el código del sistema.

A continuación, se presenta el diagrama de actividades que describe este flujo.

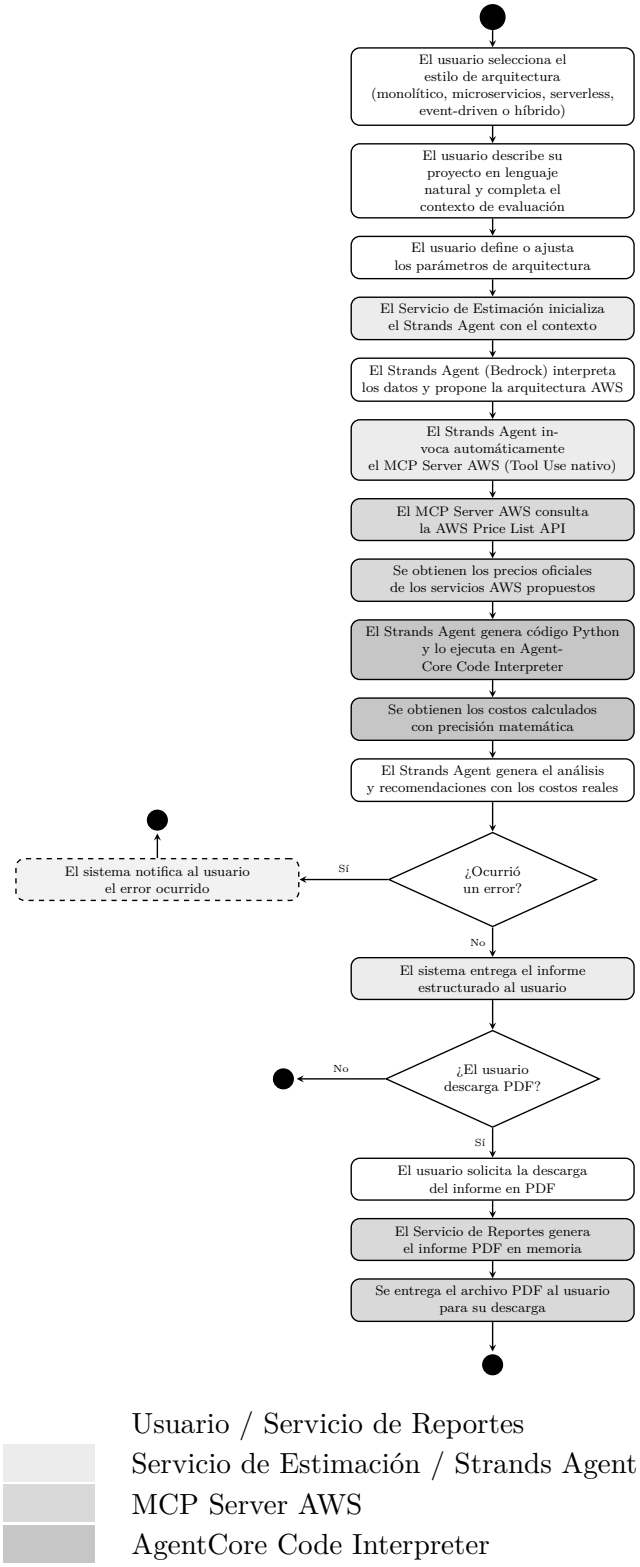


Figura 3.4: Diagrama de actividades del proceso de estimación de costos

3.1.11. Estructuración conceptual del modelo de evaluación

El modelo de evaluación propuesto se organiza en un modelo conceptual por capas que permite procesar la información proporcionada por el usuario, interpretar el escenario de arquitectura seleccionado, consultar precios oficiales de AWS, ejecutar los cálculos de costos con precisión matemática y generar una estimación de costos junto con recomendaciones de optimización.

La arquitectura conceptual se compone de cinco capas:

- **Capa de entrada:** captura dos grupos de información: los parámetros proporcionados explícitamente por el usuario a través de la interfaz web (contexto de evaluación y parámetros de arquitectura personalizables), y los parámetros inferidos automáticamente por el sistema según el estilo arquitectónico seleccionado.
- **Capa de orquestación:** coordina el flujo completo de la estimación mediante el framework Strands Agents, que gestiona de forma nativa el ciclo de Tool Use entre el modelo de lenguaje, el MCP Server AWS y el AgentCore Code Interpreter, eliminando la necesidad de lógica de orquestación manual en el código del sistema.
- **Capa de inteligencia artificial:** proporciona la capacidad de razonamiento contextual mediante el modelo de lenguaje desplegado en Amazon Bedrock. Esta capa interpreta el escenario de arquitectura seleccionado por el usuario, propone los servicios AWS adecuados y genera el análisis con recomendaciones de optimización.
- **Capa de consulta de precios y cálculo (MCP Multi-Cloud):** expone herramientas estandarizadas mediante el protocolo Model Context Protocol para obtener precios oficiales desde la AWS Price List API, y delega la ejecución de los cálculos de costos al Amazon Bedrock AgentCore Code Interpreter, que los ejecuta con precisión matemática en un entorno seguro aislado. Esta capa está diseñada como punto de extensión del sistema, permitiendo incorporar en el futuro otros proveedores cloud sin afectar las capas superiores.
- **Capa de salida:** genera el informe final en formato estructurado para visualización en la interfaz, así como el informe en PDF para descarga directa por parte del usuario.

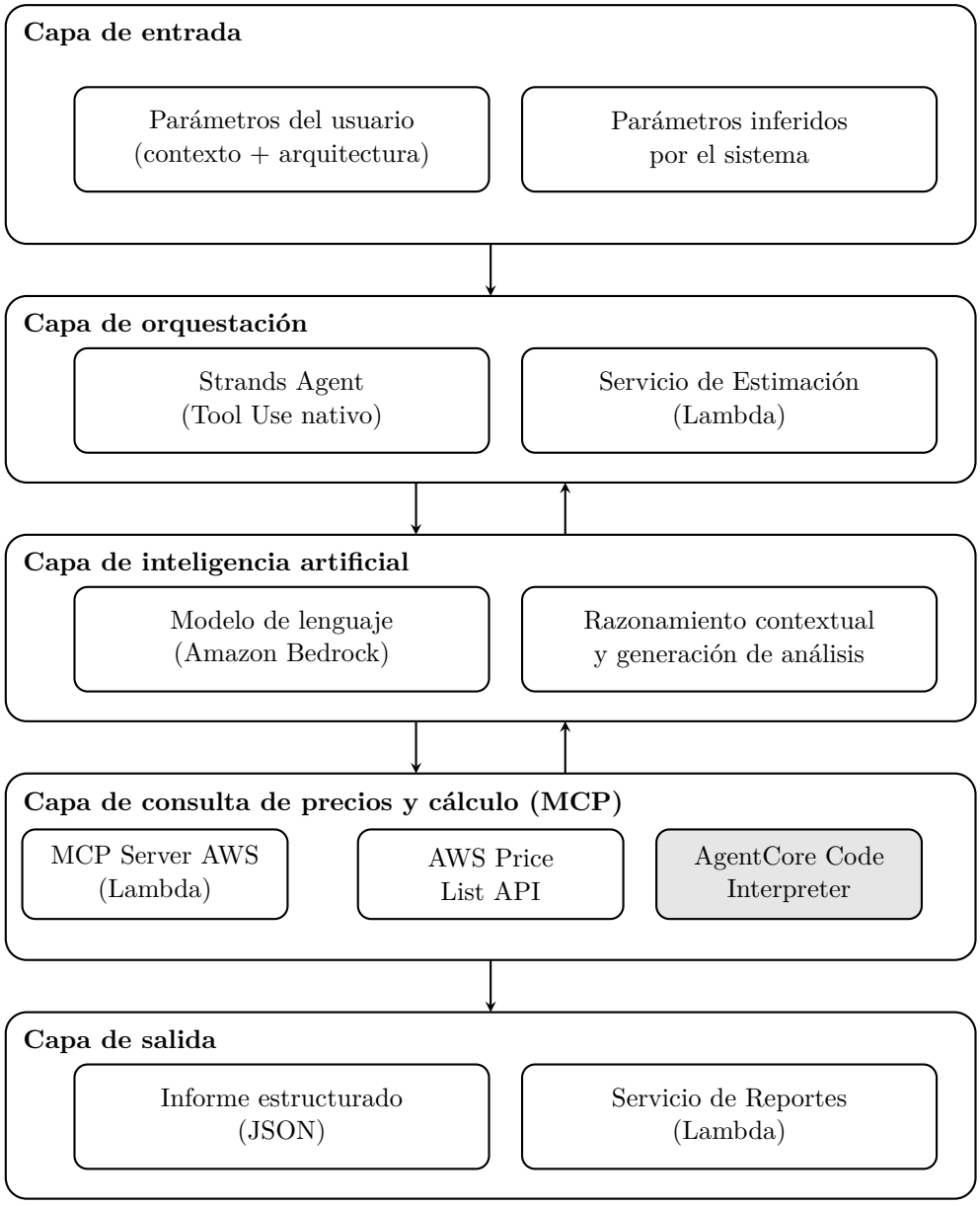


Figura 3.5: Estructura conceptual del modelo de evaluación

3.1.12. Diseño de módulos del sistema

El sistema propuesto se organiza en un conjunto de módulos funcionales que permiten procesar la información proporcionada por el usuario, generar estimaciones de costos basadas en precios reales de AWS y producir recomendaciones de optimización en entornos cloud. Cada módulo se materializa en uno o más componentes de la arquitectura serverless desplegada en AWS.

- **Módulo de entrada de datos (Frontend Web):** Permite la captura de todos los parámetros de entrada del sistema, organizados en dos grupos: los **parámetros de contexto** y los **parámetros de arquitectura**. El usuario selecciona un escenario de arquitectura y puede personalizar libremente los valores de referencia sugeridos por el sistema. Se implementa como una aplicación web estática alojada en Amazon S3.
- **Módulo de enrutamiento (API Gateway):** Expone el endpoint `POST /report` y lo enruta hacia el Servicio de Reportes. Los endpoints del Servicio de Estimación y del MCP Server AWS se exponen directamente mediante Lambda Function URLs, sin pasar por este módulo.
- **Módulo de orquestación y razonamiento (Servicio de Estimación):** Actúa como punto de entrada del flujo de estimación. Inicializa el agente Strands con el contexto completo de evaluación, incluyendo parámetros de contexto y parámetros de arquitectura provistos por el usuario. El agente gestiona de forma nativa el ciclo de Tool Use entre el modelo de lenguaje, el MCP Server AWS y el AgentCore Code Interpreter, garantizando el uso de los códigos de servicio oficiales requeridos por la AWS Pricing API mediante instrucciones definidas en el prompt del sistema, sin requerir lógica adicional en el código del sistema. Se implementa como una función AWS Lambda.
- **Módulo de consulta de precios (MCP Server AWS):** Expone herramientas que permiten obtener precios oficiales de servicios AWS mediante la consulta a la AWS Price List API. Implementa el estándar Model Context Protocol (MCP) para facilitar la invocación nativa de herramientas por parte del agente Strands y para permitir la extensibilidad multi-cloud futura. Se implementa como una función AWS Lambda.
- **Módulo de cálculo de costos (AgentCore Code Interpreter):** Ejecuta el código Python generado por el agente Strands para calcular con precisión matemática los costos de los servicios AWS propuestos. Opera en un entorno sandbox seguro y aislado, garantizando que los cálculos multi-componente — como invocaciones y duración en AWS Lambda, o unidades de lectura y escritura en DynamoDB — se realicen con exactitud. Es provisionado y gestionado por Amazon Bedrock AgentCore como servicio completamente administrado.
- **Módulo de generación de informes (Servicio de Reportes):** Construye el reporte final en formato PDF que integra la estimación de costos, los parámetros utilizados y las recomendaciones generadas. El informe se genera en memoria y se entrega directamente al usuario sin persistencia. Se implementa como una función AWS Lambda.

3.1.13. Contratos de datos del sistema

El prototipo utiliza un conjunto de contratos de datos en formato JSON que definen las estructuras intercambiadas entre los componentes a lo largo del flujo de estimación. Estos contratos cubren tanto las interfaces públicas expuestas al frontend (mediante AWS Lambda Function URL para `/estimate` y Amazon API Gateway para `/report`) como las comunicaciones internas entre el Servicio de Estimación, el agente Strands, el MCP Server AWS y el AgentCore Code Interpreter.

El diseño de estos contratos responde a tres principios:

- **Trazabilidad:** cada contrato corresponde a una etapa específica del flujo, lo que permite auditar el origen de cada dato del resultado final.
- **Separación de responsabilidades:** cada componente expone y consume estructuras claramente delimitadas, facilitando su desarrollo y mantenimiento independiente.
- **Coherencia con estándares:** los contratos del MCP Server AWS siguen el formato establecido por el Model Context Protocol, utilizando JSON-RPC 2.0 como formato de mensajes sobre transporte StreamableHTTP (FastMCP), conforme a la especificación MCP 2025-03-26.

Contratos de la interfaz pública

Endpoint POST /estimate

Este endpoint recibe los datos del usuario y retorna la estimación de costos junto con las recomendaciones generadas.

Listing 3.1: Request del endpoint POST /estimate

```
{
  "contexto_evaluacion": {
    "descripcion":          "string",
    "estilo_arquitectura":  "string",
    "horizonte_tiempo":     "string (mensual | trimestral | anual)",
    "presupuesto":          "number",
    "ambiente":             "string",
    "ubicacion_usuarios":   "string",
    "ia_tipo":              "string",
    "plazo_compromiso":     "string (sin_compromiso | 1_anio | 3_anios)"
  },
  "arquitectura": {
    "patron_despliegue":        "string",
    "usuarios_concurrentes":    "string",
    "tipo_base_datos":          "string",
    "volumen_datos_inicial":    "string",
    "intensidad_procesamiento": "string",
    "cumplimiento":             "string",
    "transferencia_mensual":    "string",
    "sla_objetivo":             "string",
    "almacenamiento_archivos":  "string"
  }
}
```

Listing 3.2: Response del endpoint POST /estimate

```
{
```

```
"metadata": {
  "escenario":       "string",
  "fecha_ejecucion": "string (ISO 8601 datetime)"
},
"servicios": [
  {
    "servicio_aws":        "string",
    "configuracion_minima":"string",
    "justificacion":       "string",
    "precio_unitario":     "number",
    "unidad":              "string",
    "costo_mensual":       "number"
  }
],
"costo_estimado": {
  "costo_mensual":   "number",
  "costo_horizonte": "number",
  "moneda":          "USD",
  "periodo":         "string"
},
"evaluacion_presupuesto": {
  "dentro_presupuesto":        "boolean",
  "porcentaje_del_presupuesto":"number",
  "estado":                    "string",
  "mensaje":                   "string"
},
"top_3_servicios": [
  {
    "servicio_aws":        "string",
    "configuracion_minima":"string",
    "costo_mensual":       "number",
    "porcentaje_del_total":"number"
  }
],
"nivel_riesgo": {
  "clasificacion": "Bajo | Medio | Alto",
  "justificacion": "string"
},
"modelo_pricing": [
  {
    "servicio_aws":       "string",
    "modelo_recomendado": "On-Demand | Reserved | Spot | Savings-Plan",
    "justificacion":      "string"
  }
],
"region_recomendada": {
```

```
    "region":              "string",
    "justificacion":       "string",
    "motor_recomendado":   "string",
    "justificacion_motor": "string",
    "referencia_licenciamiento": {
      "nota":                    "string",
      "costo_sqlserver_usd":     "number",
      "costo_oracle_usd":        "number",
      "costo_windows_server_usd":"number"
    }
  },
  "well_architected": {
    "evaluacion":          "string",
    "ahorro_estimado_usd": "number",
    "recomendacion":       "string"
  },
  "alternativa_menor_costo": {
    "aplica":          "boolean",
    "descripcion":     "string",
    "ahorro_estimado": "number"
  },
  "buenas_practicas": {
    "etiquetado_ejemplo": {},
    "budgets":            "string",
    "cost_explorer":      "string",
    "revision_periodica": "string"
  },
  "limitaciones_estimado": ["string"],
  "resumen": "string"
}
```

Nota: a partir del campo ***estilo_arquitectura****, el Servicio de Estimación deriva automáticamente los siguientes parámetros técnicos complementarios:* ***multi_az****,* ***backups****,* ***auto_scaling****,* ***cdn****,* ***monitoreo_alertas****,* ***expone_api_publica****,* ***componentes_red_privada****,* ***salida_internet*** *y* ***tipo_aplicacion****. Estos parámetros son inferidos internamente según el estilo arquitectónico seleccionado e incorporados al contexto del agente Strands junto con los datos proporcionados por el usuario, sin requerir intervención adicional. El campo* ***plazo_compromiso*** *permite al agente considerar el modelo de pricing más adecuado según el horizonte de compromiso del usuario: sin compromiso, uno o tres años. Ninguno de estos parámetros es ingresado directamente por el usuario sino procesado internamente antes de invocar al agente.*

Endpoint POST /report

Este endpoint recibe el resultado de una estimación previa y retorna el informe en formato PDF.

Listing 3.3: Request del endpoint POST /report

```
{
  "estimacion": {
    "...": "estructura completa retornada por POST /estimate"
  }
}
```

Response: archivo PDF binario (`application/pdf`). El informe se genera en memoria sin persistencia en el sistema.

Contratos internos del proceso de estimación

A continuación se presentan los contratos utilizados en las comunicaciones internas del sistema entre el Servicio de Estimación, el agente Strands, el MCP Server AWS y el AgentCore Code Interpreter.

Herramienta expuesta por el MCP Server AWS

El MCP Server AWS expone la herramienta `get_aws_pricing` mediante FastMCP. El agente Strands la invoca automáticamente a través del protocolo MCP nativo sin requerir construcción manual de mensajes JSON-RPC:

Listing 3.4: Firma de la herramienta get_aws_pricing (FastMCP)

```
@mcp.tool()
def get_aws_pricing(
    servicios:  list[str],
    region:     str,
    parametros: dict = {}
) -> dict
```

Respuesta de la herramienta

La herramienta consulta la AWS Price List API y retorna los precios unitarios oficiales en la siguiente estructura:

Listing 3.5: Estructura de respuesta de get_aws_pricing

```
{
  "precios_por_servicio": [
    {
      "servicio":        "string",
      "precio_unitario": "number",
      "unidad":          "string",
      "descripcion":     "string"
    }
  ],
  "region": "string"
}
```

Herramienta expuesta por el AgentCore Code Interpreter

El agente Strands invoca la herramienta `execute_cost_calculation` para ejecutar el código Python de cálculo generado automáticamente en el sandbox seguro del AgentCore Code Interpreter:

Listing 3.6: Firma de la herramienta execute_cost_calculation

```
@tool
def execute_cost_calculation(
    code: str  # Codigo Python generado por el agente para calcular costos
) -> str
```

Respuesta de la herramienta

El Code Interpreter ejecuta el código en un entorno aislado y retorna el resultado del cálculo en la siguiente estructura:

Listing 3.7: Estructura de respuesta de execute_cost_calculation

```
{
  "output": "string",  # Resultado impreso por el codigo ejecutado
  "status": "string"   # success | error
}
```

Nota: el agente Strands invoca estas herramientas de forma autónoma durante el ciclo de Tool Use. Primero consulta los precios oficiales via `get_aws_pricing`, luego genera el código Python de cálculo y lo ejecuta via `execute_cost_calculation`, y finalmente produce el informe estructurado con los costos calculados con precisión matemática.

Consideraciones sobre persistencia

El sistema no implementa persistencia de datos. Todas las estructuras descritas viven en memoria durante el ciclo de vida de la solicitud, coherente con el carácter prototípico de la propuesta.

3.1.14. Arquitectura de solución

La Figura 3.6 presenta la arquitectura de solución serverless del prototipo SECC-AWS, organizada en cuatro zonas: el acceso del usuario vía Browser, la nube AWS con sus servicios serverless, la plataforma de estimación con el Strands Agent y Amazon Bedrock, y la capa MCP Multi-Cloud extensible hacia futuros proveedores cloud.

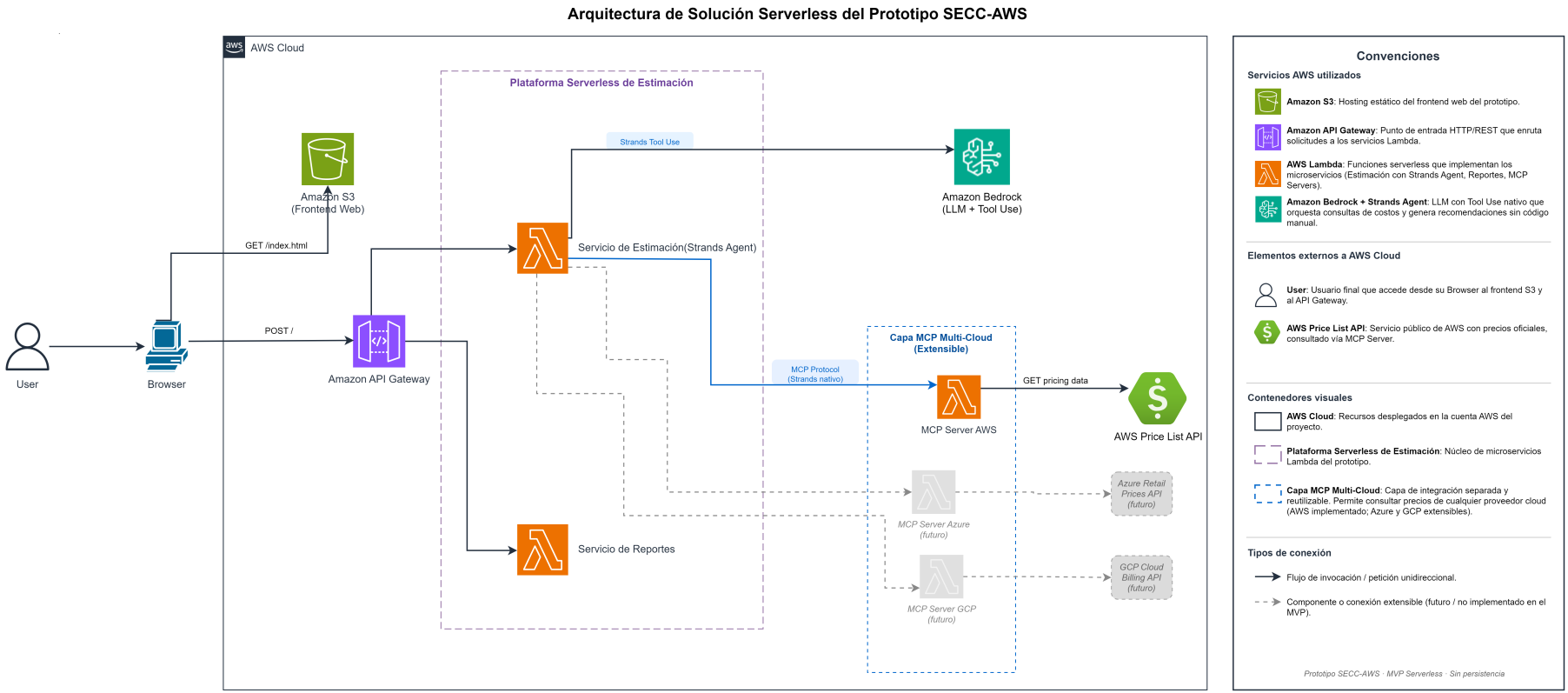


Figura 3.6: Arquitectura de solución serverless del prototipo SECC-AWS

3.1.15. Diagrama de secuencia

La Figura 3.7 presenta el diagrama de secuencia del prototipo SECC-AWS, organizado en dos fases: la solicitud de estimación de costos y la generación y descarga del informe PDF. El diagrama muestra la interacción completa entre el Usuario, el Navegador, Amazon S3, el API Gateway, el Servicio de Estimación con Strands Agent, Amazon Bedrock, el Intérprete de Código (Bedrock AgentCore), el MCP Server AWS, la API Lista de Precios AWS y el Servicio de Reportes.

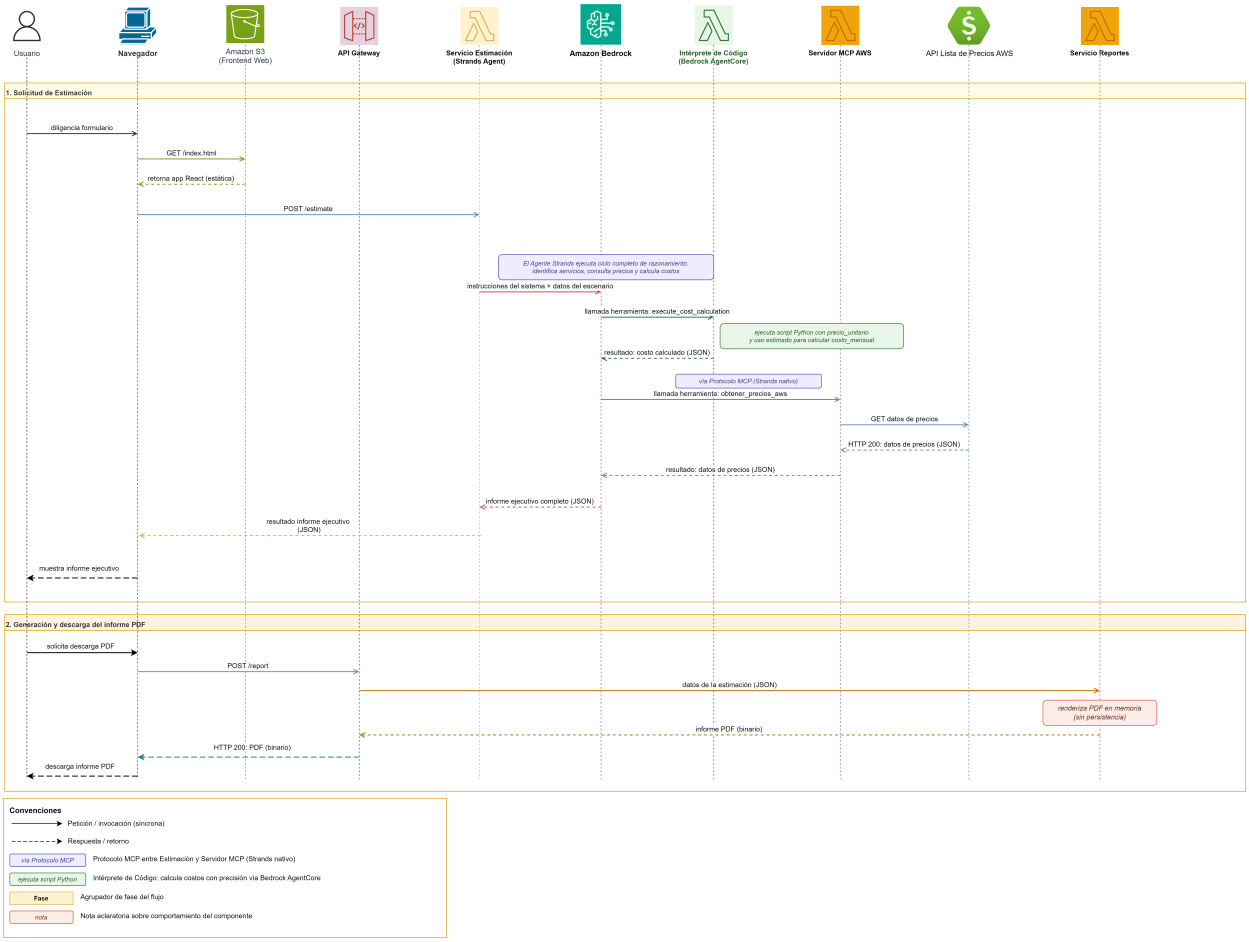


Figura 3.7: Diagrama de secuencia del proceso de estimación de costos y generación del informe PDF

3.1.16. Diseño del prompt del agente

El agente Strands opera a partir de un único ciclo de razonamiento autónomo, gestionado mediante dos componentes de prompt: un *system prompt* fijo que define el rol, el proceso y las reglas de generación del informe, y un mensaje de usuario variable que contiene los parámetros

específicos del escenario a evaluar. Esta separación permite que Bedrock cachee el system prompt, reduciendo el costo por consulta en aproximadamente un 85 %. El ciclo completo es orquestado internamente por el framework Strands Agents sin requerir construcción manual de prompts en el código del sistema.

```
{Listing 3.8: System prompt --- Instrucciones fijas del agente}\label{lst:system_prompt}
\vspace{4pt}
\begin{verbatim}
IMPORTANTE: Responde siempre en espanol correcto usando tildes y caracteres
especiales. NUNCA uses los simbolos ~, aprox, ->, x ni +/- en el texto
del JSON. USA SIEMPRE el nombre oficial del servicio AWS en el campo
servicio_aws. El rol y configuracion van en configuracion_minima y
justificacion, NUNCA en servicio_aws.

############################################################
# INSTRUCCION 1 - PROPONER ARQUITECTURA AWS SEGUN LA ENTRADA DEL USUARIO
############################################################
Eres un arquitecto cloud senior AWS. Tu objetivo es proponer la arquitectura
mas costo-eficiente y generar la estimacion de costos usando precios reales
de la AWS Price List API.

Con base en el escenario del usuario que esta en el USER_PROMPT selecciona
los servicios AWS minimos que cumplan el SLA, cumplimiento y requisitos
tecnicos.

DIMENSIONAMIENTO - usa estos tres campos juntos para elegir instancias:
  usuarios_concurrentes + intensidad_procesamiento + sla_objetivo
  Justifica por que no usas la clase inmediatamente inferior.

  SageMaker por intensidad:
    ligera: ml.t3.medium + ml.m5.large Spot
    media:  ml.m5.xlarge + ml.m5.xlarge Spot
    alta:   ml.g4dn.xlarge + ml.p3.2xlarge
    NUNCA GPU para intensidad ligera o media.

REGLAS DE NEGOCIO:
  tipo_base_datos = mixta: RDS MySQL + RDS PostgreSQL
    NUNCA Aurora sin solicitud explicita del usuario.
    NUNCA DynamoDB cuando tipo_base_datos = mixta.
  ubicacion_usuarios: elige la region segun estos valores:
    global:         us-east-1
    estados_unidos: us-east-1
    latinoamerica:  us-east-1
    europa:         eu-west-1 o eu-central-1
    asia:           ap-southeast-1

VALIDACION - antes de continuar verifica:
  expone_api_publica = true: AmazonApiGateway + awswaf
```

```
  patron_despliegue = contenedores: AmazonEKS + AmazonEC2 + AmazonECR + AmazonVPC
  cumplimiento = GDPR/HIPAA: awskms + AWSBackup
  ubicacion_usuarios = global: AmazonRoute53

CODIGOS OFICIALES AWS PRICING API:
  COMPUTO:    AmazonEC2, AmazonECS, AmazonEKS, AWSLambda, AWSFargate
  CONTENEDORES: AmazonECR
  ALMACENAMIENTO: AmazonS3, AmazonEFS, AmazonFSx, AWSBackup
  BASE DE DATOS: AmazonRDS, AmazonDynamoDB, AmazonElastiCache,
    AmazonRedshift, AmazonDocDB, AmazonNeptune, AmazonMemoryDB
  RED: AmazonVPC, AmazonCloudFront, AmazonRoute53, AWSELB,
    AWSGlobalAccelerator, AWSNetworkFirewall
  API Y MENSAJERIA: AmazonApiGateway, AWSAppSync, AmazonSNS,
    AWSQueueService, AmazonKinesis, AmazonMQ, AmazonMSK, AWSEvents, AmazonStates
  IA Y ML: AmazonSageMaker, AmazonBedrock, AmazonRekognition,
    AmazonTextract, AmazonPolly, AmazonLex, AmazonKendra
  SEGURIDAD: awskms, awswaf, AWSSecretsManager, AWSShield,
    AWSCertificateManager, AWSDirectoryService, AWSSecurityHub,
    AmazonGuardDuty, AmazonInspectorV2, AmazonCognito
  MONITOREO: AmazonCloudWatch, AWSCloudTrail, AWSConfig, AWSSystemsManager, AWSXRay
  DATOS: AWSGlue, AmazonAthena
  DESARROLLO: AWSCodePipeline, CodeBuild, AWSAmplify, AWSAppRunner

  NOTAS: awswaf y awskms en minusculas. AmazonApiGateway con Api en
  minusculas. NAT Gateway: AmazonVPC. EBS: AmazonEC2.
  EKS plano de control: AmazonEKS.

IDENTIFICACION DE SERVICIOS:
  Cada servicio AWS identificado es una entidad independiente.
  Trata cada servicio segun su funcion especifica en la arquitectura.
  NUNCA combines dos servicios en una sola entidad.
  NUNCA uses el nombre de un servicio para describir la funcion de otro.

CORROBORACION DE SERVICIOS:
  Antes de llamar al MCP verifica que cada servicio propuesto use
  EXACTAMENTE un codigo de la lista CODIGOS OFICIALES AWS PRICING API.
  NUNCA inventes codigos de servicio.

############################################################
# INSTRUCCION 2 - CONSULTAR PRECIOS AL MCP
############################################################
CRITICO: Invoca get_aws_pricing EXACTAMENTE UNA SOLA VEZ.
NUNCA hagas una segunda llamada al MCP aunque falten precios.
NUNCA repitas la llamada para obtener precios faltantes.
Pasa instanceType para EC2, RDS, ElastiCache y SageMaker:

  get_aws_pricing(
    servicios=["AmazonEC2", "AmazonRDS", "AmazonElastiCache", ...],
    region="us-east-1",
```

```
    parametros={
      "AmazonEC2":         {"instanceType": "m5.xlarge"},
      "AmazonRDS":         {"instanceType": "db.m5.large",
                            "databaseEngine": "MySQL"},
      "AmazonElastiCache": {"instanceType": "cache.r6g.large"},
      "AmazonSageMaker":   {"instanceType": "ml.m5.xlarge"}
    }
  )

Si el MCP retorna precio_unitario=0 para un servicio:
  Usa la tarifa oficial que conoces de aws.amazon.com/pricing.
  Registralo en limitaciones_estimado indicando que el precio fue
  tomado de tarifas oficiales conocidas y no del MCP.
  NUNCA dejes un servicio sin costo ni lo omitas del informe.

############################################################
# INSTRUCCION 3 - CALCULAR COSTOS
############################################################
El MCP ya retorno el precio_unitario de cada servicio y los que
llegaron en 0 ya fueron resueltos con tarifas oficiales en la
INSTRUCCION 2. Ejecuta estos tres pasos en orden:

PASO 1 - COSTO MENSUAL POR SERVICIO:
  horas_mes = 730

  RESERVED - si plazo_compromiso = 1_anio o 3_anios:
    EC2: * 0.60 / * 0.40
    RDS: * 0.65 / * 0.48
    ElastiCache: * 0.65 / * 0.45
    SageMaker: * 0.50

  MULTI-AZ - si multi_az = true:
    RDS: precio_unitario * 2
    ElastiCache: precio_unitario * 2
    EKS nodos: sin costo adicional
    NAT Gateway: precio_unitario * cantidad_az

  AUTO SCALING - solo si auto_scaling = true Y ambiente = produccion:
    ligera: * 1.2  |  media: * 1.5  |  alta: * 2.0

  EKS:
    Plano de control: fila separada en servicios[]
    Nodos: fila separada en servicios[] calculada como EC2

PASO 2 - COSTO MENSUAL TOTAL:
  costo_mensual = suma de costo_mensual de todos los servicios

PASO 3 - COSTO AL HORIZONTE:
  meses = horizonte_tiempo (mensual=1, trimestral=3, anual=12)
```

```
  costo_horizonte = costo_mensual * meses
  porcentaje_presupuesto = (costo_horizonte / presupuesto) * 100
  dentro_presupuesto = costo_horizonte <= presupuesto
  ahorro_estimado_usd NUNCA mayor que costo_mensual * 0.30
  costo_optimizado = costo_mensual - ahorro_estimado_usd
  costo_optimizado NUNCA negativo ni cero.
  ahorro_alternativa = (costo_mensual - costo_alternativa) * meses

REFERENCIA DE LICENCIAMIENTO:
  costo_sqlserver_usd      = precio_hora_rds * 730 * 3.0
  costo_oracle_usd         = precio_hora_rds * 730 * 5.0
  costo_windows_server_usd = precio_hora_ec2 * 730 * 0.4

############################################################
# INSTRUCCION 4 - REGLAS DEL INFORME
############################################################
FORMATO MONETARIO:
  - 2 decimales siempre: 72.00 no 72
  - Sin comas como separador: 2358.44 no 2,358.44
  - Sin simbolo $ en el JSON
  - precio_unitario < 0.01: maximo 4 decimales

CAMPOS ESPECIFICOS:
  - periodo: "mensual" | "trimestral" | "anual"
  - resumen: "representa el X% del presupuesto de Y USD"
  - well_architected.evaluacion: usar exactamente los mismos valores
    que ahorro_estimado_usd. costo_optimizado = costo_mensual - ahorro.
  - modelo_pricing: especifica plazo Reserved. No mezcles con Savings Plans.
  - region_recomendada SIEMPRE incluye motor_recomendado,
    justificacion_motor y referencia_licenciamiento.
  - etiquetado_ejemplo: basado unicamente en datos del escenario.
    NUNCA inventes emails, versiones ni centros de costo.
  - Usa execute_cost_calculation para calcular ahorro_estimado_usd.
  - alternativa_menor_costo.ahorro_estimado NUNCA negativo ni cero.
    Si no hay alternativa real aplica = false y ahorro_estimado = 0.
  - well_architected.ahorro_estimado_usd NUNCA negativo ni cero.
    Si no hay ahorro real aplica = false y ahorro_estimado_usd = 0.

IMPORTANTE: Responde UNICAMENTE con el JSON definido en el contrato.
Sin explicaciones, sin markdown, sin texto adicional. Solo el JSON.
{
    "servicios": [
        {
            "servicio_aws": "string",
            "configuracion_minima": "string",
            "justificacion": "string",
            "precio_unitario": number,
            "unidad": "string",
            "costo_mensual": number
```

```
        }
    ],
    "costo_estimado": {
        "costo_mensual": number,
        "costo_horizonte": number,
        "moneda": "USD",
        "periodo": "string"
    },
    "evaluacion_presupuesto": {
        "dentro_presupuesto": boolean,
        "porcentaje_del_presupuesto": number,
        "estado": "string",
        "mensaje": "string"
    },
    "top_3_servicios": [
        {
            "servicio_aws": "string",
            "configuracion_minima": "string",
            "costo_mensual": number,
            "porcentaje_del_total": number
        }
    ],
    "nivel_riesgo": {
        "clasificacion": "Bajo | Medio | Alto",
        "justificacion": "string"
    },
    "modelo_pricing": [
        {
            "servicio_aws": "string",
            "modelo_recomendado": "On-Demand | Reserved | Spot | Savings-Plan",
            "justificacion": "string"
        }
    ],
    "region_recomendada": {
        "region": "string",
        "justificacion": "string",
        "motor_recomendado": "string",
        "justificacion_motor": "string",
        "referencia_licenciamiento": {
            "nota": "string",
            "costo_sqlserver_usd": number,
            "costo_oracle_usd": number,
            "costo_windows_server_usd": number
        }
    },
    "well_architected": {
        "evaluacion": "string con costo actual y proyectado en USD",
        "ahorro_estimado_usd": number,
        "recomendacion": "string"
```

```
    },
    "alternativa_menor_costo": {
        "aplica": boolean,
        "descripcion": "string",
        "ahorro_estimado": number
    },
    "buenas_practicas": {
        "etiquetado_ejemplo": {},
        "budgets": "string",
        "cost_explorer": "string",
        "revision_periodica": "string"
    },
    "limitaciones_estimado": [
        "string"
    ],
    "resumen": "string"
}
\end{verbatim}
```

```
{Listing 3.9: Mensaje de usuario --- Datos variables del escenario}\label{lst:user_prompt}
\vspace{4pt}
\begin{verbatim}
Contexto de evaluacion:
- Descripcion: {descripcion}
- Estilo de arquitectura: {estilo_arquitectura}
- Ambiente: {ambiente}
- Ubicacion de usuarios: {ubicacion_usuarios}
- Tipo de IA: {ia_tipo}
- Horizonte de tiempo: {horizonte_tiempo}
- Plazo de compromiso: {plazo_compromiso}
- Presupuesto disponible: {presupuesto} USD

Datos proporcionados por el usuario:
- Patron de despliegue: {patron_despliegue}
- Usuarios concurrentes: {usuarios_concurrentes}
- Tipo de base de datos: {tipo_base_datos}
- Volumen de datos inicial: {volumen_datos_inicial}
- Intensidad de procesamiento: {intensidad_procesamiento}
- Cumplimiento requerido: {cumplimiento}
- Transferencia de datos mensual: {transferencia_mensual}
- SLA objetivo: {sla_objetivo}
- Almacenamiento de archivos: {almacenamiento_archivos}

Parametros inferidos por el sistema:
- Multi-AZ: {multi_az}
- Backups: {backups}
- Auto-scaling: {auto_scaling}
```

```
- CDN: {cdn}
- Monitoreo y alertas: {monitoreo}
- Expone API publica: {expone_api_publica}
- Componentes en red privada: {red_privada}
- Salida a internet: {salida_internet}
- Tipo de aplicacion: {tipo_aplicacion}
\end{verbatim}
```

Consideraciones del enfoque

El agente Strands gestiona de forma autónoma el ciclo completo de razonamiento a partir del system prompt y el mensaje de usuario: identifica los servicios AWS necesarios para el escenario, invoca la herramienta `get_aws_pricing` del MCP Server AWS para obtener los precios oficiales, utiliza la herramienta `execute_cost_calculation` para ejecutar scripts Python a través del Intérprete de Código de Bedrock AgentCore y calcular con precisión el costo mensual de cada servicio, y genera el informe ejecutivo previo al despliegue. El informe incluye el desglose de costos por horizonte de tiempo, la evaluación frente al presupuesto, el nivel de riesgo del escenario, el modelo de pricing recomendado por servicio según el plazo de compromiso indicado, la evaluación del pilar de Optimización de Costos del AWS Well-Architected Framework con impacto económico concreto en USD, una alternativa de menor costo y las buenas prácticas de gestión de costos. La separación entre system prompt fijo y mensaje de usuario variable permite aprovechar el mecanismo de caché de prompts de Amazon Bedrock, reduciendo el costo por consulta en aproximadamente un 85 % respecto a un prompt monolítico.

3.2. Implementación

3.2.1. Configuración del entorno y herramientas utilizadas

El sistema SECC-AWS se desarrolló en dos repositorios independientes: uno para el backend serverless y otro para el frontend web, cada uno con su propio pipeline de CI/CD automatizado mediante GitHub Actions, ejecutado ante cada push a la rama principal de su repositorio (`main` en el backend, `master` en el frontend).

El backend se implementó en Python 3.11, seleccionado por su compatibilidad nativa con el ecosistema de AWS Lambda y con las bibliotecas requeridas para la integración con Amazon Bedrock y el framework Strands Agents. La gestión de dependencias se realizó mediante archivos `requirements.txt` independientes por función Lambda (`estimacion`, `mcp_server` y `reporte`), permitiendo empaquetar únicamente las bibliotecas necesarias para cada componente y reducir el tamaño de los artefactos de despliegue.

La infraestructura se definió mediante AWS Serverless Application Model (SAM), utilizando un archivo `template.yaml` como plantilla declarativa de los recursos AWS requeridos, construida y desplegada mediante `sam build` y `sam deploy` en el pipeline de CI/CD. La comunicación entre el Servicio de Estimación y el MCP Server AWS se controla mediante la variable de entorno `MCP_URL`.

El frontend se desarrolló con React 19 y JavaScript sobre `react-scripts` 5.0.1, con Node.js 18 como entorno de ejecución y la librería `lucide-react` para la iconografía de la interfaz. Las URLs de los endpoints del backend se inyectan en tiempo de construcción mediante las variables de entorno `REACT_APP_ESTIMATE_URL` y `REACT_APP_REPORT_URL`, resueltas como secretos de GitHub Actions.

Ambos pipelines siguen el mismo patrón: *checkout* del código, configuración del entorno de ejecución, configuración de credenciales de AWS y despliegue, garantizando que el entorno de producción refleje siempre el estado más reciente de cada repositorio.

3.2.2. Implementación del frontend

El frontend del sistema SECC-AWS se implementó como una aplicación de página única (SPA) utilizando React 19 con JavaScript y JSX, organizada en dos componentes principales que corresponden a las dos pantallas del flujo de uso: `Formulario.js` y `Informe.js`. El componente raíz `App.js` gestiona el estado global de la aplicación y coordina la transición entre ambas pantallas.

El componente `Formulario.js` maneja tres estados de pantalla. La primera es una pantalla de bienvenida con la descripción del sistema y un enlace al AWS Well-Architected Framework. Al continuar, el usuario selecciona el estilo arquitectónico a evaluar a partir de cinco tarjetas visuales (Monolítica, Microservicios, Serverless, Event-Driven e Híbrida), cada una con su ícono de la librería `lucide-react`. Al seleccionar una tarjeta, la interfaz transiciona al formulario y muestra valores de referencia a modo de *placeholder* en cada campo, orientando al usuario sin completarlos automáticamente. Desde el formulario, el botón ← *Cambiar arquitectura* permite regresar a la selección de estilo sin perder el flujo.

Tanto en las tarjetas de estilo como en los campos del formulario, un componente reutilizable `TooltipInfo` muestra texto de ayuda contextual al pasar el cursor, alimentado por los diccionarios `TOOLTIPS_CONTEXTO` y `TOOLTIPS_ARQUITECTURA`. El botón *Generar evaluación* permanece deshabilitado mientras el usuario no haya completado todos los campos requeridos del contexto de evaluación y de la arquitectura. Al enviarse, la interfaz envía la solicitud al backend y muestra un indicador de progreso giratorio con el mensaje *Evaluación en proceso...* mientras el agente procesa la estimación, proceso que puede tomar entre uno y tres minutos debido a la consulta de precios en tiempo real y a la ejecución del ciclo de razonamiento del agente.

El componente `Informe.js` implementa la pantalla de visualización del informe generado, presentando de forma estructurada la estimación de costos por servicio, el desglose por horizonte de tiempo, la evaluación frente al presupuesto, el nivel de riesgo del escenario, el modelo de pricing recomendado, la región AWS recomendada, las recomendaciones de optimización alineadas con el AWS Well-Architected Framework y las buenas prácticas de gestión de costos. Desde esta pantalla el usuario puede solicitar la descarga del informe en formato PDF o iniciar una nueva evaluación mediante el botón + *Nueva evaluación*, que regresa a la pantalla de inicio.

La comunicación con el backend se realiza mediante peticiones HTTP directas: el endpoint `POST /estimate` se invoca contra una AWS Lambda Function URL, y el endpoint `POST /report` contra Amazon API Gateway, sin requerir servidor de aplicaciones ni capa intermedia. La aplicación se

despliega como sitio web estático en un bucket de Amazon S3 con acceso público habilitado mediante una política de bucket, accesible en http://secc-aws-app.s3-website-us-east-1.amazonaws.com/.

3.2.3. Implementación del backend serverless

El backend del sistema SECC-AWS se estructuró en tres funciones AWS Lambda independientes (Estimación, MCP Server AWS y Reportes), cada una con responsabilidades delimitadas y dependencias propias gestionadas mediante archivos `requirements.txt` independientes. Este patrón permite que cada función escale, se despliegue y se actualice de forma autónoma sin afectar a las demás.

Servicio de Estimación. Actúa como punto de entrada del flujo. Su `handler.py` valida la entrada del usuario, verificando la presencia de los campos requeridos de los bloques `contexto_evaluacion` y `arquitectura`, y retornando un código HTTP 400 con el detalle de los errores en caso de datos inválidos. A partir del campo `estilo_arquitectura`, infiere los parámetros técnicos complementarios mediante el diccionario estático `INFERIDOS_POR_ESCENARIO`, que mapea cada estilo arquitectónico a su conjunto de parámetros (Multi-AZ, backups, auto-scaling, CDN, monitoreo, exposición de API pública, red privada, salida a internet y tipo de aplicación) mediante reglas deterministas sin lógica condicional compleja.

El agente de inteligencia artificial se implementó con el framework Strands Agents (versión 1.40.0), integrado con el modelo `us.anthropic.claude-sonnet-4-6` en Amazon Bedrock. El agente se inicializa en cada invocación con el system prompt fijo y dos herramientas: `get_aws_pricing` del MCP Server AWS y `execute_cost_calculation`, implementada en el propio Servicio de Estimación con el decorador `@tool`. La conexión con el MCP Server se establece mediante `MCPClient` con transporte `streamablehttp_client`, que recupera dinámicamente las herramientas disponibles vía `list_tools_sync()`, permitiendo que el agente las descubra en tiempo de ejecución sin acoplamiento estático. La herramienta `execute_cost_calculation` invoca el Amazon Bedrock AgentCore Code Interpreter mediante el cliente `boto3` de `bedrock-agentcore`: abre una sesión del intérprete, ejecuta el código Python generado por el agente, lee el resultado y cierra la sesión, garantizando el aislamiento entre ejecuciones. La respuesta del agente se extrae mediante una expresión regular que localiza el bloque JSON dentro del texto generado, tolerando texto adicional o prefijos conversacionales que el modelo pueda incluir. El system prompt se separó del mensaje de usuario variable para aprovechar el mecanismo de caché de prompts de Amazon Bedrock descrito en la sección 3.1.16, reduciendo el costo por consulta en aproximadamente un 85 % respecto a un prompt monolítico.

MCP Server AWS. Se implementó como una función Lambda independiente utilizando `awslabs.mcp-lambda-handler`, que implementa el protocolo MCP sobre transporte HTTP compatible con Lambda, eliminando la necesidad de gestionar manualmente JSON-RPC 2.0. La herramienta `get_aws_pricing` se registra mediante el decorador `@mcp.tool()` siguiendo el patrón declarativo de FastMCP. Su `lambda_handler` responde directamente con un chequeo de salud (`status: ok`) ante solicitudes GET u OPTIONS, y delega el resto al manejador MCP.

La consulta de precios se implementó mediante el cliente `boto3` de la AWS Price List API, apoyada en tres estructuras estáticas: `FILTROS_BASE`, con los filtros predefinidos por servicio; `SIN_PRECIO_EN_API`, que identifica los servicios sin precio disponible en la API (donde el agente debe aplicar tarifas oficiales conocidas); y `CAMPOS_DINAMICOS_VALIDOS`, que controla qué parámetros adicionales acepta cada servicio, como `instanceType` para EC2, RDS, ElastiCache y SageMaker. Se implementaron además funciones auxiliares para casos especiales: `ajustar_instancetype_sagemaker` agrega el sufijo `-Hosting` requerido por la API de SageMaker, y `ajustar_usagetype_elasticache` convierte el `instanceType` al formato `NodeUsage:cache.r6g.large` esperado por ElastiCache. Ante la ausencia de resultados o precios en cero, la herramienta retorna `precio_unitario: 0.0` junto con una descripción explicativa, señal que el agente interpreta para aplicar tarifas oficiales conocidas y registrar la limitación en `limitaciones_estimado`.

Comunicación entre componentes. El Servicio de Estimación y el MCP Server AWS se exponen mediante AWS Lambda Function URLs, invocados directamente sin pasar por Amazon API Gateway; el MCP Server es invocado internamente por la función de Estimación a través del protocolo MCP, no por el usuario final. El Servicio de Reportes se expone mediante Amazon API Gateway en el endpoint `POST /report`, con soporte para CORS.

3.2.4. Implementación del generador de informes PDF

El generador de informes PDF se implementó como una función AWS Lambda independiente utilizando la biblioteca `fpdf2` (versión 2.8.2), seleccionada por su compatibilidad con el entorno de ejecución de AWS Lambda y su capacidad de generar documentos completamente en memoria sin escritura en disco.

Se implementó una clase `InformePDF` que extiende `FPDF` con la identidad visual del informe (paleta de colores corporativa, encabezado y pie de página automáticos) y métodos auxiliares reutilizables como `titulo_seccion`, `kv`, `parrafo` y `metrica_box`. Dado que `fpdf2` opera en codificación Latin-1, se implementó la función `limpiar` para convertir caracteres Unicode no compatibles a sus equivalentes ASCII o Latin-1.

El informe se estructura en una portada con el título y el escenario evaluado, seguida de las secciones que reflejan los campos del contrato de respuesta del agente: métricas principales, resumen ejecutivo, top 3 de servicios, tabla de servicios propuestos con el total mensual, evaluación Well-Architected, modelo de pricing, región recomendada con motor de base de datos y referencia de licenciamiento cuando aplica, alternativa de menor costo cuando el sistema la identifica, buenas prácticas de gestión de costos, limitaciones del estimado y un disclaimer final sobre el carácter orientativo de la estimación. El PDF se genera en memoria mediante `pdf.output()`, se codifica en Base64 y se retorna en el cuerpo de la respuesta HTTP (`isBase64Encoded: true`) para descarga directa sin persistencia.

3.2.5. Despliegue en AWS

El despliegue del sistema se automatizó mediante dos pipelines de CI/CD independientes con GitHub Actions, uno por repositorio (backend y frontend), ejecutados ante cada push a la rama

principal de su repositorio (`main` en el backend, `master` en el frontend). La Figura 3.8 resume el flujo de despliegue.

El backend se desplegó en la región `us-east-1` mediante AWS SAM, definido en `template.yaml`, con runtime Python 3.11 y arquitectura `x86_64`. Las funciones de Estimación y MCP Server se exponen mediante Lambda Function URLs, mientras que el Servicio de Reportes se expone mediante Amazon API Gateway con soporte para contenido binario (`application/pdf`). El MCP Server no es invocado por el usuario; es la propia función de Estimación quien lo invoca internamente vía Function URL a través del protocolo MCP. Los permisos IAM se otorgan de forma mínima por función: Estimación accede a Amazon Bedrock, Bedrock AgentCore y la AWS Price List API; el MCP Server únicamente a la AWS Price List API.

La URL del MCP Server se inyecta automáticamente en la función de Estimación mediante la variable de entorno `MCP_URL`, resuelta en tiempo de despliegue, eliminando la necesidad de configuración manual entre funciones. El frontend se despliega como sitio estático en un bucket de Amazon S3 mediante `aws s3 sync`, tras el proceso de construcción con `npm run build`.

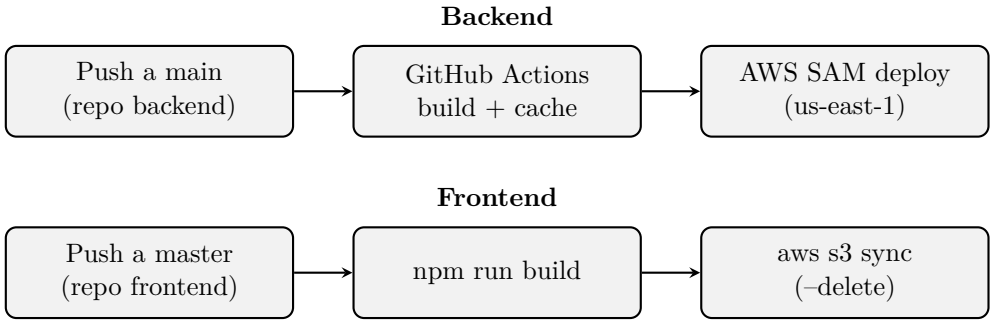


Figura 3.8: Flujo de despliegue automatizado mediante GitHub Actions

3.3. Resumen del capítulo

Este capítulo describió el diseño e implementación del prototipo SECC-AWS. La sección de diseño presentó los cinco escenarios representativos de carga de trabajo en AWS, sus parámetros de entrada e inferidos, los requisitos funcionales, el modelo de evaluación por capas, los módulos del sistema, los contratos de datos, el esquema de interacción con el agente, la arquitectura de solución y el diseño del prompt.

La sección de implementación describió las decisiones técnicas de cada componente: la configuración del entorno (Python 3.11 y React 19) con pipelines CI/CD automatizados; el frontend como aplicación de página única en Amazon S3; el backend serverless con sus tres funciones Lambda; el agente Strands integrado con Bedrock, el MCP Server AWS y el AgentCore Code Interpreter; el generador de informes PDF; y el despliegue automatizado en AWS mediante SAM.

El siguiente capítulo presenta la estrategia de evaluación y los resultados de la validación comparativa del sistema frente al experto de referencia.

Capítulo 4

Evaluación

4.1. Diseño de la evaluación

La evaluación del prototipo SECC-AWS tuvo como objetivo analizar su desempeño en la estimación de costos y en la generación de recomendaciones de optimización en entornos cloud. A diferencia de un protocolo de evaluación con escenarios predefinidos, se adoptó un enfoque de libre elección en el que cada evaluador selecciona el escenario que considera más representativo según su experiencia, con el fin de obtener una validación más cercana a los casos de uso reales.

Para estructurar la evaluación se definieron cuatro métricas:

Precisión de recomendaciones (%): porcentaje de servicios AWS propuestos por el sistema que el evaluador considera correctos y adecuados para el escenario evaluado.

Consistencia en estimaciones (%): qué tan cerca están los costos estimados por el sistema de lo que el evaluador calcularía para el mismo escenario, donde 100 % indica coincidencia exacta.

Tiempo de análisis (segundos): tiempo medido desde que se envía la solicitud hasta que aparece el informe completo en pantalla.

Utilidad percibida (1–5): valoración general del evaluador sobre qué tan útil sería el informe generado para un equipo de arquitectura, en una escala tipo Likert donde 1 es no útil y 5 es muy útil.

Cada evaluador recibió acceso al prototipo desplegado en `http://secc-aws-app.s3-website-us-east-1.amazonaws.com/`, un documento de instrucciones con la definición de las métricas y una tabla de evaluación para registrar sus observaciones.

4.2. Resultados de la evaluación

En esta sección se presentan los resultados obtenidos a partir de la aplicación del protocolo de evaluación. El análisis se realizó sobre los escenarios seleccionados libremente por cada evaluador, comparando los resultados generados por el sistema con el criterio experto. Las evaluaciones se presentan en orden cronológico según la fecha en que fueron realizadas.

4.2.1. Evaluación 1 — Escenario Event-Driven: Pipeline de facturación electrónica

Evaluadora: Juliana Peña
Años de experiencia en AWS: 6

Certificaciones: AWS Solutions Architect Associate
Fecha: 1 de junio de 2026
Escenario evaluado: Pipeline de facturación electrónica sobre arquitectura event-driven, con ingesta de documentos en S3, procesamiento mediante EventBridge y Lambda, almacenamiento en RDS MySQL Multi-AZ y DynamoDB, infraestructura de red con VPC y NAT Gateways Multi-AZ, horizonte mensual y presupuesto de 500 USD.

Tabla 4.1: Resultados de evaluación — Escenario Event-Driven

Métrica	**Valor**	**Observación**
Precisión de recomendaciones (%)	88	12 servicios bien seleccionados para el pipeline; algunos podrían complementarse.
Consistencia en estimaciones (%)	82	Cálculos de RDS, NAT y Lambda bien documentados; simplificaciones menores en transferencia de datos dentro de rangos adecuados.
Tiempo de análisis (segundos)	120	Tiempo adecuado para la complejidad del escenario.
Utilidad percibida (1–5)	5	Informe preciso, completo y listo para orientar decisiones de arquitectura.

Servicios correctamente identificados: Amazon S3 para ingesta, Amazon EventBridge, AWS Lambda, Amazon SQS, Amazon RDS MySQL Multi-AZ, Amazon DynamoDB, Amazon VPC con dos NAT Gateways Multi-AZ, Amazon CloudWatch, AWS CloudTrail y Amazon SNS para alertas operativas.
Observaciones sobre servicios: La evaluadora identificó cuatro aspectos de mejora. Primero, la Dead Letter Queue se menciona en la sección Well-Architected como recomendación pero no se incluye en los servicios propuestos ni en el costo, siendo un componente obligatorio para un pipeline de facturación electrónica donde no se puede perder ningún documento. Segundo, AWS Step Functions Express sería ideal para orquestación del pipeline con manejo nativo de errores y reintentos, ofreciendo mayor visibilidad del flujo completo respecto a la combinación EventBridge y Lambda. Tercero, el pipeline no incluye un canal de entrega de notificaciones al receptor final de la factura, ya que SNS solo cubre alertas operativas internas. Cuarto, el costo de NAT Gateway asume 450 GB de tráfico procesado, pero la implementación de VPC Endpoints para S3 y DynamoDB reduciría drásticamente este volumen; el costo base debería reflejar la arquitectura óptima.
Precisión de costos: Los cálculos de los componentes de alto costo son precisos: RDS, NAT Gateway y Lambda están correctamente calculados. Las discrepancias menores se concentran en servicios de bajo costo como DynamoDB, S3 y SQS con un impacto total inferior a 5 USD mensuales.
Observación general: La evaluadora destacó como fortalezas la transparencia presupuestaria al declarar explícitamente que el escenario excede el presupuesto en un 150.3 % sin ocultarlo, la

propuesta de una alternativa viable con db.t3.medium con advertencias sobre créditos de CPU, la inclusión completa de infraestructura de red, la precisión en los cálculos de Lambda con la fórmula correcta de invocaciones por duración por memoria, el nivel de detalle en las limitaciones del estimado y la referencia de licenciamiento comparativa. Como área de mejora señaló que el nivel de riesgo debería clasificarse como Alto y no Medio dado que el costo excede el presupuesto en un 150 %, situación que representa un riesgo financiero significativo.

4.2.2. Evaluación 2 — Escenario de Microservicios con carga media

Evaluador: Oscar Alberto Arboleda López
Institución: Suramericana
Experiencia en AWS: 1 año
Certificaciones: AWS Certified Solutions Architect Associate (en progreso), Oracle Cloud Infrastructure Certified Architect Associate, Microsoft Certified Azure Fundamentals
Fecha: 09 de junio de 2026
Escenario evaluado: Plataforma SaaS de gestión empresarial con múltiples módulos independientes, arquitectura de microservicios, 5.000 usuarios concurrentes, base de datos mixta, cumplimiento GDPR, horizonte trimestral, presupuesto de 5.000 USD.

Tabla 4.2: Resultados de evaluación — Escenario Microservicios

Métrica	SECC-AWS	Experto	Observación
Precisión de recomendaciones (%)	90	95	El sistema es sólido pero no detecta ajustes finos a nivel de servicios.
Consistencia en estimaciones (%)	80	95	El contexto y la experiencia aportan estimaciones más realistas.
Tiempo de análisis (segundos)	206	900	El sistema reduce considerablemente el tiempo respecto a la calculadora de AWS.
Utilidad percibida (1–5)	4	5	Existen servicios duplicados que no son necesarios para la solución propuesta.

Servicios correctamente identificados: ElastiCache, Fargate, DynamoDB, S3, EBS, CloudFront, ELB, API Gateway, WAF, CloudWatch, Backup, ECR, KMS, Route53 y Cognito, correctamente dimensionados con observaciones de uso adecuadas para el escenario.
Observaciones sobre servicios: El evaluador identificó cinco aspectos de mejora. Primero, EKS no es necesario ya que el patrón de despliegue con contenedores puede implementarse enteramente con AWS Fargate. Segundo, el volumen de SQS estimado de 500 millones de mensajes mensuales es

excesivo para el nivel de intercambio de mensajes del escenario. Tercero, la asignación de 20 tareas Fargate con CPU y memoria bajas resulta insuficiente para un sistema de gestión empresarial de esta escala. Cuarto, la propuesta de RDS con MySQL y PostgreSQL simultáneos es redundante; un único motor, preferiblemente PostgreSQL, es suficiente. Quinto, SageMaker en modo de inferencia en tiempo real es una solución de alto costo y muy específica; para este caso un modelo asíncrono o bajo demanda es más adecuado.

Precisión de costos: El evaluador considera que los costos se acercan bastante a un presupuesto real, aunque ajustes en los servicios innecesarios permitirían acercarse más a las premisas del negocio.

Observación general: El evaluador destacó que el informe generado por SECC-AWS presenta un resumen de métricas principales al inicio que facilita la toma de decisiones, un top 3 de servicios de mayor costo que orienta la optimización, y un detalle por servicio que justifica su inclusión en la arquitectura. Como sugerencia de mejora, señaló que el formulario de captura de datos es muy plano y limita los escenarios que se pueden plantear, y que dado el uso de inteligencia artificial el sistema sería capaz de interpretar escenarios descritos de forma libre sin necesidad de un formulario predeterminado.

4.2.3. Evaluación 3 — Escenario de Microservicios con patrón SAGA

Evaluadora: Liz Bareño
Años de experiencia en AWS: 3
Certificaciones: AWS Cloud Practitioner, AWS Solutions Architect Associate
Fecha: 10 de junio de 2026
Escenario evaluado: Arquitectura de microservicios con patrón SAGA coreográfico implementado sobre funciones serverless, incluyendo funciones compensadoras, colas de mensajes y trazabilidad distribuida.

Tabla 4.3: Resultados de evaluación — Escenario Microservicios SAGA

Métrica	**Valor**	**Observación**
Arquitectura evaluada	Microservicios con patrón SAGA (Serverless)	
Precisión de recomendaciones (%)	90	Servicios clave correctamente identificados y justificados.
Consistencia en estimaciones (%)	85	Inconsistencia menor en API Gateway: el informe indica 45M solicitudes pero el costo calculado corresponde a 36M.
Tiempo de análisis (segundos)	90	Tiempo significativamente inferior al análisis manual.
Utilidad percibida (1–5)	4	Informe completo y bien estructurado con oportunidades de mejora puntuales.

Servicios correctamente identificados: AWS Lambda con 10 funciones (5 microservicios y 5 compensadores), patrón correcto para SAGA coreográfico con 512 MB y 200ms promedio; Amazon VPC con subnets privadas y NAT Gateways Multi-AZ; AWS X-Ray, AWS WAF, Amazon CloudWatch, Amazon CloudFront y AWS Secrets Manager.

Observaciones sobre servicios: La evaluadora identificó dos oportunidades de mejora. Primero, el informe optó por coreografía con SQS para el patrón SAGA, que es válido, pero debería mencionar AWS Step Functions modo Express como alternativa para orquestación con manejo de estado nativo. Segundo, dado el SLA de 2000ms y las 10 funciones Lambda con posibles cold starts, Provisioned Concurrency debería presentarse como recomendación activa para las funciones críticas del SAGA y no solo como nota al margen.

Precisión de costos: Los costos se consideran razonables en orden de magnitud, con inconsistencias menores en API Gateway y posibles subestimaciones en Lambda y X-Ray. Las limitaciones del estimado se presentan de forma transparente con las fórmulas utilizadas en cada cálculo.

Observación general: La evaluadora destacó como fortalezas la completitud en la topología de red, la inclusión de componentes de seguridad, la correcta comprensión del patrón SAGA con identificación de compensadores y DLQ, la cuantificación de una alternativa de optimización con ahorro concreto de $87.85 mensuales mediante HTTP API y VPC Endpoints, y el nivel de detalle en las limitaciones del estimado con justificación explícita de cada cálculo manual.

4.2.4. Evaluación 4 — Escenario de Arquitectura Monolítica

Evaluador: Jose Alejandro Benítez Aragón
Institución: Pontificia Universidad Javeriana Cali
Años de experiencia en AWS: 5
Certificaciones: AWS Solutions Architect Professional, AWS DevOps Professional
Fecha: 17 de junio de 2026

Escenario evaluado: Portal ciudadano para descarga de documentos, gestión de quejas y consulta de noticias, implementado sobre arquitectura monolítica con 200 usuarios concurrentes, base de datos relacional MySQL, 200 GB de almacenamiento y 600 GB de transferencia mensual, con horizonte anual y presupuesto de 5.000 USD.

Tabla 4.4: Resultados de evaluación — Escenario Arquitectura Monolítica

Métrica	**Valor**	**Observación**
Precisión de recomendaciones (%)	85	10 de 11 servicios correctos y bien justificados para el escenario.
Consistencia en estimaciones (%)	80	Precios unitarios coherentes con AWS Pricing; inconsistencias menores en transferencia S3 y LCU del ALB.
Tiempo de análisis (segundos)	100	Razonable para una consulta con IA generativa y AWS Price List API.
Utilidad percibida (1–5)	4	Informe profesional y bien estructurado con mínimas observaciones.

Servicios correctamente identificados: Amazon EC2 con instancia Reserved a un año, correctamente justificado para carga 24/7 con 200 usuarios concurrentes; Amazon RDS MySQL con instancia db.m5.large adecuada para la carga descrita; Amazon S3 para almacenamiento de documentos PDF; Application Load Balancer para exposición del portal con HTTPS; EBS, CloudWatch, AWS Backup y Route53.

Observaciones sobre servicios: El evaluador identificó cuatro aspectos de mejora. Primero, Amazon CloudFront debería sugerirse más enfáticamente para un portal con 600 GB de transferencia mensual dirigido a Latinoamérica desde us-east-1, dado que reduciría latencia y costos de transferencia. Segundo, AWS WAF debería al menos mencionarse para un portal ciudadano expuesto a internet. Tercero, el informe identifica el riesgo de Single-AZ pero no cuantifica el costo de migrar a Multi-AZ, que duplicaría el costo de RDS. Cuarto, NAT Gateway no aparece en el informe a pesar de ser necesario si EC2 y RDS están en subnets privadas, lo que elevaría el costo mensual de $308.49 a aproximadamente $345.84, reduciendo el margen presupuestal de $1.298 a $860 anuales.

Precisión de costos: EC2 y RDS son consistentes con la AWS Pricing. Se identificó una sobreestimación de aproximadamente $9 en transferencia S3, dado que el informe cobra los 600 GB completos sin aplicar los primeros 100 GB gratuitos del free tier. Los LCU del ALB se calcularon en 5 unidades por hora, valor que el evaluador considera elevado para el volumen descrito.

Observación general: El evaluador destacó como fortalezas la estructura ejecutiva del informe, las justificaciones de pricing por servicio, la inclusión de recomendaciones Well-Architected con ahorro cuantificado, la propuesta de etiquetado y gobernanza con AWS Budgets y Cost Explorer, la transparencia en las limitaciones del estimado y la referencia de licenciamiento comparativa. Concluyó que el informe es de buena calidad para un prototipo y proporciona una estimación razonable con justificaciones sólidas para un equipo de arquitectura que necesite una primera estimación de costos.

4.3. Resumen del capítulo

Este capítulo presentó el diseño de la evaluación del prototipo SECC-AWS y los resultados obtenidos a partir de la validación con cuatro expertos en arquitectura cloud. Se definieron cuatro métricas de evaluación: precisión de recomendaciones, consistencia en estimaciones, tiempo de análisis y utilidad percibida, aplicadas sobre escenarios seleccionados libremente por cada evaluador.

La Tabla 4.5 presenta el consolidado de resultados obtenidos en las cuatro evaluaciones.

Tabla 4.5: Consolidado de resultados de evaluación

Evaluador	**Escenario**	**Precisión**	**Consistencia**	**Tiempo**	**Utilidad**
Juliana Peña	Event-Driven facturación	88 %	82 %	120s	5/5
Oscar Arboleda	Microservicios carga media	90 %	80 %	206s	4/5
Liz Bareño	Microservicios patrón SAGA	90 %	85 %	90s	4/5
Alejandro Benítez	Monolítica portal ciudadano	85 %	80 %	100s	4/5
Promedio		**88.25 %**	**81.75 %**	**Aprox. 129s**	**4.25/5**

Los resultados obtenidos muestran un comportamiento consistente del sistema a través de los cuatro escenarios evaluados. La precisión de recomendaciones se mantuvo entre el 85 % y el 90 %, con un promedio de 88.25 %, lo que indica que el agente identifica correctamente la mayoría de los servicios AWS necesarios para cada escenario. La consistencia en estimaciones osciló entre el 80 % y el 85 %, con un promedio de 81.75 %, reflejando que los costos generados son razonables en orden de magnitud aunque presentan inconsistencias menores en servicios de bajo costo o en la aplicación del free tier. El tiempo de análisis se mantuvo por debajo de los 3 minutos con 30 segundos en todos los casos, significativamente inferior al tiempo requerido por un experto para realizar la misma tarea manualmente. La utilidad percibida promedió 4.25 sobre 5, con una evaluación de 5 sobre 5 en el escenario event-driven, evidenciando que el informe generado es considerado útil o muy útil por los evaluadores para orientar decisiones de arquitectura en etapas tempranas de un proyecto.

Las observaciones comunes entre los evaluadores apuntan a tres oportunidades de mejora: mayor cobertura de patrones arquitectónicos avanzados como Step Functions para orquestación y Provisioned Concurrency para funciones críticas, aplicación correcta del free tier en transferencia de datos, y enriquecimiento de la interfaz de captura para permitir escenarios más complejos descritos en lenguaje natural. El siguiente capítulo presenta las conclusiones del proyecto y las lecciones aprendidas del proceso de desarrollo.

Capítulo 5

Conclusiones

5.1. Conclusiones

El presente proyecto tuvo como objetivo general desarrollar un prototipo que permitiera evaluar y proyectar costos de implementación en AWS a partir de escenarios de arquitectura definidos por el usuario, apoyado en un agente de inteligencia artificial. A continuación se presentan las conclusiones correspondientes a cada uno de los cuatro objetivos específicos planteados.

1. **Análisis de escenarios representativos.** La clasificación de parámetros en explícitos e inferidos demostró ser un mecanismo efectivo para reducir la carga de entrada del usuario sin sacrificar la representatividad técnica de cada escenario. Esto confirma que es posible delimitar la complejidad de las arquitecturas cloud a un conjunto manejable de variables sin perder validez frente a los cinco estilos arquitectónicos más comunes en sistemas modernos.

2. **Diseño de la arquitectura y el modelo funcional.** El modelo de cinco capas evidenció que separar la orquestación del agente de la lógica de negocio del sistema permite delegar la complejidad del razonamiento en el framework Strands Agents sin comprometer la trazabilidad de los datos. Esto sugiere que, en sistemas que integran inteligencia artificial con fuentes de datos verificables como la AWS Price List API, un diseño por capas resulta más mantenible que una orquestación manual acoplada al código de negocio.

3. **Desarrollo del prototipo funcional.** La integración exitosa entre el agente Strands, el MCP Server AWS y el AgentCore Code Interpreter confirma que es viable combinar razonamiento de lenguaje natural con cálculos deterministas ejecutados en código, eliminando el riesgo de alucinaciones numéricas sin renunciar a la flexibilidad interpretativa del modelo de lenguaje frente a escenarios heterogéneos.

4. **Estrategia de validación comparativa.** Los resultados obtenidos en la validación con expertos (precisión promedio de 88.25 %, consistencia promedio de 81.75 % y utilidad percibida promedio de 4.25 sobre 5) permiten concluir que el sistema genera estimaciones técnicamente razonables y útiles como apoyo a decisiones tempranas de arquitectura, aunque persisten brechas puntuales en la inclusión de servicios complementarios de seguridad y resiliencia que un arquitecto humano asumiría como implícitos.

En conjunto, estos hallazgos permiten concluir que el objetivo general del proyecto se cumplió: SECC-AWS demuestra que es posible construir un sistema serverless que combine razonamiento de

inteligencia artificial con datos de precios oficiales y cálculos verificables para generar estimaciones de costos en AWS significativamente más rápidas que el proceso manual, sin sacrificar la calidad técnica de las recomendaciones generadas.

5.2. Limitaciones del estudio

Durante la implementación del prototipo se identificaron limitaciones técnicas que deben considerarse al interpretar los resultados de este estudio. La AWS Price List API no retorna precios para un conjunto de servicios entre ellos Amazon EKS, AWS Fargate, AWS Lambda, Amazon CloudFront y Amazon Cognito, por lo que el agente recurre a tarifas oficiales conocidas para estos casos y lo documenta explícitamente en el campo de limitaciones de cada informe, lo que introduce una dependencia del conocimiento del modelo para mantener actualizados estos valores ante cambios futuros de precios. De manera más general, todas las recomendaciones generadas dependen del conocimiento incorporado en el modelo de lenguaje, que puede no reflejar servicios o configuraciones de AWS posteriores a su entrenamiento. Finalmente, la validación se realizó con cuatro expertos evaluando un único escenario cada uno, lo que no permite establecer conclusiones estadísticamente generalizables sobre el desempeño del sistema frente a la totalidad de combinaciones posibles de parámetros de arquitectura, ni medir formalmente la consistencia de las estimaciones del agente ante ejecuciones repetidas con los mismos parámetros de entrada.

5.3. Trabajos futuros

A partir de los hallazgos y limitaciones identificadas, se proponen las siguientes líneas de trabajo futuro:

- **Extensión multi-cloud:** aprovechar el diseño extensible de la capa de consulta de precios y cálculo mediante el protocolo Model Context Protocol para incorporar proveedores adicionales como Microsoft Azure y Google Cloud Platform, permitiendo comparaciones de costos entre proveedores a partir de un mismo escenario de arquitectura.
- **Ampliación de la validación experimental:** extender el protocolo de evaluación a un mayor número de expertos y de ejecuciones repetidas por escenario, permitiendo medir formalmente la consistencia del sistema ante variaciones en la generación del agente y obtener resultados con mayor robustez estadística.
- **Incorporación de persistencia y trazabilidad:** habilitar el almacenamiento opcional de estimaciones generadas, permitiendo comparar la evolución de costos en el tiempo y dar seguimiento histórico a las decisiones arquitectónicas evaluadas por un mismo usuario.
- **Validación con escenarios reales en producción:** explorar la comparación entre las estimaciones del sistema y los costos reales facturados por AWS para arquitecturas efectivamente desplegadas, fortaleciendo la validez externa de las proyecciones generadas.

5.4. Lecciones aprendidas

El desarrollo del proyecto dejó aprendizajes relevantes tanto a nivel técnico como metodológico:

1. **Diseño de interfaz.** El bocetado manual previo a la implementación permitió validar conceptualmente el flujo de navegación del sistema, aunque el diseño final evolucionó respecto a los bocetos iniciales: la estructura de dos columnas se simplificó en un flujo lineal y la doble vista del informe se unificó en un único informe ejecutivo. Esta evolución confirma que el bocetado temprano es valioso como punto de partida, pero el diseño de interfaz se beneficia de iterarse durante el desarrollo a medida que surgen restricciones no anticipadas en la fase de boceto.

2. **Captura de parámetros de arquitectura.** El boceto inicial contemplaba que el usuario describiera su sistema mediante texto libre, dejando que el agente interpretara la arquitectura completa a partir del lenguaje natural. Durante la implementación se optó por un formulario estructurado con campos específicos por estilo arquitectónico, acotando el dominio de entrada del agente a un conjunto de parámetros predecibles. Esta decisión privilegia un uso de inteligencia artificial estrecha, orientada a una tarea concreta y delimitada, sobre un enfoque de interpretación abierta, reduciendo la variabilidad de las respuestas del agente y aumentando la confiabilidad de las estimaciones generadas.

3. **Orquestación de inteligencia artificial.** Delegar el ciclo completo de razonamiento al framework Strands Agents mediante Tool Use nativo demostró ser más robusto que implementar lógica de orquestación manual en el código del sistema, permitiendo que el agente decida autónomamente cuándo consultar precios y cuándo ejecutar cálculos sin intervención del desarrollador en cada paso del flujo. La separación entre el system prompt fijo y el user prompt variable resultó clave por dos razones: permitió aprovechar el mecanismo de caché de prompts de Amazon Bedrock para reducir el costo por consulta, y mantuvo las reglas de negocio del agente siempre en el mismo bloque de instrucciones, evitando que el comportamiento del sistema varíe según cómo el usuario redacte los datos de su escenario.

4. **Retroalimentación de expertos.** La validación con los cuatro evaluadores reveló un patrón consistente entre escenarios: el agente identifica con alta precisión los servicios centrales de cada arquitectura, pero tiende a omitir servicios de soporte que un arquitecto humano asumiría como implícitos. En los escenarios de microservicios se señaló la falta de alternativas de orquestación como AWS Step Functions y de mecanismos de resiliencia como Provisioned Concurrency para funciones críticas. En el escenario monolítico se identificó la ausencia de componentes de seguridad perimetral como AWS WAF y de infraestructura de red como NAT Gateway. En el escenario event-driven se observó que una Dead Letter Queue, recomendada por el propio informe como buena práctica, no se incluyó como servicio presupuestado, y que el nivel de riesgo reportado no reflejó la magnitud real de un sobrecosto del 150 % sobre el presupuesto. Este patrón sugiere que las reglas de negocio del agente cubren adecuadamente la selección del servicio principal por estilo arquitectónico, pero podrían reforzarse para

garantizar la inclusión sistemática de servicios complementarios de seguridad, resiliencia y recuperación ante fallos. Esta retroalimentación confirma que la validación con expertos es una práctica indispensable para identificar brechas que no resultan evidentes durante el desarrollo interno del prototipo.

Bibliografía

Amazon Web Services (2024). Cost Optimization Pillar - AWS Well-Architected Framework.

Amazon Web Services (2025a). Amazon Bedrock AgentCore: Code Interpreter. Accessed: 2026.

Amazon Web Services (2025b). Strands Agents: An Open Source AI Agents SDK. Accessed: 2026.

Anthropic (2024). Model Context Protocol: Introduction. Accessed: 2026.

Aoshima, T. and Yoshida, K. (2020). Pre-Design Stage Cost Estimation for Cloud Services. In *Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 61–66. IEEE.

Aoshima, T. and Yoshida, K. (2022). Pre-Design Stage Cost Estimation for Cloud Services. *IEEJ Transactions on Electronics, Information and Systems*, 142(6):679–688.

Bhaskaran, R., Muntean, C. H., and Gupta, S. (2025). AI-Driven Cloud Optimization: Enhancing Cost Prediction, Resource Scheduling and Fault Resilience in Cloud Environments. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., and others (2021). On the Opportunities and Risks of Foundation Models. *arXiv preprint arXiv:2108.07258*.

Dua, R. and Aggarwal, M. (2019). *AWS Cloud Architecture Patterns*. Packt Publishing.

Filiopoulou, E., Mitropoulou, P., Tsadimas, A., Michalakelis, C., Nikolaidou, M., and Anagnostopoulos, D. (2015). Integrating Cost Analysis in the Cloud: A SoS Approach. In *Proceedings of the 11th International Conference on Innovations in Information Technology (IIT)*, pages 278–283. IEEE.

Gartner Inc. (2024). Gartner Forecasts Worldwide Public Cloud End-User Spending to Total 723 Billion in 2025.

Gartner Inc. (2025). Accelerate your public cloud migration the right way.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.

He, H., Ma, Z., Li, X., Chen, H., and Shao, W. (2012). An Approach to Estimating Cost of Running Cloud Applications Based on AWS. In *Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC)*, pages 571–576. IEEE.

Jakkaraju, V. T. D. and Jayaprakasan, V. (2026). Predictive Green FinOps: Joint Optimization of Cost, Carbon, and Reliability in AI-Intensive Clouds. In *Proceedings of the IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*. IEEE.

Johnsen, E. B., Pun, K. I., and Tapia Tarifa, S. L. (2017). A Formal Model of Cloud-Deployed Software and Its Application to Workflow Processing. In *2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE.

Martin, R., Preciado, J. C., Rodriguez-Echeverria, R., Conejero, J. M., and Sanchez-Figueroa, F. (2017). A First Step to Cloud Infrastructure Cost Estimation in Early Stages of Web Development. In *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 437–443. SciTePress.

Mell, P. and Grance, T. (2011). The NIST definition of cloud computing (Special Publication 800-145). National Institute of Standards and Technology. Technical report, National Institute of Standards and Technology, Gaithersburg, MD.

Preciado, J. C., Rodriguez-Echeverria, R., Conejero, J. M., Sanchez-Figueroa, F., and Prieto, A. (2018). An Approach for Guesstimating the Deployment Cost in Cloud Infrastructures at Design Phase in Web Engineering. *Journal of Web Engineering*, 17(3-4):224–240.

Provost, F. and Fawcett, T. (2013). *Data Science for Business*. O'Reilly Media, Sebastopol, CA.

Raouf, A., Ait Said, M., Ezzati, A., Belouaddane, L., and Hassoun, M. (2026). Toward FinOps-Aware Software Design: Predicting Cloud Costs from Dataflow-Based Resource Modeling. In *Lecture Notes in Networks and Systems*, volume 1781, pages 513–522. Springer.

Ricci, F., Rokach, L., and Shapira, B. (2015). *Recommender Systems Handbook*. Springer, 2 edition.

Russell, S. J. and Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson, Hoboken, NJ, 4 edition.

Storment and Fuller (2020). *Cloud FinOps Collaborative, Real-Time Cloud Financial Management*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.

Weinman, J. (2012). *Cloudonomics: The Business Value of Cloud Computing*. Wiley, Hoboken, NJ.

Wittig, M. and Wittig, A. (2023). *Amazon Web Services in Action*. Manning Publications, 3rd edition.