Implementación de una Arquitectura Cloud en AWS Orientada a Microservicios para un Cliente de la Empresa ACCENTURE

Sebastián Aguirre Yacup

Trabajo de Grado para optar al título de Ingeniero Electrónico

Directora académica:
Mg. Yuli Sidney Garcés

CORPORACIÓN UNIVERSITARIA AUTÓNOMA DEL CAUCA
FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA
PASANTÍA

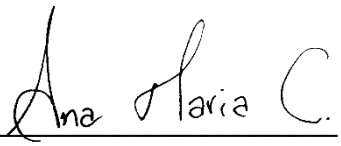
2023

NOTA DE ACEPTACION

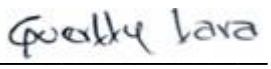
Aprobado por el comité de grado en cumplimiento de los requisitos exigidos por la Corporación Universitaria Autónoma del Cauca para optar el titulo del Ingeniero Electrónico.

Yuli Sidney Garcés Bolaños

Directora



Ana Maria Caviedes
Jurado



Jose Guerlly Lara

Jurado

Tabla de Contenido

Lista de Figuras

Lista de Tablas

RESUMEN

Uno de los temas más importantes respecto al desarrollo y uso de nuevas tecnologías es su integración con los diferentes sectores de la sociedad, incluyendo el sistema bancario como uno de los principales para el funcionamiento en el sistema económico de cada país. En este contexto, este proyecto proporciona a las entidades bancarias una solución tecnológica moderna y confiable, impulsando su transformación digital y permitiéndoles afrontar los desafíos del sector de manera eficiente y segura. Mediante la implementación de una arquitectura cloud en AWS, orientada a microservicios tipo serverless, se logra optimizar los procesos y mejorar la eficiencia en el ámbito bancario. Esta arquitectura permite gestionar las operaciones de los métodos de un CRUD y del método GET de forma ágil y segura, optimizando el tiempo de los clientes y usuarios del banco al momento de realizar consultas sobre el estado de su cuenta bancaria, así como permitiendo al asesor modificar dichos datos. Durante el desarrollo del proyecto, se han utilizado servicios clave de AWS, como AWS Lambda, Amazon API Gateway y Amazon DynamoDB, para construir una plataforma robusta y altamente disponible. Además, se ha puesto especial énfasis en la seguridad y protección de los datos sensibles, aplicando las mejores prácticas y controles adecuados. Para evaluar el funcionamiento de la arquitectura, se han realizado pruebas exhaustivas con Postman utilizando el paquete Newman, obteniendo resultados óptimos en términos de tiempo de respuesta y porcentaje de éxito en las consultas.

PALABRAS CLAVE: banco, arquitectura, serverless, microservicios , CRUD, GET, métodos, AWS, SCRUM, Postman, Servicios, AWS Lambdas, AWS API GATEWAY, AWS DynamoDB, Microsoft Azure Devops , seguridad, desarrollo de software

ABSTRACT

One of the most important topics regarding the development and use of new technologies is their integration with different sectors of society, including the banking system as one of the main components of each country's economic system. In this context, this project provides modern and reliable technological solutions to banking institutions, driving their digital transformation and enabling them to efficiently and securely face the challenges of the sector. By implementing a cloud architecture on AWS, focused on serverless microservices, the project achieves process optimization and improved efficiency in the banking domain. This architecture allows for agile and secure management of CRUD methods and the GET method, optimizing the time for clients and bank users when querying the status of their bank accounts, as well as enabling advisors to modify such data. Throughout the project development, key AWS services such as AWS Lambda, Amazon API Gateway, and Amazon DynamoDB have been utilized to build a robust and highly available platform. Special emphasis has been placed on security and protection of sensitive data, following best practices and implementing appropriate controls. Exhaustive testing using Postman and the Newman package has been conducted to evaluate the architecture's performance, resulting in optimal response times and query success rates.

KEYWORDS: bank, architecture, serverless, microservices, CRUD, GET, methods, AWS, SCRUM, Postman, services, AWS Lambdas, AWS API Gateway, AWS DynamoDB, Microsoft Azure DevOps, security, software development.

INTRODUCCIÓN

Este informe da cuenta de la ejecución de la pasantía para optar por el título profesional en ingeniería electrónica. El trabajo se desarrolló en la empresa ACCENTURE y se ocupó de construir una arquitectura Cloud en AWS orientada a microservicios tipo serveless, es decir sin servidor; proceso que incluyó el diseño, implementación y pruebas, teniendo en cuenta los requerimientos del cliente.

La implementación de la arquitectura se hizo en AWS, una plataforma líder debido a las múltiples ventajas y servicios que ofrece en la nube. Se usó la metodología XP programación extrema bajo el marco de trabajo Scrum, y se realizó desde el 22 de septiembre de 2022 hasta el 10 de enero de 2023.

La organización en la que se ejecutó el desarrollo es ACCENTURE, una empresa que se dedica a brindar soluciones a todo tipo de organizaciones que requieren procesos de migración a la nube o que quieren optimizar sus tareas con estas metodologías. Para eso, cuenta con un equipo multidisciplinar y políticas internas que ayudan a optimizar los flujos de trabajo. A esto se suma su amplia experiencia en el mundo de la tecnología.

Lo anterior la convierte en un escenario ideal para realizar prácticas profesionales, ya que proporciona entrenamiento y contacto con desafíos reales para la ingeniería electrónica. En este sentido, llevar a cabo la pasantía allí permitió aplicar conocimientos adquiridos durante la formación académica, tener un acercamiento al campo laboral, sus necesidades, tendencias y formas de trabajo; lo que aumenta el nivel de competitividad del profesional.

I. MOTIVACIÓN

A. Planteamiento del Problema

En el mundo como en Colombia se usan las tecnologías de información y la telemática para optimizar procesos en las empresas públicas y/o privadas, generar ofertas de productos y servicios e incrementar la productividad de las compañías [1]. Con la pandemia de COVID-19 hubo una desaceleración de las economías mundiales, entre los países desarrollados y las cadenas de suministros, pero se incrementó el trabajo en casa y de forma digital. Esto llevó a que en el país el Ministerio del Trabajo expidiera la Circular 018 del 10 de marzo de 2020 para aplicar en los ambientes laborales, en entidades del sector público y privado [2], y la Ley 2088 de 2021, por la cual se regula el Trabajo en Casa en Colombia con el fin de crear elementos jurídicos para proteger el empleo en el marco de situaciones ocasionales como la generada por la pandemia del Covid-19" [3].

Ahora bien, generalmente las empresas tienen problemas operacionales dentro de sus fábricas, sobre todo, no cuentan con plataformas (Cloud o servidores remotos conectados a internet), para poder alojar todos sus datos y lograr los mejores resultados de interacción entre cliente, usuario, empresa y problema o incidente del usuario. En este sentido, la multinacional ACCENTURE en el departamento de Tecnology, en el área de Desarrollado se resuelve esa problemática personalizando las soluciones a los desafíos de otras empresas. Durante la pasantía se tuvo la oportunidad de diseñar una Arquitectura Cloud en AWS [4], orientada a microservicios [5] para usar lenguajes de programación con los cuales se desarrollaron módulos de procesamiento, denominados contenedores [6], que facilitaron a los usuarios el acceso a datos dinámicos. Las empresas que usan estas tecnologías reducen costo y debido a la elasticidad de la plataforma pueden gestionar niveles pico de actividad a futuro y aprovisionar los recursos necesarios, ampliando la capacidad de usuarios o disminuyéndola según la demanda.

Es así como migrando a la nube la problemática que se resolvió para el cliente de ACCENTURE fue el uso de almacenamiento físico, el cual tiene desventajas como seguridad, dependencia software y hardware, vulnerabilidad a incidentes locales, costos de instalación y mantenimiento; movilidad y acceso a los datos, escasa flexibilidad, escalabilidad del servicio [7].

B. Justificación

Los servidores físicos actualmente enfrentan problemáticas relacionadas con la flexibilidad, agilidad y costos, generando pérdidas económicas a las empresas. La implementación de servidores Cloud ha tenido un mayor auge en las organizaciones, ya que garantizan más flexibilidad respecto al incremento de la demanda y a su vez ofrecen un mejor servicio a los clientes. Estos servidores se caracterizan porque permiten automatizar las tareas de seguridad manual, lo que les admite a las empresas enfocarse en la innovación de su negocio.

A esto se suma que el entorno tecnológico está en constante evolución y es difícil justificar la inversión necesaria para adquirir servidores físicos, los costos asociados a su funcionamiento y mantenimiento. En este sentido, el auge de la computación en la nube ofrece a las empresas de todos los tamaños ahorros significativos en estos costos y mayor efectividad en sus procesos. Esto último se consigue con una arquitectura basada en microservicios, es decir, pequeños servicios que se ejecutan de forma autónoma y se comunican entre sí, permitiendo responder las demandas con mayor rapidez, ya que el desarrollo y la adaptación son más ágiles para las aplicaciones, lo que lo convierte en un enfoque diferente al desarrollo de software tradicional.

Ahora bien, tener la posibilidad de enfrentarse a ese contexto y responder a la solicitud de un cliente es una experiencia y una oportunidad académica y profesional más que relevante para una persona que está a portas de egresar. Precisamente este fue el resultado del convenio entre la empresa ACCENTURE y la Corporación Universitaria Autónoma del Cauca por medio de la práctica

profesional, que permitió la aplicación de conocimientos técnicos, además del entrenamiento (onboarding) por parte de la organización para explorar alternativas que pudieran suplir las necesidades expuestas por el cliente, al mismo tiempo de constituirse en una alternativa de grado académico.

Una vez inició la pasantía, las dinámicas de la empresa permitieron entender las actividades que se ejecutan internamente, lo que facilitó la adaptación y creación de un sistema informático capaz de solventar las carencias actuales, mediante el cumplimiento del objetivo en el marco de la implementación y verificación de la arquitectura Cloud en AWS orientada a microservicios para el cliente establecido por la empresa ACCENTURE.

Cabe mencionar que, antes de llegar a la resolución definitiva, se hizo una exhaustiva evaluación de diferentes escenarios hipotéticos. Se exploraron alternativas como la implementación de una arquitectura basada en contenedores con Docker y Kubernetes, así como la adopción de plataformas de nube privada para mayor control y seguridad de los datos. Sin embargo, al evaluar los requisitos del cliente, los costos asociados y la escalabilidad necesaria, se concluyó que la arquitectura Cloud en AWS, enfocada en microservicios, era la opción más adecuada.

Estas evaluaciones adicionales permitieron tener una visión más amplia y fortalecer la solidez de la solución desarrollada. En el marco de la implementación y verificación de la arquitectura Cloud en AWS orientada a microservicios para el cliente de ACCENTURE, se creó y adaptó un sistema informático capaz de solventar las necesidades actuales. Esto generó beneficios para la empresa al aportar una arquitectura Cloud AWS a su cliente, y para el practicante al obtener una valiosa experiencia y adquirir conocimientos profesionales en un entorno de innovación tecnológica.

II. OBJETIVOS

A. Objetivo General

Implementar una arquitectura Cloud en AWS orientada a microservicios para un cliente establecido por la empresa ACCENTURE.

B. Objetivos Específicos

- Modelar la arquitectura Cloud orientada a los microservicios para las necesidades de la empresa especificada.
- Construir la arquitectura modelada a través de Contenedores AWS.
- Validar la implementación de la arquitectura propuesta en AWS para el cliente especificado.

III. MARCO TEÓRICO

A. Marco Referencial

1) Antecedentes

La pasantía se desarrolló en Accenture L.T.D.A., un grupo empresarial que adopta AWS con sus clientes debido a que es eficiente y organizada por su seguridad, velocidad y escalabilidad. Su organización de trabajo se aprecia en la Figura 1.

La empresa ha sumado todos sus recursos, experticia técnica y conocimiento de la industria al uso de Amazon Web Services, logrando la transformación e innovación global en un único punto de contacto que acelera la generación de valor a los negocios por medio de la adopción de Cloud. De esta manera, ha aportado y apoyado a los clientes end-to-end, desde la ideación de la estrategia, migración, operación y ejecución; siendo líderes mundiales en el uso de la nube.

Esta alianza de Accenture con Amazon lleva más de 12 años facilitando el alcance rápido de la evolución tecnológica en el área Cloud AWS del equipo Accenture, que cuenta con arquitectos entrenados y certificados con recursos globales de entrega y las últimas tecnologías de AWS, las cuales dan como resultado soluciones que promueven la transformación apoyada por la nube. Así

mismo, se destaca Accenture AWS Center of Excellence, un portal donde los clientes pueden ver, aprender y colaborar en los proyectos realizados por esta empresa [8].

Accenture y AWS han trabajado en más de 20.000 proyectos de soluciones Cloud y 47.000 profesionales están registrados en AWS Global Service Integrator Partner & Premier Partner en las regiones de NA, APAC y EMEA, por sus siglas en inglés, relacionados con múltiples servicios e infraestructura globales. La empresa cuenta con nueve competencias AWS y seis AWS Partners Programs con énfasis en la innovación [8]. Además, se destacan los 12 años de trayectoria colaborando para apoyar a clientes en su cambio a la nube, la implementación de AWS Cloud en toda la empresa y la transformación digital incluyendo Data Analytics, IoT y AI.

Esta organización cuenta con un amplio portafolio de ofertas compatibles para apoyar de manera E2E la transformación a *Cloud*, dentro de la seguridad en AWS se encuentran ventajas tales como la reducción del riesgo y aceleración de la adopción de híbridos e implementaciones en la nube en AWS, utilizando estrategias y capacidades de defensa en profundidad; incluyendo seguridad administrada por AWS y a su vez el aprovechamiento de las arquitecturas de referencia vetadas, aceleradores y automatización. La realización de las migraciones de SAP e implementaciones en AWS proporcionan servicios industrializados impulsados por aceleradores y automatización que contiene configuración de infraestructura y soporte de ejecución.

El desarrollo nativo para la nube de AWS ha estado enfocado en la visualización, la creación rápida de prototipos y el desarrollo de soluciones arquitectónicas y de aplicaciones fluidas más dinámicas que les permitan a las empresas innovar y acelerar los resultados en el sector público, de la salud, servicios financieros y demás cuando se trata de soluciones industriales habilitadas por AWS.

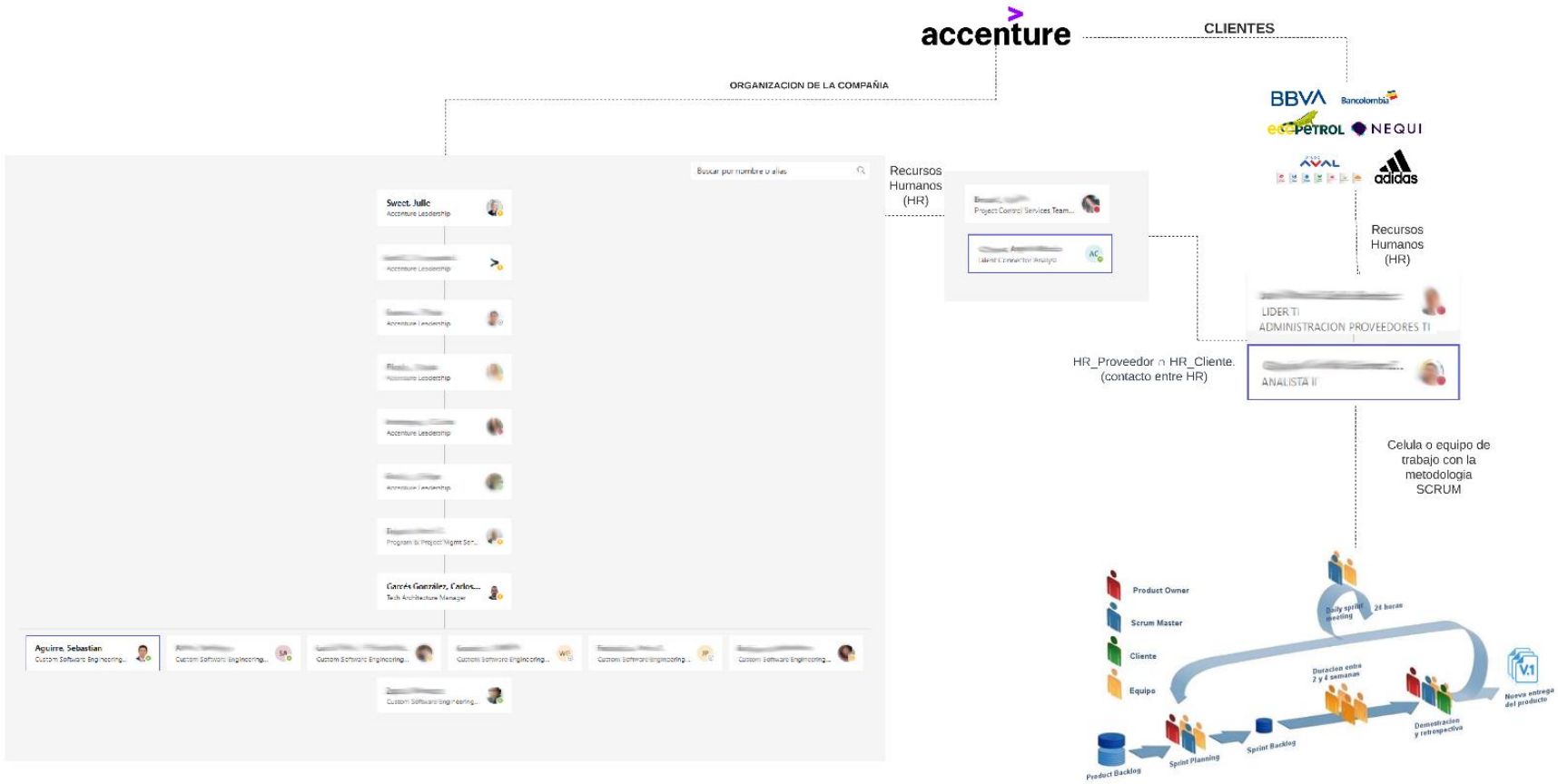
Por último, la arquitectura en AWS permite analizar rápidamente cualquier carga de trabajo dirigida o ya en ejecución, al igual que la reducción del riesgo, ya que proporciona comprensión desde el principio del proceso y define los próximos pasos en su desarrollo.

Ejemplo de esa transformación e innovación son los bancos, que buscan ofrecer mejores servicios a sus usuarios. Es el caso de las tres aplicaciones que se reseñan a continuación y que tienen funcionalidades similares a las que poseerá el entregable del proyecto en el que se desarrolló la pasantía. En primer lugar, se encuentra la sucursal virtual personas de Bancolombia, que es el portal digital en el que los usuarios pueden realizar transferencias, pagos y otras consultas de manera segura y ágil.

Otro caso es el de Banca Móvil, la aplicación del Banco de Bogotá, que se constituye como una de las novedades tecnológicas en lo que a transacciones financieras se refiere, pues, desde su creación en 2014 se han hecho 105 millones de transacciones, optimizando el tiempo de los clientes y transformando su relación con el banco.

Finalmente, Davivienda Móvil también es una aplicación similar que les permite a las personas realizar consultas y transacciones, no solo a sus clientes, sino también a otros usuarios.

FIGURA 1. ORGANIZACIÓN Y EQUIPOS DE TRABAJO.



B. Marco Conceptual

1) Base De Datos (BD)

La base de datos (BD) es un concepto clave en este proyecto porque funciona como un almacén organizado y estructurado de información relacionada con el desarrollo que se propone. La BD se compone de tablas con columnas y filas para almacenar los datos, garantizando el acceso simultáneo, la integridad, la seguridad y el monitoreo continuo. Además, ofrece respaldo y aislamiento lógico y físico de la información [9]. La BD está intrínsecamente ligada a los objetivos de modelar y validar la arquitectura propuesta en este trabajo.

2) Back End

El Back End desempeña un papel fundamental en la implementación de una arquitectura Cloud en AWS orientada a microservicios porque es el responsable de la lógica interna que brinda funcionalidad a la interfaz gráfica y trabaja directamente con lenguajes de programación [10]. Su función incluye optimizar recursos, garantizar seguridad y estándares de calidad, y asegurar la accesibilidad. Internamente, el Back End se encarga de satisfacer las necesidades del usuario, como búsquedas, reportes y modificaciones de datos. Esta relación entre el Back End y Front End se ilustra en la Figura 2.

FIGURA 2. BACK END Y FRONT END DEVELOPMENT

Fuente: [11]

3) Cloud

La computación en la nube implica alojar servicios informáticos de forma remota en centros de datos externos en lugar de utilizar servidores dedicados [12]. En el contexto de este proyecto, esto le permitirá al cliente acceder a los servicios de manera remota y pagar por su uso a través de proveedores de la nube, evitando la necesidad de adquirir y mantener activos digitales en sus propias instalaciones. La Figura 3 proporciona una comparativa visual de las diferencias entre los enfoques tradicionales de servidores y la computación en la nube.

FIGURA 3. SERVIDOR- CLOUD

Servidor en la nube	Servidor local
Precios más bajos y escalables	Altos costes de equipo y servicios
Actualizaciones automáticas	Costes de actualización y renovación
99,9 % accesibilidad	Susceptible de sufrir problemas o fallos
Sin coste de infraestructura ni soporte	Necesidad de un espacio físico
Sin necesidad de backup	Respaldo manual
Sin consumo energético	Alto consumo energético.
Información disponible 24/7/ 365	Coste por acceso remoto
Altos estándares de seguridad	La seguridad depende de la empresa
Pago por servicio SaaS	Coste de servidor + configuración
Escalabilidad infinita	Limitado al crecimiento de la empresa

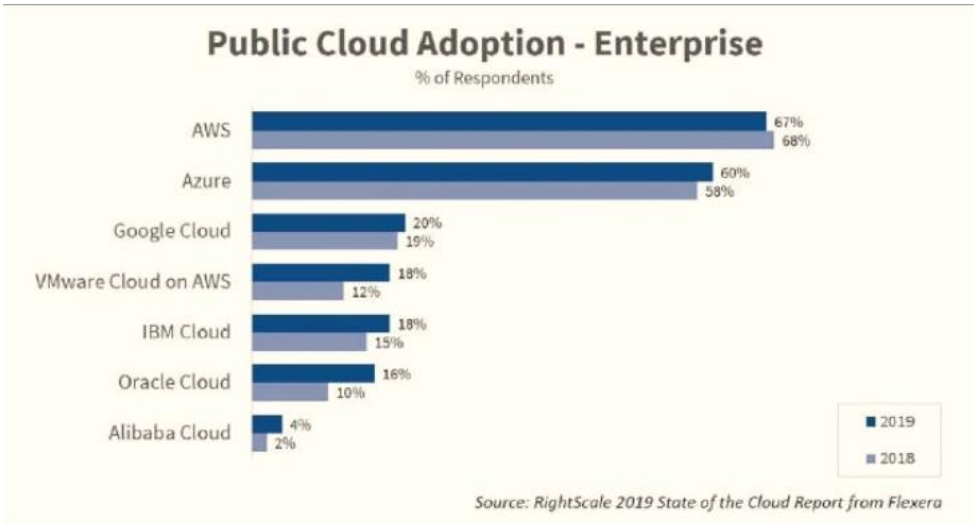
Fuente: [13]

4) Proveedores Cloud

En el entorno empresarial actual, la migración a la nube es una tendencia dominante en la era digital, y existen varios proveedores que ofrecen estos servicios. Según una encuesta realizada a múltiples industrias y compañías [14], AWS fue altamente valorado con un 69% de uso de aplicaciones en la nube en 2018, seguido de cerca por Azure con un 58%. Esto demuestra la amplia aceptación y satisfacción de los usuarios y de ahí la pertenencia de usar estos proveedores para el desarrollo del

que se ocupó este trabajo. La Figura 4 respalda estos datos y muestra la preferencia de los proveedores en el mercado de la nube.

FIGURA 4. PROVEEDORES NUBE PÚBLICA, CLOUD.



Fuente: [15]

5) Amazon Web Services (AWS)

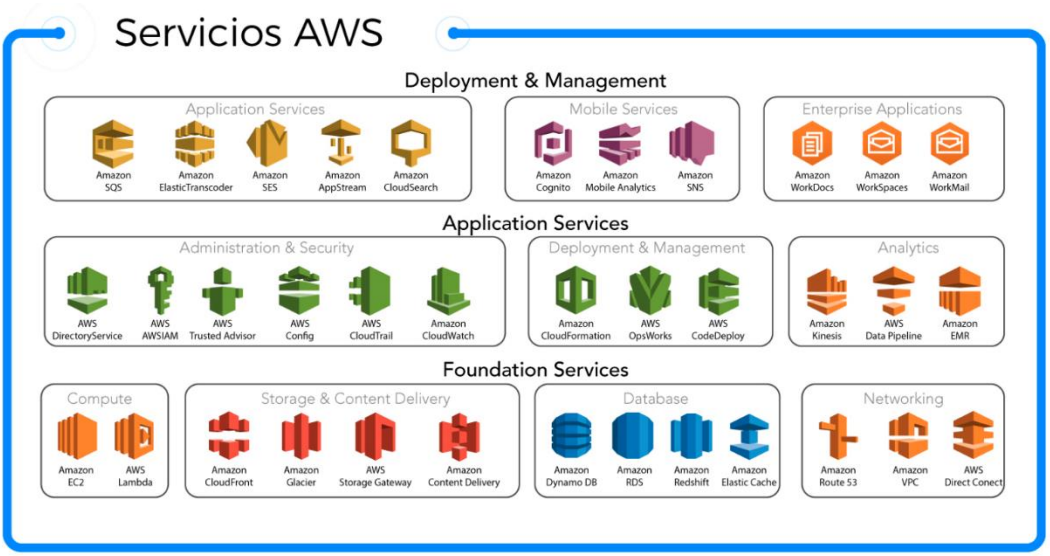
AWS es una plataforma líder en el ámbito de las tecnologías de la información, ofreciendo beneficios como flexibilidad, aplicaciones rápidas y sencillas, escalabilidad y un sólido mantenimiento seguro de datos y operaciones a gran escala. Además, se destaca por su amplia variedad de servicios y su capacidad para soportar otras plataformas [4]. Una ventaja destacada de AWS es su competitividad en precios, ya que busca reducir costos sin comprometer la calidad del producto. Estos aspectos respaldan la elección de AWS como la plataforma preferida para este proyecto.

6) Servicios de AWS

Los productos y servicios que ofrece Amazon se adaptan a las necesidades de la arquitectura que se implementó, abarcando áreas de cómputo, almacenamiento, base de datos, redes, entrega de contenido y herramientas para desarrolladores [16]. Estos servicios permitieron gestionar y supervisar de manera

eficiente los recursos de la infraestructura, proporcionando aprovisionamiento, configuración y administración automática a escala. Además, ofrecen herramientas para el monitoreo en tiempo real, el control de registros y métricas, y garantizan la seguridad y el cumplimiento normativo; aspectos que fueron requeridos por el cliente. La Figura 5 muestra la composición y funcionalidad de estos servicios de AWS.

FIGURA 5. SERVICIOS AMAZON WEB SERVICES



Fuente: [17]

7) API

Las API proporcionan una forma simplificada de conectar la infraestructura y desarrollar aplicaciones nativas de la nube, así como compartir datos con clientes y usuarios externos [18]. Estas API permitieron integrar de manera eficiente los servicios y funcionalidades del cliente en el entorno de microservicios, facilitando la conexión la arquitectura.

8) Arquitectura Orientada a Microservicios

Para comprender la arquitectura de microservicios, se debe definir qué son los microservicios. Como sugiere el nombre, estos son pequeños servicios que se ejecutan de forma autónoma y se

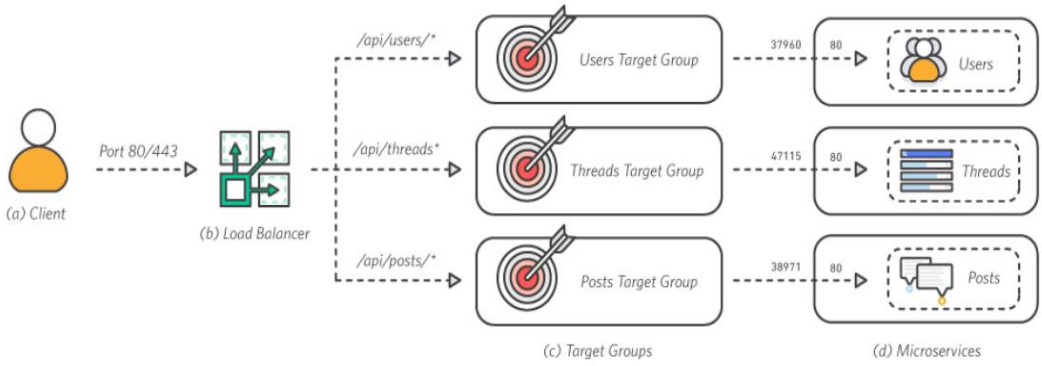
comunican entre sí. Debido a su fácil escalabilidad, este enfoque arquitectónico es particularmente adecuado cuando se requiere compatibilidad con diferentes plataformas (web, móvil, entre otras).

Fundamentalmente, el desarrollo de proyectos basado en esta metodología forma una aplicación o herramienta a través de la relación de varios servicios independientes desplegados según sea necesario [19].

Esta arquitectura es altamente escalable y compatible con diferentes plataformas, lo que la hace adecuada para satisfacer las necesidades del cliente en términos de flexibilidad y compatibilidad. Al construir una aplicación o herramienta mediante la relación de varios servicios independientes desplegados según sea necesario, los microservicios permiten una rápida respuesta a las demandas del negocio y evitar en caso de errores el fallo de toda la arquitectura. [20].

Cada servicio es un código base independiente, que puede administrarse por un equipo de desarrollo, como se aprecia en la Figura 4[21]. Estos servicios se pueden implementar de forma independiente, y son responsables de mantener sus propios datos o estado externo, además de comunicarse entre sí, en cuanto a los detalles de implementación interna de cada servicio están ocultos para otros tipos servicios y no tienen que compartir la misma pila de tecnología, biblioteca o marco de trabajo. A continuación, se ilustra la arquitectura de los microservicios en la Figura 6.

FIGURA 6. ARQUITECTURA DE MICROSERVICIOS



Fuente: [22]

9) Contenedores

Los contenedores son una tecnología que permite empaquetar y ejecutar aplicaciones de forma ligera y aislada en diferentes entornos, tanto locales como en la nube. Proporcionan un entorno ágil y flexible para el desarrollo, implementación y administración de aplicaciones, lo que es especialmente relevante para adaptarse rápidamente a los requisitos cambiantes del negocio [23]. AWS es una plataforma líder en la ejecución de contenedores en la nube, con un alto porcentaje de contenedores alojados en su infraestructura. Clientes como Samsung, Expedia, GoDaddy y Snap eligen ejecutar sus contenedores en AWS debido a su seguridad, confiabilidad y escalabilidad [24]. Al aprovecharlos se pudo construir y desplegar eficientemente la arquitectura de microservicios para el cliente de Accenture, brindando los beneficios de agilidad, escalabilidad y confiabilidad necesarios para el éxito del proyecto.

10) Infrastructure as a Service (IaaS)

La IaaS se refiere al aprovisionamiento y gestión de la infraestructura de forma automatizada y basada en código, en lugar de depender de configuraciones manuales. Mediante definiciones declarativas o scripts se puede gestionar y desplegar la infraestructura necesaria, incluyendo hardware,

recursos virtuales, plataformas, sistemas de contenedores y servicios, de manera eficiente y escalable. La IaaS permite separar la configuración de la infraestructura de los componentes en los que se implementa, lo que facilita el almacenamiento, compartición, auditoría y aplicación de estas configuraciones como código [25]. Al utilizar la IaaS en AWS se pudo modelar y construir la arquitectura de microservicios de manera eficiente y automatizada, cumpliendo así con los objetivos específicos de este proyecto, y asegurando la escalabilidad, flexibilidad y eficiencia de la infraestructura.

11) CDK

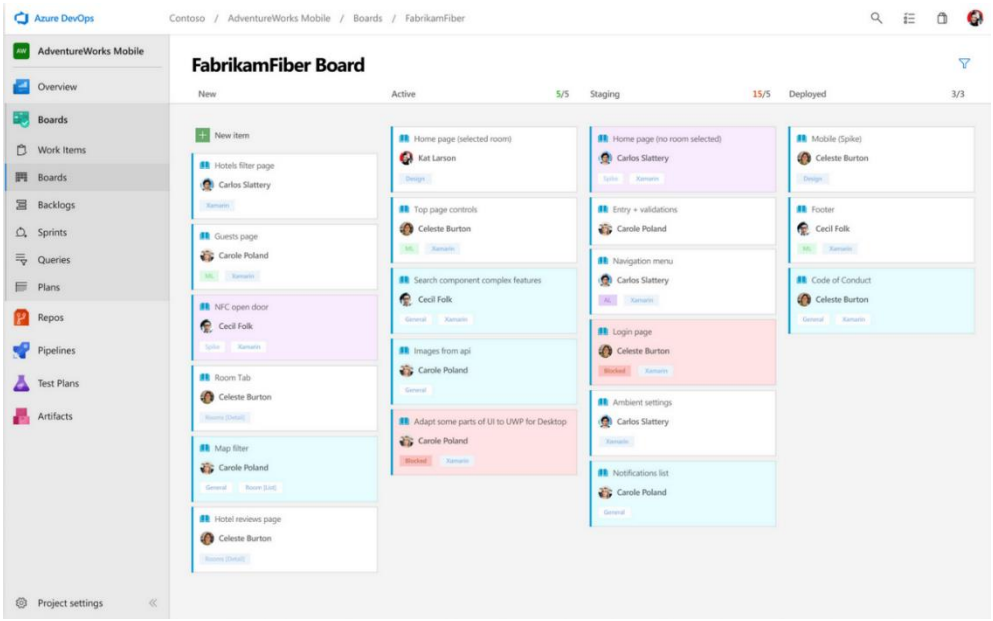
El CDK es un marco de desarrollo de software de código abierto que permite definir los recursos de las aplicaciones en la nube utilizando lenguajes de programación conocidos como TypeScript, Python, Java y .NET [26]. Al usar el CDK se pudo modelar y construir la arquitectura de forma más eficiente y familiar, aprovechando las capacidades y flexibilidad de TypeScript, lo cual permitió adaptar y personalizar la arquitectura según las necesidades específicas del cliente y validar su implementación en AWS.

12) Azure DevOps

Azure DevOps es una herramienta que se usó en este proyecto porque proporciona capacidades colaborativas para la gestión de proyectos de desarrollo de software y se integra con varias plataformas y lenguajes de programación [27]. Entre sus herramientas se encuentran Azure Boards, el cual se muestra en la Figura 7 y permite hacer seguimiento y gestionar las tareas del proyecto. Azure Repos, que se aprecia en la Figura 8, proporciona control de versiones para el código fuente; Azure Pipelines facilita la compilación, implementación y prueba del desarrollo como se ve en la Figura 9. Por último, Azure Artifacts permite almacenar y generar alimentaciones de componentes de proyectos, ver la Figura 10 [27]. Al utilizar estas herramientas se aprovecharon sus beneficios para modelar, construir

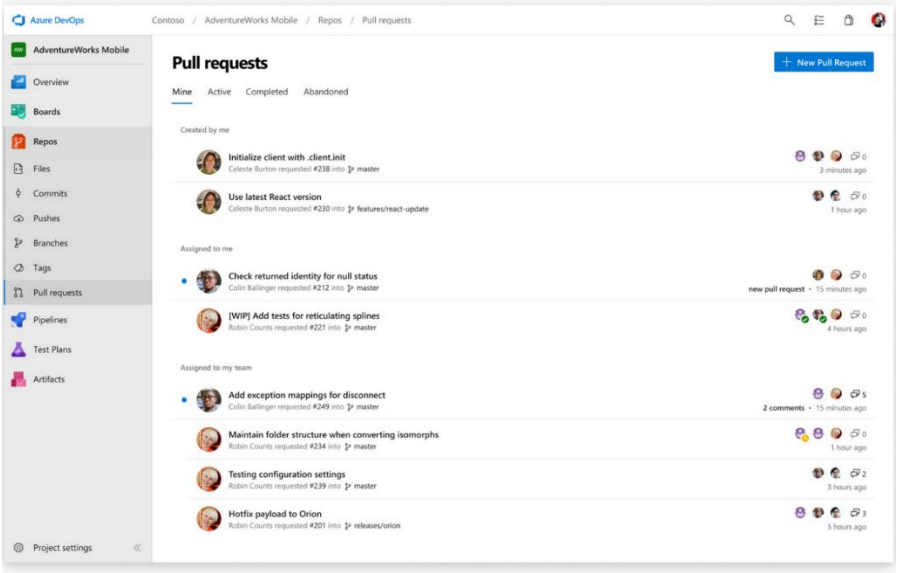
y validar la arquitectura de microservicios en AWS, optimizando el proceso de desarrollo y colaboración en el proyecto.

FIGURA 7. AZURE BOARDS.



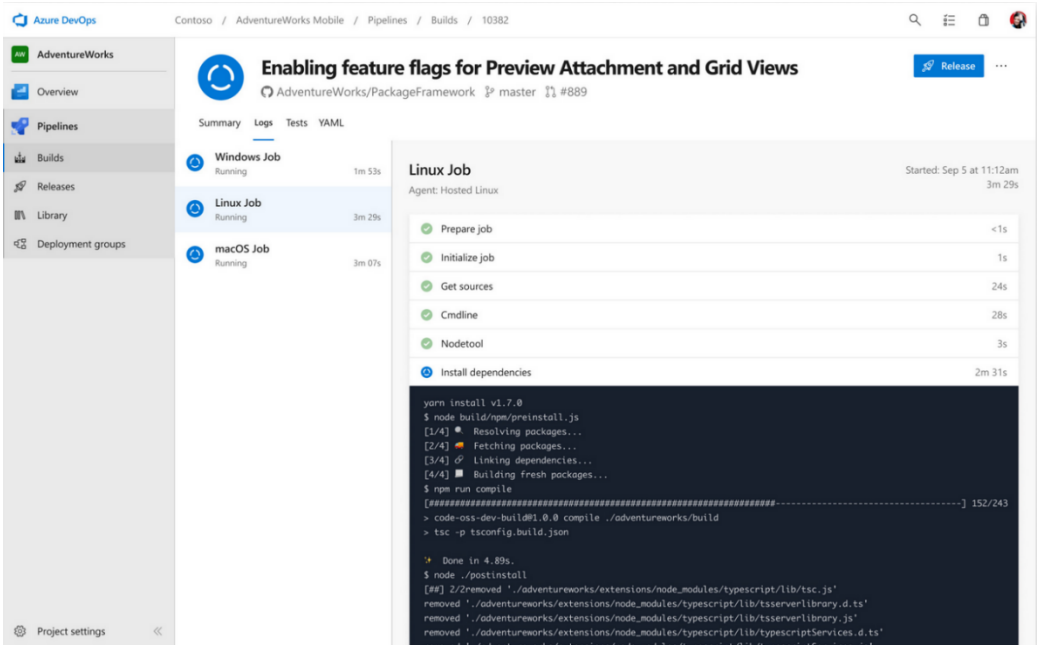
Fuente [27]

FIGURA 8. AZURE REPOS.



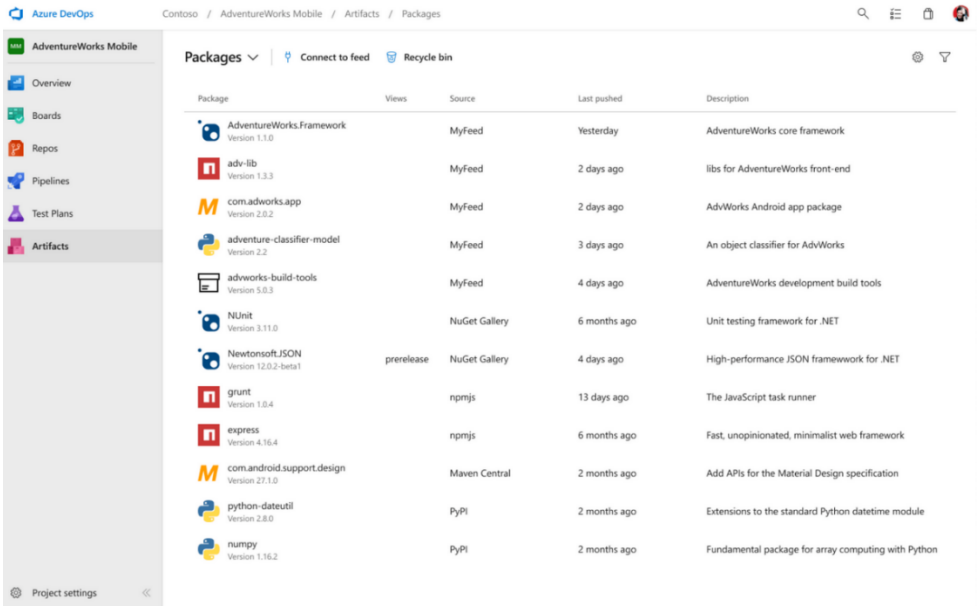
Fuente: [27]

FIGURA 9. AZURE PIPELINES.



Fuente: [27]

FIGURA 10. AZURE ARTIFACTS.



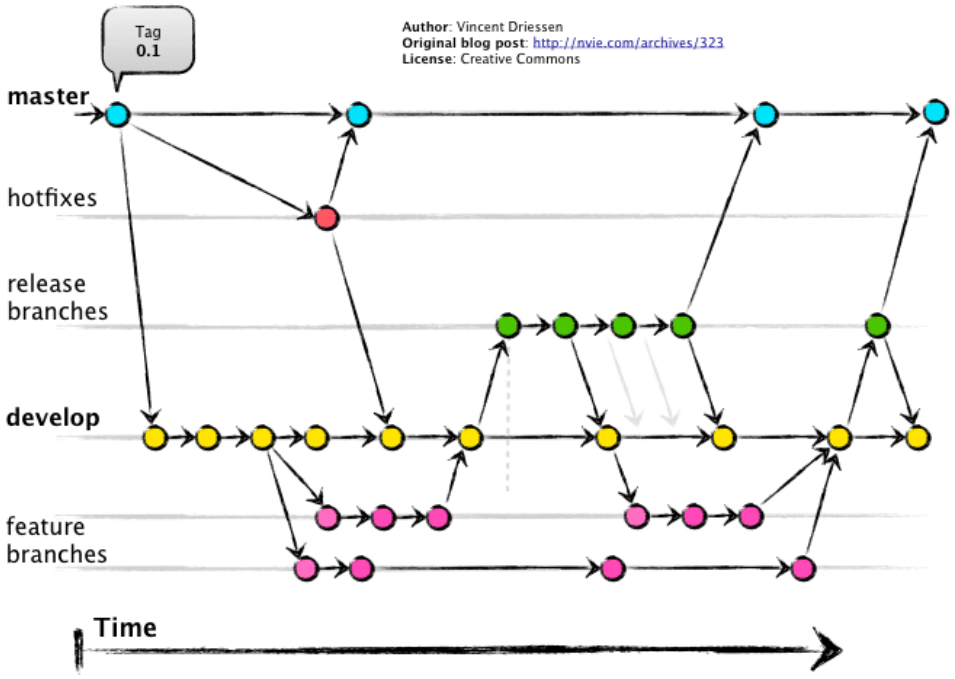
Fuente: [27]

13) Versionamiento con Git y Git Hub

Permite almacenar en repositorios las diferentes versiones de un proyecto, para eso se crean ramas en las que se van incorporando las modificaciones que cada desarrollador hace al código fuente, como se observa en la Figura 11. Existen cinco tipos de ramas [28], [29].

En este caso se usó la rama Feature para cada funcionalidad, Develop para integrar nuevas funcionalidades, Release para pruebas y regresiones, Master para el historial de lanzamientos y Hotfix para corregir errores. Esta metodología de versionamiento brindó un control eficiente sobre el desarrollo y permite una colaboración efectiva entre los desarrolladores, asegurando la calidad y la estabilidad de la arquitectura implementada.

FIGURA 11. VERSIONAMIENTO CON GIT.



Fuente: [30]

14) Postman

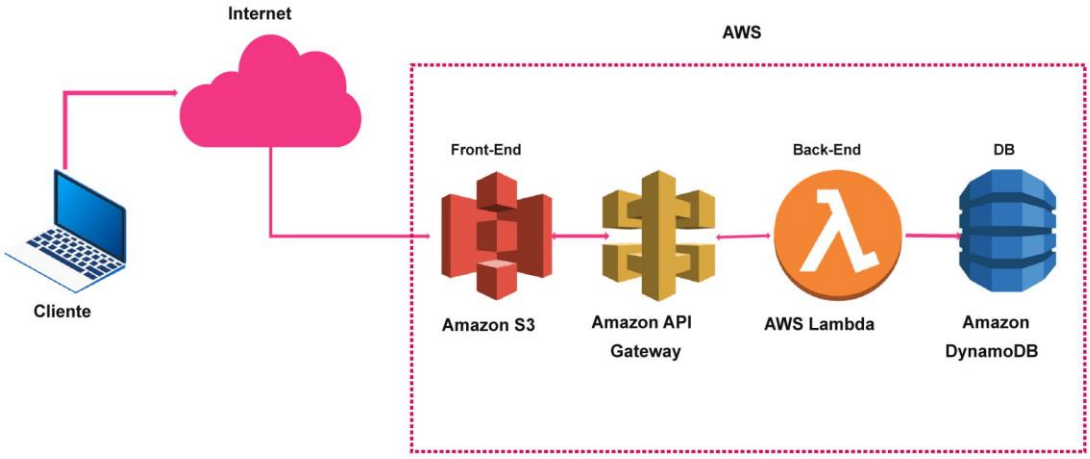
Es una aplicación que tiene herramientas nativas para sistemas operativos como Windows, Linux y Mac. Cuenta con varios métodos que permiten tomar acción ante las solicitudes [31].

Sus métodos Get, Post, Put, Patch y Delete permiten interactuar con la base de datos y realizar operaciones como obtener, agregar, reemplazar, actualizar y eliminar información. Además, Postman proporciona respuestas claras y categorizadas mediante códigos de estado como el rango 200 para solicitudes exitosas, el rango 400 para errores del cliente y el rango 500 para errores del servidor. Esta herramienta fue fundamental para probar y validar la funcionalidad de la arquitectura implementada en AWS, asegurando la calidad y correcto funcionamiento del proyecto.

15) Arquitectura Tipo Serverless

La adopción de una arquitectura tipo serverless se relaciona directamente con los objetivos del proyecto de implementación de una arquitectura Cloud en AWS orientada a microservicios. Al utilizar esta arquitectura se elimina la necesidad de administrar servidores por parte del cliente, ya que AWS se encarga de esa tarea [32], como se muestra en la Figura 12. Esto permitió enfocarse en la modelación, construcción y validación de la arquitectura cloud y los microservicios, optimizando así el desarrollo y la implementación del proyecto.

FIGURA 12. ARQUITECTURA SIN SERVIDOR.

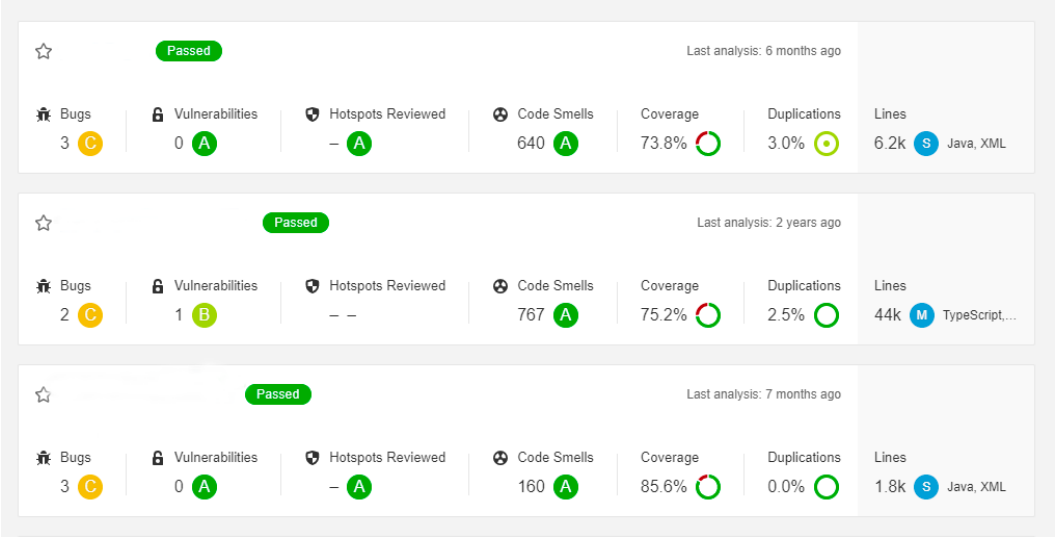


Fuente: [33]

16) SonarQube

Con esta plataforma de código abierto se realizaron inspecciones continuas de la calidad del código fuente, lo cual fue crucial durante la fase de testing y desarrollo del proyecto. Con soporte para 29 lenguajes de programación diferentes [34], SonarQube brindó resultados detallados sobre la calidad del código, lo que ayudó a identificar y abordar posibles problemas y mejorar la eficiencia y confiabilidad de la arquitectura implementada en AWS. Los resultados de las pruebas se observan en la Figura 13.

FIGURA 13. ANÁLISIS EN SONARQUBE.

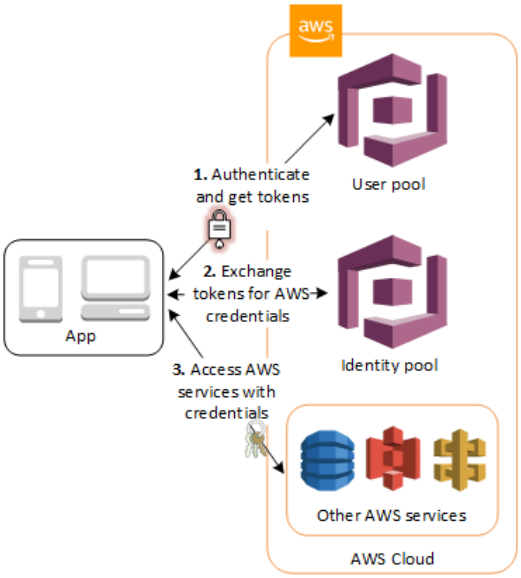


Fuente: [34]

17) Servicios AWS

El uso de servicios como AWS WAFF, AWS Cloudfront, AWS S3 [37], AWS Cognito, AWS API REST, AWS Lambda y AWS DynamoDB posibilitó el diseño de una arquitectura robusta y escalable. Estos servicios proporcionan funcionalidades clave como la protección contra ataques a aplicaciones web, la distribución eficiente de contenido, el almacenamiento seguro de datos, la gestión de autenticación y autorización como se ve en la Figura 14; la creación de API y la ejecución de código sin necesidad de administrar servidores. Además, DynamoDB ofrece una base de datos NoSQL con baja latencia y cifrado en reposo para garantizar la seguridad de la información [35, 36, 37, 38, 39, 40].

FIGURA 14. AMAZON COGNITO.



Fuente: [38]

18) Npm

Npm es una herramienta fundamental en el desarrollo de aplicaciones JavaScript, ya que permite gestionar las dependencias del proyecto, compartir e instalar paquetes. En el caso de la implementación de una arquitectura Cloud en AWS orientada a microservicios, el uso de Npm es relevante para administrar las dependencias de los microservicios desarrollados, lo que facilita la construcción y despliegue de los mismos a través de contenedores AWS. Además, Npm proporciona un repositorio online para publicar los paquetes y una herramienta para interactuar con dicho repositorio. [42].

19) Npm run test

Es un módulo para crear pruebas de JavaScript, cuyos resultados estarán en formato TAP [43]. En el desarrollo que se describe en este documento permitió realizar pruebas unitarias y de integración en los diferentes microservicios. Esto contribuye a garantizar la calidad y el correcto funcionamiento de los microservicios en el entorno de AWS

20) Npm Audit

El comando de auditoría envía la descripción de las dependencias y se verifica si hay vulnerabilidades conocidas. En caso de existir estas últimas se calcula el impacto y la corrección apropiada, la cual se hará en el árbol de paquetes [44]. Debido a que hace una evaluación de seguridad, identificando posibles vulnerabilidades en las dependencias utilizadas en los microservicios permite tomar medidas correctivas adecuadas para mitigar riesgos y garantizar la integridad y seguridad de los microservicios implementados; requisito del cliente que se pudo cumplir.

21) Lenguajes de Programación

Este conjunto de instrucciones les permite a las personas interactuar con las computadoras, a través de algoritmos e instrucciones escritas que la computadora convierte en lenguaje de máquina para procesar grandes cantidades de información [45].

En la implementación de la arquitectura se emplearon TypeScript como lenguaje de programación. Por su parte, Node.js , se empleo como entorno de ejecución para compilar y ejecutar el código Typescript. Además, se utilizó JSON para la configuración de datos, y Python se empleo usó en el desarrollo de las lambdas encargadas de las peticiones [46, 47, 48, 49, 50].

22) Serialización de Datos

La serialización de datos es un proceso esencial en el proyecto, ya que permite transformar objetos en una secuencia de bytes para su almacenamiento, transmisión y posterior reconstrucción. En este contexto, el uso de YAML como lenguaje de serialización facilita la legibilidad por parte de las personas y su interoperabilidad con diversos lenguajes de programación. Además, la implementación de operaciones CRUD (Create, Read, Update, Delete), que se observan en la Figura 15, en las bases de datos permite gestionar la información almacenada de forma eficiente y cumplir con los requisitos del proyecto [51, 52, 53].

FIGURA 15. CRUD.

Fuente: [53]

23) Editores de Código Fuente

Son herramientas que le permiten al desarrollador editar el código fuente en diferentes lenguajes de programación [54]. En este caso, Visual Studio y AWS Cloud9 ofrecieron capacidades de edición, depuración y compilación de código, así como otras funcionalidades que agilizaron el desarrollo de las aplicaciones en la arquitectura Cloud orientada a microservicios en AWS [55, 56].

24) Ubicaciones de Bordes (Edges Locatios)

Son las que alojan a Amazon Cloud Front y permite distribuir contenido a los clientes. Esas solicitudes se dirigen automáticamente a las ubicaciones de bordes más cercanas para que los contenidos lleguen rápido a los usuarios; esto se puede observar en la Figura 16. Generalmente están ubicadas en zonas muy pobladas [57], lo cual garantiza una entrega rápida de los contenidos y una mejor experiencia para los usuarios.

FIGURA 16. UBICACIONES DE BORDES.

Fuente: [57]

25) Programación Extrema (XP)

La programación extrema (XP) se relaciona con el proyecto al proporcionar una metodología de desarrollo de software que fomenta la calidad del código. Sus prácticas, como el desarrollo de pruebas (TDD), la integración continua, las pruebas al lado del cliente y las iteraciones cortas, contribuyeron a garantizar la implementación exitosa de la arquitectura Cloud orientada a microservicios en AWS Estas prácticas permiten una mayor eficiencia en el proceso de construcción y validación de la arquitectura, resultando en un código de alta calidad [58]. El proceso se aprecia en la Figura 17.

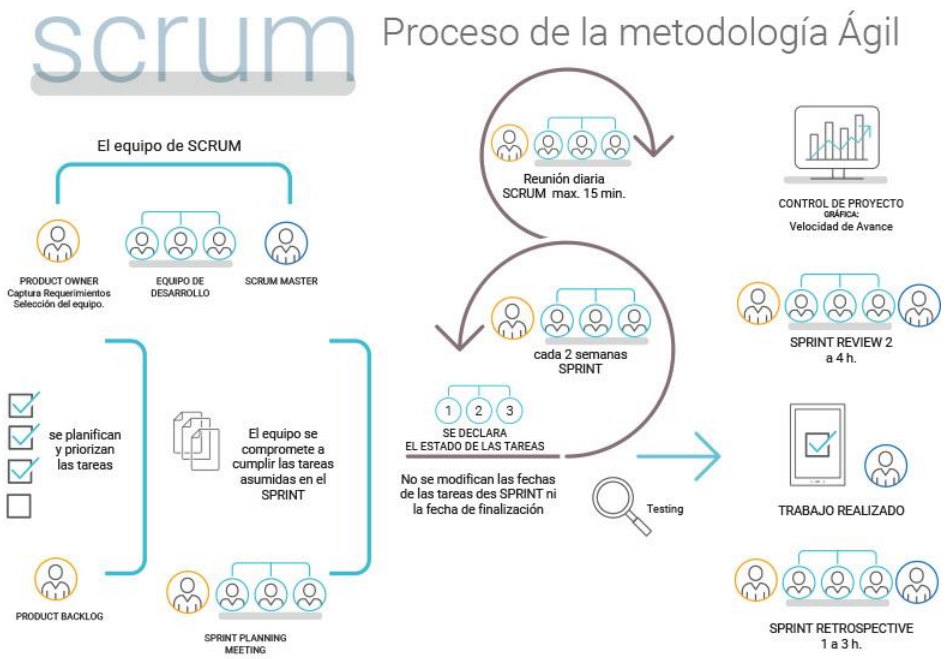
FIGURA 17. METODOLOGÍA XP.

Fuente:[59]

26) SCRUM

Este marco de trabajo se eligió porque ofrece un enfoque flexible y mediante la realización de sprints en un intervalo de tiempo de dos a cuatro semanas, SCRUM facilita la entrega de requerimientos funcionales y otras tareas establecidas por el cliente [60]. Esta metodología ágil promueve la colaboración, la transparencia y la adaptabilidad, contribuyendo al éxito de la implementación de la arquitectura Cloud orientada a microservicios en AWS. De manera visual este proceso se presenta en la Figura 18.

FIGURA 18. METODOLOGÍA SCRUM.



Fuente: [61]

27) Servicios de identificación en AWS

Los servicios de identificación en AWS, como IAM (Identity and Access Management), los roles y las políticas, son fundamentales en la implementación de una arquitectura Cloud orientada a microservicios en AWS, ya que permiten controlar y gestionar los permisos de acceso a los recursos de AWS para el usuario y el asesor, incluyendo el inicio de sesión y la definición de políticas de permisos específicas. Proporcionan una capa de seguridad y control en la arquitectura, garantizando que solo los usuarios autorizados tengan acceso adecuado a los recursos [62, 63, 64].

28) SQS (Amazon Simple Queue Service)

Es un servicio de AWS que se encarga del manejo de la cola de mensajes (ver Figura 18), está gestionado por el sistema y es ideal para trabajar microservicios, aplicaciones sin servidor y otros sistemas distribuidos [65]. En la Figura 19 se muestra cómo opera.

Su integración en el proyecto facilita la coordinación y evita la pérdida de información en caso de alto flujo, contribuyendo a una implementación segura y exitosa de la arquitectura en AWS.

FIGURA 19. AMAZON SQS.

Fuente: [66]

29) Token

Es un identificador que devuelve que regresa a los datos sensibles mediante la tokenización para garantizar la seguridad [67]. Aquí protegen y controlan el acceso a los datos, evitando la exposición directa de la información sensible. Contribuye a la seguridad y confidencialidad de los datos almacenados y transmitidos cuando ingresen los usuarios a través del login.

30) Crecimiento horizontal

El conocimiento y entendimiento del crecimiento horizontal es fundamental en la implementación de una arquitectura Cloud orientada a microservicios en AWS dado que la arquitectura se basa en un enfoque serverless, es decir, sin la gestión directa de servidores individuales. Por tanto, el crecimiento horizontal se convierte en el mecanismo principal para escalar y aumentar la capacidad de procesamiento del sistema. Al agregar o eliminar recursos informáticos de manera flexible y automática según la demanda, se logra una escalabilidad eficiente y optimizada, permitiendo que los

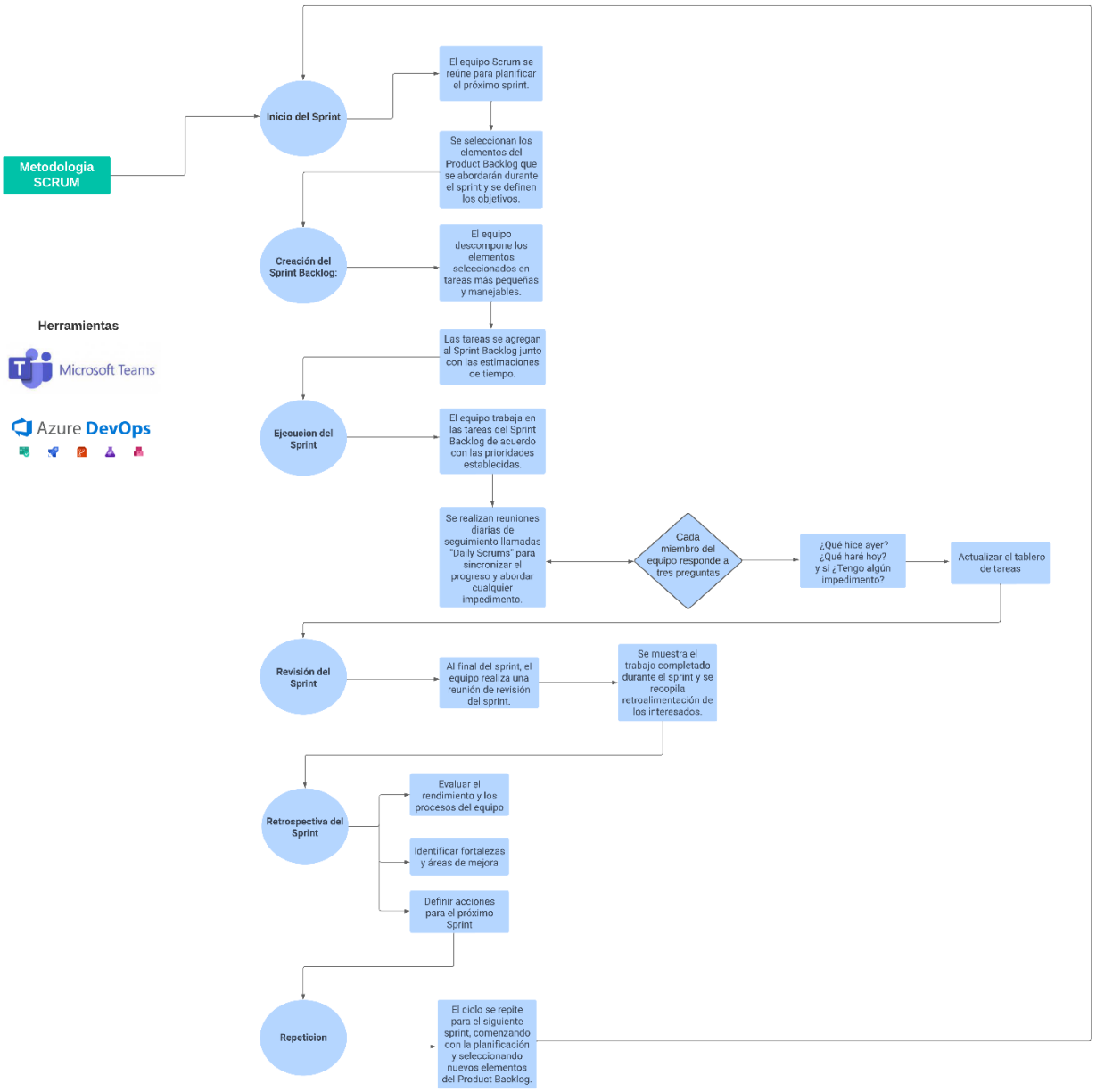
microservicios se adapten a las necesidades del cliente. En suma, se garantiza un rendimiento óptimo y una respuesta ágil ante cambios en la carga de trabajo [68].

IV. Metodología

La metodología que se usó para la ejecución de este proyecto es XP dentro del marco de trabajo SCRUM, los cuales fueron escogidos por el cliente debido a su agilidad, eficiencia, orden y fluidez en los equipos de trabajo. Además, se usó la herramienta Azure DevOps para crear los Product Backlog, es decir, las épicas y las tareas a seguir, asignándoles un nivel de prioridad según el esfuerzo requerido. A esta se reconoce como "boards", la cual se muestra en la Figura 6, y es donde se crean las respectivas tareas de acuerdo con las épicas.

Otra herramienta que también se utilizó fue Microsoft Teams, la cual permitió que todo el equipo se reuniera durante todos los sprints, facilitando la constante comunicación. Esta metodología se puede observar de manera gráfica en la Figura 20.

FIGURA 20. METODOLOGÍA DE TRABAJO.



Fuente: [69]

El proceso se llevó a cabo en seis fases con varias actividades cada una. A continuación, se describen por orden de realización y que se pueden observar en la Figura 21.

FIGURA 21. FASES DEL PROCESO.

La primera fase se denominó **Conceptualización. Inducción en arquitectura Cloud en AWS y Microsoft Azure Devops,** la cual consistió en hacer la recolección de información base para afianzar conocimientos y definir los servicios en los que se trabajaría.

La actividad **Capacitación: Inducción en arquitectura Cloud en AWS (onboarding) (**1.1.) consistió en hacer la inducción en la plataforma AWS Training and Certification mediante diferentes módulos que brindaron toda la información de los servicios de AWS.

Luego, en el **Reconocimiento de lenguajes de programación con CDK** (1.2.) se realizaron varios talleres de documentación investigativa del proyecto, los cuales se llevaron a cabo en la plataforma AWS CDK Workshop (https://cdkworkshop.com/).

En la actividad **Reconocimiento de Arquitectura AWS orientada a microservicios (**1.3.) se recopiló información sobre esta arquitectura, aprendizaje que se adquirió durante la revisión de los módulos que brinda AWS Training and Certification.

La actividad **Reconocimiento de Arquitectura AWS tipo serverless (**1.4.) consistió en reunir información de la arquitectura tipo serverless (sin servidor), y para eso se utilizó AWS Training and Certification.

La fase culminó con la actividad **Capacitación en la plataforma Microsoft Azure Devops (**1.5.)**,** la cual permitió recolectar toda la información sobre los servicios de Azure Devops por medio de talleres y prácticas.

Resultados de la fase 1: se adquirió la comprensión de todos los conceptos necesarios para desarrollar una arquitectura en AWS orientada a microservicios tipo serverless. Esto fue posible gracias a la participación en cursos, capacitaciones y talleres, como se puede observar en la Figura 24. Estas actividades permitieron no solo familiarizarse con los fundamentos teóricos, sino también tomar decisiones acertadas sobre la elección de las librerías más adecuadas para el proyecto, tal como se refleja en la Figura 25. Además, se logró un sólido conocimiento del entorno de Microsoft Azure

DevOps, a través de su plataforma Learn. Estos logros sentaron las bases para avanzar con confianza hacia las siguientes etapas del proyecto.

Luego de identificar cada uno de los servicios de AWS en la **fase uno**, se pasó a la **Fase II. Diseño. Modelar los componentes de la arquitectura Cloud en AWS,** donde se hizo el respectivo análisis según la necesidad del cliente y fue aplicado para la modelación de la Arquitectura Cloud en AWS.

Esta etapa inició con la actividad **Revisión de bases de datos cliente Accenture (2.1.),** que consistió en examinar la base de datos del cliente de Accenture, la cual se encuentra alojada en AWS en el servicio de DynamoDB.

Después, en la actividad **Modelación de la arquitectura Cloud AWS (2.2.), orientada a microservicios tipo serverless** se usaron los servicios de AWS para hacer la modelación, proceso en el que se tuvieron en cuenta los requisitos que exigió el cliente con las prácticas recomendadas por el supervisor organizacional.

Para terminar esta fase se llevó a cabo la actividad **Diseño de la arquitectura Cloud AWS, orientada a microservicios tipo serverless** (**2.3.).** Aquí se tuvo en cuenta el modelo con los requisitos del cliente y las respectivas correcciones del supervisor organizacional, se procedió a realizar el diseño, que fue elaborado en Lucidchart e integró los componentes previamente seleccionados y avalados por el cliente de Accenture.

Resultado de la fase 2: se comprendió la base de datos del cliente de Accenture, incluyendo su jerarquía y los datos que la integran, tal como se evidencia en la Figura 27 (la imagen de la jerarquía BD). A partir de esta comprensión, se diseñó el modelo de la arquitectura, considerando los dos roles principales: asesor y usuario, como se aprecia en las figuras 27 y 28. Este logro fue crucial para sentar las bases del proyecto, ya que permitió visualizar de manera clara cómo se relacionan los datos y cómo interactúan los diferentes roles dentro del sistema para desarrollar la arquitectura.

Ahora bien, en la **Fase III. Desarrollo s**e hizo la construcción de los módulos de la arquitectura Cloud en AWS, orientada a microservicios tipo serverless con sus respectivas pruebas.

Se inició con la actividad **Implementación de la arquitectura Cloud en IaaC (3.1.),** desarrollada a través de CDK, el cual permite programar con lenguajes de programación como Typescript, Javascript(Node.js, Json), Python, Yaml (para configurar permisos).

Seguidamente, la actividad **Creación de un contenedor para la Arquitectura AWS (3.2.)** se llevó a cabo con un lenguaje de programación (TypeScript) con los requerimientos identificados y priorizados por la empresa cliente de Accenture.

En la actividad **Implementación de la arquitectura AWS (3.3.)** se programaron en el entorno cloud9 todos los servicios de Cloud en AWS, integrándolos en el contenedor para que toda la arquitectura de microservicios tipo serverless funcionara con los requisitos establecidos por el cliente de Accenture.

Esta parte del proceso terminó con la actividad **Pruebas de la Arquitectura Cloud AWS (3.4.), orientada a microservicios tipo serverless.** Se ejecutó el "npm run-test" con el supervisor organizacional y se validó que cumpliera todos los requerimientos previos en el cloud9 (editor de código fuente) con los módulos ya instalados.

Resultado de la fase 3: a través del entorno del servicio AWS Cloud9, como se muestra en la Figura 31, se desarrolló la arquitectura deseada. Se configuró el código necesario para crear los servicios de AWS, asegurando su correcto funcionamiento como se observa en las figuras 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45. Además, se crearon repositorios específicos para la infraestructura como se observa en Figura 46 y para las lambdas en la Figura 47. Una vez implementada la arquitectura, se llevaron a cabo pruebas internas del cliente utilizando Npm Test, las cuales fueron superadas exitosamente, tal como se aprecia en la Figura 49. Estos logros demuestran el progreso alcanzado y la efectividad de la arquitectura desarrollada en el proyecto.

Después, en la **Fase IV. Despliegue** se realizó el despliegue de la arquitectura en la cuenta del cliente de Accenture en AWS, a través de la plataforma Azure Devops.

Para eso, se realizó la **Creación del repositorio y la rama en Azure Devops (4.1.)** donde se elaboró el repositorio y la rama con el nombre que estableció el cliente para poder guardar la infraestructura como código con el comando "Git Push".

Posteriormente, en la actividad **Conexión de la cuenta Azure Devops con la cuenta de AWS del cliente de Accenture (4.2.)** se estableció el vínculo entre ambas cuentas con un token que genera la cuenta en AWS.

El siguiente paso fue la actividad **Diseñar el Pipeline encargado de hacer el despliegue en Azure Devops (4.3.).** Aquí se realizó un YAML con los requisitos previos del cliente, añadiendo sus políticas y módulos de seguridad ya establecidos y creados para poder realizar el despliegue de la arquitectura de microservicios tipo serverless.

Finalmente, se hizo el **Despliegue de la Arquitectura de microservicios tipo serverless en Azure Devops (4.4.)**, logrando que quedara en la nube con todos los requisitos previamente mencionados.

Resultado de la fase 4: se avanzó significativamente en el despliegue de la infraestructura como código. En primer lugar, se creó una rama específica para el despliegue, como se puede observar en la Figura 48. A continuación, se procedió al despliegue de toda la infraestructura, tal como se muestra en la Figura 49. Además, se configuró y desplegó el repositorio de las lambdas, como se aprecia en la Figura 50, verificando que las lambdas se alojaran correctamente en el servicio S3, como se evidencia en la Figura 51. Por último, tras el despliegue, se observó en el servicio AWS CloudFormation las pilas generadas con todos los servicios desarrollados, como se observa en la Figura 52. Esto demuestra el éxito del proceso de despliegue y la correcta implementación de la infraestructura como código en el proyecto.

En la **Fase V. Evaluación y Pull Request** se verificó el paso a paso de las fases operacionales del diseño, realizado en contenedores a través de Sonarqube para unirlo a la rama Developer.

Para eso, se hizo la **verificación "Sonarqube" (5.1.) e**n Azure Devops de toda la infraestructura como código para revisar si cumplía con todos los estándares exigidos por el cliente de Accenture.

Luego, se hizo la actividad **Verificación de toda la arquitectura a través de Postman (5.2.)** donde se realizaron todas las pruebas correspondientes para cada método de la API con el fin de revisar si la arquitectura cumplía con todos los requerimientos previos.

Una vez la infraestructura pasó todas las pruebas del código se procedió, en compañía del supervisor organizacional, con el **Pull Request a la rama Developer (5.3),** que debió ser aceptado para lograr la unión de la arquitectura a todo el proyecto.

Resultado de la fase 5: en esta fase los logros estuvieron relacionados con las pruebas exhaustivas de la arquitectura. Primero, se revisó el código utilizando SonarQube, como se observa en la Figura 49, superando con éxito todos los filtros del análisis. Luego, se verificó el funcionamiento de la arquitectura utilizando Postman. En las figuras 60, 61, 62, 63, 64, 64, 65, 66, 67, 68, 69, 70, 71 y 72 se muestran los resultados de las pruebas realizadas para cada método del usuario y del asesor.

Se comprobó que la arquitectura respondiera de manera adecuada en todas las pruebas. Además, se creó un dashboard en Postman utilizando Newman, donde se visualiza el tiempo de respuesta, el porcentaje de éxito de cada prueba, el método utilizado y la respuesta obtenida, como se muestra en las figuras 75, 76, 77, 78, 79, 80, 81 82, 83, 84, 85, 86 y 87. Con estos resultados satisfactorios se procedió a enviar el Pull Request a la rama "developer" para integrar la arquitectura al proyecto en su totalidad.

El proceso terminó con la Fase VI. Documentación de los resultados.

Esta última etapa inició con la **Documentación final (6.1.)** necesaria para la descripción de la arquitectura y funcionamiento de la esta, y la requerida para la entrega de la pasantía.

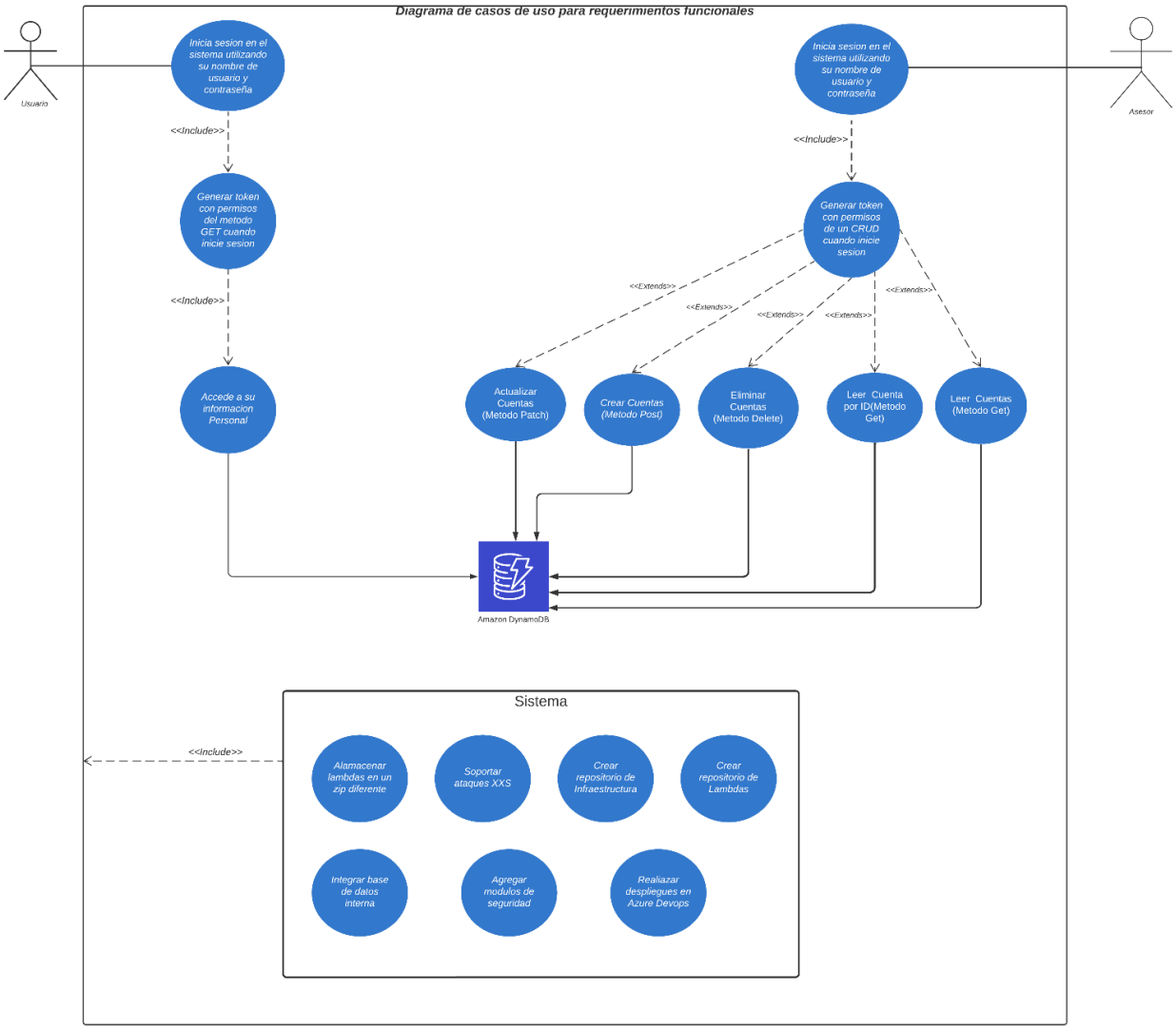
Resultados de la fase 6: se llevó a cabo la documentación exhaustiva de todos los aspectos relevantes del proyecto. Se describió detalladamente el funcionamiento de la arquitectura implementada, incluyendo cada uno de los elementos que la componen. Se proporcionó información precisa sobre las librerías utilizadas en el desarrollo, destacando su relevancia y contribución al proyecto. Además, se recopiló toda la información necesaria para la entrega final de la pasantía, garantizando una presentación completa y comprensible del trabajo realizado.

A. Definición de los Requerimientos

Según la información adquirida y analizada, el cliente, una entidad bancaria de Centroamérica, pidió construir una arquitectura de microservicios tipo serveless en AWS, realizando el despliegue en Azure Devops y cumpliendo con los requerimientos que pueden ser consultados en Anexos la Tabla 1 y Tabla 2.

En función de dichos requerimientos se buscaron las soluciones, las cuales se relacionan en la Figura 22.

FIGURA 22. REQUERIMIENTOS Y SOLUCIONES.



Fuente: [69]

V. **DESARROLLO**

A continuación, en la Figura 23 se presenta un diagrama que resume el trabajo realizado en la pasantía y que será explicado en este apartado del documento.

El objetivo principal del presente trabajo de práctica profesional fue desarrollar una arquitectura serverless en AWS, orientada a microservicios, que se caracteriza porque no se necesita administrar servidores, ya que AWS se encarga de la gestión de los servidores subyacentes, la escalabilidad automática y el balanceo de carga. Los microservicios se ejecutan en función de la demanda y se pagan únicamente por el tiempo que se ejecutan.

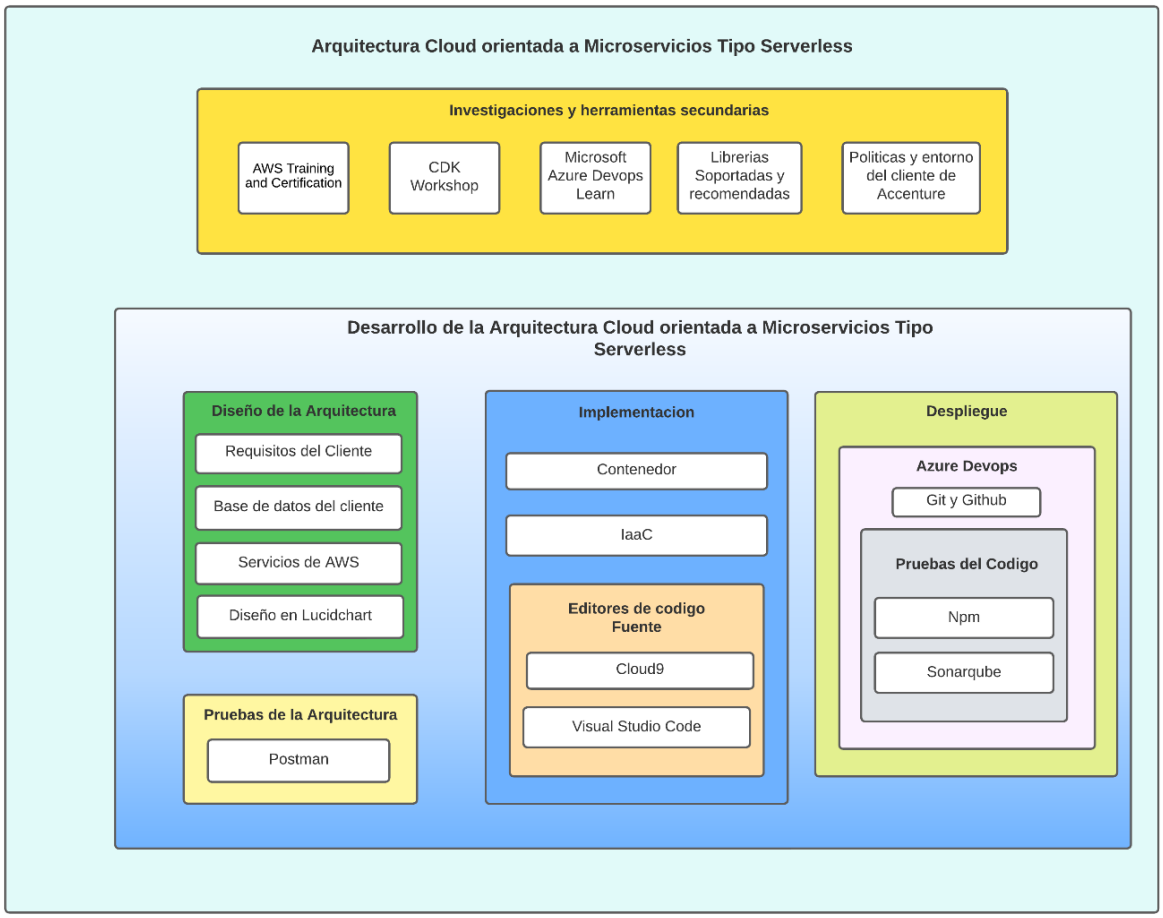
También, tiene como ventaja la flexibilidad, es decir, los microservicios se pueden desarrollar y desplegar de forma independiente para un desarrollo y actualización de la aplicación más flexibles.

Así mismo, la aplicación tiene mayor eficiencia porque cada microservicio realiza una tarea específica y optimizada para su rendimiento. Además, cada microservicio tiene alta disponibilidad porque están distribuidos en diferentes zonas y se ejecutan según la función que se demande.

Ahora bien, en la arquitectura desarrollada en la pasantía se usaron 11 servicios de AWS, está orientada a microservicios y tiene dos funciones: usuario y asesor. Ninguno de los servicios tiene servidor, requerimiento necesario porque estos se deben integrar al desarrollo total del proyecto, que es de gran tamaño y en su construcción intervinieron más células de trabajo.

La arquitectura se albergó en Azure Devops en un repositorio creado por el cliente. Los editores de código usados para la infraestructura fueron Cloud9 y Visual Studio Code.

FIGURA 23. DIAGRAMA DE FLUJO DE LA PASANTÍA.



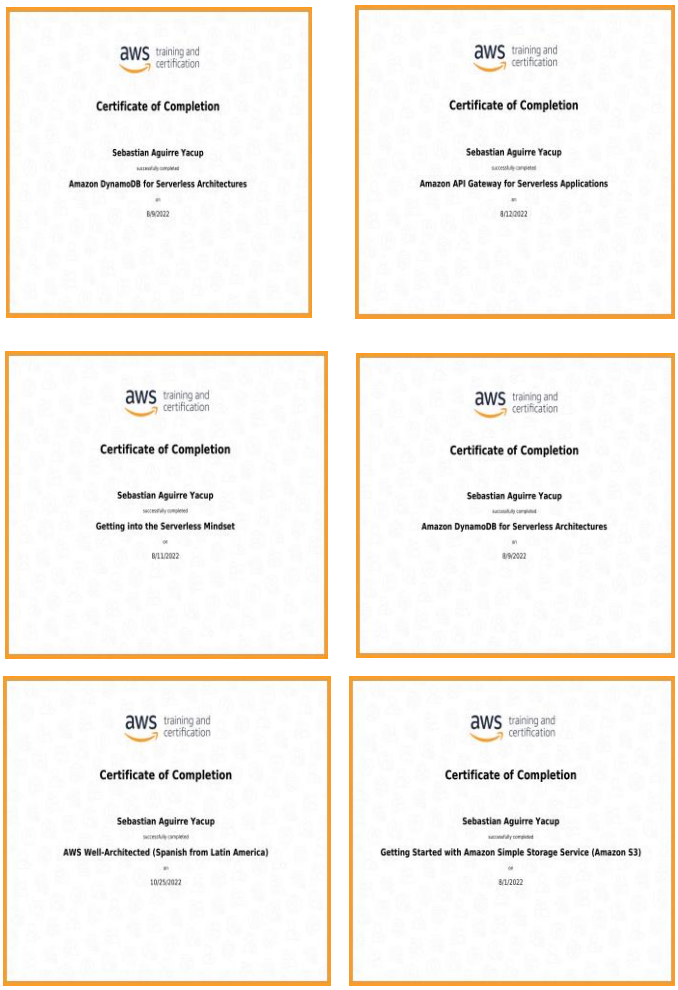
Fuente: [69]

A. Investigaciones y herramientas secundarias

Descripcion del trabajo: esta tarea consistió en investigar sobre las herramientas útiles para el desarrollo de la arquitectura cloud en AWS, las cuales eran perninentes para el proyecto. A su vez, con los resultados de esta pesquisa se estudiaron varios servicios que tienen AWS,con el fin de evaluar cuál era óptimo en la arquitectura y luego presentar el diseño al cliente.

AWS Training & Certifications: se usó la plataforma AWS Training & Certifications porque tiene un repertorio con más de 500 cursos online gratutitos, orientados por expertos en AWS. Allí se adquirieron conocimientos del entorno de AWS, consultar Figura 24, todos los tipos de arquitecturas y servicios, logrando identificar las debilidades y ventajas de cada uno.

FIGURA 24. CERTIFICACIONES DE ENTRENAMIENTO.



Fuente: [69]

CDK WORKSHOP: esta es una plataforma de AWS y gratituta, que permitió el aprendizaje para llevar una arquitectura a una infrastructura como código (IaaC), mediante ejemplos y talleres del proceso a travás de diferentes lenguajes.

Microsoft Azure Devops Learn: es una plataforma que explica, a través de diferentes módulos, todo el entorno de Azure Devops, permite entender a profundidad cada una de las opciones que tiene. Allí se tiene acceso a workshops o talleres.

Investigacion de librerías y módulos: se investigaron las librerías y módulos soportados en CDK con el lenguaje que pidió el cliente (Typescript), que cumpliera sus requirimientos y con el diseño que se estableció. Luego, se eligieron las librerías y módulos que se aprecian en la Figura 25 y que se iban a utilizar para facilitar el desarrollo de la infrastructura como código.

FIGURA 25. LIBRERÍAS.

```
import { Stack, StackProps, Duration } from 'aws-cdk-lib';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import * as apigateway from 'aws-cdk-lib/aws-apigateway';
import * as apigatewayv2_authorizers from '@aws-cdk/aws-apigatewayv2-authorizers';
import * as dynamodb from "aws-cdk-lib/aws-dynamodb";
import * as lambda from "aws-cdk-lib/aws-lambda";
import * as s3 from 'aws-cdk-lib/aws-s3';
import { CloudFrontToS3 } from '@aws-solutions-constructs/aws-cloudfront-s3';
import { WafwebaclToCloudFront } from "@aws-solutions-constructs/aws-wafwebacl-cloudfront";
```

Fuente: [69]

Librería " 'aws-cdk-lib'": es una librería de construcción de AWS CDK, ofrece módulos que se pueden usar para modelar los recursos proporcionados por servicios de AWS [70].

Libería "aws-cdk-lib/aws-cognito": esta librería se utiliza para la creación y configuración del servicio de AWS Cognito [71].

Librería "aws-cdk-lib/aws-apigateway'": es una librería que permite crear y configurar todo tipo de API dentro de AWS [72].

Librería "aws-cdk/aws-apigatewayv2-authorizers": permite crear y configurar la API, controlarla y administrarla. Se clasifica principalmente en autorizadores de Lambda, roles y políticas estándar de AWS IAM [73].

Librería "aws-cdk-lib/aws-dynamodb": con esta se crea y configura la base de datos DynamoDB alojada en AWS.

Librería "aws-cdk-lib/aws-lambda": es una librería de construcciones que permite crear y configurar funciones de AWS Lambda [74].

Librería "aws-cdk-lib/aws-S3": permite crear y configurar el servicio de AWS S3.

Módulo "aws-solutions-constructs/aws-wafwebacl-cloudfront": esta construcción de soluciones de AWS implementa un servicio AWS WAF, conectado a Amazon CloudFront [75].

Modulo "aws-solutions-constructs/aws-cloudfront-s3": esta construcción de soluciones de AWS implementa un servicio de AWS CloudFront conectado a un de Bucket AWS.

B. Inducción de las Políticas y Entorno del Cliente de Accenture.

La inducción se llevó a cabo con los cursos establecidos por el cliente de Accenture, lo que permitió conocer todas las políticas que se exigen el momento de crear código y arquitectura.

También, se conocieron todos los entornos y cómo son usados en las plataformas de AWS y en Azure Devops.

C. Diseño de la Arquitectura

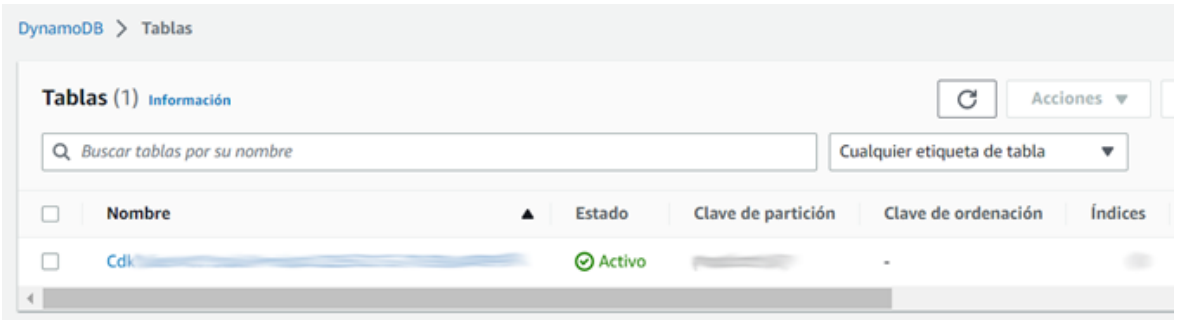
Descripcion del trabajo: se diseñó la arquitectura Cloud oritenda a microservicios tipo serverless con todos los requisitos previos, exigidos por el cliente y por el tutor organizacional; se escogieron los mejores servicios de AWS que se pudieran acoplar a la arquitectura.

Requisitos del cliente: primero establecimos los requisitos del cliente como se muestran en la tabla de requerimientos del cliente (ver en Anexos las tablas 1 y 2).

Luego, se eligieron los mejores servicios de AWS que cumplieran los requisitos previamente mencionados.

D. Base de Datos del Cliente

FIGURA 26. BASE DE DATOS DYNAMOBD.



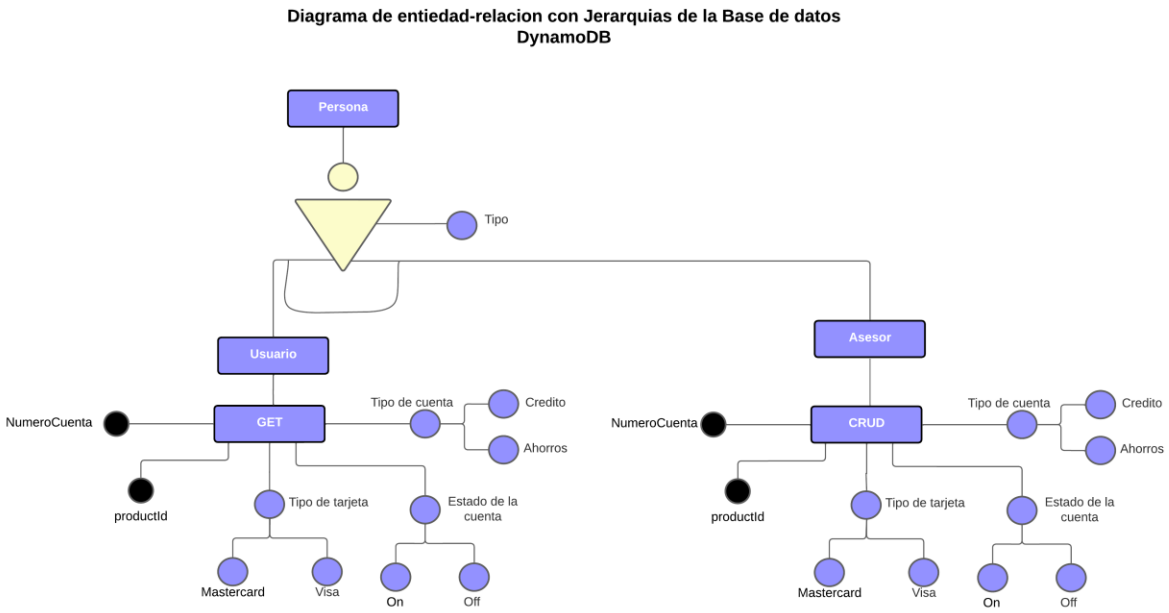
Fuente: [69]

Esta base de datos se encuentra en AWS en el servicio de DynamoDB, no tiene servidor y es completamente administrada por AWS (ver Figura 26), que se encarga del buen rendimiento y la escalabilidad horizontal y vertical.

DynamoDB está diseñada para ejecutar aplicaciones de alto rendimiento a cualquier escala, ya que ofrece seguridad integrada, copias de seguridad continuas, replicación automatizada en varias regiones, almacenamiento de caché en memoria y herramientas de importación y exportación de datos [76].

Una vez identificada la base datos del cliente y sus características se procedió a buscar el END-POINT de la BD, que se necesitaría para poder integrarla a la arquitectura. A continuación, se presenta un diagrama entidad-relación que le permitirá al lector comprender el funcionamiento de la BD, este se puede consultar en la Figura 27.

FIGURA 27. ENTIDAD-RELACIÓN BASE DE DATOS.



Fuente: [69]

E. Diseño de la Arquitectura Orientada a Microservicios Tipo Serverless

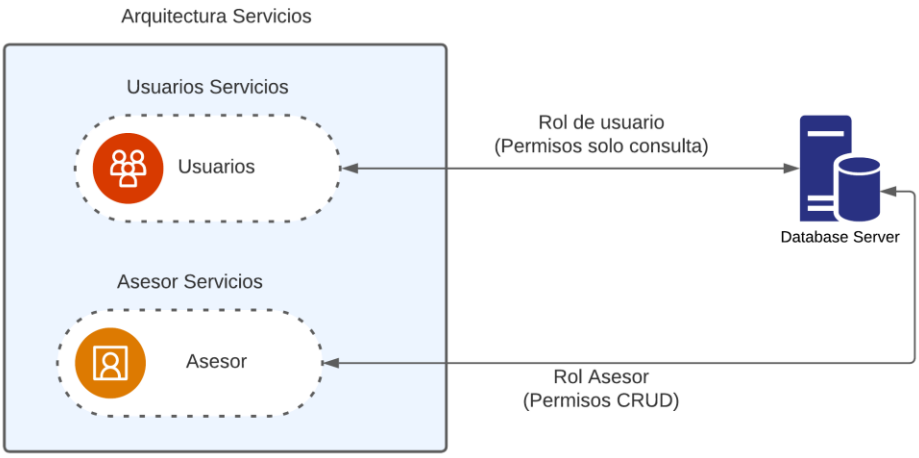
Debido a que este es un mega proyecto, uno de los requisitos del cliente fue que cada arquitectura estuviera orientada a microservicios porque se debe integrar con las arquitecturas de los demás equipos o células con el fin de no generar conflictos ni bugs. Cuando se identificó esa solicitud del cliente, se procedió a realizar la arquitectura con dos servicios.

El servicio de consulta para usuarios (ver Anexos Tabla 1).

El servicio de permisos CRUD para el asesor (ver Anexos Tabla 1).

Estos servicios deben ser diferentes por su rol y por sus permisos, pero deben estar integrados en la misma arquitectura como se ilustra en la Figura 28.

FIGURA 28. ARQUITECTURA Y SERVICIOS.



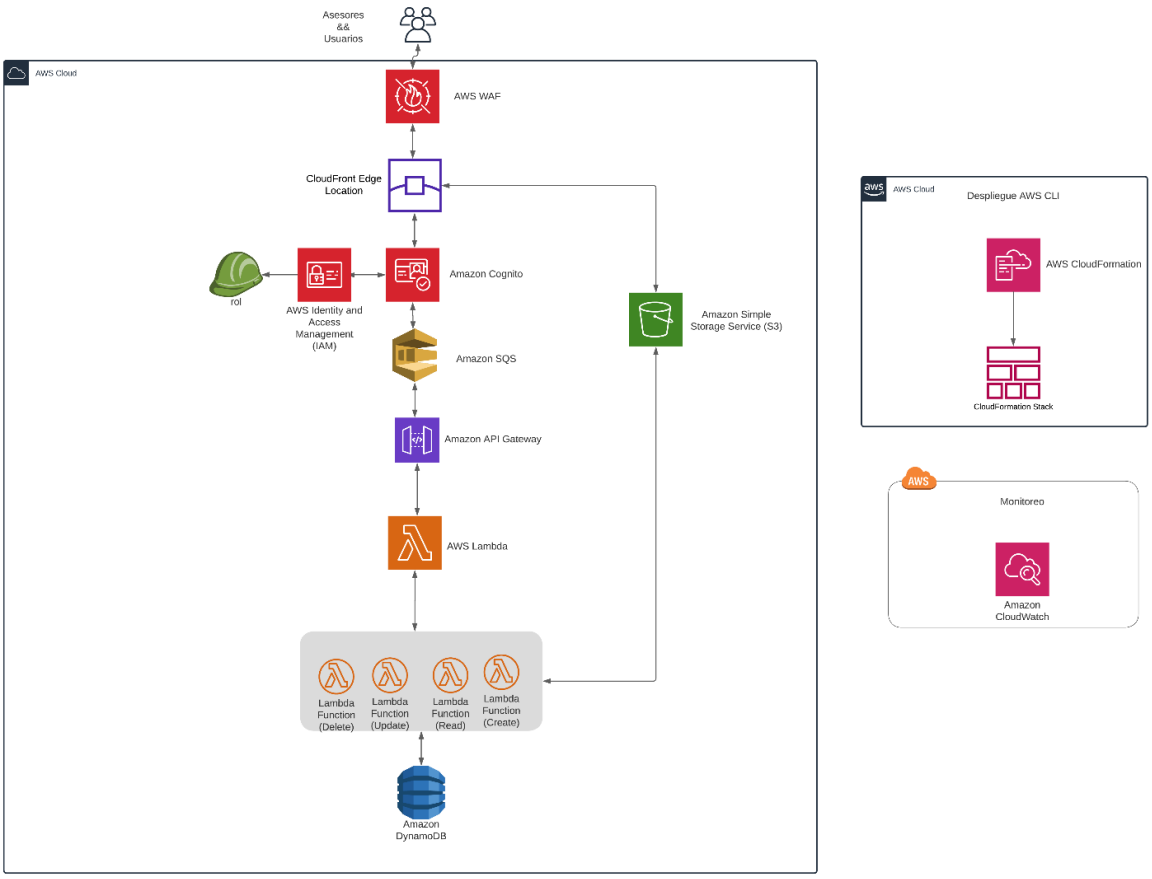
Fuente: [69]

Dentro de esta arquitectura los servicios se crean para las capacidades empresariales y cada uno de ellos desempeña una sola función. Esto se debe a que se ejecutan de forma independiente para que cada uno pueda actualizar, implementar y escalar; satisfaciendo la demanda de funciones específicas de una aplicación [77].

Cabe anotar que, este tipo de arquitectura tiene una ventaja, cuando se genera un error en un servicio, toda la arquitectura seguiría funcionando. De ahí que, el cliente la eligiera, ya que en un futuro será más fácil aplicar cambios.

Ahora bien, cuando ya se había identificado la arquitectura de microservicios se procedió a buscar los servicios con los que se iba a diseñar. Dicha búsqueda debió tener en cuenta los requerimientos del cliente (ver Anexos Tabla 1 y 2), es decir, que sea tipo serverless o sin servidor, por eso se todos los servicios que se aplicaron cumplían con dicha característica, como se aprecia en la Figura 29.

FIGURA 29. DISEÑO DE ARQUITECTURA (ROL ASESOR).



Fuente: [69]

Este diseño fue aprobado por el tutor organizacional del pasante y por el cliente de Accenture. Consta de 11 servicios de AWS y dos funciones (usuario y a asesor). Los servicios que se escogieron fueron:

AWS WAFF: se utiliza en la arquitectura para prevenir ataques XXS, ya que el cliente exigió que la arquitectura debe llevarlo por políticas de seguridad.

AWS Cloudfront: permite configurar los Edge Location para dar respuestas más rápidas. Este servicio se implementa debido a que el cliente requiere ubicaciones al borde para disminuir la latencia cuando la persona ingrese a la plataforma. Este se conecta con el Frontend ya diseñado en angular por otra célula o equipo de trabajo.

AWS Cognito: identifica a través de un login si es usuario o es asesor. Se aplica con el fin de crear dos roles uno para usuarios y otro para asesores, así, cada uno tendrá un permiso especial. El login al que se conecta es el establecido por el cliente de ACCENTURE. Una vez el asesor o el usuario ingrese generará un token, el cual tendrá los permisos según el rol de la persona que accedió.

AWS SQS: este servicio se incluyó por dos requisitos, el primero es que en caso de presentarse alta demanda puedan llegar todas las peticiones de manera ordenada y no se pierda ninguna; el segundo tiene que ver con las políticas de seguridad que cifran el mensaje.

AWS API Gateway: permite desarrollar los cuatro 4 métodos diferentes de un CRUD para el servicio de AWS Lambdas. Su funcionalidad es hacer de "puerta principal" para que los métodos accedan a los datos que se encuentran en AWS DynamoDB.

AWS S3: almacena las lambdas en un .ZIP para que no haya conflicto cuando se requiera configurar cada método. Así mismo, en este bucket o S3 se almacenan las configuraciones de Cloudfront.

AWS Lambdas: es el encargado de hacer los cuatro métodos de un CRUD para el asesor o solo el método get para el usuario. Por petición del cliente, estas lambdas se programaron en el lenguaje Python y se desplegaron en un repositorio diferente.

AWS DynamoDB: es donde el cliente de Accenture almacena todos los datos de los usuarios.

AWS Cloudformation: se crearon dos stacks, el primero almacena la configuración de los permisos de todos los servicios y el segundo guarda la configuración de estos en el momento de hacer el despliegue. De esta manera, cuando se realice un cambio solo se introduce el comando "cdk deploy" y se actualizará el servicio que fue modificado sin afectar los demás.

AWS CloudWatch: este servicio nos permite ver las fallas o cambios, a través de unos logs que genera cada servicio.

La arquitectura fue diseñada para que funcione con todos los requisitos previamente mencionados por el cliente. Es por eso que se les dio prioridad a los servicios de AWS tipo serverless. Así, el cliente

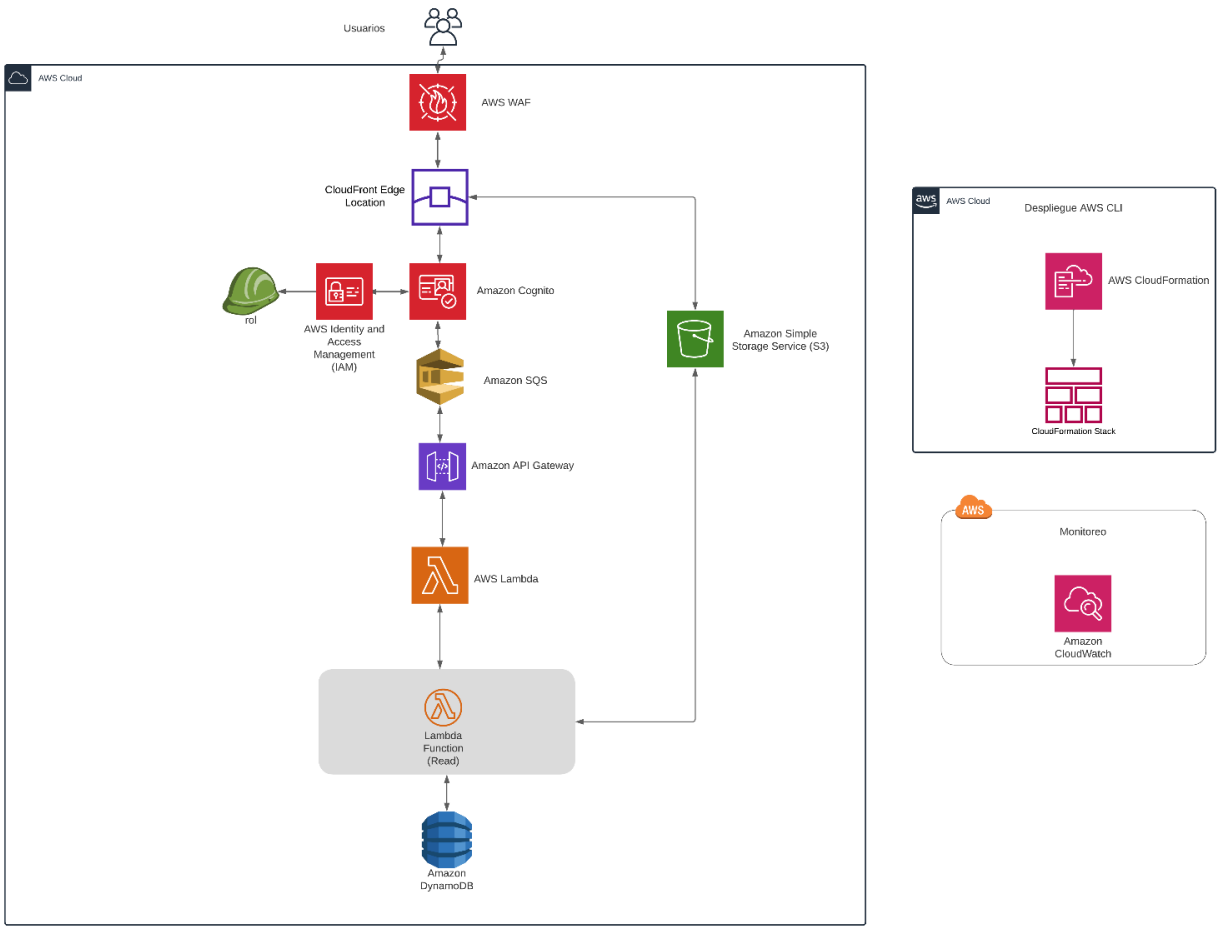
no tendrá que preocuparse por la administración del servicio y se enfocará en ingresar información para los asesores o mostrar lo que el usuario requiere.

Al ser una arquitectura sin servidor, el cliente no tendrá que preocuparse si en algún momento los servicios o la arquitectura en general tienen una alta demanda, ya que en su momento AWS de manera automática dispondrá más recursos de manera horizontal para que no colapse ni aumente el tiempo de respuesta. Así mismo, cuando haya menos influencia reducirá la capacidad para disminuir los precios; recordemos que en AWS se paga por uso.

En cuanto el almacenamiento de la base datos, el asesor no deberá agregar más información porque cuando se requiera más almacenamiento automáticamente AWS asignará mayor capacidad.

En relación con las funciones que cumplirá la arquitectura, estas se relacionan con el asesor que tendrá todos los permisos como un CRUD (leer, actualizar, crear, eliminar) y un usuario que solo podrá leer su información, como se muestra en la Figura 30.

FIGURA 30. FUNCIONES DE LA ARQUITECTURA (ROL CLIENTE).



Fuente: [69]

Al ser dos roles distintos, tanto el usuario como el asesor generan un token diferente, que tendrá permisos especiales según cada caso. Al primero, el token solo le otorgará permiso para ver la información, como se muestra en la tabla de requerimientos funcionales en el código RF01.

En cambio, al asesor el token le dará permisos de un CRUD, como se ve en la tabla de requerimientos funcionales en los códigos RF02 y RF04.

F. Implementación

Descripcion del trabajo: esta tarea consitió en realizar toda la infraestructura como código (IaaC) de la arquitectura diseñada con el objetivo de cumplir los requisitos y servicios previamente mencionados.

La arquitectura fue implementada a través de CDK (“AWS Cloud Development Kit”), es decir, infraestructura como código. El cliente solicitó en los requerimientos que la infraestructura se realizara con el lenguaje Typescript para acoplar la arquitectura al proyecto sin generar conflictos.

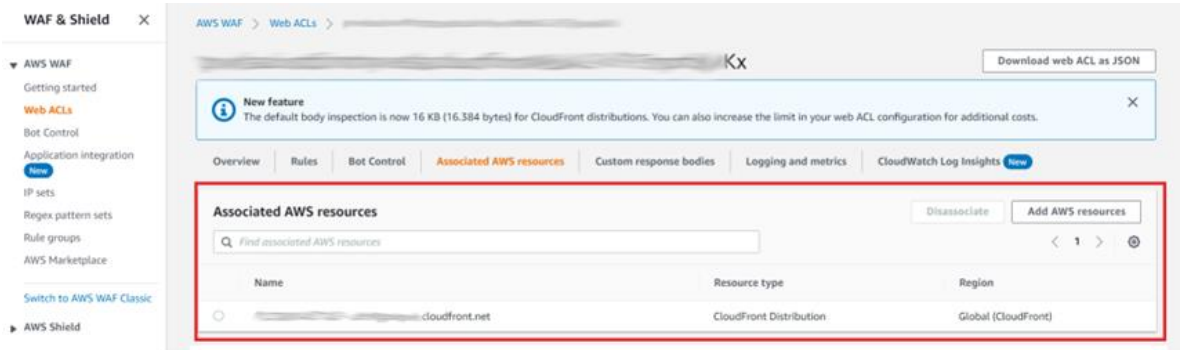
El desarrollo se hizo en el servicio de AWS Cloud9, creando dos entornos: uno para toda la infraestructura y otro para las lambdas, como se aprecia en la Figura 31.

FIGURA 31. DESARROLLO DE SERVICIO EN AWS CLOUD9.

Fuente: [69]

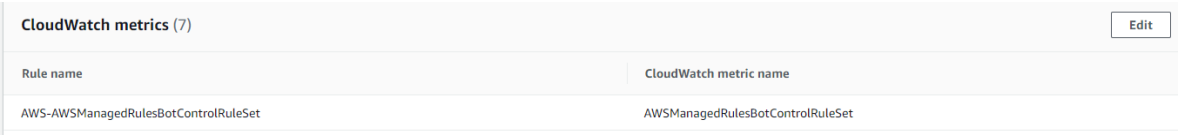
Los primeros dos servicios se crearon para el desarrollo en la infraestructura como código fueron: AWS WAF y Cloudfront, ver Figura 32, con la ayuda del módulo “**aws-solutions-constructs/aws-wafwebacl-cloudfront”.** Estos se configuraron para que se acoplaran y cuando se recibieran ataques XXS la arquitectura los soportara, como se muestra en la Figura 33. De esta manera, se dio cumplimiento a uno de los requisitos del cliente.

FIGURA 32. CREACIÓN DE CLOUDFRONT.



Fuente: [69]

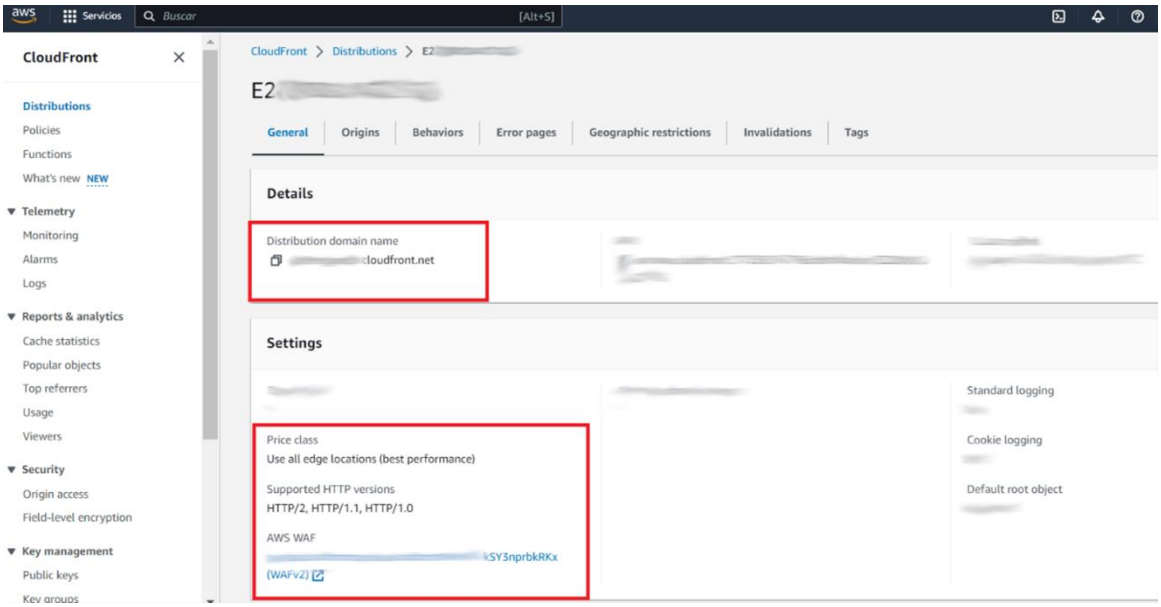
FIGURA 33. CONFIGURACIÓN DE WAFF PARA ATAQUES XXS



Fuente: [69]

Luego, se hizo la configuración de cloudfront, como se observa en la Figura 34, en la infraestructura como código. En esta fase se conectó el Frontend ya diseñado por otro equipo o célula del trabajo y, por último, se configuraron las ubicaciones al borde para que el tiempo de respuesta sea menor cuando un usuario o asesor ingrese; esto se consigue porque el servicio buscará en cualquier ubicación donde AWS tenga un servidor y le enviará la petición. Igualmente, dentro del cloudfront se incorporó el WAFF para prevenir ataques XXS.

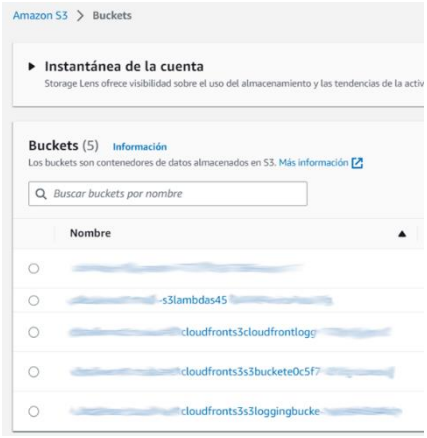
FIGURA 34. CONFIGURACIÓN DE CLOUDFRONT



Fuente: [69]

Para configurar el cloudfront se desarrolló el servicio AWS S3 en IaaC, lo cual se llevó a cabo con ayuda de la librería "**aws-cdk-lib/aws-S3**". Este servicio es el encargado de alojar los métodos de las lambdas como la configuración del frontend con cloudfront, como se aprecia en la Figura 35.

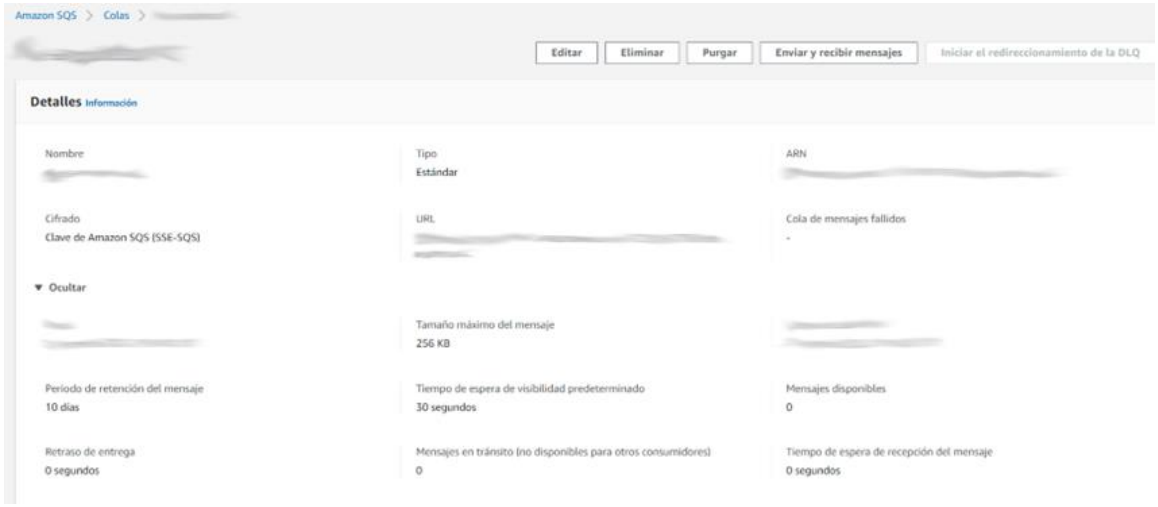
FIGURA 35. S3 ALMACENA LAS LAMBDAS Y EL CLOUDFRONT.



Fuente: [69]

El cuarto servicio que se configuró en la infraestructura como código fue el SQS, cuya función es evitar la pérdida de peticiones cuando haya bastante flujo o demasiada demanda, de manera que lleguen todas al asesor o el usuario. En la Figura 36 se aprecia que se configuró con retraso de entrega de 0 segundos para que sea lo más rápido posible. Así mismo, por solicitud del cliente, se le añadió un periodo de retención de 10 días en caso de que el mensaje o la petición se deba recuperar o deshacer los cambios. Por último, se configuró con el cifrado que viene con el servicio por políticas del cliente de Accenture.

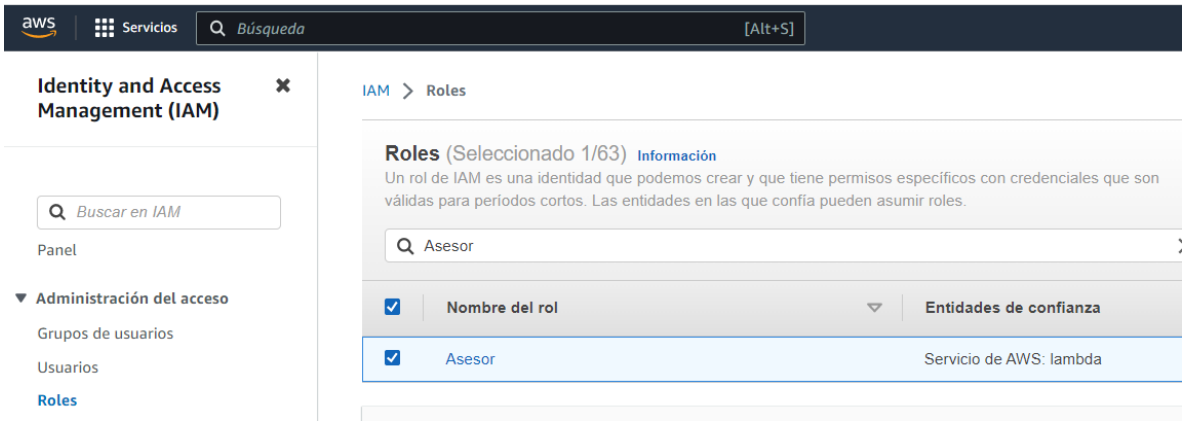
FIGURA 36. CONFIGURACIÓN DE SQS.



Fuente: [69]

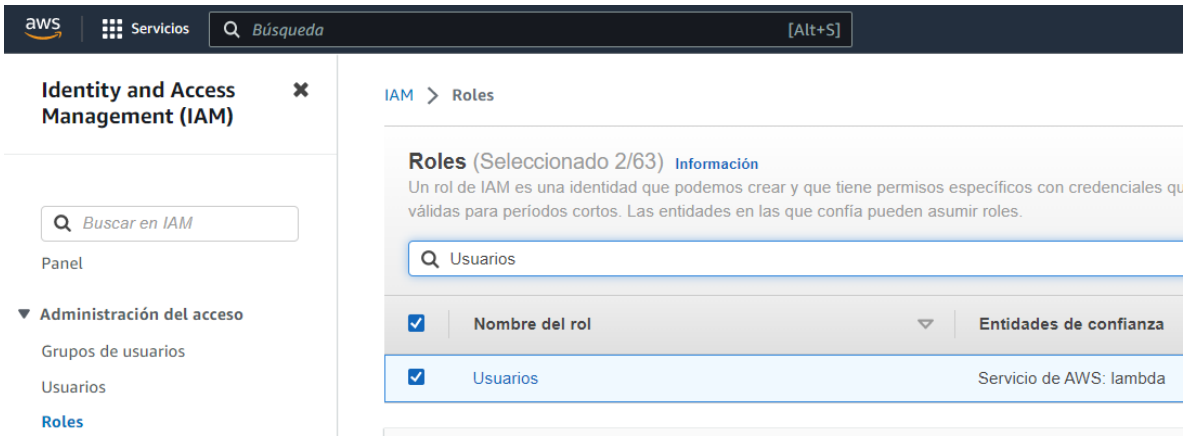
El quinto servicio que se configuró fue AWS IAM con el fin de crear dos roles (Figura 37 y 38) con políticas distintas y asignárselas al cognito (usuario y asesor).

FIGURA 37. CREACIÓN Y CONFIGURACIÓN DEL ROL DE ASESOR.



Fuente: [69]

FIGURA 38. CREACIÓN Y CONFIGURACIÓN DEL ROL DE USUARIO.

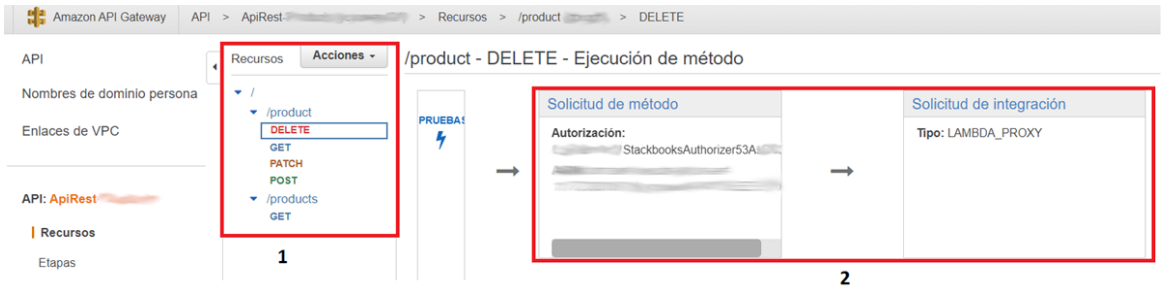


Fuente: [69]

El sexto servicio de AWS que se configuró fue API Gateway (Figura 35), a través de la infraestructura como código. Esto permitió enlazar cada método de API con su función lambda, que es la que admite la petición a la base de datos. En la Figura 39, en el cuadro 1, se observan cuatro métodos y dos recursos que son: *product* y *products.*

En el recurso *product* hay cuatro métodos, todos corresponden a un crud para hacer las peticiones de crear, actualizar, eliminar y leer los datos en DynamoDB que es la base de datos del cliente. En *products* hay un get que se hará cargo cuando el asesor necesite buscar a través del ID alguna información en la base de datos o si requiere que le muestre toda la información que está almacenada.

FIGURA 39. CONFIGURACIÓN DE LA API.



Fuente: [69]

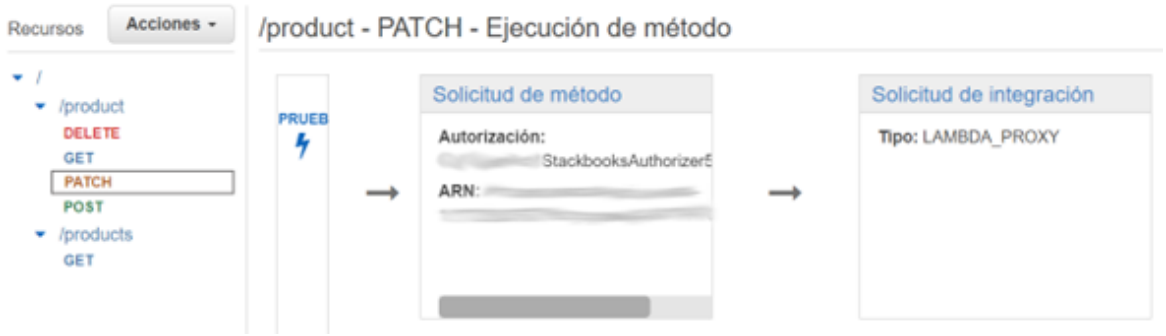
En el cuadro número dos, pintado de rojo, se observa que cada método está asociado a una autorización, que, mediante la infraestructura como código, se conecta con el cognito para identificar la persona que ingresa y que esta tenga los permisos para hacer las peticiones; es decir, si accede un asesor podrá usar todos los métodos y recursos, pero si inicia sesión un usuario solo tendrá el permiso de un método *get*, que es para que le muestre únicamente su información.

En las figuras 40, 41, 42 y 43 se ve que cada uno de estos métodos tiene conectado el servicio de cognito, es decir, la autorización y el servicio de lambdas. Cada lambda va a ser diferente para cada método; esta es una petición del cliente para que cuando haya cambios, actualizaciones o fallas solo deje de funcionar el método que está afectado y los demás sigan trabajando.

FIGURA 40. MÉTODO GET(PRODUCT)

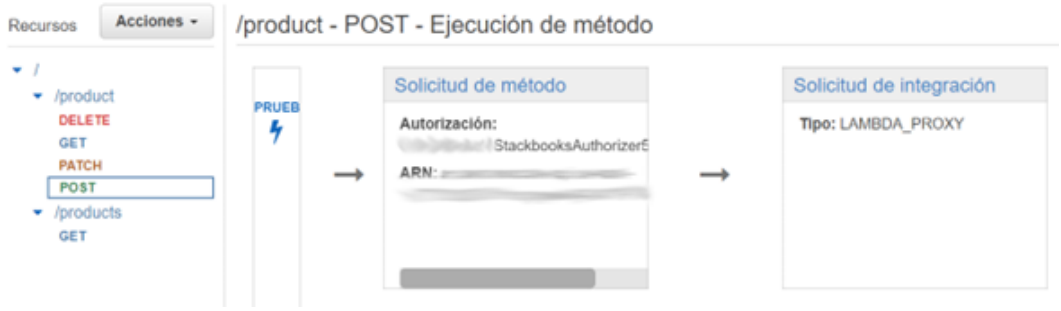
Fuente: [69]

FIGURA 41. MÉTODO PATCH.



Fuente: [69]

FIGURA 42. MÉTODO POST.



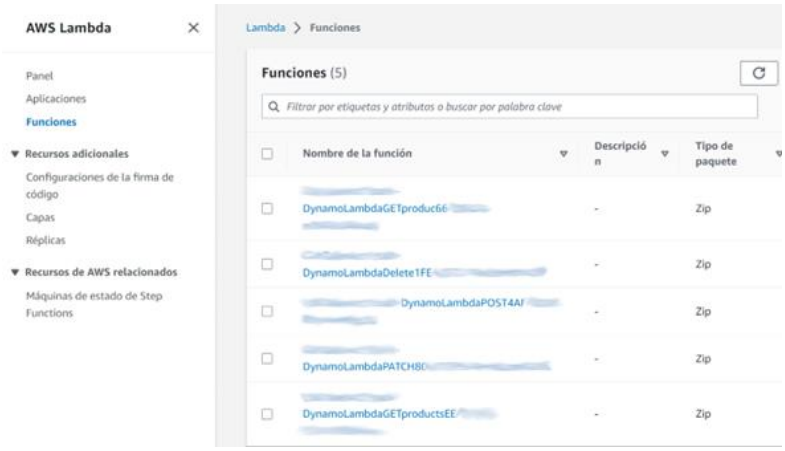
Fuente: [69]

FIGURA 43. MÉTODO GET(PRODUCTS).

Fuente: [69]

Por último, a través de la infraestructura como código se procedió a crear el servicio de AWS Lambda (ver Figura 44). Este es el encargado de llevar un código realizado en Python para hacer las consultas en la base de datos. Es por eso por lo que se hacen cinco lambdas para cada método de API, como se ve en la siguiente imagen.

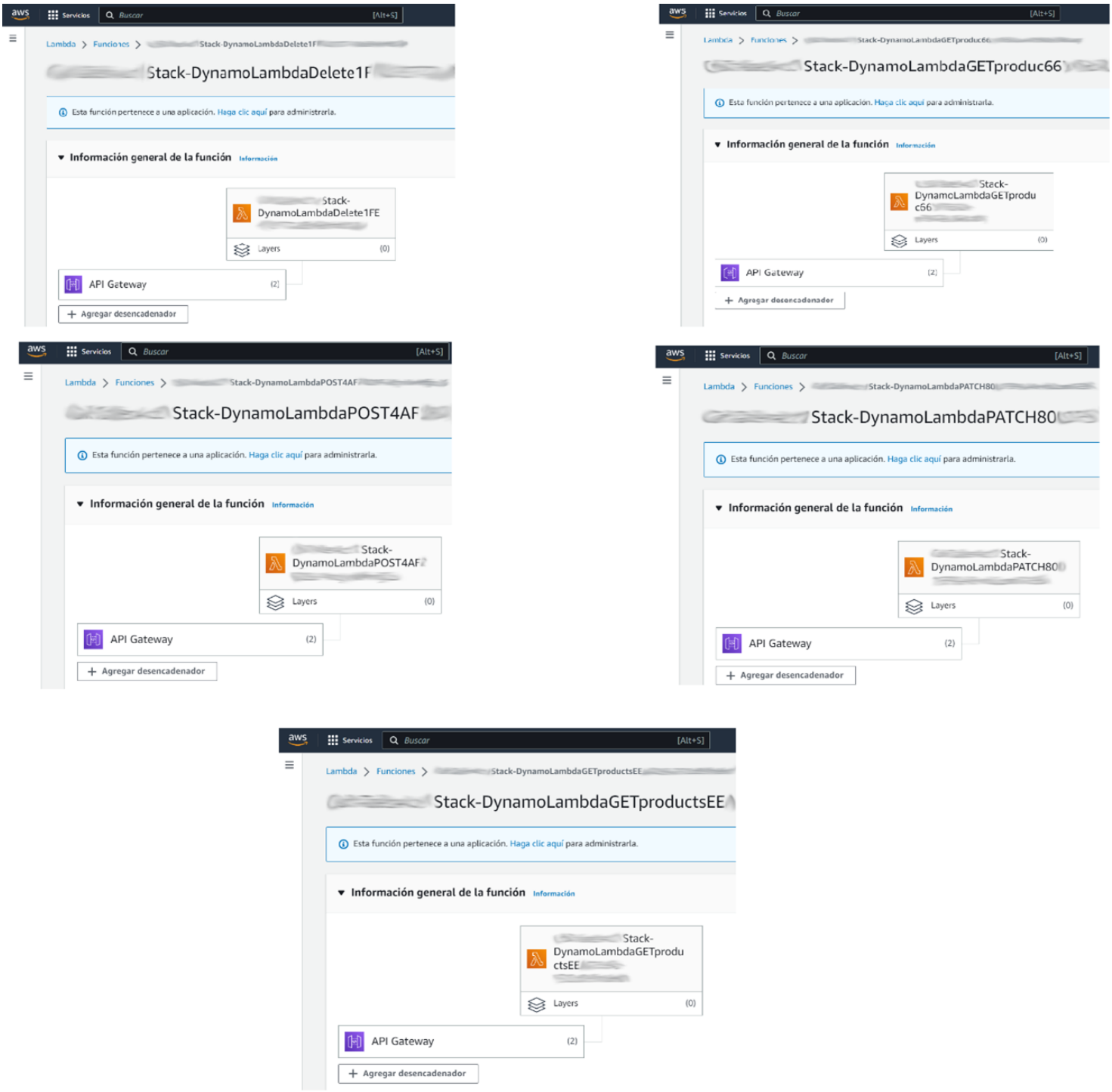
FIGURA 44. CREACIÓN DEL SERVICIO AWS LAMBDA.



Fuente: [69]

Cada una de estas funciones lambdas está en un archivo .zip como lo exigió el cliente en los requerimientos funcionales RF05. Estas lambdas, cada una en la infraestructura como código, se configuraron para que estén asociadas a su API con el fin de hacer las peticiones a la base de datos del cliente; esto se observa en la Figura 45.

FIGURA 45. LAMBDAS ASOCIADAS A LAS API.



Fuente: [69]

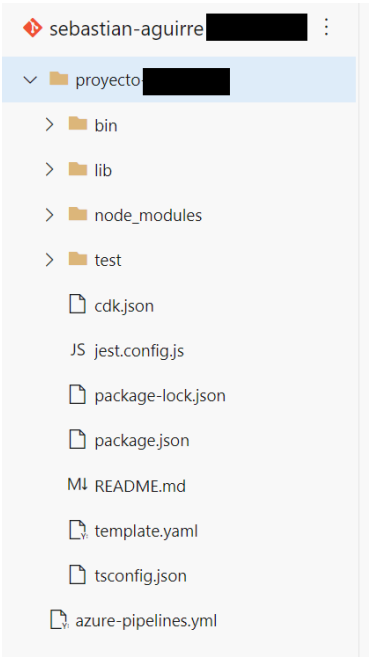
Luego de asociar las lambdas a las API y programarlas en Phyton, se creó un contenedor donde se empaquetó toda la infraestructura como código y se procedió a hacer el despliegue de la IaaC con todos los servicios del diseño; esto para ver si cumplía con todas las pruebas exigidas por el cliente de Accenture.

G. Despliegue y pruebas del código

Descripcion del trabajo: esta tarea incluyó todo el despliege y pruebas de la infraestructura como código en la plataforma *Azure Devops*, generando stacks en *Cloudformation* de los servicios que se implmentaron en AWS y pasando todas las pruebas de código solicitadas por el cliente de Accenture.

Todo este proceso se realizó en *Azure Devops*, plataforma donde el cliente hace los despliegues por petición. El primer paso fue crear un repositorio donde se albergó el container con toda la infraestructura como código (ver la Figura 46).

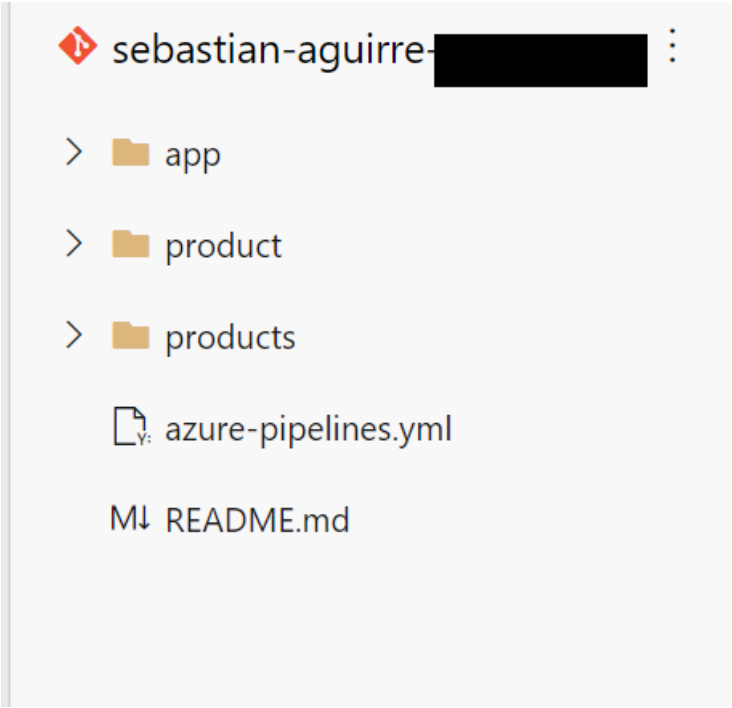
FIGURA 46. CONTAINER PARA ALMACENAR LA INFORMACIÓN.



Fuente: [69]

Por otro lado, se creó otro repositorio para las lambdas (Figura 47), por petición del cliente, como se referencia en la tabla de requerimientos con el código RF08.

FIGURA 47. REPOSITORIO PARA LAMBDAS.



Fuente: [69]

Después de subir los dos repositorios, uno con la infrastructura y otro con las lambdas, se creó la rama con un comando de *git*, consultar Figura 48, cumpliendo con todas las políticas que exijió el cliente para preparar el despliegue.

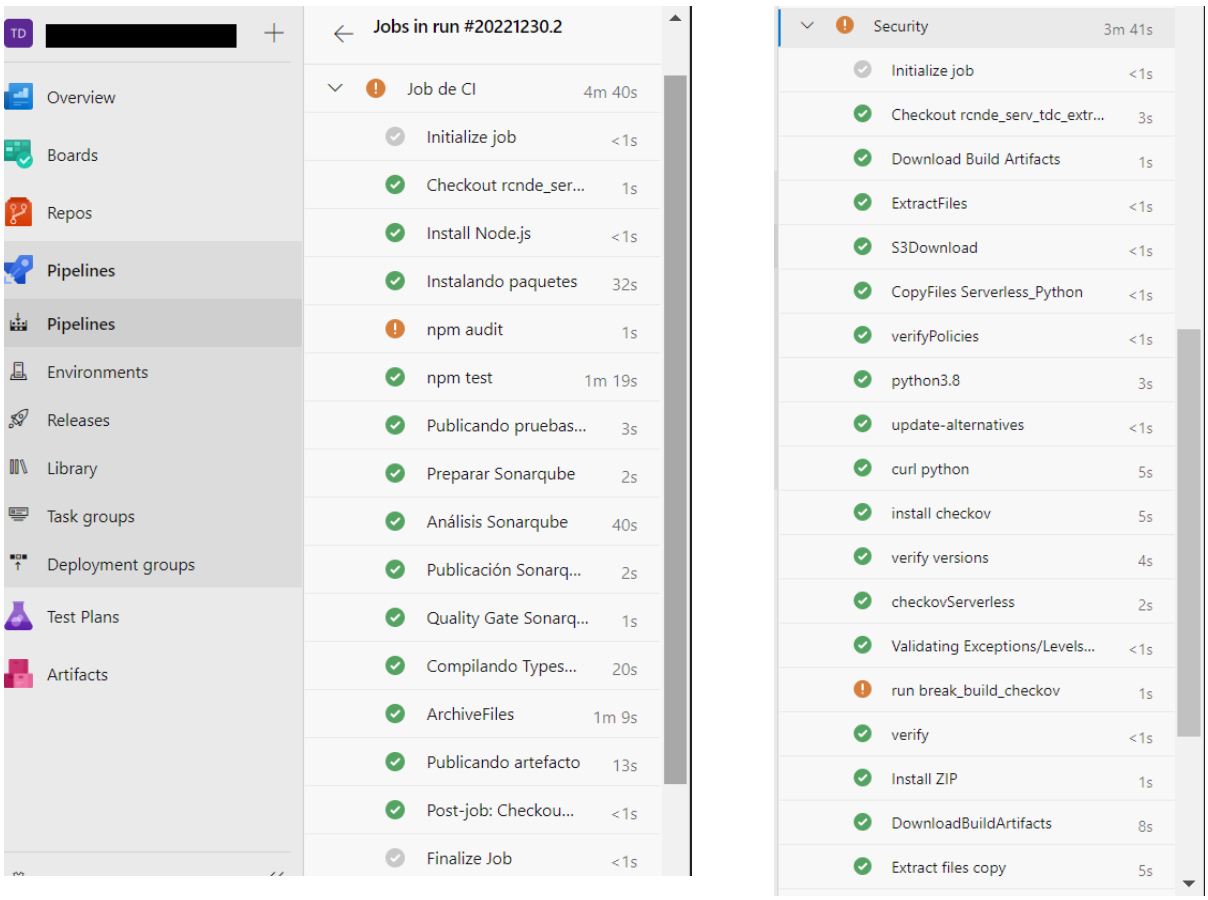
FIGURA 48. CREACIÓN DE RAMA.

Fuente: [69]

Después de crear la rama se configuró el YAML de Azure para hacer el despliegue de la infraestructura, el cual se ve en la Figura 49. Dentro de este YAML se integraron los artefactos que luego le hicieron las pruebas al código de la infrastructura para cumplir con todos los requerimientos de seguridad y políticas del cliente de Accenture.

Posteriormente, se hizo el despliegue para ver si superaba todas las pruebas del código para la infraestructura.

FIGURA 49. DESPLIEGUE EN AZURE DEVOPS.

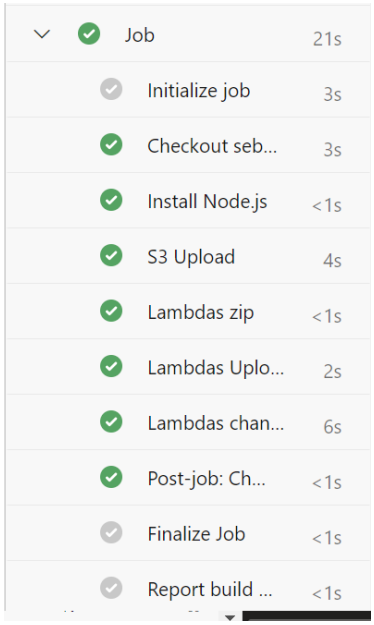


Fuente: [69]

Después de que el código de la infraestructura pasó las dos pruebas de código “npm test”, “Análisis Sonarqube” y las políticas de seguridad del cliente de Accenture, se procedió a configurar el YAML

(ver Figura 50) para realizar el despliegue de las lambdas y su integración al código de Python para hacer las peticiones y cumplir con otro requerimiento funcional (RF05).

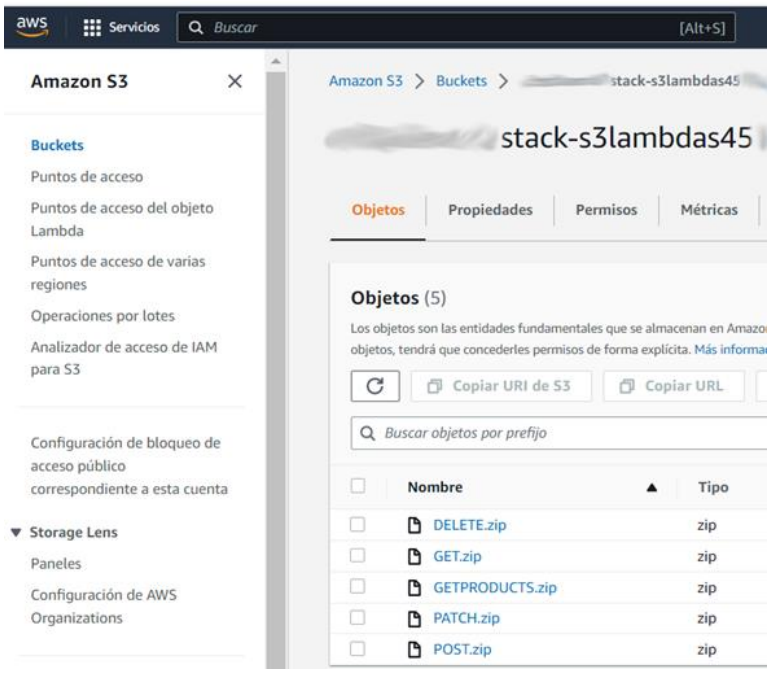
FIGURA 50. CONFIGURACIÓN DE YAML.



Fuente:[69]

El siguiente paso fue comprobar si dentro del *bucket* de S3 se encontraban los cinco métodos lambdas en un .ZIP, proceso que se observa en la Figura 51.

FIGURA 51. COMPROBACIÓN MÉTODOS LAMBDA.

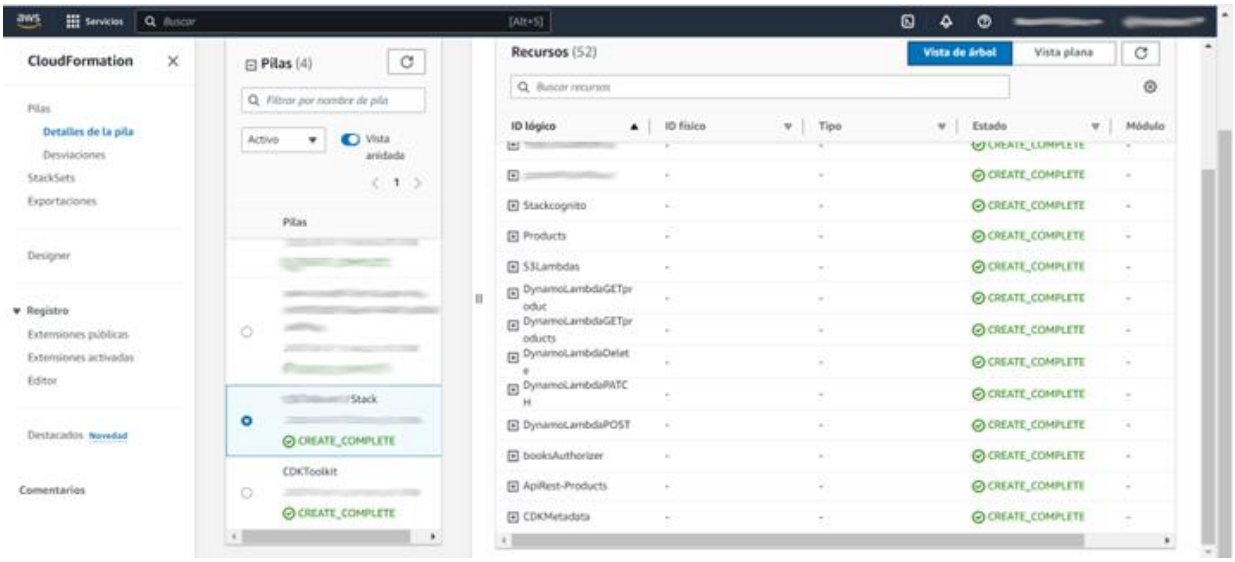


Fuente: [69]

Posterior a ello, se revisó que en el servicio de AWS Cloudformation se desplegaran dos *stacks*, uno para todos los recursos de la infraestructura, como se puede ver en la Figura 52, y otro para los permisos de cada rol y servicios que se utilizaron en la arquitectura.

En el *stack* de la infraestructura se puede ver que todos los servicios se desplegaron correctamente. Dentro de cloudformation, en la pestaña de recursos, se creó cada servicio de AWS.

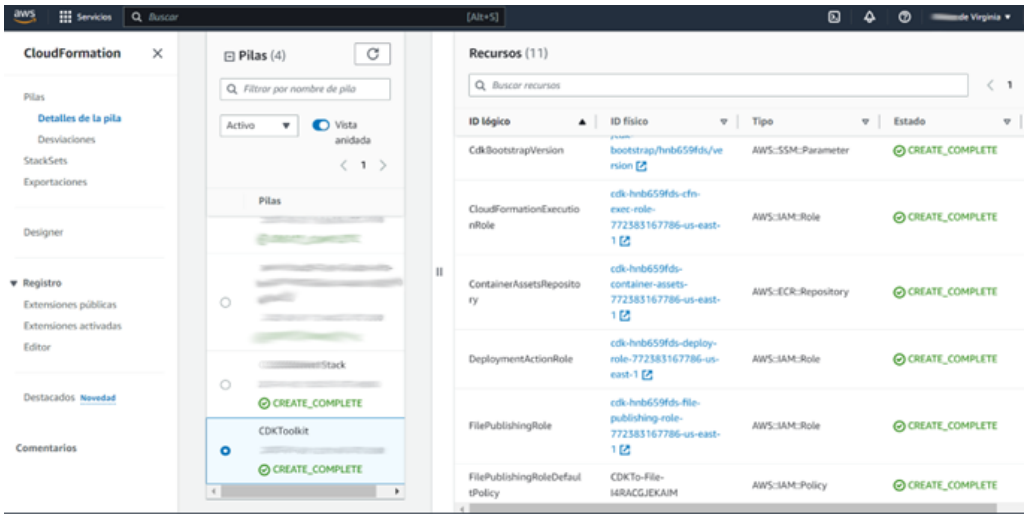
FIGURA 52. STACK DE LA INFRAESTRUCTURA CON AWS CLOUDFORMATION.



Fuente: [69]

Aquí también se ve el stack de CdkToolkit en la Figura 53, que se crea cuando se hace el despliegue para que se guarden todos los permisos y roles que va a tener toda la infraestructura.

FIGURA 53. STACK DE TOOLKIT EN AWS CLOUDFORMATION.



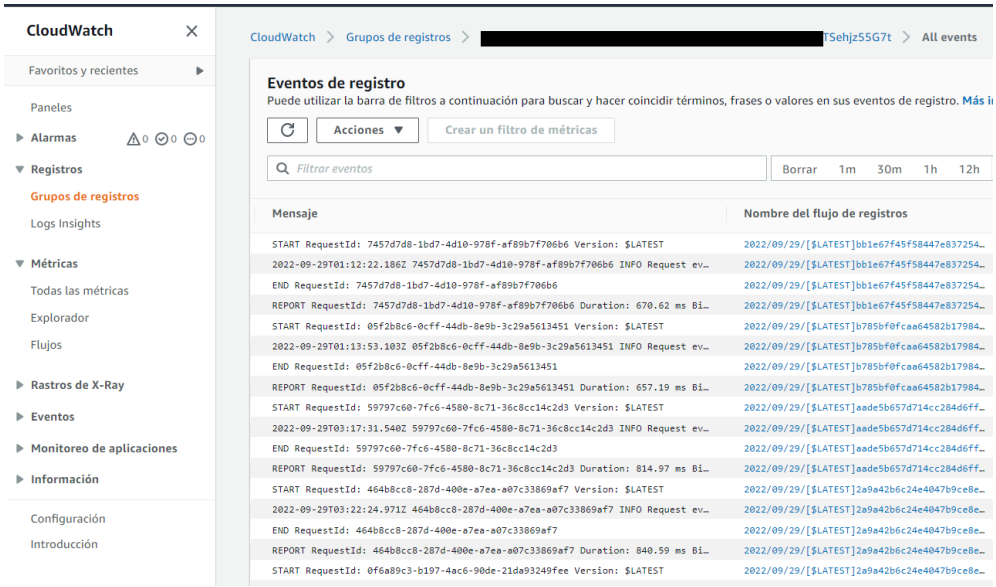
Fuente: [69]

El servicio AWS Cloudwatch permite revisar si todas las peticiones se están haciendo bien, es decir, que AWS Lambda esté integrada con la API para hacer la petición a la base de datos DynamoDB.

Todas las solicitudes se pueden revisar por medio de registros y verificar si son exitosas o existió algún error.

En caso de presentarse un error, por medio de los *logs* se puede revisar en qué solicitud o petición está la falla para corregirla como se puede observar en la Figura 54.

FIGURA 54. REGISTRO DE LOGS EN CLOUDWATCH.



Fuente: [69]

H. Pruebas de la Arquitectura

Descripcion del trabajo: esta tarea incluyó todas las pruebas de la arquitectura a través de Postman, solicitando a las API que hicieran las respectivas peticiones del usuario y del asesor.

Para rectificar que todas las peticiones de la infraestructura se hicieran bien, ya sea para el usuario o el cliente, se usó el programa Postman que permite llevar a cabo pruebas y comprobar el correcto funcionamiento de las API [78].

En dicho proceso el primer paso fue buscar el enlace de las API de cada petición (como se puede ver en el cuadro rojo de la Figura 55, 56, 57, 58 y 59). Estas API tienen un enlace diferente para cada método con el fin de probarlas y verificar que hagan la solicitud a la base de datos.

FIGURA 55. LINK DE LA API GATEWAY AL MÉTODO GET.

Fuente: [69]

FIGURA 56. LINK DE LA API GATEWAY AL MÉTODO DELETE.

Fuente: [69]

FIGURA 57. LINK DE LA API GATEWAY AL MÉTODO POST.

Fuente: [69]

FIGURA 58. LINK DE LA API AL MÉTODO PATCH.

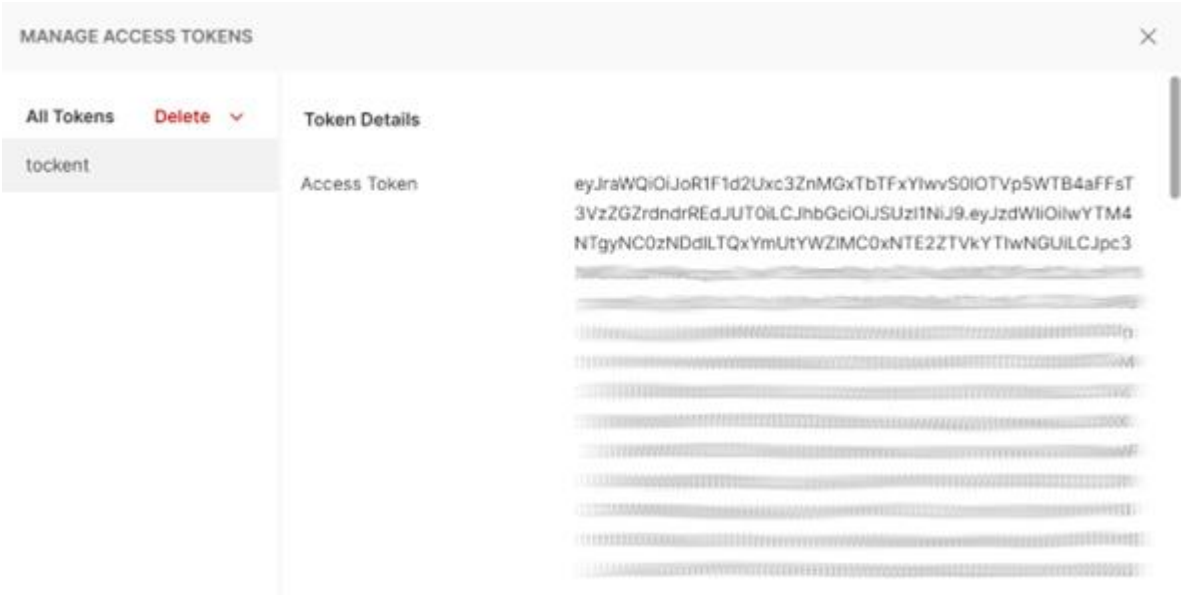
Fuente: [69]

FIGURA 59. LINK DE LA API AL MÉTODO GET (PRODUCTS).

Fuente: [69]

Luego, se procedió a probar cada método a través de Postman. El primero fue el del usuario, que se identifica cuando ingresa a través del log in y genera un token, paso que se aprecia en la Figura 60, que tiene permisos especiales que se le asignaron en la infraestructura como código, ya que el usuario solo va a acceder a su información personal.

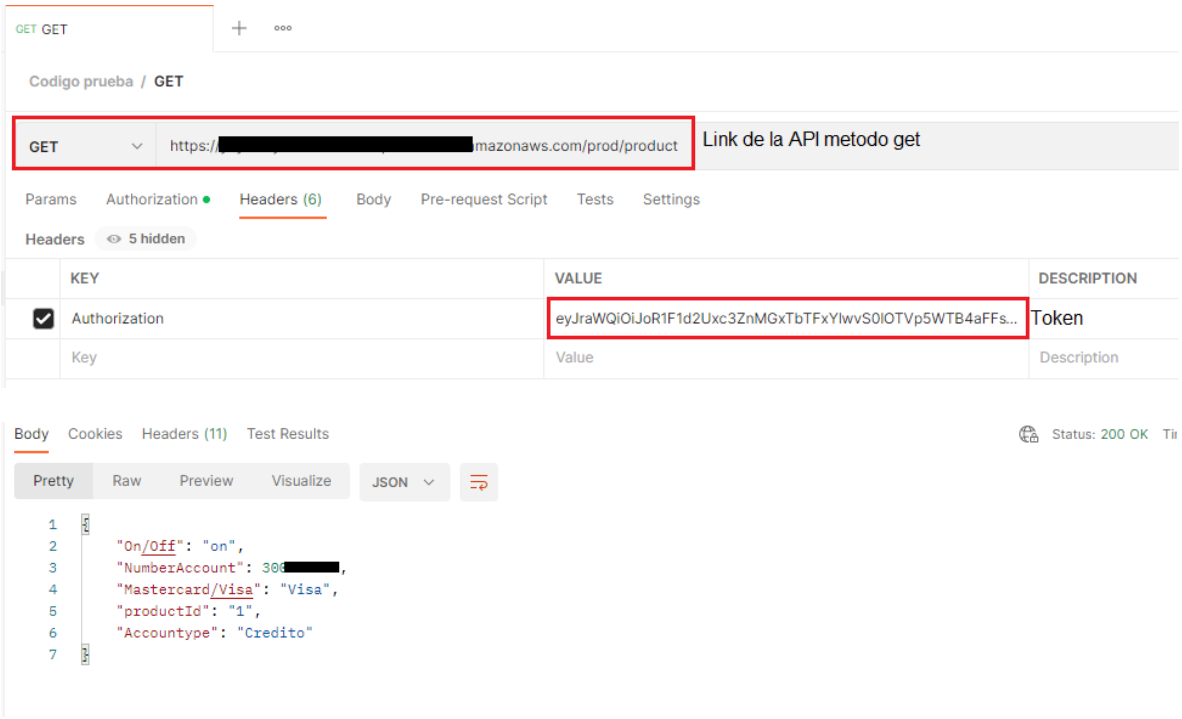
FIGURA 60. TOCKEN QUE GENERA EL USUARIO DESPUÉS DE LOGUEARSE.



Fuente: [69]

Luego de obtener el token, se utilizó el programa Postman para probar la API. Primero, se configuró con el enlace de la API, lo cual se puede consultar en el cuadro rojo No. 1 de la Figura 61. Después con el token que generó el usuario cuando ingresó, como se observa en el cuadro rojo No.2 de la Figura 61, para pedir la información a la base de datos. Seguidamente, se procedió a realizar la petición del usuario como se observa en la Figura 61.

FIGURA 61. PETICIÓN A TRAVÉS DE POSTMAN DE LA API MÉTODO GET (USUARIO).

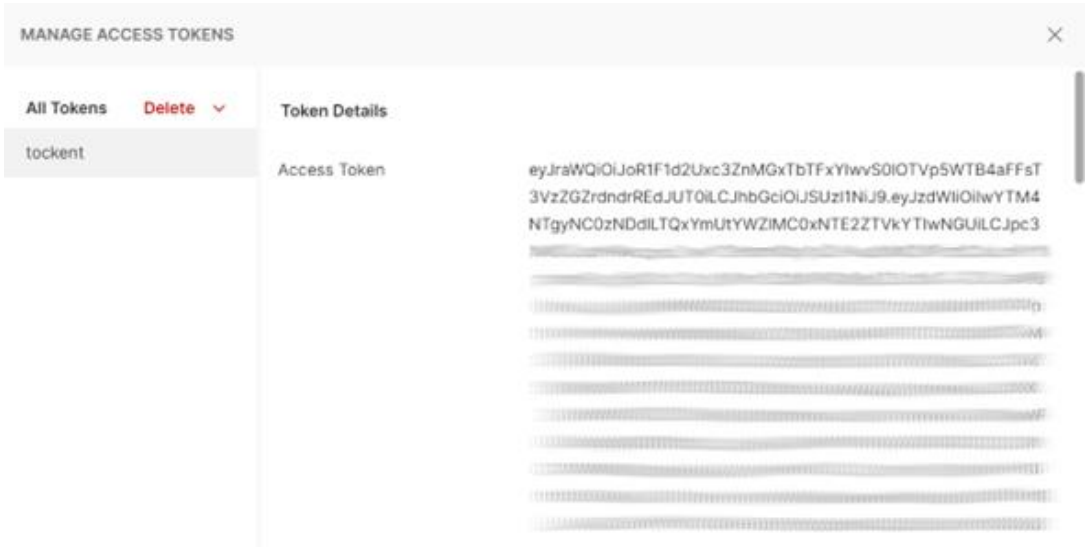


Fuente: [69]

De esta manera, se comprobó que la arquitectura funcionara bien para cuando el usuario acceda y solicite su información. Luego, se verificó la función del asesor que tiene cuatro métodos *delete*, *update*, leer con el ID o leer todos los registros y post.

Para eso se accedió (log in) con el fin de generar el token que se ve en la Figura 62 con los permisos y este se puso en Postman con su respectivo método. También, se incluyó el enlace de la API que se iba a probar.

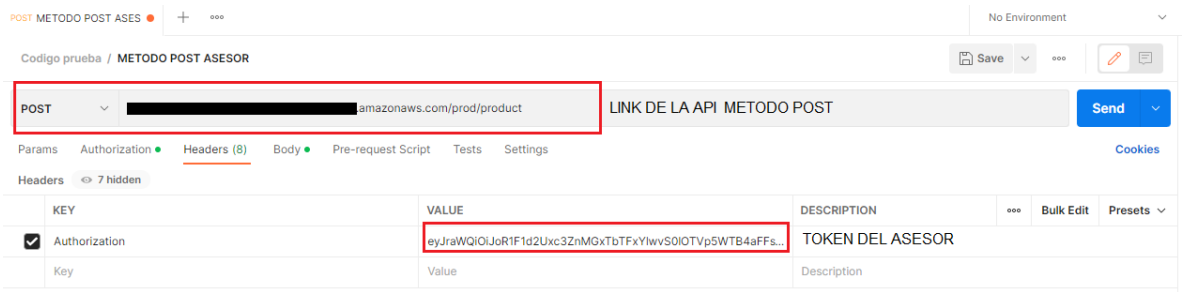
FIGURA 62. TOKEN DE UN ASESOR QUE INGRESA.



Fuente: [69]

Con el fin de probar cada método del asesor, se procedió a hacer las pruebas con Post en Postman, cambiando el enlace de las API y el token del asesor; ver Figura 63.

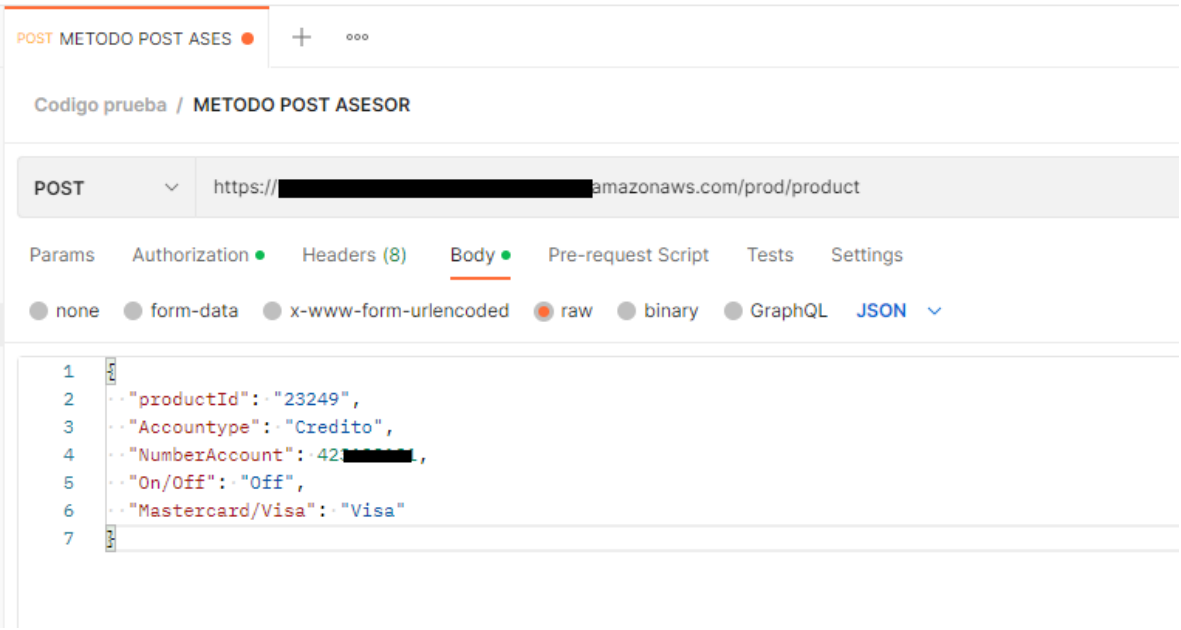
FIGURA 63. CONFIGURACIÓN DEL MÉTODO POST CON EL TOKEN DEL ASESOR Y ENLACE A API.



Fuente: [69]

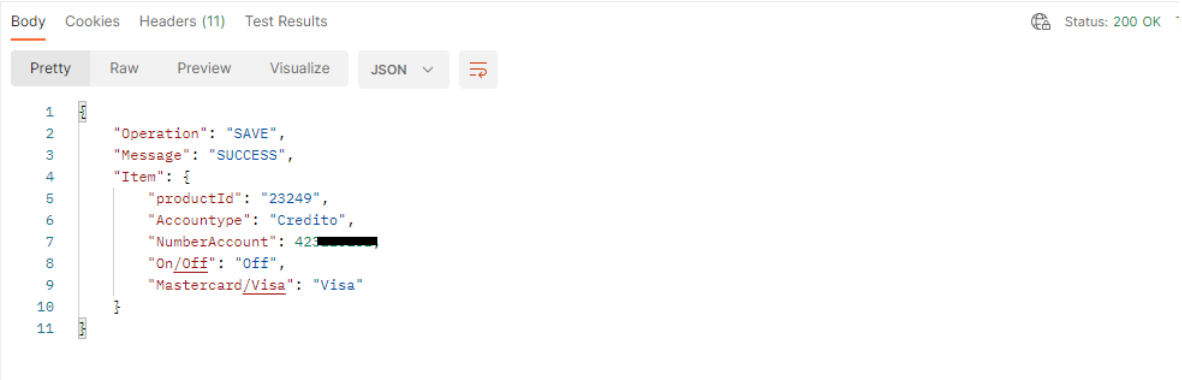
Seguidamente, se procedió a realizar la prueba de la arquitectura con función del asesor en el método Post. Aquí se debe asignar un valor a las variables (productID, AccounType, NumberAccount, On/off, Mastercard/Visa) como se puede ver en la Figura 64, para que cuando envíe la solicitud de Post se pueda guardar toda esta información en la base de datos DynamoDB. Por último, se envía la petición para que guarde toda la información que solicitó el asesor; ver Figura 64.

FIGURA 64. INFORMACIÓN QUE SOLICITA AL ASESOR PARA GUARDAR EL REGISTRO EN LA BASE DE DATOS.



Fuente: [69]

FIGURA 65. SOLICITUD DEL ASESOR CON EL MÉTODO POST.

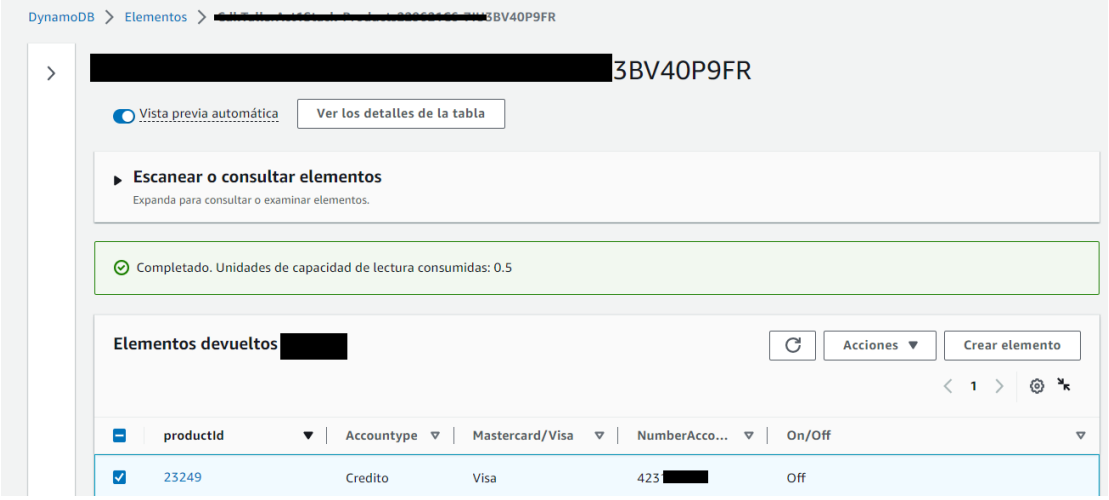


Fuente: [69]

De esta manera se comprobó que respondiera con el Message “SUCCESS” y Operation: “SAVE”, que puede consultar en la Figura 65, notificando que la petición hecha por el asesor se logró satisfactoriamente.

Para comprobar esto se revisó la base de datos de DynamoDB, paso que se aprecia en la Figura 66, donde se verificó si se había guardado lo que el asesor ingresó.

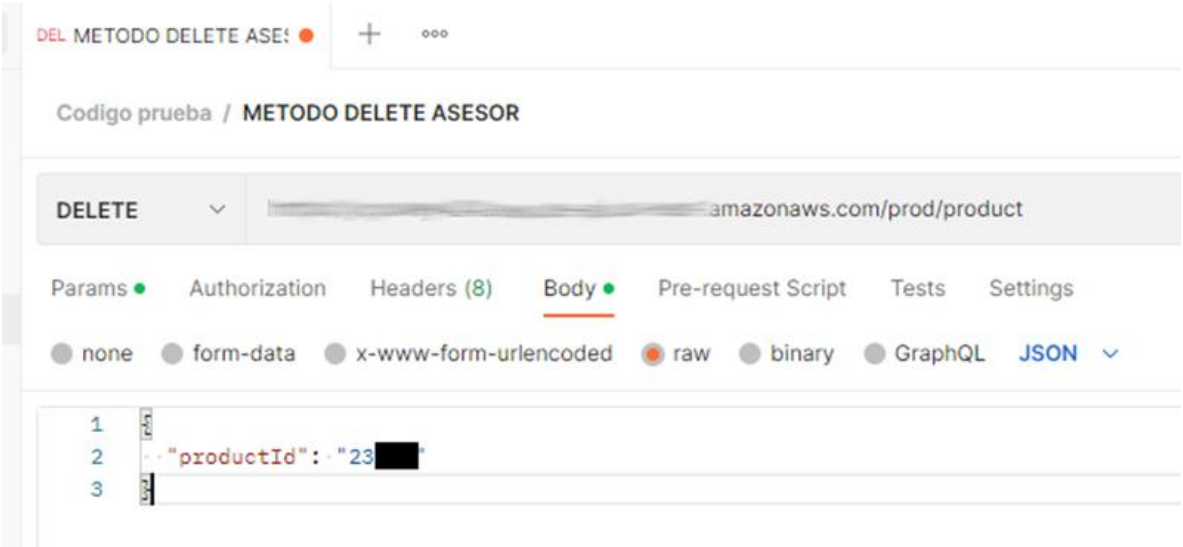
FIGURA 66. REVISIÓN DE LA BASE DE DATOS DYNAMODB.



Fuente: [69]

El próximo método para comprobar para el asesor fue el Delete, el cual se ve en la Figura 67. En sus pruebas en Postman solo se cambió el método y el enlace, ya que el token que generó anteriormente tenía los permisos incluidos. Una vez se hicieron los cambios, se probó la API del método Delete. Cabe anotar que, para eliminar un registro de la base de datos se necesita ingresar su ID.

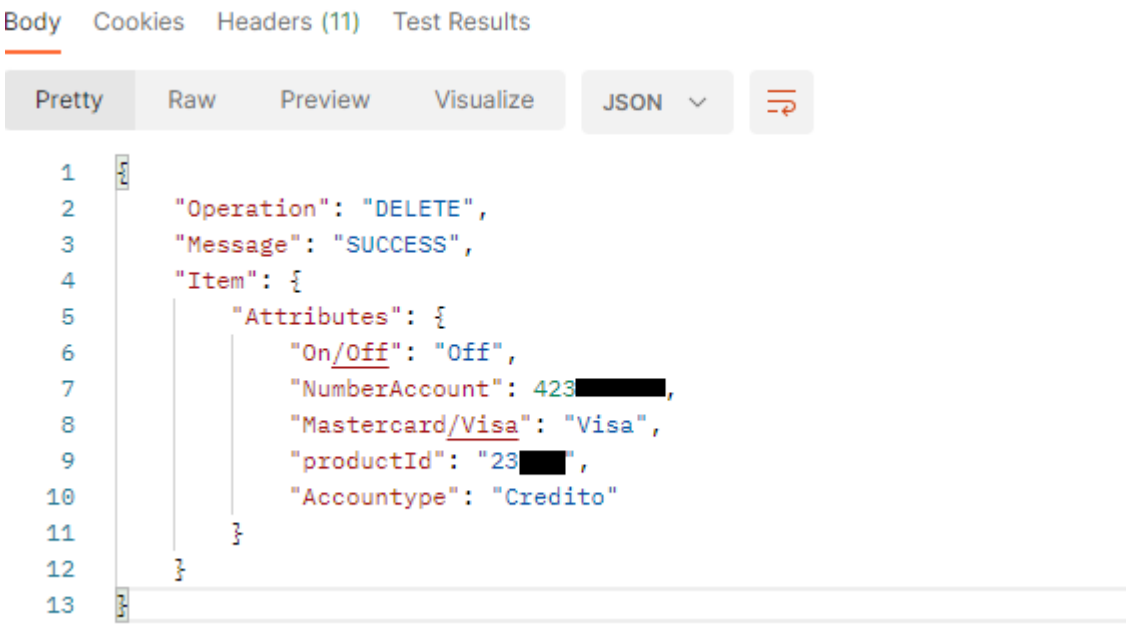
FIGURA 67. COMPROBACIÓN DEL MÉTODO DELETE.



Fuente: [69]

Posteriormente, se procedió a enviar la solicitud para que lo eliminara de la base de datos.

FIGURA 68. MÉTODO DELETE APROBADO.



Fuente: [69]

Cuando se elimina un registro se muestra el Message “Success” como se observa en la Figura 64 y que confirma que se ha borrado.

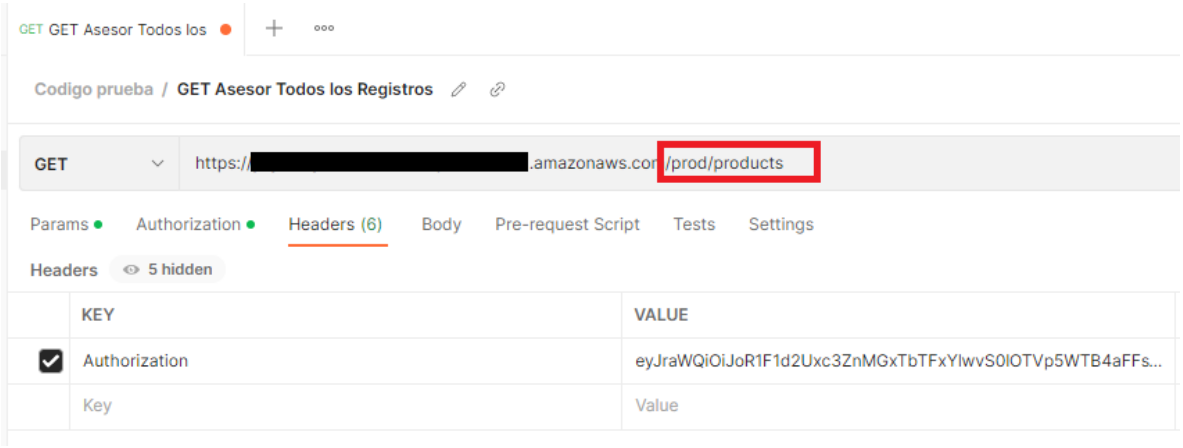
El siguiente método para comprobar fue el GET, el cual puede ser usado por el asesor con dos funciones:

1. Buscar todos los registros de la base de datos.

2. Buscar el registro por un ID específico.

En este caso, para las pruebas de Postman solo se cambió el método y el enlace, teniendo en cuenta que para la función uno del método GET la API usó el recurso “products” (como se ve en el cuadro rojo de la Figura 69) para que aparecieran todos los registros de la base de datos.

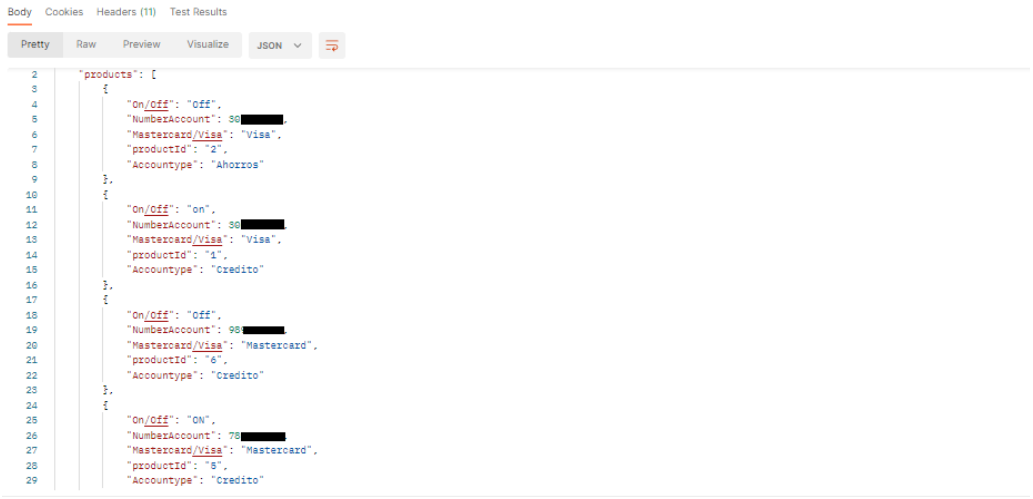
FIGURA 69. MÉTODO GET PARA TODOS LOS REGISTROS DE LA BASE DE DATOS.



Fuente: [69]

Luego, se le hizo la petición de buscar todos los registros que hay en la base de datos; ver Figura 70.

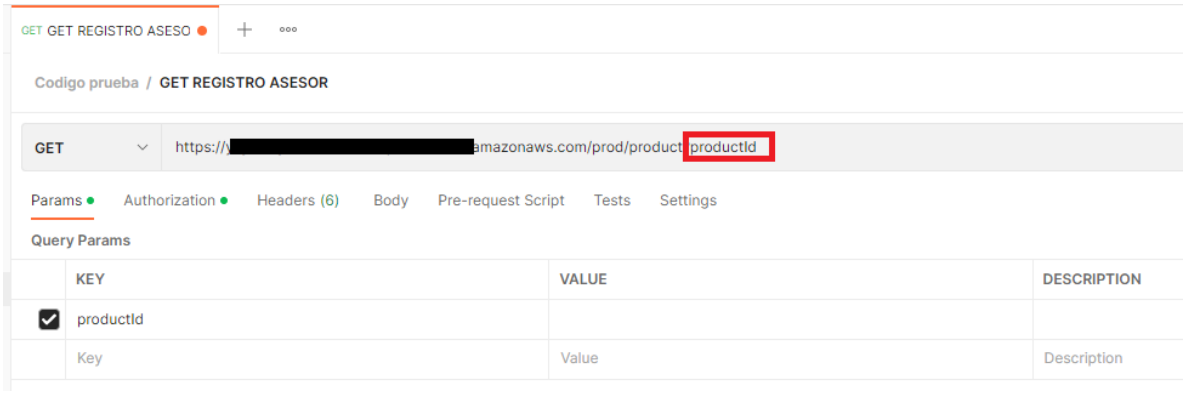
FIGURA 70. MÉTODO GET CON TODOS LOS REGISTRO APROBADOS.



Fuente: [69]

Después hacer la petición a la API y recibir todos los registros se comprobó que la infraestructura del método GET con la primera función operara satisfactoriamente. El siguiente paso fue comprobar la función dos, para lo cual se cambió el enlace del método a “productId” (ver Figura 71), ya que su misión es buscar por el ID el registro que requiera el asesor.

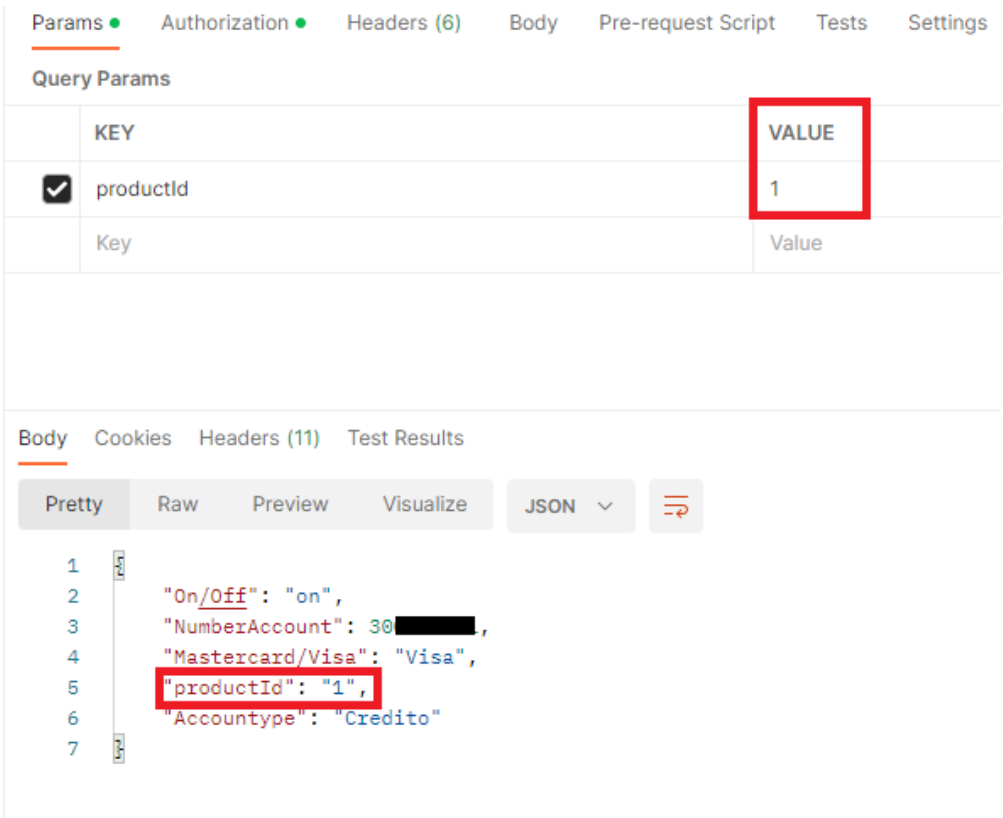
FIGURA 71. CONFIGURACIÓN DEL LINK DEL MÉTODO GET PARA BUSCAR POR EL ID EN LA BASE DE DATOS.



Fuente: [69]

Luego de configurar el enlace, se inició la prueba del método poniendo en “value” el ID que el asesor buscaba, lo que se observa en la Figura 72.

FIGURA 72. MÉTODO GET. BUSCAR EL REGISTRO POR EL ID APROBADO.

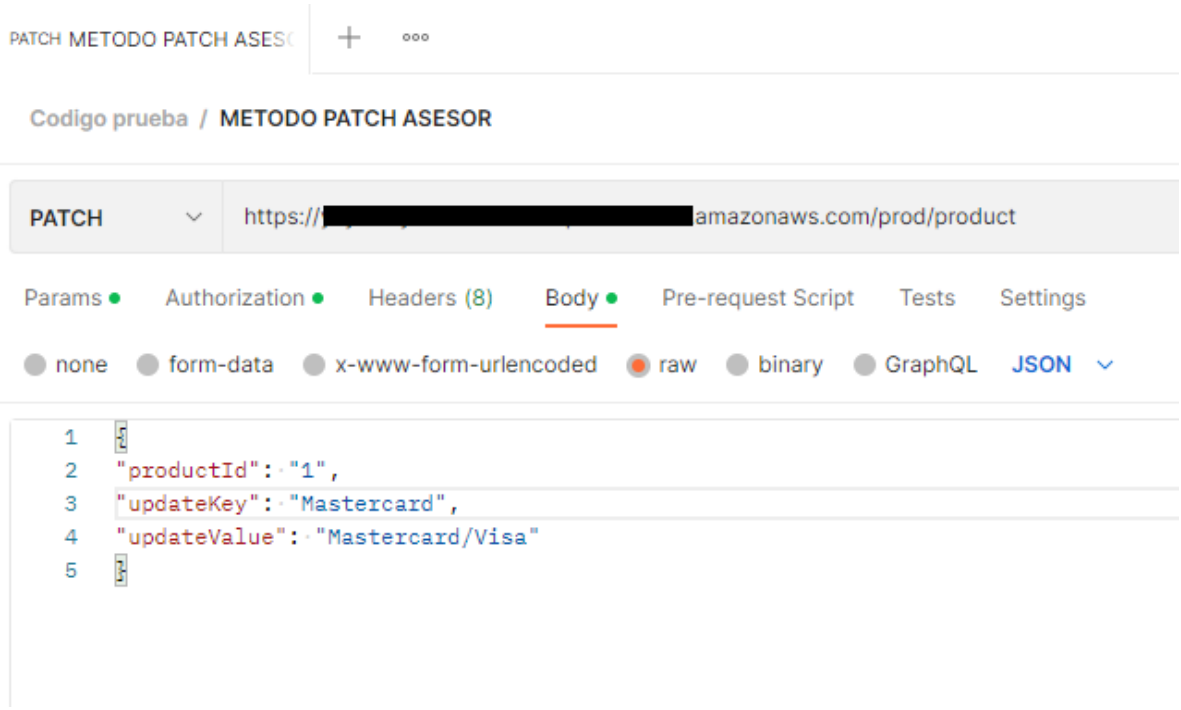


Fuente: [69]

Después de hacer la petición a la API del método GET con la función de buscar un registro por su ID, fue aprobada la arquitectura para realizar las dos búsquedas (ver la Figura 73).

Para terminar, se hizo prueba al método “PATCH” o Update (consultar la Figura 73) para que el asesor pudiera modificar algún registro en la base datos. Dicha petición necesitó que en Postman se cambiara el enlace de la API, después el “productID” del registro que se iba modificar y el dato.

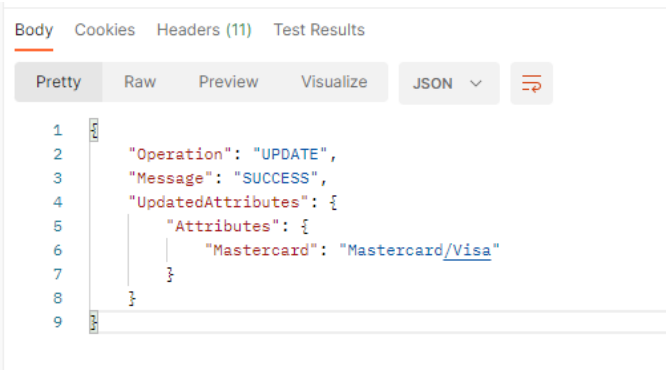
FIGURA 73. CONFIGURAR POSTMAN PARA EL MÉTODO PATCH.



Fuente: [69]

Una vez se ingresaron los datos que se deseaban actualizar en la base de datos, se envió la petición a la API, cuya respuesta fue exitosa como se observa en la Figura 74.

FIGURA 74. MÉTODO PATCH APROBADO.



Fuente: [69]

Es así como se demostró, a través de Postman, que la Arquitectura Cloud orientada a Microservicios Tipo Serverless funciona en todos los servicios de AWS con cada uno de sus métodos CRUD y funciones, tanto para el usuario como para el asesor.

Además, se llevaron a cabo pruebas utilizando Newman, una biblioteca de Npm que proporciona resultados detallados de las pruebas realizadas en Postman. Con esta herramienta se obtuvo información valiosa como el porcentaje de éxito en cada prueba, el tiempo de respuesta y el código de respuesta para cada método probado a través de su correspondiente URL. Newman también posibilitó la inspección del cuerpo de respuesta generado durante las pruebas y verificar si cumplía con los criterios establecidos para cada método. Esta integración resultó fundamental para evaluar el rendimiento y la calidad de nuestras pruebas automatizadas.

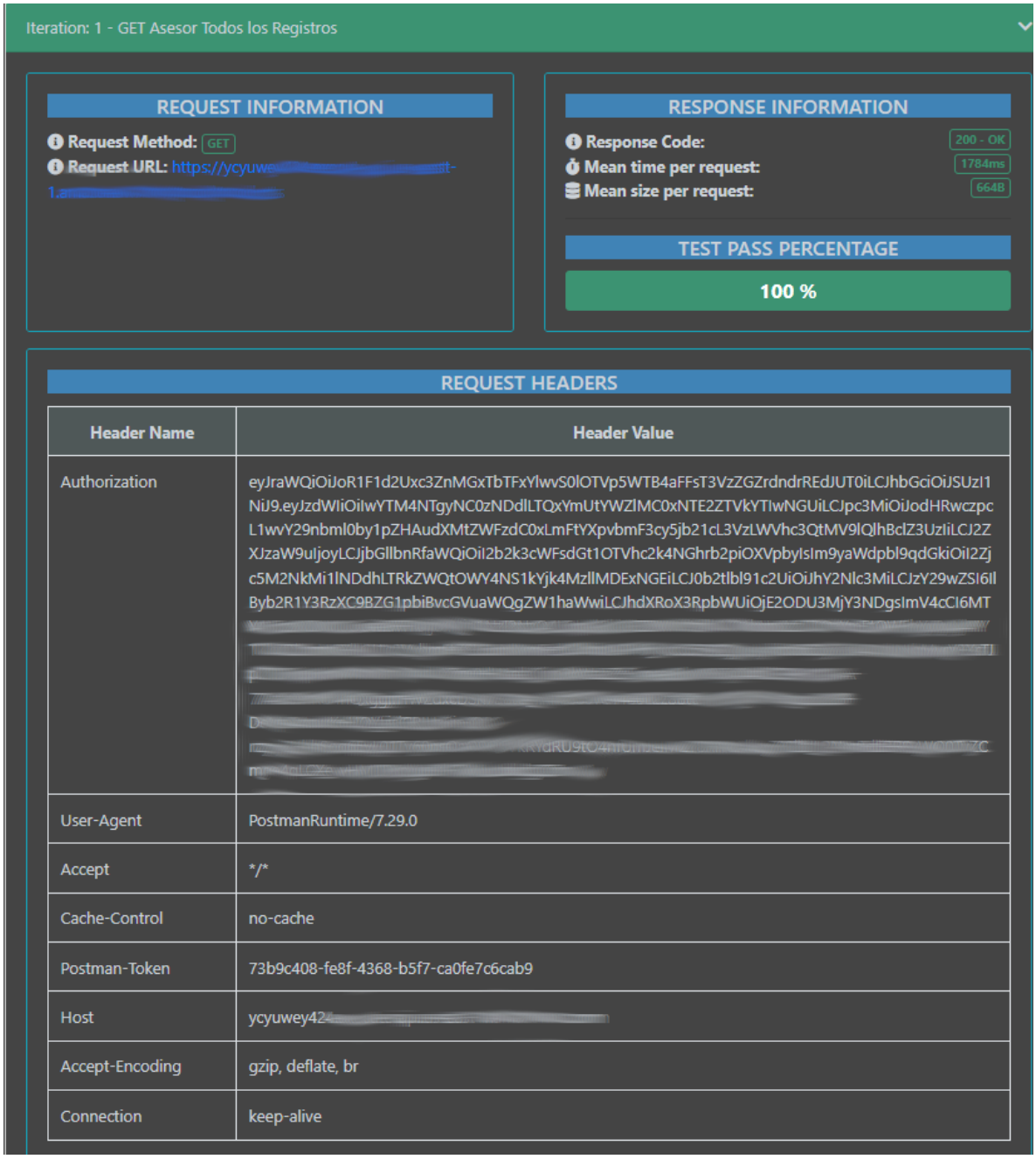
En la Figura 75, se puede apreciar cómo Newman generó un panel de control que muestra todas las API y métodos probados tanto para el usuario como para el asesor. Cabe destacar que, todas las pruebas fueron exitosas. Además, la biblioteca Newman examinó cada método de forma individual.

FIGURA 75. DASHBOARD DE LOS 6 MÉTODOS.

Fuente: [69]

Un ejemplo de esto se muestra en la figura 76, donde se realiza una solicitud GET por parte del asesor para obtener todos los registros de la base de datos. En este panel de control se puede observar el URL de la API utilizado, el método empleado (en este caso GET), así como el tiempo de respuesta y el porcentaje de éxito de la prueba.

FIGURA 76. DASHBOARD DEL MÉTODO GET (TODOS LOS REGISTROS).

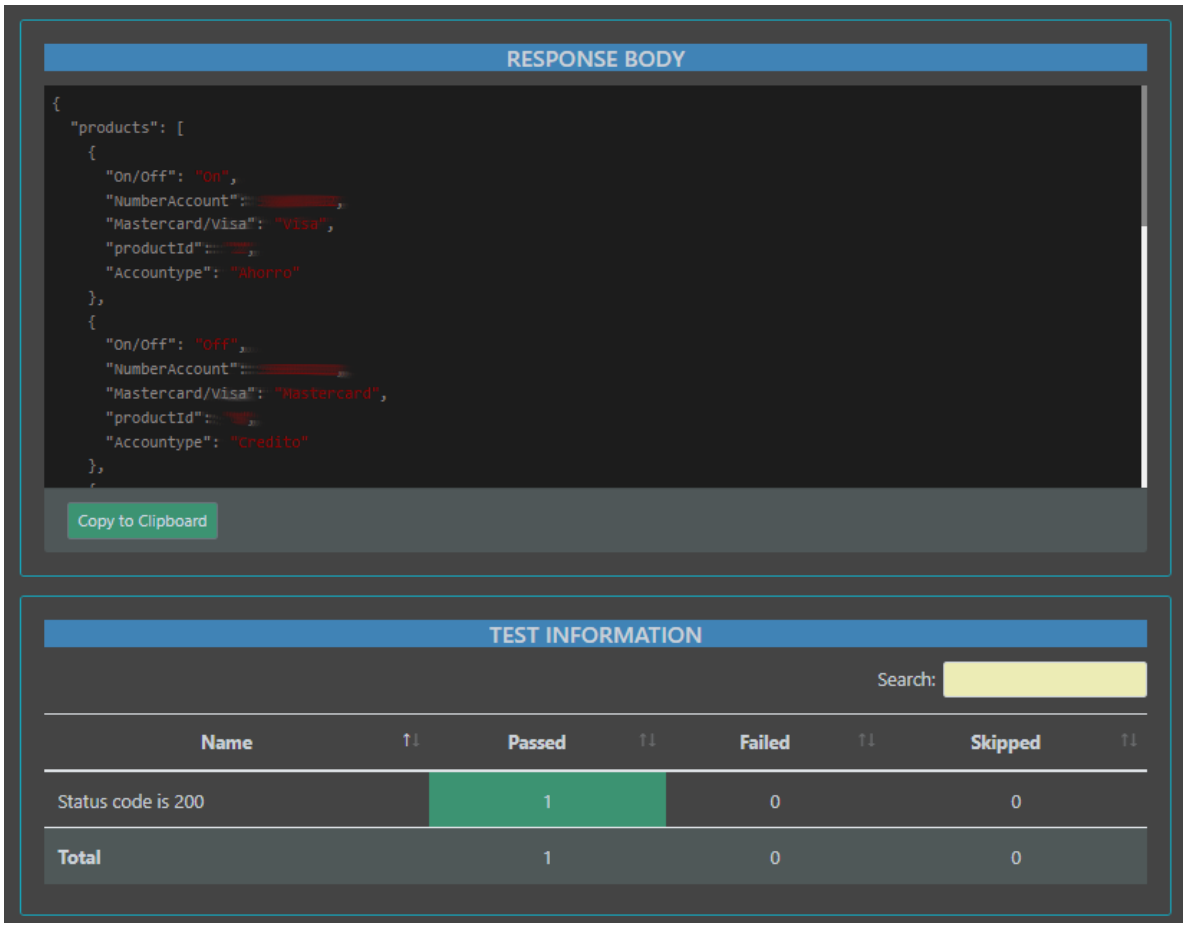


Fuente: [69]

Adicionalmente, se examinaron los encabezados de la solicitud, donde destaca la presencia de un campo de autorización (authorization) que contiene el token generado por el asesor al momento de ingresar. Por último, el cuerpo de respuesta en las pruebas también puede ser visualizado, en este caso

la solicitud GET, mostrando todos los datos de los usuarios en la base de datos y proporcionando un gráfico que representa el código de estado (status code) y si la prueba fue superada satisfactoriamente.

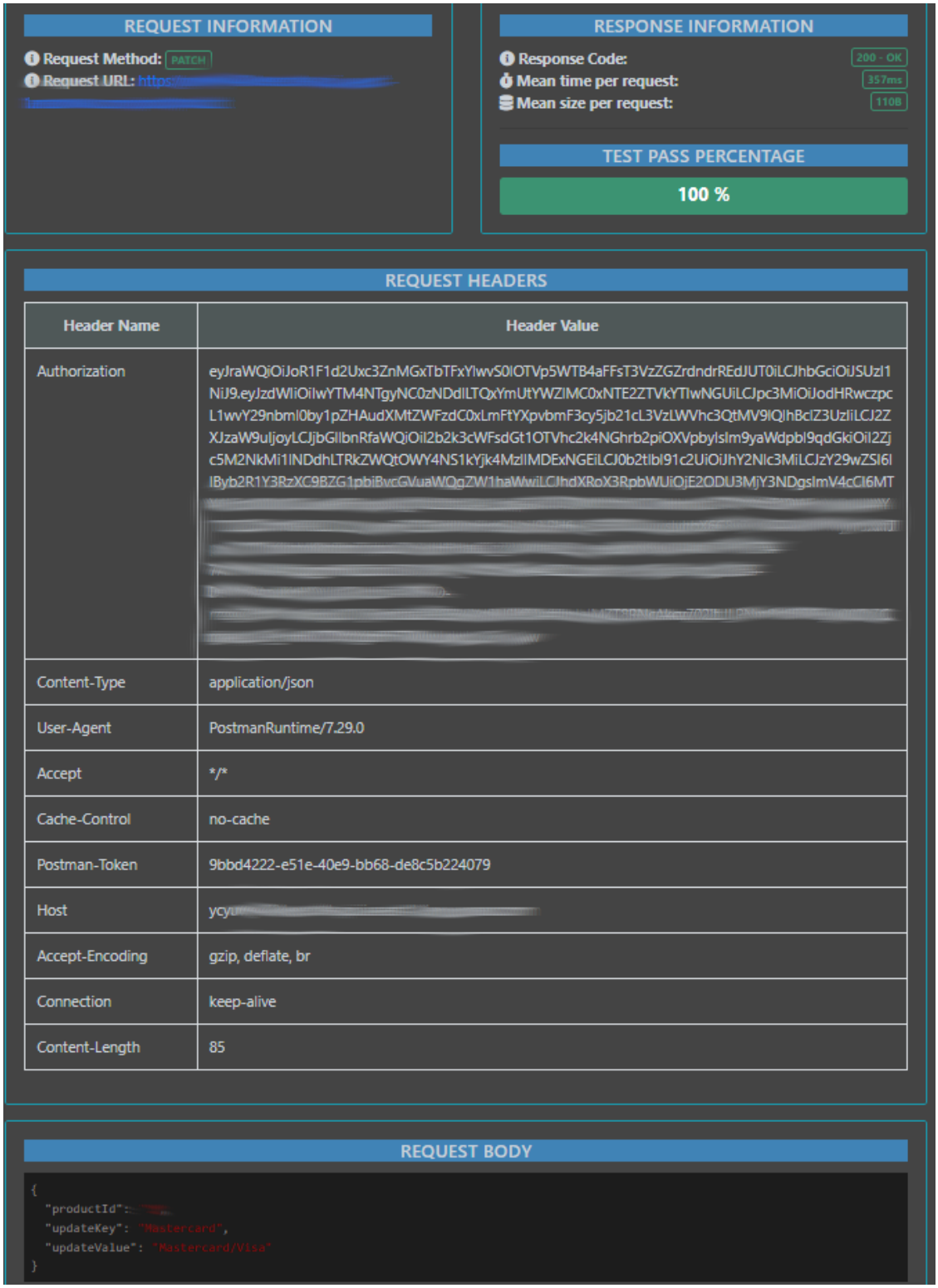
FIGURA 77.RESPONSE DEL MÉTODO GET.



Fuente: [69]

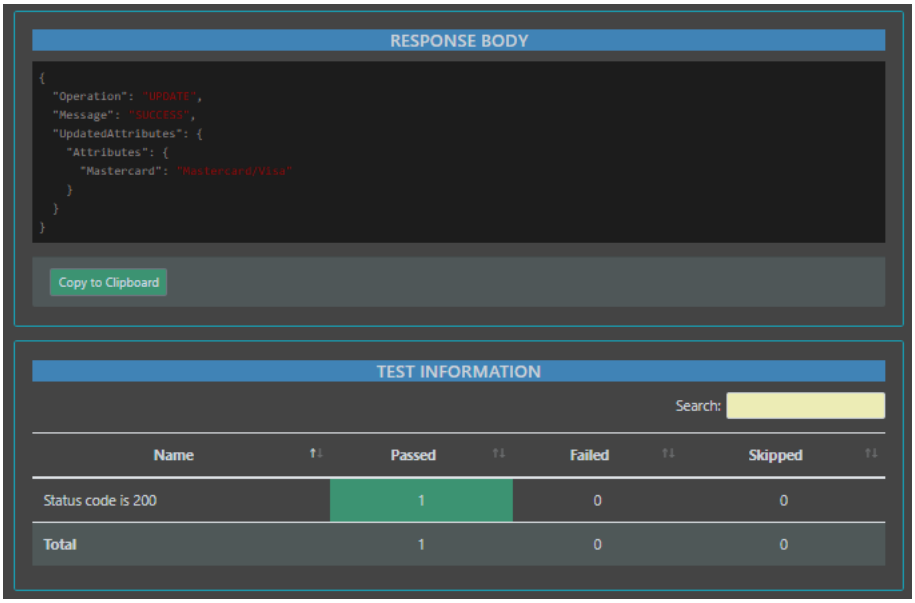
Luego, se llevó a cabo la prueba del método PATCH, como se muestra en las figuras 77 y 78, con el objetivo de permitir al asesor realizar modificaciones en la base de datos. Al igual que en los métodos anteriores, en este panel de control se puede visualizar información relevante como la URL de la solicitud, el tiempo de respuesta, el porcentaje de éxito de la prueba y la petición del body que es lo que se le pide actualizar con su respectiva respuesta. Es importante mencionar que, para todos estos métodos se requiere el token generado por el asesor al momento de iniciar sesión, lo cual garantiza la seguridad y autorización adecuada.

FIGURA 78. DASHBOARD DEL MÉTODO PATCH.



Fuente: [69]

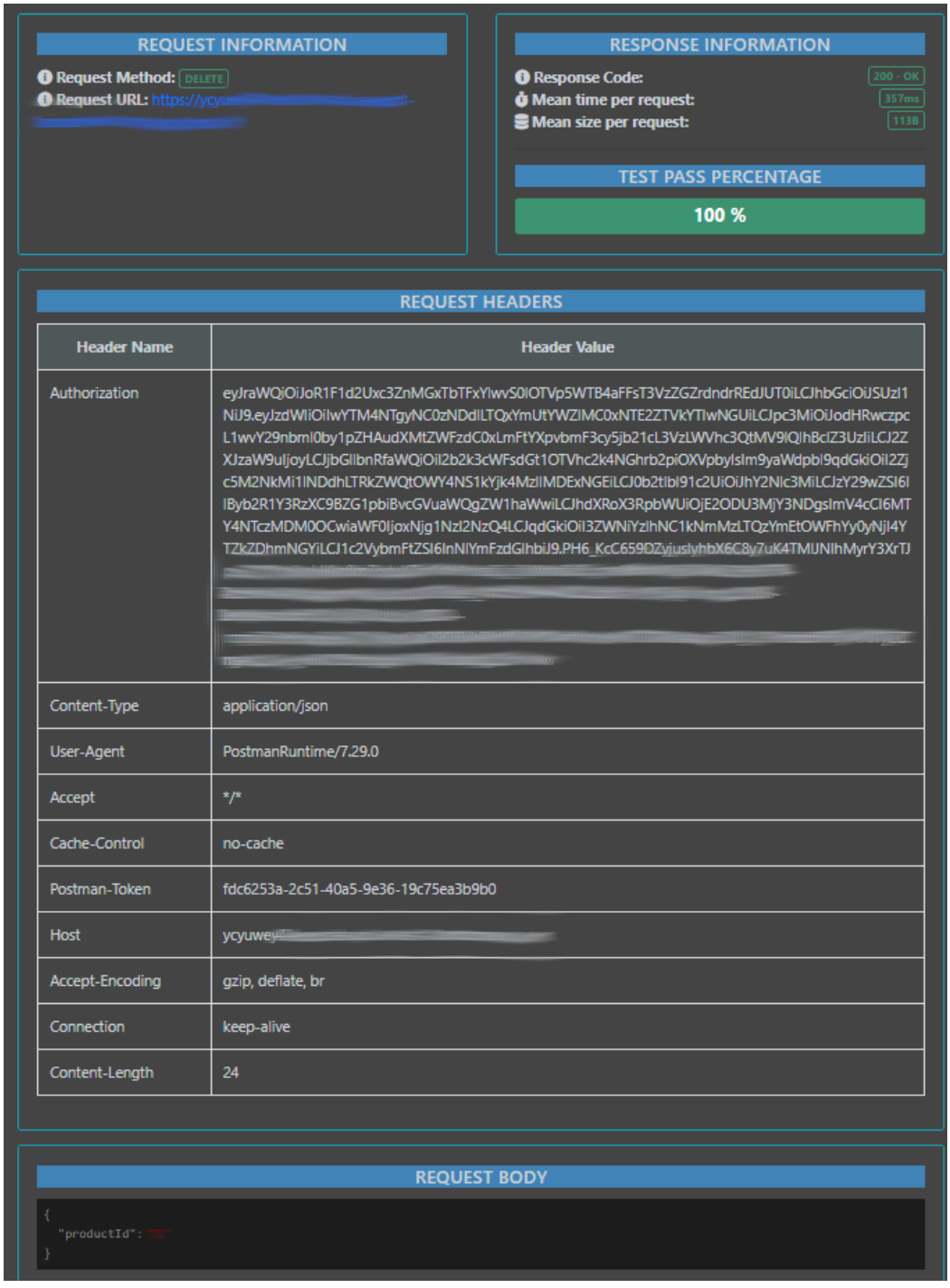
FIGURA 79. RESPONSE DEL MÉTODO PATCH.



Fuente: [69]

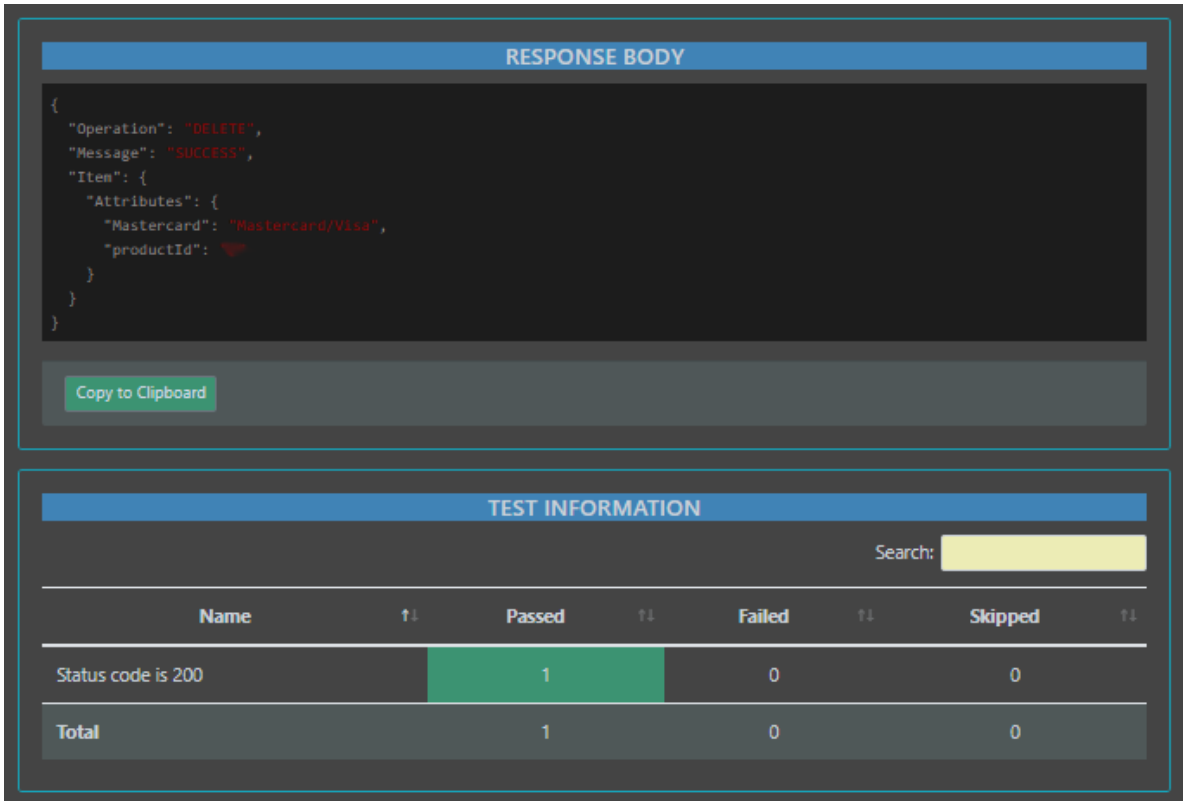
La prueba del método Delete, como se muestra en la figura 80 y 81, tuvo como objetivo permitir al asesor eliminar registros de la base de datos. Aquí también se observan los detalles esenciales. Esto brinda una visión clara de la interacción entre el asesor y la base de datos, permitiendo un adecuado control y gestión de los datos.

FIGURA 80. DASHBOARD MÉTODO DELETE.



Fuente: [69]

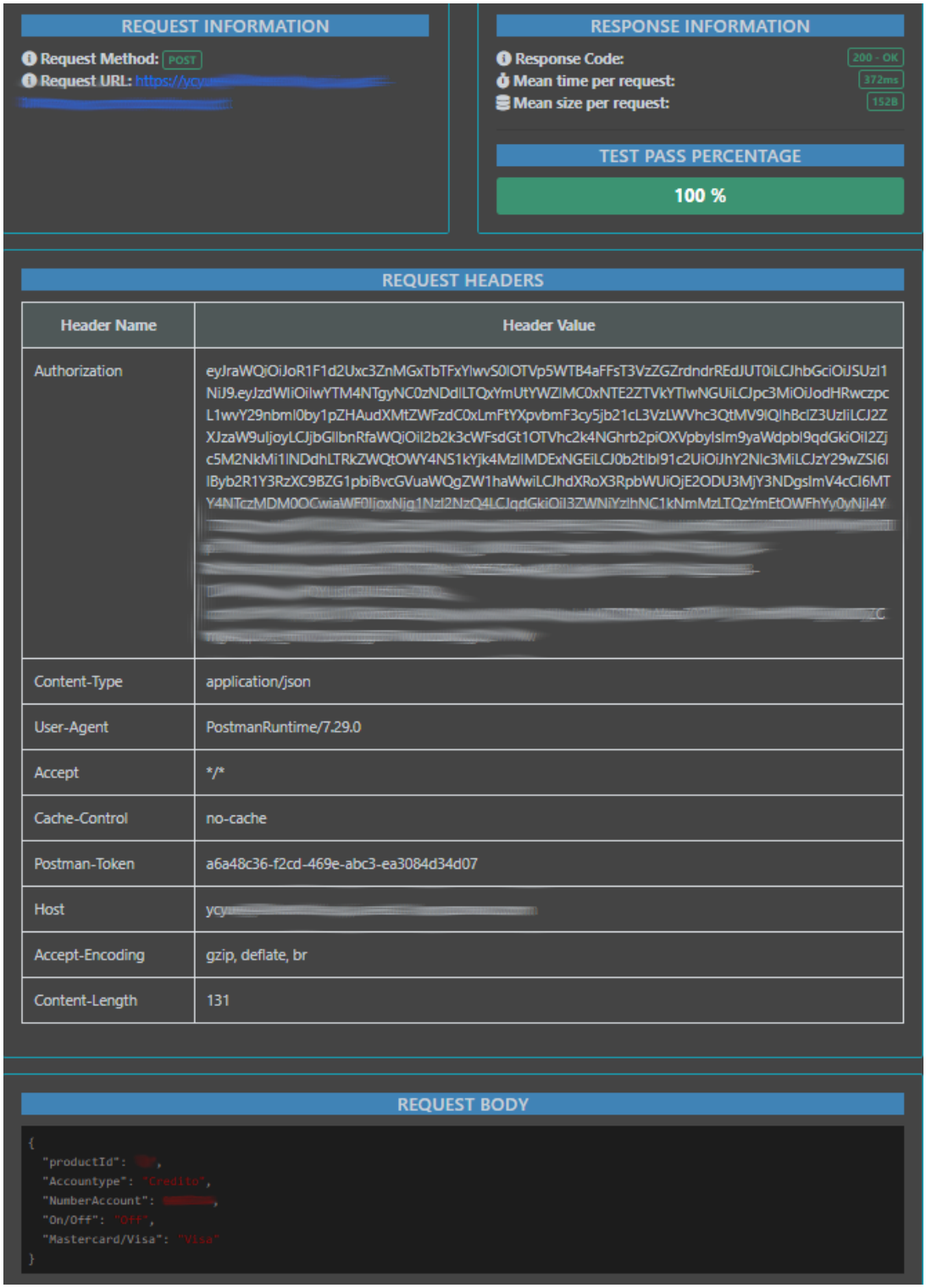
FIGURA 81. RESPONSE DEL MÉTODO DELETE.



Fuente: [69]

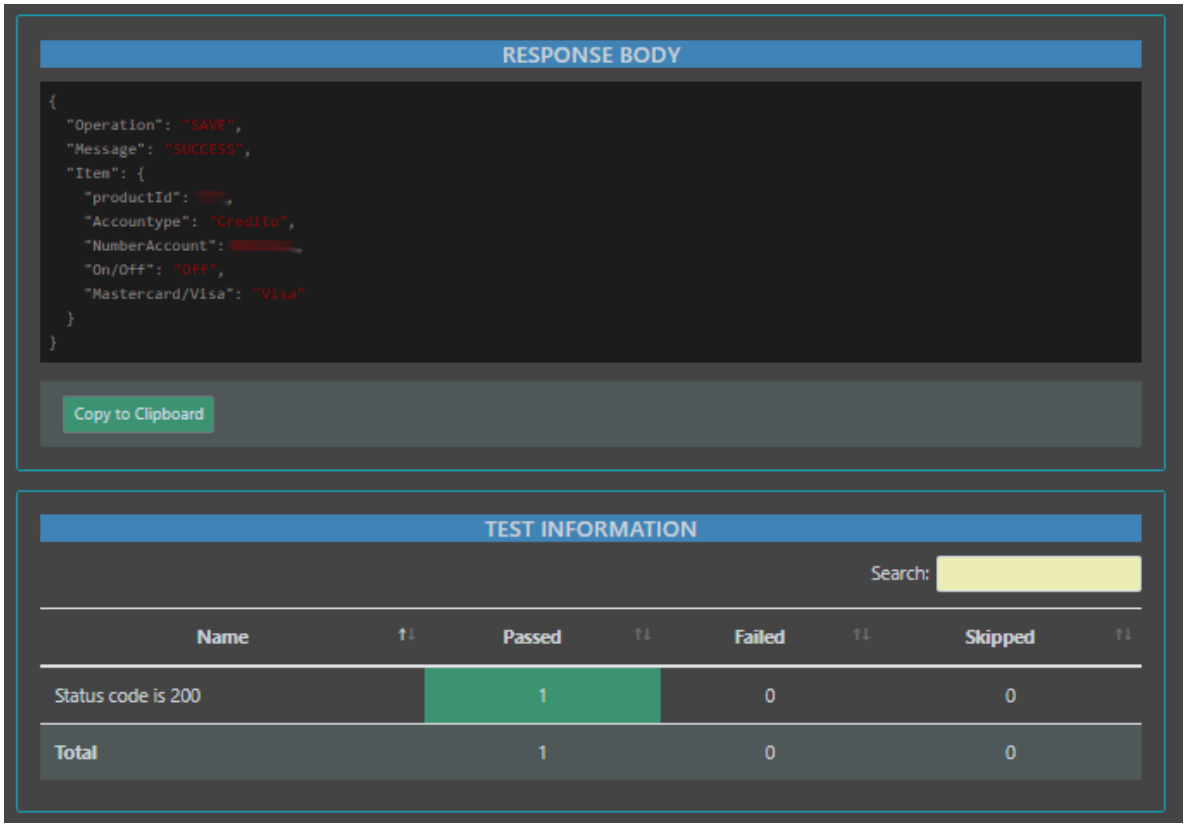
Continuando con las pruebas, se procedió a evaluar el método POST, como se puede apreciar en las figuras 82 y 83. En este escenario se hizo necesario enviar los datos correspondientes en el requestBody y se obtuvo una respuesta exitosa. La visualización detallada de la solicitud y la respuesta en el dashboard proporciona una visión clara del proceso de inserción de nuevos registros en la base de datos. Este enfoque permite verificar y validar el correcto funcionamiento del método POST, asegurando la integridad y consistencia de los datos ingresados.

FIGURA 82. DASHBOARD DEL MÉTODO POST.



Fuente: [69]

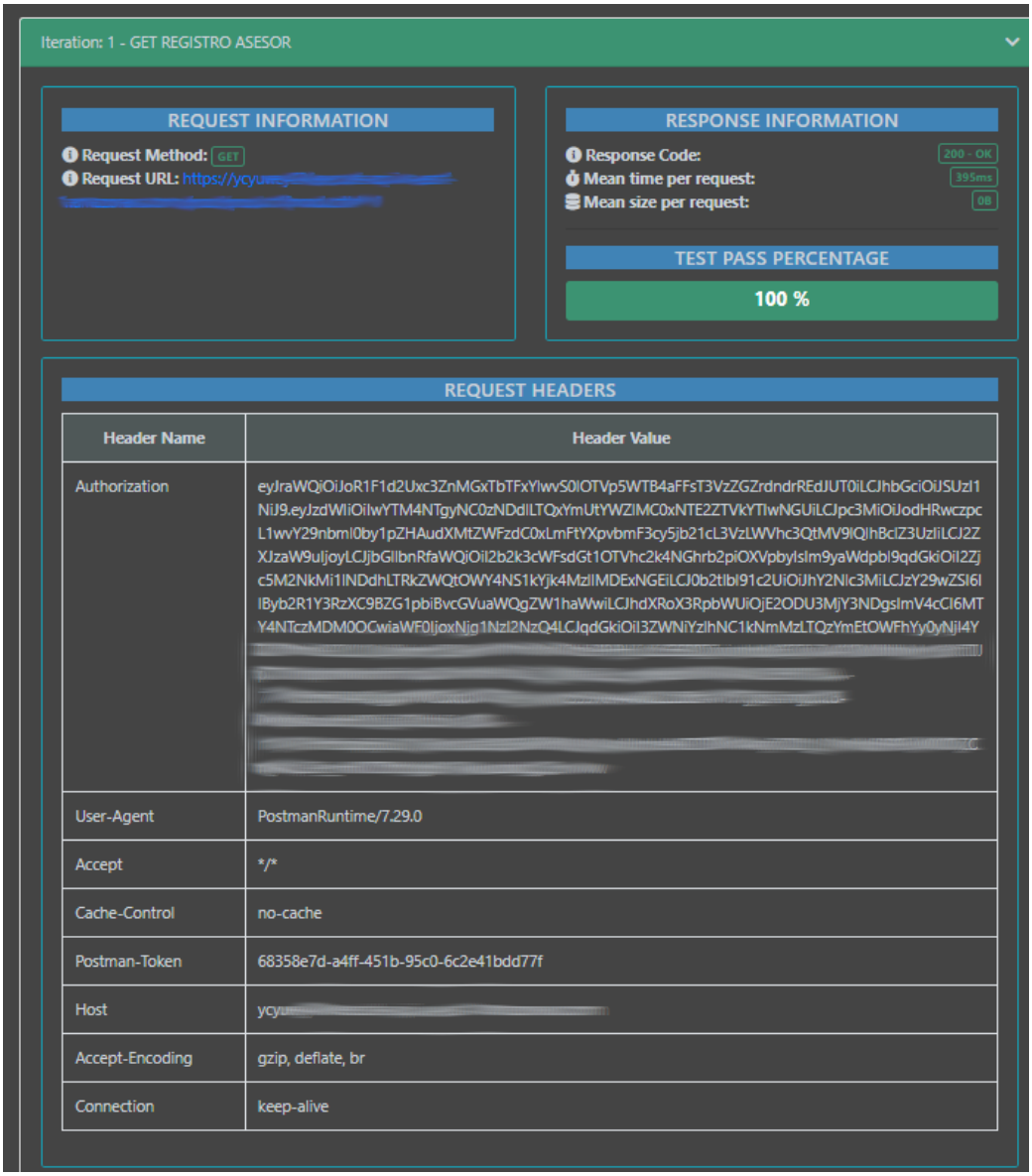
FIGURA 83. RESPONSE DEL MÉTODO POST.



Fuente: [69]

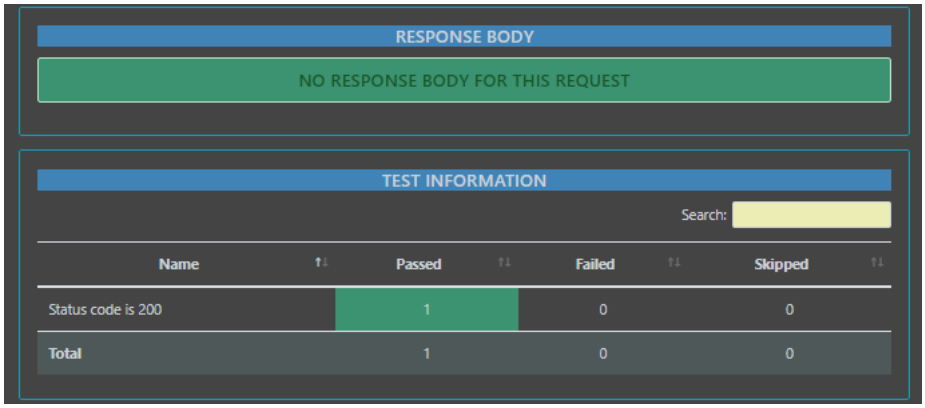
Por último, se encuentra el método GET, que hace posible que el asesor busque un usuario en la base de datos, como se muestra en las figuras 84 y 85. A diferencia de los otros métodos, en este caso no se visualiza la información en el body, ya que se pasa el ID a través de los parámetros de la consulta (query params). Esta prueba permite verificar la funcionalidad de búsqueda precisa y proporciona al asesor acceso a los detalles específicos de un usuario en particular.

FIGURA 84. DASHBOARD MÉTODO GET ASESOR (ID).



Fuente:[69]

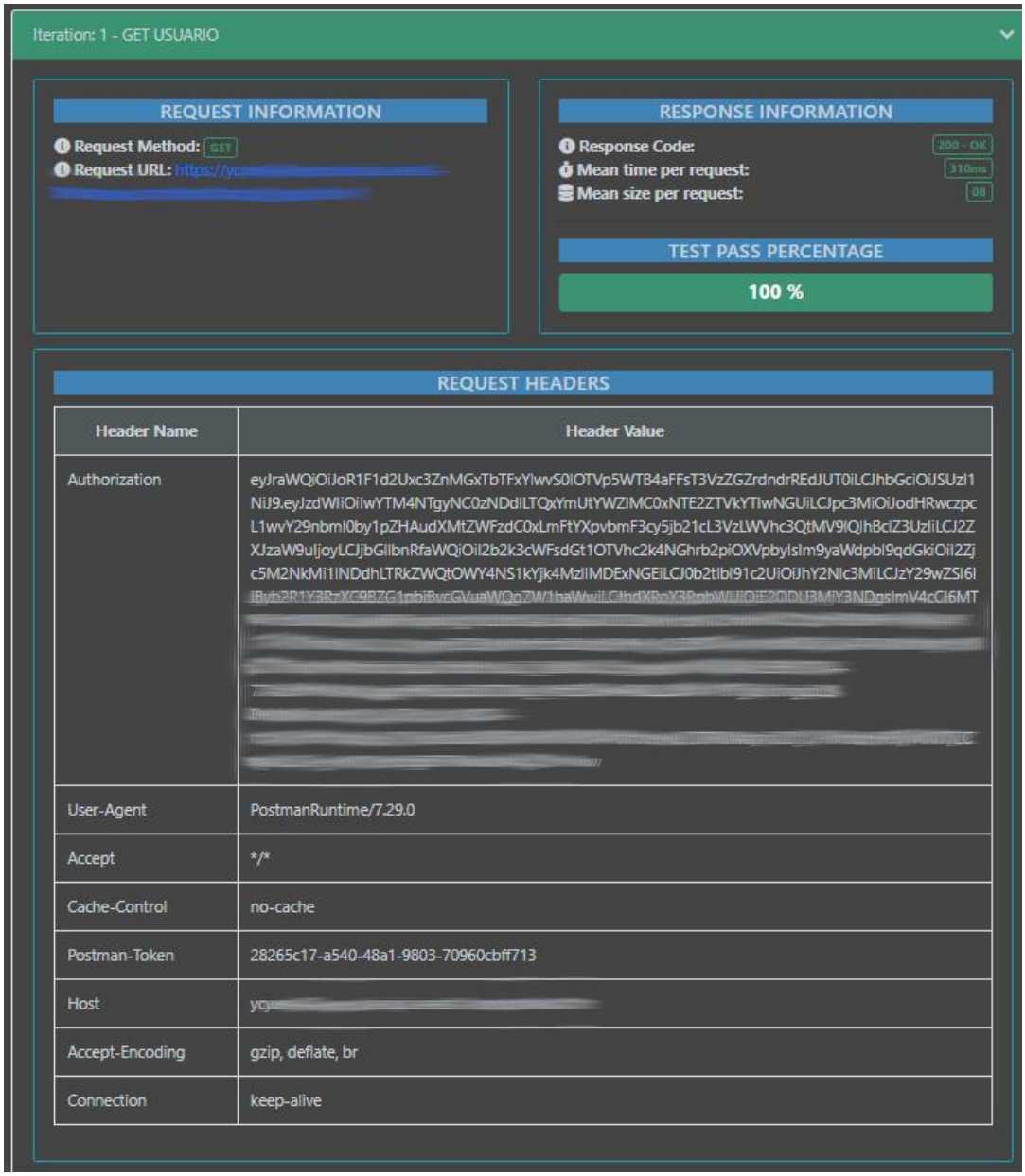
FIGURA 85. RESPONSE MÉTODO GET.



Fuente: [69]

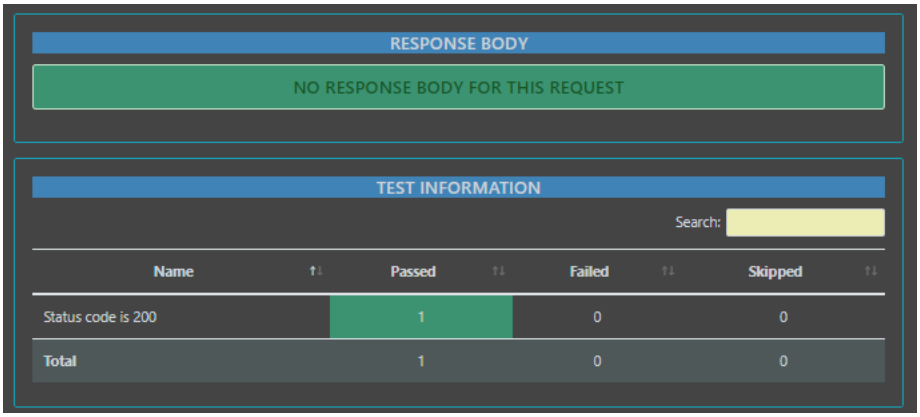
En cuanto al método GET para el usuario, que tiene como objetivo permitirle consultar su información, en esta solicitud se realiza una consulta a la base de datos utilizando los parámetros de búsqueda (query params), lo cual garantiza que el usuario solo pueda visualizar su información. Es importante resaltar que esta tiene una cobertura del 100%, tal como se muestra en la Figura 86 y 87. Además, la seguridad se mantiene mediante el uso del token.

FIGURA 86. DASHBOARD MÉTODO GET USUARIO.



Fuente: [69]

FIGURA 87. RESPONSE MÉTODO GET USUARIO.



Fuente: [69]

El análisis exhaustivo de la infraestructura como código, donde se superaron todos los filtros de seguridad y calidad, se hizo en SonarQube (Figura 49). Los resultados mostraron que la infraestructura cumplía con los más altos estándares de calidad y eficiencia en cuanto al código.

Este paso es relevante porque la calidad de la infraestructura como código es esencial para garantizar el éxito del proyecto, ya que esto permite crear y mantener una infraestructura robusta, escalable y segura para los usuarios. Por eso se usan herramientas como SonarQube, que ayuda a identificar y corregir errores en el código, lo que mejora la calidad y la eficiencia de la infraestructura.

VI. CONCLUSIONES

Una vez terminado el desarrollo de la pasantía se pudo concluir que los objetivos se cumplieron en su totalidad, ya que se hizo la implementación exitosa de la arquitectura Cloud en AWS orientada a microservicios tipo serverless y se satisfizo las necesidades del cliente establecido por ACCENTURE.

Este logro y el proceso para llegar a dicho resultado, en primer lugar, le aportó al pasante experiencia profesional y aprendizaje acerca de los servicios AWS, la metodología usada y el trabajo en equipo. Además, se aplicaron conocimientos propios de la Ingeniería Electrónica, los cuales fueron adquiridos en el transcurso de la formación académica, tales como programación, calidad de software, herramientas y procesos de desarrollo de software, entre otros, en un proyecto empresarial de Centroamérica en el que se emplearon buenas prácticas, principios SOLID, pruebas unitarias, documentación y código bien estructurado.

En este sentido, el pasante obtuvo un beneficio, pero también le aportó a la empresa su tiempo, trabajo y capacidad de resolución de necesidades, en este caso las planteadas por el cliente.

En segundo lugar, el cliente recibió una solución tecnológica a su solicitud: una arquitectura Cloud basada en microservicios que se modeló, construyó e implementó con contenedores AWS, lo que le proporciona una fácil escalabilidad y flexibilidad en la gestión de recursos.

Ahora bien, el uso de metodologías ágiles como SCRUM y XP hizo posible la entrega del proyecto en un período corto de tiempo y con altos estándares de calidad. Los 11 servicios AWS se usaron para su diseño y permitieron integrarla a la totalidad del proyecto. Los elementos y servicios con los que cuenta cumplen con los principios Solid de la programación orientada a objetos, y permiten desacoplar la lógica entre clases y módulos, lo cual facilita la creación de pruebas y garantiza estándares y características de calidad como la mantenibilidad, la eficiencia y la seguridad. En conjunto, hacen que la aplicación sea un producto valioso y de alta calidad para el cliente.

Cabe resaltar que, durante el desarrollo de este proyecto se observaron ciertas problemáticas en la logística de desarrollo, tales como conflictos entre tareas relacionadas que eran trabajadas simultáneamente por diferentes desarrolladores y que involucraban la modificación de la misma parte del código, lo que resultó en un consumo innecesario de tiempo y esfuerzo para su resolución. Con el fin de evitar estos problemas se recomienda una planificación cuidadosa por parte de los desarrolladores, incluyendo la creación de diagramas generales de la aplicación y sus componentes. También, se sugiere asignar estas tareas a un rol específico en el equipo de trabajo para garantizar una gestión eficiente.

Finalmente, este proyecto demuestra que hacer pasantía es una alternativa valiosa para la realización de trabajos de grado, debido a que con esta experiencia se adquiere conocimiento en el ámbito laboral y empresarial; se lleva a cabo una investigación constante similar a la del ámbito académico. Se tiene acceso a problemáticas y necesidades de los clientes que requieren un estudio detallado para la creación e implementación de nuevos conocimientos con el fin de desarrollar soluciones a dichas problemáticas y satisfacer esas necesidades. Además, es una gran oportunidad para hacer relaciones que posibilitan el fortalecimiento del ejercicio profesional del pasante.

VII. TRABAJOS FUTUROS

La implementación de arquitectura Cloud en AWS orientada a microservicios es un tema que no se agota en este trabajo de pasantía, al contrario, existen muchas posibilidades de ampliación en su estudio y una de ella es la siguiente.

Se propone la implementación del "Real-Time Monitoring System" como una plataforma de supervisión en tiempo real. El objetivo principal es lograr una integración fluida y sin errores entre los diferentes equipos de trabajo. Esta solución permitiría evitar la aparición de bugs, lo que redundaría en ahorro de tiempo y costos adicionales para la empresa.

La aplicación del "Real-Time Monitoring System" se extendería a lo largo de todo el proyecto, facilitando la comunicación y comprensión entre los equipos mediante esquemas, diagramas y gráficas, etc. Al contar con una visibilidad completa del estado de los microservicios en tiempo real, todos los participantes en el proyecto podrían tener una comprensión clara de lo que se está desarrollando, evitando posibles conflictos y problemas de integración.

La supervisión constante durante todo el proceso de desarrollo permitiría detectar y resolver rápidamente cualquier error o inconveniente que pueda surgir. Al prevenir la generación de bugs, se aseguraría la funcionalidad y calidad del proyecto, evitando retrasos y mejorando la eficiencia en el desarrollo. Además, esta solución contribuiría a reducir los costos asociados con la corrección y depuración posterior de errores.

En conclusión, la implementación del "Real-Time Monitoring System" en proyectos de arquitectura Cloud en AWS orientados a microservicios garantiza una integración sin problemas y evita la aparición de bugs.

VIII. REFERENCIAS

[1] E. Ortiz Pabón y N. Nagles García, *Gestión de Tecnología e Innovación. Teoría, proceso y práctica*. 2013. Doi: 10.21158/9789587562552.

[2] Ministerio de Salud y Protección social y Ministerio del Trabajo, "Circular Externa 018 de 2020", pp. 1–3, 2020.

[3] Ministerio de trabajo de la república de Colombia, "Ley 2088 de 2021". [En línea]. Disponible en: https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=162970

[4] Y. R. Serrano Molina, "Aws S3 como mecanismo de recuperación ante desastres tecnológicos en pymes", Univ. Dist. Fr. José Calda, 2020.

[5] R. B. Vilchez, "Arquitectura de Back End con Amazon Web Services (Aws) para Sistemas Escolares", Syria Stud., vol. 7, no 1, pp. 37–72, 2015, [En línea]. Disponible en: https://dspace.um.edu.mx/handle/20.500.11972/1043

[6] N. Alzate, "Cloud gaming: a survey", Entre Cienc. e Ing., vol. 10, no 20, pp. 82–91, 2016, [En línea]. Disponible en: http://www.scielo.org.co/pdf/ecei/v10n20/v10n20a12.pdf

[7] Dataprius, "Desventajas Almacenamiento Fisico". 2021. [En línea]. Disponible en: https://blog.dataprius.com/index.php/2020/02/27/almacenamiento-en-servidores-fisicos-frente-al-almacenamiento-en-la-nube/

[8] ACCENTURE AWS, "Grupo Empresarial ACCENTURE AWS", 2021, [En línea]. Disponible en: https://www.accenture.com/cl-es/service-aws-cloud

[9] O. Díaz Rodríguez, "Metodología para diseñar bases de datos relacionales con base en el análisis de escenarios, sus políticas y las reglas del negocio.", vol. 4(3), pp. 197-209., doi: https://doi.org/10.17993/3ctic.2015.43.197-209.

[10] M. K. Gunnulfsen, "Scalable and Efficient Web Application Architectures", Univ. Oslo, pp. 87–102, 2013

[11] B. Engard, "The sides of software development: From Front End vs Back End to full stack.", Softw. Guild., [En línea]. Disponible en: https://www.thesoftwareguild.com/ %0Ablog/front-end-vs-back-end

[12] Salesforce, "Cloud Computing", 2021, [En línea]. Disponible en: https://www.salesforce.com/mx/cloud-computing/#:~:text=De%20una%20manera%20simple%2C%20la,computadora%20personal%20o%20servidor%20local

[13] Podware, "Servidor Cloud", [En línea]. Disponible en: https://blog.prodware.es/servidor-cloud-local-mejor-opcion-pymes/#.YvOrE3ZBw2w

[14] RightScale, “State of the Cloud report: Data to navigate your multicloud strategy”, Estados Unidos: RightScale, 2018, [En línea]. Disponible en: https://www.suse.com/media/report/ rightscale_2018_state_of_the_cloud_report.pdf

[15] Cloud, “Proveedores Nube Pública”, [En línea]. Disponible en: https://www.itsitio.com/ar/proveedores-nube-publica-los-elegidos-las-organizaciones/

[16] C. Rojas Albarracín, G., Páramo Fonseca, J. y Hernández Merchán, “Plataforma computacional sobre Amazon Web Services (AWS) de renderizado distribuido.”, Rev. científica, vol. 3(30), pp. 337-356., 2017, [En línea]. Disponible en: https://doi.org/10.14483/23448350.12362

[17] C. S. T. T, “Servicios Amazon Web Services”, 2018, [En línea]. Disponible en: https://www.clickittech.com.mx/cloud-computing/porque-elegir-la-nube-aws-para-tu-aplicacion-web/

[18] Write a Citix Company, “API”, 2021, [En línea]. Disponible en:

https://www.wrike.com/es/blog/que-es-una-api-necesitas-saber/#:~:text=LasAPIconectandiferentespartes,accedanalamisma%20informaci%C3%B3n

[19] V. Gadge, S. y Kotwani, “Microservice architecture: API gateway considerations.”, 2018, [En línea]. Disponible en: https://www.globallogic.com/pl/gl_news/microservice-architec- ture-api-gateway-considerations/

[20] J. D. López Hinojosa, “Arquitectura de software basada en microservicios para desarrollo de aplicaciones web de la Asamblea Nacional (Tesis de maestría).”, Univ. Técnica del Norte, Ibarra, Ecuador., 2017.

[21] W. E. Salazar Hernández, “Implementación de arquitectura de microservicios utilizando virtualización por sistema operativo (Tesis de licenciatura).”, Universi- dad San Carlos Guatemala, Ciudad Guatemala, Guatemala, 2017.

[22] AWS, “Aplicación monolitica en microservicios”, 2022, [En línea]. Disponible en: https://aws.amazon.com/es/getting-started/hands-on/break-monolith- app-microservices-ecs-docker-ec2/module-three/

[23] Microsoft, “Windows y contenedores”, 2022, [En línea]. Disponible en: https://docs.microsoft.com/es-es/virtualization/windowscontainers/about/

[24] AWS, “Contenedores AWS”. 2021. [En línea]. Disponible en: https://aws.amazon.com/es/containers/services/

[25] K. Ayyapazham, R. y Velautham, “Proficient decision making on virtual machine creation in laaS cloud environment.”, Int. Arab J. Inf., p. Technology, 14(3), 314-323., 2017.

[26] AWS, “Kit de desarrollo de la nube de AWS”, 2022, [En línea]. Disponible en: https://aws.amazon.com/es/cdk/

[27] C. Díaz Alcoleda, “Qué es Azure DevOps”, 2021, [En línea]. Disponible en: https://openwebinars.net/blog/que-es-azure-devops/

[28] Git. (2020). Git. https://git-scm.com/

[29] Atlassian. (2017). Gitflow Workflow | Atlassian Git Tutorial. Atlassian.Com. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[30] Una taza de Java, "Apuntes de Java en español", 2019, [En línea]. http://dar10comyr.blogspot.com/ Disponible en: http://dar10comyr.blogspot.com/2019/07/git-git-flow-github-flow.html

[31] Y. Muradas, "Qué es posman y primeros pasos", 2019, [En línea] Open Wbinars. Disponible en: https://openwebinars.net/blog/que-es-postman/

[32] Amazon Web Services, "Creación de aplicaciones con arquitectura sin servidor", [En línea] www.aws.amazon.com Disponible en: https://aws.amazon.com/es/lambda/serverless-architectures-learn-more/#:~:text=%C2%BFQu%C3%A9%20es%20una%20arquitectura%20sin,la%20administraci%C3%B3n%20de%20los%20servidores.

[33] F. Flores, "Qué es serveless, ventajas y servicios", 2021, [En línea] Open Webinars. Disponible en: https://openwebinars.net/blog/que-es-serverless-ventajas-y-servicios/

[34] Sentrio Connect, "Qué es SonarQube: verifica y analiza la calidad de tu código", 2021, [En línea] https://sentrio.io/ Disponible en: https://sentrio.io/blog/que-es-sonarqube/#%C2%BFQue_es_SonarQube

[35] Amazon Web Services, "Preguntas frecuentes", [En línea] https://aws.amazon.com/ Disponible en: https://aws.amazon.com/es/waf/faq/#:~:text=AWS%20WAF%20es%20un%20firewall,las%20condiciones%20que%20usted%20defina.

[36] Amazon Web Services, "Qué es Amazon CloudFront", [En línea] https://aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/AmazonCloudFront/latest/DeveloperGuide/Introduction.html

[37] Amazon Web Services, "Qué es Amazon S3", [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html

[38] Amazon Web Services, "Qué es Amazon Cognito", [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/cognito/latest/developerguide/what-is-amazon-cognito.html

[39] Amazon Web Services, "Uso de la API de Rest", [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/apigateway-rest-api.html

[40] Amazon Web Services, "Qué es AWS Lambda", [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html

[41] Amazon Web Services, "¿Qué es Amazon DynamoDB?", [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/amazondynamodb/latest/developerguide/Introduction.html

[42] M. Hernandez, "¿Qué es npm?", 2021, Free Code Camp, [En línea] https://www.freecodecamp.org/ Disponible en: https://www.freecodecamp.org/espanol/news/que-es-npm/

[43] NPM, "The test npm package", 2023, NPM, [En línea] https://www.npmjs.com/ Disponible en: https://www.npmjs.com/package/test

[44] NPM, "Npm-audit. Run a Security Audit", NPM Docs, [En línea] https://docs.npmjs.com/ Disponible en: https://docs.npmjs.com/cli/v9/commands/npm-audit

[45] M. López Mendosa, "Qué es un lenguaje de programación", 2020, Open Webinars, [En línea] https://openwebinars.net/ Disponible en: https://openwebinars.net/blog/que-es-un-lenguaje-de-programacion/

[46] Wikipedia. La Enciclopedia Libre, "TypeScript", 2022, Wikipedia, [En línea] https://es.wikipedia.org/ Disponible en: https://es.wikipedia.org/wiki/TypeScript

[47] Amazon Web Services, "¿Qué es JavaScript?", AWS, [En línea] https://docs.aws.amazon.com/ Disponible en: https://aws.amazon.com/es/what-is/javascript/

[48] C. Simoes, "¿Qué es Node.js y para qué Sirve?", 2021, ITDO, [En línea] https://www.itdo.com/ Disponible en: https://www.itdo.com/blog/que-es-node-js-y-para-que-sirve/

[49] Arimetrics, "Qué es Json", [En línea] https://www.arimetrics.com/ Disponible en: https://www.arimetrics.com/glosario-digital/json

[50] Amazon Web Services, "¿Qué es Python?", AWS, [En línea] https://aws.amazon.com/ Disponible en: https://aws.amazon.com/es/what-is/python/

[51] B. Wagner et al, "Serialización (C#)", 2023, Microsoft, [En línea] https://learn.microsoft.com/ Disponible en: https://learn.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/serialization/

[52] Keep Coding, "¿Qué es y para qué sirve el fichero YAML?, 2022, Keepcoding Tech School, [En línea] https://keepcoding.io/ Disponible en: https://keepcoding.io/blog/que-es-y-para-que-sirve-el-fichero-yaml/

[53] CDA Informática, "¿Qué es CRUD?", CDA Soluciones Confiables, [En línea] https://www.cdainfo.com/ Disponible en: https://www.cdainfo.com/es/noticias/182-que-es-crud

[54] Desarrollo Web, "Editores de Código", [En línea] https://desarrolloweb.com/ Disponible en: https://desarrolloweb.com/colecciones/editores-codigo

[55] Microsoft, "Así Es Como Se Crea Software", 2022, Microsoft Visual Studio, [En línea] https://visualstudio.microsoft.com/es/ Disponible en: https://visualstudio.microsoft.com/es/

[56] Amazon Web Services, "¿Qué es Aws Cloud9?, AWS, [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/cloud9/latest/user-guide/welcome.html

[57] M.C.R.A., "Infraestructura Global de aws", 2020, DEV, [En línea] https://dev.to/ Disponible: https://dev.to/maricarmenra/infraestructura-global-de-aws-3djl

[58] Bibik, I. (2018). How to kill the scrum monsterquick start to agile scrum methodology and the scrum master role. Palgrave Macmillan.

[59] Sinnaps, "Metodología XP o Programación Extrema", [En línea] https://www.sinnaps.com/ Disponible en: https://www.sinnaps.com/blog-gestion-proyectos/metodologia-xp

[60] Hayat, F., Rehman, A. U., Arif, K. S., Wahab, K., & Abbas, M. (2019, July). The influence of agile methodology (Scrum) on software project management. In 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) (pp. 145-149). IEEE

[61] Grupo Garatu Development, "Te Contamos el Método Colaborativo más Eficaz en el Desarrollo de Proyectos Estratégicos en la Industria", [En línea] https://development.grupogaratu.com/ Disponible en: https://development.grupogaratu.com/metodologia-scrum-desarrollo-software/

[62] Amazon Web Services, "¿Qué es IAM?", AWS, [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/introduction.html

[63] Amazon Web Services, "Roles de IAM", AWS, [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/id_roles.html

[64] Amazon Web Services, "Políticas y permisos de IAM", AWS, [En línea] https://docs.aws.amazon.com/ Disponible en:

https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/access_policies.html

[65] Keep Coding, "¿Qué es Amazon SQS?", 2022, Keepcoding Tech School, [En línea] https://keepcoding.io/ Disponible en:

https://keepcoding.io/blog/que-es-amazon-sqs/#Que_es_Amazon_SQS

[66] Tic.PORTAL, "Amazon SQS: el Servicio de Cola de Mensajes de AWS", 2022, [En línea] https://www.ticportal.es/ Disponible en: https://www.ticportal.es/temas/cloud-computing/amazon-web-services/amazon-sqs/

[67] Wikipedia La Enciclopedia Libre, "Tocken (informática)", 2022, Wikipedia, [En línea] https://es.wikipedia.org/ Disponible en: https://es.wikipedia.org/wiki/Token_(inform%C3%A1tica)

[68] O. Blancarte, 2017, "Escalabilidad Horizontal y Vertical", [En línea] Oscar Blancarte Software Architect, Disponible en: https://www.oscarblancarteblog.com/2017/03/07/escalabilidad-horizontal-y-vertical/

[69] S. Aguirre, 2023. "Implementación de una Arquitectura Cloud en AWS Orientada a Microservicios para un Cliente de la Empresa ACCENTURE".

[70] Amazon Web Services, “Introducción a Amazon ECS Mediante el AWS CDK”, AWS, [En línea] https://docs.aws.amazon.com/ Disponible en: https://docs.aws.amazon.com/es_es/AmazonECS/latest/developerguide/tutorial-ecs-web-server-cdk.html

[71] Amazon Web Services, “Amazon Cognito Construct Library”, 2023, Construct Hub, [En línea] https://constructs.dev/ Disponible en: https://constructs.dev/packages/@aws-cdk/aws-cognito/v/1.190.0?lang=typescript

[72] Amazon Web Services, “Amazon API Gateway Construct Library”, 2023, Construct Hub, [En línea] https://constructs.dev/ Disponible en: https://constructs.dev/packages/@aws-cdk/aws-apigateway/v/1.190.0?lang=typescript

[73] Amazon Web Services, “AWS APIGatewayv2 Authorizers”, 2023, Construct Hub, [En línea] https://constructs.dev/ Disponible en: https://constructs.dev/packages/@aws-cdk/aws-apigatewayv2-authorizers/v/1.190.0?lang=typescript#introduction

[74] Amazon Web Services, “AWS Lamda Construct Library”, 2023, Construct Hub, [En línea] https://constructs.dev/ Disponible en: https://constructs.dev/packages/@aws-cdk/aws-lambda/v/1.190.0?lang=typescript

[75] Amazon Web Services, “Aws-wafwebacl-cloudfront-module”, 2022, Construct Hub, [En línea] https://constructs.dev/ Disponible en: https://constructs.dev/packages/@aws-solutions-constructs/aws-wafwebacl-cloudfront/v/2.30.0?lang=typescript

[76] Amazon Web Services, “Amazon DynamoDB”, AWS, [En línea] https://aws.amazon.com/ Disponible en: https://aws.amazon.com/es/dynamodb/

[77] Amazon Web Services, “Microservicios”, AWS, [En línea] https://aws.amazon.com/ Disponible en: https://aws.amazon.com/es/microservices/

[78] Assembler Institute of Technology, “Qué es Postaman? Características y ventajas,[En Línea] https://assemblerinstitute.com/ Disponible en: https://assemblerinstitute.com/blog/que-es-postman/

IX. Anexos

TABLA 1. REQUERIMIENTOS FUNCIONALES.

Cod.	Historia	Rol	Criterio de aceptación
RF01	Como usuario del banco deseo acceder a toda mi información por medio de un usuario y contraseña.	Usuario	Al ingresar a la aplicación en la pestaña de información se muestra: tipo de tarjeta (ahorros o crédito), número de tarjeta, procesadores de pago (Visa o Mastercard) y el estado (on - off).
RF02	Como asesor bancario deseo tener los permisos: crear, leer, actualizar y eliminar de la cuenta.	Asesor	Al ingresar a la aplicación, con su cuenta el asesor, generará un token que tendrá los permisos de crear, leer, actualizar, listar con el fin de añadir, editar y eliminar usuarios de la base de datos del banco.
RF03	Como usuario del banco deseo generar un token cuando me autentique para iniciar de forma segura la sesión.	Usuario	Al ingresar a la aplicación, con su respectivo usuario y contraseña, se crea un token que contiene los permisos de inicio de sesión de forma segura.
RF04	Como asesor bancario deseo tener el permiso en la cuenta para realizar la búsqueda general o filtrada de los usuarios.	Asesor	Al ingresar a la aplicación, el asesor se autenticará para tener el permiso de buscar en la base de datos los usuarios de manera general o filtrada individual por su ID.
RF05	Se desea almacenar las lambdas en un zip diferente.	Sistema	Al implementar las lambdas en la arquitectura se almacenarán en un zip distinto, dentro del servicio de AWS S3.
RF06	Se desea que la aplicación soporte ataques XXS	Sistema	Al implementar la arquitectura se tendrá que incorporar el servicio de AWS WAFF para soportar ataques XXS.
RF07	Se desea que la aplicación tenga los repositorios: Infraestructura y Lambdas.	Sistema	Al desplegar la arquitectura se crearán dos repositorios en Azure Devops; uno albergará la infraestructura como código y el otro las lambdas.
RF08	Se desea integrar la base de datos interna a la arquitectura	Sistema	Incorporar la base de datos interna del banco a la arquitectura.
RF09	Se desea agregar los módulos de seguridad del banco a la infraestructura.	Sistema	Integrar a la infraestructura como código todos los módulos de seguridad del banco.
RF10	Se desea realizar los despliegues de la	Sistema	Desplegar toda la arquitectura en Azure Devops.

Cod.	Historia	Rol	Criterio de aceptación
	arquitectura en Azure Devops.		

TABLA 2. REQUERIMIENTOS NO FUNCIONALES.

Cod.	Historia	Rol	Criterio de aceptación
RNF01	Se desea que la aplicación tenga un alto rendimiento y se ejecute con rapidez.	Sistema	Al implementar la arquitectura, se incorporarán las ubicaciones al borde del servicio AWS Cloudfront para disminuir la latencia.
RNF02	Se desea que la arquitectura sea tipo Serverless.	Sistema	Al implementar la arquitectura debe contener solo servicios de AWS que sean tipo serverless (sin servidor).
RNF03	Se desea implementar buenas prácticas de programación.	Sistema	Al implementar la infraestructura como código se deben usar las buenas prácticas de programación exigidas por el banco.
RNF04	Se desea integrar las políticas internas del banco en los despliegues.	Sistema	Incorporar en los despliegues en Pipelines todas las políticas del banco.
RNF05	Se desea utilizar el versionamiento de código con Git	Sistema	utilizar las estrategias de versionamiento de código con Git en Azure Devops
RNF06	Se desea implementar Typescript para toda la infraestructura como código (IaaC).	Sistema	Implementar TypeScript para todo el desarrollo de la infraestructura como código.
RNF07	Se desea implementar el lenguaje de programación Python para el servicio de AWS Lambdas.	Sistema	Implementar Python para la programación del servicio de AWS Lambdas.
RNF08	Se desea realizar las pruebas de funcionamiento a toda la arquitectura.	Sistema	Realizar pruebas a toda la arquitectura para garantizar su correcto funcionamiento.

Cod.	Historia	Rol	Criterio de aceptación
RNF09	Se desea implementar librerías recomendadas y soportadas.	Sistema	Investigar, seleccionar y utilizar librerías recomendadas por AWS y por la comunidad, que sean de calidad y reciban soporte continuo.