Análisis de comportamientos de compra basado en grafos

Infraestructura cloud en AWS y entorno de desarrollo

Grado: Ciencia e Ingeniería de Datos
Universidad: Universidad de Las Palmas de Gran Canaria

Autora: Gisela Isabel Belmonte Cruz

Curso académico: 2025–2026

7 de enero de 2026

Índice

1. Descripción del proyecto

El proyecto *Shopping Graph Analysis* tiene como objetivo el análisis de tickets de compra para identificar relaciones de co-ocurrencia entre productos en entornos de supermercado. Estas relaciones se modelan mediante grafos, donde los nodos representan productos y las aristas reflejan compras realizadas de forma conjunta.

El análisis del grafo permite detectar patrones de consumo relevantes que pueden servir de apoyo en la toma de decisiones relacionadas con la disposición de productos en tienda, el diseño de promociones y el desarrollo de sistemas de recomendación.

2. Arquitectura general del sistema

La Figura 1 muestra la arquitectura general del sistema desplegado en AWS.

Las operaciones de consulta se realizan a través de una API expuesta mediante **API Gateway**, que actúa como punto de entrada para los clientes. Dichas peticiones son gestionadas por una función **AWS Lambda de búsqueda**, encargada de ejecutar consultas sobre la base de datos de grafos y devolver los resultados.

De forma independiente, la ingesta de datos sigue un enfoque orientado a eventos. La subida de archivos `.csv` a un bucket de **Amazon S3** desencadena automáticamente la ejecución de una función **AWS Lambda de ingesta**, responsable del procesamiento y transformación de los datos, sin intervención directa de API Gateway.

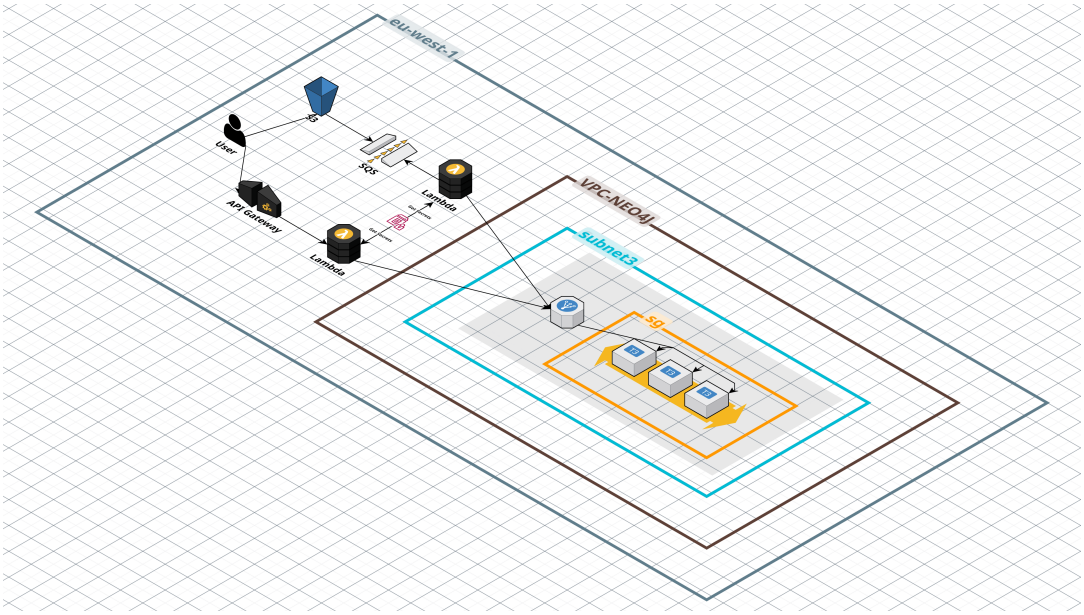

Figura 1: Arquitectura general del sistema desplegado en AWS.

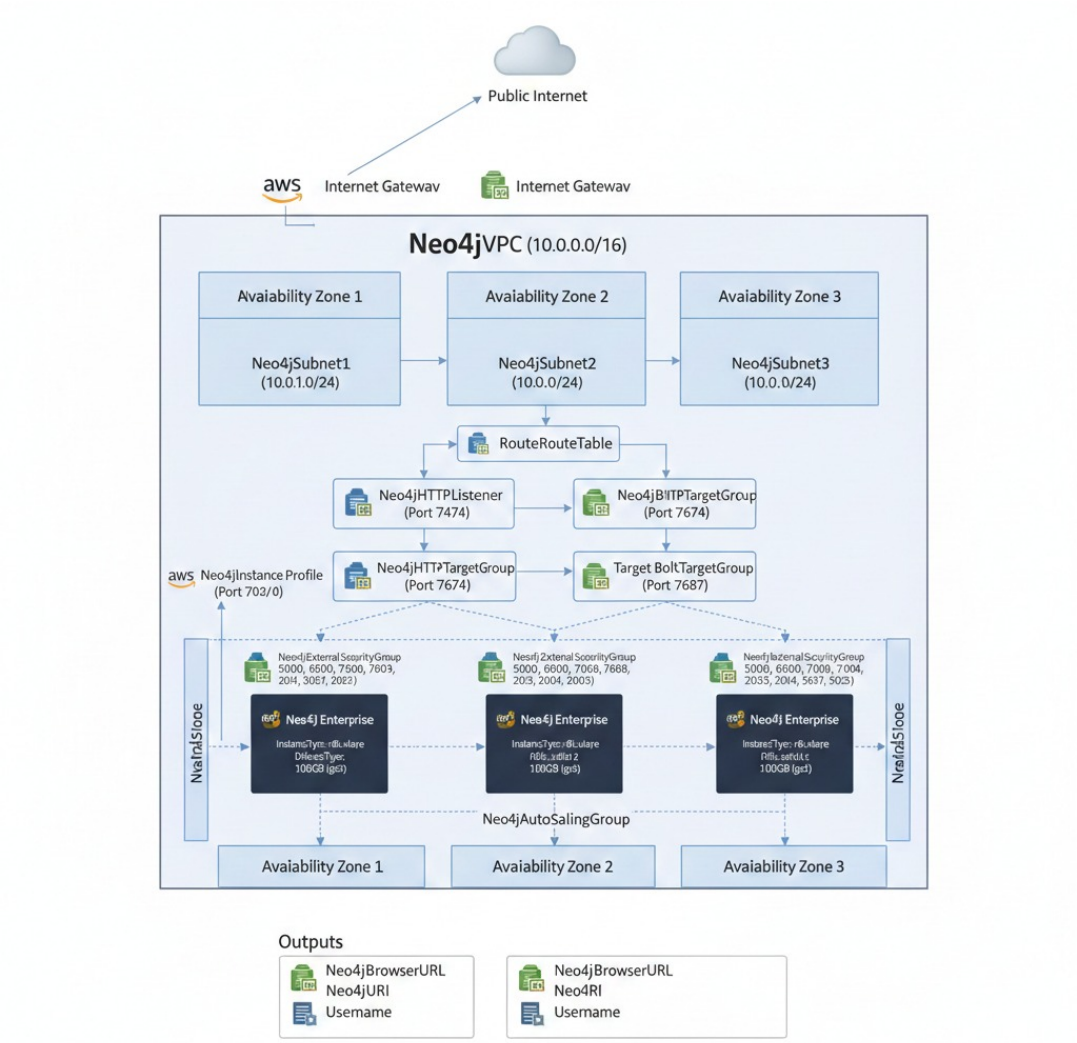

Figura 2: Arquitectura del Network Load Balancer y clúster Neo4j. Imagen generada mediante IA (Gemini) con fines ilustrativos.

La Figura 2 muestra la arquitectura del *Network Load Balancer* configurado para un clúster de **Neo4j Community Edition** desplegado sobre instancias EC2 dentro de una VPC dedicada.

El sistema está distribuido en tres zonas de disponibilidad de la región `eu-west-1` (`eu-west-1a`, `eu-west-1b` y `eu-west-1c`), lo que garantiza alta disponibilidad y tolerancia a fallos ante la caída de una zona completa.

Las instancias EC2 forman parte de un *Auto Scaling Group* definido a partir de una **plantilla de lanzamiento (Launch Template)**, que centraliza la configuración de los nodos y permite una escalabilidad controlada y consistente.

3. Infraestructura Cloud: Cuenta de AWS

Para el desarrollo del proyecto se ha utilizado una **cuenta personal de AWS**, acogida a la capa gratuita (*Free Tier*) y a un programa de **créditos promocionales**,

que proporciona un saldo de **200 dólares para su consumo durante un periodo de 12 meses**. Esta configuración permite trabajar con servicios cloud reales manteniendo el control del gasto.

4. Control de versiones y organización en GitHub

El proyecto se estructura de forma modular dentro de la siguiente organización:

`https://github.com/Shopping-graph-analysis`

5. Metodología y gestión del proyecto

El desarrollo del proyecto se ha organizado siguiendo una metodología **Kanban**, utilizando GitHub Projects para la planificación, seguimiento y priorización de tareas.

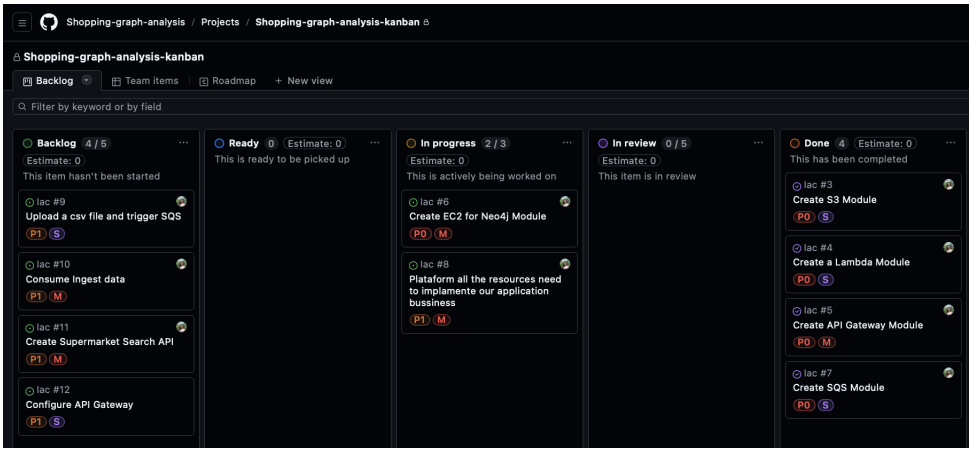

Figura 3: Estado del tablero Kanban durante el desarrollo.

6. Entorno de desarrollo y automatización

6.1. Automatización mediante GitHub Actions

Todo el ecosistema del proyecto se despliega mediante **GitHub Actions**, siguiendo principios de integración y despliegue continuo (CI/CD). La infraestructura y las funciones Lambda se actualizan de forma automática a partir de los cambios en los repositorios.

6.2. CI: generación de artefactos

Dentro del flujo de integración continua (CI), se genera un **artefacto comprimido en formato** `.zip` que contiene el código y las dependencias necesarias de cada función Lambda. Este artefacto es el que se utiliza posteriormente en el proceso

de despliegue, garantizando consistencia entre el código validado en CI y el código ejecutado en el entorno cloud.

6.3. Backend remoto y seguridad

El estado de Terraform se gestiona mediante un backend remoto en **Amazon S3**, utilizando autenticación **OpenID Connect (OIDC)** para la comunicación segura entre GitHub y AWS, eliminando el uso de credenciales estáticas.

6.4. Gestión de errores mediante Dead Letter Queue

Con el objetivo de mejorar la resiliencia del sistema y evitar bloqueos en el procesamiento de eventos, se ha configurado una **Dead Letter Queue (DLQ)** asociada a la cola principal de mensajería utilizada en la ingesta de datos.

El mecanismo de funcionamiento es el siguiente: si un evento no puede ser procesado correctamente y falla en su consumo tras **tres intentos consecutivos**, dicho evento se considera problemático y es redirigido automáticamente a la **Dead Letter Queue**. De este modo, se evita que la cola principal quede bloqueada por un único mensaje erróneo y se garantiza la continuidad del procesamiento del resto de eventos.

La DLQ permite aislar los eventos fallidos para su posterior análisis, depuración o reprocesamiento manual, sin afectar al flujo normal del sistema. Esta estrategia sigue las buenas prácticas recomendadas en arquitecturas orientadas a eventos en AWS, mejorando la tolerancia a fallos y la capacidad de observabilidad del sistema.

6.5. Destrucción controlada de la infraestructura

Como parte del enfoque de automatización y gestión responsable de recursos, se ha implementado un flujo específico en **GitHub Actions** que permite la destrucción controlada de la infraestructura desplegada en AWS. Este proceso se basa en la ejecución de un workflow manual que expone una opción seleccionable (*checkbox*) dentro de la interfaz de GitHub.

Al activar dicha opción, el pipeline ejecuta un **plan de destrucción de Terraform** (`terraform plan -destroy`), mostrando de forma explícita los recursos que serán eliminados. Este paso requiere una **aprobación manual** antes de proceder, lo que introduce una salvaguarda adicional frente a eliminaciones accidentales.

Este mecanismo permite eliminar de forma ordenada y reproducible todos los recursos cloud asociados al proyecto cuando dejan de ser necesarios, facilitando la limpieza del entorno, el control de costes y el cumplimiento de buenas prácticas en entornos de desarrollo y experimentación.

7. Infraestructura de base de datos orientada a grafos

El almacenamiento y análisis de las relaciones entre productos se realiza mediante **Neo4j Community Edition**. Dado el carácter académico del proyecto, no se dispone de licencia para la versión Enterprise, por lo que se utilizan exclusivamente las funcionalidades disponibles en la edición comunitaria.

Aunque la arquitectura se inspira en patrones de alta disponibilidad recomendados por Neo4j, algunas capacidades avanzadas propias de la versión Enterprise no están disponibles. No obstante, la configuración resulta suficiente para validar el modelo de datos y las consultas planteadas.

7.1. Monitorización

Las instancias EC2 del clúster se monitorizan mediante **Amazon CloudWatch**, permitiendo analizar métricas como uso de CPU y tráfico de red.

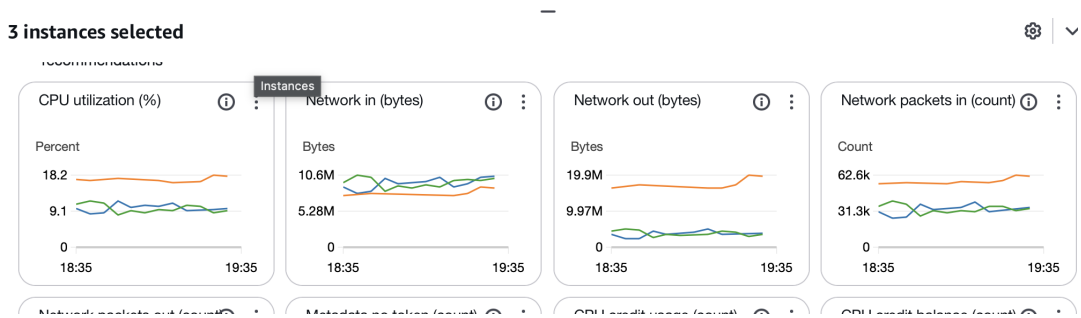

Figura 4: Monitorización de las instancias EC2 del clúster Neo4j.

8. Análisis de costes

El proyecto se ha desarrollado utilizando la capa gratuita (*Free Tier*) de AWS y créditos promocionales disponibles en la cuenta. El coste asociado al uso de instancias EC2 y componentes de red se ha mantenido bajo control durante las fases de despliegue, prueba y validación.

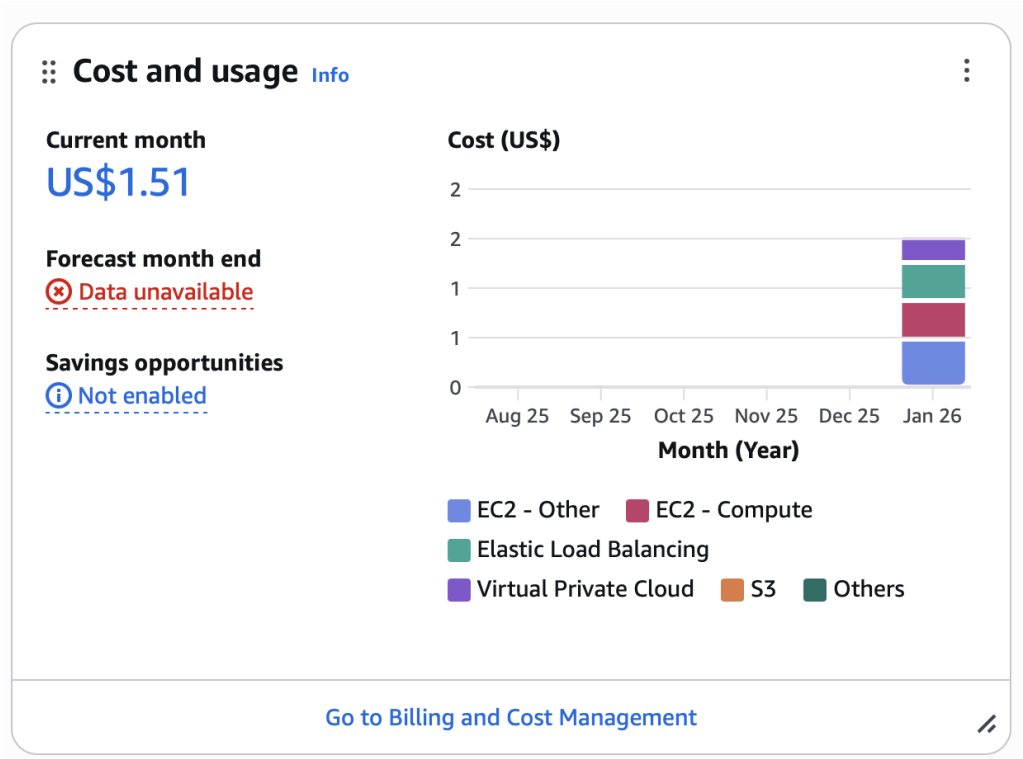

Figura 5: Costes por servicio en AWS durante el desarrollo del proyecto.

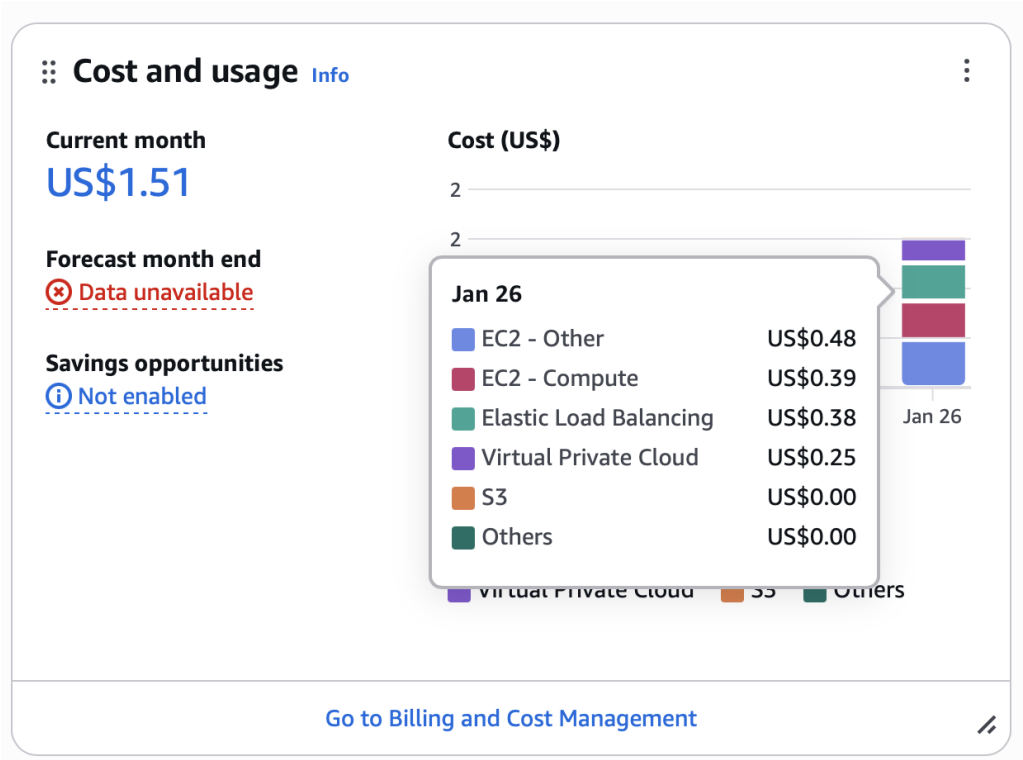

Figura 6: Desglose de costes por servicio en AWS durante el desarrollo del proyecto.

El **Load Balancer** representa uno de los costes más elevados debido a su facturación continua por horas de uso y por volumen de tráfico gestionado, inde-

pendientemente de la carga real del sistema. Por su parte, las **instancias EC2** suponen un coste relevante al tratarse de recursos computacionales activos de forma persistente, necesarios para garantizar la disponibilidad del servicio de base de datos orientada a grafos.

El resto de servicios utilizados, como **Amazon S3**, **AWS Lambda**, **API Gateway** y **CloudWatch**, presentan un impacto económico reducido al estar mayoritariamente basados en modelos de pago por uso y beneficiarse de la capa gratuita (*Free Tier*) disponible en la cuenta.

La arquitectura diseñada resulta adecuada desde el punto de vista económico para un entorno académico y de experimentación.

9. Validación funcional y resultados obtenidos

9.1. Validación de la infraestructura de balanceo

La Figura 7 muestra el estado real del *Network Load Balancer* desplegado en AWS, junto con los grupos de destino y las instancias EC2 asociadas. Se observa que todos los *targets* se encuentran en estado **Healthy**, lo que confirma el correcto funcionamiento del balanceo de tráfico y la disponibilidad del clúster de Neo4j.

Esta configuración permite distribuir las peticiones entrantes entre múltiples nodos, mejorando la tolerancia a fallos y garantizando la continuidad del servicio ante incidencias puntuales en alguna de las instancias.

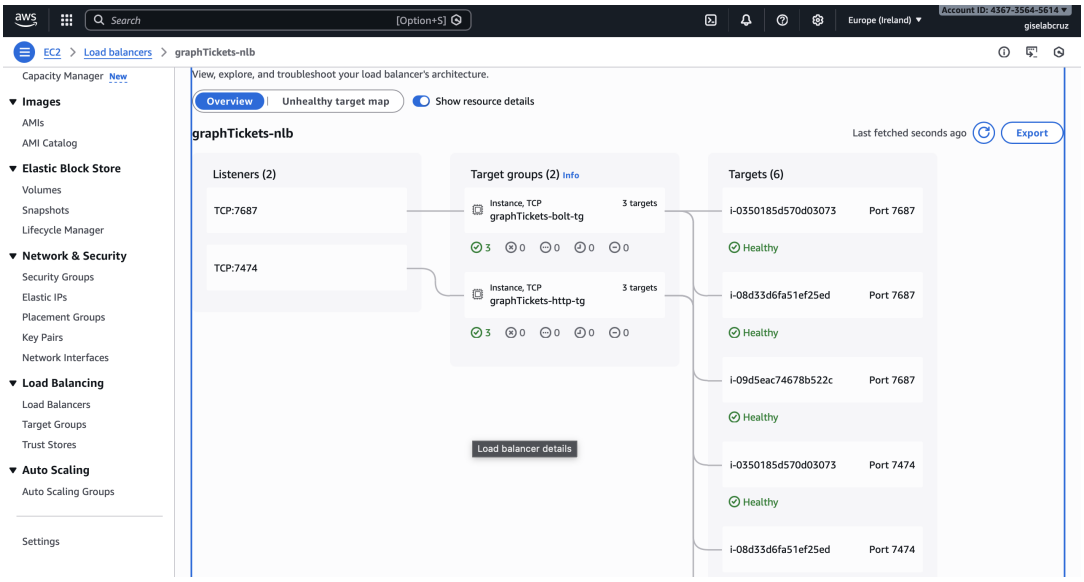

Figura 7: Estado del Network Load Balancer y grupos de destino activos en AWS.

9.2. Pruebas funcionales de la API y sistema de recomendaciones

Para validar el correcto funcionamiento del sistema, se realizaron pruebas sobre el endpoint `/search` expuesto mediante **API Gateway**, el cual invoca una función **AWS Lambda** encargada de consultar la base de datos Neo4j y devolver recomendaciones de productos basadas en relaciones de co-ocurrencia.

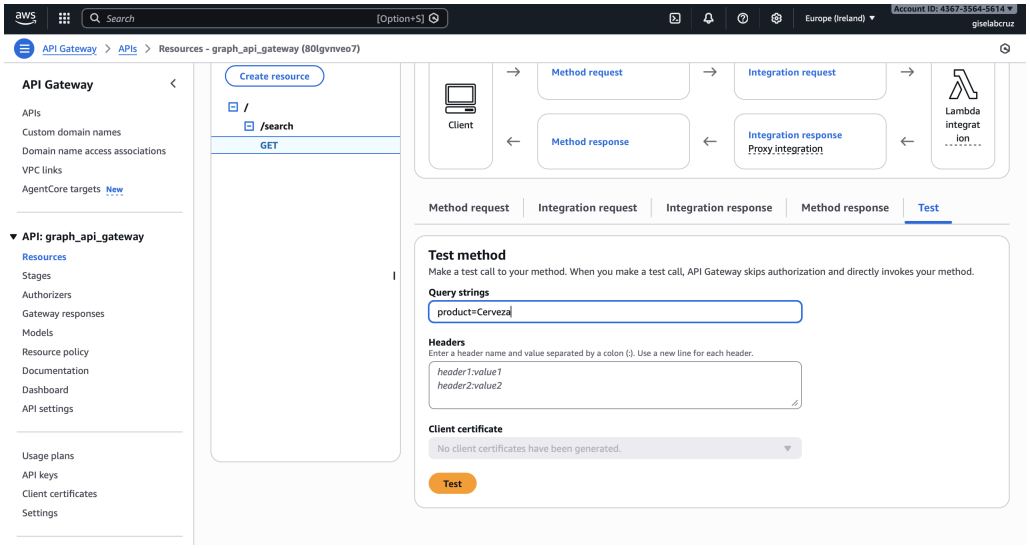

Figura 8: Prueba del endpoint `/search` para el producto *Cerveza*.

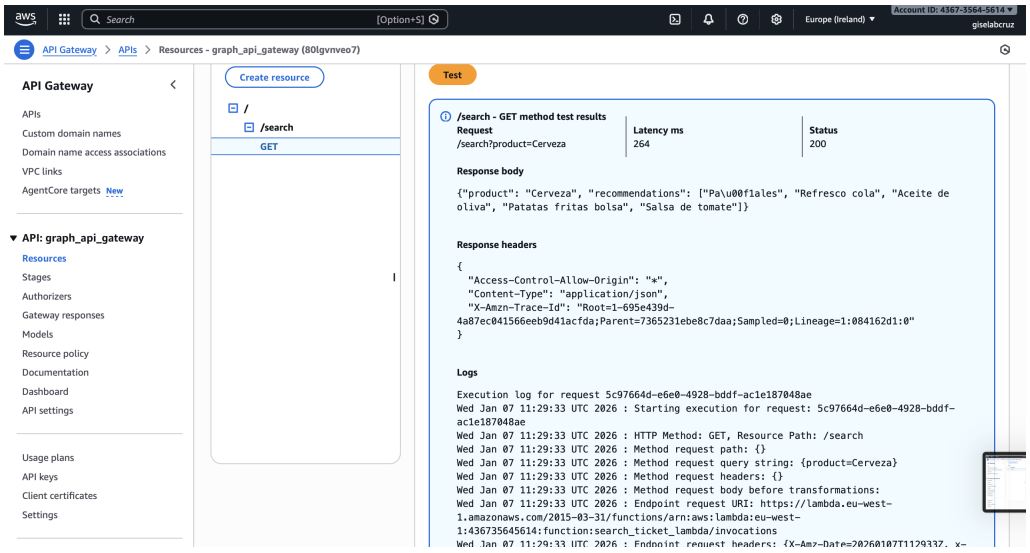

Figura 9: Resultado de recomendaciones asociadas al producto *Cerveza*.

En el caso del producto *Cerveza*, el sistema devuelve recomendaciones como papas fritas, refrescos, salsas y aceite, lo que refleja patrones de compra habituales en contextos de consumo social o eventos informales. Esto puede ser útil para el supermercado, que pondrá los paquetes de papas fritas cerca o incluso en promoción de la cerveza.

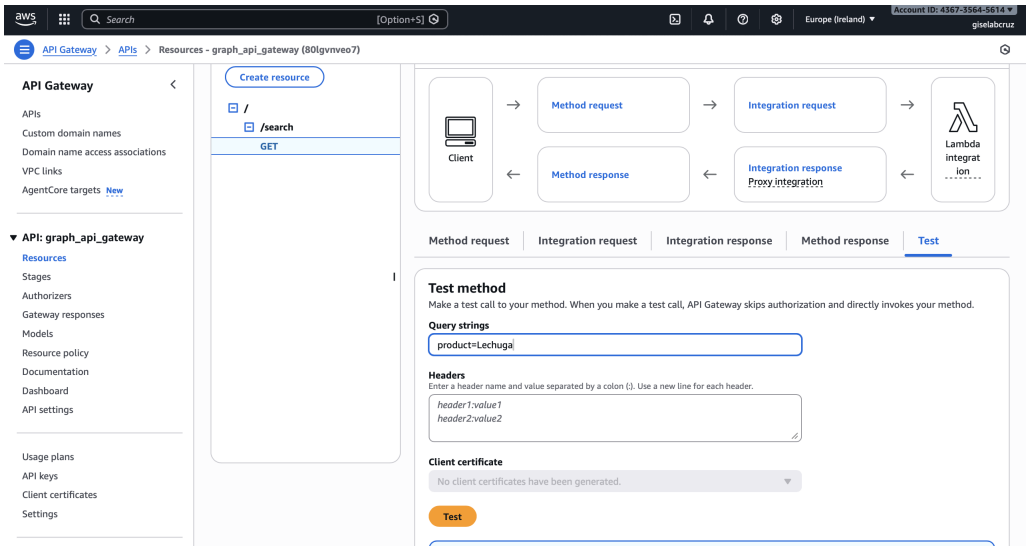

Figura 10: Prueba del endpoint `/search` para el producto *Lechuga*.

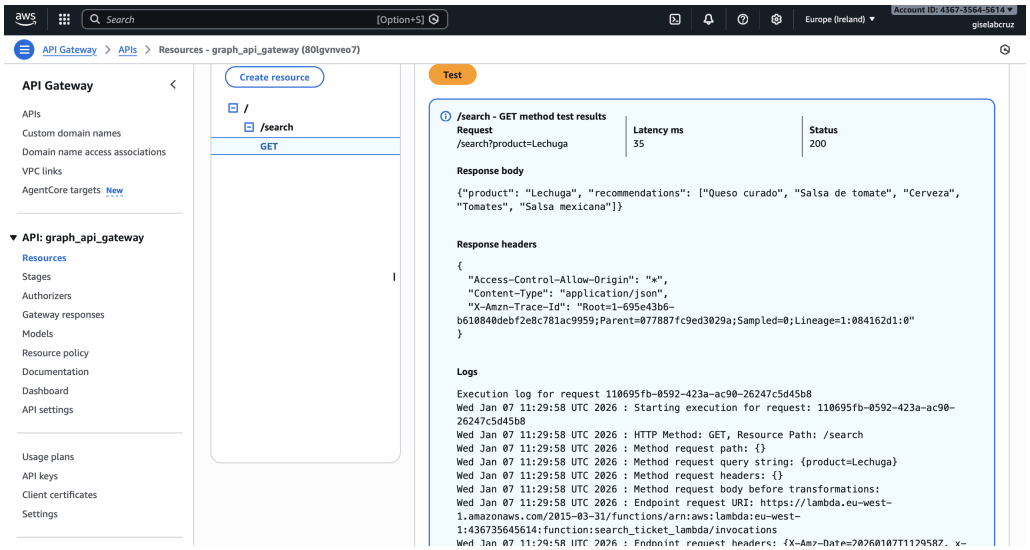

Figura 11: Resultado de recomendaciones asociadas al producto *Lechuga*.

Para el producto *Lechuga*, las recomendaciones incluyen otros ingredientes habituales en una dieta saludable o en la preparación de comidas, como tomates, queso o salsas. Al igual que también aparece la cerveza dentro de las recomendaciones.

9.3. Impacto y aplicabilidad en entornos empresariales

La identificación de relaciones frecuentes entre productos ayuda a comprender mejor cómo compran los clientes y qué artículos suelen adquirir conjuntamente. Esta información resulta especialmente útil para crear promociones más atractivas, mejorar la organización de los productos dentro de la tienda y ofrecer recomendaciones más ajustadas a los hábitos reales de compra.