Práctica Final. Caso de uso

Sistemas y Servicios en la Nube

Master Universitario en Ingeniería Informática

2024 – 2025

Grupo 1

Integrantes:

Alejandro Navarro de la Cruz

Javier Domingo Collado

Alonso Illán Martínez del Santo

Juan Miguel García González

Índice de contenido

Índice de figuras

Introducción

Este proyecto tiene como objetivo desarrollar una solución tecnológica basada en servicios en la nube que permita la gestión eficiente de procesos logísticos y comerciales. A través de la integración de servicios ofrecidos por AWS, la propuesta busca optimizar tareas críticas como la gestión de inventarios y la creación de pedidos asegurando una interacción eficiente entre los distintos componentes del sistema.

La arquitectura planteada incluye el uso de una serie de servicios de AWS seleccionados para garantizar un funcionamiento eficiente, escalable y seguro. Los datos se gestionarán mediante Amazon DynamoDB, donde se almacenará información clave de los productos y pedidos. Las operaciones serán orquestadas por AWS Lambda, con funciones específicas para la gestión de productos, el procesamiento de pedidos y la generación automática de tickets en formato PDF. Además, el sistema incorporará Amazon API Gateway para la comunicación entre los usuarios y las funciones Lambda, asegurando un acceso controlado y eficiente a las funcionalidades del sistema.

Por otro lado, se empleará Amazon Simple Notification Service (SNS) como medio de notificación para la gestión de alertas relacionadas con el stock y la generación de tickets de compra. Para la interacción del usuario, se desplegará una aplicación web basada en AWS Amplify, que facilitará la visualización y gestión de datos en tiempo real. También, se pretende que la solución haga uso de un bucket S3 para el almacenamiento seguro de tickets y otros documentos generados.

Se ha automatizado la orquestación de todos los servicios AWS a través de la herramienta AWS SAM, para simplificar la implementación, gestión y escalabilidad de nuestra aplicación mediante una plantilla YAML donde se describen los recursos necesarios que permiten realizar despliegues consistentes y automatizados.

La propuesta se fundamenta en un enfoque modular y escalable que permite cubrir las necesidades operativas actuales mientras se adapta al crecimiento futuro del sistema. Con un diseño centrado en la eficiencia, el sistema no solo asegura una integración fluida entre los distintos componentes, sino que también optimiza los costos operativos gracias al modelo de pago por uso de los servicios en la nube y aprovechando el Free Tier de algunos servicios de AWS.

A lo largo de este proyecto se representa una solución innovadora que aprovecha las capacidades de AWS para transformar los procesos comerciales, proporcionando una plataforma robusta y eficiente que responde a las necesidades del entorno empresarial moderno.

Arquitectura y descripción de componentes

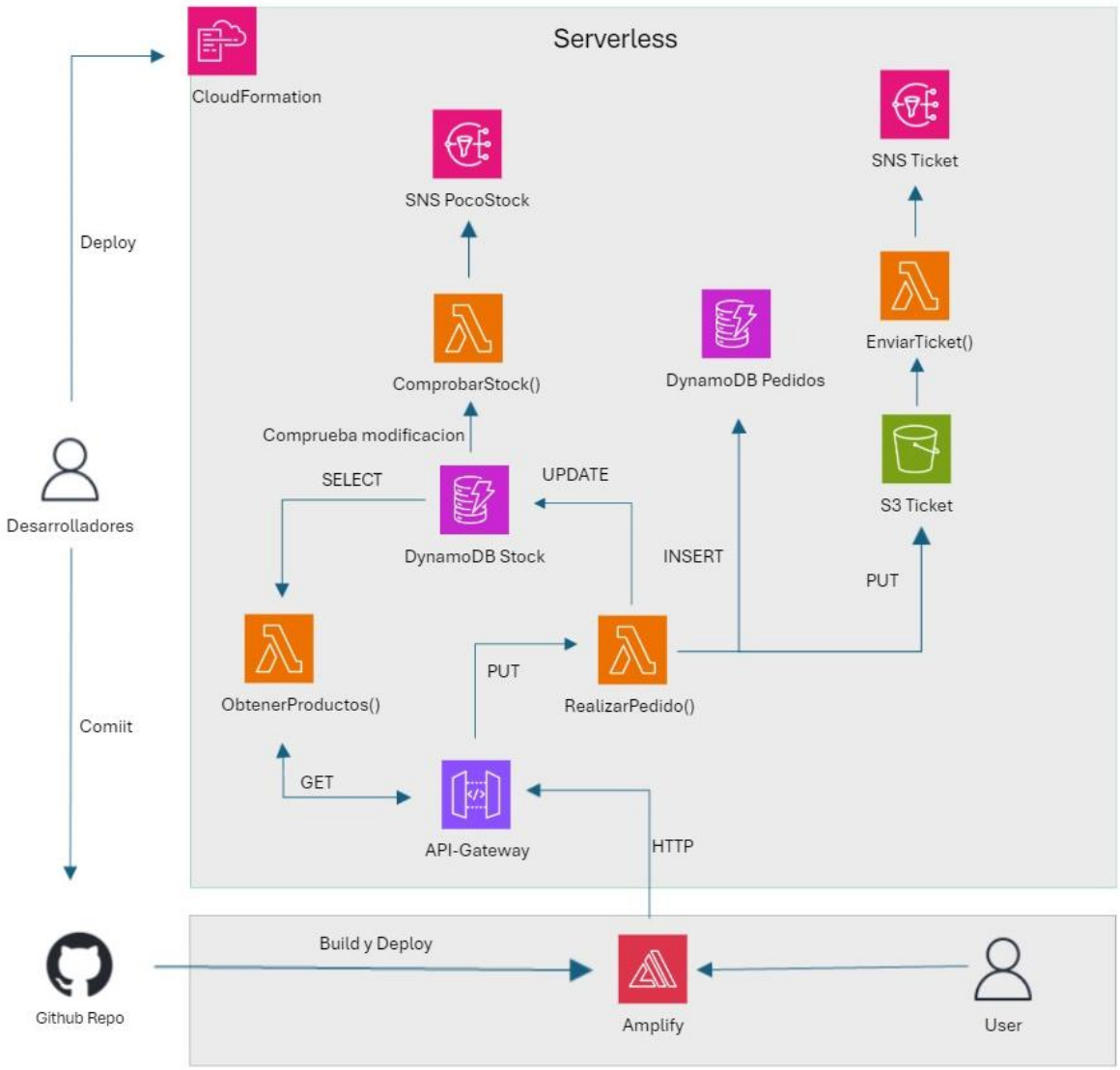

Figura 1. Arquitectura del caso de uso

- GitHub Repo y Despliegue con Script
  - El código fuente de la aplicación se almacena en un repositorio de GitHub, donde los desarrolladores gestionan versiones y actualizaciones. El proceso de construcción y despliegue de la infraestructura y funciones se ejecuta mediante un script automatizado.
- Despliegue con CloudFormation
  - CloudFormation facilita la creación y configuración de la infraestructura, incluyendo tablas DynamoDB, funciones Lambda y temas SNS. Esta herramienta asegura la replicabilidad y gestión centralizada de los recursos necesarios para la solución.

- DynamoDB Stock
    - Es la base de datos que almacena el inventario de productos, proporcionando soporte para las siguientes operaciones:
        - Consultar productos disponibles desde la interfaz o mediante la función Lambda que comprueba el stock.
        - Actualizar el stock una vez que se ha procesado un pedido.
- DynamoDB Pedidos
    - Es utilizada para registrar los pedidos realizados. Las operaciones de inserción de datos (INSERT) son gestionadas a través de la función Lambda RealizarPedido().
- Funciones Lambda, Las funciones Lambda ejecutan las principales operaciones del sistema:
    - ObtenerProductos(): Obtiene datos desde DynamoDB Stock (GET) y puede ser accedido mediante el API Gateway.
    - RealizarPedido(): Actualiza el stock en DynamoDB Stock (UPDATE),registra pedidos en DynamoDB Pedidos (INSERT) y genera los tickets. También interactúa con el frontend a través de solicitudes HTTP.
    - ComprobarStock(): Publica mensajes en el tema SNS PocoStock para alertar sobre niveles bajos de inventario cuando dectecta cambios en DynamoDB Stock.
    - GenerarTicket(): Crea tickets de pedidos procesados y los almacena en un bucket S3 para garantizar su disponibilidad.
    - EnviarTicket(): Publica notificaciones en SNS Ticket para informar a los usuarios que los tickets han sido generados.
- Simple Notification Service (SNS), SNS es usado para el manejo de notificaciones importantes dentro del sistema:
    - SNS PocoStock: Envía alertas automáticas cuando el stock de productos alcanza niveles bajos.
    - SNS Ticket: Informa a los usuarios sobre la creación de tickets de pedidos procesados.
- S3 Ticket
    - Un bucket S3 actúa como repositorio para almacenar los tickets generados por la función RealizarPedido(). Este almacenamiento es seguro y fácilmente accesible para otras partes del sistema.
- API Gateway
    - Este servicio actúa como el punto de entrada principal de la aplicación, permitiendo a los usuarios interactuar con el backend a través de rutas preconfiguradas que redirigen a las funciones Lambda correspondientes.
- Amplify
    - Amplify es utilizado para hospedar la interfaz de usuario, proporcionando acceso a los usuarios finales.

Detalles de la implementación

En esta sección se va a analizar la implementación de la arquitectura propuesta en la sección anterior, describiendo como la información fluye a través de la arquitectura para lograr el objetivo de la aplicación. De esta manera, veremos como la implementación debe garantizar una experiencia de usuario fluida, de manera que se ha de automatizar la gestión de productos y la generación de recibos, gracias a una arquitectura desacoplada, serverless y escalable.

- **Interacción con el usuario**: El usuario accede a la aplicación web a través de AWS Amplify (dependiendo del stock). Para insertar los productos en la base de datos pulsa un botón que se encuentra en la interfaz que hace una petición a la función lambda que se encarga de insertar los productos en la base de datos. Una vez recibida la información, selecciona los productos que desea comprar y presiona un botón para realizar la compra. Una vez procesada la compra, le llegará un recibo a su correo electrónico como comprobante de compra.
- **Gestión de Stock**: Gracias a la comprobación que realizamos del stock de los productos seleccionados en la compra, actualizamos su stock que esta almacenado en DynamoDB.
- **Avisos de Stock**: Cuando un producto baje de una franja establecida en su stock, se enviará a través de SNS un aviso al mánager de la tienda, de manera que pueda realizar las gestiones necesarias para reponer el stock.
- **Creación y almacenamiento de comprobantes**: Gracias a la lambda de compra, se pueden generar PDF que contengan esta información para los clientes. Por otra parte, los bucket S3 almacenan los comprobantes de compra de los clientes, de manera que sean fácilmente recuperables por los clientes y trabajadores de la tienda.

De esta manera, podemos comprobar que la implementación descrita demuestra la eficacia de la arquitectura, donde se ha creado una solución desacoplada, permitiendo que el sistema gestione de manera eficiente los recursos y asegure la capacidad de respuesta de los usuarios y el negocio.

Cuando ejecutamos el comando *npm run sam*, que inicia el proceso de despliegue de nuestros servicios AWS, nos solicita una serie de parámetros:

- **Configuración de AWS**: Access key, Secret Access, Region name y AWS Session token.
- **Configuración del despliegue:**
  - *Stack Name*: sam-app
  - *AWS Region*: us-east-1
  - *Parameter RoleARN*: ARN del rol LabRole en AWS AIM
  - *Parameter EmailSubscription*: Correo electrónico al que llegarán mensajes de los SNS.
  - *Parameter GitHubOauthToken*: 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
  - *Todos los demás parámetros*: “Y” salvo el *SAM configuration file* y el *SAM configuration environment* que los dejamos por defecto pulsando “Enter”

Análisis de prestaciones y costo

Para el análisis de prestaciones y costo vamos a ir analizando servicio a servicio de los utilizados en la aplicación y algunos de los parámetros más importantes que hemos supuesto para estimar el costo de ejecutar la solución en la región us-east-1 durante 12 meses. Para realizar la estimación total del costo de la solución, hemos utilizado la plataforma de Amazon Princing Calculator [1], que permite ir configurando diferentes servicios y añadirlo.

Para el servicio de AWS Amplify, se han supuesto 30 minutos por build al mes para actualizaciones frecuentes de la aplicación, 1GB de almacenamiento para la app y 2GB servidos teniendo en cuenta un tráfico inicial bajo. Además, hemos supuesto unas 10 solicitudes por hora y 500ms por solicitud, representando una carga baja típica para un sistema en una etapa temprana.

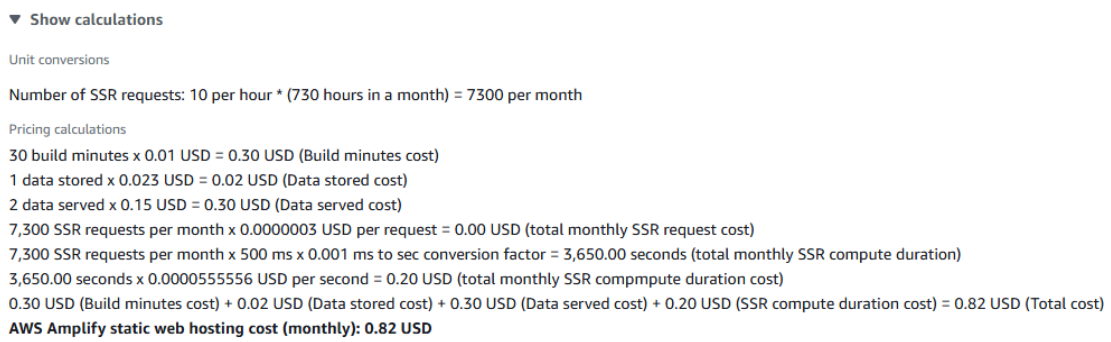

Figura 2. Cálculo de costos del servicio Amplify

El servicio de API Gateway lo hemos configurado con 50.000 solicitudes mensuales para reflejar un tráfico inicial moderado con el uso de Amplify. Suponemos que la tienda online tiene unos 10.000 usuarios mensuales, y cada usuario realiza promedio unas 5 interacciones con el backend por sesión, contando con consultas de productos, creación de pedidos o actualizaciones de datos.

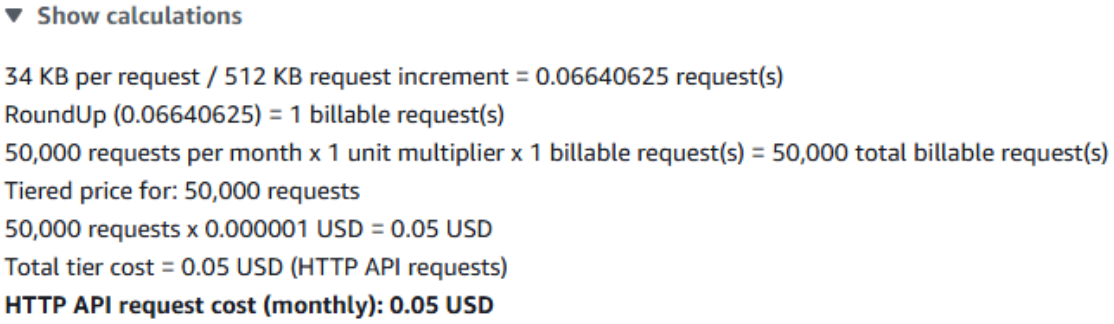

Figura 3. Cálculo de costos de servicio API Gateway

El servicio de AWS Lambda lo hemos configurado con unas 50.000 solicitudes mensuales, 500 ms de duración por solicitud y 512MB de memoria y almacenamiento, lo que suponemos adecuado para un sistema de tienda online en una etapa inicial. Dado que las solicitudes y el tiempo de ejecución estimados están dentro de los límites del Free Tier, hemos seleccionado esa opción por ser la más rentable.

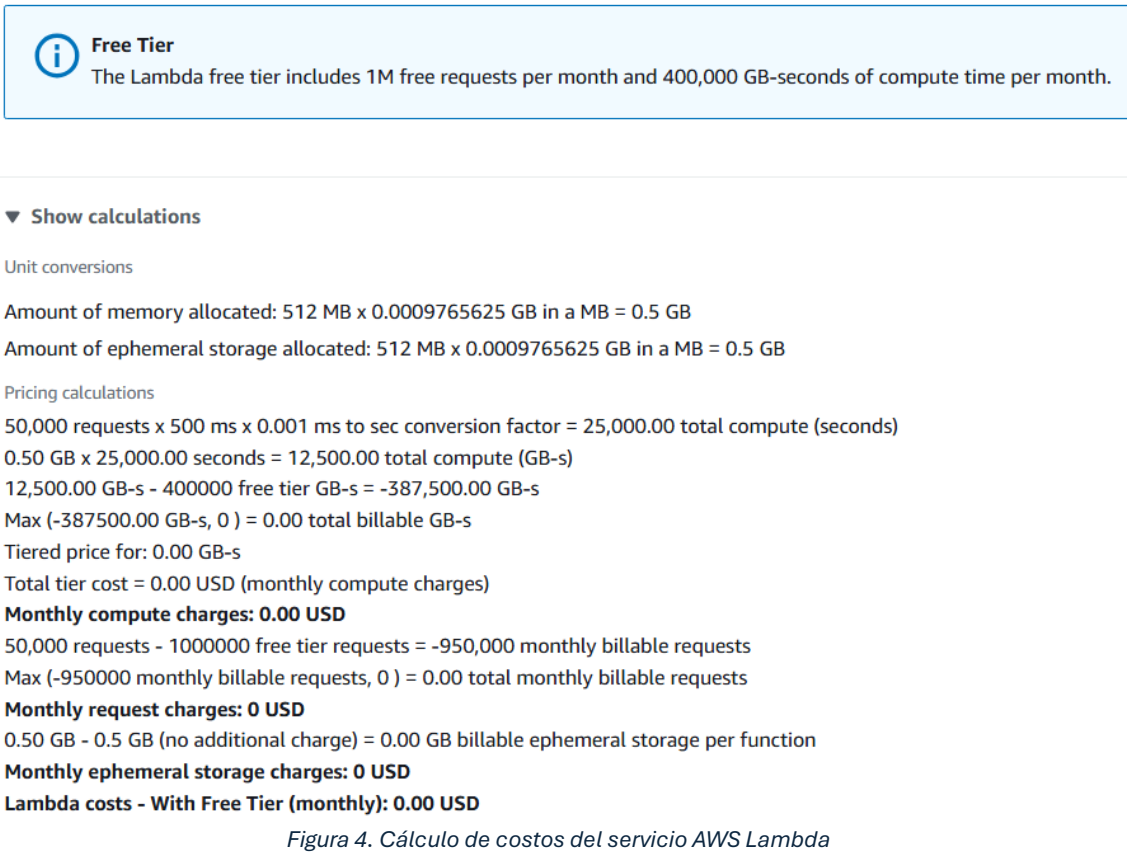

Figura 4. Cálculo de costos del servicio AWS Lambda

El servicio DynamoDB hemos supuesto una tabla estándar y un almacenamiento de 1GB, porque pensamos que es lo adecuado al tratarse de texto plano para almacenar información sobre productos y pedidos en la tienda online. Además, se han estimado que las operaciones de lectura y escritura serán entre 100 y 400 por segundo en los picos bajos y altos de actividad respectivamente.

Se ha estimado un almacenamiento de 5GB en Amazon S3 Standard para guardar documentos como tickets o facturas generadas a través de las lambdas del sistema. Además, hemos considerado unas 50.000 solicitudes mensuales porque al iniciar el sistema queremos recoger los productos existentes en la base de datos y, cuando un cliente compra algún producto queremos actualizar su stock y crear un pedido.

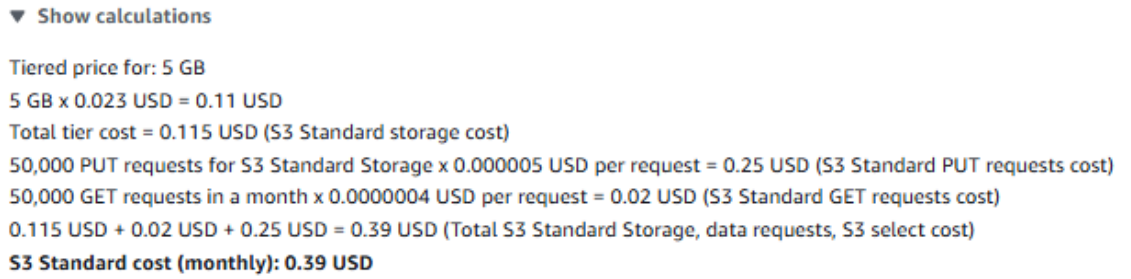

Figura 5. Cálculo de costos del servicio AWS S3

El servicio SNS lo utilizaremos para enviar notificaciones tanto a los clientes con el ticket de la compra y al manager cuando el stock de un producto sea inferior a un número. En este caso, hemos supuesto 50.000 notificaciones para los clientes y 500 para el reabastecimiento de los productos al manager.

▼ Show calculations

50,050 requests - 1000000 free tier requests = -949,950.00 billable SNS requests per month
Max (-949950.000000 requests, 0 requests) = 0.00 requests
Tiered price for: 50,050 calls
1,000 calls x 0.00 USD = 0.00 USD
49,050 calls x 0.00002 USD = 0.98 USD
Total tier cost: 0.00 USD + 0.98 USD = 0.98 USD (EMAIL/EMAIL-JSON Notifications cost)
SNS Requests and Notifications cost (monthly): 0.98 USD

Figura 6. Cálculo de costos de servicio AWS SNS

Finalmente, vamos a analizar el resumen general de la estimación del costo de los servicios de AWS anteriormente mencionados. Como se puede observar en la , en AWS hay una gran cantidad de servicios con muestras gratuitas, lo que nos permite optimizar los costos de ciertos servicios. Por ejemplo, servicios como AWS Lambda y SNS son bastante económicos porque no superan los límites del Free Tier, permitiendo aprovechar la cuota gratuita de solicitudes y procesamiento.

El total mensual estimado es relativamente bajo con 28.63 USD mientras que la estimación para los próximos 12 meses asciende a 523.56 USD. El mayor gasto proviene del servicio de DynamoDB, debido al almacenamiento y las solicitudes de lectura y escritura, así como en S3 por los costos de almacenamiento. Pese a eso, creemos que es un costo razonable dada la infraestructura creada y todos los recursos que pretendemos utilizar en la solución.

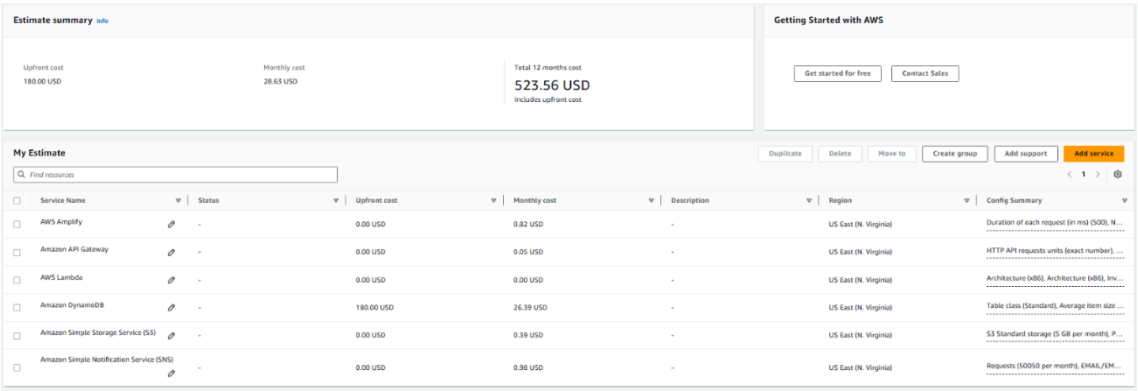

Figura 7. Resumen de la estimación de costos de los servicios de AWS

Bibliografía

[1] A. W. Services, «AWS Pricing Calculator,» AWS, 2024. [En línea]. Available: https://calculator.aws/#/. [Último acceso: 10 Enero 2025].