UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA DE COMPUTACIÓN

IMPLEMENTACIÓN SERVERLESS DEL ECOMMERCE SHOPIFY EN AMAZON WEB SERVICES

Trabajo de titulación previo a la obtención del
Título de Ingenieros en Ciencias de la Computación

AUTORES: JOSÉ FRANCISCO MACAS MELO
STALIN DAVID PILLAJO MASACHE

TUTOR: JULIO RICARDO PROAÑO ORELLANA

Quito – Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, José Francisco Macas Melo con documento de identificación N° 1726396672 y Stalin David Pillajo Masache con documento de identificación N° 1754065116; manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 04 de marzo de 2024

Atentamente,

José Francisco Macas Melo
1754065116

Stalin David Pillajo Masache
1726396672

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, José Francisco Macas Melo con documento de identificación N° 1726396672 y Stalin David Pillajo Masache con documento de identificación N° 1754065116, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico : “Implementación serverless del ecommerce shopify en amazon web services”, el cual ha sido desarrollado para optar por el título de: Ingenieros en Ciencias de la Computación, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Quito, 04 de marzo de 2024

Atentamente,

José Francisco Macas Melo

1754065116

Stalin David Pillajo Masache

1726396672

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Julio Ricardo Proaño Orellana con documento de identificación N°. 0103909412, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: IMPLEMENTACIÓN SERVERLESS DEL ECOMMERCE SHOPIFY EN AMAZON, realizado por José Francisco Macas Melo con documento de identificación N° 1726396672 y Stalin David Pillajo Masache con documento de identificación N° 1754065116, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 04 de marzo de 2024

Atentamente,

Ing. Julio Ricardo Proaño Orellana, MSc.
0103909412

ÍNDICE DE CONTENIDO

Introducción **1**

Problemática **3**

Alcance **4**

Justificación **5**

Objetivos **6**

1.1. Objetivo General *6*

1.2. Objetivos Específicos *6*

CÁPITULO I **7**

Marco Teórico **7**

1.3. Cloud Computing *7*

1.4. ¿Qué es Serverless? *8*

1.4.1 Características 10
1.4.2 Ventajas y Desventajas 11
1.4.3 Casos de Uso 12
1.4.4 Su Futuro 13
1.4.5 E-commerce 13
1.4.6 Plataformas personalizables para e-commerce 14
1.4.7 Ventajas y Desventajas 14
1.4.8 Shopify 15

CÁPITULO II **17**

Marco Metodológico **17**

1.5. Requerimientos Funcionales y no Funcionales *17*

1.6. Casos de Uso *18*

1.7. Diagrama de Actividades *23*

1.8. Herramientas de Desarrollo *33*

1.9. Arquitectura General *36*

1.10. Implementación de Base de Datos en AWS RDS *38*

1.10.1 Modelado 44

1.11. Desarrollo de Servicios para Capturar Webhooks de Shopify *56*

1.11.1 Servicio Broker 57

1.11.2 Servicio de Clientes 62
1.11.3 Servicio de Productos 68
1.11.4 Servicio de Ordenes 71
1.11.5 Servicio de Consulta 75

1.12. Desarrollo Servicio API Gateway en AWS 79
1.12.1 Recurso Lista de Clientes 85
1.12.2 Recurso Plantilla de Correo 88
1.12.3 Recurso Envió de Correo 91
1.12.4 Servicio de Registro de Eventos de Envió de Correo 95
1.12.5 Recurso Métricas de Envió de Correos 99

1.13. Consumo de Servicios definidos en AWS con App Remix 102
1.13.1 Creación e Inicialización de Proyecto Shopify/Remix 103
1.13.2 Configuración y Comunicación de Eventos Shopify (Webhooks) 104
1.13.3 Desarrollo de Recomendación de Productos 106
1.13.4 Desarrollo de Envió de Recomendaciones por Correo Electrónico 108
1.13.5 Desarrollo de Visualización de Métricas de Envió 109

CÁPITULO III 111

Resultados 111

1.14. Pantalla de Métrica de Envió de Correo Electrónicos 111

1.15. Pantalla de Recomendaciones Automáticas para Varios Clientes 112

1.16. Pantalla de Recomendaciones para un Cliente 114

1.17. Envió de Recomendaciones de Productos con Código de Descuento 116

Conclusiones 118

Recomendaciones 119

ÍNDICE DE TABLAS

Tabla 1 Especificación de Caso de Uso 001 19
Tabla 2 Especificación de Diagrama de Caso de Uso 002 20
Tabla 3 Especificación de Caso de Uso 003 21
Tabla 4 Especificación de Diagrama de Caso de Uso 004 22
Tabla 5 Especificación de Caso de Uso 005 23
Tabla 6 Flujo de Procesos 001 25
Tabla 7 Flujo de Procesos 002 27
Tabla 8. Flujo de Procesos 003 29
Tabla 9 Flujo de Procesos 004 31
Tabla 10 Flujo de Proceso 005 33
Tabla 11 Servicios de Amazon Web Services 34
Tabla 12 Librerías implementadas 34
Tabla 13 Frameworks utilizados 35
Tabla 14 Servicios y Herramientas de terceros utilizados 35
Tabla 15 Configuración de Servicio RDS 39
Tabla 16 Configuración Función Lambda RDS Connection 41
Tabla 17 Tabla de Código Función Lambda 001 43
Tabla 18 Diccionario de Datos 001 46
Tabla 19 Diccionario de Datos 002 47
Tabla 20 Diccionario de Datos 003 48
Tabla 21 Diccionario de Datos 004 49
Tabla 22 Diccionario de Datos 005 50
Tabla 23 Diccionario de Datos 006 51
Tabla 24 Diccionario de Datos 007 52
Tabla 25 Tabla de Procedimientos Almacenados 53
Tabla 26 Tabla de Eventos 001 57
Tabla 27 Configuración cola SQS Webhook Máster 59
Tabla 28 Configuración Función Lambda Event Broker 60
Tabla 29 Tabla de Código Función Lambda 002 62
Tabla 30 Configuración cola SQS Slave Customer 63
Tabla 31 Configuración Función Lambda Event Customer 65
Tabla 32 Tabla de Código de Función Lambda 003 67

Tabla 33 Configuración cola SQS Slave Products.. 68
Tabla 34 Configuración Función Lambda Event Product 69
Tabla 35 Tabla de Código de Función Lambda 004 71
Tabla 36 Configuración Cola SQS WebHook Slave Orders... 72
Tabla 37 Configuración Función Lambda Event Customer 73
Tabla 38 Tabla Código de Función Lambda 005 75
Tabla 39 Configuración Cola SQS Query... 76
Tabla 40 Configuración Lambda RDS Query To 77
Tabla 41 Tabla de Código Función Lambda 006 79
Tabla 42 Configuración servicio API Gateway 80
Tabla 43 Configuración Servicio AWS Simple Email Service..................................... 82
Tabla 44 Tabla de recursos API REST.. 84
Tabla 45 Configuración Función Lambda Customer List 85
Tabla 46 Tabla de Código Función Lambda 007 87
Tabla 47 Configuración Función Lambda Email Template.. 88
Tabla 48 Tabla Código Función Lambda 008... 91
Tabla 49 Configuración Función Lambda Deliver Email... 92
Tabla 50 Tabla Código Función Lambda 009... 94
Tabla 51 Configuración de AWS SES configurations sets 95
Tabla 52 Configuración AWS SNS SES Deliver.. 96
Tabla 53 Configuración Función Lambda Event Mailer 97
Tabla 54 Tabla Código Función Lambda 010... 98
Tabla 55 Configuración Función Lambda Mail Metrics 100
Tabla 56 Tabla Código Función Lambda 011 102
Tabla 57 Tabla de directorio y ficheros relevantes 104
Tabla 58 Características de objeto para suscripción de webhooks en Shopify App 105
Tabla 59 Tabla Código para petición al recurso Customers 106
Tabla 60 Tabla Código para de recomendaciones de productos 107
Tabla 61 Tabla Código Generación de Código de Descuento 108
Tabla 62 Tabla Código de envió de Recomendaciones 109
Tabla 63 Tabla Código para Petición al Recurso Mail Metrics 110

ÍNDICE DE FIGURAS

Figura 1 Diagrama de Caso de Uso 001 18
Figura 2 Diagrama de Caso de Uso 002 19
Figura 3. Diagrama de Caso de Uso 003 20
Figura 4 Diagrama de Caso de Uso 004 21
Figura 5 Diagrama de Caso de Uso 006 22
Figura 6 Diagrama de Actividades 001 24
Figura 7 Diagrama de Actividades 002 26
Figura 8 Diagrama de Actividades 003 28
Figura 9 Diagrama de Actividades 004 30
Figura 10 Diagrama de Actividades 005 32
Figura 11 Arquitectura General del Proyecto Técnico 37
Figura 12 Diagrama de Arquitectura Servicio RDS 38
Figura 13 Servicio RDS Inicializado 40
Figura 14 Función Lambda vinculada a la VPC del servicio RDS 42
Figura 15 Código Función Lambda RDS Connection 43
Figura 16 Diagrama Entidad Relación de la Base de Datos 45
Figura 17 Diagrama de Arquitectura para procesamiento de Webhooks 56
Figura 18 Arquitectura Servicio Event Broker 58
Figura 19 Regla de bus de eventos del servicio Event Broker 58
Figura 20 Servicio SQS de Destino 59
Figura 21 Patrón de filtración de Eventos de Cola SQS Webhook Master 60
Figura 22 Función Lambda Event Broker 61
Figura 23 Código Función Lambda Event Broker 62
Figura 24 Arquitectura Servicio de Clientes 63
Figura 25 Plantilla de datos para transformación del servicio Pipe de evento clientes. 64
Figura 26 Función Lambda Event Custome 66
Figura 27 Código Función Lambda Event Customer 67
Figura 28 Arquitectura Servicio Productos 68
Figura 29 Plantilla de datos para transformación del servicio Pipe de evento productos 69
Figura 30 Función Lambda Event Product 70
Figura 31 Código Función Lambda Event Product 71
Figura 32 Arquitectura servicio Event Order 72

Figura 33 Plantilla de datos para transformación del servicio Pipe de evento productos 73
Figura 34 Función Lambda Event Order.. 74
Figura 35 Código función Lambda Event Orders 75
Figura 36 Arquitectura servicio query.. 76
Figura 37 Función Lambda RDS Query To.. 78
Figura 38 Código Función Lambda RDS Query To.. 79
Figura 39 Diagrama arquitectura servicio API Gateway... 80
Figura 40 Servicio API Gateway/API REST 81
Figura 41 Dominio en proveedor GoDaddy... 82
Figura 42 Servicio AWS Simple Email Service 83
Figura 43 Arquitectura Recurso Customers 85
Figura 44 Función Lambda API Customer List 86
Figura 45 Código Función Lambda API Customer List.. 87
Figura 46 Arquitectura Recurso Create Template 88
Figura 47 Función Lambda SES Email Template 89
Figura 48 Código Función Lambda Email Template 90
Figura 49 Arquitectura del Recurso Deliver Email 91
Figura 50 Función Lamnda SES Deliver Email... 93
Figura 51 Código Función Lambda SES Deliver Email 94
Figura 52 Arquitectura para captura de eventos en servicio SES...................................... 95
Figura 53 Función Lambda Event Mailer 97
Figura 54 Código Función Lambda Event Mailer.. 98
Figura 55 Arquitectura Recurso Mail Metrics.. 99
Figura 56 Función Lambda Mail Metrics.. 101
Figura 57 Código Función Lambda Email Metrics.. 102
Figura 58 Estructura de Carpetas Proyecto App Remix Shopify 103
Figura 59 Objecto para suscripción a webhooks.. 105
Figura 60 Código para petición al recurso Customers 106
Figura 61 Código para generación de recomendaciones de Productos 107
Figura 62 Código para creación y registro de Código de descuento................................ 108
Figura 63 Código para envió de recomendaciones a correos electrónicos....................... 109
Figura 64 Código para realizar petición al recurso Mail Metrics......................................110

Figura 65 Pantalla Métricas de envió de correo electrónicos de Aplicación Integrada en Shopify 112
Figura 66 Pantalla de lista seleccionable de contactos para recomendaciones de productos 113
Figura 67 Pantalla de formulario para ingreso de información de recomendación......114
Figura 68 Pantalla de recomendación de productos seleccionable para un único cliente. 115
Figura 69 Pantalla para ingreso de información de las recomendaciones....................116
Figura 70 Correo electrónico de recomendación de producto con código de descuento 117

RESUMEN

Serverless es un tipo de arquitectura que no requiere de la administración de un servidor, la cual se puede implementar en cualquier tipo de proyecto con el fin mejorar, factores claves como administración, monitoreo, seguridad, escalabilidad etc. Con esta implementación se permite entender el funcionamiento lógico de este modelo arquitectural, como la dependencia entre servicios que funcionan conjuntamente para optimizar procesos o funcionalidades de diversos proyectos.

Este trabajo se enfoca en la creación de un prototipo de aplicación local en un ambiente de e-commerce específicamente de la plataforma de Shopify por medio de una arquitectura serverless. En el cual se integra exitosamente un generador de recomendaciones automáticas de productos para clientes y un sistema eficiente de envío de correos masivos.

Palabras clave: arquitectura serverless, Shopify, e-commerce

ABSTRACT

Serverless is a type of architecture that does not require server management and can be implemented in any project to improve key factors such as administration, monitoring, security, scalability, etc. This implementation allows understanding the logical functioning of this architectural model, including the interdependence of services working together to optimize processes or functionalities in various projects.

This work focuses on creating a prototype of a local application in an e-commerce environment, specifically on the Shopify platform, using a serverless architecture. It successfully integrates an automatic product recommendation generator for customers and an efficient email delivery system.

Keywords*:* serverless architecture, Shopify, e-commerce

INTRODUCCIÓN

En un mundo digital donde la Internet y las compras en línea predominan, existen plataformas de comercio electrónico (e-commerce) que hacen posible comprar productos o servicios con un solo clic. Como resultado, la mayoría de la población con acceso a una computadora o dispositivo móvil y conexión a Internet puede realizar compras en línea desde cualquier lugar y en cualquier momento.

Esta tendencia ha impulsado a que empresas de todos los tamaños busquen formas innovadoras de ofrecer sus productos y servicios en línea, para mantenerse competitivas y satisfacer las expectativas de un mercado en constante evolución.

En este contexto, la arquitectura sin servidor (serverless) emerge como una solución potencial, la cual ofrece escalabilidad, aumentando la eficiencia de estas plataformas, así como la reducción de costos de mantenimiento, la gestión de recursos más eficiente, automatización y mejora de funcionalidades.

Este proyecto que tiene como propósito implementar una arquitectura sin servidor en una plataforma e-commerce de Shopify, haciendo uso de los servicios proporcionados por Amazon Web Services (AWS).

A continuación, se presentan las diferentes configuraciones para la base de datos alojada en el servicio AWS RDS, continuando con el detalle de configuraciones para la conexión entre Shopify y AWS a través de "Webhooks". Posteriormente, se aborda la configuración e implementación para la obtención de datos mediante el servicio de API Gateway para alimentar a un generador de recomendaciones de productos automático. Al mismo tiempo, se estructuró una aplicación App Remix que actúa como puente entre todos los servicios utilizados en el proyecto, este puente permitirá envío masivo de correos a clientes. Finalmente se acopla este generador de recomendaciones dentro de la

funcionalidad de correos masivos para un despacho más efectivo de recomendaciones que beneficiaran a clientes del e-commerce Shopify. Todo esto desplegado en un prototipo de aplicación local.

Este desarrollo del proyecto no solo proporcionara una guía conceptual para la transición hacia una arquitectura serverless en un entorno de Shopify, sino también los métodos y casos de uso, ofreciendo una perspectiva práctica que puede ser aplicada para desarrolladores, ingenieros de software y tomadores de decisiones en el ámbito empresarial.

La investigación concluye con la evidencia de esta arquitectura puesta en marcha y funcionando con todo sus servicios activados y sincronizados al igual que las funcionalidades ya mencionadas operativas y en funcionamiento. Asegurando que los lectores obtengan una comprensión integral del proceso y de cómo la adopción de una arquitectura serverless en un e-commerce de Shopify con AWS puede conducir a una operación comercial más ágil y económicamente eficiente.

PROBLEMÁTICA

En el cambiante panorama del comercio electrónico, donde plataformas en línea como Shopify desempeñan un papel fundamental, surge la necesidad de abordar las limitaciones intrínsecas de la arquitectura cliente-servidor. Aunque este modelo ha demostrado eficacia en la mayoría de los sitios web, su aplicación en entornos específicos de e-commerce revela desafíos significativos.

La problemática que motiva esta investigación se centra en la necesidad de demostrar la viabilidad y eficacia de la implementación de una arquitectura serverless en un entorno de e-commerce, específicamente en el marco de Shopify. La arquitectura cliente – servidor, prevalente en la mayoría de las plataformas, ha demostrado limitaciones en términos de eficiencia para abordar ciertos procesos y funcionalidades críticas.

El proyecto tiene como objetivo superar estas limitaciones mediante la adopción de una arquitectura sin servidor, La clave radica en la creación de un prototipo de aplicación local integrado en la infraestructura de Shopify con una arquitectura serverless, implementando de manera efectiva tanto un generador de recomendaciones automáticas como un sistema de envío de correos a través de esta arquitectura sin servidor.

ALCANCE

El alcance de este proyecto se orienta a la implementación de un prototipo de aplicación local integrada en Shopify con una arquitectura serverless. Dado el amplio espectro de funcionalidades dentro de esta plataforma, el enfoque principal del proyecto recae exclusivamente en la creación de dos herramientas funcionales, las cuales serán apoyadas y desarrolladas meramente con servicios de Amazon Wed Services (AWS) las cuales son: un generador de recomendaciones automático para clientes de la plataforma y un sistema de envío de correos.

La implementación práctica se centra en la integración de una arquitectura serverless con estas herramientas para demostrar su fusión dentro del entorno de Shopify. Los resultados obtenidos, medidos mediante criterios de evaluación que incluyen la eficiencia en la generación de recomendaciones y la efectividad del sistema de envío de correos, contribuyen significativamente a proporcionar una comprensión más profunda de la aplicación práctica y los beneficios potenciales de esta reconocida arquitectura en un entorno de e-commerce. Además, ayuda a enriquecer el conocimiento existente en este campo de cloud computing y proporcionar recomendaciones prácticas para su desarrollo y ejecución efectiva.

JUSTIFICACIÓN

En la actualidad, la arquitectura serverless no recibe tanta atención en comparación con otras arquitecturas de sistemas distribuidos. Esta falta de conocimiento es comprensible dado que la arquitectura tiene menos de 9 años de existir. Como resultado, muchos desarrolladores desconocen su potencial para aplicaciones en diversas páginas web, como los e-commerce y de sus capacidades y oportunidades para la creación de herramientas dentro de estos sitios web.

Basándose en esta premisa, se implementa una arquitectura serverless en un contexto especifico, como un generador de recomendaciones automáticas de productos y el envío de correos a los clientes del e-commerce Shopify. Esta perspectiva abre una oportunidad significativa para potenciar la eficiencia y la funcionalidad de estos procesos críticos en un entorno de un e-commerce.

El generador de recomendaciones automáticas de productos, al depender de servicios que funcionan de manera conjunta en un modelo serverless, puede optimizar la generación de sugerencias personalizadas para los usuarios, mejorando así la experiencia de compra y la retención de clientes. De manera similar, la implementación de serverless en él envió masivo de correos permite una gestión eficiente y escalable de las comunicaciones, asegurando una interacción efectiva con la base de clientes.

Esta justificación se apoya en la premisa de que el adoptar una arquitectura serverless, no solo se pretende mejorar aspectos técnicos y operativos, sino también se permite una comprensión más profunda de las interdependencias entre servicios, fortaleciendo la capacidad de optimizar procesos, funcionalidades, herramientas clave en un entorno de e-commerce

OBJETIVOS

1.1. OBJETIVO GENERAL

Desarrollar una arquitectura serverless en un e-commerce de Shopify utilizando servicios de Amazon Web Services (AWS)

1.2. OBJETIVOS ESPECÍFICOS

- Investigar y analizar los servicios computacionales idóneos para la implementación de una arquitectura serverless disponibles en Amazon Web Services.
- Implementar con los servicios de AWS una funcionalidad de envío de correos para clientes del e-commerce.
- Implementar un servicio de recomendaciones de productos con servicios de AWS en la plataforma de Shopify.
- Evaluar el comportamiento de cada funcionalidad implementada con servicios AWS en la aplicación integrada en Shopify.

CÁPITULO I

MARCO TEÓRICO

1.3. CLOUD COMPUTING

Según la Corporación Internacional de Máquinas de Negocio (IBM) (Boss, Malladi, Quan, & Legregni, 2001), cloud computing es tanto una plataforma como un tipo de aplicación. Estas aplicaciones basadas en la nube realizan el aprovisionamiento, configuración, reconfiguración y desaprovisionamiento de servidores, almacenamiento y servicios según las demandas del momento. De igual manera, su rentabilidad se incrementa al mejorar la utilización de recursos.

Se conoce que existen dos categorías utilizadas para clasificar el cloud computing, según el límite de servicio y según el tipo de servicio. El límite de servicio hace referencia al alcance de la información y recursos que se pueden obtener, clasificándose como nube publica, privada o híbrida. Por otro lado, el tipo de servicio son los diseños para refinar procesos como: "Infraestructura como servicio" (IaaS), "Plataforma como servicio" (PaaS), "Software como servicio" (SaaS) y "Funciones como servicio" (FaaS) (Mell & Grance, 2011; Jinfeng, Zhenpeng, Xin, & Xuanzhe, 2023). Los servicios ofrecidos por cloud computing son caracterizados como independientes, reutilizables y altamente portátiles (Blake & Wei, 2010).

Cabe mencionar que estos servicios son proporcionados por proveedores como Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, Alibaba Cloud, IBM Cloud, entre otros.

Cloud computing ofrece la posibilidad de un diseño sin servidores con todas estas herramientas y servicios enfocándose en el mejoramiento y optimización en el desarrollo de aplicaciones o proyectos, un claro ejemplo de este diseño es Serverless.

1.4. ¿QUÉ ES SERVERLESS?

Serverless o computación sin servidor, es un tipo de arquitectura que no necesita de un servidor central para su funcionamiento. Esta arquitectura es posible por la creación de eventos que desencadenan en acciones para ejecutar y gestionar servicios de aplicaciones. Estos llamados eventos son funciones individuales que se comunican entre varios servicios dependiendo de su enfoque. La utilización de serverless evita preocuparse por la administración y mantenimiento de la infraestructura que está presente en servidores comunes. Así mismo, (Manoj, 2019) se refiere a serverless como una metodología de programación que habilita a cualquier persona para escribir y ejecutar código sin la necesidad de preocuparse por la gestión de servidores.

Este diseño aplicativo cuenta con dos modelos de servicio, "Funciones como Servicio" (FaaS), incorpora código de ejecución breve dentro de las funciones, donde también se pueden crean desencadenadores para la realización de tareas específicas. Además, esto permite a los desarrolladores construir y controlar la lógica de sus aplicaciones. Este se complementa con "Backend como servicio" (BaaS), donde se ofrece servicios de almacenamiento, modificación y eliminación dentro de las bases de datos preexistentes o hechas a la medida del programador, estas son fáciles de utilizar ya que son proporcionados por los proveedores que son los administradores principales. BaaS y FaaS no requieren recursos de gestión por parte de clientes (HOSSEIN, AHMAD, & PAYAM, 2022). Esta definición concuerda con (Jinfeng, Zhenpeng, Xin, & Xuanzhe, 2023; Khol, 2014) que mencionan a BaaS como servicio de nube personalizada que se relaciona con FaaS para simplificar el desarrollo de la funcionalidad de la lógica del código (backend).

La combinación de ambos modelos da lugar a una arquitectura sin servidor que se centra en eventos para el procesamiento individual de estos, apoyándose con herramientas como Application Program Interface (API) que facilita la comunicación e interacción entre dos o más aplicaciones o servicios. Esto permite que la arquitectura sin servidor se active para proporcionar instancias aisladas bajo demanda.

En base a esta premisa se conoce que la arquitectura se centra en dos fases: la programación funcional y el servicio de ejecución de funciones.

Programación funcional. En esta fase, los desarrolladores generan sus funciones conforme a las directrices establecidas por los proveedores de plataformas de arquitectura sin servidor en un entorno local. Luego, despliegan estas funciones en la plataforma mediante líneas de comandos. Durante este proceso, la plataforma primero almacena las funciones en una base de datos y envía las ejecuciones de las funciones a un repositorio, para luego proporcionar a los desarrolladores las URL correspondientes (Zijun, Linsong , Jiagan , & Quan , 2022).

Servicio de Ejecución de Funciones. En esta segunda fase se puede utilizar la URL asignada o un evento de activación preconfigurado para llamar a la función, esto sucede a través de una puerta de conexión que utiliza una API, proporcionada por la plataforma. El balanceador de carga o el programador de la plataforma recupera estas funciones desde una base de datos y prepara un entorno de ejecución, así se obtiene el tiempo de respuesta de la función desde un repositorio remoto (Zijun, Linsong , Jiagan , & Quan , 2022).

Esta arquitectura sin servidor fusiona estas dos fases para facilitar la reducción de responsabilidades asociadas a la gestión de servidores. También, permite el escalamiento

automático, la transmisión en eventos, aplicaciones sin estado y aplicación de tarifas basadas en uso.

1.4.1 Características

En el siguiente análisis, se explora más a detalle las características de serverless.

Pago por uso. La operatividad de serverless se basa en la utilización de servicios en función de su rendimiento. Siguiendo esta idea, el costo se estima solo por los recursos que están en uso, sin importar si están en ejecución (Zijun, Linsong , Jiagan , & Quan , 2022).

Servicio medido. Esta característica está relacionada con el pago por uso ya que, gestiona de manera automática y optimiza el empleo de los recursos. Adaptándose a la demanda del servicio para evaluar y ajustar la asignación de recursos según sea necesario.

Sin administración de servidores. Como ya se ha mencionado en párrafos anteriores, esta arquitectura no necesita de una administración directa del usuario o cliente que lo vaya a utilizar. (Shillaker & Pietzuch, 2020) respaldan esta característica indicando que libera a los usuarios de laborales referentes al aprovisionamiento, gestión y mantenimiento de servidores porque la responsabilidad integral de la infraestructura recae en el proveedor.

Falta de estado. A diferencia de las aplicaciones con estado donde existe un almacenamiento constante que guarda estados pasajeros de ejecución para ser recordados posteriormente. Las aplicaciones sin estado, al no tener un almacenamiento continuo, emplean contenedores que permanecen activos por un breve periodo de tiempo. Estos contenedores permiten a los usuarios generar instancias adicionales de manera sencilla (Shillaker & Pietzuch, 2020; V, 2020).

1.4.2 Ventajas y Desventajas

En esta sección se mencionan las ventajas de serverless en la gestión y desarrollo de las aplicaciones.

Ventajas

Escalabilidad flexible. Un producto desarrollado en serverless maneja un escalado automático, esto significa que, dependiendo de la demanda, sea mayor o menor, la infraestructura que da soporte se autoajusta. Además, esta ventaja permite al desarrollador no responsabilizarse por ajustar o planificar la capacidad de cómputo que necesita su desarrollo (Bocanegra, 2023; CNCF, 2022).

Bajo Coste. Se evita el pago por cada código que se prepara y no recibe ningún tipo de petición para ser ejecutado, esto se refiere a que no se cobra por los denominados tiempos muertos (Bocanegra, 2023).

Alta disponibilidad. Esta ventaja señala que serverless está disponible en todas las regiones, a toda hora y en todo momento.

Como cualquier otra tecnología, serverless no está exenta de puntos negativos. Es por eso, que en esta sección se explicaran alguna de estas desventajas.

Desventajas

Falta de supervisión. Para este diseño sin servidor, es más complicado el desplegar y monitorear el comportamiento de las múltiples funciones por que cuenta con varios servicios que trabajan distribuidamente.

Inicio tardío. El proceso de activación para la ejecución de las funciones pasa por un lapso de inactividad hasta reactivarse. Este periodo de inactividad resta tiempo para las demás funciones. Gias y otros (Casale & Gias, 2021) lo define como el problema de

inicio en frio (o cold start) además recomienda que se debe tomar en cuenta para diseñar una arquitectura serverless.

Dependencia del proveedor. Al manejarse con proveedores que facilitan los servicios, serverless depende de la administración de plataformas de terceros. Por lo tanto, si existe un déficit en la gestión de estas plataformas, la arquitectura se verá directamente afectada o en el peor de los casos dejará de funcionar.

1.4.3 Casos de Uso

Este diseño sin servidores aporta con diferentes soluciones factibles en desarrollo de pruebas piloto, realidad aumentada, comercio electrónico, redes sociales.

A continuación, se contemplan algunos escenarios de uso de esta arquitectura.

Internet de las cosas (IoT). Dentro de este campo serverless beneficia el tratamiento de datos que provienen de dispositivos dotados de conexión a Internet y cierta inteligencia de software (Rodríguez, y otros, 2020). Como lo menciona Hassan y otros (B. Hassan, A. Barakat, & I. Sarhan, 2021), se puede aprovechar la arquitectura serverless en este campo, para el procesamiento de datos bajo demanda y la ejecución de tareas intensivas en recursos. Esto se logra al ejecutar la misma función de manera paralela en varios puntos de procesamiento (nodos), donde cada instancia opera en una partición más pequeña de los datos originales.

Comercio electrónico. Al aplicar serverless en una página web de alta demanda, como son los sitios web de compra en línea (e-commerce), se mejora la eficiencia en el manejo de entrada y salida a peticiones, optimizando el uso del ancho de banda, siempre y cuando los recursos informáticos cumplan con el rendimiento requerido (B. Hassan, A. Barakat, & I. Sarhan, 2021). Este enfoque contribuye a prevenir la acumulación de peticiones, evitando afectar al rendimiento de las páginas. Cuando estas peticiones se

realizan, se activa un servicio que identifica su función, procede a descomprimir el paquete, lo carga en un contenedor y, finalmente lo deja disponible para su ejecución (Zanon, 2017).

1.4.4 Su Futuro

Al ser una tecnología reciente en el ámbito tecnológico y administrativo de servicios, con un desarrollo de no más de 9 años, su nivel de madurez no ha estado a la altura de su competidor principal que es la arquitectura cliente-servidor. A pesar de esta particularidad serverless se llega a considerar una evolución lógica para la innovación de arquitecturas sin servidor (Camargo, Rozo, Ponzo, & Gonzaléz, 2023).

En la actualidad, el modelo serverless ha avanzado a pasos agigantados y se ha implementado en varias áreas de la tecnología. Los e-commerce o comercios electrónicos han optado por la aplicación de esta arquitectura para incrementar su escalabilidad, su rendimiento y su disponibilidad.

1.4.5 E-commerce

El e-commerce o comercio electrónico, se basa en procesos que permiten realizar transacciones comerciales a través de plataformas electrónicas. Las transacciones que se realizan en dichas plataformas son la compra de productos, la exploración de catálogos de productos y la ejecución de pagos (Yudiyanto, 2023). Estos procesos se materializan mediante el intercambio de datos, protocolos seguros y utilización de servicios para pago electrónico.

Según Nevárez (Nevárez Montes, 2014), e-commerce opera como una modalidad de intercambio de bienes y servicios realizados por medio de las Tecnologías de la Información (TI) y respaldado por plataformas. Este modelo de negocio, gracias a su baja inversión económica, permite a tiendas físicas (off-line) expandirse, llegando a nuevos clientes en todo el mundo (Ramos, 2012).

Las transacciones comerciales mencionadas se realizan electrónicamente y en línea, así surgen interacciones entre empresas, individuos y consumidores. Estas relaciones se clasifican como E-commerce de Empresa a Empresa (B2B), Consumidor a Consumidor (C2C), Consumidor a Empresa (C2B) y Empresa a Gobierno (B2G) (Yudiyanto, 2023).

1.4.6 Plataformas personalizables para e-commerce

Un e-commerce se concreta en forma de tiendas en línea o sitios web de comercio electrónico, que usan herramientas digitales como plataformas. Dichas plataformas representan modelos de negocio que simplifican el intercambio entre grupos independientes, como lo son los consumidores, las empresas y los productores (Ortega Fernández, 2017). Entre las empresas y los productores más conocidos está Mercado Libre, OLX, Amazon, Shein y muchas más.

Estas plataformas facilitan la creación y posterior aprobación de solicitudes de compra, el software empleado utiliza tecnología de internet y adquisiciones electrónicas. Además, pueden identificar nuevos proveedores, crear y gestionar sitios web.

1.4.7 Ventajas y Desventajas

Algunas de las ventajas vinculadas al uso de plataformas de e-commerce personalizables van de la mano con sus características, siendo estas:

- Fáciles e intuitivas de utilizar.
- Gestión automatizada de todo el inventario de productos.
- Gran capacidad para la recopilación de datos de sus consumidores.
- Personalización completa de su ambiente.
- Manejo de código abierto y gratuito.
- Servicio de dominio propio.

- Variedad de opciones de personalización.

A pesar de las ventajas mencionadas, es importante mencionar algunas desventajas asociadas al uso de estas plataformas, considerándose las siguientes:

- Dependencia al proveedor de la plataforma.
- Algunas configuraciones son complicadas de manejar.

Se conoce que existe una amplia variedad de plataformas personalizables para iniciar un e-commerce, donde su funcionamiento se basa en Sistemas de Gestión de Contenidos (CMS). Los CMS son sistemas que administran los elementos (imágenes, videos, documentos) que pueden llegar a constituir un sitio web. Esta técnica facilita la creación, publicación y suministración de material digital, además, logra separar la gestión de las plataformas de los diseños preexistentes.

Según Domínguez y otros (Dominguez & Gonzalez Urra, 2006), los CMS almacenan los diseños ya configurados en plantillas, mientras que, los recursos digitales se guardan de manera independiente en bases de datos o ficheros.

En algunos ejemplos de CMS podemos mencionar Wix, WordPress, Drupal, Magento y la más destacada Shopify ya que proporciona herramientas y configuraciones más extensas, diseñadas para desarrolladores.

1.4.8 Shopify

Shopify es un Sistema de Gestión de Contenidos (CMS), diseñado para un e-commerce, donde se brinda la posibilidad de crear y diseñar sitios web. Con una interfaz flexible y un proceso de construcción intuitivo, la creación de la tienda en línea se simplifica. Así, este CMS facilita las tareas de gestión desde un principio (Barreto, Villacreses, Chóez, & Figueroa, 2021).

Este sistema cumple con su objetivo a través de un proceso simple, se comienza por elegir un dominio, luego se escoge un plan de pago, posteriormente se selecciona la plantilla para su personalización y se elige sus funciones, por último, agregan los productos que se desea vender.

CÁPITULO II

MARCO METODOLÓGICO

1.5. REQUERIMIENTOS FUNCIONALES Y NO FUNCIONALES

Los requerimientos funcionales que se determinaron para el desarrollo de la aplicación fueron los siguientes:

- Crear una aplicación en Shopify que se encargue de generar las recomendaciones de productos.
- La aplicación debe registrar el historial generado una vez se haya agregado o instalado la aplicación en la tienda.
- La aplicación debe usar un algoritmo de recomendación para generar las recomendaciones personalizadas de cada cliente que se exista en su historial de compras.
- Las recomendaciones deben ser enviadas mediante correo electrónico a los clientes que tengan aceptado el marketing por correo electrónico.

Considerando también su comportamiento del aplicativo creado, a continuación, se estipularon los requerimientos no funcionales:

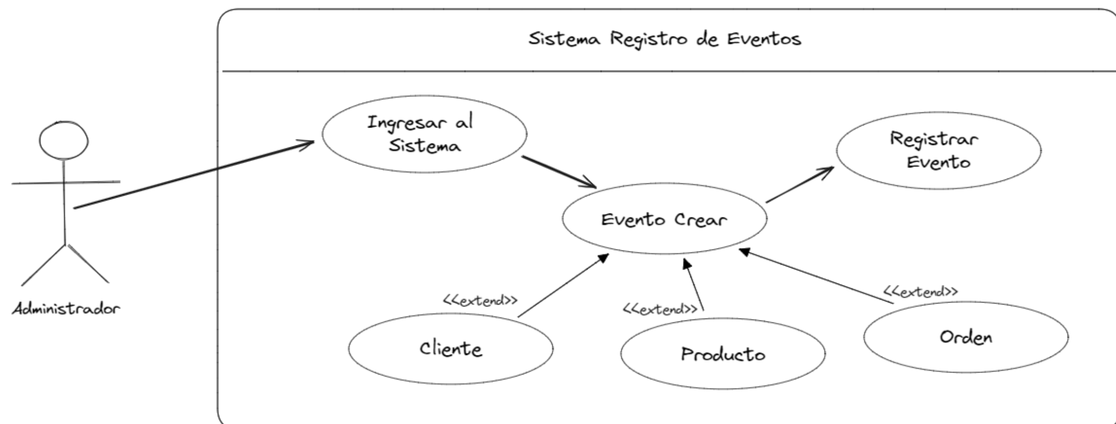
- La aplicación debe manejar la privacidad de los datos de los clientes que se vayan registrando.
- La aplicación debe ser escalable, con el fin de manejar un gran número de clientes.
- La aplicación debe ser fácil de usar por parte de los administradores, con una interfaz intuitiva.
- La aplicación debe manejar un mismo estilo de diseño basado en el panel de administración de Shopify.
- La aplicación debe poseer un panel de métricas que informe de los correos enviados a los clientes.

1.6. CASOS DE USO

En la figura 1 se muestra el registro de eventos de creación generados por el administrador del e-commerce con relación a los clientes, productos y órdenes surgidos en la plataforma de Shopify y en la tabla 1 se muestra la especificación del caso de uso número 1.

Figura 1

Diagrama de Caso de Uso 001



Nota. DCU Registro de Eventos de Creación del Administrador. Elaborado por: Los autores

Tabla 1

Especificación de Caso de Uso 001

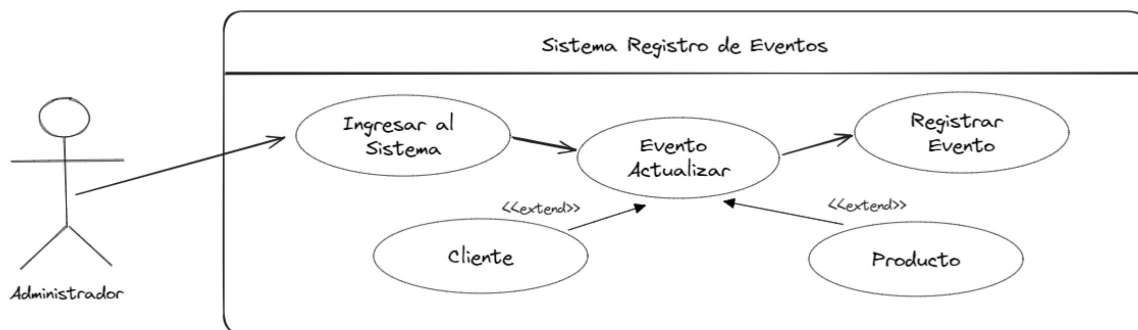
Especificación de Caso de Uso: ECU001	
Descripción	1. El administrador ingresa las credenciales a la plataforma de Shopify 2. La plataforma valida que las credenciales sean correctas 3. El administrador realiza acciones de creaciones en la plataforma 4. El sistema registra la información relacionada a las actividades de creación de manera automática.
Actores	Administrador del e-commerce de Shopify.
Precondición	1. Ingresar al Sistema del e-commerce, mediante las credenciales registradas. 2. Poseer la aplicación integrada instalada en la plataforma de Shopify.
Postcondiciones	Se registra automáticamente la información en el sistema.
Escenario	Usuario definió por el rol administrador el cual gestiona, crea, modifica, consulta o elimina datos del e-commerce

Nota. *Especificación Caso de Uso Registro de Eventos de Creación del Administrador. Elaborado por: Los autores*

En la figura 2 se muestra el registro de eventos de actualización generados por el administrador en la plataforma de Shopify y en la tabla 2 se muestra la especificación del caso de uso número 2.

Figura 2

Diagrama de Caso de Uso 002



Nota. *DCU Registro de Eventos de Actualización del Administrador. Elaborado por: Los autores*

Tabla 2

Especificación de Diagrama de Caso de Uso 002

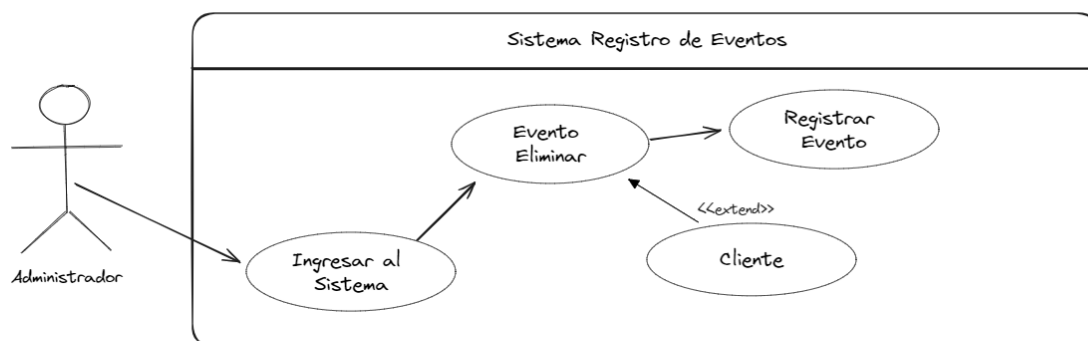
Especificación de Caso de Uso: ECU002	
Descripción	1. El administrador ingresa las credenciales a la plataforma de Shopify 2. La plataforma valida que las credenciales sean correctas 3. El administrador realizas las acciones de actualizaciones en la plataforma. 4. El sistema registra la información relacionada a las activades de actualizaciones de manera automática.
Actores	Administrador del e-commerce de Shopify.
Precondición	1. Ingresar al sistema del e-commerce de Shopify, mediante las credenciales registradas. 2. Poseer la aplicación integrada instalada en la plataforma de Shopify
Postcondiciones	Se registra automáticamente la información en el sistema.
Escenario	Usuario definió por el rol administrador el cual gestiona, crea, modifica, consulta o elimina datos del e-commerce.

Nota. Especificación de Caso de Uso Registro de Eventos de Actualización del Administrador. Elaborado por: Los autores

En la figura 3 se muestra el registro de eventos de eliminación generados por el administrador en la plataforma de Shopify y en la tabla 3 se muestra la especificación del caso de uso número 3.

Figura 3.

Diagrama de Caso de Uso 003



Nota. DCU Registro de Eventos de eliminación del administrador. Elaborado por: Los Autores

Tabla 3

Especificación de Caso de Uso 003

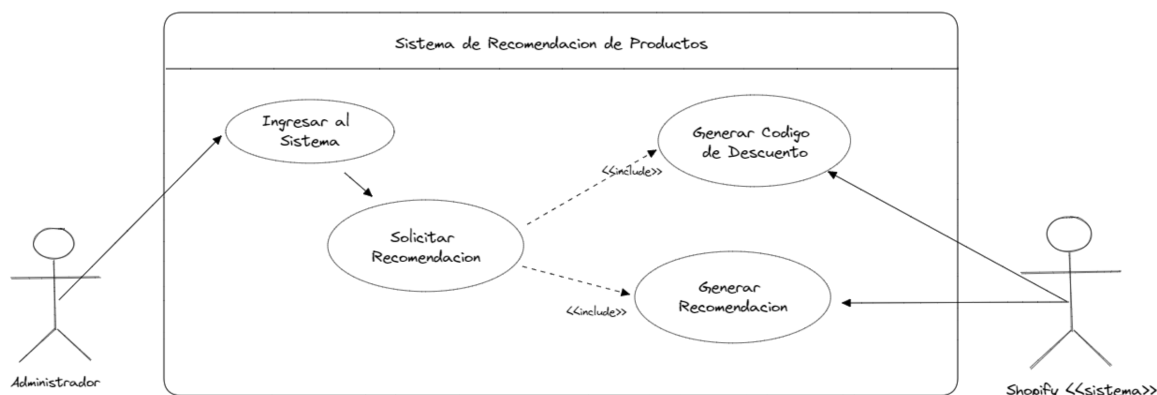
Especificación de Caso de Uso: ECU003	
Descripción	1. El administrador ingresa las credenciales a la plataforma de Shopify 2. La plataforma valida que las credenciales sean correctas 3. El administrador realiza acciones de eliminación en la plataforma 4. El sistema registra la información relacionada a las actividades de eliminación de manera automática.
Actores	Administrador del e-commerce de Shopify.
Precondición	1. Ingresar al Sistema del e-commerce, mediante las credenciales registradas. 2. Poseer la aplicación integrada instalada en la plataforma de Shopify.
Postcondiciones	Se registra automáticamente la información en el sistema.
Escenario	Usuario definió por el rol administrador el cual gestiona, crea, modifica, consulta o elimina datos del e-commerce

Nota. Especificación Caso de Uso Registro de Eventos de Eliminación del Administrador. Elaborado por: Los autores

En la figura 4 se muestra las acciones cuando el administrador solicita una recomendación de productos y en la tabla 4 se muestra la especificación del caso de uso número 4.

Figura 4

Diagrama de Caso de Uso 004



Nota. Especificación Caso de Uso Registro de Eventos de Eliminación del Administrador. Elaborado por: Los autores

Tabla 4

Especificación de Diagrama de Caso de Uso 004

Especificación de Caso de Uso: ECU004	
Descripción	1. El administrador ingresa las credenciales a la plataforma de Shopify 2. La plataforma valida que las credenciales sean correctas 3. El administrador selecciona la aplicación integrada en la plataforma de Shopify. 4. El administrador elige la opción de recomendaciones 5. Se carga los clientes a de los cuales se tiene información registrada en el sistema 6. Se selecciona el o los clientes de los cuales se generan recomendaciones productos 7. Se generan recomendaciones de productos y el código de descuento.
Actores	Administrador del e-commerce de Shopify.
Precondición	1. Ingresar al Sistema del e-commerce, mediante las credenciales registradas. 2. Poseer la aplicación integrada instalada en la plataforma de Shopify.
Postcondiciones	Se registra automáticamente la información en el sistema.
Escenario	Usuario definió por el rol administrador el cual gestiona, crea, modifica, consulta o elimina datos del e-commerce

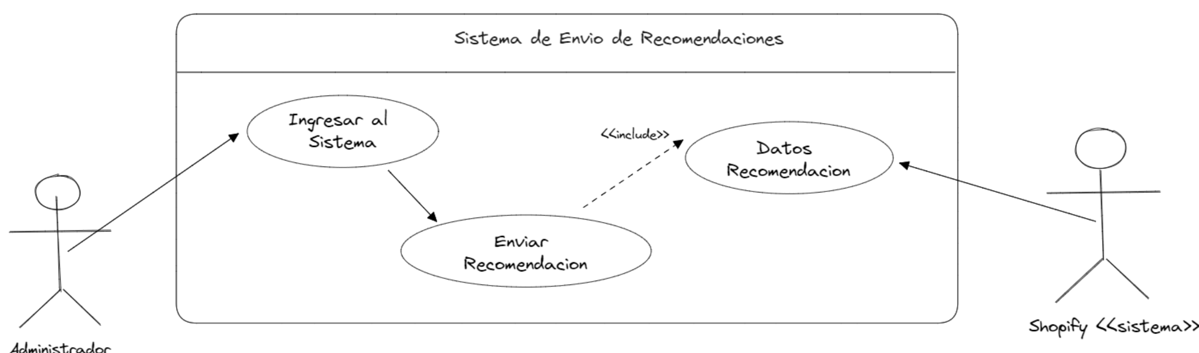
Nota. Especificación Caso de Uso Generación de Recomendación de Productos.

Elaborado por: Los autores

En la figura 5 se muestra las acciones cuando el administrador solicita enviar las recomendaciones a los clientes y en la tabla 5 se muestra la especificación del caso de uso número 5.

Figura 5

Diagrama de Caso de Uso 006



Nota. DCU Envió de recomendaciones generados. Elaborado por: Los autores

Tabla 5

Especificación de Caso de Uso 005

Especificación de Caso de Uso: ECU005	
Descripción	1. El administrador ingresa las credenciales a la plataforma de Shopify 2. La plataforma valida que las credenciales sean correctas 3. El administrador solicita recomendaciones de productos 4. El administrador llenar la información del formulario 5. Se envía formulario mediante petición POST 6. Se envía las recomendaciones mediante correo electrónico a los clientes de destino.
Actores	Administrador del e-commerce de Shopify.
Precondición	1. Ingresar al Sistema del e-commerce, mediante las credenciales registradas. 2. Poseer la aplicación integrada instalada en la plataforma de Shopify.
Postcondiciones	Se registra automáticamente la información en el sistema.
Escenario	Usuario definió por el rol administrador el cual gestiona, crea, modifica, consulta o elimina datos del e-commerce

Nota. Especificación Caso de Uso Envió de Recomendaciones. Elaborado por: Los autores

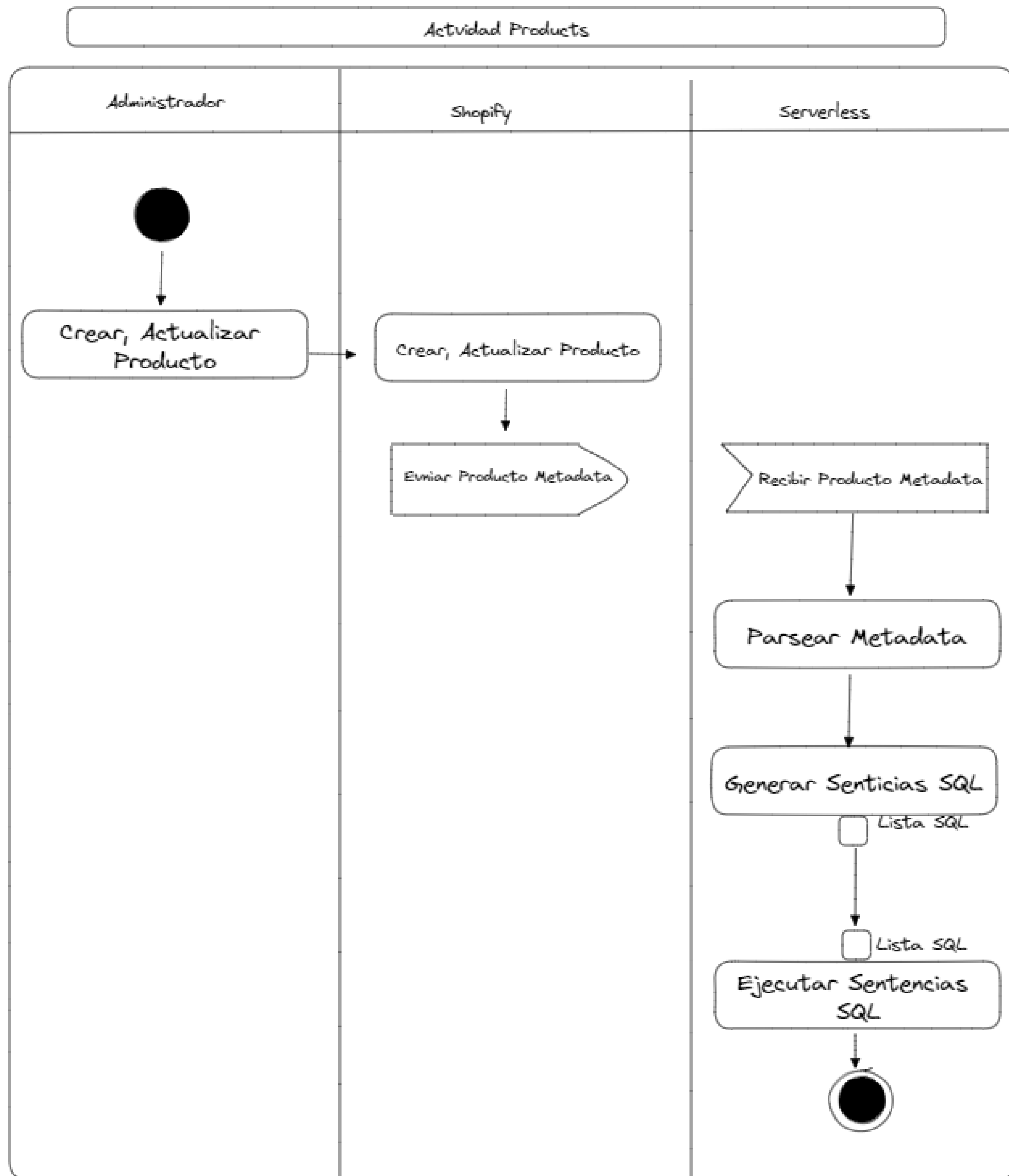
1.7. DIAGRAMA DE ACTIVIDADES

Los diagramas de actividades nos permiten mostrar el flujo de las actividades o procesos ejecutadas por un sistema para realizar una acción o actividad en concreto. Por medio de símbolos como inicio de actividad, actividad, conector, envío de señal, recepción de señal, finalización de actividad, etc. (Lucidchart, n.d.)

En la figura 6 se muestra el proceso para el registro de los eventos relacionados con los productos del e-commerce de Shopify y en la tabla 6 se detalla el flujo del proceso.

Figura 6

Diagrama de Actividades 001



Nota. DA: Actividad de Registro de Eventos relacionados a Productos. Elaborado por: Los autores

Tabla 6

Flujo de Procesos 001

Flujo de Procesos: FP001	
Descripción	Actividad para registro de eventos vinculados a los productos del e-commerce.
Autores	Administrador (Usuario), App Remix (Shopify), Plataforma Serverless (AWS)
Flujo	1. Administrador genera evento de creación o actualización de productos 2. Plataforma Shopify captura información relacionada al evento de creación o actualización del producto. 3. Plataforma Shopify envía mensaje con la información a la plataforma serverless 4. Plataforma Serverless recibe el mensaje enviado por la plataforma de Shopify 5. Plataforma Serverless obtiene la información del mensaje 6. Plataforma Serverless genera la lista de sentencias SQL con la información obtenida 7. Plataforma Serverless ejecuta la lista de sentencias SQL

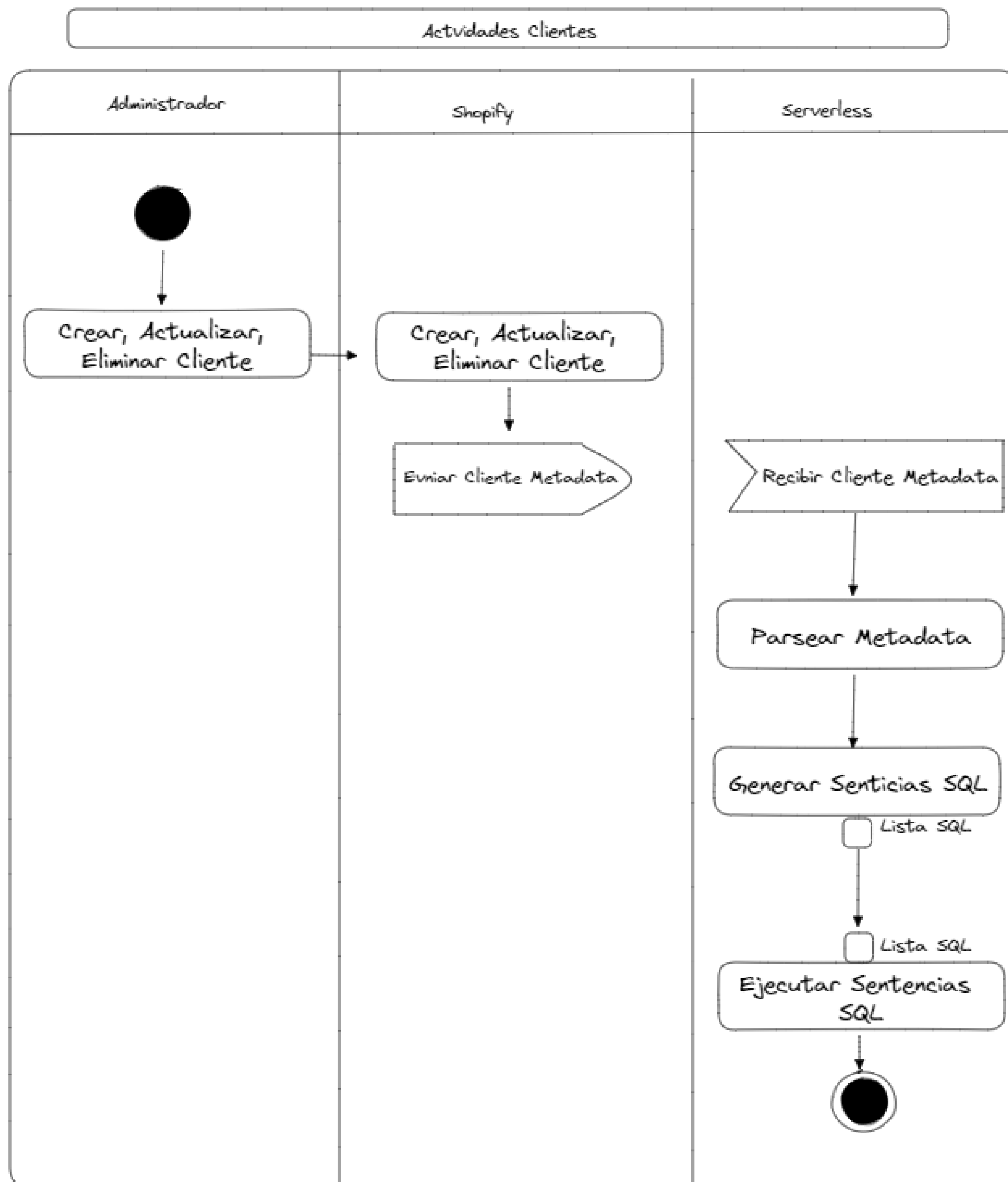
Nota. Tabla de Flujo de Proceso de Registro de Eventos Relacionados a Productos.

Elaborado por: Los autores

En la figura 7 se muestra el proceso para el registro de los eventos relacionados con los clientes del e-commerce de Shopify y en la tabla 7 se detalla el flujo del proceso.

Figura 7

Diagrama de Actividades 002



Nota. DA: Actividad de Registro de Eventos relacionados a Clientes. Elaborado por: Los autores

Tabla 7

Flujo de Procesos 002

Flujo de Procesos: FP002	
Descripción	Actividad para registro de eventos vinculados a los clientes del e-commerce.
Autores	Administrador (Usuario), App Remix (Shopify), Plataforma Serverless (AWS)
Flujo	1. Administrador genera evento de creación, actualización o eliminación de clientes 2. Plataforma Shopify captura información relacionada al evento de creación o actualización del producto. 3. Plataforma Shopify envía mensaje con la información a la plataforma serverless 4. Plataforma Serverless recibe el mensaje enviado por la plataforma de Shopify 5. Plataforma Serverless obtiene la información del mensaje 6. Plataforma Serverless genera la lista de sentencias SQL con la información obtenida 7. Plataforma Serverless ejecuta la lista de sentencias SQL

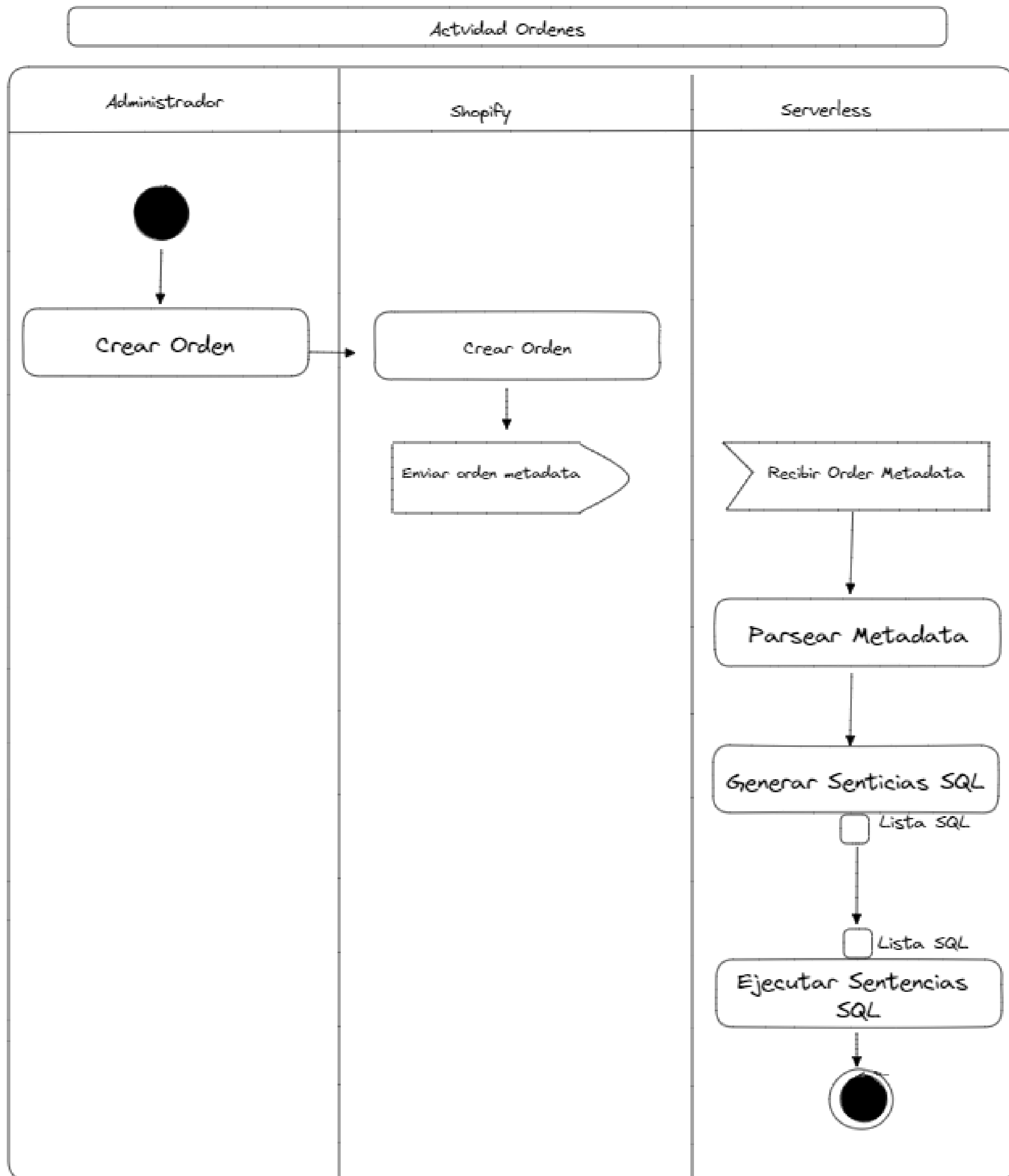
Nota. *Tabla de Flujo de Proceso de Registro de Eventos Relacionados a Clientes.*

Elaborado por: Los autores

En la figura 8 se muestra el proceso para el registro de los eventos relacionados con las órdenes del e-commerce de Shopify y en la tabla 8 se detalla el flujo del proceso.

Figura 8

Diagrama de Actividades 003



Nota. DA: Actividad de Registro de Eventos relacionados a Ordenes. Elaborado por: Los autores

Tabla 8.

Flujo de Procesos 003

Flujo de Procesos: FP003	
Descripción	Actividad para registro de eventos vinculados a las órdenes del e-commerce.
Autores	Administrador (Usuario), App Remix (Shopify), Plataforma Serverless (AWS)
Flujo	1. Administrador genera evento de creación de orden 2. Plataforma Shopify captura información relacionada al evento de creación o actualización del producto. 3. Plataforma Shopify envía mensaje con la información a la plataforma serverless 4. Plataforma Serverless recibe el mensaje enviado por la plataforma de Shopify 5. Plataforma Serverless obtiene la información del mensaje 6. Plataforma Serverless genera la lista de sentencias SQL con la información obtenida 7. Plataforma Serverless ejecuta la lista de sentencias SQL

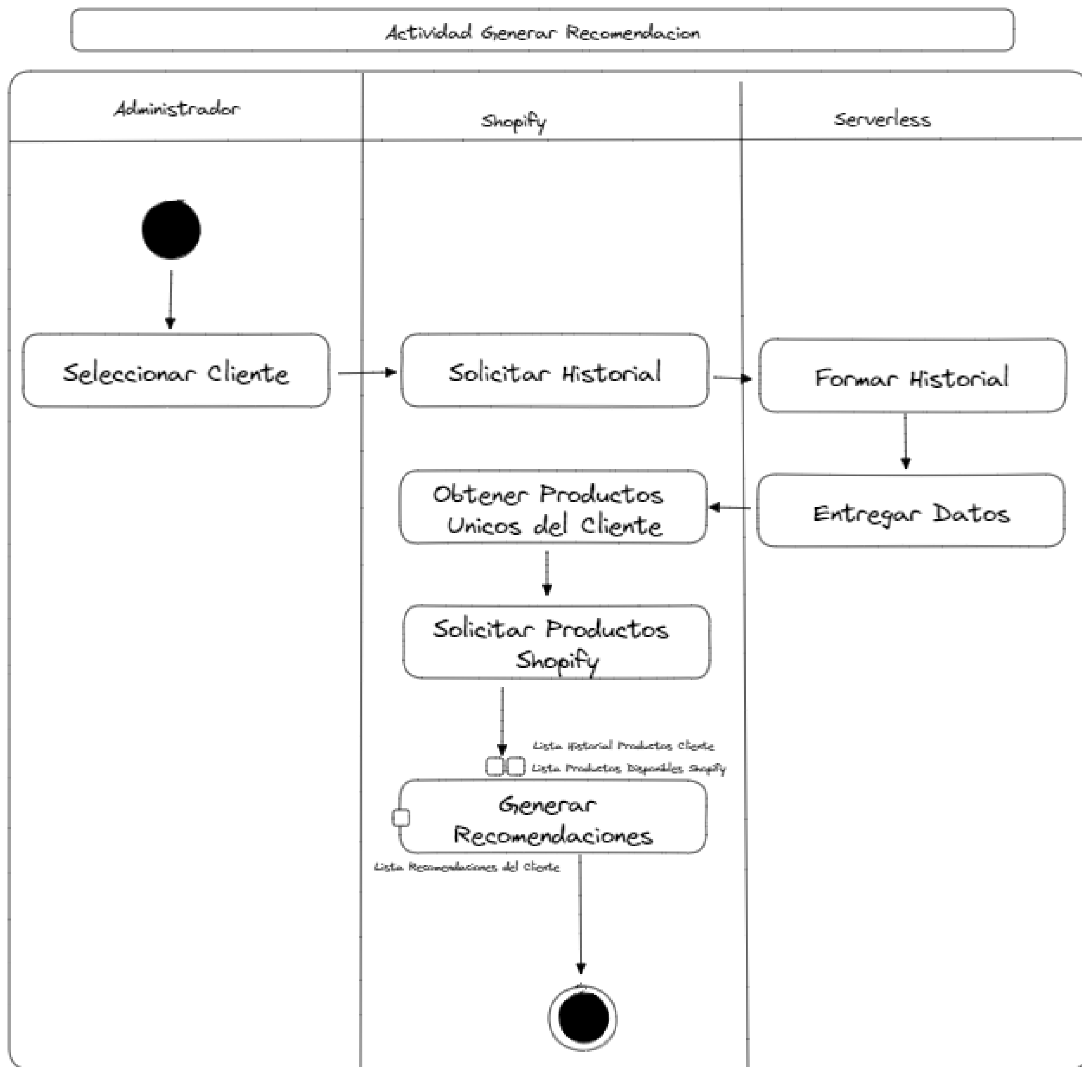
Nota. Tabla de Flujo de Proceso de Registro de Eventos Relacionados a Ordenes.

Elaborado por: Los autores

En la figura 9 se muestra el proceso para la generación de recomendaciones de productos y en la tabla 9 se detalla el flujo del proceso.

Figura 9

Diagrama de Actividades 004



Nota. DA: Actividad de Generación de Recomendación de Producto. Elaborado por: Los autores

Tabla 9

Flujo de Procesos 004

Flujo de Procesos: FP004	
Descripción	Actividad para la generación de recomendaciones
Autores	Administrador (Usuario), App Remix (Shopify), Plataforma Serverless (AWS)
Flujo	1. Administrador selecciona un cliente 2. Plataforma Shopify solicita el historial del cliente a la plataforma serverless. 3. Plataforma Serverless forma el historial 4. Plataforma Serverless envía el historial a la plataforma de Shopify 5. Plataforma Shopify obtiene la información de los productos relacionado al cliente 6. Plataforma Shopify solicita productos disponibles y comercializados en el e-commerce 7. Plataforma Shopify genera las recomendaciones de productos

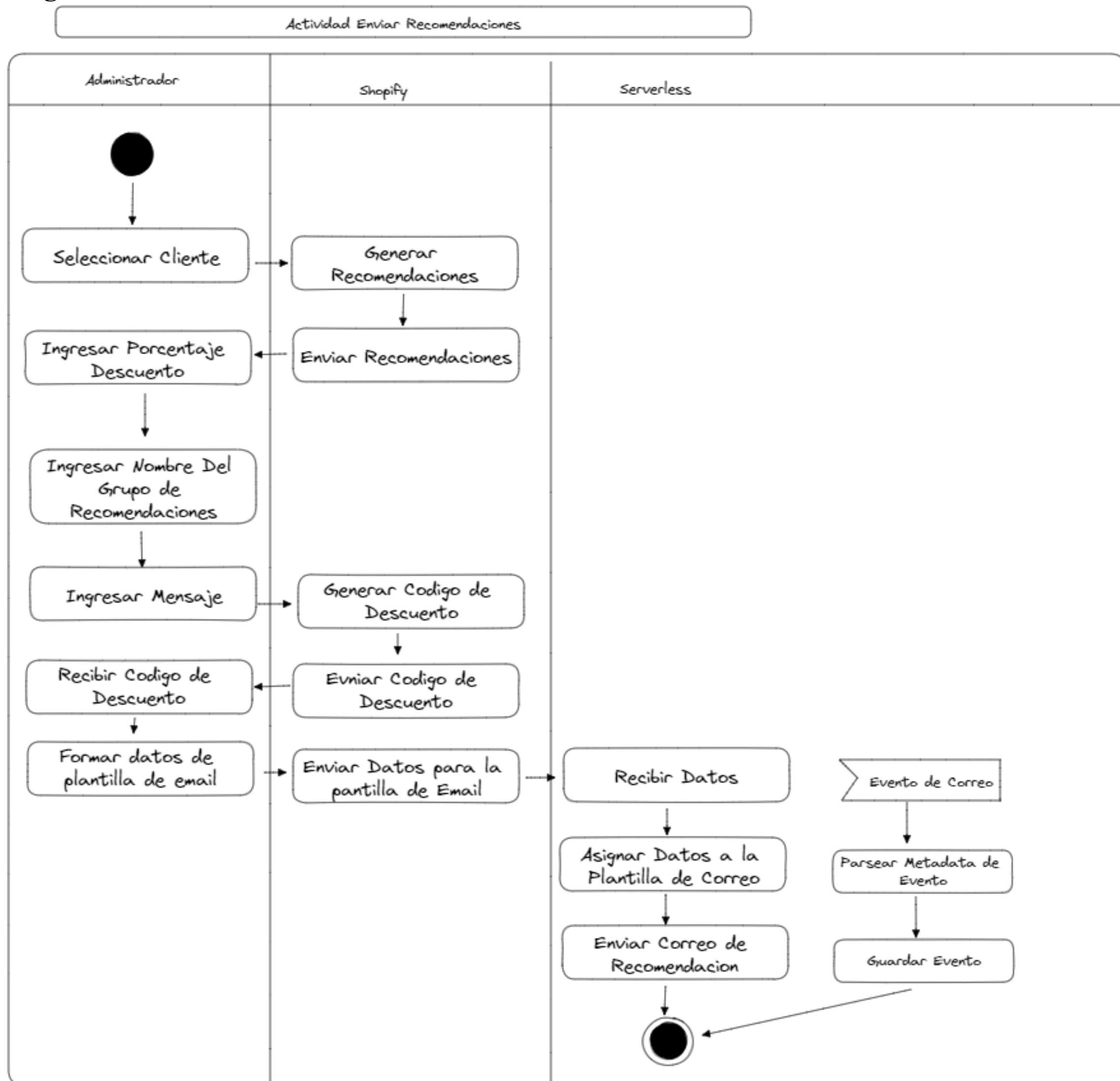
Nota. Tabla de Flujo de Proceso de Generación de Recomendación de Producto.

Elaborado por: Los autores

En la figura 10 se muestra el proceso para él envió de recomendaciones de productos a los clientes y en la tabla 10 se detalla el flujo del proceso.

Figura 10

Diagrama de Actividades 005



Nota. DA: Actividad de Envió de Recomendación de Producto. Elaborado por: Los autores

Tabla 10

Flujo de Proceso 005

Flujo de Proceso: FP005	
Descripción	Actividad para él envió de recomendaciones de productos mediante correo electrónico.
Autores	Administrador (Usuario), App Remix (Shopify), Plataforma Serverless (AWS)
Flujo	1. Administrador selecciona un cliente 2. Plataforma Shopify genera recomendaciones de productos 3. Plataforma Shopify envía recomendaciones al administrador 4. Administrador ingresa descuento a aplicar para las recomendaciones 5. Administrador ingresa la información en el formulario de envió de correo 6. Plataforma Shopify genera y registra el código de descuento 7. Plataforma Shopify envía el código de descuento 8. Administrador recibe el código de descuento registrado 9. Administrador envía información del correo a la plataforma serverless 10. Plataforma serverless recibe información para él envió de recomendación 11. Plataforma serverless asigna información recibida a la plantilla de correo 12. Plataforma serverless envía correo al cliente de destino
Flujo Complementario	1. Plataforma serverless captura eventos generados por el correo del cliente de destino. 2. Plataforma serverless registra los eventos en la base de datos.

Nota. Tabla de Flujo de Proceso de envió de Recomendación de Producto. Elaborado por: Los autores

1.8. HERRAMIENTAS DE DESARROLLO

Para el desarrollo del proyecto se implementaron las siguientes herramientas de desarrollo que se detallan en las diferentes tablas presentadas a continuación.

En la tabla 11, se definió los servicios de AWS esenciales para el desarrollo de una plataforma serverless.

Tabla 11

Servicios de Amazon Web Services

Tecnología	Descripción
Lambda	Servicio que permite ejecutar código sin administración de servidores.
RDS/MySQL	Colección de servicios administrados que permite configuración, operación y escalado de base de datos en la nube.
EventBridge	Servicio para creación de aplicación basada en eventos.
Simple Queue Service	Servicio de cola de mensajes que envía, almacena, recibe y comunica mensajes entre componentes de software
API Gateway	Servicio administrado para el desarrollo, creación, publicación, mantenimiento, monitoreo y protección de API's.
Simple Email Service	Servicio de correo electrónico que permite la automatización de envió de grandes volúmenes de correos electrónicos.
Cloudwatch	Servicio de monitoreo de aplicación, el cual response a cambios de rendimiento y optimiza los recursos.
Simple Notification Service	Servicio de envió de notificación, el cual brinda mensajería de muchos a muchos de alto rendimiento basado en push para sistemas distribuidos, microservicios y serverless.

Nota. Servicios de Amazon Web Services implementados. Elaborado por: Los autores

En la tabla 12, se definió las librerías esenciales para el desarrollo de la aplicación web.

Tabla 12

Librerías implementadas

Tecnología	Enfoque	Versión	Descripción
ReactJS	Librería de Desarrollo	18.2.0	Librería para desarrollo de interfaces basado en componentes de piezas individuales.
Typescript	Superset de Tipado en JS	5.2.2	Lenguaje de programación fuertemente tipado basado en JavaScript.
Shopify API	Librería API de Shopify Admin	8.0.2	Librería para interacción con la plataforma de Shopify, tanto al servicio REST como GraphQL
D3js	Librería de desarrollo de gráficos interactivos	7.8.5	Librearía open-source para visualización de datos, con enfoque de bajo nivel para flexibilidad, y gráficos dinámicos.
Nanoid	Librearía para generar UUID	3.3.7	Librería generadores de Identificadores Únicos.
Mysql2	Librería de Conexión a MySQL	3.65	Librería nodejs para comunicación entre cliente – servidor para motor de base de datos MySQL

Nota. Librerias implementados. Elaborado por: Los autores

En la tabla 13, se definió los frameworks esenciales para el desarrollo de la aplicación web y mejorar la experiencia de desarrollo.

Tabla 13
Frameworks utilizados

Tecnología	Enfoque	Versión	Descripción
Remix	Framework de Desarrollo	2.0.0	Framework web full stack para el desarrollo de aplicaciones web basado en aplicaciones MPA y en estándares web
Shopify Polarios	Template de Componentes	12.0.0	Sistema de diseño para el administrador de Shopify, con lenguaje compartido para crear experiencias de alto nivel.
Shopify App Remix	Framework para Desarrollo de Aplicaciones Integradas en Shopify Admin	2.1.0	Paquete para habilitar autenticación y realizar llamadas a la API en aplicaciones Remix.

Nota.Frameworks de desarrollo implementados. Elaborado por: Los autores

En la tabla 14, se definieron los servicios y herramientas de terceros complementarios al desarrollo del proyecto.

Tabla 14
Servicios y Herramientas de terceros utilizados

Tecnología	Enfoque	Descripción
Github	Repositorio de Código	Plataforma para creación, almacenamiento y administración del código de los desarrolladores.
Jira	Organizador de Actividades	Plataforma administración de proyectos basado en metodologías agiles que permite el seguimiento de problemas, bugs, historias de usuario, etc.
GoDaddy	Proveedor de Dominio	Plataforma para el registro de dominios, hosting web, etc. [x]
Cloudflare	Proveedor de Servicio en la Nube	Plataforma proveedora de servicios como CDN, servicios web, seguridad en la nube, registro de dominios, etc
Eraser	Herramienta de Diagramas	Aplicación web para el desarrollo de diagramas UML colaborativo entre usuarios
Exalidraw	Herramienta de Traficación	Aplicación web colaborativa para el diseño de diagramas basado en un pizarrón en blanco.
Postman	Cliente para peticiones http/s.	Aplicación de cliente http para realizar peticiones a servicios API

Nota. Servicio y Herramientas de terceros implementados. Elaborado por: Los autores

1.9. ARQUITECTURA GENERAL

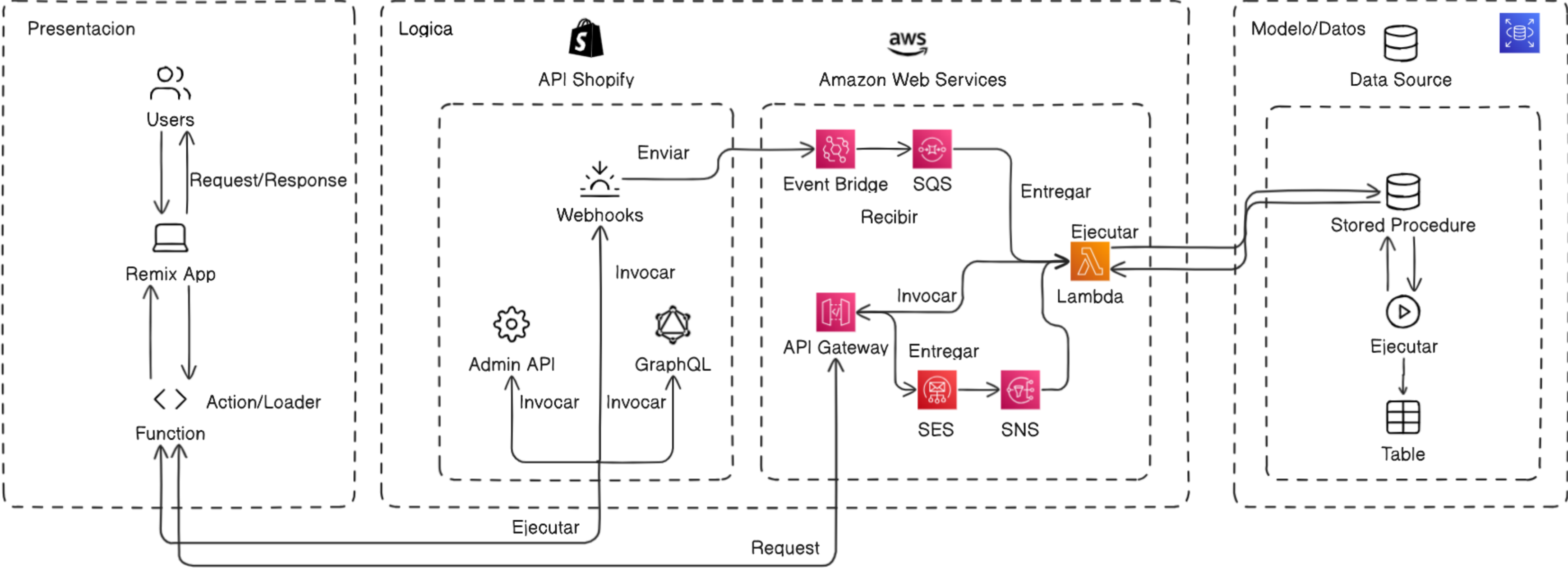
La arquitectura está basada en servicios serverless y suscripción y/o de eventos. Por lo que se definieron los servicios y formas de comunicación del sistema, que se muestra en la figura 11. En donde la forma de interacción del usuario es por medio de hooks de useLoaderData y useActionData pertenecientes a Remix, los cuales invocan los servicios de Admin API, GraphQL y Webhooks de Shopify en la aplicación.

El servicio de Webhooks o Event es la forma mediante se captura un evento generado por parte del usuario en la plataforma de Shopify, en donde se envía un mensaje con la información del evento generado al servicio de AWS Event Bridge, el cual a su vez reenvía dicho mensaje a una cola de mensajes del servicio AWS SQS, con el fin de ejecutar con una función lambda que interactúe con procedimientos almacenados definido en la base de datos del servicio AWS RDS.

El otro servicio con el que interactúa el usuario, es el servicio de API Gateway, en el cual se realiza peticiones a los recursos publicados en la API REST definida. En el primer caso invoca una función lambda la cual recupera información de la base de datos por medio de un procedimiento almacenado y en el segundo caso invoca al servicio AWS SES el cual envía el correo electrónico e invoca al servicio SNS para notificar los eventos surgidos de dicho evento, con el fin de invocar una función lambda que registre dicha información mediante un procedimiento almacenado

Figura 11

Arquitectura General del Proyecto Técnico



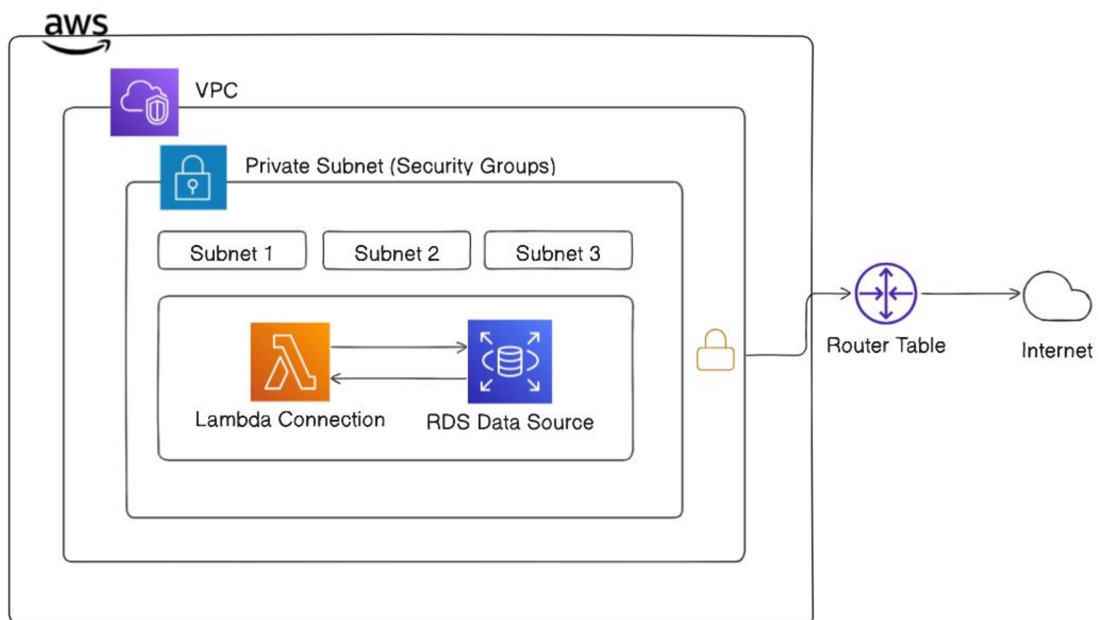
Nota. Arquitectura de Software General del Proyecto Tecnico. Elaborado por: Los autores

1.10. IMPLEMENTACIÓN DE BASE DE DATOS EN AWS RDS

Para la implementación del servicio RDS en AWS se definió la arquitectura presentada en la figura 12, la cual es generada de manera automática mediante la configuración establecida en la tabla 15.

Figura 12

Diagrama de Arquitectura Servicio RDS



Nota. Arquitectura de Software del Servicio RDS. Elaborado por: Los autores

Tabla 15

Configuración de Servicio RDS

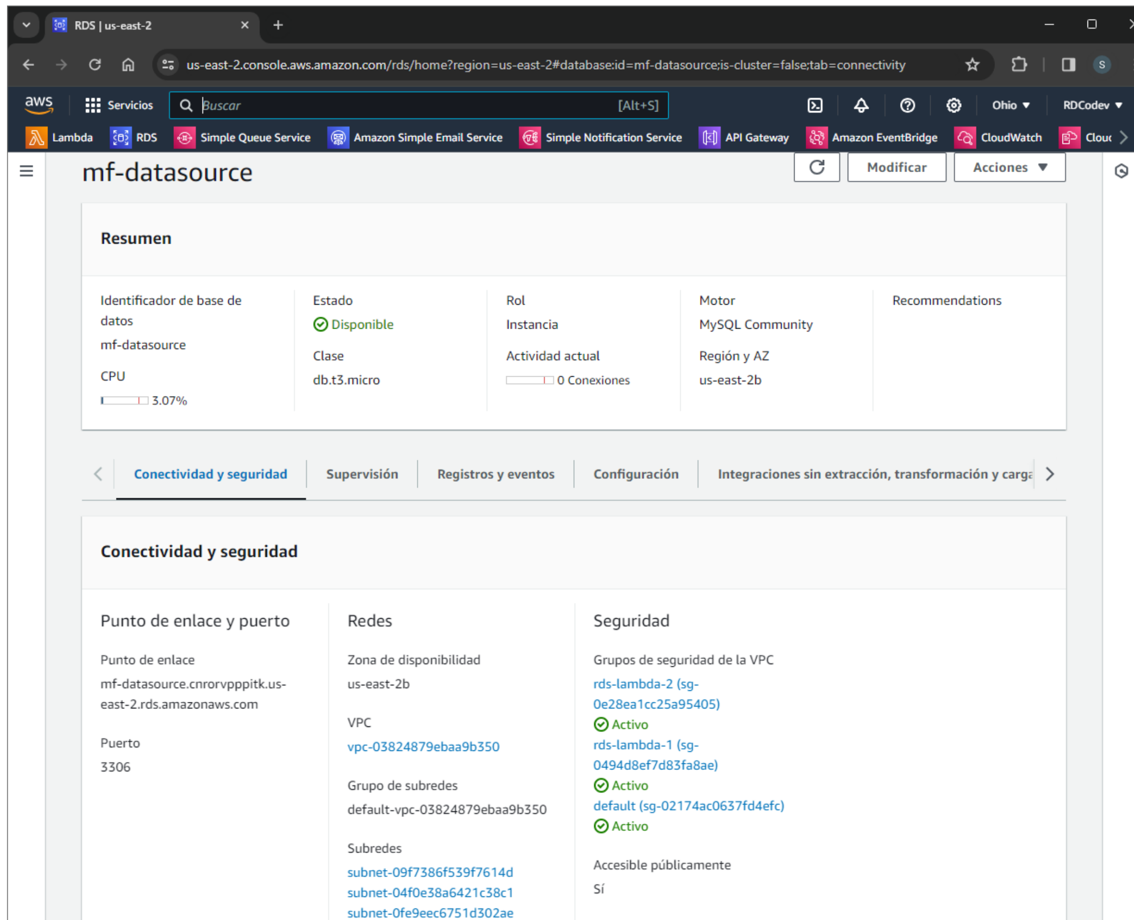
Opción	Configuración	Descripción
Encryption	Enable	Permite el cifrado de la base de datos con claves que se administra desde AWS Key Management Service (KMS).
VPC	Default-vpc-03824879ebaa9b350	Permite aislar las instancias de la base de datos en su propia red virtual.
Option Group	Default: mysql-8-0	Configuración que permite acceder a instancias específicas de la base de datos para ser modificadas.
Subnet Group	Subnet-09f7386f539f7614d	Grupo de la subred publica a la que pertenece
Automatic bakcups	Enabled	Permite la creación de respaldos
VPC security group	Default-sg-02174ac0637fd4efc	Actúa como firewall virtual que controla el tráfico de una o más instancias.
Publicly accesible	Yes	Configuración para mantener un acceso publico
Database port	3306	Puerto en la red de la base de datos
DB instance identifier	Mf-datasource	Este identificador se genera como instancia de la base de datos, seguido por la creación de una cuenta de usuario principal que se utiliza solo contexto de Amazon RDS.
DB engine version	8.0.33	Versión de la base de datos que se maneja.
DB parameter group	Default:mysql8.0	Grupo de AWS AIM que realiza recursos específicos.
Performance insights	Disabled	Información del rendimiento
Monitoring	Enabled	Información de monitoreo
Maintenance	Auto minor versión	Mantenimiento del servicio
Delete protection	Disabled	Protección contra eliminación de la base de datos

Nota. Tabla de configuracion implementada en el servicio RDS. Elaborado por: Los autores

Configurado el servicio RDS tenemos un punto de almacenamiento para consultar y guardar información como se muestra en la figura 13.

Figura 13

Servicio RDS Inicializado



Nota. *Servicios de AWS RDS. Elaborado por: Los autores*

Además, como se mostró en la figura 12, se requiere una función lambda que permite la administración de la instancia de la base de datos generada por el servicio RDS, por lo que la configuración de la función lambda implementada en la misma VPC asociada a la base de datos se presenta en la tabla 16.

Tabla 16

Configuración Función Lambda RDS Connection

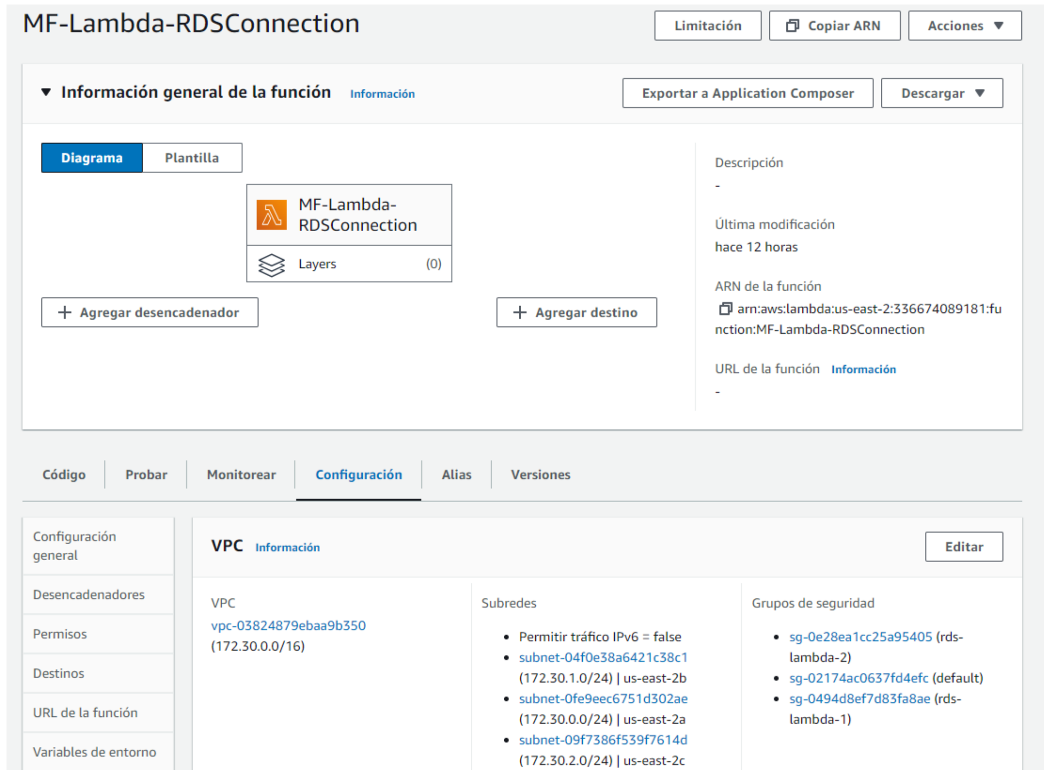
Función Lambda RDSConnection	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Ec2: CreateNetworkInterace, DeleteNetworkInterface, DescribeNetworkInterface.
	Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocación	Asíncrona
RDS Associated	MF-DataSoruce
Versión	v0.1.0
Subnet	04f0e38a6421c38c1, 0fe9eec6751d302ae, 09f7386f539f7614d
VPC	03824879ebaa9b350
Security Groups	0e28ea1cc25a95405, 02174ac0637fd4efc, 049d8ef7d83fa8ae
RDS databases	Mf-datasource

Nota. Configuración implementada en función lambda. Elaborado por: Los autores

Implementada la función lambda tenemos una manera de administrar la conexión de la base de datos y realizar ejecuciones de sentencias SQL mediante un servicio serverless, como es muestra en la figura 14.

Figura 14

Función Lambda vinculada a la VPC del servicio RDS



Nota. Función Lambda RDS Connection. Elaborado por: Los autores

El código empleado para la administración de la base de datos y ejecución de sentencias SQL es presentado en la figura 15 y los detalles más importantes del código se presenta en la tabla 17.

Figura 15

Código Función Lambda RDS Connection

```
import { connection } from './config.mjs';

export const handler = async (event) => {

  const { queries } = event;
  const conn = await connection.getConnection();

  try {

    const [rows, _] = await Promise.all(
      queries.map(sql => connection.query(sql))
    );

    if (!rows.length) throw new Error("Result's not found");

    return { "result": [...rows.flat(Infinity)] };

  } catch (error) {
    throw error;
  } finally {
    connection.releaseConnection(conn);
  }
};
```

Nota. Código implementado en la función lambda. Elaborado por: Los autores

Tabla 17

Tabla de Código Función Lambda 001

FL001: Código Relevante Función Lambda RDS Connection	
Función	Descripción
async handler(event)	Se encarga de obtener la conexión con la base de datos y ejecutar las sentencias recibidas por parámetros de manera asíncrona.
Promise.all()	Método encargado de realizar ejecución de promesas de manera paralela
connection.query()	Método encargado de realizar las ejecuciones de las sentencias SQL de manera asíncrona.

Nota. Tabla de detalles relevantes del código de la función lambda RDS Connection.

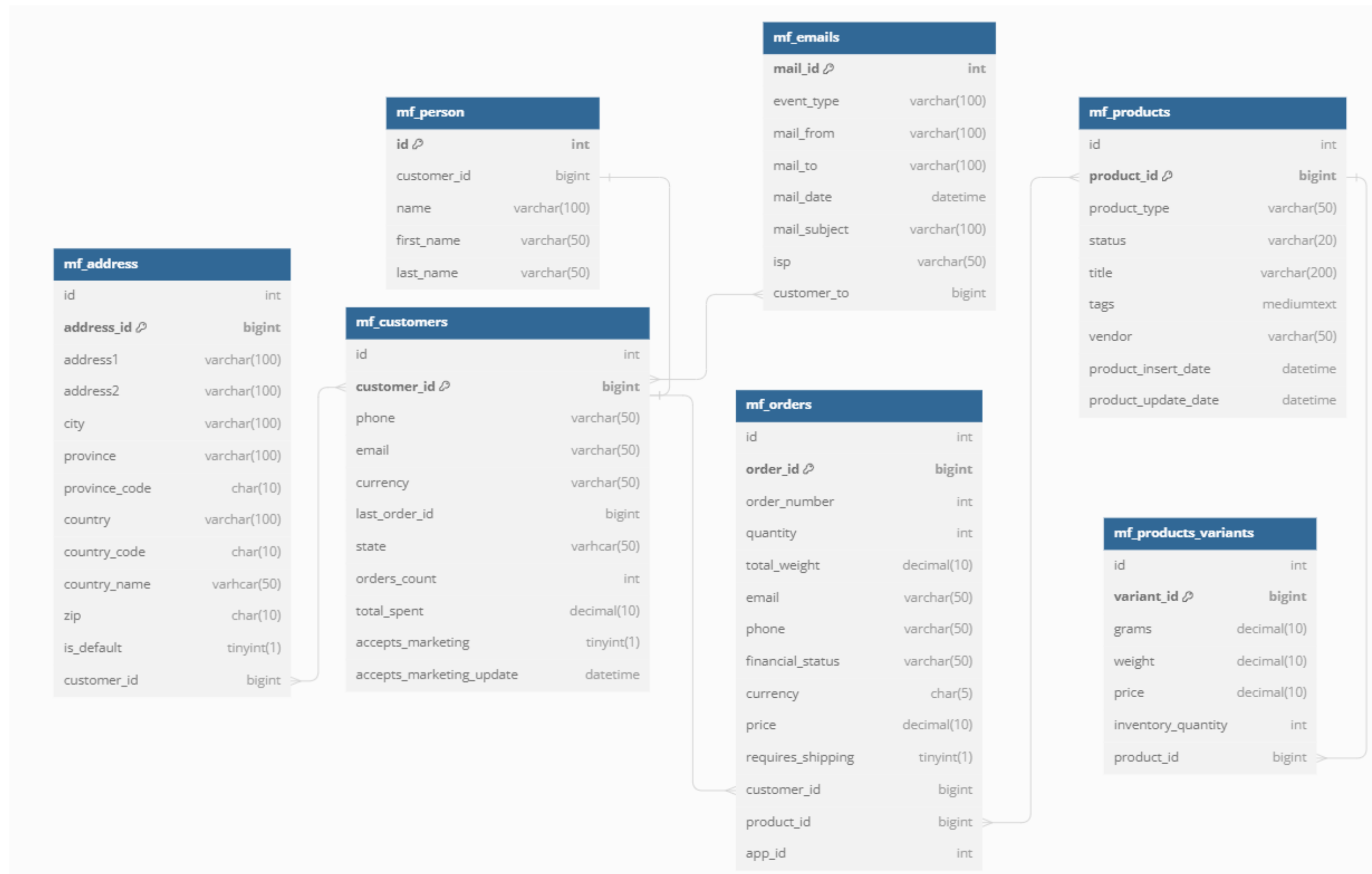
Elaborado por: Los autores

1.10.1 Modelado

El modelo de la base de datos implementado en el proyecto para el almacenamiento de la información se presenta en la figura 16 y los diccionarios de datos correspondientes al modelo en las tablas 18, tabla 19, tabla 20, tabla 21, tabla 22, tabla 23 y tabla 24. Además, los procedimientos almacenados definidos se presentan en la tabla 25.

Figura 16

Diagrama Entidad Relación de la Base de Datos



Nota. *Servicios de Amazon Web Services implementados. Elaborado por: Los autores*

Tabla 18

Diccionario de Datos 001

Tabla mf_person						
Descripción		Almacena la información básica de un cliente del e-commerce de Shopify				
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-	Si	-	Si	Campo autogenerado y autoincremental
customer_id	Bigint unsigned	-	-	Si	Si	Campo perteneciente un cliente de la plataforma de Shopify y asociación
name	varchar	100	-	-	Si	Nombre obtenido de la unión del primer nombre y apellido
first_name	varchar	50	-	-	Si	Primer nombre del cliente
last_name	varchar	50	-	-	Si	Apellido del cliente

Nota. Diccionario de Datos de la tabla mf_person. Elaborado por: Los autores

Tabla 19

Diccionario de Datos 002

Tabla mf_customers						
Descripción	Almacena la información relacionada a un cliente del e-commerce de Shopify					
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-			Si	Campo autogenerado y autoincremental
customer_id	Bigint unsigned	-	Si		Si	Campo perteneciente un cliente de la plataforma de Shopify
phone	Varchar	50			No	Teléfono de contacto de un cliente registrado
email	Varchar	50			No	Email de contacto de un cliente registrado
currency	Varchar	50			Si	Tipo de moneda de un cliente
last_order_id	Bigint unsigned	-			No	Numero de la última orden registrada y asociación
state	Varchar	50			Si	Estado del cliente, activo o desactivo
orders_count	Int	-			No	Total de ordenes registrada por el cliente
total_spent	Decimal	10			No	Total de consumo registrado por el cliente
accepts_marketing	Tinyint	1			Si	Estado de marketing por email
accepts_marketing_update	datetime	-			Si	Fecha de actualización del estado del marketing por email

Nota. Diccionario de Datos de la tabla mf_customers. Elaborado por: Los autores

Tabla 20

Diccionario de Datos 003

Tabla mf_address						
Descripción				Almacena las direcciones vinculadas a los clientes del e-commerce		
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-			Si	Campo autogenerado y autoincrementado
address_id	Bigint	-	Si		Si	Campo perteneciente a la plataforma de Shopify
address1	Varchar	100			Si	Nombre de la calle principal de un cliente
address2	Varchar	100			No	Nombre de la calle secundaria de un cliente
city	Varchar	100			Si	Nombre de la cuidad de residencia de un cliente
province	Varchar	100			Si	Nombre de la provincia o estado de una residencia de un cliente
province_code	Char	10			Si	Código (3 caracteres) de la provincia o estado del cliente
country	Varchar	100			Si	Nombre del país del cliente
country_code	Char	10			Si	Código del país del cliente
country_name	Varchar	50			No	Nombre común del país del cliente
zip	Char	10			No	Código zip de la ubicación del cliente
is_default	Tinyint	1			No	Estado de dirección principal del cliente
customer_id	Bigint unsigned	-		Si	Si	Campo perteneciente a la plataforma de Shopify y asociación

Nota. Diccionario de Datos de la tabla mf_address. Elaborado por: Los autores

Tabla 21

Diccionario de Datos 004

Tabla mf_products						
Descripción		Almacena los productos disponibles en el e-commerce de Shopify				
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-	No		Si	Campo autogenerado y autoincremental
product_id	Bigint	-	Si		Si	Campo perteneciente a la plataforma de Shopify y asociación
product_type	Varchar	50			Si	Tipo de producto
status	Varchar	20			Si	Estado del producto en el panel de administración
title	Varchar	200			Si	Titulo o Nombre del producto
tags	Mediumtext	-			Si	Etiquetas asociadas al producto
vendor	Varchar	50			Si	Distribuidor asociado al producto publicado en la tienda
product_insert_date	Datetime	-			Si	Fecha en la que el producto fue agregado
product_update_date	Datetime	-			Si	Fecha de actualización de los datos del producto

Nota. Diccionario de Datos de la tabla mf_products. Elaborado por: Los autores

Tabla 22

Diccionario de Datos 005

Tabla mf_products_variants						
Descripción				Almacena las variaciones de un mismo producto		
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-	No		Si	Campo autogenerado y autoincremental
variant_id	Bigint unsigned	-	Si		Si	Campo perteneciente a la plataforma de Shopify
grams	Decimal	10			No	Gramos totales de la variante del producto
weight	Decimal	10			No	Peso total de la variante del producto
price	Decimal	10			No	Precio de distribución del producto
inventory_quantity	Int	-			No	Cantidad disponible para venta de la variante del producto
product_id	Bigint unsigned	-		Si	Si	Campo perteneciente a la plataforma de Shopify y de asociación

Nota. Diccionario de Datos de la tabla mf_products_variants. Elaborado por: Los autores

Tabla 23

Diccionario de Datos 006

Tabla mf_orders						
Descripción			Almacena la orden creada en el e-commerce de Shopify			
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
id	Int	-			Si	Campo autogenerado y autoincremental
order_id	Bigint unsigned	-	Si		Si	Campo perteneciente a la plataforma de Shopify
order_number	Int	-			Si	Numero de la orden generada a partir de 1000
quantity	Int	-			No	Cantidad total de productos agregados
total_weight	Decimal	10			No	Cantidad total de los productos agregados
email	Varchar	50			Si	Correo de contacto del cliente
phone	Varchar	50			No	Teléfono de contacto del cliente
financial_status	Varchar	50			Si	Estado del pago de la orden
currency	Char	5			Si	Moneda de pago de la orden
price	Decimal	10			Si	Precio total productos agregados, incluido impuestos
requires_shipping	Tinyint	1			No	Estado para determinar si la orden requiere envío
customer_id	Bigint	-		Si		Campo perteneciente a la plataforma de Shopify y asociación
product_id	Bigint	-		Si		Campo perteneciente a la plataforma de Shopify y asociación
app_id	int	-			No	Campo para conocer que aplicación genero la orden

Nota. Diccionario de Datos de la tabla mf_orders. Elaborado por: Los autores

Tabla 24

Diccionario de Datos 007

Tabla mf_emails						
Descripción				Almacena la información surgida de los eventos de envió de correos		
Nombre	Tipo de Datos	Tamaño	Primary	Foreign	Not Null	Descripción
mail_id	Int	-	Si		Si	Campo autogenerado y autoincremental
event_type	Varchar	100			Si	Tipo de evento generado por el envío de correo
mail_from	Varchar	100			Si	Correo de Envío
mail_to	Varchar	100			Si	Correo de Destino
mail_date	Datetime	-			Si	Fecha de Envío
mail_subject	Varchar	100			No	Tema de Conversación asociado al correo
isp	Varchar	50			No	Proveedor de correo de destino
customer_to	bigint	-		Si	Si	Campo perteneciente a la plataforma de Shopify y asociación

Nota. Diccionario de Datos de la tabla mf_email. Elaborado por: Los autores

Tabla 25

Tabla de Procedimientos Almacenados

Tabla de Procedimiento Almacenados				
Nombre	Tipo	Parametros	Tipo de Dato	Descripción
sp_I_customer	IN	customer_id_I	Bigint	Este procedimiento se encarga de agregar nueva información relacionada a un cliente.
	IN	name_I	Varchar	
	IN	first_name_I	Varchar	
	IN	last_name_I	Varchar	
	IN	pone_I	Varchar	
	IN	email_I	Varchar	
	IN	currency_I	Currency	
	IN	state_I	Varchar	
	IN	orders_count_I	Int	
	IN	total_spent_I	Decimal	
	IN	accepts_marketing_I	Tinyint	
sp_U_customer	IN	customer_id_U	Bigint	Este procedimiento se encarga de actualizar la información relacionada a un cliente, basado en el código de identificación de este mismo
	IN	name_U	Varchar	
	IN	first_name_U	Varchar	
	IN	last_name_U	Varchar	
	IN	pone_U	Varchar	
	IN	email_U	Varchar	
	IN	currency_U	Char	
	IN	state_U	Varchar	
	IN	orders_count_U	Int	
	IN	total_spent_U	Decimal	
	IN	accepts_marketing_U	Tinyint	

sp_D_customer	IN	customer_id_D	Bigint	Este procedimiento se encarga de eliminar toda la información relacionada con un cliente basado en el código de identificación de este.
sp_I_order	IN	order_id_I	bigint	Este procedimiento se encarga de agregar nueva información relacionada a las ordenes generados en el e-commerce de Shopify.
	IN	order_number_I	int	
	IN	total_weight_I	decimal	
	IN	email_I	varchar	
	IN	pone_I	varchar	
	IN	financial_status_I	varchar	
	IN	currency_I	char	
	IN	prince_I	decimal	
	IN	requires_shipping_I	tinyint	
	IN	customer_id_I	bigint	
	IN	app_id_I	int	
	IN	quantity_I	int	
	IN	product_id_I	bigint	
sp_I_products	IN	product_id_I	Bigint	Este procedimiento se encarga de agregar nueva información relacionada a los productos creado para la comercialización en el e-commerce.
	IN	product_type_I	Varchar	
	IN	status_I	Varchar	
	IN	title_I	Varchar	
	IN	tags_I	Mediumtext	
	IN	vendor_I	Varchar	
sp_I_Address	IN	address_id_I	Bigint	Este procedimiento se encarga de agregar una nueva dirección relacionada a un cliente del e-commerce.
	IN	address1_I	Varchar	
	IN	address2_I	Varchar	
	IN	city_I	Varchar	
	IN	province_I	Varchar	
	IN	province_code_I	Char	

	IN	country_I	Varchar	
	IN	country_code_I	Char	
	IN	county_name_I	Varchar	
	IN	zip_I	Char	
	IN	is_default_I	Tinyint	
	IN	customer_id_O	Bigint	
sp_I_mail	IN	event_type_I	Varchar	Este procedimiento busca encargase de agregar información relacionada al envió de los correos electrónicos.
	IN	email_from_I	Datetime	
	IN	email_to_I	Varchar	
	IN	subject_I	Varchar	
	IN	isp_I	Varchar	
			Varchar	
sp_S_customer_by_App	IN	app_id_G	Int	Este procedimiento busca entregar los clientes almacenados y los productos relacionados basado en las ordenes registradas de cada uno.
sp_s_email_metrics	-	-	-	Este procedimiento busca entregar las métricas relacionada al envió de recomendaciones basado en fechas de envió.
sp_U_customer_marketing	IN	customer_id_U	Bigint	Este procedimiento se encarga de actualizar el estado del marketing por correo basado en el evento generado en el lado del cliente.
	IN	state_U	tinyint	

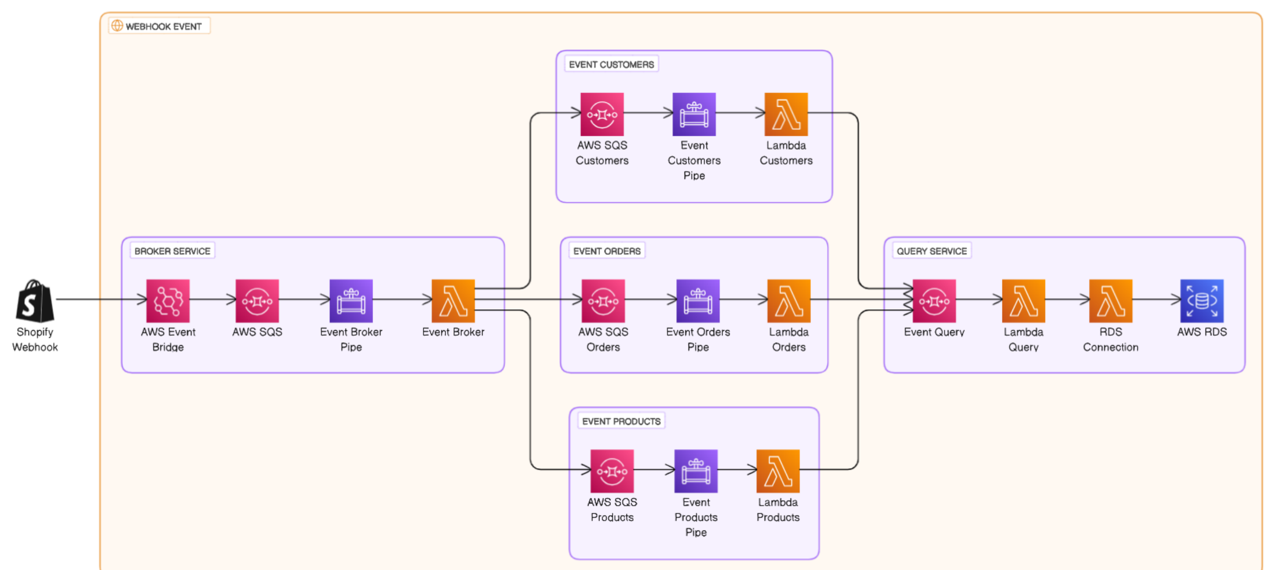
Nota. Definición de Procedimientos Almacenados Implementados. Elaborado por: Los autores

1.11. DESARROLLO DE SERVICIOS PARA CAPTURAR WEBHOOKS DE SHOPIFY

La forma de comunicar los eventos correspondientes a las acciones ejecutadas por el administrador del e-commerce de Shopify es por medio de los webhooks que genera la misma plataforma. Esto webhooks son enviados a cualquier otro sistema mediante una petición POST. En nuestro caso dichos eventos son enviados a la plataforma serverless desarrollada en AWS específicamente al servicio de AWS Event Bridge, debido a esta característica es que la comunicación entre servicios es definida por medio de eventos, dándonos como resultado la arquitectura presentada en la figura 17 para manejar dichos eventos. Y de manera detallada se presentan los eventos de entrada, el servicio encargado de manejar dicho evento y el evento de salida en la tabla 26.

Figura 17

Diagrama de Arquitectura para procesamiento de Webhooks



Nota. Arquitectura para manejo de Webhooks de Shopify en AWS. Elaborado por: Los autores.

Tabla 26
Tabla de Eventos 001

Eventos AWS/Shopify Webhooks			
Shopify Webhook	Evento de Entrada	Servicio	Evento de Salida
Event Broker	Create Order Create Product Update Product Create Customer Update Customer Delete Customer Update Customer Marketing	Broker	Event Parse
Orders Create	Event Parse	Orders	Queries
Products Create	Event Parse	Products	Queries
Products Update	Event Parse		Queries
Customer Create	Event Parse	Customer	Queries
Customer Update	Event Parse		Queries
Customer Delete	Event Parse		Queries
Customer Marketing Update	Event Parse		Queries
Event Query	Queries	Query	Execution Success

Nota. Eventos Webhooks AWS/Spotify capturados. Elaborado por: Los autores.

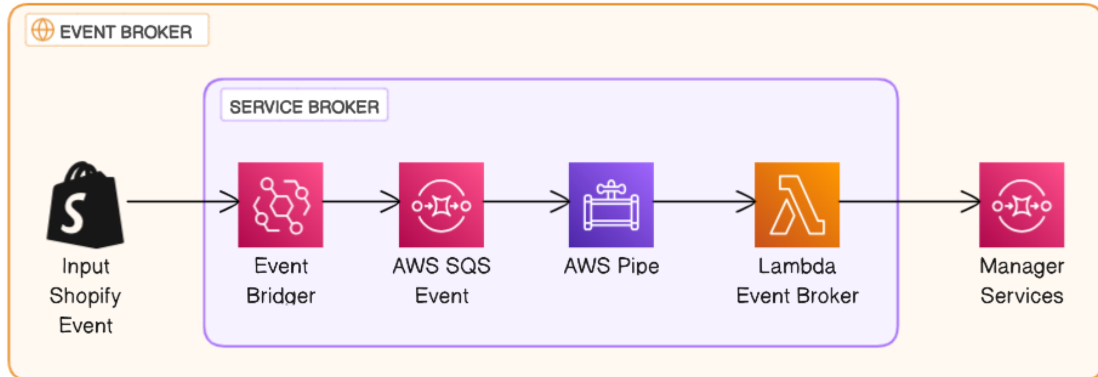
Los servicios encargados de manejar cada uno de los eventos recibidos por parte de la plataforma de Shopify se presentan de manera detallada a continuación.

1.11.1 ***Servicio Broker***

El servicio broker o event broker, es el encargado de determinar a qué servicio va a ir el evento captura para su procesamiento. Este servicio tiene la arquitectura presentada en la figura 18. Donde el servicio Event Bridger posee un bus de evento el cual captura los eventos recibidos y los procesa en una regla la cual es presentada en la figura 19, esto con el fin de enviar dicho evento a un servicio de destino el cual es el servicio SQS como se muestra en la figura 20.

Figura 18

Arquitectura Servicio Event Broker



Nota. Arquitectura de Software del Servicio Broker. Elaborado por: Los autores

Figura 19

Regla de bus de eventos del servicio Event Broker

Nota. Plantilla de patrón para captura de eventos. Elaborado por: Los autores.

Figura 20

Servicio SQS de Destino

Patrón de eventos | Destinos | Supervisión | Etiquetas

Destinos

Editar

Detalles	Nombre del destino	Tipo	ARN	Entrada	Rol
▼	MF-WebhookQueue-Master	Cola de SQS	arn:aws:sqs:us-east-2:336674089181:MF-WebhookQueue-Master	Evento coincidente	-

Entrada para el destino: Evento coincidente

Parámetros adicionales: --

Cola de mensajes fallidos (DLQ): MF-WebhookQueueError (Amazon SQS)

Nota. Servicios de AWS SQS. Elaborado por: Los autores

Posteriormente una vez se recibido el evento en la cola de mensajes, con la configuración presentada en la tabla 27 en el servicio SQS, se le asocio un AWS Event Bridge Pipe con el fin de aplicar un patrón de filtro de eventos, defino en la figura 21 que permita capturar solamente los eventos generados por parte de la plataforma de Shopify.

Tabla 27

Configuración cola SQS Webhook Máster

Cola SQS WebHook Máster	
Type	Standar
Maximun Message Size	256 KB
Message Retention Period	4 days
Default visibility delay	30s
Encrpytion	Amazon SQS Key (SSE-SQS)

Nota. Servicios de Amazon Web Services implementados. Elaborado por: Los autores.

Figura 21

Patrón de filtración de Eventos de Cola SQS Webhook Master

Nota. Patrón de filtrado de eventos. Elaborado por: Los autores

Una vez realizado el filtrado del evento, este es enviado a la función lambda definida como destino y con la configuración presentada en la tabla 28. Lo cual nos permite tener una función que determine el servicio a cuál va a ser dirigido el evento capturado como se presenta en la figura 22.

Tabla 28

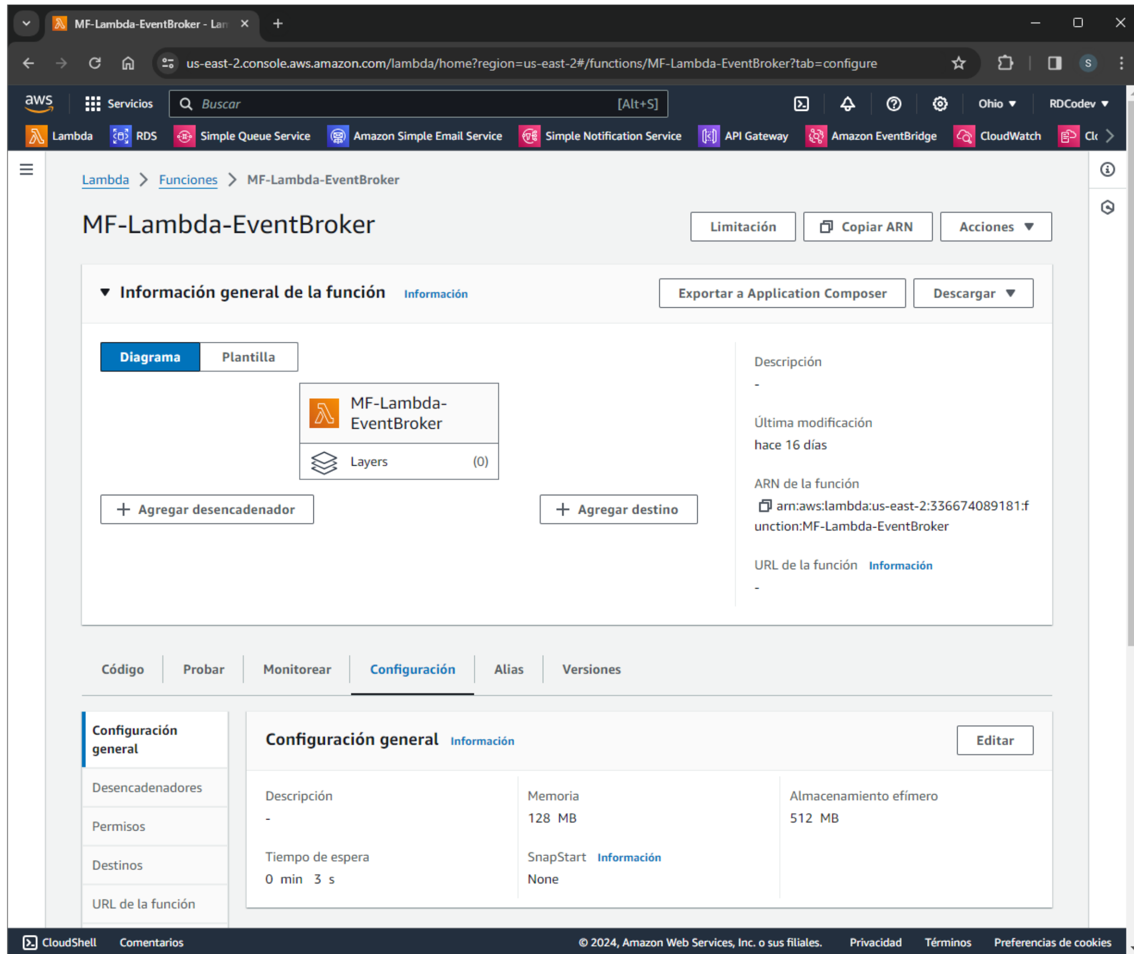
Configuración Función Lambda Event Broker

Configuración Función Lambda Event Broker	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda SQS: ReceiveMessage, DeleteMessage, GetQueueAttributes, CreateLogGroup, CreateLogStream, PutLogEvents
	Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
	SQS: Full Access
Invocación	Asíncrona
Versión	v0.1.0

Nota. Configuración implementada en el servicio SQS. Elaborado por: Los autores.

Figura 22

Función Lambda Event Broker



Nota. *Función Lambda Event Broker. Elaborado por: Los autores.*

El código empleado para determinar el servicio de destino del evento se presenta en la figura 23 y los detalles más importantes de la función en la tabla 29.

Figura 23

Código Función Lambda Event Broker

```
import { SHOPIFY_METADATA_TOPIC } from "./config.mjs";
import { contextAction, retrieveSQSURL, sendSQSMessage } from './utils.mjs'

//********** HANDLER **********
export const handler = async (event) => {
  const [{ body }] = event

  const { detail: { payload, metadata } } = JSON.parse(body);
  const { [SHOPIFY_METADATA_TOPIC]: topic } = metadata;
  const { context, action } = contextAction(topic);

  const { url } = retrieveSQSURL(context);

  try {
    await sendSQSMessage({ action, ...payload }, url);

  } catch (error) {

    throw error
  }

};
```

Nota. Código implementado en funcion lambda event broker. Elaborado por: Los autores.

Tabla 29
Tabla de Código Función Lambda 002

FL002: Código Relevante Función Lambda Event Broker	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
contextAction()	Función encargada de obtener el tipo de acción y el tipo de evento del evento recibido.
retrieveSQSURL()	Función encargada de obtener las direcciones ARN de los diferentes servicios de destino en base al tipo de evento.
sendSQSMessage()	Función encarga de enviar la información del evento a la cola de mensajes del administrador.

Nota. Tabla de detalles relevantes del código de la función lambda Event Broker. Elaborado por: Los autores

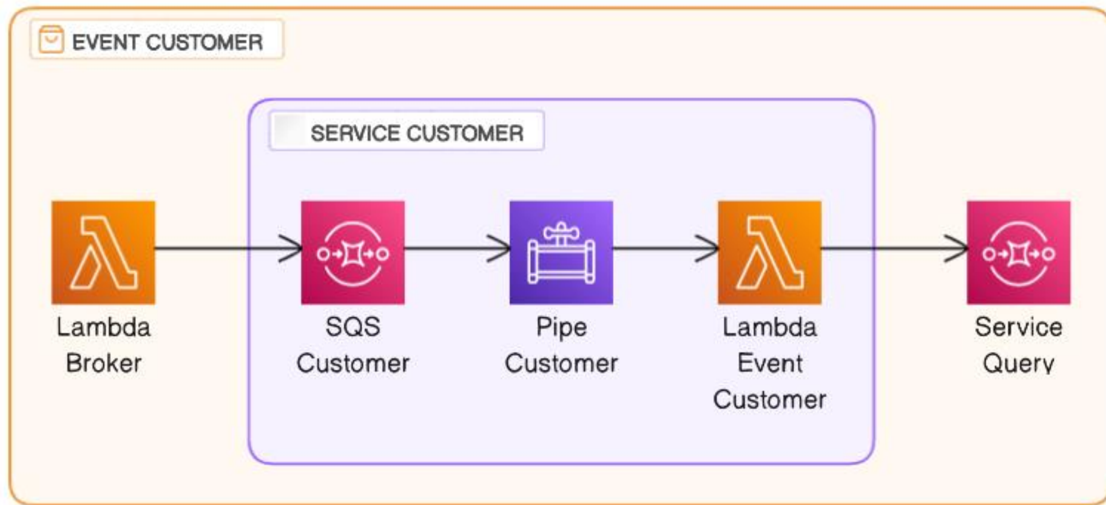
1.11.2 Servicio de Clientes

El servicio de clientes o event customer, es el encargado de procesar la información relacionada a los eventos de los clientes de la plataforma de Shopify. Debido

a esto se definió la arquitectura presentada en la figura 24, para manejar dichos eventos, donde inicialmente se recibe la información del servicio event broker en la cola de mensajes del service customer con la configuración presentada en la tabla 30.

Figura 24

Arquitectura Servicio de Clientes



Nota. Arquitectura de software implementada en el servicio de clientes. Elaborado por: Los autores

Tabla 30
Configuración cola SQS Slave Customer

Cola SQS WebHook Slave Customer	
Type	Estándar
Maximun Message Size	256 KB
Message Retention Period	4 days
Default visibility delay	30s
Encrpytion	Amazon SQS Key (SSE-SQS)

Nota. Configuración implementada en servicio SQS. Elaborado por: Los autores

Recibida la información en la cola de mensajes, se aplicó un AWS Event Bridge Pipe el cual permite realizar una transformación de la información, mediante una plantilla de información de datos, la cual es presentada en la figura 25, esto con el fin de obtener

la información específica antes de que ingrese a la función lambda de destino con la configura presentada en la tabla 31.

Figura 25

Plantilla de datos para transformación del servicio Pipe de evento clientes

Transformador de entrada de destino

```
1 {
2   "customer_id": "<$.body.id>",
3   "name": "<$.body.first_name> <$.body.last_name>",
4   "first_name": "<$.body.first_name>",
5   "last_name": "<$.body.last_name>",
6   "phone": "",
7   "email": "<$.body.email>",
8   "currency": "<$.body.currency>",
9   "state": "enabled",
10  "orders_count": <$.body.orders_count>,
11  "total_spent": "<$.body.total_spent>",
12  "accepts_marketing": <$.body.email_marketing_consent.state>,
13  "default_address": <$.body.default_address.id>,
14  "addresses": {
15    "address_id": <$.body.addresses[*].id>,
16    "address1": <$.body.addresses[*].address1>,
17    "address2": <$.body.addresses[*].address2>,
18    "city": <$.body.addresses[*].city>,
19    "province": <$.body.addresses[*].province>,
20    "province_code": <$.body.addresses[*].province_code>,
21    "country": <$.body.addresses[*].country>,
22    "country_code": <$.body.addresses[*].country_code>,
23    "country_name": <$.body.addresses[*].country_name>,
24    "zip": <$.body.addresses[*].zip>,
25    "is_default": <$.body.addresses[*].default>,
26    "customer_id": <$.body.addresses[*].customer_id>
27  },
28  "action": "<$.body.action>",
29  "marketing_email_update": {
30    "customer_id": <$.body.customer_id>,
31    "marketing_state": "<$.body.email_marketing_consent.state>"
32  }
33 }
```

Nota. Plantilla de transformación para el servicio de clientes. Elaborado por: Los autores

Tabla 31

Configuración Función Lambda Event Customer

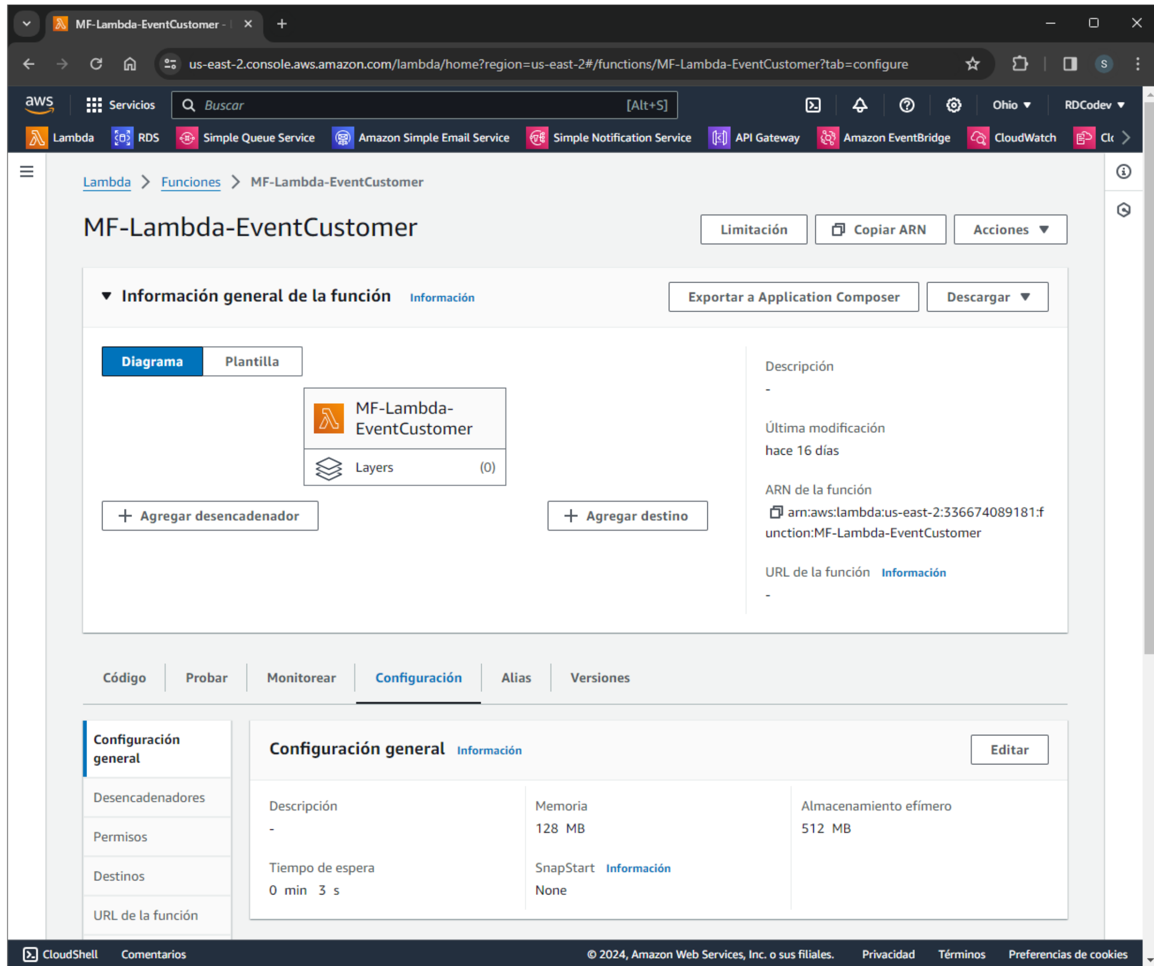
Función Lambda Event Customer	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: BasicExecution
	Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
	SQS: Full Access
Invocación	Asíncrona
Versión	v0.1.0

Nota. Configuración de función lambda para el servicio cliente. Elaborado por: Los autores

Una vez aplicado la configuración tenemos una función lambda la cual se presenta en la figura 26, que sea capaz de administrar la información recibida por parte de los eventos generados por el administrador del e-commerce de Shopify y recibidos por el servicio broker con relación a los clientes.

Figura 26

Función Lambda Event Customer



Nota. Función lambda event customer. Elaborado por: Los autores

Adicional, el código empleado para manejar la información relacionada a los clientes y generar las sentencias SQL para ser enviadas al servicio de Query se presenta en la figura 27 y los detalles más importantes de la función se presentan en la tabla 32.

Figura 27

Código Función Lambda Event Customer

```
import { retrieveSQLStatements, sendSQSMessage } from './utils.mjs'
import { SQS_SQL } from './config.mjs'
//********** HANDLER **********

export const handler = async (event) => {

  const [data] = event;
  const { url } = SQS_SQL;
  const { payload } = retrieveSQLStatements(data);

  try {
    const res = await sendSQSMessage({queries: payload}, url);
    if(res)
      return {
        statusCode: 200,
        body: JSON.stringify(`SQS ${url} delivered.`),
    };


  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify(`Error ${error}.`)
    };
  }
};
```

Nota. Código implementado en la función lambda event customer. Elaborado por: Los autores

Tabla 32

Tabla de Código de Función Lambda 003

FL003: Código Relevante Función Lambda Event Customer	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
retrieveSQLStatements ()	Función encargada procesar la información recibida y obtener la lista de sentencias SQL en forma de arreglo.
sendSQSMessage()	Función encarga de enviar las sentencias SQL generadas al servicio de query para su ejecución.

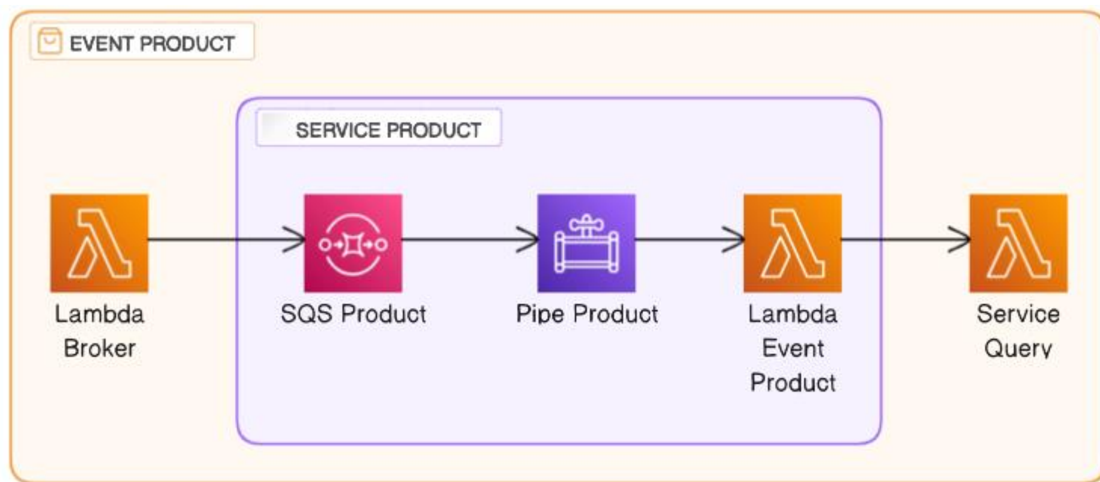
Nota. Tabla de detalles importantes del código de la función lambda Event Customer. Elaborado por: Los autores

1.11.3 Servicio de Productos

El servicio de productos o event product, es el encargado de procesar la información relacionada a los eventos de los productos de la plataforma de Shopify. Debido a esto se definió la arquitectura presentada en la figura 28, para manejar dichos eventos, donde inicialmente se recibe la información del servicio de event broker en la cola de mensajes service products con la configuración presentada en la tabla 33.

Figura 28

Arquitectura Servicio Productos



Nota. Arquitectura de software implementado en el servicio de productos. Elaborado por: Los autores

Tabla 33

Configuración cola SQS Slave Products

Cola SQS WebHook Slave Products	
Type	Standar
Maximun Message Size	256 KB
Message Retention Period	4 days
Default visibility delay	30s
Encrpytion	Amazon SQS Key (SSE-SQS)

Nota. Configuración implementada en el servicio SQS. Elaborado por: Los autores

Recibida la información en la cola de mensajes, se aplicó un AWS Event Bridge Pipe el cual permite realizar una transformación de la información, mediante una plantilla de información de datos presentada en la figura 29, esto con el fin de obtener la información específica antes de que ingrese a la función lambda de destino con la configura presentada en la tabla 34.

Figura 29

Plantilla de datos para transformación del servicio Pipe de evento productos

Transformador de entrada de destino

```
1  {
2    "product_id": <$.body.id>,
3    "product_type": "<$.body.product_type>",
4    "status": "<$.body.status>",
5    "title": <$.body.title>,
6    "tags": "<$.body.tags>",
7    "vendor": "<$.body.vendor>",
8    "variants": [{
9      "variant_id": <$.body.variants[*].id>,
10     "grams": <$.body.variants[*].grams>,
11     "weight": <$.body.variants[*].weight>,
12     "inventory_quantity": <$.body.variants[*].inventory_quantity>,
13     "price": <$.body.variants[*].price>,
14     "product_id": <$.body.variants[*].product_id>
15   }],
16   "action": "<$.body.action>"
17 }
```

Nota. Plantilla para transformación de datos. Elaborado por: Los autores

Tabla 34

Configuración Función Lambda Event Product

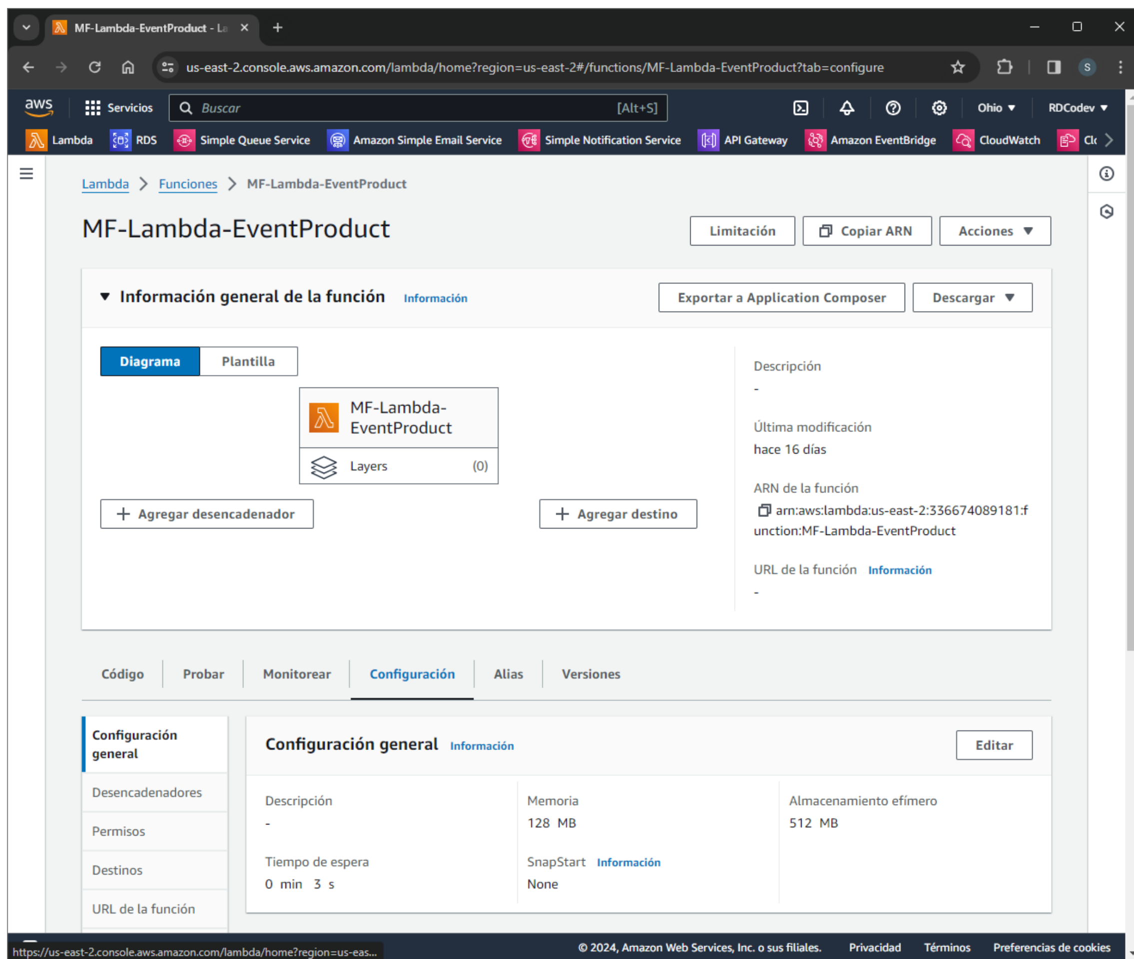
Función Lambda Event Product	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: BasicExecution
	Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
	SQS: Full Access
Invocación	Asíncrona
Versión	v0.1.0

Nota. Configuración implementada en la función lambda. Elaborado por: Los autores

Una vez aplicado la configuración tenemos una función lambda la cual se presenta en la figura 30, que sea capaz de administrar la información recibida por parte de los eventos generados por el administrador del e-commerce de Shopify y recibidos por el servicio broker con relación a los productos.

Figura 30

Función Lambda Event Product



Nota. *Función lambda Event Product. Elaborado por: Los autores*

Adicional, el código empleado para manejar la información relacionada a los productos y generar las sentencias SQL para ser enviadas el servicio de Query se presenta en la figura 31 y los detalles más importantes de la función se presentan en la tabla 35.

Figura 31

Código Función Lambda Event Product

```
import { SQS_SQL } from './config.mjs'
import { retrieveSQLStatements, sendSQSMessage} from './utils.mjs'

//********** HANDLER **********

export const handler = async (event, context) => {

  const [data] = event;

  const { url } = SQS_SQL;
  const { payload } = retrieveSQLStatements(data);

  try {
    const res = await sendSQSMessage({queries: payload}, url);
    if(res)
      return {
        statusCode: 200,
        body: JSON.stringify(`SQS ${url} delivered.`),
    };


  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify(`Error ${error}.`)
    };
  }
};
```

Nota. Código implementado en la función lambda event product. Elaborado por: Los autores

Tabla 35

Tabla de Código de Función Lambda 004

FL004: Código Relevante Función Lambda Event Product	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
retrieveSQLStatements ()	Función encargada procesar la información recibida y obtener la lista de sentencias SQL en forma de arreglo.
SendSQSMessage()	Función encarga de enviar las sentencias SQL generadas al servicio de query para su ejecución.

Nota. Tabla de detalles importantes del código de la función lambda Event Products

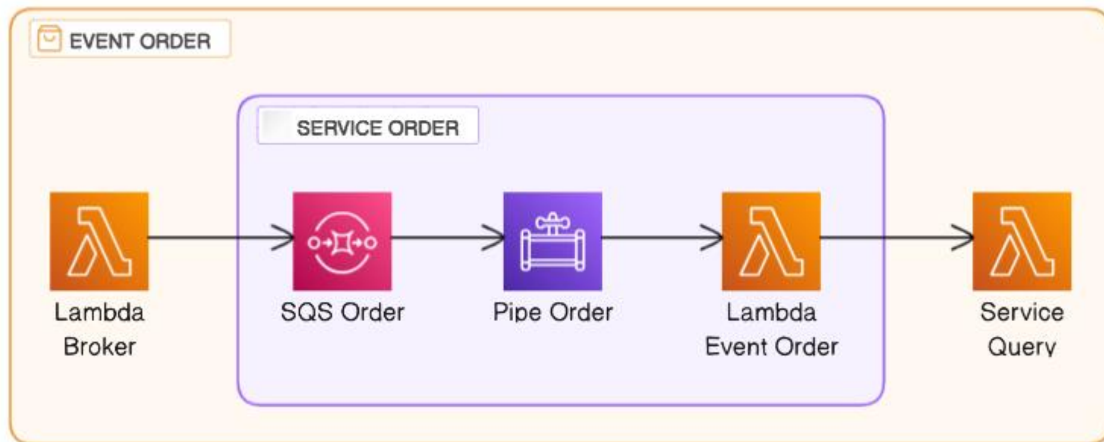
1.11.4 Servicio de Ordenes

El servicio de ordenes o event order, es el encargado de procesar la información relacionada a los eventos de las órdenes generadas en la plataforma de Shopify. Debido a

esto se definió la arquitectura presentada en la figura 32, para manejar dichos eventos, donde inicialmente se recibe la información del servicio de event broker en la cola de mensajes del service order con la configuración presentada en la tabla 36.

Figura 32

Arquitectura servicio Event Order



Nota. Arquitectura de software para el servicio de ordenes. Elaborado por: Los autores

Tabla 36

Configuración Cola SQS WebHook Slave Orders

Cola SQS WebHook Slave Orders	
Type	Standar
Maximun Message Size	256 KB
Message Retention Period	4 days
Default visibility delay	30s
Encrpytion	Amazon SQS Key (SSE-SQS)

Nota. Configuración implementada en el servicio SQS. Elaborado por: Los autores

Recibida la información en la cola de mensajes, se aplicó un AWS Event Bridge Pipe el cual permite realizar una transformación de la información, mediante una plantilla de información de datos presentada en la figura 33, esto con el fin de obtener la información específica antes de que ingrese a la función lambda de destino con la configura presentada en la tabla 37.

Figura 33

Plantilla de datos para transformación del servicio Pipe de evento productos

Transformador de entrada de destino

```
1 {
2   "order_id": <$.body.id>,
3   "order_number": <$.body.order_number>,
4   "quantity": <$.body.line_items[*].quantity>,
5   "total_weight": <$.body.total_weight>,
6   "email": "<$.body.email>",
7   "phone": "",
8   "financial_status": "<$.body.financial_status>",
9   "currency": "<$.body.currency>",
10  "price": <$.body.current_total_price>,
11  "requires_shipping": 1,
12  "customer_id": <$.body.customer.id>,
13  "product_id": <$.body.line_items[*].product_id>,
14  "app_id": <$.body.app_id>,
15  "action": "<$.body.action>"
16 }
```

Nota. Plantilla para transformación de datos. Elaborado por: Los autores

Tabla 37
Configuración Función Lambda Event Customer

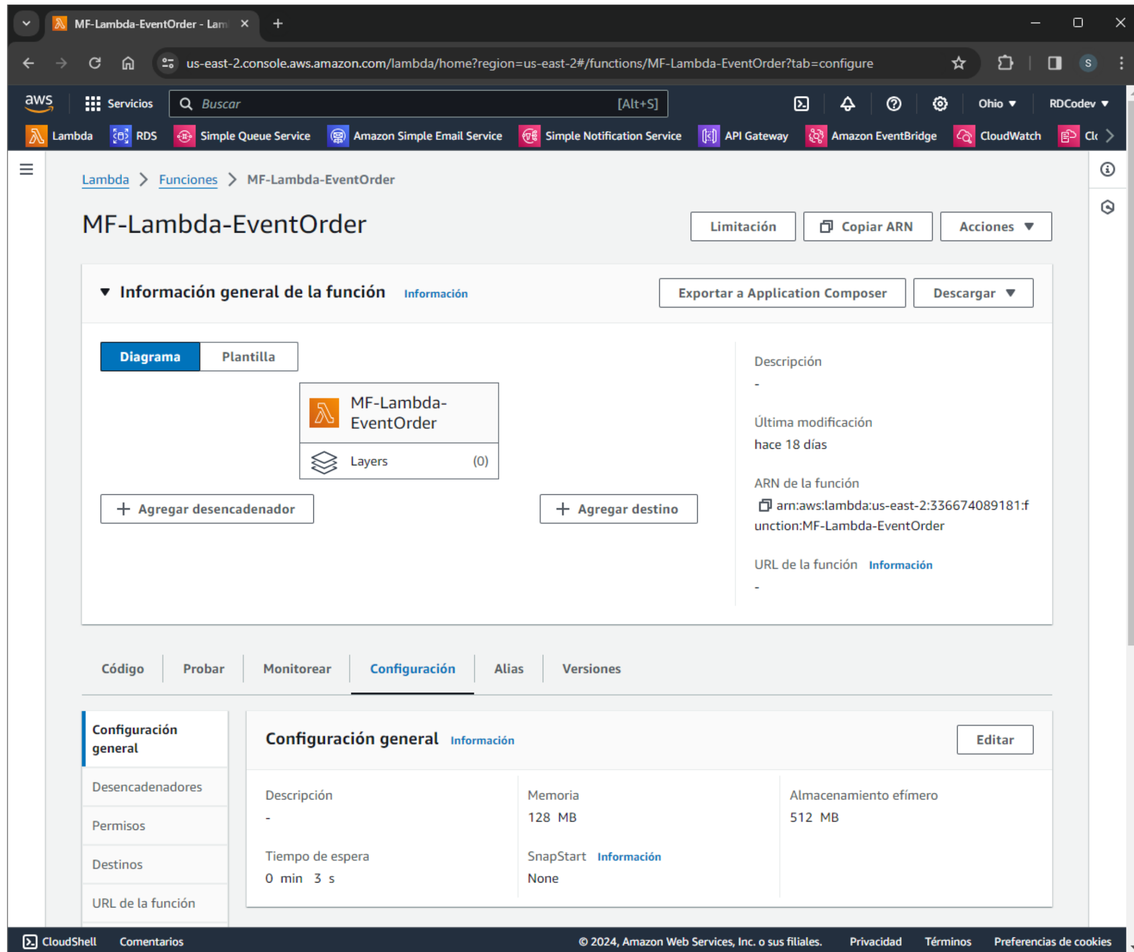
Función Lambda Event Customer	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: BasicExecution
	Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
	SQS: Full Access
Invocación	Asíncrona
Versión	v0.1.0

Nota. Configuración implementada en función lambda. Elaborado por: Los autores

Una vez aplicado la configuración tenemos una función lambda la cual se presenta en la figura 34, que sea capaz de administrar la información recibida por parte de los eventos generados por el administrador del e-commerce de Shopify y recibidos por el servicio broker con relación a las órdenes.

Figura 34

Función Lambda Event Order



Nota. *Función lambda Event Order. Elaborado por: Los autores*

Adicional, el código empleado para manejar la información relacionada a los eventos de las órdenes y generar las sentencias SQL para ser enviadas al servicio de Query se presenta en la figura 35 y los detalles más importantes de la función se presenta en la tabla 38.

Figura 35

Código función Lambda Event Orders

```
import { SQS_SQL } from './config.mjs'
import { retrieveSQLStatements, sendSQSMessage} from './utils.mjs'
//************* HANDLER *********

export const handler = async (event) => {
  const [data] = event;

  const { url } = SQS_SQL;
  const { payload } = retrieveSQLStatements(data);

  console.log(payload)

  try {
    const res = await sendSQSMessage({ queries: payload }, url);
    if (res)
      return {
        statusCode: 200,
        body: JSON.stringify(`SQS ${url} delivered.`),
      };

  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify(`Error ${error}.`)
    };
  }
};
```

Nota. Código implementado en la función lambda. Elaborado por: Los autores

Tabla 38

Tabla Código de Función Lambda 005

FL005: Código Relevante Función Lambda Event Orders	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
retrieveSQLStatements ()	Función encargada procesar la información recibida y obtener la lista de sentencias SQL en forma de arreglo.
SendSQSMessage()	Función encarga de enviar las sentencias SQL generadas al servicio de query para su ejecución.

Nota. Tabla de detalles importantes del código de la función lambda Event Products

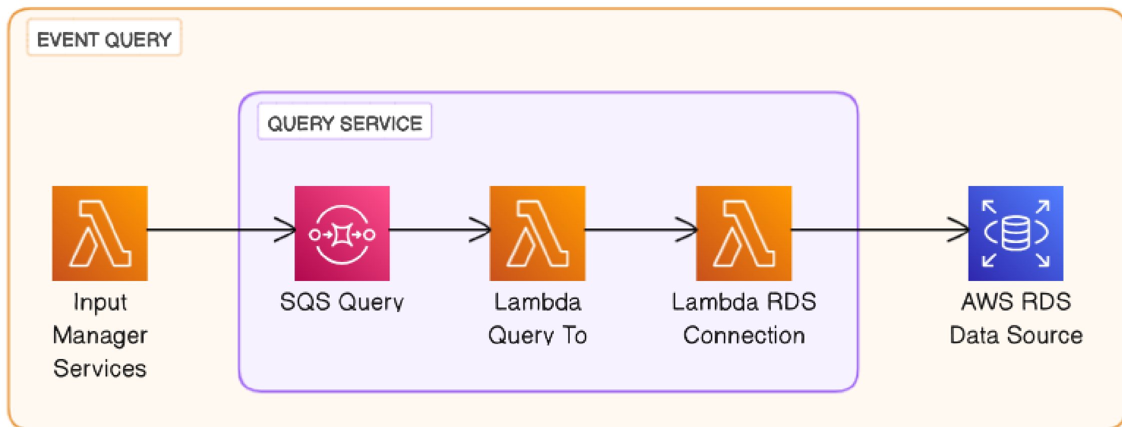
1.11.5 Servicio de Consulta

El servicio de Consulta o event query, es el encargado de ejecutar las funciones SQL generadas por los diferentes servicios destinos detallados anteriormente. Debido a esto se definió la arquitectura presentada en la figura 36, donde inicialmente se recibe la

información de cualquier de los servicios destino en la cola de mensajes con la configuración presentada en la tabla 39.

Figura 36

Arquitectura servicio query



Nota. Arquitectura de software del servicio Query. Elaborado por: Los autores

Tabla 39
Configuración Cola SQS Query

Cola SQS Query	
Type	Estándar
Maximun Message Size	256 KB
Message Retention Period	4 days
Default visibility delay	30s
Encrpytion	Amazon SQS Key (SSE-SQS)

Nota. Configuración implementada en el servicio SQS. Elaborado por: Los autores

Recibida la información en la cola de mensajes, se configuro una función lambda de destino con la configuración presentada en la tabla 40, la cual es la encargada de invocar a la función lambda RDS Connection definida en la figura 14 debido a que tiene acceso a la instancia de la base de datos como se muestra en la figura 12 con el fin de ejecutar las sentencias SQL ingresadas por los parámetros de invocación.

Tabla 40

Configuración Lambda RDS Query To

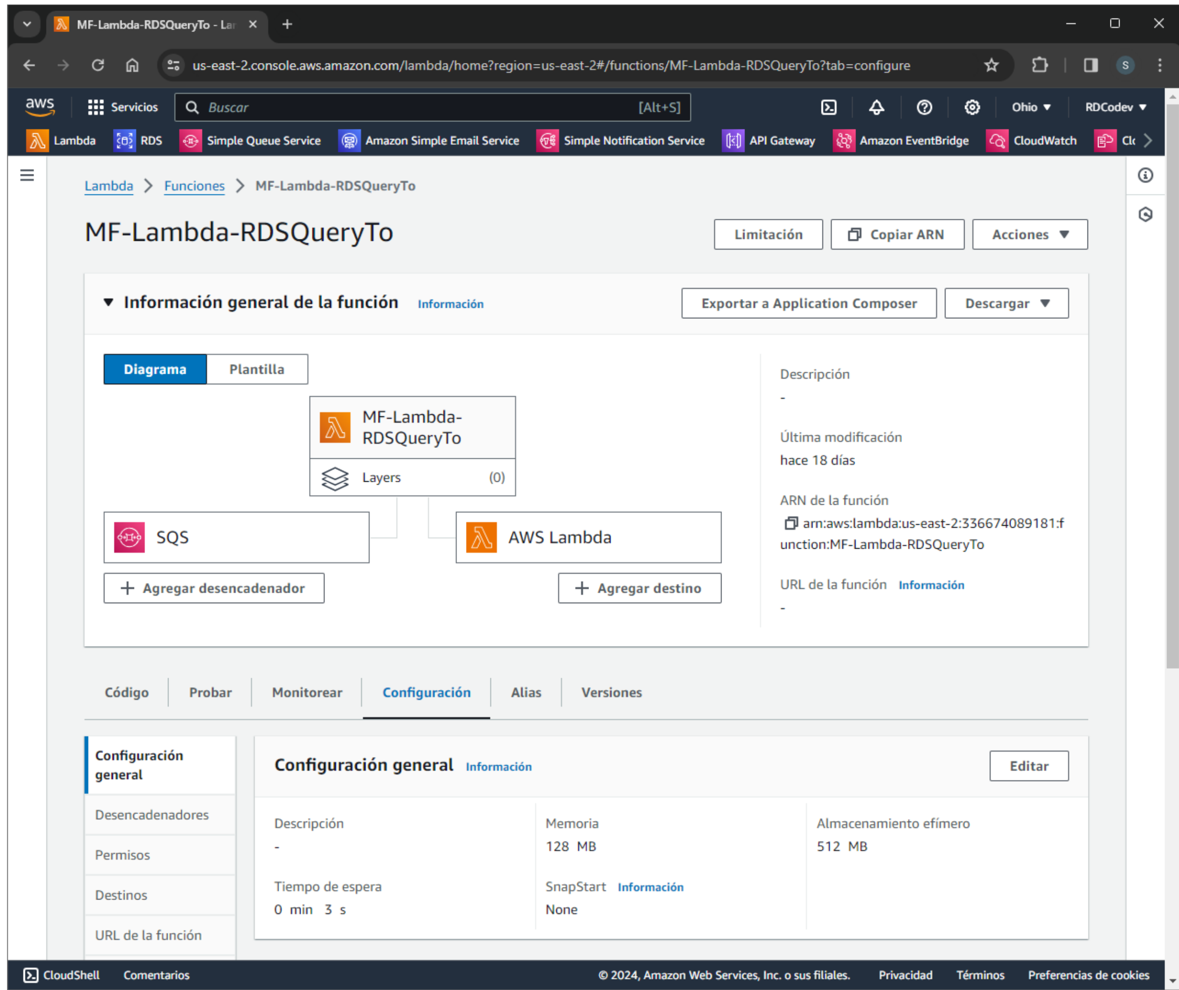
Función Lambda RDS Query To	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: Full Access, InvokeFunctions Logs: PutLogEvents, CreateLogStream, CreateLogGroup. SQS: ReceiveMessage, DeleteMessage, GetQueueAttributes, CreateLogGroup, CreateLogStream, PutLogEvents
Invocación	Asíncrona
Versión	v0.1.0

Nota. Configuración implementada en la función lambda. Elaborado por: Los autores

Una vez aplicado la configuración tenemos una función lambda la cual se presenta en la figura 37, que sea capaz de administrar y reenviar la sentencias SQL recibida por parte de los diferentes servicios de destino para su ejecución en la instancia de la base de datos del servicio RDS por medio de la invocación de la función lambda.

Figura 37

Función Lambda RDS Query To



Nota. Función lambda RDS Query To. Elaborado por: Los autores

El código empleado para manejar la invocación de la función lambda RDS Connection y ejecutar las sentencias SQL se presenta en la figura 38 y los detalles más importantes de la función se presentan en la tabla 41.

Figura 38

Código Función Lambda RDS Query To

```
import { LambdaClient, InvokeCommand } from "@aws-sdk/client-lambda";
import { region, FunctionName } from "./config.mjs";

export const handler = async (event) => {

  try {

    const { Records: [data] } = event
    const { body } = data

    const command = new InvokeCommand(paramsLambdaInvoke(body));
    const { Payload } = await client.send(command)

    const result = parseInvokeResponse(Payload)

    return { result }

  } catch (error) {
    throw error
  }
};
```

Nota. Código implementado en la función lambda. Elaborado por: Los autores

Tabla 41
Tabla de Código Función Lambda 006

FL006: Código Relevante Función Lambda Event Query	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
paramsLambdaInvoke ()	Función encargada de formar los parámetros para la ejecución de la invocación lambda.
InvokeCommand ()	Función encargada de realizar la invocación de la función lambda.
ParseInvokeResponse()	Función encargada de transforma el resultado de la invocación lambda.

Nota. Tabla de detalles importantes del código de la función lambda RDS Query To Elaborado por: Los autores

1.12. DESARROLLO SERVICIO API GATEWAY EN AWS

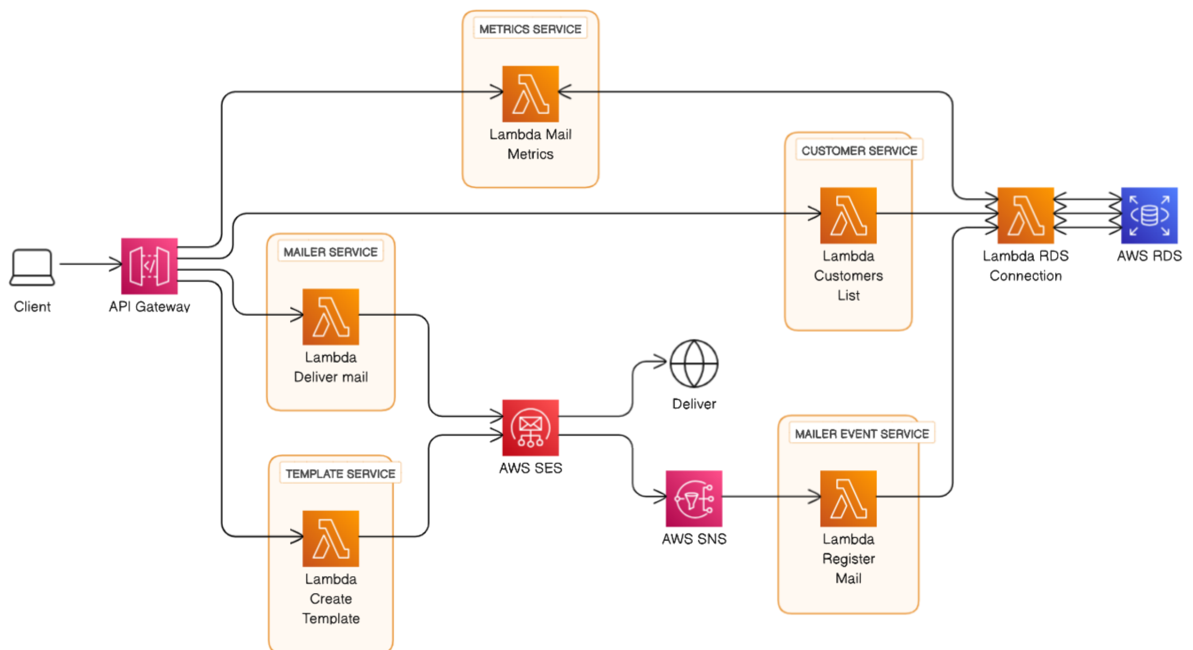
Se definió el servicio API Gateway en la plataforma serverless con el fin de permitir al usuario del e-commerce de Shopify acceder a los recursos definidos en la API REST mediante peticiones HTTP y poder recuperar la información guardada en la

instancia de la base de datos o interactuar con la lógica para llamadas de otros servicios instanciados en AWS.

La arquitectura definida para manejar dichas interacciones y realizar la comunicación entre los diferentes servicios de AWS se presenta en la figura 39. Además, la configuración implementada en el servicio API Gateway se presenta en la tabla 42 y de manera detallada los diferentes recursos definidos en la API REST se presentan en la tabla 44.

Figura 39

Diagrama arquitectura servicio API Gateway



Nota. Diagrama del funcionamiento de la arquitectura de API Gateway Elaborado por: Los Autores

Tabla 42
Configuración servicio API Gateway

API Gateway	
Protocolo	REST
API Name	MF-REST-App
Stage Development	Enabled
Binary media types	All Types (*)

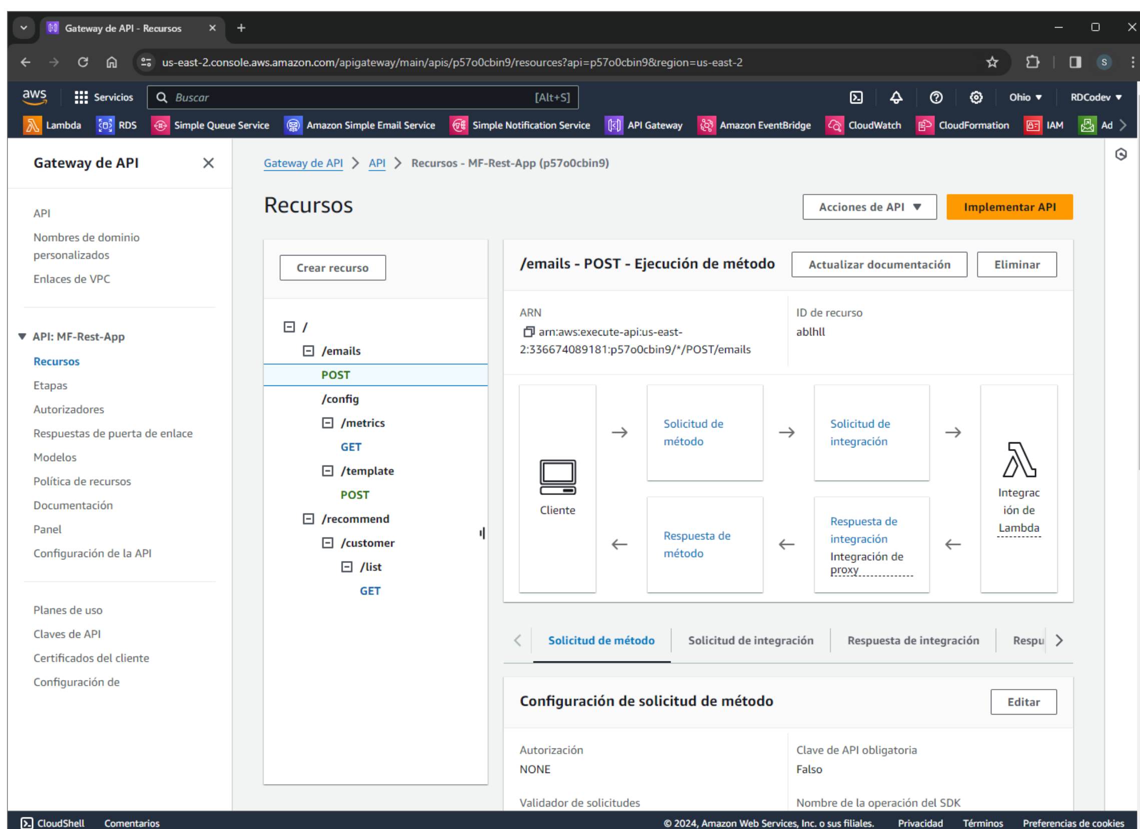
Content encoding	Inactive
Request rate	1000 per second
Quote per day	1000

Nota. *Tabla de configuraciones al servicio de API Gateway Elaborado por: Los Autores*

Una vez aplicada la configuración presentada tenemos una API REST, en la cual podemos agregar recursos e integraciones con diferentes servicios con el fin de administrar las diferentes peticiones, justo como se muestra en la figura 40.

Figura 40

Servicio API Gateway/API REST



Nota. *Servicio de API/REST para agregar recursos e integraciones Elaborado por: Los Autores*

Adicionalmente como se muestra en la figura 39, los diferentes servicios interactúan con el servicio de AWS SES y este mismo al ser un servicio independiente pero esencial, se lo inicializo con la configuración presentada en la tabla 43. En donde al crear el servicio, es menester verificar las identidades de correo electrónico y dominios,

este último es necesario para el proceso de verificación de correos por parte de los diferentes proveedores de servicio de correos. Por lo que se configuro los DKIM (DomainKeys Identified Mail) en los DNS del dominio adquirido como se muestra en la figura 41.

Tabla 43

Configuración Servicio AWS Simple Email Service

AWS Simple Email Service	
Daily sending quota	50000 emails per 24h
Maximum send rate	14 per second
SMTP TLS Port	25, 587, 2587
DomainKeys Identified Mail (DKIM)	RSA_2038_BIT

Nota. Tabla de configuración del servicio de automatización de correo electrónicos elaborado por: Los Autores

Figura 41

Dominio en proveedor GoDaddy

Filtros ··· Acciones

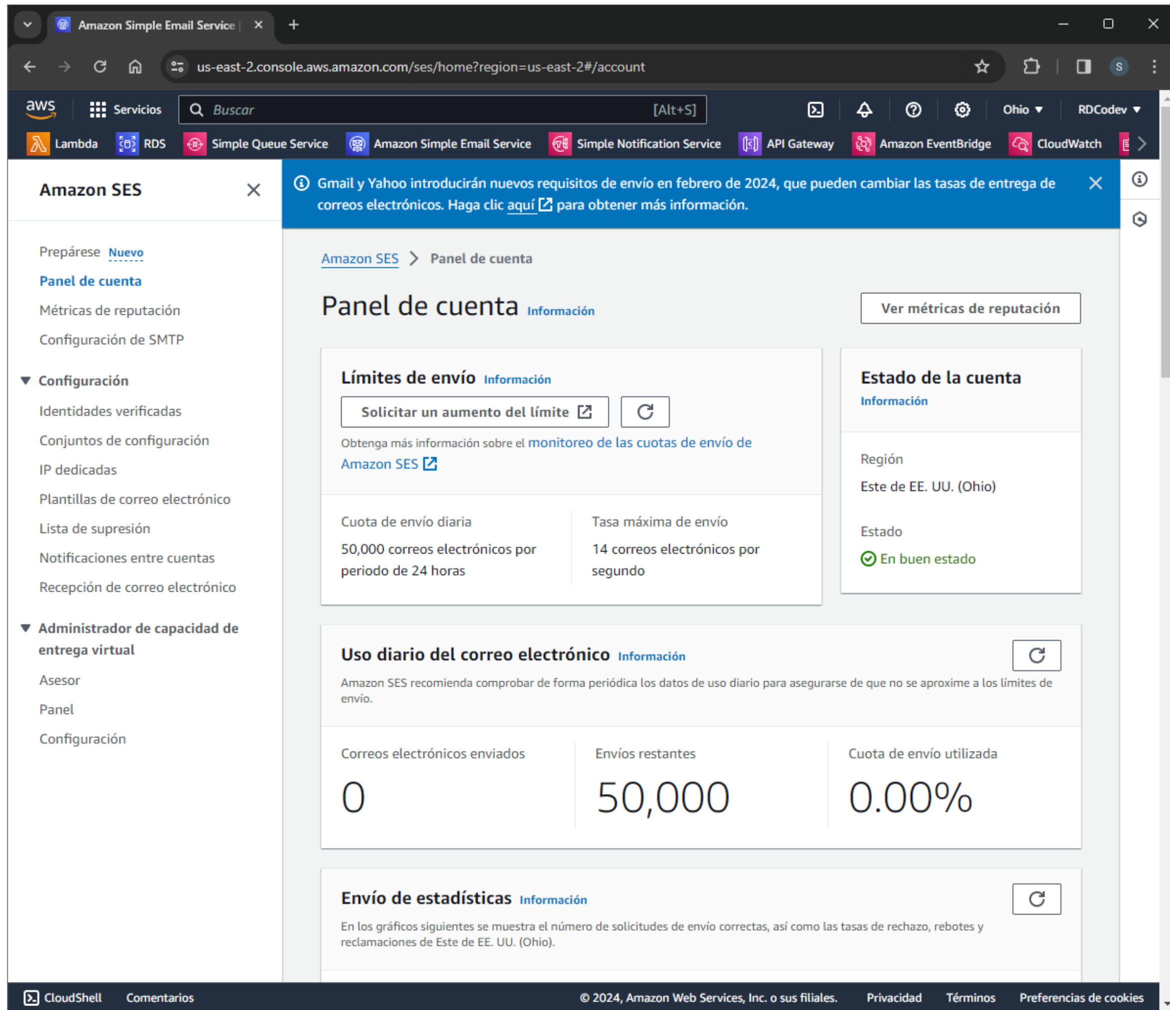
	Tipo	Nombre	Datos	TTL	Eliminar	Editar
	CNAME	4v4swtftovo4hox6rvg5w4jawxenquub._domainkey	4v4swtftovo4hox6rvg5w4jawxenquub.dkim.amazonses.com.	1 Hora		
	CNAME	wlojd5eii37tl2piiyzutmboswj53gcz._domainkey	wlojd5eii37tl2piiyzutmboswj53gcz.dkim.amazonses.com.	1 Hora		
	CNAME	www	minifashions.shop.	1 Hora		
	CNAME	xzxlx7j5mqtsqpwxt7sywa4etkhitqph._domainkey	xzxlx7j5mqtsqpwxt7sywa4etkhitqph.dkim.amazonses.com.	1 Hora		

Nota. Configuración CNAME en los DNS del dominio minifashion.shop en GoDaddy

Una vez aplicada la configuración y verificada las identidades obtuvimos el servicio SES para la recepción y envío de correos electrónicos, justo como se muestra en la figura 42.

Figura 42

Servicio AWS Simple Email Service



Nota. Funcionamiento del servicio de automatización de correos Elaborado por: Los Autores

Tabla 44

Tabla de recursos API REST

API Gateway Recursos						
Recurso	Dirección	Método	Content-Type	Body	Query Params	Descripción
Deliver Email	/emails	POST	application/json	Source: Correo electronico de origen Template: Nombre de la plantilla registrada en AWS SES Destionation/ToAddresses: Arreglo con las direcciones de correo electrónico de destino TemplateData/name: Nombre del cliente a ser reemplazo en la plantilla TemplateData/products: Arreglo de productos a ser reemplazo en la platnilla, estos tienen título del producto, precio anterior y precio actual al aplicar el descuento, y url de la imagen de producto.	NA	Recurso que captura información para enviarse por correo electrónico, usando el servicio AWS SES.
Create Template	/emails/template	POST	application/multi-form-data	TemplateName: Nombre de la plantilla a registrar en el servicio de AWS SES SubjectPart: Asusto asociar en el correo electrónico. HtmlPart: Plantilla html con sintaxis handlebars.	NA	Recurso que captura información para crear una plantilla de correo electrónico mediante el servicio AWS SES.
Customers	/recommend/customer/list	GET	application/json	NA	App_id: Identificador de la aplicación asociada a las ordenes registradas.	Recurso para recuperar información relacionada a los clientes registrados en la base de datos.
Mail Metrics	/emails/metrics	GET	application/json	NA	NA	Recurso para recuperar información de eventos registrados en la base de datos.

Nota. Tabla de los recursos que maneja la API/REST Elaborado por: Los Autores

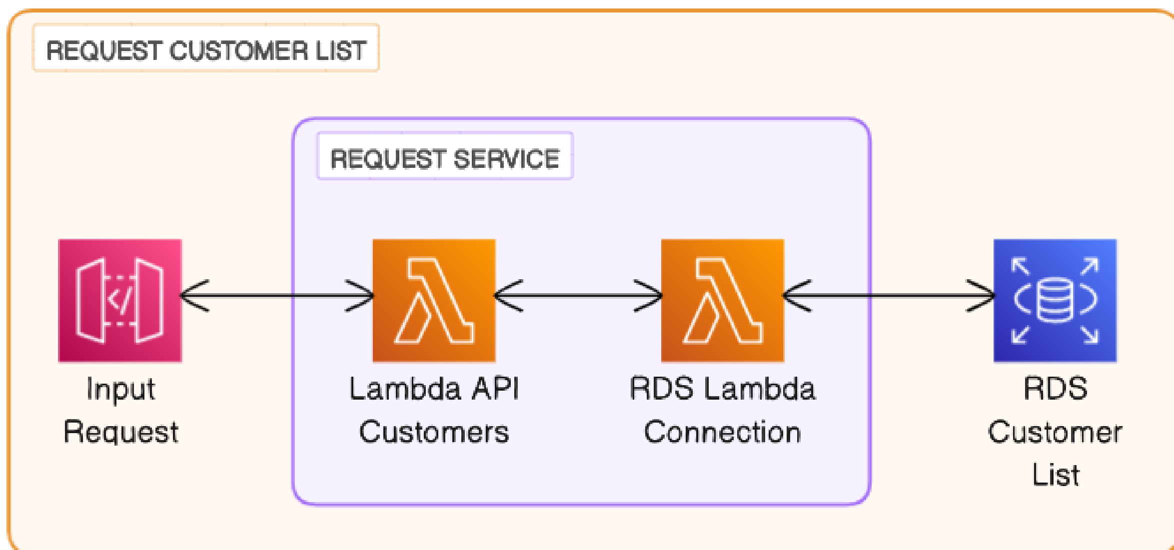
A continuación, se detallan los diferentes recursos y sus interacciones con los diferentes servicios.

1.12.1 Recurso Lista de Clientes

El recurso "lista de clientes" o "customers" presentado en la tabla 44 se encarga de recibir las peticiones del usuario en la ruta /recommend/customer/list con el fin de realizar la recuperación de la información de los clientes mediante la integración de una función lambda que posee la configuración presentada en la tabla 45, en la instancia en la base de datos definida en el servicio RDS, como se muestra en la figura 43.

Figura 43

Arquitectura Recurso Customers



Nota. Diagrama de la arquitectura del recurso de lista de clientes Elaborado por: Los Autores

Tabla 45
Configuración Función Lambda Customer List

Función Lambda Customer List	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: Full Access, InvokeFunctions Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocacion	Asincrona
Version	v0.1.0

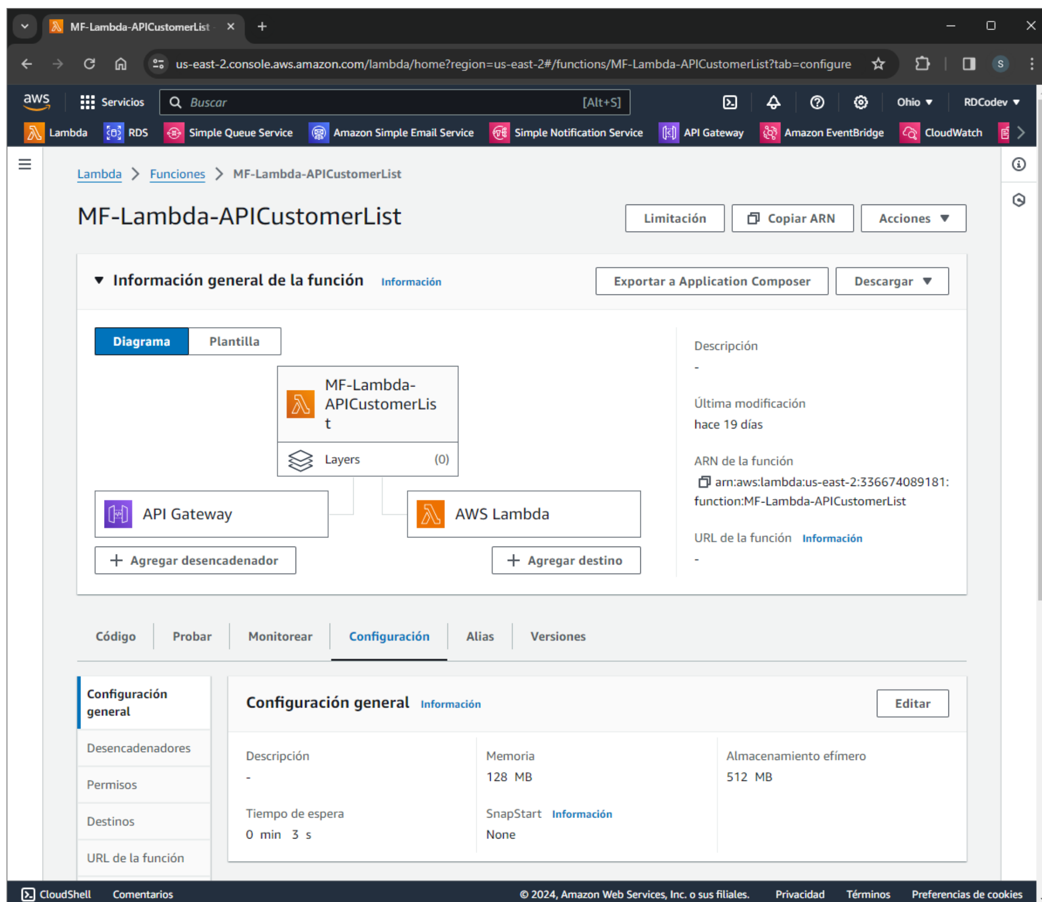
Triggers	API Gateway: MF-REST-App
Destinations	Lambda: RDS Connection

Nota. Tabla de la configuración principal de la función Lambda Customer List Elaborado por: Los Autores

Una vez integrada la función lambda en el recurso publicado de la API REST junto con su configuración, obtuvimos una función lambda, presentada en la figura 44, la cual se ejecuta en cada petición con el fin de realizar una invocación a la función RDS Connection y obtener un resultado el cual es enviado como respuesta a dicha petición.

Figura 44

Función Lambda API Customer List



Nota. Función Lambda API Customer List implementada Elaborado por: Los Autores

El código implementado para el procesamiento de la petición y generación de respuesta se presenta en la figura 45 y los detalles más relevantes de dicha función se presenta en la tabla 46.

Figura 45

Código Función Lambda API Customer List

```
import { LambdaClient, InvokeCommand } from "@aws-sdk/client-lambda";
import { REGION } from './config.mjs'
import { paramsLambdaInvoke, parseInvokeResponse } from './utils.mjs'
import { sqlEndPoints } from './uris.mjs'

const client = new LambdaClient({ region: REGION })

export const handler = async (event) => {

  const {
    queryStringParameters: params,
    requestContext: { httpMethod } } = event

  const [sql] = sqlEndPoints.filter(endpoint => endpoint.httpRequest == httpMethod)

  try {

    const queries = { queries: [sql.query(params.app_id)] }

    const command = new InvokeCommand(paramsLambdaInvoke(JSON.stringify(queries)));
    const { Payload } = await client.send(command)
    const { result } = parseInvokeResponse(Payload)

    const [customers] = result

    return {
      statusCode: 200,
      body: JSON.stringify(customers)
```

Nota. Codigo para el funcionamiento de la función Lambda Elaborado por: Los Autores

Tabla 46

Tabla de Código Función Lambda 007

FL007: Código Relevante Función Lambda Customer List	
Función	Descripción
paramsLambdaInvoke ()	Función encargada de formar los parámetros para la ejecución de la invocación lambda.
InvokeCommand ()	Función encargada de realizar la invocación de la función lambda.
paramsLambdaInvoke ()	Función encargada de formar los parámetros para la ejecución de la invocación lambda.
ParseInvokeResponse()	Función encargada de transforma el resultado de la invocación lambda.

Nota. Tabla de detalles importantes del código de la función lambda API Customer List

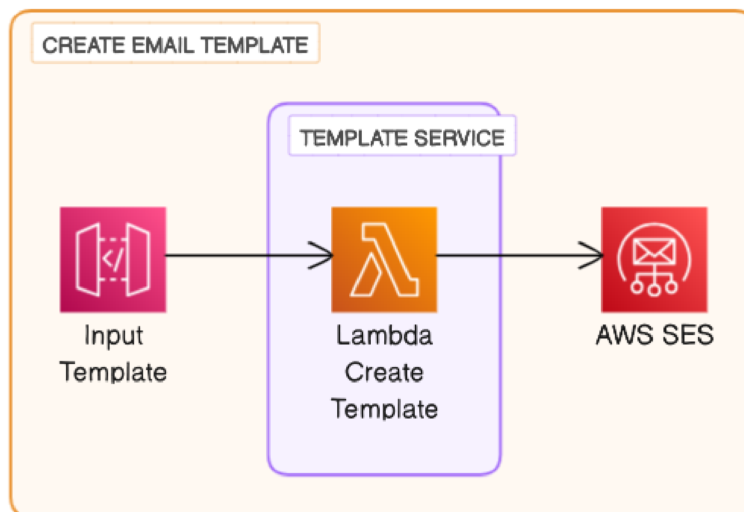
1.12.2 Recurso Plantilla de Correo

El recurso plantilla de correo o *template email* presentado en la tabla 44 se encarga de recibir las peticiones en la ruta /email/template por medio de la integración de una función lambda con la configuración presentada en la tabla 47. Esto con el fin de procesar la información que viene asociada a dichas peticiones en el body o cuerpo de la petición e interactuar con el servicio de AWS SES previamente inicializo en la figura 42.

En la figura 46 se presenta la arquitectura implementada para realizar dicho proceso.

Figura 46

Arquitectura Recurso Create Template



Nota. Diagrama de la arquitectura del recurso créate template Elaborado por: Los Autores

Tabla 47
Configuración Función Lambda Email Template

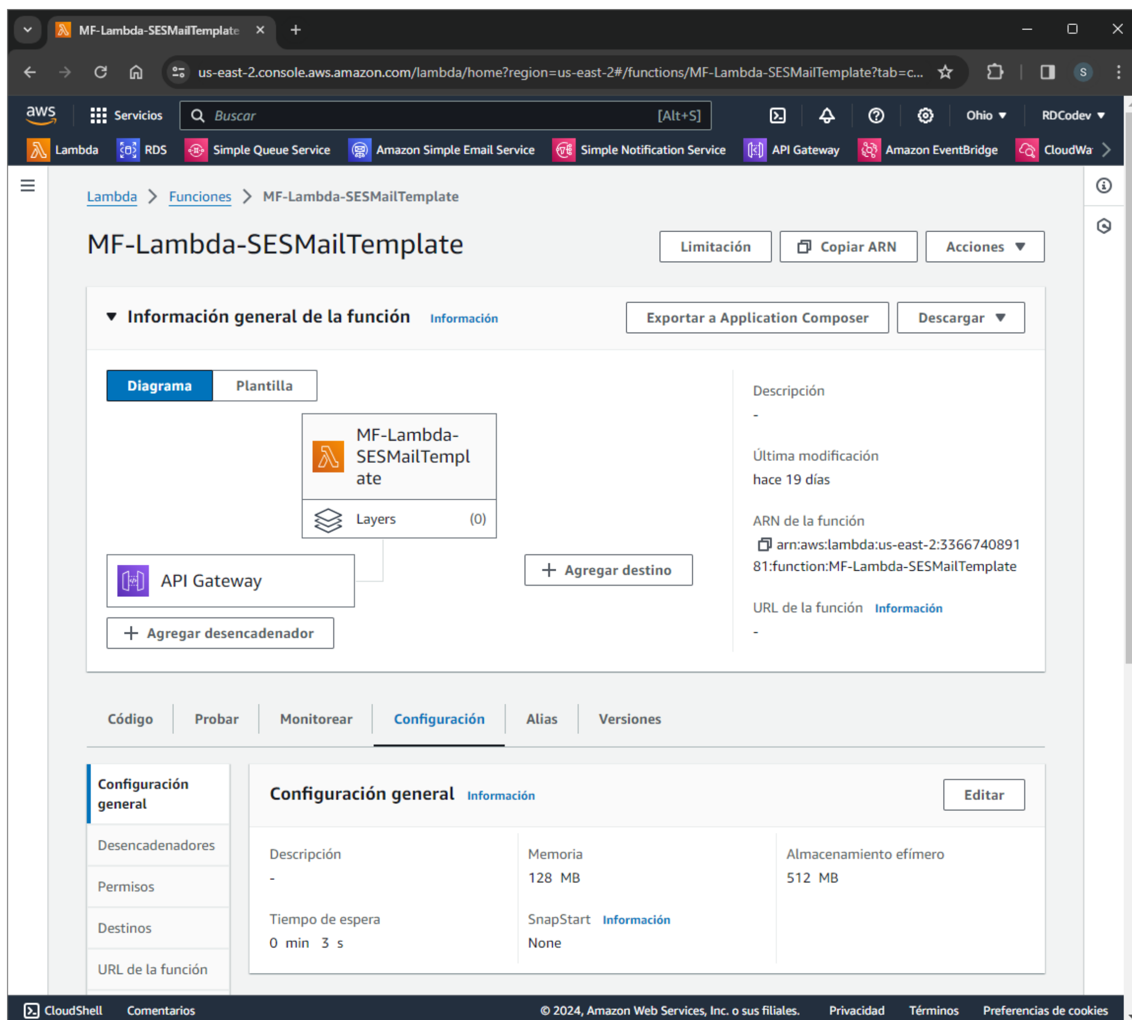
Función Lambda Email Template	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	SES: Full Access Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocación	Asíncrona
Versión	v0.1.0
Triggers	API Gateway: MF-REST-App

Nota. Tabla de la configuración de los parámetros importantes de la función Lambda Email Elaborado por: Los Autores

Una vez integrada la función lambda en el recurso publicado en el servicio API Gateway junto con su configuración, obtuvimos una función lambda, presentada en la figura 47, la cual puede interactuar con el servicio SES para realizar la creación de plantillas de correo en cada petición recibida.

Figura 47

Función Lambda SES Email Template



Nota. Función Lambda SES Email implementada Elaborado por: Los Autores

El código implementado para el procesamiento de la petición y generación de plantillas en el servicio AWS SES se presenta en la figura 48 y los detalles más relevantes de dicha función se presenta en la tabla 48.

Figura 48

Código Función Lambda Email Template

```
import { SESClient, CreateTemplateCommand } from '@aws-sdk/client-ses';
import multipart from 'parse-multipart-data';
import { REGION } from './config.mjs'
import { parseTemplate } from './utils.mjs'

const client = new SESClient({region: REGION});

export const handler = async (event, context, callback) => {

  const { headers, body } = event
  const boundary = headers['Content-Type'].split('=')[1]
  const buffer = Buffer.from(body, 'base64');

  const data = multipart.parse(buffer, boundary)

  const template = {
    Template: parseTemplate(data)
  }

  try{

    const command = new CreateTemplateCommand(template);
    const response = await client.send(command)

    res.body =  JSON.stringify({"message": response})
  } catch (err){
    res.body =  JSON.stringify({"message": err, template})
  } finally {
```

Nota. Código para el funcionamiento de la función Lambda Email Elaborado por: Los Autores

Tabla 48

Tabla Código Función Lambda 008

FL008: Código Relevante Función Lambda Email Template	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
multipart.parse()	Método de la librearía multipart para convertir del formato multipart-form-data al formato json
CreateTemplateCommand ()	Clase para crear los parámetros de creación de plantillas de correo para el servicio SES
SESClient	Clase para obtener el cliente de ejecución de servicios o comandos de AWS
client.send()	Método para ejecutar comando en servicios de AWS

Nota. Tabla de detalles relevantes del código de la función lambda Email Template

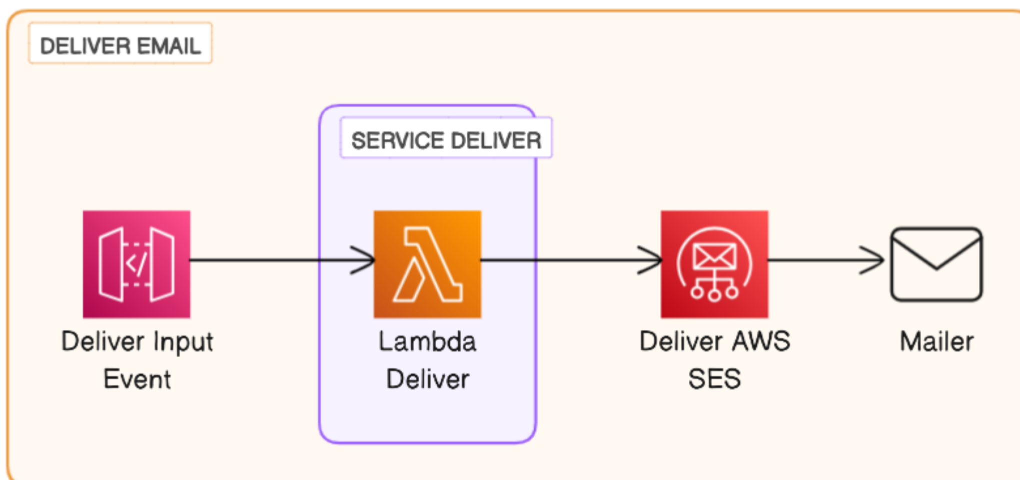
1.12.3 Recurso Envió de Correo

El recurso envió de correo o *deliver email* presentado en la tabla 44 se encarga de recibir las peticiones en la ruta /emails/ por medio de la integración de una función lambda con la configuración presentada en la tabla 49, esto con el fin de procesar la información que viene asociada a dicha peticiones en el body o cuerpo de la petición e interactuar con el servicio de AWS SES previamente inicializado en la figura 42 para él envió de correos electrónicos.

En la figura 49 se presenta la arquitectura implementada para realizar dicho proceso.

Figura 49

Arquitectura del Recurso Deliver Email



Nota. Diagrama de cómo funciona el recurso de deliver email Elaborado por: Los Autores

Tabla 49

Configuración Función Lambda Deliver Email

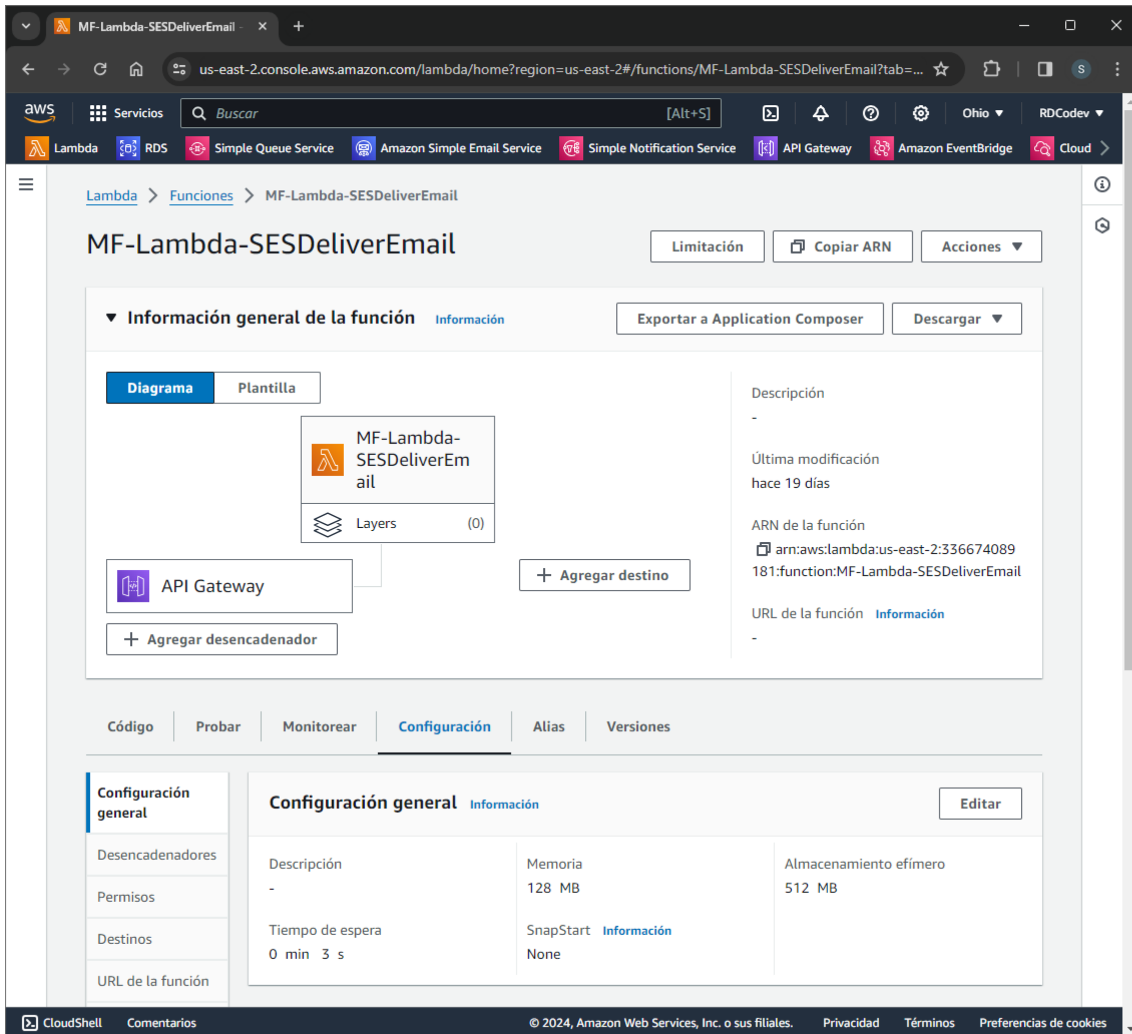
Función Lambda Deliver Email	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	SES: Full Access Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocación	Asíncrona
Versión	v0.1.0
Triggers	API Gateway: MF-REST-App

Nota. Tabla de configuración de los parámetros importantes de la función Lambda deliver
Elaborado por: Los Autores

Una vez integrada la función lambda en el recurso publicado en el servicio API Gateway junto con su configuración, obtuvimos una función lambda, presentada en la figura 50, la cual puede interactuar con el servicio SES para realizar él envió de correo electrónico a los usuarios de destino en cada petición recibida y hacer uso de las plantillas de correo definidas en el servicio de AWS SES.

Figura 50

Función Lambda SES Deliver Email



Nota. *Función Lambda SES deliver email implementado Elaborado por: Los Autores*

El código implementado para el procesamiento de la petición y envió de correos electrónicos se presenta en la figura 51 y los detalles más relevantes de dicha función se presenta en la tabla 50.

Figura 51

Código Función Lambda SES Deliver Email

```
import { SESClient, SendTemplatedEmailCommand  } from "@aws-sdk/client-ses";
import { REGION } from './config.mjs'

const client = new SESClient({region: REGION});

export const handler = async (event, context, callback) => {

  const { body } = event
  const buffer = Buffer.from(body, 'base64');

  const formData = JSON.parse(buffer.toString())

  formData.TemplateData = JSON.stringify(formData.TemplateData)

  const command = new SendTemplatedEmailCommand(formData);

  try {
    const response = await client.send(command);
    return {
      statusCode: 200,
      body: JSON.stringify({"message": response})
    }
  } catch (err) {

    return {
      statusCode: 400,
      body: JSON.stringify({"message": err})
    }
  }
```

Nota. Código para que funcione la función lambda Deliver Email Elaborado por: Los Autores

Tabla 50
Tabla Código Función Lambda 009

FL009: Código Relevante Función Lambda Deliver Email	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
SendTemplatedEmailCommand ()	Clase encargada de generar el comando para enviar correos electrónicos junto con sus plantillas a los clientes de destino.
SESClient ()	Clase para obtener el cliente de ejecución de servicios o comandos de AWS.
client.send()	Método para ejecutar comando en servicios de AWS

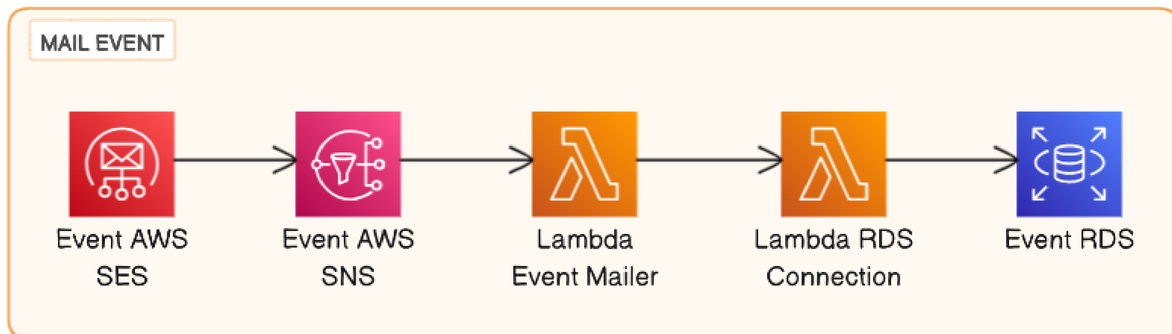
Nota. Tabla de detalles importantes del código de la función lambda Deliver Email Elaborado por: Los Autores

1.12.4 Servicio de Registro de Eventos de Envió de Correo

El servicio de registro de eventos de envió de correo, se lo definió como un servicio que administra los eventos generados por los servidores de los clientes de correos de destino, esto con el fin de registrar dichos eventos en la instancia de base de datos y sean recuperable la información para el análisis o procesamiento de estos. En la figura 52 se presenta la arquitectura para realizar dicho proceso. En donde cada evento generado en el servicio AWS es capturado por medio del set de configuración presentado en la tabla 51, el cual enviara cada evento al servicio SNS con la configuración presentada en la tabla 52.

Figura 52

Arquitectura para captura de eventos en servicio SES



Nota. Código para que funcione la función lambda Deliver Email Elaborado por: Los Autores

Tabla 51
Configuración de AWS SES configurations sets

AWS SES Configurations sets		
Destination type	Amazon SNS	-
Event publishing	Enabled	-
Event types	Hard bounces	Evento relacionado al rechazo permanente del correo electrónico.
	Clicks	Evento relacionado al ocurrir un evento de tipo click en el cuerpo del correo electrónico.
	Complaints	Evento relacionado al marcar al correo electrónico como spam.
	Deliveries	Evento relacionado al correo electrónico y cuando es entregado correctamente al servidor de correo del destinatario.
	Opens	Evento relacionado al correo electrónico y cuando este es abierto de la bandeja de entrada por parte del destinatario.
	Rejects	Evento relacionado al correo electrónico y cuando el servicio SES determino que poseía virus, evitando la entrega al servidor de correo del destinatario.
	Sends	Evento relacionado al correo electrónico y cuando el servicio SES recibió la ejecución e intenta entregar el mensaje al servidor de correo del destinatario.
	Subscriptions	Evento relacionado al correo electrónico y cuando el destinatario actualizo la suscripción asociada en el correo electrónico.
SNS Topic	AWS SES Deliver	-

Nota. Tabla de configuración de los parámetros importantes de AWS SES Elaborado por: Los Autores

Tabla 52
Configuración AWS SNS SES Deliver

AWS SNS SES Deliver	
Name	AWS_SES_Deliver
Type	Standard
Encryption	Disabled
Subscription Protocol	Lambda
Subscription Status	Confirmed
Subscription End Point	MF-Lambda-EvetMailer
Http Deliver Delay	20s
Http Deliver Retries	3

Nota. Tabla de la configuración de los parámetros importantes de AWS SENS Elaborado por: Los Autores

Una vez aplicada las configuraciones para captura de eventos, se integró una función lambda con la configuración presentada en la tabla 53, la cual será ejecutada en cada notificación recibida en el servicio SNS. Esto con el fin de conseguir una función lambda que registre cada evento surgido en el servicio de AWS SES, en la instancia de la base de datos del

servicio RDS, por medio de la invocación de la función lambda RDS Connection definida en la figura 14. En la figura 53 se presenta la función lambda definida para este propósito.

Tabla 53

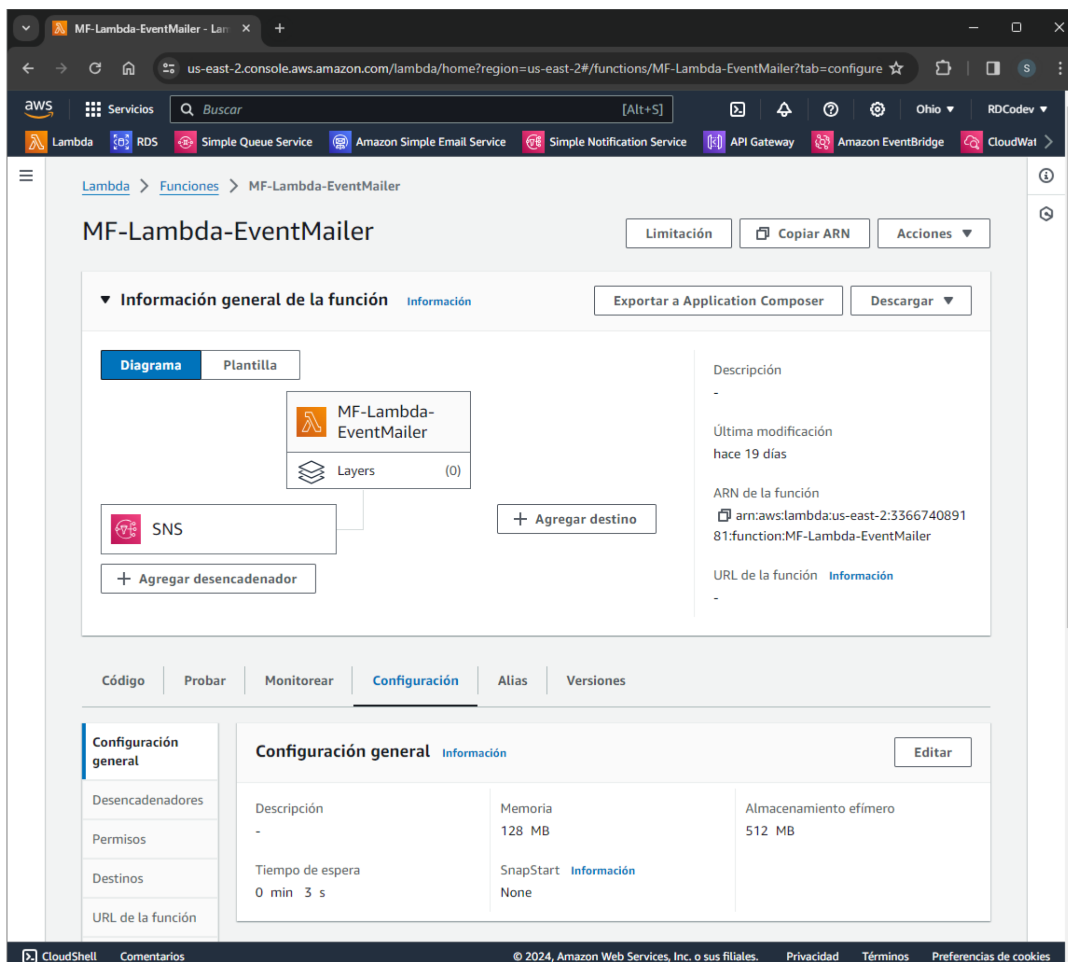
Configuración Función Lambda Event Mailer

Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: Full Access, InvokeFunctions Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocación	Asíncrona
Versión	v0.1.0
Triggers	SNS: AWS_SES_Deliver

Nota. *Tabla de la configuración de parámetros importantes de la función Lambda evento mailer Elaborado por: Los Autores*

Figura 53

Función Lambda Event Mailer



Nota. *Función Lambda evento Mailer implementado Elaborado por: Los Autores*

El código implementado para el registro de los eventos en la instancia de la base de datos se presenta en la figura 54 y los detalles más relevantes de dicha función se presenta en la tabla 54.

Figura 54

Código Función Lambda Event Mailer

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';

export const handler = async (event) => {

  try {
    const { Records } = event;
    const [ data ] = Records;
    const { Sns } = data;
    const { Message } = Sns;

    const {⟺} = JSON.parse(Message);

  const isp = tags["ses:recipient-isp"];
  const from = tags["ses:sender-identity"];
  const date = new Date(timestamp).toISOString().slice(0, 19).replace('T', ' ');

  const queries = {⟺};

  const command = new InvokeCommand(paramsLambdaInvoke(JSON.stringify(queries)));

  const {Payload} = await client.send(command);
  const {result} = parseInvokeResponse(Payload);

    return {
      statusCode: 200,
      body: JSON.stringify(JSON.stringify(result))
    }
  } catch(error) {
```

Nota. *Código para que funcione la función Lambda event Mailer Elaborado por: Los Autores*

Tabla 54

Tabla Código Función Lambda 010

FL010: Código Relevante Función Lambda Event Mailer	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
InvokeCommand ()	Función encargada de realizar la invocación de la función lambda.
paramsLambdaInvoke ()	Función encargada de formar los parámetros para la ejecución de la invocación lambda.
parseInvokeResponse	Función encargada de transforma el resultado de la invocación lambda.
client.send	Método para ejecutar comando en servicios de AWS

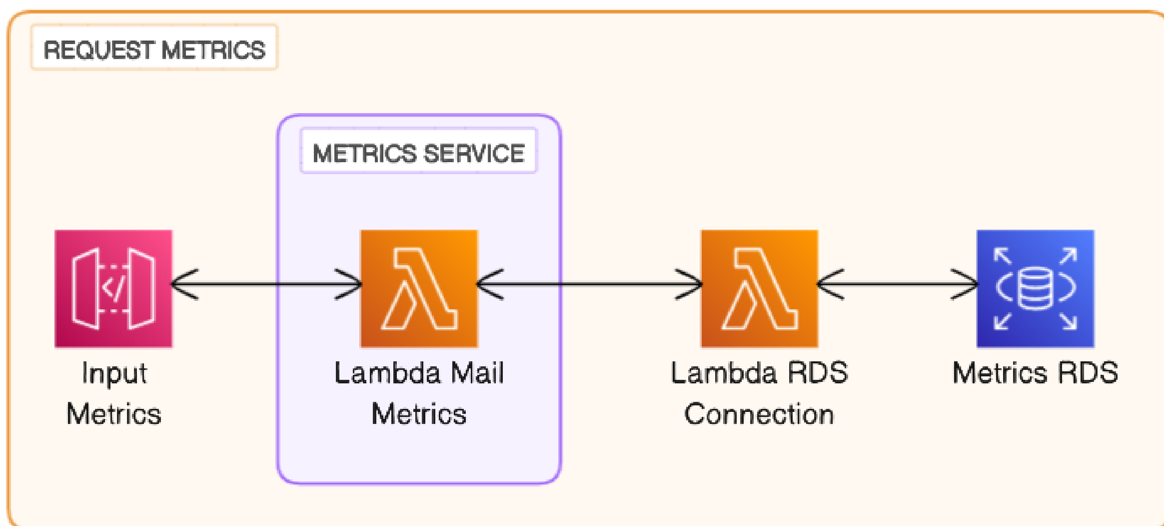
Nota. *Tabla de detalles importantes del código de la función lambda Event Mailer*

1.12.5 Recurso Métricas de Envió de Correos

El recurso métricas de envío de correos o *mail metrics* presentado en la tabla 44 se encarga de recibir las peticiones en la ruta /emails/metrics por medio de la integración de una función lambda con la configuración presentada en la tabla 55, esto con el fin de realizar la recuperación de la información de los eventos registrados por el servicio de registro de eventos de correos definido en la figura 53, el cual consultara los datos en la instancia de la base de datos definida en el servicio RDS y los devolverá como respuesta a la peticiones, justo como se muestra en la figura 55.

Figura 55

Arquitectura Recurso Mail Metrics



Nota. Diagrama del funcionamiento del recurso mail metrics Elaborado por: Los Autores

Tabla 55

Configuración Función Lambda Mail Metrics

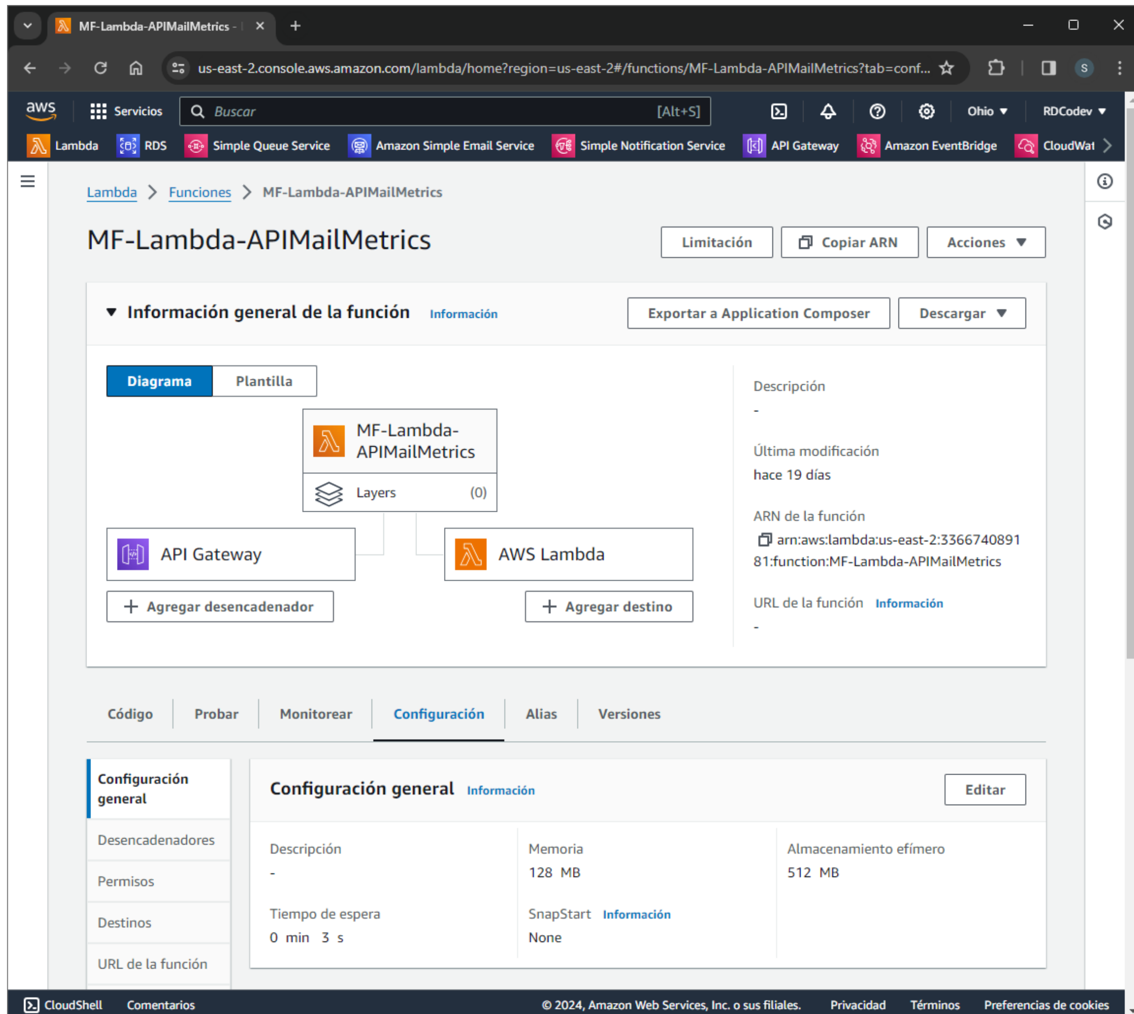
Función Lambda Mail Metrics	
Ephemeral storage	512 MB
Memory	128 MB
Permissions	Lambda: Full Access, InvokeFunctions Logs: PutLogEvents, CreateLogStream, CreateLogGroup.
Invocación	Asíncrona
Versión	v0.1.0
Triggers	API Gateway: MF-REST-App
Destinations	Lambda: RDS Connection

Nota. Tabla de la configuración de los parámetros importantes de la función Lambda mail metrics Elaborado por: Los Autores

Una vez integrada la función lambda en el recurso publicado en el servicio API Gateway junto con su configuración, obtuvimos una función lambda, presentada en la figura 56, la cual se ejecuta en cada petición con el fin de realizar una invocación a la función RDS Connection para obtener el resultado que es enviado como respuesta a dicha petición.

Figura 56

Función Lambda Mail Metrics



Nota. Función Lambda email metrics implementado Elaborado por: Los Autores

El código implementado para el procesamiento de la petición y generación de respuesta se presenta en la figura 57 y los detalles más relevantes de dicha función se presenta en la tabla 56.

Figura 57

Código Función Lambda Email Metrics

```
import { LambdaClient, InvokeCommand } from '@aws-sdk/client-lambda';
import { REGION } from './config.mjs'
import { paramsLambdaInvoke, parseInvokeResponse } from './utils.mjs'

const sql = `CALL sp_S_mail_metrics()`;

const client = new LambdaClient({ region: REGION })

export const handler = async (event) => {

  const queries = { queries: [sql] }

  try {
    const command = new InvokeCommand(paramsLambdaInvoke(JSON.stringify(queries)));
    const { Payload } = await client.send(command)
    const { result } = parseInvokeResponse(Payload)
    const [metrics] = result

    return {
      statusCode: 200,
      body: JSON.stringify(metrics)
    }
  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify(`Error on Server: ${error}`)
    }
  }
};
```

Nota. Código para que funcione la función Lambda email metrics Elaborado por: Los Autores

Tabla 56

Tabla Código Función Lambda 011

FL011: Código Relevante Función Lambda Event Orders	
Función	Descripción
async handler(event)	Se encarga de manejar la información del evento recibido y realizar el procesamiento.
InvokeCommand ()	Función encargada de realizar la invocación de la función lambda.
paramsLambdaInvoke ()	Función encargada de formar los parámetros para la ejecución de la invocación lambda.
ParseInvokeResponse()	Función encargada de transforma el resultado de la invocación lambda.
client.send	Método para ejecutar comando en servicios de AWS

Nota. Tabla de detalles importantes del código de la función lambda Email Metrics

1.13. CONSUMO DE SERVICIOS DEFINIDOS EN AWS CON APP REMIX

El consumo de los servicios definidos en AWS se lo realizo mediante el desarrollo de una aplicación integrada en el área de administrador del e-commerce de Shopify por medio del

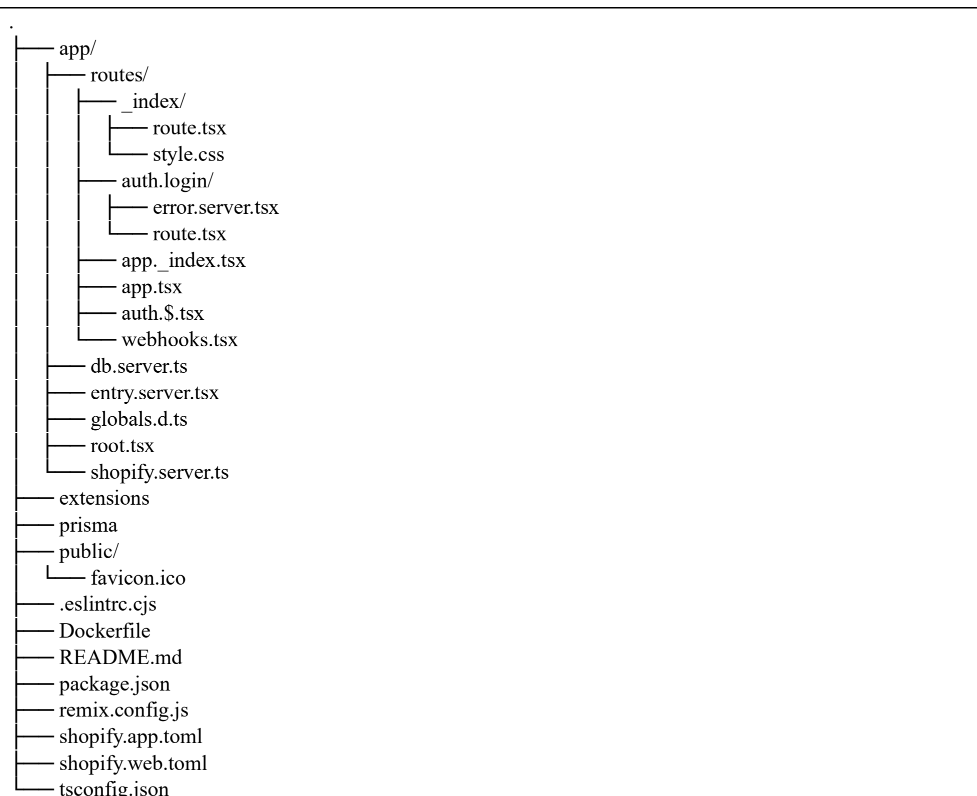
paquete de desarrollo de Shopify para el Aplicaciones en el framework de Remix. Debido a que permite la autenticación e integración en la plataforma de Shopify de manera sencilla con el servicio de AWS Event Bridge para captura los eventos generados en el área de administración del e-commerce.

1.13.1 Creación e Inicialización de Proyecto Shopify/Remix

Para la creación del proyecto se hizo uso del CLI desarrollado por Shopify para la creación de aplicaciones basadas en Remix. Aquí se ejecutó el comando de **npm init @shopify/app@latest@** el cual nos creó la estructura de carpeta presentada en la figura 58 y los directorios y ficheros más importantes se presentan en la tabla 57.

Figura 58

Estructura de Carpetas Proyecto App Remix Shopify



Nota. Estructura que se creó para las carpetas del proyecto App Remix Elaborado por: Los Autores

Tabla 57

Tabla de directorio y ficheros relevantes

Descripción ficheros y directorio App Remix		
shopify.server.ts	/app/	Fichero encargado de la inicialización de la configuración de la aplicación.
routes/	/app/	Directorio en donde se definen las diferentes rutas url accesibles de la aplicación.
app._index.tsx	/app/routes/	Fichero de la ruta inicial y por defecto consultado y/o mostrado al ingresar a la aplicación
Route.tsx	/app/route/auth.login	Fichero encargado de realizar la autenticación entre la aplicación y la plataforma de Shopify.
package.json	/	Fichero encargado de definir las dependencias de librearías del proyecto.
Shopify.app.toml	/	Fichero de configuración de permisos de la aplicación en la plataforma de Shopify, y urls asociadas a la aplicación para la creación de túneles en el servicio de Cloudflare para despliegue del proyecto local en la plataforma de Shopify.

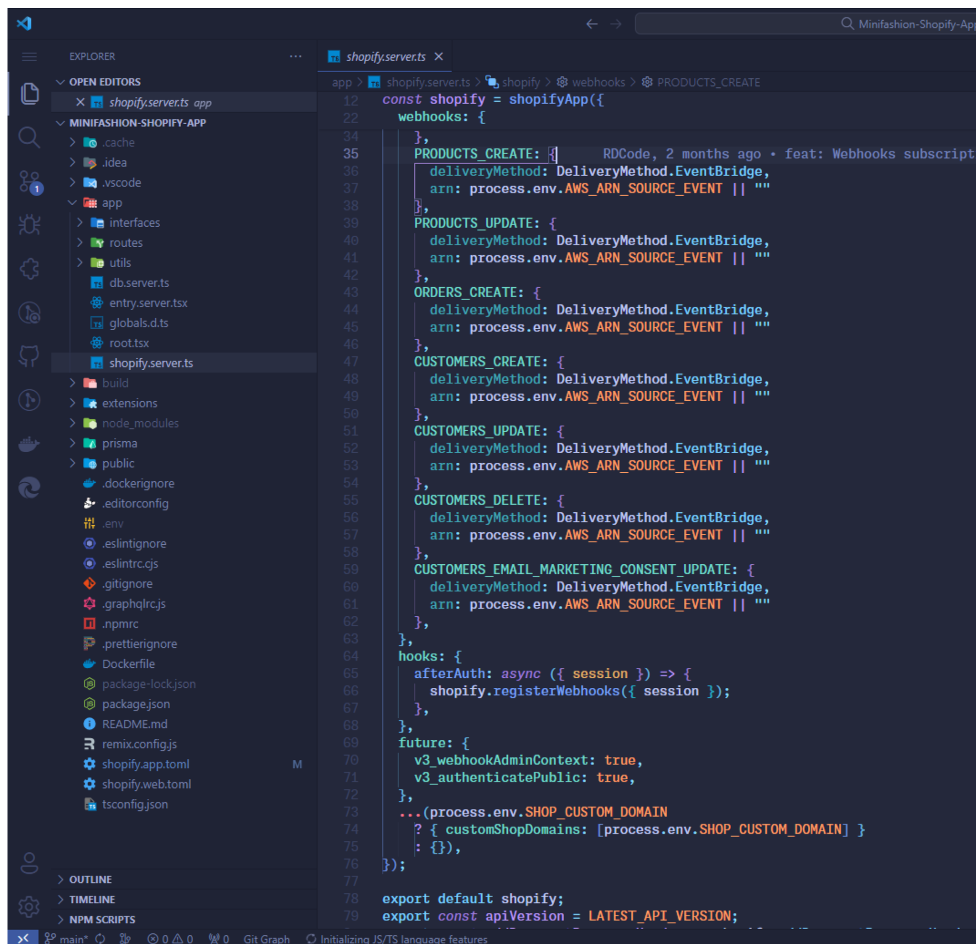
Nota. Tabla de directorios y ficheros importantes Elaborado por: Los Autores

*1.13.2 **Configuración y Comunicación de Eventos Shopify (Webhooks)***

La configuración en la aplicación integrada en Shopify para el envío de los webhooks generados en la plataforma se lo definió en el fichero **/app/shopify.server.ts** en donde se agregaron más objetos javascript en la configuración de la aplicación los cuales poseen la estructura presentada en la tabla 58, justo como se muestra en la figura 59.

Figura 59

Objeto para suscripción a webhooks



Nota. Objeto para la comunicación entre webhooks Elaborado por: Los Autores

Tabla 58

Características de objeto para suscripción de webhooks en Shopify App

Objeto para Configuración de Webhooks	
Nombre	Nombre del webhook a configurar, basado en la documentación oficial de desarrollo de Shopify
deliveryMethod	Tipo de método de entrega del evento generado, ya sea http, eventbridge o pubsub
arn	Dirección de destino en la plataforma para entrega de evento generado en la aplicación.

Nota. Características de los objetos de webhooks Elaborado por: Los Autores

1.13.3 Desarrollo de Recomendación de Productos

Para obtener las recomendaciones de productos de los clientes en la plataforma de Shopify, de manera inicial se hizo uso del código presentado en la figura 60, en donde los detalles más importantes del código se explican en la tabla 59. Esto con el fin de obtener la lista de clientes por medio de una petición al recurso Customers definido en el servicio AWS API Gateway.

Figura 60

Código para petición al recurso Customers

```
export const loader = async ({ request }: LoaderFunctionArgs) => {
  const { admin } = await authenticate.admin(request);

  const queryFilter = (query: any) => query.context === QueriesContexts.PRODUCTS
  const respQLResponse = await admin.graphql(grahpqlQueries.filter(queryFilter)[0].query(100))

  const { data: { products: { edges: savedProducts } } } = await respQLResponse.json()

  const api = AWS_ENDPOINTS.customersList(SHOPIFY_APP_ID)
  const response = await fetch(api)
  const { customers }: CustomerList = await response.json()

  return json({ customers: customers || [], savedProducts })
}
```

Nota. *Código que pide el recursos de los Customers Elaborado por: Los Autores*

Tabla 59

Tabla Código para petición al recurso Customers

Código para petición al recuso Customers	
Función	Descripción
authenticate.admin()	Función que obtiene la autenticación de la aplicación en la plataforma para consumir los recursos de Admin API o GrapQL API de Shopify.
fetch()	Función encarga de realizar peticiones a un recurso por defecto GET, en este caso se hace una petición al recurso Customers definido en la plataforma serverless.
Json()	Función para transformar el resultado de las peticiones de formato json a un objeto JavaScript.

Nota. *Tabla de detalles relevantes del código para petición al recurso Customers*

Posteriormente se realiza las recomendaciones de productos en base al historial de los clientes obtenidos y seleccionado/s, mediante el código presentado en la figura 61 y los detalles más relevantes de dicho código se presenta en la tabla 60.

Figura 61

Código para generación de recomendaciones de Productos

```
const updateSelection = useCallback(
  (selected: string[]) => {
    const selectedValue = selected.map((selectedItem) => {
      const matchedOption = options.find((option: any) => {
        return option.value.match(selectedItem);
      });
      return matchedOption && matchedOption.label;
    });

    const customerProps = updateCustomer(selectedValue[0] || '', customers as Customer[])
    const { common_products, favorite_vendors } = customerProps

    setSelectedOptions(selected);
    setCustomer(customerProps)
    wrapperState(retrieveCommonObjectByFields(products, 10, [...common_products, ...favorite_vendors]))
    setInputValue(selectedValue[0] || '');
  },
  [options, customers, products, wrapperState, updateCustomer],
);
```

Nota. *Código para la generación de recomendaciones Elaborado por: Los Autores*

Tabla 60
Tabla Código para de recomendaciones de productos

Código Generación de recomendaciones de Productos	
Función	Descripción
updateCustomer()	Función la cual obtiene las propiedades de los clientes seleccionado/s.
RetrieveCommonObjectByFields()	Función la cual genera y obtiene las recomendaciones de los productos, basado en los productos disponibles en el e-commers y en los productos y distribuidores más comunes del cliente seleccionado/s.

Nota. *Tabla de detalles relevantes del código de generación de recomendación de productos*

Finalmente, para la creación y registro del código de descuento para los productos recomendados en la plataforma de Shopify, se hizo uso del código presentado en la figura 62, en donde los detalles más importantes del código se explican en la tabla 61.

Figura 62

Código para creación y registro de Código de descuento

```
export async function action({ request }: ActionFunctionArgs) {
  const { admin } = await authenticate.admin(request)

  const formData = await request.formData();

  const response = await admin.graphql(⋯
  )

  const responseJson = await response.json();
  const {code} = flat(responseJson, {})

  return {code}
}
```

Nota. *Código para creación y registro de un código de descuento Elaborado por: Los Autores*

Tabla 61
Tabla Código Generación de Código de Descuento

Código para generación de código de descuento a los productos recomendados	
Función	Descripción
authenticate.admin()	Función que obtiene la autenticación de la aplicación en la plataforma para consumir los recursos de Admin API o GrapQL API de Shopify.
request.formData()	Función la cual obtiene los datos enviados por la función de submit del framework remix, en este caso obtiene los datos de los productos y clientes para genera el codigo de descuento.
admin.graphql()	Función la cual ejecuta una query o mutation ingresada por parámetros en el servicio de GraphQL de Shopify, en este caso ejecuta un mutation para registrar el codigo de descuento generado para los productos al cliente asociado o seleccionado/s.

Nota. *Tabla de detalles relevantes del código de generación de recomendación de productos*

1.13.4 Desarrollo de Envió de Recomendaciones por Correo Electrónico

Él envió de recomendaciones generadas en la plataforma de Shopify a los correos de los clientes de destino, se lo realizo por medio del código presentado en la figura 63, en donde los detalles más relevantes del código se presentan en la tabla 62.

Figura 63

Código para envió de recomendaciones a correos electrónicos

```
const handleSubmit = useCallback(async (code: any) => {

  if(!code) return

  const templateDeliver = parseTemplateEmailData(emails, collection, discount, body, offerProducts, template, code)

  setTemplate(templateDeliver)
  setDeliver(true)

  try {
    await fetch(AWS_ENDPOINTS.deliverEmails(), {
      method: 'POST',
      body: JSON.stringify(templateDeliver),
      mode: 'no-cors'
    })

    onSend(true)
  } catch (error) {

  } finally {
    setDeliver(false)
    wrapperState(false)
  }

}, [emails, collection, discount, body, offerProducts, template, onSend, wrapperState])
```

Nota. Código para envío de recomendaciones Elaborado por: Los Autores

Tabla 62
Tabla Código de envió de Recomendaciones

Código para envió de recomendaciones a correos electrónicos.	
Función	Descripción
parseTemplateEmailDate()	Función encargada de obtener el objeto con la información que será enviada en el cuerpo de la petición realizada al servicio AWS para él envió de correos de recomendaciones
fetch()	Función para realizar la petición a un recurso, en este caso al recurso deliver email mediante el método POST.

Nota. Tabla de detalles relevantes del código para envió de recomendaciones

1.13.5 Desarrollo de Visualización de Métricas de Envió

Para la visualización de las métricas de los eventos de correos electrónicos, registrados en la plataforma serverless desarrollada, se hizo uso del código presentado en la figura 64 y los detalles más relevantes del código se explican en la tabla 63.

Figura 64

Código para realizar petición al recurso Mail Metrics

```
export const loader = async ({ request }: LoaderFunctionArgs) => {
  await authenticate.admin(request);
  const api = AWS_ENDPOINTS.emailsMetrics();
  const response = await fetch(api)
  const { emails_metrics } = await response.json();

  return json({ emails_metrics })
};
```

Nota. Código para pedir al recurso de mail metrics Elaborado por: Los Autores

Tabla 63
Tabla Código para Petición al Recurso Mail Metrics

Código para peticiones al recurso Mail Metrics	
Función	Descripción
fetch()	Función para realizar peticiones a un recurso, en este caso al recurso mail metrics mediante el método por defecto GET, este caso se hace una petición al recurso Mail Metrics definido en la plataforma serverless.
json()	Función para transformar el resultado de las peticiones de formato json a un objeto JavaScript.

Nota. Tabla de detalles relevantes para realizar petición al recurso Mail Metrics

CÁPITULO III

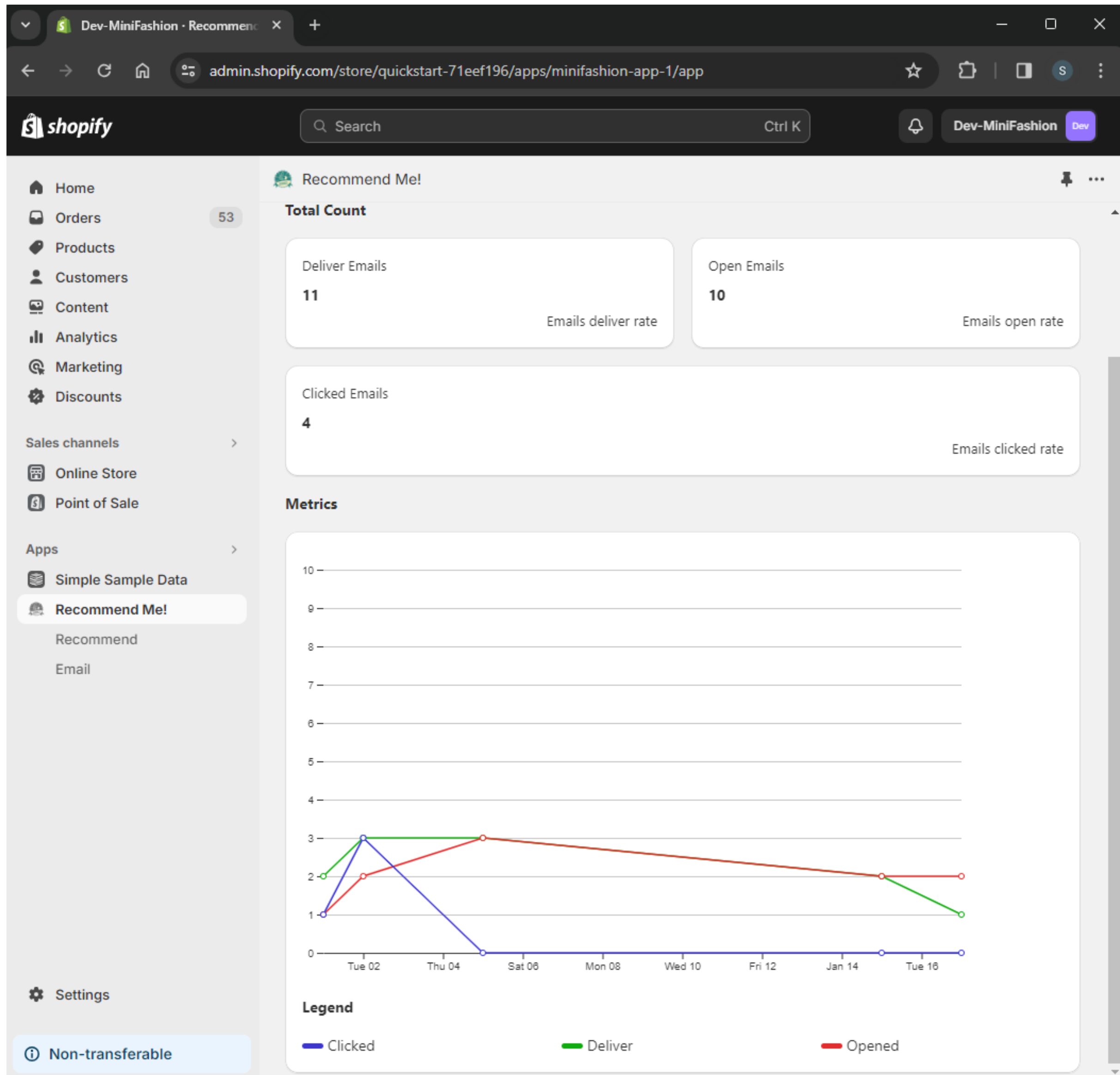
RESULTADOS

1.14. PANTALLA DE MÉTRICA DE ENVIÓ DE CORREO ELECTRÓNICOS

En la figura 65 se aprecia la sección en la que se administrador del e-commerce puede visualizar los diferentes eventos surgidos del envió de recomendaciones por correo electrónico, tanto de manera individual como de manera general en una gráfica que representa el tipo de evento en el tiempo basado en días.

Figura 65

Pantalla Métricas de envió de correo electrónicos de Aplicación Integrada en Shopify



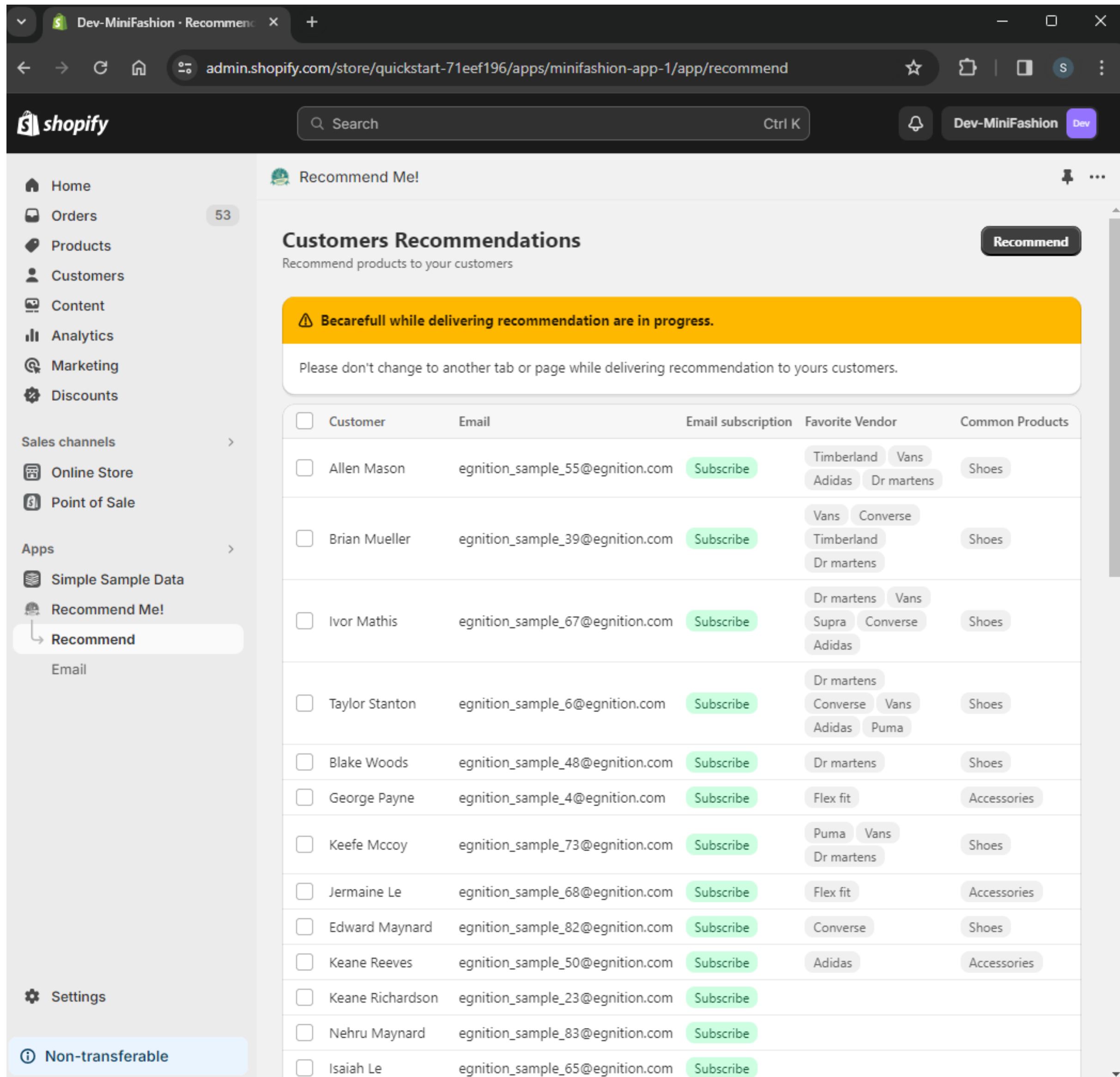
Nota. Pantalla de las métricas de envío de correos Elaborado por: Los Autores

1.15. PANTALLA DE RECOMENDACIONES AUTOMÁTICAS PARA VARIOS CLIENTES

En la figura 66, se aprecia la sección de la lista de clientes que se encuentran registrados en el sistema por medio de los eventos generados por el administrador del e-commerce en relación con las ordenes, clientes y productos.

Figura 66

Pantalla de lista seleccionable de contactos para recomendaciones de productos

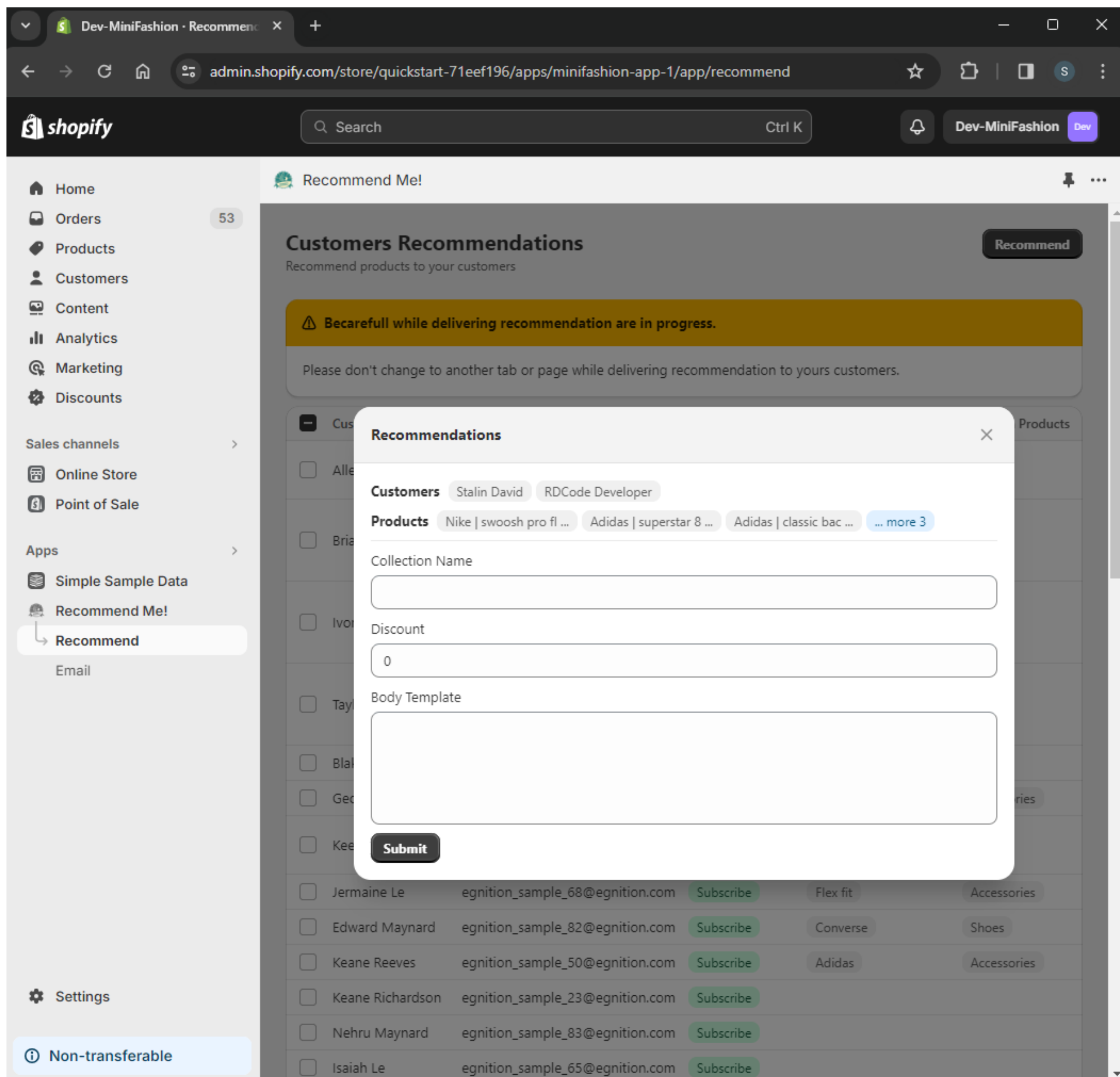


Nota. Pantalla de lista seleccionable de contactos para la recomendación de productos Elaborado por: Los Autores

En la figura 67, se presenta la sección en donde el administrador puede observar los clientes y los productos recomendados a ser enviados y definir la información como nombre de la recomendación, descuento, aplicado y el mensaje de la recomendación que se incluirá en el correo electrónico de recomendación.

Figura 67

Pantalla de formulario para ingreso de información de recomendación



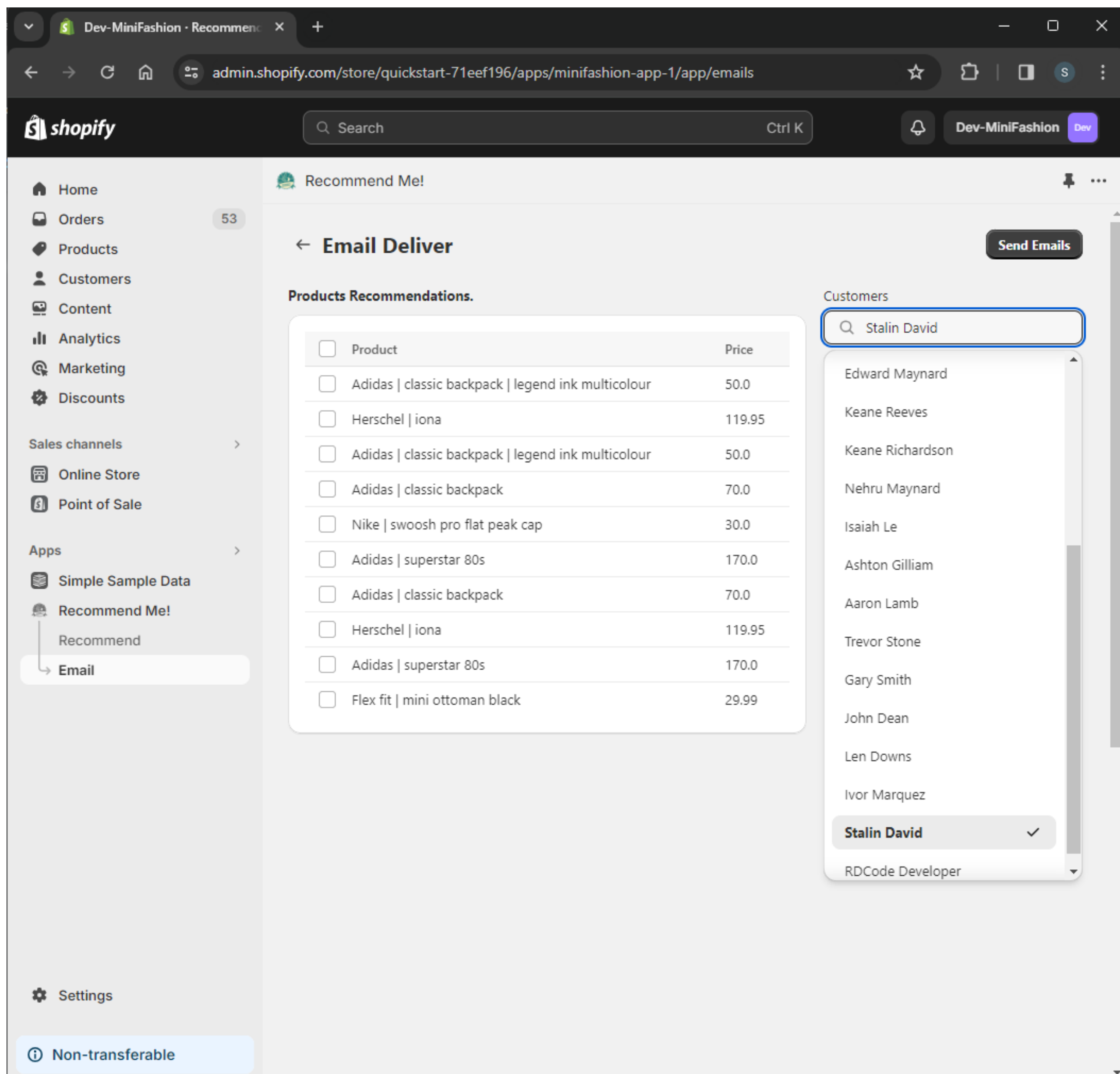
Nota. *Pantalla de formulario para la información de la recomendación Elaborado por: Los Autores*

1.16. PANTALLA DE RECOMENDACIONES PARA UN CLIENTE

En la figura 68, se presenta la sección de recomendación para un cliente, en donde se puede visualizar y seleccionar las recomendaciones generadas por un cliente seleccionado, así como la información básica de este mismo cliente.

Figura 68

Pantalla de recomendación de productos seleccionable para un único cliente.

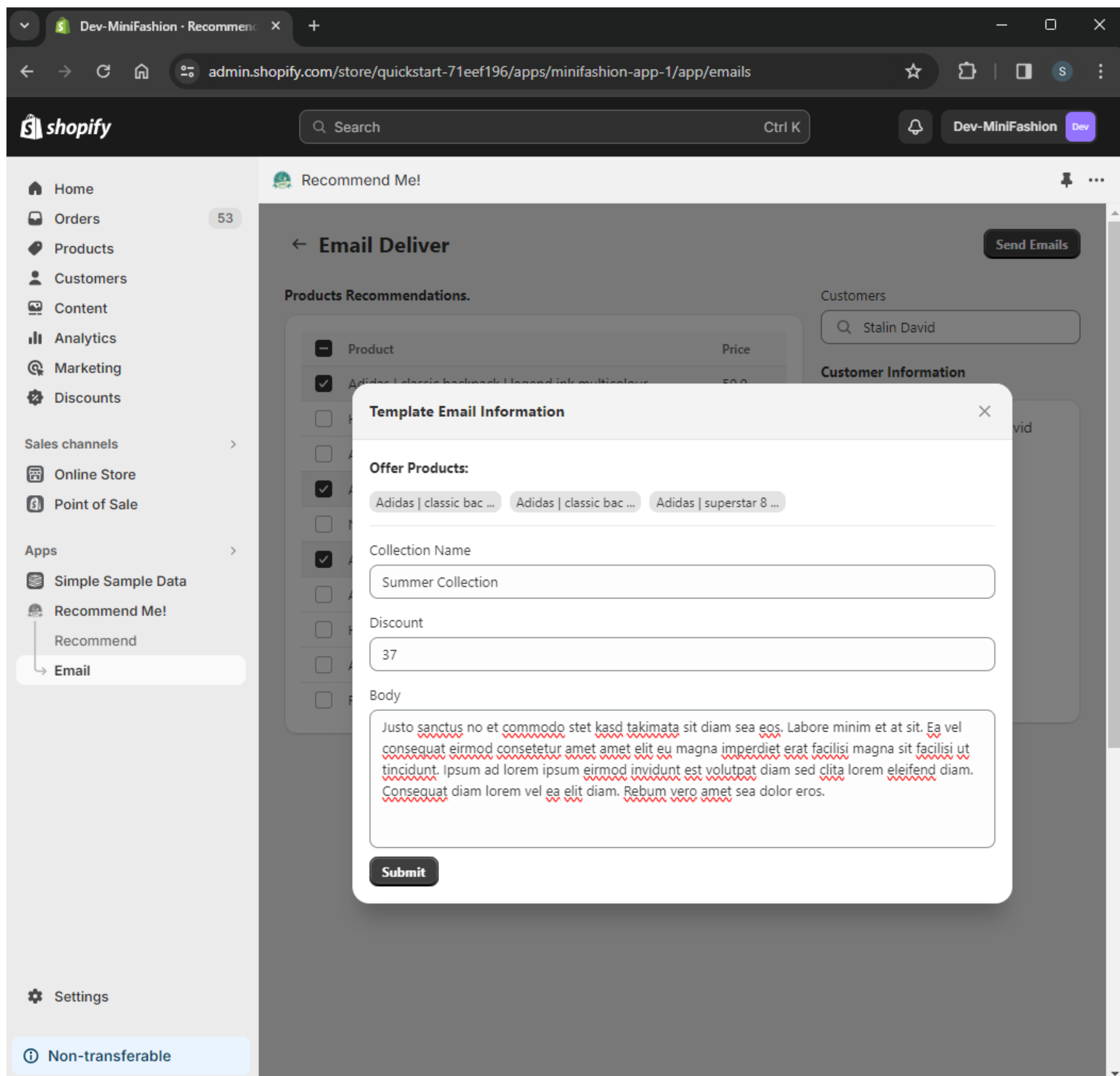


Nota. Pantalla de recomendación de productos seleccionable para un solo cliente Elaborado por: Los Autores

En la figura 69, se presenta la sección en la que el administrador puede ver los productos del cliente seleccionado al cual se le enviará la recomendación, además de permitir definir la información que será enviada en el correo de recomendación como nombre de la recomendación, descuento aplicado y el mensaje de la recomendación.

Figura 69

Pantalla para ingreso de información de las recomendaciones



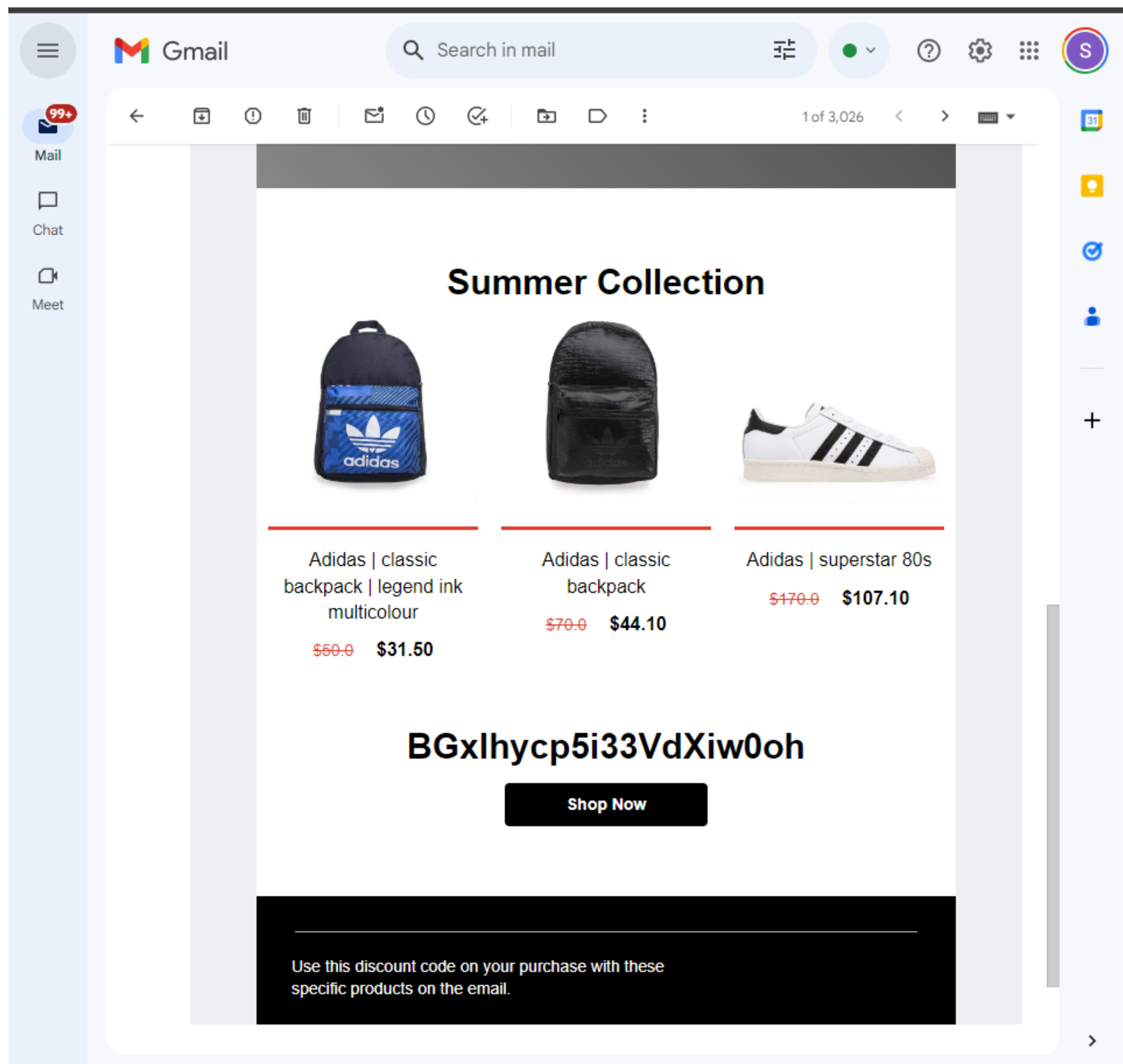
Nota. *Pantalla de información de recomendaciones Elaborado por: Los Autores*

1.17. ENVIÓ DE RECOMENDACIONES DE PRODUCTOS CON CÓDIGO DE DESCUENTO

En la figura 70, se presenta el correo recibido por el cliente destino, seleccionado por el administrador del e-commerce, en el cual se incluye la información solicitada en los formularios para él envió del correo de recomendación.

Figura 70

Correo electrónico de recomendación de producto con código de descuento



Nota. *Correo electrónico de recomendación de productos Elaborado por: Los Autores*

CONCLUSIONES

- La implementación exitosa de la arquitectura serverless en un entorno de e-commerce Shopify, respalda de manera contundente que este enfoque representa una respuesta efectiva para optimizar diversos procesos presentes en la plataforma. Gracias a su diseño distribuido no monolítico, fomenta un trabajo colaborativo entre servicios conectados distribuyendo las responsabilidades de estos mediante una interconexión activa que funciona bajo eventos.
- La creación de las funcionalidades dio apertura a que se pueda utilizar servicios para crear funcionalidades específicas, pero de la misma forma entre servicios se pueden intercomunicar estas funcionalidades y ligarse una con otra.
- El desarrollo del proyecto técnico mediante servicios de computación basados en la nube para el desarrollo de una arquitectura serverless permitió redirigir los esfuerzos de implementación de características como monitoreo, administración, escalabilidad, etc, en la plataforma al desarrollo de las funcionales o lógica de negocio y características de la aplicación implementada.
- El resultado final y el análisis de costos, permitió determinar que existen costos de operación y mantenimiento significativamente bajo, debido a que la capa gratuita de Amazon Web Services cubre las necesidades de desarrollo de proyecto básicos, permitiendo incursionar en el mercado con prototipos con funcionales complejas a muy bajo costo de operatividad.

RECOMENDACIONES

- Se recomienda a los desarrolladores, ingenieros de software y tomadores de decisiones en el ámbito empresarial que estén considerando implementar una arquitectura serverless, realizar una evaluación precisa y clara de los servicios en la nube que planean utilizar en sus proyectos. Esto es crucial para dirigirse a los servicios que realmente son necesarios, evitando la complejidad innecesaria y optimizando la eficiencia de sus proyectos.
- Debido a la curva de aprendizaje que presenta la adquisición e implementación de servicios en la nube es menester realizar capacitaciones al equipo de desarrollo con el fin de que los mismo tengan un mejor desenvolvimiento en el desarrollo de nuevas funcionales.
- El desarrollo e implementación de la lógica de negocio en una arquitectura serverless si bien puede ser manejado con el paradigma orientado a objetos o POO, es recomendable utilizar un paradigma de programación funcional debido a que es el que mejor se adapta al flujo de comunicación de servicio basado en eventos.

REFERENCIAS

B. Hassan, H., A. Barakat, S., & I. Sarhan, Q. (Julio de 2021). Survey on serverless computing. *Journal of Cloud Computing*. doi:https://doi.org/10.1186/s13677-021-00253-7

Barreto, J. X., Villacreses, C. A., Chóez, J. E., & Figueroa, V. A. (2021). El impacto de la plataforma Shopify en el Comercio electrónico. *Serie Científca de la Universidad de las Ciencias Informáticas, 14*, 85-98.

Blake, M. B., & Wei, Y. (2010). Service-Oriented Computing and Cloud Computing. *IEEE Computer Society*. doi:1089-7801/10/$26.00

Bocanegra, J. (18 de Octubre de 2023). Paradigma Serverless:función como servicio44SISTEMAS . *ACIS*. doi:10.29236/sistemas.n168a7

Boss, G., Malladi, P., Quan, D., & Legregni, L. (8 de Octubre de 2001). Cloud Computing. *IBM Corporation*. Obtenido de www.ibm.com/developerworks/websphere/zones/hipods/

Camargo, J. E., Rozo, N., Ponzo, J., & Gonzaléz, F. (2023). Paradigma Serverless: ¿evolución de la computación en la nube? *Revista Sistemas*. (S. Gallardo, Entrevistador) doi:10.29236/sistemas.n168a5

Casale, G., & Gias, A. (2021). Cold Start Aware Capacity Planning for Function-as-a-Service Platforms. *COCOA, 17*, Mayo. doi:1109/mascots50786.2020.9285966

CNCF. (2022). CNCF Serverless Paper. *CNCF*. Obtenido de https: //www.cncf.io/

Dominguez, S. Y., & Gonzalez Urra, P. (2006). Sistemas de gestión de contenidos: En busca de una plataforma ideal. *ACIMED*.

HOSSEIN, S., AHMAD, K., & PAYAM, M. (November de 2022). Serverless Computing: A Survey of Opportunities, and Applications. *ACM Computing Surveys,*. doi:https://doi.org/10.1145/3510611

Jinfeng, W., Zhenpeng, C., Xin, J., & Xuanzhe, L. (2023). Rise of the Planet of Serverless Computing: A Systematic Review. *ACM Trans. Softw. Eng. Methodol, 32*, 61. doi:https://doi.org/10.1145/3579643

Khol, W. (2014). BackendAsAService UsingtheExampleofEnginioaCloudService forQt. *Institute of Distributed Systems*.

Lucidchart. (s.f.). Recuperado el 18 de 01 de 2024, de https://www.lucidchart.com/pages/es/tutorial-diagrama-de-actividades-uml

Manoj, K. (25 de Enero de 2019). Serverless Architectures Review, Future Trend and the Solutions to Open Problems. *American Journal of Software Engineering,, 6*. doi:10.12691/ajse-6-1-1

Mell, P., & Grance, T. (2011). Recommendations of the National Institute of Standards and Technology. *National Institute of Standars and Technology*.

Nevárez Montes, J. (2014). *E-commerce.* La Loma Tlalnepantia: Digital UNID.

Ortega Fernández, C. (2017). CÓMO LAS EMPRESAS PUEDEN IMPULSAR SU NEGOCIO A TRAVÉS DE LAS PLATAFORMAS E-COMMERCE CON EL BIG DATA, EL APRENDIZAJE AUTOMÁTICO Y EL MANAGEMENT CIENTÍFICO. *Quality Excellence S.L.*, 75-86.

Ramos, J. (2012). *E-Commerce 2.0.: Cómo montar su propio negocio de comercio.*

Rodríguez, N., Murazzo, M., Medel, D., Parra, L., Laura, A., Sánchez, F., . . . Atencio, H. (2020). Computación Serverless para tratamiento de datos provenientes de dispositivos de IoT . *F.C.F.M*, 753-757.

Shillaker, S., & Pietzuch, P. (2020). Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing. *Proc. USENIX Annu. Tech.*, 4-19-433.

V, S. (2020). Cloudburst: Stateful functions-as-a-service. *VLDB Endowment, 13*, 2438–2452. doi:https://doi.org/10.14778/3407790.3407836

Yudiyanto, J. P. (Agosto de 2023). Digital Marketing Strategy to Increase Sales Conversion on E-commerce Platforms. *Journal of Contemporary Administration and Management (ADMAN), 1*, 54-62. doi:https://doi.org/10.61100/adman.v1i2.23

Zanon, D. (2017). Building Serverless Web Applications. *Packt Publishing Ltd.*

Zijun, L., Linsong , G., Jiagan , C., & Quan , C. (Septiembre de 2022). The Serverless Computing Survey: A Technical Primer for. *ACM Computing Surveys,, 54*.