UNIVERSIDAD DE CUENCA

Facultad de Ingeniería

Maestría en Gestión Estratégica de Tecnologías de la Información

DISEÑO E IMPLEMENTACIÓN DE MICROSERVICIOS PARA GENERAR UN MODELO DE ANÁLISIS DE TRANSACCIONES MASIVAS DE TARJETAS DE DÉBITO PARA UNA INSTITUCIÓN FINANCIERA.

Trabajo de titulación previo a la obtención del título de Magíster en Gestión Estratégica de Tecnologías de la Información.

Autora:

Tatiana Ximena Muñoz Sánchez

CI:0104586292

tatiana.munozs@hotmail.com

Director:

Jorge Mauricio Espinoza Mejía

CI:0102778818

Cuenca-Ecuador

12-abril-2022

Resumen: El presente trabajo presenta una propuesta para la generación de microservicios que permitan capturar la información de transacciones masivas de tarjetas de débito de una institución financiera. La meta es generar desde una base de datos relacional un proceso de extracción, transformación, y carga (ETL) y llevarla hacia una base de datos no relacional que permita a través de mecanismos de inteligencia de negocios convertir los datos almacenados en información oportuna y valiosa para la toma de decisiones. Todo el proceso de diseño e implementación de la arquitectura propuesta involucró tres fases: la primera utiliza un mecanismo dentro de la base de datos relacional para la detección de los cambios producidos (CDC) dentro del repositorio relacional; la segunda, el diseño e implementación de un consumidor encargado de escuchar los mensajes y generar el proceso respectivo de transformación y carga a la base de datos no relacional asegurando tolerancia a fallos, fiabilidad, y escalabilidad. Por último, a través de indicadores definidos se diseñó un tablero de control que ofrezca a los directivos lineamientos necesarios para tomar decisiones, medir el logro de objetivos, y ofrecer mediante una herramienta visual la transformación de los datos en conocimiento para generar una ventaja competitiva en procesos gerenciales y operativos de la institución.

La integración de los componentes planteados dio como resultado una arquitectura que gestiona de manera eficiente miles de registros en tiempo real permitiendo a su vez mayor velocidad, rendimiento, y optimización en búsquedas de grandes volúmenes de datos.

Palabras claves: Consumidor. CDC. Escalabilidad. ETL. Indicadores. Inteligencia de negocios.

Abstract: This work presents a proposal for the generation of microservices to capture the information of massive transactions of debit cards of a financial institution from a relational database, generate the respective process of extraction, transformation and load (ETL) and take it towards a database of non-relational data that allows through business intelligence mechanisms to convert stored data into timely and valuable information for decision-making. For the methodology, three phases were used: the first uses a mechanism within the relational database to detect the changes produced (CDC); the second, the design and implementation of a consumer in charge of listening to the messages and generating the respective process of transformation and loading to the non-relational database ensuring fault tolerance, reliability and scalability. Finally, through defined indicators, a control panel is designed that offers managers the necessary guidelines to generate decisions, measure the achievement of objectives and goals, thus offering, thanks to this visual tool, converting data into knowledge that generates competitive advantage in management and operational processes of the institution.

The integration shows as a result the design of an architecture that efficiently manages thousands of records in real time, allowing at the same time greater speed, performance and optimization in searches of large volumes of data.

Keywords: Consumer. CDC. Scalability. ETL. Indicators. Business intelligence.

Índice del Trabajo

Contenido

Índice de Figuras 7
Índice de Tablas 9
Índice de Códigos 10
Agradecimientos 13
Dedicatoria 14
CAPITULO 1 - INTRODUCCIÓN 15
1.2 ANTECEDENTES 16
1.3 JUSTIFICACIÓN 17
1.4 PROBLEMÁTICA 17
1.5 OBJETIVOS 18
1.6 METODOLOGÍA 18
1.7 ALCANCE 20
1.8 ESTRUCTURA DEL DOCUMENTO 21
CAPITULO 2 CONTEXTO TECNOLÓGICO 22
2.1 INTRODUCCIÓN 22
2.2 ARQUITECTURA LAMBDA 22
2.3 ARQUITECTURA KAPPA 24
2.3.1 FUENTES DE DATOS 26
2.3.2 ALMACENAMIENTO 28
2.3.2.1 SISTEMA DE FICHEROS DISTRIBUIDO DE HADOOP (HDFS) 28
2.3.2.2 BASES DE DATOS RELACIONALES 29
2.3.2.3 BASE DE DATOS NO RELACIONALES (NoSQL) 29
2.3.3 PROCESAMIENTO EN TIEMPO REAL (INGESTIÓN) 30
2.3.3.1 CAPTURA DE CAMBIOS EN LOS DATOS (CDC) 30
2.3.3.2 BROKER DE MENSAJERÍA 32
2.3.4 ALMACENAMIENTO DE DATOS ANALÍTICOS 35
2.3.5 ANÁLISIS 35
2.3.6 VISUALIZACIÓN Y REPORTES 36
2.3.6.1 INDICADORES CLAVES DE RENDIMIENTO (KPI) 36

2.3.6.2 TABLERO DE CONTROL 36
CAPITULO 3 TRABAJOS RELACIONADOS 39
CAPITULO 4 ARQUITECTURA PROPUESTA 43
4.1 COMPONENTE 1 – DETECTOR DE CAMBIOS 45
4.1.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA CAPTURA DE CAMBIOS DE DATOS 45
4.1.2 HERRAMIENTA SELECCIONADA 47
4.2 COMPONENTE 2 – NÚCLEO 49
4.2.1 BROKERS DE MENSAJERÍA 49
4.2.1.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA HERRAMIENTAS BROKER DE MENSAJERÍA 50
4.2.1.2 HERRAMIENTA SELECCIONADA 53
4.2.2 MICROSERVICIO 57
4.2.2.1 NORMAS ISO 59
4.2.2.2 CONECTOR DESTINO (SINK CONNECTOR) 62
4.2.3 ALMACENAMIENTO NO SQL Y MOTORES DE BÚSQUEDA 63
4.2.3.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA HERRAMIENTAS DE ALMACENAMIENTO NO SQL 63
4.2.3.2 HERRAMIENTA SELECCIONADA 66
4.3 COMPONENTE 3 – TABLERO DE CONTROL 67
4.3.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA TABLEROS DE CONTROL (DASHBOARD) 67
4.3.2 HERRAMIENTA SELECCIONADA: KIBANA 70
CAPITULO 5 IMPLEMENTACIÓN DEL DISEÑO PROPUESTO 75
5.1 PREPARACIÓN DEL ENTORNO 77
5.2 COMPONENTES DE LA ARQUITECTURA 77
5.2.1 DETECTOR DE CAMBIOS 78
5.2.2 NÚCLEO 80
5.2.2.1 CONFIGURACIÓN DE APACHE ZOOKEPER 81
5.2.2.2 CONFIGURACIÓN DE DEBEZIUM 83
5.2.2.3 CONFIGURACIÓN DE SERVER APACHE KAFKA 83
5.2.2.4 GESTIÓN DE LOS TEMAS (TOPICS) 84
5.2.2.5 CONFIGURACIÓN AUTÓNOMA DE KAFKA 86
5.2.2.6 CONSUMIDORES 88
5.2.2.6.1 CONSUMIDOR EN CONSOLA 89

5.2.2.6.2 CONSUMIDOR JAVA (MICROSERVICIO) 89
5.2.2.7 CARGA 90
5.2.3 TABLERO DE CONTROL 91
5.2.3.1 INDICADORES CLAVES DE RENDIMIENTO (KPI) 91
5.2.3.2 KIBANA 92
5.3 ANÁLISIS DE RESULTADOS 93
CAPÍTULO 6 CONCLUSIONES Y TRABAJOS FUTUROS 103
6.1 CONCLUSIONES 103
6.1.1 OBJETIVO GENERAL 103
6.1.2 OBJETIVO ESPECÍFICO 1 104
6.1.3 OBJETIVO ESPECÍFICO 2 105
6.1.4 OBJETIVO ESPECÍFICO 3 106
6.1.5 OBJETIVO ESPECÍFICO 4 107
6.1.6 OBJETIVO ESPECÍFICO 5 107
6.1.7 OBJETIVO ESPECÍFICO 6 108
6.2 TRABAJOS FUTUROS 109
6.3 DIFUSIÓN DE RESULTADOS 109
6.3.1 ACEPTACIÓN ARTÍCULO CIENTÍFICO 109
ANEXO 1: CONFIGURACIÓN CDC EN SQL SERVER 2019 111
ANEXO 2: DEBEZIUM CONECTOR SQL SERVER 114
ANEXO 3: CONFIGURACIÓN DE APACHE KAFKA 117
ANEXO 4: MICROSERVICIO PARA TARJETAS DE DÉBITO 120
ANEXO 5: ALMACENAMIENTO EN ELASTICSEARCH 130
ANEXO 6: TABLERO DE CONTROL DE TARJETAS DE DÉBITO 132
BIBLIOGRAFÍA 140

Índice de Figuras

Figura 1 Arquitectura Lambda. 23
Figura 2 Arquitectura Kappa 24
Figura 3 Arquitectura para procesamiento en tiempo real 26
Figura 4 Proceso ETL 33
Figura 5 Representación Drill Down 37
Figura 6 Representación mapa de calor 38
Figura 8 Arquitectura Debezium 48
Figura 9 Funcionamiento General Apache Kafka 55
Figura 10 Grupo de consumidores 56
Figura 11 Replicación de eventos en Apache Kafka 57
Figura 12 Funcionamiento del microservicio 58
Figura 13 ISO8583 60
Figura 14 Funcionamiento Kafka con conector destino 63
Figura 15 Cuadrante Mágico de Gartner para Plataformas Analíticas y de Business Intelligence 2021 68
Figura 17 Dashboard Kibana 72
Figura 18 Arquitectura Final 73
Figura 19 Entradas y Salidas de Formatos y Estructuras de Datos en Arquitectura Propuesta 74
Figura 20 Arquitectura Propuesta 76
Figura 21 Base de datos activada con CDC 80
Figura 22 Tabla para capturar las transacciones de tarjetas de débito con opción CDC activada 80
Figura 23 Inicio Zookeper 82
Figura 24 server.properties 84
Figura 25 worker.properties 87
Figura 26 connector.properties 87
Figura 27 Inicio Elasticsearch 91
Figura 28 Kibana 93
Figura 29 Top 5 de países con mayor consumo de compras con tarjeta de débito 94
Figura 30 Monto total en compras con tarjeta de débito 95
Figura 31 Número de transacciones realizadas con tarjeta de débito 96
Figura 32 Top 5 de comercios con mayor número de compras con tarjeta de débito 97
Figura 33 Porcentajes a nivel de comercios de acuerdo al número de transacciones 97
Figura 34 Mapa de calor de tarjetas de débito 102
Figura 35 Contenido carpeta conector debezium sql server 115
Figura 36 Decodificador online 116
Figura 37 worker.properties 117
Figura 38 connector.properties 117
Figura 39 Configuración worker.properties 119
Figura 40 Archivo JSON de países 125
Figura 41 Archivo JSON de tipos de comercio 126

Figura 42 Archivo JSON de expresiones para determinar el nombre del comercio..... 127
Figura 43 Resultado de trama iso8583 en isotester 129
Figura 44 Trama iso8583 en microservicio 129
Figura 45 Campos almacenados en Elasticsearch 132
Figura 46 Patrón de índice en Kibana 135
Figura 47 Visualización de transacciones en tiempo real en Kibana..... 136
Figura 48 Tablero de control de tarjetas de débito de la institución financiera 137

Índice de Tablas

Tabla 1 Tabla comparativa de herramientas CDC basados en archivos logs 47
Tabla 2 Tabla comparativa Apache Kafka vs RabbitMQ 52
Tabla 3 Ejemplos MTI 61
Tabla 4 Ejemplo ISO 3166 para formato de Ecuador 62
Tabla 5 Ranking de motores de búsqueda 64
Tabla 6 Tabla comparativa de tecnologías para almacenamiento NoSQL como motores de búsqueda 66
Tabla 7 Tabla comparativa de tecnologías para tableros de control 70
Tabla 8 Características de Software y Hardware 77
Tabla 9 Top 10 de tipos de comercio en Ecuador 98
Tabla 10 Top 10 de lugares de comercio para Tiendas de Comestibles 99
Tabla 11 Top 10 de lugares de comercio para Droguerías y Farmacias 99
Tabla 12 Top 10 de lugares de comercio en Grandes Almacenes 100
Tabla 13 Top 10 de tipos de comercio en Estados Unidos 100
Tabla 14 Top 10 de lugares de comercio en Medios de productos digitales como Juegos. 101
Tabla 15 Top 10 de lugares de comercio en Tiendas de software informático 101

Índice de Códigos

Código 1 Script para activación de base de datos y tabla de transacciones de tarjetas de débito 111
Código 2 Script para desactivación de base de datos y tabla de transacciones 112
Código 3 Script para comprobación de funciones habilitadas bajo CDC 113
Código 4 Dependencia de Maven para el conector de Apache Kafka..... 120
Código 5 Propiedades de Configuración de un consumidor Kafka..... 122
Código 6 Consumidor Kafka para procesamiento de registros 123
Código 7 Transformación trama ISO8583 128
Código 8 Dependencia de Maven para el conector de Apache Kafka..... 130

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Tatiana Ximena Muñoz Sánchez en calidad de autora y titular de los derechos morales y patrimoniales del trabajo de titulación "DISEÑO E IMPLEMENTACIÓN DE MICROSERVICIOS PARA GENERAR UN MODELO DE ANÁLISIS DE TRANSACCIONES MASIVAS DE TARJETAS DE DÉBITO PARA UNA INSTITUCIÓN FINANCIERA", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 12 de abril de 2022

Tatiana Ximena Muñoz Sánchez

C.I: 0104586292

Cláusula de Propiedad Intelectual

Tatiana Ximena Muñoz Sánchez, autora del trabajo de titulación "DISEÑO E IMPLEMENTACIÓN DE MICROSERVICIOS PARA GENERAR UN MODELO DE ANÁLISIS DE TRANSACCIONES MASIVAS DE TARJETAS DE DÉBITO PARA UNA INSTITUCIÓN FINANCIERA", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autora.

Cuenca, 12 de abril de 2022

Tatiana Ximena Muñoz Sánchez

C.I: 0104586292

Agradecimientos

Este trabajo de titulación ha sido posible gracias al apoyo del Ing. Mauricio Espinoza por su dedicación y paciencia para dirigir esta tesis e impartir su conocimiento y estar siempre presto a ayudarme. De igual manera, agradezco a la institución financiera y en especial al Director de Tecnología por el apoyo brindado para el desarrollo de este proyecto.

Tatiana

Dedicatoria

Dedico este trabajo de titulación primeramente a Dios por darme fortaleza en todo momento. A mis padres Francisco y Susana que siempre me han motivado a seguir mis sueños y me han apoyado en cada nueva meta que me he propuesto cumplir. A ti mamita en el cielo, gracias por todo lo que me enseñaste y sobre todo con tu bendición diaria a impulsarme a no rendirme. A mi abuelito, Papito Genaro mi angelito que siempre con sus palabras de aliento me daba ánimos para esforzarme a conseguir mis objetivos. Esta tesis está especialmente dedicada a UD.

Tatiana

CAPITULO 1 - INTRODUCCIÓN

A pesar de los marcados beneficios de la incorporación de ambientes tecnológicos que soporten procesos de recolección, transmisión, y análisis de grandes volúmenes de datos, estas soluciones no han sido completamente explotadas en el sector financiero. Según (Vázquez, 2019) los desafíos que enfrentan este tipo de entidades no son sencillos sobre todo lo relacionado al volumen, privacidad, ética, y seguridad de los datos; todos estos aspectos constituyen retos para garantizar que los datos recopilados estén a salvo de vulnerabilidades de seguridad, robos de identidad y venta indebida a terceros.

Existen muy pocas soluciones en la literatura que permiten identificar las dificultades y retos de incorporar ambientes tecnológicos para la toma de decisiones en el sector financiero. La mayoría de propuestas existentes en la literatura ofrecen una visión muy general de: i) la arquitectura que puede ser usada para recolectar datos en tiempo real y las herramientas que pueden ser usadas para el análisis de datos (TAT, 2019), o ii) los algoritmos y herramientas que pueden ser usados para detectar fraudes en tarjetas de crédito (Montero, 2016).

En general, las instituciones financieras se caracterizan por ofrecer a los usuarios una gama muy amplia de servicios, los cuales generalmente están enfocados en ahorro y crédito. Algunos ejemplos de estos servicios son: cuentas de ahorro, tarjetas de crédito, tarjetas de débito, entre otros. Una característica que identifica a la mayoría de estos servicios es la gran cantidad de datos que generan, lo cual ha provocado un desafío importante en cómo gestionar estos datos y sobre todo cómo usarlos para la toma de decisiones inteligentes y oportunas.

De entre los servicios ofertados por las instituciones financieras, el uso de tarjetas de crédito y débito son muy demandadas por la mayoría de socios (Minsait, 2021). Especialmente las tarjetas de débito, se están posicionando como uno de los servicios más utilizados ya que permiten realizar transacciones (sin mayor costo) a través de medios electrónicos como puntos de venta (POS) y comercio electrónico. Las transacciones producidas por el uso de una tarjeta de débito, generan un crecimiento exponencial de los datos en una entidad financiera, provocando además que éstos sean inmanejables. Aquí es donde, técnicas como la Minería de Datos, Big Data, y la utilización de herramientas de Inteligencia de Negocios ayudan a obtener el máximo provecho posible del valor de estos datos.

Dentro del sector bancario, el no contar con un ambiente tecnológico que permita recolectar y analizar los grandes volúmenes de datos producidos por las operaciones efectuadas a través de una tarjeta de débito, provoca que la entidad pierda ventaja competitiva al, no aprovechar estos datos para la toma de decisiones. De hecho, el poder gestionar en tiempo real datos como: i) el flujo de transacciones de las tarjetas de débito, ii) los comercios en donde se ha efectuado una transacción, iii) el dato del país en donde se efectuó una compra, o iv) el sector comercial en donde más se usa una tarjeta de débito; puede ofrecer una gran oportunidad para ampliar y mejorar los procesos del modelo del negocio actual.

Para evitar adaptaciones o configuraciones posteriores ante cambios en los procesos generados a través de las transacciones de las tarjetas de débito, se propone en este trabajo, el uso de

un sistema de almacenamiento flexible que no utilice estructuras fijas como el modelo relacional para el almacenamiento de los datos. Las bases de datos NoSQL pueden ayudar a este propósito. Además, este sistema de almacenamiento presenta algunas ventajas importantes que permiten: i) procesar los datos en máquinas con pocos recursos, ii) mejorar la escalabilidad horizontal mediante la incorporación de más nodos, y iii) manejar grandes cantidades de datos. Estas características son esenciales para el diseño e implementación del ambiente tecnológico que se propone en este trabajo.

1.2 ANTECEDENTES

Hoy en día, la instalación y configuración de un ambiente para transmisión y análisis de datos en tiempo real en el sector financiero, junto con herramientas de inteligencia de negocios es una temática aún no explotada en su totalidad. Esto tiene como consecuencia que existan muy pocas guías relacionadas al montaje de un ambiente de flujos de datos en tiempo real que hagan uso de bases de datos no relacionales que ayuden a la toma de decisiones oportunas a través de mecanismos de inteligencia de negocios. Bajo esta perspectiva, se han propuesto algunos trabajos de investigación en ambientes generales y relacionados con el sector financiero, que ofrecen una visión de la arquitectura para procesar grandes volúmenes de datos en tiempo real. El proyecto del TAN KIN TAT de la Universitat de Lleida, titulado "Arquitectura Big Data para la ejecución de Reglas de Negocio en Tiempo Real" (TAT, 2019), expone de una manera amplia, como las herramientas de procesamiento aplicada en flujos de datos continuos genera un valor positivo dentro de las empresas bancarias. Otra investigación importante es la realizada como trabajo de fin de master por Javier Mansilla Montero, titulado" Detección de fraude bancario en tiempo real utilizando tecnologías de procesamiento distribuido", (Montero, 2016) donde se explica los beneficios que conlleva el uso de algoritmos a través de colas de mensajes con la herramienta TIBCO y Kafka para detectar fraudes en tarjetas de crédito.

Otro de los aspectos que ha requerido atención por parte del sector bancario es la intermediación de mensajes en el uso de servicios de pago como las tarjetas de crédito y débito, donde las transacciones se generan desde los puntos de venta (lugar en el cual se efectúa la compra) hasta las entidades financieras (donde se realiza la actualización del saldo de la cuenta en caso de las tarjetas de débito). En (Vlad Bucur, Ovidiu Stan, Liviu Micle, 2020) los autores describen que "La mayoría de las empresas en el ámbito financiero emplean un mecanismo de cola para administrar el flujo de mensajes. …. Kafka, es uno de los principales sistemas de mensajería actualmente en uso, pudiendo simplificar varios sistemas, como la intermediación de mensajes, la persistencia o la tolerancia a fallas". Por todo lo antes mencionado, si bien existen trabajos relacionados al manejo de grandes volúmenes de datos dentro de entidades financieras, no existe evidencia de propuestas que ofrezcan detalles de la implementación de microservicios para la intermediación de mensajes y el uso de inteligencia de negocios en la obtención de información para toma de decisiones en servicios con alta demanda dentro de las instituciones financieras como son las tarjetas de débito.

1.3 JUSTIFICACIÓN

El presente trabajo propone el diseño e implementación de una arquitectura basada en tecnologías flexibles como los microservicios (aplicadas al sector financiero), de forma que se pueda mejorar la escalabilidad, mantenibilidad y versatilidad en el desarrollo de mecanismos para el manejo en tiempo real de los datos, y que se pueda, a través de los datos generar información para la toma de decisiones oportunas sobre el uso de tarjetas de débito (como medio de pago seleccionado en esta propuesta).

La importancia de esta arquitectura, radica en el uso de microservicios, lo cual mejora el rendimiento en un sistema, maximizando el despliegue de nuevas versiones del producto y mejorando la funcionalidad al ser totalmente independiente. Dentro del proyecto propuesto, se establece un enfoque inicial de uso de esta tecnología en una institución financiera, para satisfacer el manejo de una única funcionalidad del negocio característica propia de los microservicios. Además, con la combinación de herramientas de monitoreo, se busca tener una visión total de las transacciones en tiempo real. Dentro del sector financiero, la innovación deber ser constante para ser competitivos y rentables en industrias donde los procesos digitales y las nuevas tecnologías se han convertido en una necesidad existente.

Algunos reportes proporcionados por la Asociación de Bancos Privados (Asobanca) muestran una gran cantidad de datos que se generan en tiempo real desde cada uno de los servicios bancarios, especialmente, las transacciones por el uso de tarjetas de crédito y débito. “En febrero de 2021, se registraron 8,7 millones de tarjetas, lo que representó un incremento de 9,3% frente al mismo mes del año 2020. El número de transacciones también subió, lo que evidencia que las personas usan mucho estos medios de pago” (Universo, 2021). Este volumen de información, ha generado un desafío importante, lo cual conlleva a tener procesos que van desde la integración de sistemas externos a través de flujos de datos en tiempo real, generar la limpieza y análisis respectivo, y su posterior implementación en almacenes de datos flexibles que permitan ayudar a la toma de decisiones inteligentes y oportunas.

La importancia de desarrollar este proyecto radica en la falta de información actual dentro de las instituciones financieras para determinar el flujo de transacciones de tarjetas de débito, comercios, países y sectores donde se realiza el uso de este servicio en tiempo real. El contar con esta información, ofrece una gran oportunidad a las instituciones ampliando y mejorando los procesos del modelo de negocio actual, además de utilizar tecnología relativamente nueva e independiente como los microservicios que permitan adaptabilidad y mantenibilidad en caso de que se desee abordar nuevos servicios y fuentes de información a la arquitectura propuesta.

1.4 PROBLEMÁTICA

La institución financiera que servirá como escenario de aplicación del proyecto maneja un flujo de información importante, dado los múltiples productos o servicios que oferta. Uno de sus

servicios pioneros son las tarjetas de débito, que permiten a un usuario controlar los gastos o compras realizadas en base al saldo disponible en su cuenta, además de permitir a través de POS (Puntos de Venta) y comercio electrónico realizar de manera sencilla, rápida y segura compras de productos o servicios sin la necesidad de pagar con dinero físico. El volumen de datos generado por las transacciones de las tarjetas de débito requiere de procesos de recolección en tiempo real, además de un sistema de almacenamiento flexible que se adapte a las necesidades cambiantes de las transacciones efectuadas mediante este tipo de servicio. Estos mecanismos de recolección, almacenamiento y procesamiento de grandes volúmenes de datos actualmente no han sido implementados dentro de la entidad financiera, por este motivo se propone realizar una integración que va desde la captura, transformación, almacenamiento y visualización de esta información a través de herramientas de inteligencia de negocios para la toma de decisiones sobre este medio de pago.

1.5 OBJETIVOS

Objetivo General

- Diseñar e implementar microservicios para la generación de un modelo de análisis de transacciones masivas de tarjetas de débito para una institución financiera.

Objetivos Específicos

- Diseñar e implementar microservicios para capturar, transformar y cargar el flujo de datos en tiempo real de transacciones de tarjetas de débito de una institución financiera.
- Crear y configurar una base de datos NoSQL para almacenar la información de las transacciones de tarjetas de débito de una institución financiera.
- Evaluar y seleccionar las técnicas de filtrado y de análisis de datos, aplicables a la información que permita la extracción de información relevante para la toma de decisiones.
- Identificar los indicadores claves para la obtención de información que será utilizada en la elaboración de cuadros de mando a nivel estratégico.
- Crear un informe con resultados de los indicadores seleccionados.
- Diseñar representaciones gráficas de inteligencia de negocios que permitan visualizar el comportamiento de los indicadores.

1.6 METODOLOGÍA

La metodología utilizada en este trabajo de titulación aplica el marco de la ciencia de diseño empleada en sistemas de información e ingeniería de software, el cual proporciona las pautas sobre cómo estructurar los objetivos de la investigación y diseñar un enfoque de validación basado en

pruebas. Para validar el funcionamiento de la arquitectura propuesta, se utilizó el ciclo de ingeniería propuesta por Wieringa (Wieringa, 2014).

1. **Problema de la investigación**: Para analizar el problema actual del proceso de toma de decisiones con un medio de pago como son las tarjetas de débito en la institución financiera, se realizaron algunas reuniones con personal de la Dirección de Tecnología y Operaciones de Tarjetas de Crédito/Débito. Dichas reuniones permitieron determinar los indicadores clave a observarse en los tableros de control, lo cual es un elemento esencial de este proyecto. Además, a través de una búsqueda bibliográfica se identificaron los problemas en la integración de nuevas tecnologías en tiempo real que incluya microservicios, brokers de mensajería y la inteligencia de negocios. La revisión bibliográfica permitió determinar que la creación de un microservicio a medida puede mejorar la limpieza de los datos y optimizar los procesos de toma de decisiones oportunas y fiables en entidades bancarias con alto volumen de información en transacciones sensibles efectuadas por POS o canales electrónicos.

2. **Diseño:** Con base en los objetivos planteados se analizó junto al personal de las áreas involucradas el diseño de los componentes de la arquitectura propuesta. Con el fin de lograr que la arquitectura propuesta sea modular, se incorporó el uso de un microservicio que permita ejecutar tareas de limpieza y conversión de los datos, generando información valiosa que aporte a mejorar el uso de las tarjetas de débito de la entidad bancaria.

   Por otra parte, para el diseño de las gráficas del tablero de control se planteó el uso de un diseño en forma de pirámide invertida utilizada dentro de la inteligencia de negocios para presentar la información más relevante en la parte superior, seguida por los detalles de tendencias que permitan profundizar sobre su funcionamiento en la parte inferior.

   El diseño propuesto para el tablero de control dentro de la institución financiera puede facilitar la medición de transacciones originadas con el medio de pago de tarjetas de débito como son totales procesados, número de registros, comercios, países, entre otros. Toda esta información recopilada permite proyectar visualizaciones ágiles en tiempo real al personal involucrado para toma de decisiones efectivas.

3. **Validación**: Para la validación de la arquitectura propuesta se empleó una base de datos de pruebas réplica de producción con datos reales de transacciones generadas durante un período de pruebas que por motivos de confidencialidad de la información solicitada por la institución financiera no pueden ser expuestos en un repositorio

público. Además, para realizar la simulación de envío de eventos generados por POS o canales electrónicos en tiempo real se creó un script con el manejo de inserciones de registros que fueron visualizadas en el tablero de control creado en tiempo real.

4. **Implementación:** Una vez obtenidos los resultados del procesamiento con la base de datos de pruebas y generadas las gráficas del tablero de control diseñado, se creó un informe con los resultados obtenidos basándose en el objetivo planteado en la propuesta de tesis, donde se analizó además junto con el personal involucrado si es posible implementar la arquitectura propuesta en el área de producción de la entidad financiera y los beneficios o riesgos que se podrían generar. Para este proceso se analizaron las gráficas diseñadas con las áreas implicadas inicialmente y se estableció como propuesta, reuniones con áreas adicionales de la organización como cumplimiento y seguridad de la información.

1.7 ALCANCE

El desarrollo de este proyecto empezará con un análisis de las herramientas existentes que soporten el diseño e implementación de la arquitectura propuesta; que van desde la captura de datos, el procesamiento de flujos de datos en tiempo real y el almacenamiento y visualización de los datos para transformarlos en información oportuna y confiable.

Una vez efectuado este estudio, se implementará o configurará según sea el caso, la herramienta seleccionada para cada proceso en un ambiente de pruebas dentro de la institución financiera para verificar su funcionamiento con el manejo de transacciones masivas de tarjetas de débito.

A través del uso de un microservicio se deberá capturar, transformar y cargar los datos a un esquema de datos flexible que contenga los indicadores relevantes definidos previamente para la construcción del tablero de control sobre el cual se enfocará este proyecto.

Para almacenar el resultado de la información se utilizará un sistema de almacenamiento flexible que permita procesar, analizar y obtener el mayor provecho a los datos. Se utilizará este tipo de almacenamiento debido a que la estructura de las transacciones puede cambiar en el tiempo, por lo que necesita ser fácilmente escalable.

Por último, se realizará un análisis y se escogerá dos tipos de cuadros de mando de visualización de inteligencia de negocios, que permitirán generar una mejor visibilidad de las transacciones de las tarjetas de débito para ayudar a la toma de decisiones. Se pretende que los sistemas de visualización ofrezcan a la organización la posibilidad de llevar a cabo un seguimiento y análisis eficaz, generando inteligencia empresarial a través de los datos en movimiento.

Limitaciones

- La implementación del proyecto servirá únicamente para el proceso de tarjetas de

débito, cualquier otro tipo de servicio en la institución financiera no está considerado.

- Se diseñarán e implementarán únicamente dos tipos de gráficas de inteligencia de negocios para el manejo de los indicadores para toma de decisiones que reflejarán información sobre los países con más uso de la tarjeta de débito de la institución financiera, los comercios y servicios digitales con más demanda y montos estimados del consumo de este servicio.

1.8 ESTRUCTURA DEL DOCUMENTO

En este trabajo se presenta el estudio para el "DISEÑO E IMPLEMENTACIÓN DE MICROSERVICIOS PARA GENERAR UN MODELO DE ANÁLISIS DE TRANSACCIONES MASIVAS DE TARJETAS DE DÉBITO PARA UNA INSTITUCIÓN FINANCIERA" en base a la necesidad de poder generar una herramienta de visualización en tiempo real para toma de decisiones de este tipo de servicio.

Los siguientes capítulos se han estructurado de la siguiente manera:

- Capítulo 2: Contexto Tecnológico donde se describe e introduce los conceptos necesarios para el diseño de la arquitectura planteada en este trabajo.
- Capítulo 3: Trabajos Relacionados describe algunas de las investigaciones más relevantes relacionadas con el contexto de flujos de datos en tiempo real, microservicios e inteligencia de negocios.
- Capítulo 4: Arquitectura Propuesta presenta un análisis y selección de las herramientas tecnológicas necesarias que permitan diseñar el flujo de comunicación de las transacciones en tiempo real.
- Capítulo 5: Implementación del diseño propuesto, muestra de forma detallada cada uno de las actividades realizadas para conseguir los objetivos planteados, en base a las herramientas seleccionadas en el Capítulo 4.
- Capítulo 6: Conclusiones y Trabajos Futuros.

CAPITULO 2 CONTEXTO TECNOLÓGICO

2.1 INTRODUCCIÓN

En un mundo globalizado, donde los datos producidos dentro de las organizaciones juegan un papel importante, las nuevas tecnologías de gestión de datos permiten ejecutar tareas en segundo plano, evitando así interrumpir los procesos diarios y tener que esperar al finalizar el día para alimentar sistemas de soporte a decisiones. Una de las grandes ventajas de la incorporación de herramientas y técnicas de gestión de datos en una organización, es la posibilidad de procesar grandes volúmenes de datos en tiempo real y obtener información relevante para impulsar operaciones comerciales, generando una ventaja competitiva sobre otras empresas en el mercado.

A continuación, se inicia en este capítulo con una descripción de las arquitecturas de Big Data más comunes para procesos en tiempo real: Lambda y Kappa, las cuales se diferencian en los flujos de tratamiento de datos que intervienen. Además, se presentan también en este capítulo los elementos tecnológicos que ayudarán a entender el proyecto de tesis.

2.2 ARQUITECTURA LAMBDA

La arquitectura Lambda es una arquitectura de procesamiento genérica que tiene como objetivo realizar lecturas y escrituras con baja latencia; además de poseer características como ser escalable y tolerante a fallos de infraestructura o errores humanos. La arquitectura Lambda como se indica en la Figura 1 está compuesta por tres capas principales que son las responsables de la ejecución de las tareas en el procesamiento de los datos.

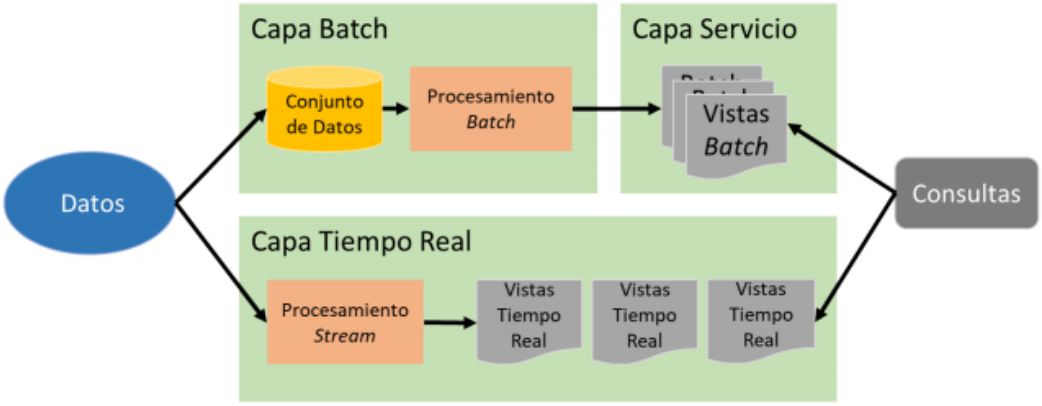

Figura 1 Arquitectura Lambda.

Disponible en: https://www.paradigmadigital.com/techbiz/de-lambda-a-kappa-evolucion-de-las-arquitecturas-big-data/

Las capas de esta arquitectura se describen a continuación como:

- **Capa batch (batch layer):** Esta capa gestiona la información que llega a la misma sin modificar. A través de un proceso batch se hace el tratamiento de los datos cuyo resultado serán las Vistas Batch (Batch Views). La capa por lotes tiene como objetivo poder corregir cualquier error (volviendo a calcular), basándose en el conjunto de datos completo y luego actualizando las vistas existentes. Para realizar la consulta de los datos se accede a estas vistas para ofrecer la información ya transformada.

- **Capa de servicio (serving layer):** La salida de la capa batch y velocidad se almacena en la capa de servicio en donde se indexa los Batch Views de forma que permite baja latencia en las consultas. Permite responder así a la solicitud del usuario combinando los resultados de las vistas del batch y las vistas en tiempo real.

- **Capa de velocidad (speed layer):** Esta capa tiene como objetivo minimizar la latencia al proporcionar vistas en tiempo real de los datos más recientes. Esencialmente, la capa de velocidad es responsable de compensar la "brecha" causada por el retraso de la capa de lote al proporcionar vistas basadas en los datos más recientes.

La idea principal de esta arquitectura es poder replicar la información que entra al proceso tanto en la capa de velocidad como en la capa de lotes para que los datos estén disponibles. Este tipo de arquitectura aprovecha además las ventajas tanto de los conceptos batch como streaming, pero no es recomendable en circunstancias en las que la toma de decisiones deba ser instantánea dado que requiere de tiempos relativamente largos para completar operaciones de escritura y consultas. Un inconveniente adicional es la necesidad de requerir el mantenimiento de los dos sistemas de

procesamiento tanto por lotes como el de tiempo real, resultando el proceso de ajuste de parámetros relativamente complicado.

2.3 ARQUITECTURA KAPPA

Esta arquitectura nació en el 2014 con Jay Kreps (Kreps, 2014), quien critica el consumo innecesario de los recursos que contemplan mantener y tratar la información en dos capas con el objetivo de llegar a tener resultados similares. Kreps opina que para esta arquitectura se debe manejar la capa de batch como un subproceso y manejar en la capa de streaming el procesamiento por lotes, pasando a considerar todo como un flujo de datos ininterrumpido como se presenta en la Figura 2.

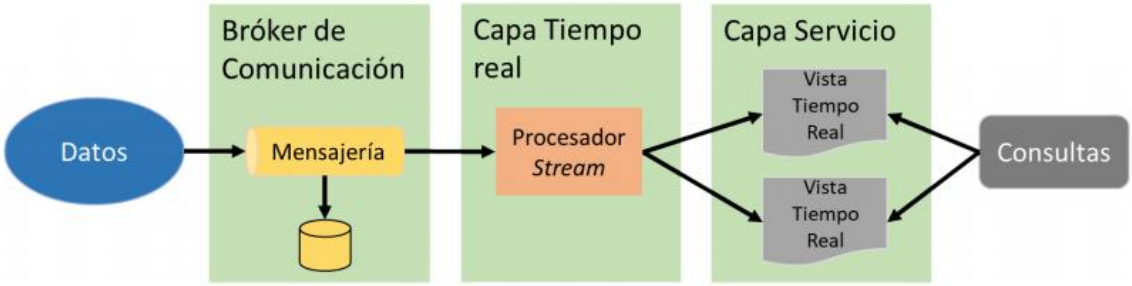

Figura 2 Arquitectura Kappa

Disponible en: https://www.paradigmadigital.com/techbiz/de-lambda-a-kappa-evolucion-de-las-arquitecturas-big-data/

Uno de los pilares fundamentales de esta arquitectura es considerar a la información como un solo flujo de procesamiento, generando bajas latencias en el sistema, además de garantizar que los eventos se leen y almacenan en el orden que se generan.

Las capas de esta arquitectura se describen a continuación como:

- **Capa de Tiempo Real (real-time layer):** Esta capa simplifica la arquitectura Lambda, en la que se elimina la capa batch y todo el procesamiento se realiza sobre la misma.
- **Capa de Servicio (serving layer):** A través de la misma se puede disponer de los resultados o de la información procesada, información original o datos sin procesar de las consultas del usuario, similarmente a la arquitectura Lambda.

De los dos tipos de arquitecturas para procesamiento de datos en tiempo real existentes, se puede inferir que la **arquitectura Kappa** es la adecuada para el presente proyecto de tesis ya que se adapta por su diseño y características al ámbito financiero, comportándose mejor en:

- **Tiempos de respuesta:** Permite hacer el desarrollo, prueba y operación de los sistemas sobre un único framework de procesamiento. Dentro de las organizaciones financieras poder tomar decisiones en tiempo real permite aprovechar la disponibilidad inmediata de los datos para análisis y monitoreo constante de su funcionamiento.
- **Costos de adaptación y mantenimiento:** En caso de que se requiera realizar mantenimiento a los parámetros de los algoritmos para el procesamiento de datos, la arquitectura Kappa brinda la ventaja de centralizar la lógica del proceso, permitiendo agilizar tanto el proceso operativo como de recursos humanos técnicos limitados que se poseen dentro de las organizaciones para solventar alguna falla que se ocasione.
- **Necesidades del negocio:** Esta arquitectura es apropiada para el volumen de transacciones que se producen en el sector financiero, donde, el flujo de las mismas es continuo a diferencia de un procesamiento en lotes permitiendo la reorganización de la información a partir de diversas fuentes de manera eficiente, proporcionando además que las consultas que se realicen sean rápidas gracias a la capa de tiempo real.

Los componentes de la arquitectura Kappa pueden ayudar a soportar la transmisión de transacciones de tarjeta de débito en tiempo real en una entidad financiera. Para facilitar el entendimiento de todos los procesos, técnicas y herramientas que están involucrados en el procesamiento streaming[1] a través de flujos constantes de datos, se propone una arquitectura genérica basada en los componentes de la arquitectura Kappa. La Figura 3, muestra un esquema general de los componentes que pueden servir para estos propósitos.

[1] El procesamiento en streaming está basado en la idea de procesar los datos de forma continua.

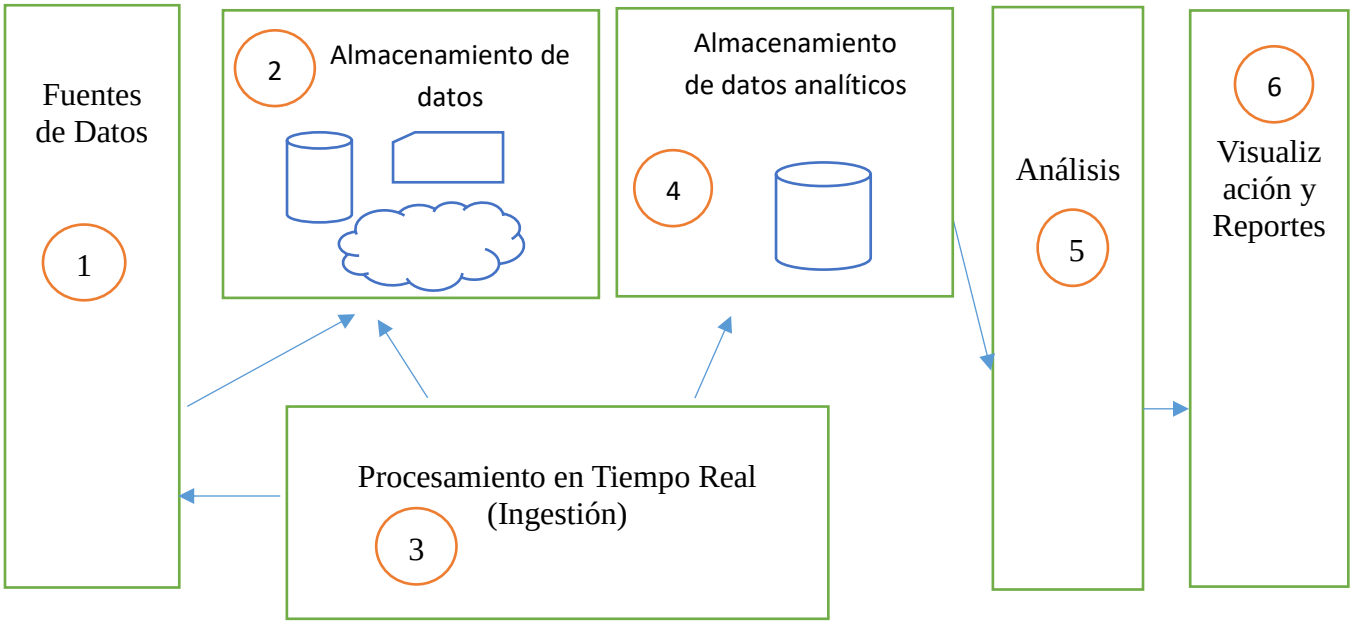

Figura 3 Arquitectura para procesamiento en tiempo real

En una institución financiera, el volumen de transacciones que se generan por los servicios que ofertan puede convertirse en grandes flujos de datos constantes que se capturen en tiempo real y se procesen con una latencia mínima para generar informes o respuestas automáticas. Por lo tanto, una arquitectura de procesamiento en tiempo real resulta adecuada para toma de decisiones inmediatas. A continuación, se describe cada uno de los componentes que se han determinado como claves para el proceso.

2.3.1 FUENTES DE DATOS

Hoy en día, existe una variedad de tipos de fuentes de datos que se pueden aprovechar para una transmisión en tiempo real y que ayuden a generar valor para la toma de decisiones en las organizaciones financieras como: sistemas externos, aplicaciones internas de backend y frontend o bases de datos. Las fuentes de datos pueden ser parte de las siguientes categorías:

- **Biométricos:** Permiten la identificación automática de una persona a través de sus características anatómicas, por ejemplo, el reconocimiento facial, firma biométrica, etc.

- **Máquina a máquina:** Son aquellas tecnologías que permiten la conexión de diferentes dispositivos entre sí. Entre algunos ejemplos se encuentran los GPS, sensores, cajeros automáticos.

- **Datos de transacciones:** Son las operaciones que se registran habitualmente dentro de una organización. Los departamentos de facturación, centro de llamadas, ERP, CRM, inventarios de venta, entre otras.

- **Redes sociales y páginas web:** Es una de las fuentes de datos más utilizadas en la actualidad. Por ejemplo, la información generada en los clics de las páginas de las instituciones.

- **Generados por humanos:** Los datos que generamos los humanos, por ejemplo, cuando se hace uso de las tarjetas de crédito o débito.

Los datos provenientes de las diferentes fuentes de acuerdo a su estructura pueden ser clasificados según su formato en:

- **Datos estructurados**: Datos ordenados y bien definidos en cuanto a su formato, tamaño y longitud. En donde se encuentran las bases de datos relacionales caracterizados por mantener un esquema determinado a través de tablas donde se almacenan los datos. Los sistemas de gestión de bases de datos relacionales (RDBMS) manejan SQL (Structured Query Language) para administrar y recuperar la información.

- **Datos no estructurados:** No tienen una estructura interna identificable. Los datos están desorganizados y no tienen valor hasta que se ordenan, se identifican y se almacenan.

- **Datos semiestructurados:** Se trata de información no regular y que no se pueden gestionar de manera estándar. Se organizan mediante etiquetas o «tags» que permiten agruparlos y jerarquizarlos. Formatos de datos semiestructurados son por ejemplo HTML, XML o JSON. Las bases de datos NoSQL forman parte de esta clasificación ya que permiten almacenar información que no se adapta al formato registro/tabla, facilitando además el intercambio de datos entre distintas bases de datos.

Los datos se pueden transportar hacia sistemas de almacenamiento gracias a diversos protocolos de red, como el conocido Protocolo de transferencia de archivos (FTP) y el Protocolo de transferencia de hipertexto (HTTP), o cualquiera de las innumerables interfaces de programación de aplicaciones (API) proporcionadas por sitios web, aplicaciones en red, servicios web, entre otros; y

llevadas a almacenes de datos o directamente capturadas por el proceso de ingestión de datos que permite el procesamiento instantáneo de la información.

Entre los formatos de mensajes en los que se puede intercambiar los datos durante el proceso de transmisión, están JSON y XML, los cuales son formatos ligeros muy usados actualmente.

2.3.2 ALMACENAMIENTO

Este componente permite recolectar la información de las fuentes de datos (componente 1) y almacenarlas en la nube, repositorio de datos, u otros. En un mecanismo de procesamiento streaming los datos son capturados a través de un intermediador de mensajes en la capa denominada "Procesamiento en Tiempo Real", por ejemplo, se puede monitorear una carpeta en busca de nuevos archivos y procesarlos a medida que se crean o actualizan. Otra forma de recolección es mediante el uso de bases de datos, las cuales almacenan las transacciones generadas por sistemas externos y son utilizadas como destino de salida para los datos capturados en tiempo real, de manera que, las mismas son detectadas cuando se realiza alguna acción sobre este almacén de datos como inserciones, modificaciones o eliminaciones.

En este tipo de arquitecturas que gestionan grandes volúmenes de datos, es posible almacenar las fuentes de datos en: sistemas de almacenamiento de archivos distribuidos HDFS, bases de datos NoSQL y/o un sistema de gestión de bases de datos relacionales (RDBMS).

2.3.2.1 SISTEMA DE FICHEROS DISTRIBUIDO DE HADOOP (HDFS)

HDFS es una tecnología para almacenamiento distribuida utilizado en sistemas Big Data para replicar los datos, escalar horizontalmente y distribuir los datos para realizar procesamiento con el framework de Hadoop.

Este sistema de almacenamiento permite tipos de datos estructurados, no estructurados y semi-estructurados, es decir, permite almacenar cualquier tipo de información y replicarla/distribuirla en múltiples nodos para asegurar su disponibilidad y que el sistema tenga tolerancia a fallos en varios de estos nodos.

HDFS tiene dos componentes principales: NameNode y DataNode. Los NameNodes están encargados de almacenar los metadatos y la localización de los bloques que componen cada archivo. Los DataNodes por su parte almacenan los datos y gestionan los discos.

2.3.2.2 BASES DE DATOS RELACIONALES

Este tipo de bases de datos cuentan con una amplia trayectoria en el mundo digital. Son ampliamente usadas por su tradición y eficiencia para organizar información. Su almacenamiento se basa en tablas definidas de forma anticipada para su funcionamiento.

Las principales características de este tipo de repositorios, ampliamente utilizadas para varios de los sistemas informáticos en cualquier institución financiera, se describen a continuación:

- Utilizan un lenguaje de consulta estructurado (SQL) para manipular los datos de filas, columnas y registro de la información almacenada en tablas.
- Permiten almacenar los datos evitando duplicidad de información.
- Son ampliamente utilizadas dado el tiempo y aceptación que han tenido.
- Asegura que las transacciones se realicen completamente “todo o nada”

2.3.2.3 BASE DE DATOS NO RELACIONALES (NoSQL)

Este tipo de bases de datos no tienen un identificador que sirva de relación entre los conjuntos de datos. Son comúnmente utilizadas en programas y aplicaciones que manejan grandes volúmenes de datos. Por esta razón, sería ideal para la cantidad de transacciones de tarjetas de débito que maneja la entidad financiera y la necesidad de flexibilidad en el modelo de los datos, pretendiendo a posterior generar nuevos análisis e indicadores de desempeño.

Además, los motores de búsqueda de este tipo de bases de datos, ayudan a generar consultas extremadamente rápidas en comparación con las bases SQL, pues:

- Poseen esquemas flexibles para crear aplicaciones modernas.
- No requieren de una estructura de datos definida para la manipulación de la información.
- Permiten adaptarse de forma rápida al crecimiento del volumen de los datos o la forma en que se ingresan.
- Es posible añadir nodos para mejorar el rendimiento de las operaciones.
- Tienen bajos requerimientos de recursos por lo que se puede montar en máquinas de coste más reducido.

2.3.3 PROCESAMIENTO EN TIEMPO REAL (INGESTIÓN)

Este componente es el encargado de la extracción o detección de información de diversas fuentes y enviarlos a un lugar de destino donde se puedan almacenar y analizar (ver Figura 3). Con este enfoque apenas los datos son capturados se transfieren para procesamiento; es decir, en lugar de tener que procesar "grandes cantidades" como el procesamiento batch, en todo momento son procesadas "pequeñas cantidades" de manera inmediata.

La ingestión de datos en tiempo real es fundamental para acceder a los datos que brindan un valor significativo a una empresa. La ingestión basada en datos actualizados y completos, permite a las organizaciones tomar decisiones operativas más eficientes y con mayor rapidez.

Una vez que los datos son capturados desde las diversas fuentes de datos, éstos son transferidos para depósito y análisis a: plataformas en la nube, bases de datos transaccionales, archivos y sistemas de mensajería; por ejemplo, usando la captura de datos modificados (CDC) no intrusiva. Existen múltiples herramientas que pueden ser utilizadas en los procesos de ingestión de datos en tiempo real como Apache Kafka, Apache Flume, RabbitMQ, entre otras. Generalmente, tres pasos ocurren durante la ingestión de datos:

- **Extracción**: usado para recolectar datos desde la fuente.
- **Transformación**: para validar, limpiar y normalizar los datos asegurándose de su confiabilidad para objetivos propuestos.
- **Carga**: que permite colocar los datos en el destino correcto para su análisis posterior.

El procesamiento de los datos en tiempo real involucra algunos procesos y tecnologías clave que se describen a continuación:

2.3.3.1 CAPTURA DE CAMBIOS EN LOS DATOS (CDC)

Captura de Cambios en los Datos (CDC) es una tecnología que actúa en segundo plano, como un histórico para almacenar los cambios realizados en una base de datos. De esta manera se podrá mejorar el flujo y conservar una trazabilidad completa de la información. Las funciones implementadas para soportar CDC utilizan un proceso para escanear los registros de transacciones de la base de datos a fin de capturar los datos modificados; las transacciones no se ven afectadas y el impacto en el rendimiento de los servidores de origen se minimiza.

Las operaciones típicas de un usuario sobre una base de datos que desencadenan un control de cambios son: inserciones, actualizaciones y eliminaciones. Las inserciones y eliminaciones, actualizan sobre la tabla de cambio un nuevo registro que almacena el dato insertado o eliminado. En

el caso de una actualización, la tabla de cambio almacena dos filas: una con los datos de antes de la actualización y otra para los nuevos datos modificados.

La mayoría de bases de datos tienen soporte para captura de cambios, y en caso de otras bases de datos como Oracle se obtiene un soporte de terceros para ejecutar esta función. La detección de fraudes o campañas de marketing dirigidas son dos ejemplos del uso de la tecnología CDC en tiempo real en el ámbito financiero.

Una vez que se conoce el propósito de este componente, se hace necesario describir algunas ventajas y desventajas que aporta el uso de esta tecnología:

Ventajas

- Permite alimentar en tiempo real las transacciones de la base de datos a las aplicaciones de análisis para decisiones más rápidas y precisas.

- No se requiere de una implementación por parte de los desarrolladores para implementar esta función.

Desventajas

- No se almacena información de quien realizó la acción.
- Algunas de las versiones antiguas de bases de datos no soportan CDC.
- Requiere de configuración adicional en la base de datos origen.

Técnicas para Captura de Cambios de Datos

Existen dos categorías principales de técnicas de CDC: intrusivas y poco intrusivas.

Técnicas Intrusivas

Las técnicas intrusivas son aquellas que generan un posible impacto en el desempeño de las fuentes donde los datos son recuperados. Dentro de esta categoría se encuentran las siguientes:

- **Marcas de Tiempo:**

Esta técnica depende de que haya un campo de marca de tiempo disponible en la(s) tabla(s) fuente para identificar y extraer conjuntos de datos de cambios.

- **Disparadores**

Esta técnica no es implementada muy a menudo por seguridad y eficiencia. Esta opción es considerada la más intrusiva, pero tiene la ventaja de detectar todos los cambios en los datos, permitiendo cargas y captura de cambios en tiempo real.

Técnicas poco intrusivas

Las técnicas poco intrusivas mantienen un bajo impacto en el desempeño de las fuentes donde se recuperan los datos. Dentro de este tipo de técnica se encuentran los archivos logs.

- **Archivos Logs.**

Esta técnica detecta cualquier modificación en una base de datos incluyendo inserciones, actualizaciones y eliminaciones. Los eventos se capturan sin necesidad de realizar cambios a nivel de aplicación y sin tener que escanear las tablas a nivel operativo, lo cual agrega una carga de trabajo adicional y reduce el rendimiento de los sistemas de origen.

2.3.3.2 BROKER DE MENSAJERÍA

Un bróker de mensajería es un mecanismo mediador de comunicación orientado a mensajes entre aplicaciones, permitiendo la transformación y el enrutamiento de mensajes de un sistema a otro: productores y consumidores. La transformación por su lado, distribuye la información entre el o los destinatarios incluso si son de distintos lenguajes de programación. Por otro lado, el enrutamiento minimiza el grado de conocimiento que deben tener las aplicaciones para poder intercambiar mensajes. Entre los brokers de mensajería más populares actualmente se encuentran: **RabbitMQ**[2], **Apache Kafka**[3] y **SQS**[4].

Con el fin de mantener un almacenamiento fiable de mensajes y una entrega garantizada, los sistemas de mensajería utilizan un componente denominado cola de mensajes, que almacena y ordena los mensajes hasta que las aplicaciones consumidoras puedan procesarlos. En una cola de mensajes, los mensajes se almacenan en el orden exacto en el que se transmitieron y permanecen en la cola hasta que se confirma la recepción.

Existen dos tipos de patrones básicos de mensajería:

[2] RabbitMQ: https://www.rabbitmq.com/. Consultado el 26 de Julio 2021
[3] Apache Kafka: https://kafka.apache.org/. Consultado el 26 de Julio 2021
[4] SQS: https://aws.amazon.com/es/sqs/. Consultado el 26 de Julio 2021

- **Mensajería de punto a punto:** Cada mensaje de la cola se envía a un solo destinatario y se consume sólo una vez. La mensajería de punto a punto es llamada cuando un mensaje debe actuar sólo una vez.

- **Mensajería de publicación/suscripción:** el productor de cada mensaje lo publica en un tema y varios consumidores de mensajes se suscriben a temas sobre los cuales desean recibir mensajes.

2.3.3.3 PROCESO ETL (EXTRACCIÓN, TRANSFORMACIÓN Y CARGA)

Los procesos ETL pueden ser técnicas, herramientas y tecnologías que ayudan a transformar los datos de varias fuentes para extraer conocimiento de los mismos que sean veraces y útiles, y **cargarlos e**n otros sistemas con el fin de que puedan ser accesibles por los niveles de la organización que lo requieran.

Las fases de un proceso ETL se ilustran en la Figura 4:

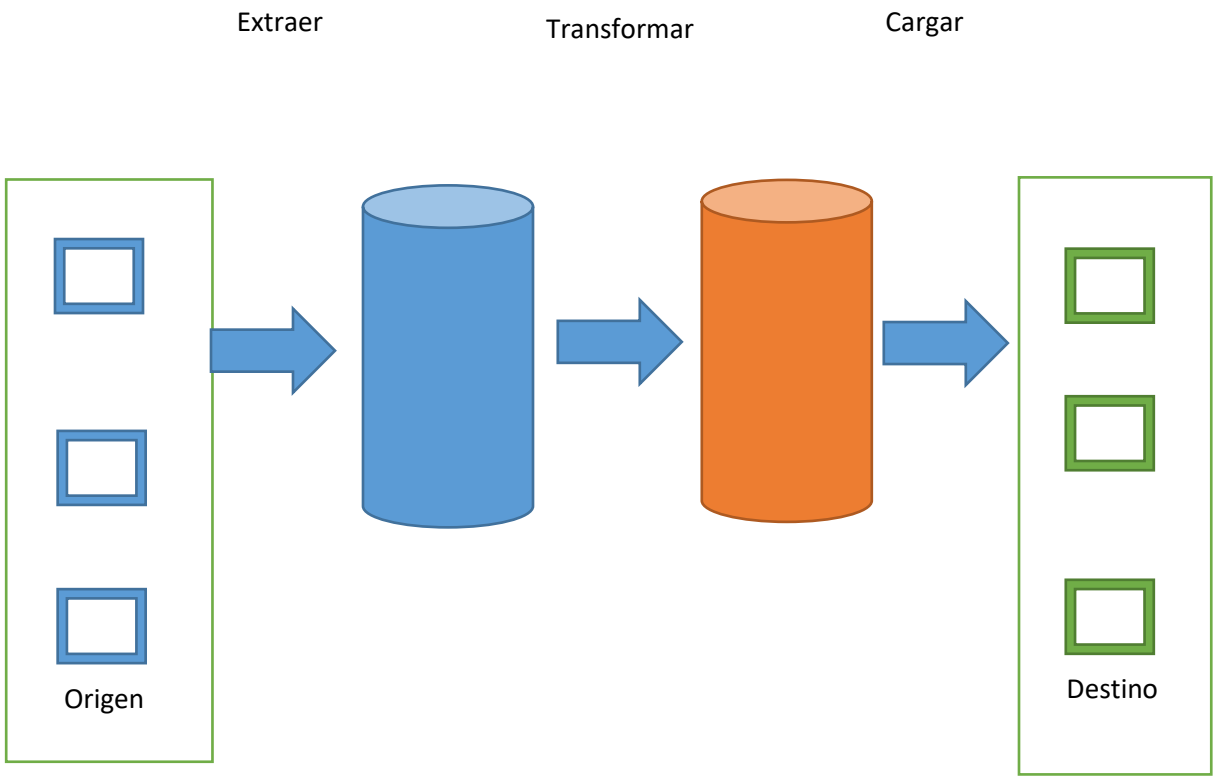

Figura 4 Proceso ETL

Fase de Extracción: Se encarga de la captura desde una o varias fuentes de datos para su uso en las próximas dos fases.

Fase de Transformación: Esta fase se encarga de aplicar las normas necesarias del negocio a los datos para que estos sean correctos, es decir se genera un proceso de limpieza.

Entre algunas de las actividades que permiten garantizar la calidad de los datos y su accesibilidad se encuentran:

- Eliminación de duplicados
- Normalización
- Verificación

Fase de Carga: La última fase tiene como meta cargar los datos extraídos y transformados a un nuevo destino.

Un proceso ETL se pueden implementar de dos formas distintas: programación propia de la ETL, o bien, uso de herramientas. La opción de desarrollar es muy flexible y las capacidades son casi ilimitadas, sin embargo, conlleva tiempos de desarrollo elevados y una depuración compleja en caso de que existan errores. Se emplean lenguajes de programación como Python o Java gracias a la gran cantidad de librerías que existen para el trabajo con volúmenes de datos. Por su parte, utilizar herramientas de terceros diseñadas para el manejo de una ETL tiene como ventaja poder realizar transformaciones a través de interfaces visuales y ágiles. Entre las herramientas más utilizadas se encuentran Pentaho[5], Talend [6], entre otras.

En resumen, la capa de ingestión permite que un conjunto de datos heterogéneos generados a partir de varias fuentes de datos puede combinarse y almacenarse en una plataforma analítica con el fin de minimizar los tiempos de respuesta en consultas y así aprovechar las ventajas que ofrece el procesamiento real. Estas actividades son la meta principal del siguiente componente en la arquitectura.

[5] Pentaho: https://www.hitachivantara.com/en-us/products/data-managemen analytics.html?source=pentaho-redirect**.** Consultado el 26 de Julio 2021
[6] Talend: https://www.talend.com/es/. Consultado el 26 de Julio 2021

2.3.4 ALMACENAMIENTO DE DATOS ANALÍTICOS

En esta capa se almacenan los grandes volúmenes de datos ya transformados y procesados de la capa de Ingestión para su posterior recuperación en herramientas de inteligencia de negocios. Los datos procesados en tiempo real se pueden almacenar en un solo lugar para poder realizar el análisis de todo el conjunto de datos. Para manejar adecuadamente las diferentes necesidades que surgen del procesamiento en tiempo real, es importante tener los sistemas adecuados para manejar el tipo de carga de trabajo y el patrón de acceso a los datos. Dependiendo de cómo se consuman los datos y del volumen y velocidad de los datos, es posible que se complemente la plataforma de datos con OLAP, OLTP, HTAP o sistemas de motor de búsqueda.

- **OLAP** acrónimo de procesamiento analítico en línea, es una solución utilizada en la inteligencia de negocios ya que permite realizar consultas de grandes cantidades de datos de manera ágil a través de estructuras multidimensionales llamadas cubos OLAP que contienen datos históricos.

- **OLTP** es un tipo de base de datos para procesamiento transaccional en línea para hacer frente a consultas relativamente simples y manejar bien tanto las operaciones de lectura como las de escritura. Los tipos que encajan en esta categoría son las bases de datos relacionales típicas (RDBMS) o bases de datos NoSQL como MongoDB o DynamoDB.

- **HTAP** procesamiento híbrido transaccional/analítico ofrece la capacidad de manejar las dos cargas de trabajo de tipo OLTP y OLAP.

- **MOTORES DE BÚSQUEDA** como ElasticSearch se integran con soluciones de tableros de control como Grafana o Kibana para análisis en tiempo real. Estas soluciones permiten realizar análisis de búsqueda específicos, lo que reduce significativamente la ejecución de latencia de las consultas.

Actualmente dentro de la institución financiera se manejan bases de datos relacionales para la representación de los datos en tablas, buscando ahora como parte de la arquitectura proponer el uso de almacén de datos no relacional que permita velocidad a la hora de las consultas para presentación en gráficas de inteligencia de negocios.

2.3.5 ANÁLISIS

El componente previo a la presentación de los resultados en informes o reportes para el monitoreo de los datos procesados es el Componente de Análisis. Este componente permite a través de métodos estadísticos y de aprendizaje automático crear conocimiento y hacer predicciones a partir de la información almacenada en tiempo real.

El análisis de los datos aporta valor en las organizaciones. Se pueden dividir las herramientas que pueden soportar este componente en cuatro categorías.

- **Análisis descriptivo:** Informa que está pasando, donde se pueden utilizar herramientas de reportes o visualización para ver el estado actual de la información en un determinado momento del tiempo.
- **Análisis de diagnóstico:** Son técnicas más avanzadas que permiten profundizar en los datos y explicar porque se originó cierta situación o las causas que provocaron determinada situación.
- **Análisis predictivo:** Este tipo de herramientas permiten a través de algoritmos avanzados predecir o generar un pronóstico sobre lo que sucederá en el futuro. En este tipo de análisis podemos encontrar técnicas como la inteligencia artificial y aprendizaje automático.
- **Análisis prescriptivo**: En este análisis se puede definir qué acciones tomar para lograr ciertos resultados utilizando técnicas muy sofisticadas de la inteligencia artificial.

En el sector financiero, los mecanismos de análisis en tiempo real aportan a la creación de productos más adaptables, inteligentes y eficientes mejorando así la experiencia de los clientes. Entre algunos de los ejemplos que se manejan actualmente dentro del mundo de la banca se encuentran control de fraude con inteligencia artificial, créditos automatizados, productos financieros online, etc. Es así que las tareas comunes que se producen dentro del ámbito financiero corresponden a análisis tanto descriptivo como predictivo permitiendo identificar mejor los comportamientos y necesidades de los socios y llevarlos a herramientas automatizadas.

2.3.6 VISUALIZACIÓN Y REPORTES

Los datos transformados en la capa previa (Análisis) son mostrados en forma de gráficos y valores resumidos para una mejor interpretación de los resultados. Estos informes son usados por las empresas para toma de decisiones basadas en datos. Entre los principales elementos que se manejan dentro de un proceso de visualización y reportes están:

2.3.6.1 INDICADORES CLAVES DE RENDIMIENTO (KPI)

Los indicadores de rendimiento permiten medir el desempeño de un proceso a través de métricas, utilizadas para cuantificar los objetivos planteados que reflejan el rendimiento de una organización, producto o una acción en concreto. Estos indicadores son llevados a cuadros de mando o tableros de control para aportar una visión general eficiente y eficaz del estado actual de la empresa.

2.3.6.2 TABLERO DE CONTROL

El tablero de control dentro de las organizaciones es considerado como una herramienta efectiva para medir la evolución, crecimiento, realización de metas, eficiencia y eficacia de las empresas; ayudando a la toma de decisiones oportunas que permitan llegar a los objetivos planteados. La productividad dentro de una organización radica en una mejor comprensión y capacidad de análisis de la información para una mejor toma de decisiones.

Algunas de las gráficas y técnicas más utilizadas para diseñar un buen tablero de control son:

- **Barras**

Es un tipo de representación en columnas que permite mostrar gráficamente una serie de valores o datos para su comparación y análisis.

- **Pie**

Este gráfico divide los datos en sectores, ilustrando cada proporción numérica con el fin de mostrar la composición como un todo.

- **Drill and down**

Es un tipo de gráfica que permite apreciar los datos a un mayor nivel de detalle (Ver figura 5). Permite entonces, a partir de un gráfico principal (como se encuentra en la imagen superior), poder interactuar con él, al seleccionar uno de sus elementos para poder obtener detalles de la información que representa a nivel de desglose.

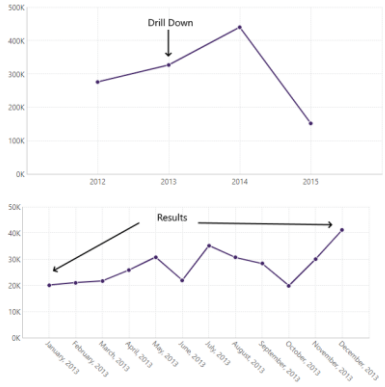

Figura 5 Representación Drill Down

- **Mapas de calor**

Los mapas de calor son elementos de comunicación que representan visualmente la información geográfica simplificada como se aprecia en la Figura 6.

Un mapa de calor es una representación gráfica, a través de colores, de distintos parámetros analizados en un territorio o en un espacio concreto. Los colores utilizados indican la mayor o menor presencia de condiciones reconocidas de impacto sobre el éxito o el fracaso de una determinada ubicación en relación con la variable requerida.

Figura 6 Representación mapa de calor

- **Consultas**

Query DSL (Lenguaje de dominio específico) permite realizar consultas a documentos indexados; siendo simples, flexibles y sencillos de leer al utilizar interfaces de tipo JSON. Las consultas se basan en dos tipos: "Leaf Query Clauses" y "Compound Query Clauses". Las primeras hacen referencia a un tipo que devuelve un valor específico solicitado. Las segundas por su lado permiten obtener información más compleja dado que manejan una combinación de las primeras.

A continuación, se presenta algunos beneficios que ofrece un procesamiento en tiempo real en el sector financiero lo que conlleva a uno de los objetivos del presente proyecto de tesis: diseñar una arquitectura simplificada que a través de microservicios genere un análisis de las transacciones de tarjetas de débito de una institución financiera como caso de estudio definida en la siguiente sección de este documento.

1. Reducción de costos: las tecnologías de big data reducen significativamente los costos de almacenamiento de grandes volúmenes de datos.

2. Mejorar la toma de decisiones: el uso del componente de ingestión permite a las empresas tomar decisiones en tiempo real.

3. Tendencias de los clientes: las empresas pueden comprender los requisitos del cliente analizando las compras anteriores y crear así nuevos productos o mejorar los mismos.

4. Información precisa: El procesamiento en tiempo real permite almacenar únicamente los datos relevantes es decir aquellos datos necesarios y útiles para la buena marcha del negocio.

CAPITULO 3 TRABAJOS RELACIONADOS

En este capítulo se analizan trabajos relacionados que soportan el diseño e implementación de microservicios para la generación de un modelo de análisis de transacciones masivas en una institución financiera. El análisis de la literatura involucró por tanto la búsqueda de trabajos que describan los siguientes temas i) el procesamiento de información en tiempo real, ii) el manejo de microservicios dentro de las organizaciones para gestionar grandes cantidades de datos, y iii) el uso de la inteligencia de negocios para integrar estas tecnologías aparentemente disjuntas en una arquitectura consolidada, tolerante a fallos que permita transformar los datos de transacciones financieras (ej. tarjetas de débito) en información que puede ayudar a la toma de decisiones operativas y gerenciales; aportando al valor del negocio. Actualmente dentro del análisis realizado de la literatura no se ha encontrado un trabajo similar que abarque el uso e integración de las acciones descritas necesarias para el desarrollo de una integración en tiempo real a través de microservicios desarrollados a medida para la comunicación de las tecnologías.

Iniciando con el procesamiento de información en tiempo real se han encontrado algunos trabajos que proponen el uso de sistemas fiables aplicados a varios entornos incluido el financiero. Estos sistemas son considerados fiables ya que permiten el procesamiento de grandes cantidades de datos a una alta velocidad, escalables, seguros y sobre todo tolerantes a fallos. Por ejemplo, (Garrido, 2017) propone una arquitectura Big data de comunicación de datos en tiempo real, a través del manejo de servicios Web para simulación de envío de flujo de datos y una interfaz Web para presentación de resultados. Su enfoque maneja tres bloques integrados para el correcto funcionamiento del sistema streaming. (Ichinose Ayae, Takefusa Atsuko, Nakada Hidemoto, Oguchi Masato., 2017) a su vez proponen un marco de tecnologías para la transferencia de información de múltiples cámaras de video en tiempo real, su enfoque presenta los parámetros eficientes de configuración para su arquitectura en cantidad de nodos y número de particiones necesarias cuando actúa en un clúster, concluyendo así que, a más nodos, particiones y replicas más alta disponibilidad y tolerancia a fallos tendrá el sistema. Por su parte (Fernández, 2017) presenta una propuesta de arquitectura para el procesamiento y análisis de flujos de datos para eventos generados por sensores y dispositivos en IoT (Internet de las cosas), es decir, dispositivos y objetos agrupados e interconectados a través de una red. En su estudio realiza además una comparativa de tecnologías actuales presentando de forma clara los resultados en rendimiento que ofrece Apache Kafka versus otros sistemas de mensajería tradicionales como RabbitMQ y ActiveMQ. En este mismo contexto sobre comparativas de rendimiento de los sistemas de mensajería en tiempo real (Jay Kreps,Neha Narkhede,Jun Rao, 2011) y (Javed, 2017) describen una visión general sobre Apache Kafka y demuestran a través de varios experimentos y escenarios en modo autónomo y distribuido la superioridad de esta herramienta frente a otras existentes, respecto a procesamiento y tolerancia a fallas mientras se ejecuta como un clúster en uno o más servidores y la configuración realizada en uno o varios productores y consumidores para mejorar el rendimiento al procesar los flujos de datos. Dentro del ámbito financiero en el artículo presentado por (Fikri Noussair, Rida Mohamed, Abghour Noureddine, Moussaid Khalid, El Omri Amina, 2019) se expone

un enfoque adaptivo y en tiempo real para solventar problemas de latencia en la integración de datos financieros y la heterogeneidad de sistemas que se mantienen en las organizaciones para producir datos. Los autores exponen el uso de la herramienta Apache Spark y el uso de ontologías hibridas para la conversión y transformación de las transacciones con el objetivo de obtener métricas financieras actualizadas. De la revisión de las propuestas descritas previamente, se ratifica que el uso de Apache Kafka (en el presente trabajo) como sistema de mensajería de tiempo real es una buena opción, considerando las evidencias que muestran la fiabilidad que brinda esta herramienta para el procesamiento de transacciones. Además, como se manifiesta en (Bucur Vlad, Stan Ovidiu, Miclea Liviu, 2020) Kafka es una herramienta poderosa especialmente en entornos donde la alta disponibilidad de los datos y la tolerancia a fallas es fundamental para el éxito de empresas financieras que requieran administrar el flujo constante de mensajes o transacciones que se producen.

El segundo tema de análisis involucró aportes sobre el uso de los microservicios. El manejo de microservicios hoy en día permite la creación de aplicaciones independientes y fáciles de escalar, las cuales no se encuentran comprometidas con un solo tipo de tecnología, ofreciendo facilidad de mantenimiento. Según (Smid Antonin, Wang Ruolin, Černý Tom, 2019) un buen diseño para la comunicación de datos a través de microservicios, reducirá la sobrecarga de las comunicaciones del sistema y mejorará el rendimiento en la transmisión de datos. La propuesta de (Benson, 2019) se puede considerar como un punto medio entre el manejo de tecnologías de tiempo real y el uso de microservicios. En dicha propuesta se describen las ventajas del uso de una arquitectura de microservicios en el manejo de datos en tiempo real a través del envío de datos aleatorios simulando identificadores de sensores. El enfoque actual para el desarrollo de software y aplicaciones distribuidas se dirigen al manejo de este tipo de mecanismo, especialmente cuando se busca facilidad en el escalamiento. De acuerdo al trabajo propuesto por (Ríos, 2020), una arquitectura de microservicios en los sistemas financieros ayuda a reducir las intermitencias y cuellos de botella que se generan en estos sistemas altamente transaccionales, generando beneficios de escalado, estabilidad y velocidad. Por su parte (Salvador, 2020), presenta un claro ejemplo del manejo de microservicios mediante una arquitectura basada en eventos con Apache Kafka (como caso de estudio) para un gestor de vuelos. En esta propuesta los microservicios son usados tanto para el frontend como el backend y un intermediario cuya función es realizar un túnel de comunicación. Para (Bucchiarone Antonio, Dragoni Nicola, Dustdar Schahram, Larsen Stephan, Mazzara Manuel, 2018) varias empresas del ámbito bancario están involucradas en una actualización de sus sistemas backend para mejorar su rendimiento y demostrar cómo la escalabilidad se ve afectada positivamente al re-implementar una arquitectura monolítica en microservicios.

Con respecto a la aplicación de sistemas con microservicios en casos de estudio como el campo financiero (donde se aplicará la propuesta actual), se encontraron algunos modelos de éxito, por ejemplo, “Danske Bank” el banco más grande de Dinamarca y una de las instituciones financieras líderes en el norte de Europa y el “Canadian Imperial Bank of Commerce”, han migrado su arquitectura monolítica a microservicios independientes conduciendo a una menor complejidad, mayor cohesión e integración simplificada y una capacidad notable de reutilización de código (Nicola Dragoni,Schahram Dustdar, Stephan T. Larsen,Manuel Mazzara, 2017).

En los casos analizados, el resultado de la implementación de microservicios han sido todos casos de éxito, puesto que permiten aprovechar las funcionalidades de integración con terceros, además de poder realizar mejoras rápidas y continuas de la funcionalidad. Estas propuestas muestran que la adopción de microservicios a este trabajo puede convertirse también en un caso de éxito.

En nuestra propuesta se integró un microservicio a través de una aplicación propia, desarrollada como componente intermedio y encargado de generar el proceso de limpieza y transformación de los datos. La integración del microservicio, además, pretende aprovechar el uso de una herramienta desarrollado a medida para que se pueda acoplar a diferentes datos de origen ofreciendo así un módulo que pueda aplicarse a usos futuros en otros entornos. El almacenamiento de los datos recopilados en el proceso de limpieza y transformación es cargado en una base de datos de tiempo real como solución para conjuntos grandes de datos que requieren de un rendimiento y velocidad mayor. (Bathla Gourav, Rani Rinkle, Aggarwal Himanshu, 2018) presentan un estudio comparativo de bases de datos NoSQL o en tiempo real a través del contraste entre varios parámetros o características ofreciendo una guía para la selección de la técnica que se adapte a nuestro trabajo.

La tercera temática de interés en este capítulo es el uso de la inteligencia de negocios. Algunos trabajos en la literatura como (Ouda, 2016) y (Mary Julieth Murillo Junco, Gustavo Cáceres Castellanos, 2013) plantean que "uno de los campos en los que la Inteligencia de Negocios es más usada por sus excelentes resultados es el de las finanzas, ya que permite la visualización, análisis, compresión y seguimiento de la información en tiempo real de manera sencilla y efectiva". (p. 123). Ahondando en esta línea, (Jebaraj Anand, Chaojie Yang, Koona Manoj, Ratnayake Dilanka, 2020) destacan los beneficios de los tableros de control como herramientas gráficas dentro del sector financiero permitiendo maximizar el beneficio del gran volumen de transacciones que se generan para proporcionar servicios más personalizados a los clientes y aumentar así la calidad de los servicios que se ofertan dentro las instituciones financieras. Para generar un diseño correcto de tableros de control es necesario antes definir los indicadores clave que se requiere en el manejo de la inteligencia de negocios de una empresa, es decir, medir a través de valores concretos obtenidos de la información que se tiene, si los objetivos están cumpliéndose o no. Dentro de los indicadores claves comunes de una organización financiera según (Freire, 2015) se encuentran la cantidad y monto total generado en créditos, inversiones o cuentas aperturadas. Para la perspectiva de análisis de este trabajo se propone la creación de nuevos indicadores que generen una visión del uso de la tarjeta de débito, permitiendo analizar el comportamiento de los clientes, comercios y lugares donde su uso es común, para así lograr obtener una ventaja competitiva frente a otras organizaciones y potenciando además el mercadeo de este servicio.

En (Filip Mezera, Jiří Křupka, 2017) se exponen algunas ventajas del uso de la inteligencia de negocios en el sector financiero. Los autores mencionan que "La inteligencia de negocios apoya a la toma de decisiones y evitan decisiones apresuradas e inexactas" (p. 10). En dicho trabajo se muestra a través de pruebas realizadas, las ganancias de clientes que solían ser poco rentables o incluso generaban pérdidas para la empresa a través del uso de métodos de modelado estadístico. En algunas empresas del sector financiero ya utilizan soluciones de inteligencia de negocios para

disponer de una visión completa del funcionamiento de su organización. Según (Basantes Espinoza Gabriela Paola, López Galarza Daniel Eduardo, 2012) exponen el caso de éxito del Banco de Guayaquil y donde a través de su plataforma tecnológicas FINANWARE visualizan la información a nivel gerencial para toma de decisiones. (Ríos Carrión Pamela Mishelle, Bermeo Pazmiño Katina Vanessa, Narváez Zurita Cecilia Ivonne, 2021) por su parte exponen en su artículo el uso del análisis de sentimientos y el comportamiento de los clientes a través de sus preferencias, mediante la integración de datos de medios sociales basándose en su rentabilidad para optimizar las estrategias de una entidad financiera.

Basados en la evidencia de la incorporación de la inteligencia de negocios en el ámbito financiero, se ha diseñado un tablero control que permita a través de su representación gráfica dar un enfoque actual del manejo de las tarjetas de débito de la institución financiera, como sectores, comercios, montos, plataformas y países que utilizan este medio de pago. Esto permite que los directivos hagan preguntas y puedan obtener respuestas rápidamente con el fin de tomar decisiones más acertadas.

Por tanto, las características que ofrecen las acciones descritas para la integración de los datos en tiempo real han sido aprovechadas para generar el diseño e implementación de la arquitectura propuesta en este trabajo de tesis.

CAPITULO 4 ARQUITECTURA PROPUESTA

En el capítulo 2 se describieron los componentes de una arquitectura genérica para soportar el procesamiento en tiempo real de datos provenientes de las transacciones de tarjetas de débito. Tomando como punto de partida esta arquitectura se diseñó una propuesta que contiene tres grandes componentes y que permite a través de sus procesos simplificar el enfoque propuesto en este trabajo (ver figura 7).

El componente 1 y 2 de la arquitectura general (ver figura 3) se encuentran ahora en el Componente 1 Captura de Datos Modificados; el proceso de Ingestión o Procesamiento en tiempo real ahora denominado Núcleo, es la base de la arquitectura y donde se ejecutan tanto el proceso ETL como el almacenamiento de datos. El componente de Análisis para el propósito de esta tesis ha sido eliminado dado que no se aplicará dentro de este estudio predicciones de la información o algoritmos automatizados que permitan descubrir eventos futuros bajo datos históricos. En lugar de ello, la arquitectura pretende mostrar los flujos de datos en tiempo real hacia tableros de control, que reflejen los resultados de los indicadores (explicados más adelante en el Capítulo 5), visualizando así a través de gráficas intuitivas el desempeño del uso de tarjetas de débito como forma de pago seleccionada en una institución financiera.

Para la selección de las herramientas que permitan instanciar los componentes de la arquitectura propuesta es importante aclarar las restricciones que se tienen dentro del caso de estudio.

- El uso de mecanismos de código abierto es clave dentro de la organización financiera, dado que ayuda a la generación de desarrollos rápidos al mantener la posibilidad de modificar y tener acceso al código fuente pudiendo generar sistemas a medida y con costos bajos de implementación.

- Otra de las restricciones para el desarrollo de la arquitectura es que el tipo de base de datos utilizada para almacenar las transacciones de tarjetas de débito es SQL Server.

- Es preciso recalcar que al ser una institución que genera miles de transacciones se ha solicitado que los conjuntos de datos generados y analizados durante el estudio actual sean confidenciales y no pueden tener un estado público, por lo que se cuenta con un ambiente de test para pruebas del diseño propuesto.

ARQUITECTURA PROPUESTA

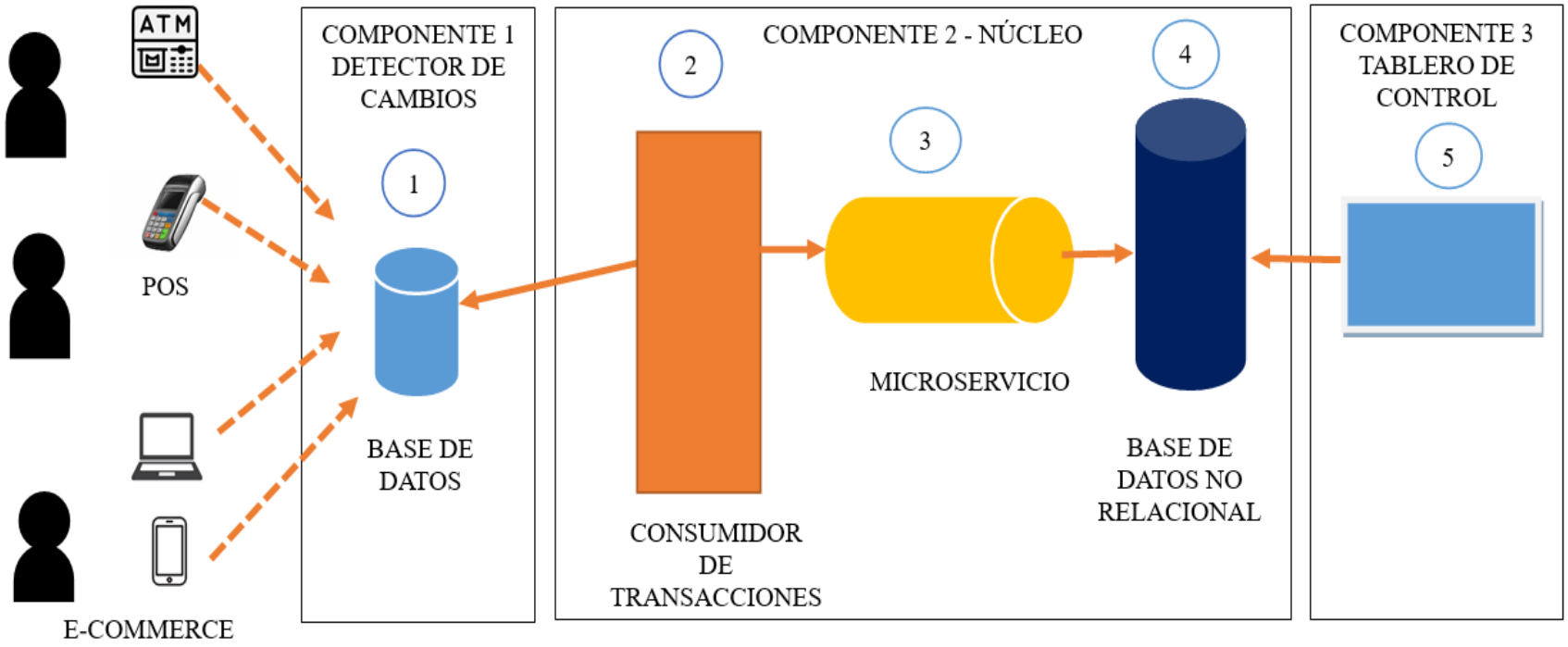

Figura 7 Arquitectura Propuesta

A continuación, se presenta para cada componente de la arquitectura una breve introducción de su funcionamiento. Luego se efectúa un análisis de las herramientas que pueden soportar la funcionalidad requerida y se describe la herramienta final seleccionada. Por cada componente además se explica la forma en cómo interactúa con los demás elementos de la arquitectura.

4.1 COMPONENTE 1 – DETECTOR DE CAMBIOS

INTRODUCCIÓN

Este componente dentro de la arquitectura planteada tiene la funcionalidad de determinar y generar un seguimiento de los datos que han cambiado sobre un almacén de datos. Actualmente dentro de la institución financiera seleccionada para este proyecto, el almacenamiento de las transacciones del método de pago con tarjeta de débito efectuadas sobre ATM (cajeros automáticos), POS (Sistemas de punto de venta) y comercio electrónico se encuentra bajo una base de datos relacional. Este escenario es algo que ocurre frecuentemente en otras instituciones financieras.

El propósito de este mecanismo es necesario dado el volumen en tiempo real de información que se produce en este tipo de organizaciones y la necesidad de entrega y sincronización de estos datos hacia almacenes de datos que permitan una toma de decisiones rápida y oportuna hacia las líneas de negocio.

Lo expuesto se detalla a continuación con un análisis de tecnologías actuales pertenecientes a este enfoque.

4.1.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA CAPTURA DE CAMBIOS DE DATOS

En el Capítulo 2 se describieron los tipos de técnicas que existen actualmente para manejo de captura de cambios de datos (CDC). De las técnicas CDC analizadas la forma más avanzada y menos intrusiva es usar una solución basada en archivos log; en los cuales puede detectarse cada operación de inserción, actualización y eliminación manejada en una base de datos y en tiempo real. Esta técnica es típicamente utilizada en conjunto con Sistemas Gestores de Bases de Datos (SGBD). (Díaz de la Paz, Lisandra & García Mendoza, Juan Luis, 2015)

De acuerdo a la revisión de las herramientas que existen en el mercado hoy en día de código abierto que ofrezcan funcionalidad para captura de cambios de una base de datos bajo archivos logs, se seleccionaron dos opciones que cumplen con la restricción sobre la base de datos a utilizar en el caso de estudio de la institución financiera.

Las dos herramientas fueron seleccionadas luego de una revisión de la literatura [1,20,33], donde se describen ventajas y desventajas sobre todo cuando se busca un procesamiento en tiempo real y un alto índice de rendimiento en las transacciones generadas.

A continuación, se presentan las características principales en una tabla comparativa (Tabla 1) que permite cubrir los puntos a considerar para elegir el proceso que se adapte a la captura de cambios producidos CDC para el caso de estudio.

- **Base de datos que soporta:** Esta característica define el tipo de base de datos que el mecanismo puede utilizar para capturar los eventos generados.

- **Tolerancia a fallos:** Permite que el funcionamiento del sistema continúe, así se genere algún fallo en el mismo como daños en el hardware, cableado, o caídas del sistema.

- **Arquitectura activa-pasiva:** Ayuda a generar alta disponibilidad al manejar instancias activas y pasivas que, ante cualquier fallo, no permitan que se detenga el procesamiento y captura de las transacciones, enviando las nuevas peticiones hacía una de las instancias pasivas y convirtiéndola en activa en caso de que haya dejado de funcionar la instancia inicialmente configurada.

- **Alta disponibilidad:** Permite garantizar la continuidad del funcionamiento del servicio CDC incluso en situaciones de deficiencias del sistema.

- **Comportamiento reactivo:** Permite disponer de los cambios en las fuentes de datos en el momento que se producen, proporcionando en tiempo real esta información.

	Debezium	**Dblog**
Descripción	Plataforma de streaming de datos de baja latencia para capturar cambios en los datos.	Plataforma desarrollada por Netflix
Base de datos que soporta	Mysql, PostreSql, Sql Server, Oracle, Cassandra y Db2	MySQL, PostgreSQL, MariaDB, Sql Server
Tolerante a fallos	Si	Si
Arquitectura Activa-Pasiva	Mediante zookeper	Mediante zookeper
Alta disponibilidad	Si	Si

Comportamiento reactivo	Si	Si

Tabla 1 Tabla comparativa de herramientas CDC basados en archivos logs

4.1.2 HERRAMIENTA SELECCIONADA

La herramienta Debezium[7] fue seleccionada para soportar la captura de cambios de datos, dada su facilidad de configuración y simplicidad a la hora de utilizarla; además de poseer un conector directo para SQL Server necesario para la conexión a los registros de las transacciones de tarjetas de débito requeridas. Provee también una baja latencia en la propagación de los cambios evitando así afectar al rendimiento de la base de datos que se monitorea.

DEBEZIUM

Debezium es una tecnología de código abierto, respaldada por Red Hat como parte de Red Hat Integration , que permite capturar los cambios a nivel de fila de la base de datos como eventos y publicarlos en temas/mensajes usando un bróker de mensajería Debezium que es altamente escalable y resistente a fallas, además posee conectores CDC para múltiples bases de datos, incluidas Postgres, Mysql, SqlServer y MongoDB.

Su enfoque se basa en la captura de cambios producidos basado en registros, lo que significa que funciona reaccionando a los cambios en la base de datos a través de logs transaccionales en el momento que se producen, permitiendo implementar soluciones orientadas a proporcionar valor en tiempo real.

Funcionamiento

La arquitectura de Debezium se centra en el concepto de conectores, lo cual permite realizar la comunicación entre las fuentes de datos y otras tecnologías como por ejemplo los brókers de mensajería. La primera vez que el conector Debezium se conecta a la base de datos, captura una instantánea (conjunto de datos necesarios para la sincronización de datos inicial de la base de datos) de los esquemas para todas las tablas que se encuentra configurado el conector para empezar a realizar la captura de cambios y convertirlos en flujos de eventos. Una vez completada la sincronización inicial, el conector comienza a capturar los cambios realizados a nivel de fila que se producen. El

[7] Debezium: Permite detectar y transmitir como flujos de eventos, los cambios que ocurren en una base de datos de manera inmmediata. https://debezium.io/

conector de Debezium es tolerante a fallas permitiendo así que sí el conector falla por cualquier motivo, después del reinicio del mismo, el conector reanuda la lectura de las tablas desde el último punto que leyó, esto gracias al manejo de la posición de los eventos en el registro de la base de datos (LSN / Número de secuencia del registro).

A partir de esta sección se desglosará el funcionamiento de cada uno de los componentes presentados en la Figura 7. Por ejemplo, en la Figura 8 se presenta como primer enfoque el flujo de la información desde las fuentes de datos hacia un consumidor de transacciones pertenecientes al punto 1 y 2 de la Figura 7.

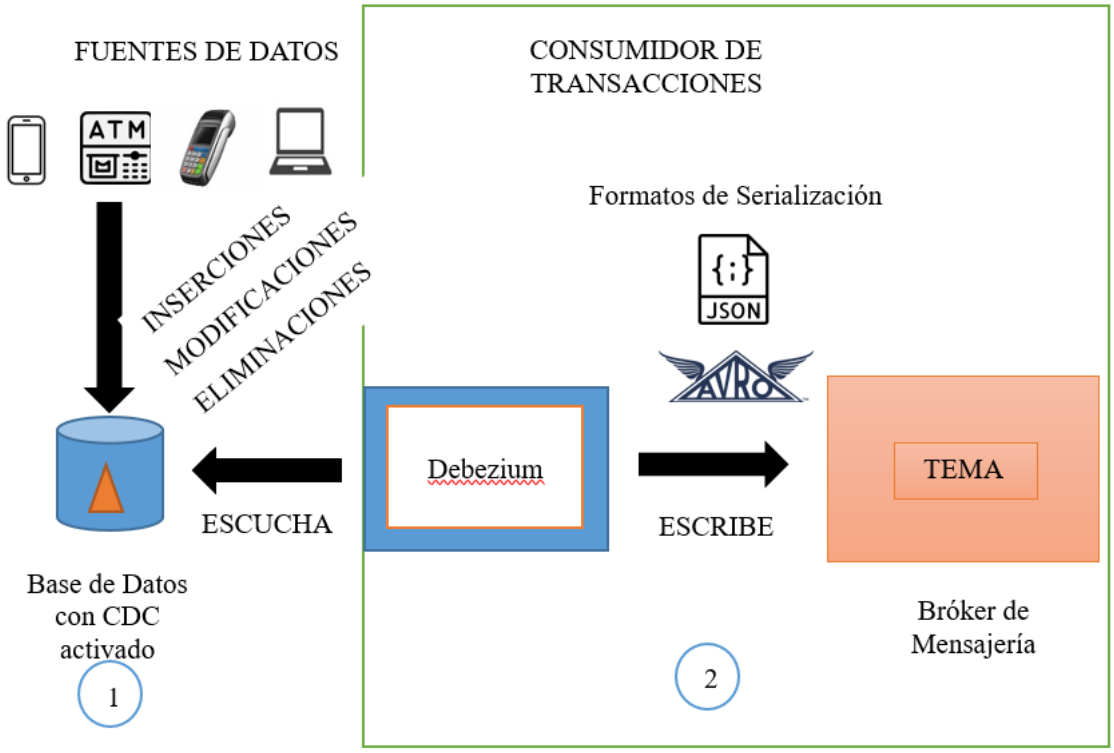

Figura 7 Arquitectura Debezium

Los datos enviados desde las fuentes de datos son capturados por el proceso CDC activado en la tabla donde se almacenan los registros de las tarjetas de débito. Debezium a través del conector para el tipo de base de datos comienza a monitorear todas las transacciones de la base de datos y convierte

el evento de cambio generado en el registro en un formato de serialización como por ejemplo (JSON o Apache Avro[8]).

Los eventos de cambio producidos se organizan de forma que, por cada tabla del modelo de datos, exista un tema asociado a la tabla. Por tanto, todos los eventos de cambio generados en una determinada tabla se envían a su tema asociado. Los temas son creados por defecto y siguen el siguiente patrón **dbserver.database.table**.

Una vez capturados los datos de las fuentes de datos y convertidos en flujos de eventos de cambio, Debezium debe apoyarse con otras tecnologías que permitan transmitir y procesar los datos en información que aporte valor para la inteligencia de negocios. Para este propósito se ha definido el siguiente componente bajo el nombre de "NÚCLEO" para la arquitectura propuesta.

4.2 COMPONENTE 2 – NÚCLEO

INTRODUCCIÓN

El segundo componente de la arquitectura propuesta es denominado "NÚCLEO". Como su nombre lo indica es considerado el mecanismo principal, permitiendo actuar como un puente entre los datos relacionales y su envío hacia una base de datos no relacional. Este componente realizará el respectivo proceso ETL [9], es decir, la captura de los datos generados en la base origen (EXTRACCIÓN), el procesamiento, limpieza y transformación de los datos (TRANSFORMACIÓN) y el envío hacía una base de datos destino (CARGA).

Entonces para llevar estas transacciones de tarjetas de débito en el sector financiero de una base datos origen a un destino en tiempo real, se requiere de una tecnología que actúe a nivel intermedio como un puente de comunicación. Estos mecanismos se denominan brókeres de mensajería.

4.2.1 BROKERS DE MENSAJERÍA

INTRODUCCIÓN

Estas tecnologías están dedicadas a procesar e intercambiar mensajes de datos entre aplicaciones, haciendo de mediador entre las mismas a una gran velocidad.

Para este caso de estudio se ha seleccionado el bróker de mensajería tipo publicador/subscriptor o llamado también productor/consumidor (descrito en el capítulo 2), permitiendo transmitir los eventos de forma asíncrona, es decir, sin la necesidad de saber dónde están

[8] Apache Avro: Avro es un marco para serialización de datos en formato binario y llamadas de procedimiento remoto orientado a filas desarrollado dentro del proyecto Hadoop de Apache.
[9] ETL (Extracción, Transformación y Carga)

los receptores de la información, si están activos o no, o cuántos de ellos hay; esto permite agilizar el proceso de desacoplamiento de los mensajes. A continuación, se detalla un análisis de las tecnologías actuales que soportan esta funcionalidad.

4.2.1.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA HERRAMIENTAS BROKER DE MENSAJERÍA

En la literatura [13,27,28] por ejemplo, existen propuestas que analizan diferentes herramientas de código abierto para procesar e intercambiar mensajes a nivel publicador/subscriptor. Basados en estos trabajos se ha efectuado un análisis de dos herramientas populares actualmente y donde a través de la tabla 2 se exploran sus características y diferencias más importantes.

Las características de las tecnologías que han sido seleccionadas para cubrir los puntos a considerar para elegir el bróker de mensajería son:

- **Enfoque:** Tipo de modelo utilizado para aprovechar el procesamiento de los mensajes.
  - **Modelo push:** Distribuye los mensajes de forma individual y rápida, para garantizar que el trabajo se paralelice de manera uniforme y que los mensajes se procesen aproximadamente en el orden en que llegaron a la cola.
  - **Modelo pull:** Este modelo permite a los usuarios aprovechar el procesamiento por lotes de mensajes para una entrega de mensajes eficaz y un mayor rendimiento.
- **Volumen:** Cantidad de eventos en el procesamiento**.**
- **Baja Latencia:** Característica que reduce el retraso en la transmisión de los datos que se produce desde que se realiza la emisión hasta que se reciben los mensajes.
- **Escalamiento:** Capacidad de adaptarse a las necesidades de rendimiento a medida que el número de usuarios crece o las transacciones aumentan.
- **Replicación de eventos:** Este mecanismo consiste en realizar una copia de una partición disponible en otro nodo. Esto permite que sea tolerante a fallos y evita pérdida de datos.
- **Comunicación Asíncrona:** Permite que los productores y consumidores no interactúen entre sí sino directamente con la cola de mensajes evitando tiempos de espera.
- **Grupo de consumidores:** Varios tipos de consumidores pueden suscribirse a muchos mensajes.
- **Alta disponibilidad:** Permite la coordinación de los nodos de un clúster en caso de fallos del sistema.

- **Atomicidad:** Garantiza que los mensajes sean entregados.
- **Retención de mensajes:** Permite mantener los mensajes para evitar perdida de datos importantes.

	Apache Kafka	**RabbitMQ**
Descripción	Un sistema de mensajería eficiente, tolerante a fallas y escalable. Kafka puede admitir una gran cantidad de publicadores y suscriptores y almacenar grandes cantidades de datos.	Es una cola de mensajes de código abierto escrita en Erlang. Permite varios protocolos como AMQP, XMPP, SMTP, STOMP. Posee buen soporte para enrutamiento, equilibrio de carga o persistencia de datos.
Año de lanzamiento	2011	2007
Escrito en	Java y Scala	Erlang
Enfoque	Modelo pull (Productor tonto, consumidor inteligente)	Modelo push (Productor inteligente, consumidor tonto)
Volumen	1 millón de mensajes / seg	4K-10K mensajes / seg
Baja Latencia	Alto rendimiento	No soporta
Licencia	Open Source: Licencia Apache 2.0	Open Source: Licencia Pública Mozilla
Lectura de datos	Múltiples conectores para conectarse a casi cualquier fuente de datos	Avro, Binary, CSV, MySQL, Postgres, JSON, Log, Protobuf, SDC Record, Text, XML

Escritura de datos	Múltiples conectores para almacenar los datos	HDFS [10]
Escalamiento	Horizontal	Vertical
Replicación de eventos	Si	Si mediante configuración
Comunicación Asíncrona	Si	Si
Grupos de Consumidores	Si	No
Alta disponibilidad	A través de Apache Zookeeper que permite gestionar el clúster	A través de la agrupación en clúster y colas de alta disponibilidad propias del mecanismo.
Atomicidad	Si	No
Retención de Mensajes	Conserva los mensajes hasta un tiempo de espera configurado por el tema denominado periodo de retención.	Elimina los mensajes tan pronto sean consumidos con éxito por los subscriptores.

Tabla 2 Tabla comparativa Apache Kafka vs RabbitMQ

[10] HDFS (Hadoop Distributed File System) es el componente principal del ecosistema Hadoop. Permite almacenar data sets masivos con tipos de datos estructurados, semi-estructurados y no estructurados como imágenes, vídeo, datos de sensores, etc.

4.2.1.2 HERRAMIENTA SELECCIONADA

La herramienta seleccionada como bróker de mensajería para el proceso de tarjetas de débito de la institución financiera es Apache Kafka. De acuerdo a la tabla comparativa (Tabla 2) ofrece un rendimiento y escalabilidad mayor que RabbitMQ por lo que es ampliamente utilizado cuando se requiere de un procesamiento de datos masivos (Big data) con un alto tráfico de información en tiempo real.

APACHE KAFKA

Es una aplicación de código abierto de Apache Software Fundation, multiplataforma especializado en el procesamiento de flujo de datos bajo el modelo publicador/subscriptor. Desarrollado por LinkedIn en el año 2011 como un sistema de mensajería altamente escalable y mantenido por la empresa Confluent, empresa que se encuentra bajo la administración de Apache.

Dentro de este marco se manejan varios componentes y términos que permiten detallar el funcionamiento de Apache Kafka y se describen a continuación:

- **Zookeeper**

Es primordial en Apache Kafka dado que actúa como un servicio de coordinación de alto rendimiento para aplicaciones distribuidas; se utiliza dentro de Kafka para gestionar los nodos. A su vez entre sus funcionalidades se encuentra la elección de los líderes de las particiones, manejar e informar a Kafka cuando se crea o elimina un nodo o un tema.

- **Topic (Temas)**

Se definen como las categorías en las cuales se clasifican los mensajes enviados a Apache Kafka.

- **Publicador/Productor**

Proceso que se encargan de publicar los mensajes en los temas/topics.

- **Subscriptor/Consumidor**

Procesos que se encargan de obtener/leer los mensajes de los temas/topics.

- **Broker (nodos) o Agentes**

El broker/agente corresponde a cada uno de los nodos que componen la topología de Apache Kafka. Se encuentran en comunicación frecuente con Zookeeper para manejar la partición líder o la partición replica de un tema.

- **Partición**

Una partición es una secuencia ordenada de mensajes consumida por un único consumidor.

- **Offset**

En un indicador que permite a cada consumidor saber cuál es el último elemento que ha leído. De manera que si se cae el sistema no se pierdan los datos, dado que al retornar el sistema comienza a procesar desde el último offset. Se utiliza un secuencial definido por Kafka.

Dentro de Apache Kafka se puede manejar dos tipos de configuración que son descritas a continuación:

- **Modo Standalone**

Es útil para realizar pruebas, desarrollos básicos y casos de uso con productores y consumidores que no necesitan ser distribuidos siguiendo un enfoque de ETL (Extracción, Transformación y Carga). En este caso la funcionalidad de Zookeper no es necesaria dado que se requiere de un único nodo para su funcionamiento.

- **Modo Distribuido**

Es escalable y tolerante a fallos dado que se ejecuta múltiples procesos en la misma máquina o distribuido en varias máquinas llamado clúster.

Funcionamiento Apache Kafka en modo standalone

Apache Kafka funciona como un sistema de mensajería como se muestra en la Figura 9. El productor envía un mensaje a un nodo Kafka (broker) donde es almacenado en un tema (topic) para que otra aplicación que es la consumidora sea la que lea y escuche estos mensajes. Dentro de la arquitectura propuesta, el productor hace referencia a Debezium que captura los cambios realizados en la tabla de tarjetas de débito y envía registros generados hacia el microservicio que actúa como consumidor.

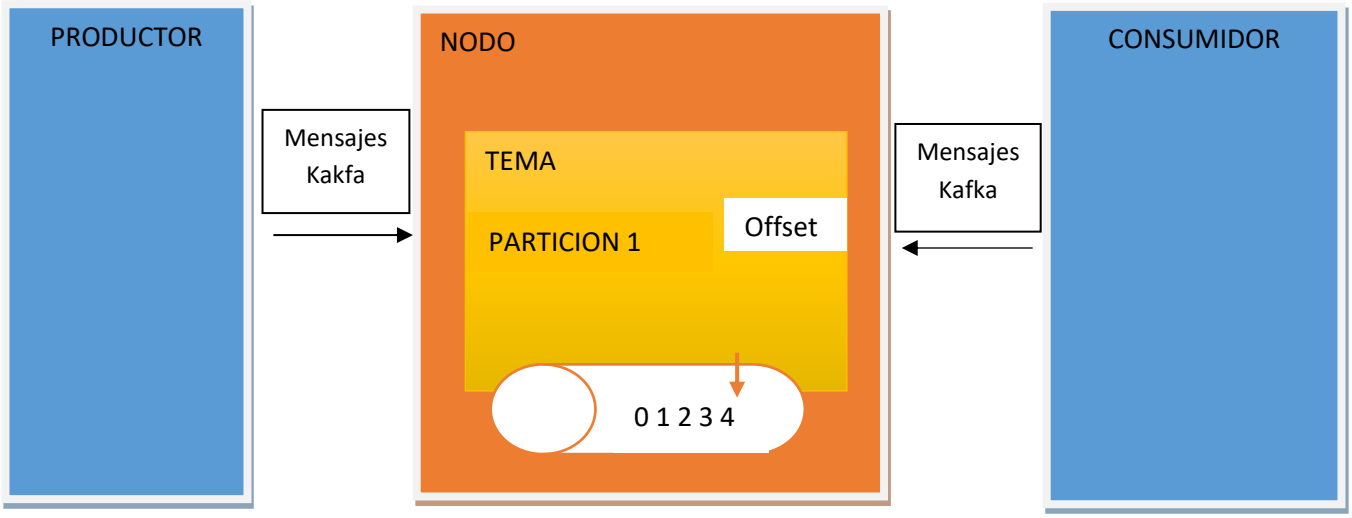

Figura 8 Funcionamiento General Apache Kafka

Funcionamiento Apache Kafka en modo distribuido

Este modelo permite gestionar la arquitectura de Apache Kafka a través de la escalabilidad y la tolerancia a fallos que ofrece. Si bien para las pruebas del caso de estudio del presente proyecto de tesis se utilizara Apache Kafka en modo standalone para demostrar el funcionamiento de la arquitectura propuesta, es importante presentar los conceptos necesarios que se describen a continuación para un entorno distribuido en caso de trabajos futuros.

- **Grupo de Consumidores**

Cada consumidor de un grupo de consumidores lee los mensajes de una o más particiones. Debido a esto, Apache Kafka permite aumentar el rendimiento general de la lectura de datos. En el ejemplo de la Figura 10, hay dos grupos de consumidores que leen mensajes de particiones del tema. El primer grupo de consumidores tiene dos consumidores y cada consumidor lee desde dos particiones. Y en el segundo grupo de consumidores, hay 4 consumidores y cada consumidor lee desde particiones separadas esto permite balancear la carga de los datos de manera uniforme, además si se agregaría un nuevo consumidor al grupo el mismo se encontraría inactivo y será activado automáticamente en caso de falla de algún consumidor del grupo. El proceso de hacer esto se llama reequilibrio del consumidor.

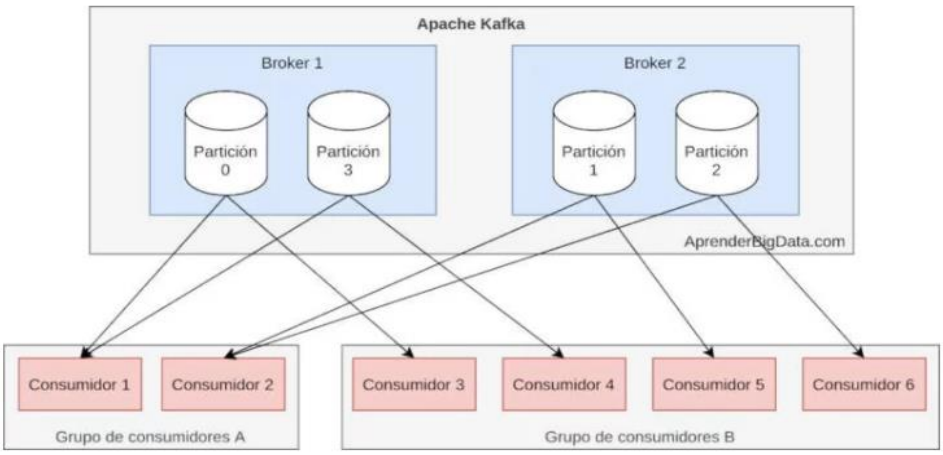

Figura 9 Grupo de consumidores

[Disponible en: https://aprenderbigdata.com/introduccion-apache-kafka/]

- **Replicación de eventos**

Uno de los puntos clave que ofrece esta tecnología es la escalabilidad horizontal. Un tema se divide en particiones, que permiten a Kafka poder distribuir los datos entre los nodos que se encuentran conectados. La replicación permite realizar una copia de la partición en otro bróker como indica la Figura 11, donde en el nodo 1 mantiene una réplica del tema B que se encuentra como líder en el nodo 3. Cuando existe varias réplicas disponibles, una de ellas es elegida como líder, y el resto son seguidores. Estos seguidores que están sincronizados con el líder se marcan con el mecanismo ISR (In Sync Replica). Este mecanismo permite que, si el líder se cae, la réplica pueda seguir entregando los datos sin problema convirtiéndose en el nuevo líder.

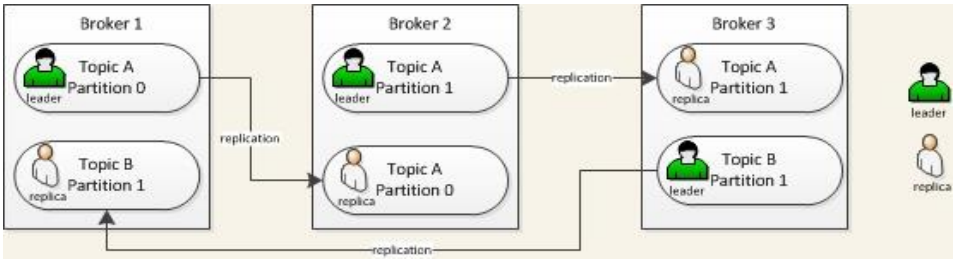

Figura 10 Replicación de eventos en Apache Kafka

[Disponible en: https://iteritory.com/beginners-guide-apache-kafka-basic-architecture-components-concepts/]

4.2.2 MICROSERVICIO

Para el desarrollo de esta arquitectura se busca que la comunicación sea rápida, segura y fiable con una alta disponibilidad. En los últimos años el manejo de las aplicaciones desarrolladas por las empresas se ha volcado considerablemente al uso de arquitecturas basadas en microservicios.

Este enfoque se basa en la creación de programas pequeños e independientes entre sí que funcionan en conjunto para llevar a cabo las mismas tareas que haría un sistema monolítico (donde toda la implementación estará contenida en una sola aplicación). Este tipo de tecnologías valoran el nivel de sencillez llevando así a un elemento fundamental dentro del desarrollo de software como es la optimización.

Es por esta razón que se procedió a crear una aplicación Java que contiene las librerías necesarias para actuar como microservicio entre la fuente de datos origen hacia un destino a través de Apache Kafka que actúa como solución para poder compartir los datos en tiempo real.

A continuación, en la Figura 12, se presenta el funcionamiento del componente “Núcleo” compuesto a su vez por los puntos 2, 3 y 4 de la Figura 7, detallando la respectiva comunicación de estos elementos para la arquitectura propuesta.

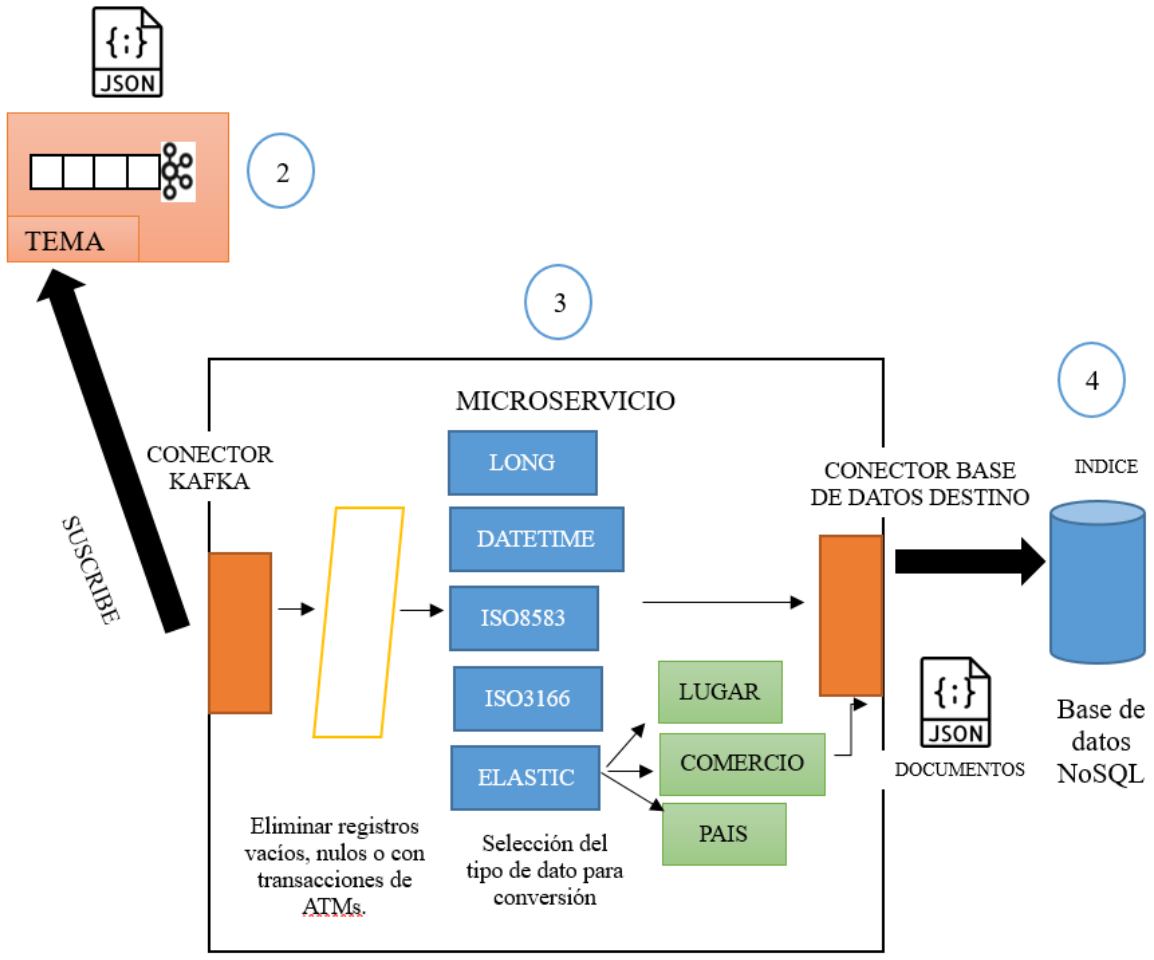

Figura 11 Funcionamiento del microservicio

El componente "Núcleo" inicia suscribiéndose al tema donde se encuentran las transacciones de las tarjetas de débito escuchadas en el proceso CDC. El conector Kafka que se encarga a su vez de deserializar los datos enviados en formato JSON y se encuentra dentro del microservicio desarrollado, comienza a escuchar los eventos de cambio y por cada uno genera el proceso ETL. Dentro del proceso ETL se realiza las siguientes acciones: 1) genera una limpieza de la información, donde se eliminan registros vacíos, nulos y transacciones generadas en cajeros automáticos (ATMs). 2) selecciona un tipo de datos y procede a la conversión. Por ejemplo: los datos de tipo long que se refieren a las horas se convierten en formato de cadena hh: mm: ss, los campos datetime en formato cadena dd / MM /

aaaa hh: mm: ss. Además, el campo ISO8583, que contiene la trama completa de una transacción de tarjeta de débito es decodificada a través de una biblioteca implementada, lo que permite extraer campos que se agregarán al registro final de la transacción. También, se realiza la respectiva transformación de códigos de país manejados bajo la norma ISO 3166 al nombre del país, estado, ciudad de donde se originó la compra con tarjeta de débito. Finalmente, el otro tipo de conversión es tipo elastic, que genera una conversión de los códigos a su respectiva descripción a través de potentes búsquedas de texto, que ofrece la base de datos NoSQL donde se registran los índices de lugares, comercios y países. 3) se carga esta información ya una vez realizada la limpieza y transformación hacía el almacén destino a través de un conector añadido al microservicio, encargado de enviar los documentos JSON generados con el registro de datos final hacía un índice creado para el almacenamiento en una base de datos NoSQL.

A continuación, se describe el funcionamiento de las normas ISO utilizadas dentro del desarrollo del microservicio.

4.2.2.1 NORMAS ISO

Siempre que usamos una tarjeta de crédito / débito / ATM, los datos pasan de un sistema a otro, viajando así entre varios sistemas. Por ejemplo, una compra realizada en una tienda puede viajar desde la terminal del comercio, a través de una red o redes de adquirentes, hasta el banco emisor donde se encuentra la cuenta del titular. En el mundo financiero para el intercambio de datos se usa los estándares establecidos por ISO (Organización Internacional de Normalización) una federación mundial de organismos nacionales de normalización. Dentro del presente proyecto de tesis se utiliza dos tipos de estándar tanto el **ISO8583** para el manejo interno de las transacciones de las tarjetas de débito y por otro lado el **ISO3166** para obtener los nombres de países, estados y ciudades a través del mapeo de los códigos internacionales.

ISO 8583

ISO 8583**,** Estándar para Transacciones Financieras con Mensajes originados en una tarjeta. Es utilizado en sistemas que intercambian transacciones electrónicas realizadas por poseedores de tarjetas de crédito/débito.

Funcionamiento

ISO 8583 define un formato de mensaje y un flujo de comunicación para que diferentes sistemas puedan intercambiar transacciones como un cajero automático ATM o POS (Puntos de venta), es decir en ATM usan este formato en algunos puntos de la cadena de comunicación, así como también en transacciones que realiza un cliente que usa una tarjeta para hacer un pago en un local.

Las redes MasterCard y Visa basan sus transacciones en el estándar ISO 8583, así como en otras instituciones y redes. En la figura 13 se puede evidenciar el uso extensivo de este formato a

través del flujo entre el usuario y los distintos sistemas en el manejo de transacciones de débito y crédito.

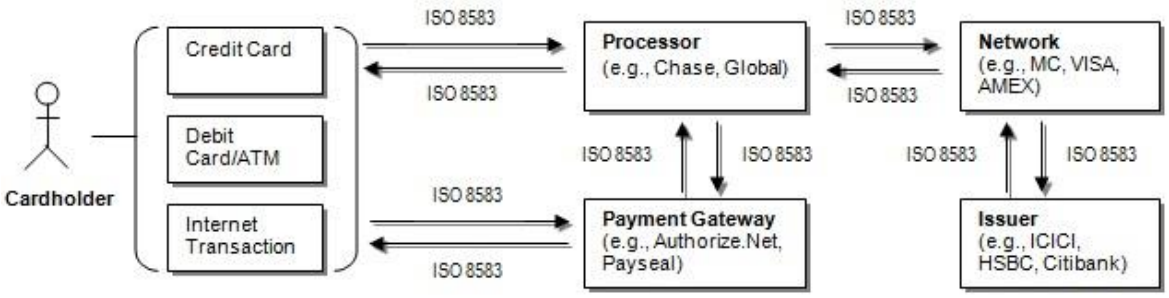

Figura 12 ISO8583

[Disponible en https://www.codeproject.com/Articles/100084/Introduction-to-ISO/]

Dentro del proyecto de tesis se generó una lectura de las tramas ISO8583 generadas en las transacciones de tarjetas de débito de la franquicia VISA para obtener campos adicionales a los que se tienen en la base de datos origen, necesarios para alimentar el almacén de datos destino con información relevante para la toma de decisiones. En el capítulo de implementación se puede observar con mayor detalle el funcionamiento de este tipo de formato.

Estructura

Un mensaje ISO 8583 consta de las siguientes partes:

- Message Type Indicator (MTI): Indicador de Tipo de Mensaje.
- Mapa de bits: Uno o más bitmaps, indicando qué elementos están presentes en el mensaje.
- Data elements: los campos del mensaje.

A continuación, se describe brevemente estos elementos:

- **MTI (Indicador del tipo de mensaje)**

Campo numérico de 4 dígitos, que determina el tipo de mensaje. En la Tabla 3 se indican los diferentes tipos de MTI y su significado.

MTI	Significado	Uso
0100	Solicitud de Autorización	Solicitar desde un terminal de punto de venta la autorización para una compra del titular de la tarjeta.
0110	Respuesta de Autorización	Solicitar respuesta a un terminal de punto de venta para la autorización de una compra del titular de la tarjeta
0200	Solicitud financiera del adquirente	Solicitud de fondos, generalmente desde un ATM (cajero automático) o un POS (Punto de venta fijo)
0210	Respuesta del emisor a la solicitud financiera	Respuesta del emisor a la solicitud de fondos

Tabla 3 Ejemplos MTI

- **Mapa de bits**

Un mapa de bits es una técnica utilizada en un mensaje ISO 8583 para indicar qué elementos de datos están activos. El mapa de bits primario indica que campos del 1 al 64 están presentes. Puede también existir un mapa de bits secundario, donde los campos del 65 al 128 están activos.

- **Data Elements**

Son los campos del mensaje que contienen la información sobre la transacción. Contiene 128 campos definidos en el standard ISO8583:1987, y 192 en posteriores actualizaciones.

ISO 3166

La norma ISO 3166 establece códigos reconocidos internacionalmente para representar nombres de países, territorios o áreas de interés geográfico y sus subdivisiones. El uso de este tipo de códigos de letras y/o números para representar el nombre del país puede ayudar a ahorrar tiempo y reducir la tasa de error en conversiones. En la tabla 4 se presenta un ejemplo de conversión, donde para el código del país Ecuador podemos obtener el nombre común o el código alfa requerido.

Nombre común	Nombre ISO oficial del país o territorio	Código alfa-2	Código alfa-3	Código numérico
Ecuador	Ecuador	EC	ECU	218

Tabla 4 Ejemplo ISO 3166 para formato de Ecuador

4.2.2.2 CONECTOR DESTINO (SINK CONNECTOR)

Transmite la información desde Kafka hacía una fuente de destino es decir procede a leer los registros de los topics/temas y los lleva hacia un almacén de datos destino a través del conector, como se aprecia en la figura 14. Si se produce un fallo durante la lectura de los datos de Kafka se notificará al conector. Actualmente ya existen conectores implementados en el mercado para: JDBC, Hadoop, HIVE, ELASTIC, Cassandra, S3, etc.

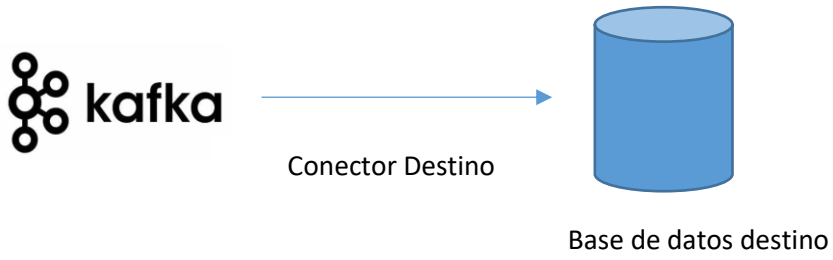

Figura 13 Funcionamiento Kafka con conector destino

4.2.3 ALMACENAMIENTO NO SQL Y MOTORES DE BÚSQUEDA

INTRODUCCIÓN

Actualmente con el almacenamiento relacional, dependiendo de la cantidad de la información y la búsqueda a realizar podría tomar una consulta desde unos pocos segundos hasta varios minutos. Para agilizar el proceso y ser eficiente en el manejo de la información en tiempo real, las características de almacenes NoSQL permiten una solución práctica para indexar miles de datos y generar búsquedas eficientes y agiles. A este tipo de tecnologías se las conoce como bases de datos NoSQL de búsqueda.

A continuación, se detalla un análisis de tecnologías actuales pertenecientes a este tipo de bases de datos no relacionales de código abierto.

4.2.3.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA HERRAMIENTAS DE ALMACENAMIENTO NO SQL

Dada la inmensa cantidad de información de transacciones de tarjetas de débito que se genera cada segundo es necesario salirse del modelo relacional de almacenamiento y llevarla hacia una modelo flexible de datos. Entre las bases de datos NoSQL más utilizadas para almacenamiento podemos mencionar a MongoDB, CouchDB como tecnologías altamente escalables, con un diseño orientado a documentos -de acuerdo al ranking obtenido de la página https://db-engines.com/en/ranking de los sistemas de administración de base de datos según su popularidad colocadas en primero y segundo lugar respectivamente-. Por otro lado, es primordial en el caso de la inteligencia de negocios tener un acceso a los datos de manera rápida, fiable y optimizada. Los motores de búsqueda NoSQL permiten hacer frente a este propósito y entre los cuales se encuentran Elasticsearch y Solr, como los más populares como se presenta en la Tabla 5.

☐ include secondary database models — 21 systems in ranking, October 2021

Rank Oct 2021	Rank Sep 2021	Rank Oct 2020	DBMS	Database Model	Score Oct 2021	Score Sep 2021	Score Oct 2020
1.	1.	1.	Elasticsearch	Search engine, Multi-model	158.25	-1.98	+4.41
2.	2.	2.	Splunk	Search engine	90.61	-0.99	+1.21
3.	3.	3.	Solr	Search engine, Multi-model	51.17	+1.36	-1.31
4.	4.	4.	MarkLogic	Multi-model	9.43	-0.16	-2.30
5.	5.	↑ 7.	Sphinx	Search engine	7.61	-0.03	+1.27

Tabla 5 Ranking de motores de búsqueda

[Disponible en: https://db-engines.com/en/ranking/search+engine]

Para el caso de este estudio la atención se centró en las tecnologías NoSQL como motores de búsqueda, pretendiendo así generar consultas rápidas en decenas de milisegundos que permitan una visualización en tiempo real dentro de los tableros de control. Para tal efecto, en la Tabla 6 se realiza una comparación de las dos primeras herramientas de código abierto definidas dentro del mundo del Big Data como los mejores motores de búsqueda obtenido del portal DB-Engine [11].

A continuación, se presentan los criterios de comparación (Tabla 6) que permiten cubrir los puntos a considerar para elegir una herramienta que ejecute un almacenamiento no relacional y búsquedas de texto en tiempo real (necesaria para el caso de estudio).

- **Funcionalidad principal:** Los motores de búsqueda son sistemas de gestión de bases de datos NoSQL dedicados para la búsqueda de contenido de datos potentes, ofreciendo consultas en casi tiempo real.
- **Funcionalidad secundaria**: Los almacenes de documentos son llamados sistemas de bases de datos orientados a documentos dado que se caracterizan por su organización de datos sin necesidad de esquemas.
- **Ranking de motores DB:** Clasificación los sistemas de gestión de bases de datos y los motores de búsqueda según su popularidad.
- **Recuperación de la información:** Lucene es una API de código abierto para recuperación de información útil para cualquier aplicación que requiera indexado y búsqueda a texto completo.

[11] https://db-engines.com/en/system/Elasticsearch%3BSolr%3BSplunk. Consultado en octubre 2021.

- **Lenguajes que soporta:** Lenguajes de programación que pueden utilizar la herramienta a través de clientes o conectores.
- **Documentación:** Tipo de documentación con la que cuenta la herramienta.
- **Query DSL:** Permite la construcción de consultas usando JSON para dar control y lógica a los datos. Los resultados son ordenados por la puntuación en cuanto a la calidad de relevancia.
- **Herramientas de visualización:** Herramientas que permite conectarse directamente al motor de búsqueda para la generación de informes analíticos y monitoreo.
- **Escalabilidad:** Permite alta disponibilidad de los nodos y escalamiento horizontal.
- **Búsqueda de texto completo:** Permite que las consultas coincidan con los documentos inmediatamente después de que fueron indexados casi en tiempo real.
- **Formato de salida:** Tipo de formato en que son devueltas las consultas realizadas.

	Elasticsearch	**Solr**
Funcionalidad Principal	Motor de búsqueda	Motor de búsqueda
Funcionalidad Secundaria	Almacén de documentos	Almacén de documentos
Ranking de motores DB	Score 158.25 Rank #8Overall #1Search engines	Score 51.17 Rank #22 Overall #3 Search engines
Desarrollador	Elastic	Apache Software Foundation
Recuperación de Información	Lucene	Lucene
Documentación	Buena documentación.	Desactualizada
Versión Inicial	2010	2004
Sistemas operativos	Todos los SO con JVM	Todos los SO con JVM
Lenguajes que soporta	.Net Groovy Community Contributed Clients Java	.Net Erlang

	JavaScript Perl PHP Python Ruby	Java JavaScript any language that supports sockets and either XML or JSON Perl PHP Python Ruby Scala
Query DSL	Si	Búsqueda a través de URIs con parámetros.
Herramientas de visualización	Kibana Grafana	Banana Apache Hue
Escalabilidad	Si	Escalable solo con la ayuda de SolrCloud y Zookeeper.
Búsqueda de Texto completo	Si	Si
Formatos de Salida	JSON	JSON, XML y CSV

Tabla 6 Tabla comparativa de tecnologías para almacenamiento NoSQL como motores de búsqueda

4.2.3.2 HERRAMIENTA SELECCIONADA

Para esta implementación, la principal funcionalidad que se busca es poder realizar búsquedas de texto potentes, precisas y eficientes; que permitan tomar decisiones en tiempo real de negocios donde se utilizan las tarjetas de débito como medio de pago. Además, la necesidad de crear índices independientes relacionados con nombres de países, posición geográfica, y búsqueda de palabras clave a través de expresiones. Bajo estas premisas se determinó que Elasticsearch para la propuesta de esta arquitectura es la mejor opción. A continuación, se detalla esta tecnología seleccionada.

ELASTICSEARCH

Elasticsearch (ES) es un motor de búsqueda potente desarrollado en Java y publicado como código abierto bajo la licencia de Apache. Gestiona múltiples fuentes con información estructurada y no estructurada permitiendo así almacenar, buscar y analizar grandes volúmenes de información de forma eficiente a través de una API web RESTful.

Por ejemplo, en una arquitectura con hardware similar, una consulta que tardaría cerca de 10 segundos en un motor de búsqueda SQL, en ES el resultado podría obtenerse en menos de 10 milisegundos.

4.3 COMPONENTE 3 – TABLERO DE CONTROL

INTRODUCCIÓN

Este componente tiene como objetivo principal presentar el estado actual de los datos a través de herramientas visuales y centralizadas que permitan una toma de decisiones más precisa de acuerdo a la línea del negocio. Los tableros de control capturan la información de diferentes indicadores de la empresa como zonas geográficas, países con mayor uso del servicio, comercios, etc. y los llevan hacia un enfoque incluso más detallado permitiendo así monitorear si la operación de la organización está en el camino correcto para alcanzar los objetivos planteados.

En consecuencia, se ha planteado a este mecanismo de visualización como el tercer componente de la arquitectura planteada en esta tesis. Este componente busca presentar en tiempo real los datos de las transacciones de tarjetas de débito, previo al proceso de limpieza y transformación obtenido del componente 2. La información obtenida en tiempo real muestra los cambios en los indicadores establecidos, ofreciendo al personal respectivo de las áreas de operaciones y negocio, herramientas para mejora continua de servicios que oferta la institución financiera. A estas herramientas se las conoce como Tableros de Control Empresarial o Dashboards.

4.3.1 ANÁLISIS DE SOLUCIONES ACTUALES PARA TABLEROS DE CONTROL (DASHBOARD)

Hoy en día, existe una enorme lista de poderosas herramientas que actúan como tableros de control para inteligencia de negocios (Businnes Inteligence - BI) que permiten ilustrar de una manera sencilla y en tiempo real los datos de una organización. Entre las plataformas más utilizadas a nivel global se encuentran Tableau y Microsoft Power BI como líderes en el mercado según se ilustra en la Figura 15 obtenida del cuadrante Mágico de Gartner sobre las mejores plataformas para análisis de

datos y la inteligencia de negocios. Sin embargo, esta comparación se centra en herramientas de código abierto que posean conexión directa con Elasticsearch (base de datos NoSQL como motor de búsqueda seleccionado en el componente 2). Se realiza el análisis de dos herramientas que serán parte de este estudio dado que contemplan las características que se requieren en el proyecto propuesto.

Figura 14 Cuadrante Mágico de Gartner para Plataformas Analíticas y de Business Intelligence 2021

[Disponible https://softwareyhardware.com/software/cuadrante-magico-de-gartner-2019-analitica-e-inteligencia-de-negocio-bi/]

Grafana y Kibana son dos herramientas populares de código abierto que ayudan a los usuarios a visualizar y comprender tendencias dentro de grandes cantidades de datos de registro y mantienen una relación estrecha con Elasticsearch.

Las características de las tecnologías que han sido seleccionadas para cubrir los puntos a considerar para elegir el tablero de control son:

- **Fuentes de datos:** Integración para visualización de la información con diferentes fuentes de datos.

- **Comunidad:** Cantidad de desarrollos realizados en Git Hub sobre la tecnología.

- **Autenticación LDAP**: Mecanismos que permite a los usuarios restringir y controlar el acceso a sus paneles.

- **Alertas:** Los motores de alertas permite a los usuarios manejar casos como por ejemplo la falta de disponibilidad de los datos o conexiones fallidas a través de la creación de alertas personalizadas.

- **Tipos de gráficas:** Capacidad de gráficas para visualización en los paneles de los tableros de control.

	Kibana	**Grafana**
Desarrollado por	Elastic NV.	Torkel Ödegaard
Funcionalidad Principal	Visualizar y analizar métricas como latencia del sistema, carga de CPU, utilización de RAM.	Análisis de archivos de registro y consultas de búsqueda de texto completo.
Fuentes de datos	Elasticsearch (ELK)	Graphite, Prometheus, InfluxDB, MySQL, PostgreSQL y Elasticsearch, y fuentes de datos adicionales mediante complementos.
Comunidad	Más de 17000 confirmaciones	Más de 14000 confirmaciones
Autenticación LDAP	No (acceso público)	Si
Alertas	No admite alertas directas pero puede realizarse con complementos.	Alertas integradas para los usuarios finales
Tipos de Gráficas	Barras Histograma Pastel Mapa de Calor Datos Geoespaciales Nubes de etiquetas	Barras Histograma Pastel Mapa de Calor

Tabla 7 Tabla comparativa de tecnologías para tableros de control

4.3.2 HERRAMIENTA SELECCIONADA: KIBANA

La herramienta de visualización seleccionada para actuar como tablero de control es Kibana dado que es una mejor alternativa para un análisis y consulta de los datos complejos; además de poseer una fácil configuración para la presentación de datos obtenidos con ElasticSearch.

KIBANA

Es un panel de visualización de datos de código abierto permitiendo a los usuarios crear gráficos de barras, líneas y dispersión, gráficos circulares y mapas de calor sobre grandes volúmenes de datos. Kibana utiliza la estructura ELK. ELK es una colección de 3 herramientas de código abierto: Elasticsearch, Logstash y Kibana como soluciones de monitoreo para administración de registros de un extremo a otro para datos de Elasticsearch.

A continuación, en la Figura 16, se presenta la comunicación entre los elementos 4 y 5 de la arquitectura planteada en la Figura 7.

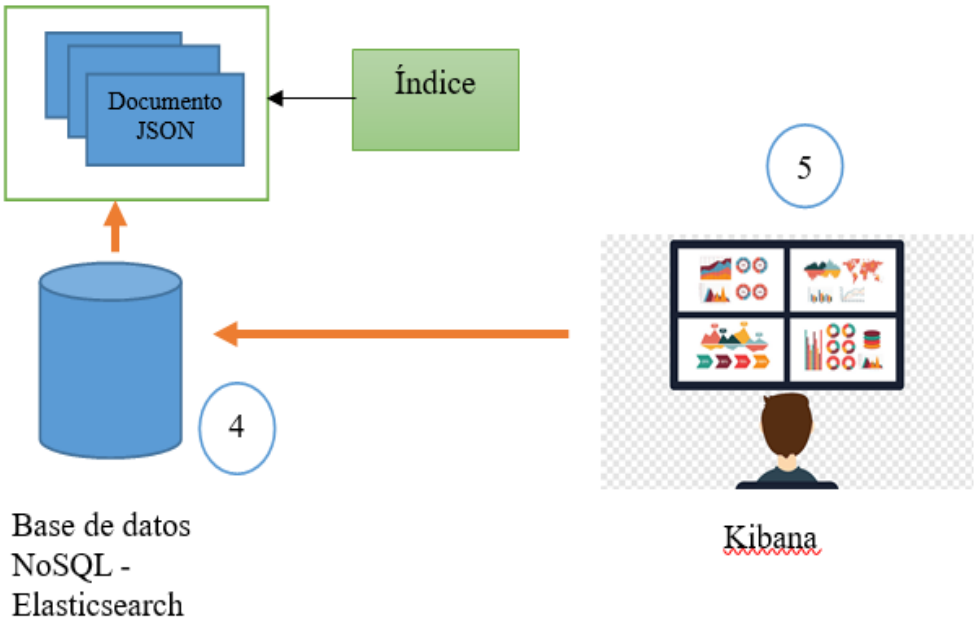

Figura 16 Funcionamiento Tablero de Control a Base de Datos NoSQL

La comunicación entre el tablero de control y la base de datos no relacional (Elasticsearch) donde se encuentran almacenados los datos finales se la realiza a través de la conexión al índice que contiene los documentos JSON creados en el microservicio como se aprecia en la Figura 16.

Las secciones que se describen a continuación son las que se han utilizado dentro del presente proyecto para configuración de los índices, documentos y representaciones gráficas que se han implementado en el dashboard o tablero de control diseñado para mostrar la información de los indicadores de tarjetas de débito en tiempo real (Figura 17).

- Discover: Pantalla que nos permite filtrar y buscar registros en un intervalo específico.

- Visualize: Pantalla donde se pueden crear, modificar y ver las visualizaciones personalizadas (Gráficos, tablas, …).

- Dashboard: Pantalla donde se pueden crear, modificar y ver sus propios cuadros de

mando personalizados.

- Settings: Pantalla que permite cambiar la configuración por defecto o patrones de índice.

Figura 15 Dashboard Kibana

[Disponible en https://www.elastic.co/es/kibana/]

Como resultado se puede apreciar un tablero de control con diferentes secciones simultaneas que cuentan con diferentes tipos de gráficas para dar una perspectiva en tiempo real de los datos y lo que está sucediendo con el negocio.

Finalmente, en la Figura 18 se presenta la Arquitectura Final compuesta de cada una de las herramientas seleccionadas dentro de cada uno de los componentes y que serán desarrolladas/configuradas según sea el caso para llevar al cumplimiento de los objetivos planteados dentro del presente proyecto de tesis.

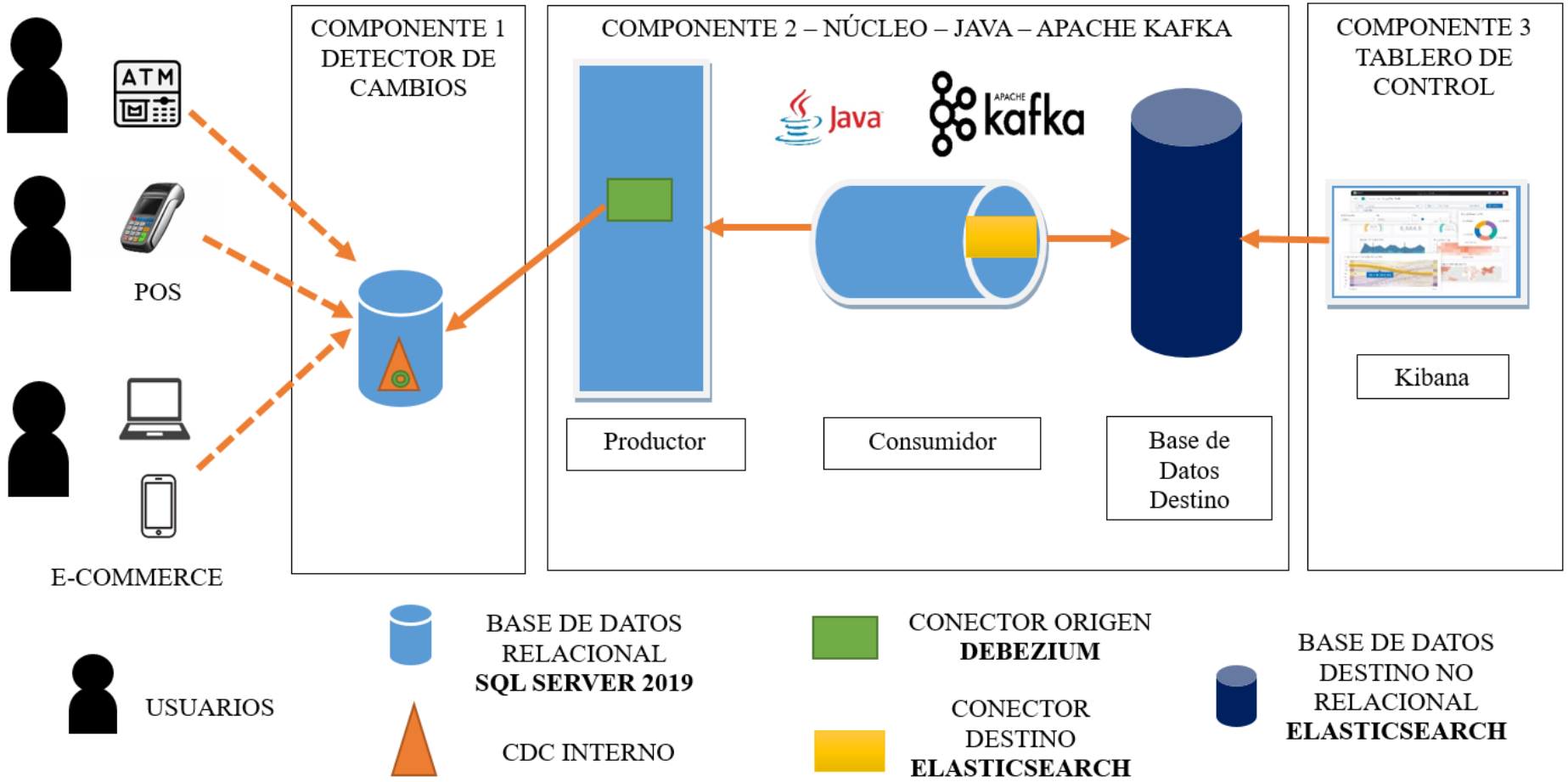

Figura 16 Arquitectura Final

En la figura anterior se encuentran instanciadas las herramientas que se utilizaran en el proyecto, mientras que en la Figura 19 se presentan las entradas y salidas que genera cada uno de los elementos que se encuentran dentro de los componentes de la arquitectura propuesta.

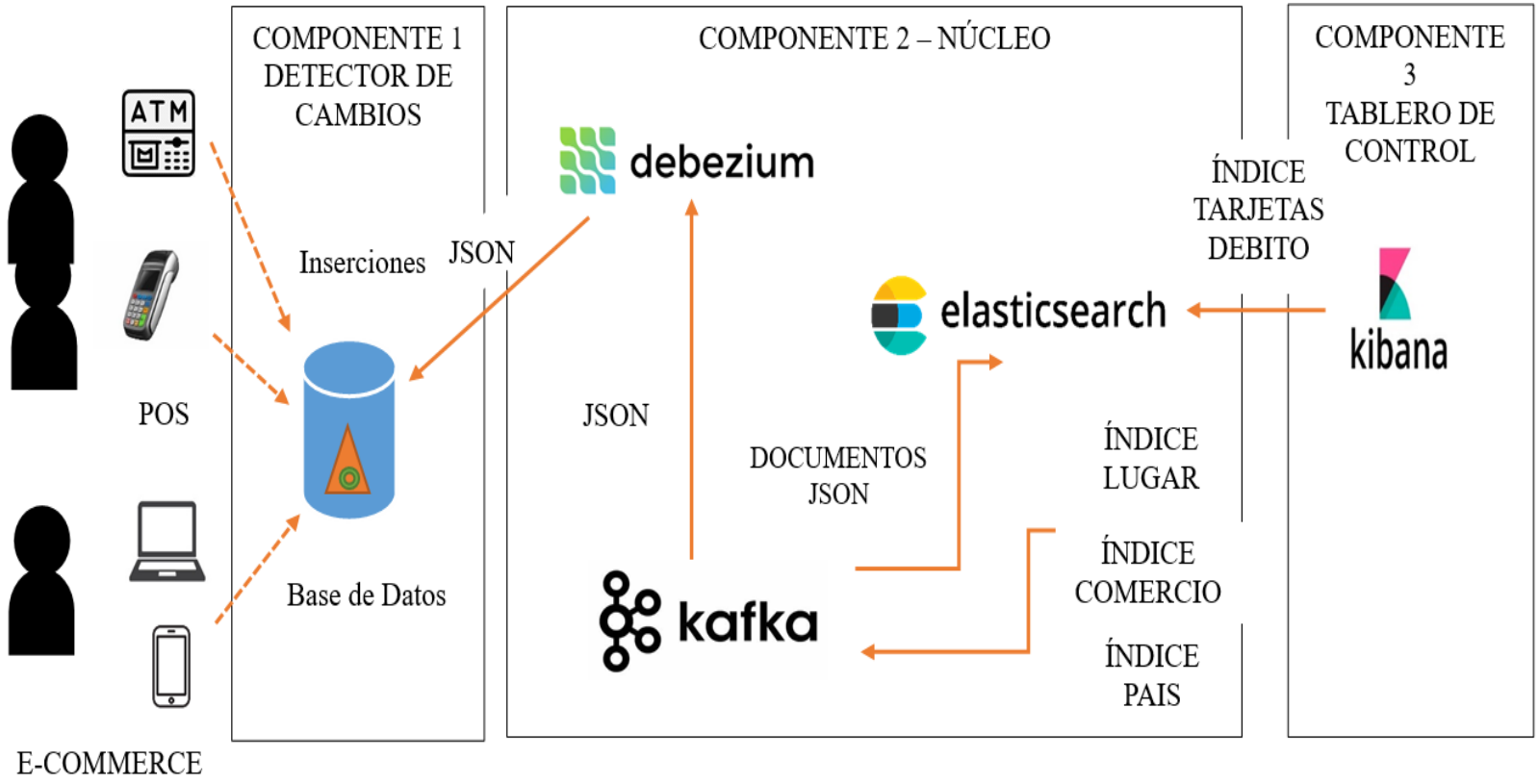

Figura 17 Entradas y Salidas de Formatos y Estructuras de Datos en Arquitectura Propuesta

CAPITULO 5 IMPLEMENTACIÓN DEL DISEÑO PROPUESTO

El desarrollo del presente capítulo se centra en la implementación de la arquitectura propuesta en el Capítulo 4 para el procesamiento en tiempo real de un flujo de datos hacia tableros de control para visualización de la información. El proyecto se centra en la integración de tecnologías que gestionan los datos en tiempo real mediante un intermediador de mensajes y donde gracias a un microservicio se procesa la limpieza y conversión de los datos. El diseño toma como premisa el volumen de información que se produce en el sector financiero como caso de estudio inicial, pero se plantea que esta misma arquitectura pueda ser empleada en otros entornos.

Para el procesamiento de los datos se ha elegido a Apache Kafka como sistema de mensajería seleccionado para transmisión y procesamiento de los flujos de datos y Elasticsearch como sistema de almacenamiento ya que cuenta con un motor de búsqueda que permite realizar consultas rápidas de los datos, además de poseer flexibilidad también para almacenar la información en tiempo real. Para crear el tablero de control que presente a través de gráficas el funcionamiento de las tarjetas de débito se empleó Kibana que permite gracias a su integración directa con Elasticsearch crear visualizaciones interactivas de la información en tiempo real. A continuación, se detalla la instalación/configuración de cada una de las tecnologías empleadas en la arquitectura propuesta en la Figura 20.

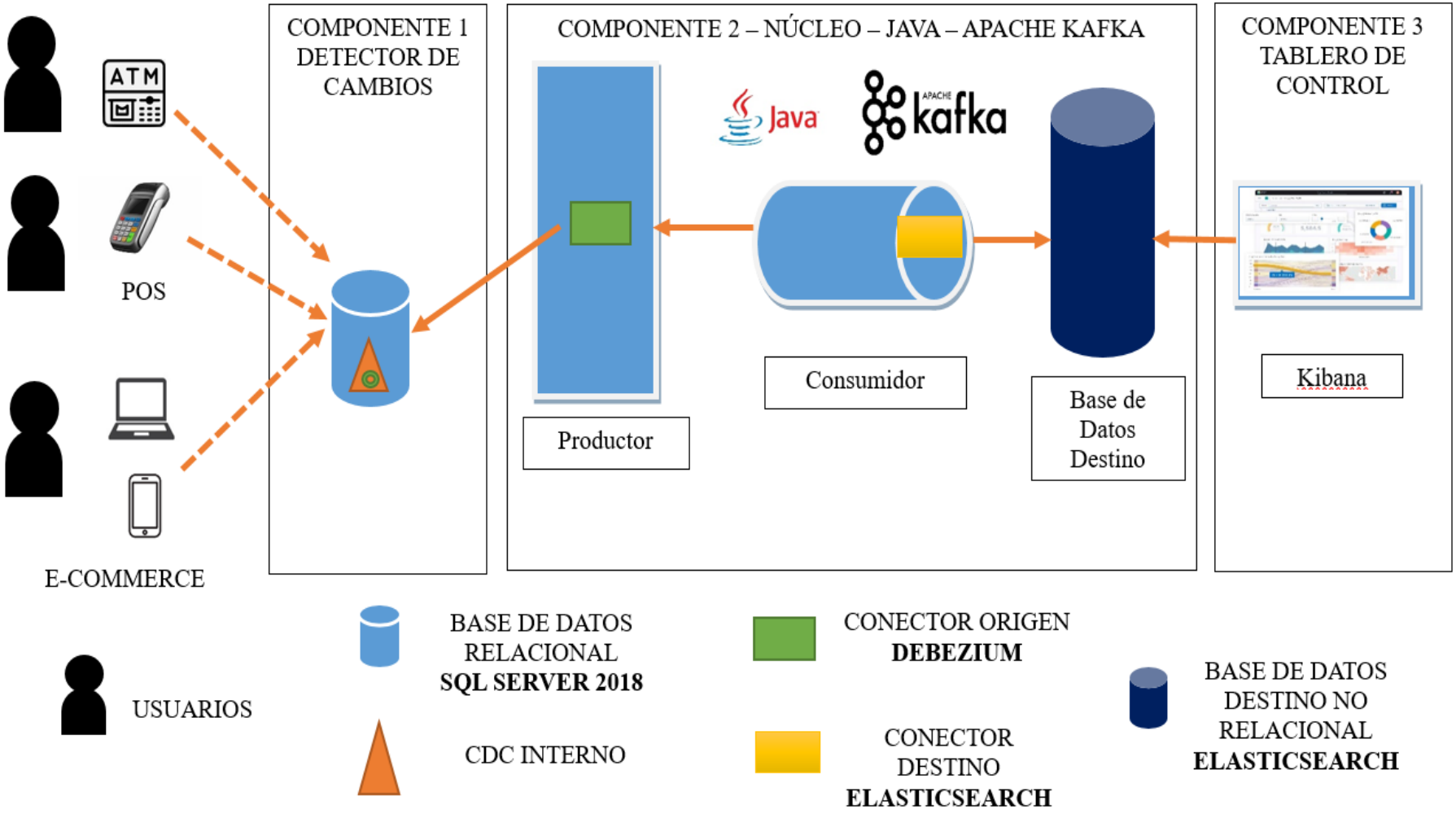

Figura 18 Arquitectura Propuesta

Para cada componente de la arquitectura propuesta se detalla la instalación o configuración necesaria para su funcionamiento, considerando las herramientas seleccionadas en el Capítulo 3.

5.1 PREPARACIÓN DEL ENTORNO

La implementación de la arquitectura propuesta se realizó sobre una máquina con las siguientes características de hardware y software (Tabla 8).

Parámetro	Valor
Sistema Operativo	Suse Linux
RAM	16 GB
CPU	Intel(R) Core(TM) i7-8700T CPU @ 2.40GHz
Versión Debezium	debezium-connector-sqlserver-1.2.0.C
Versión Kafka	2.13-2.5.0
Versión Zookeper	3.6.1
Versión Elasticsearch	7.8.0
Versión Kibana	7.8.0

Tabla 8 Características de Software y Hardware

Es importante aclarar que para verificar el funcionamiento de la arquitectura se diseñaron pruebas bajo el modelo autónomo de Apache Kafka, se deja planteado el diseño de pruebas en un entorno distribuido para trabajos futuros.

5.2 COMPONENTES DE LA ARQUITECTURA

Inicialmente como se aprecia en la Figura 20 se parte con la lectura de las transacciones almacenadas en las fuentes de datos y enviadas hacia la tabla donde se escucha los eventos de cambio. En el primer paso, se leen todos los registros que se encuentran almacenados a través de la captura de

una instantánea de la tabla, mecanismo propio que ofrece Apache Kafka. Para la simulación de un flujo en tiempo real de la información generada en los puntos de venta (POS), cajeros automáticos (ATMs) o comercio electrónico se creó un script de pruebas que realiza inserciones a la tabla para visualizar su funcionamiento en streaming hacia el tablero de control representando así un flujo de datos.

A continuación, se describe con más detalle la instalación o configuración de los componentes y la integración entre cada uno, siguiendo los siguientes pasos:

- Activar CDC para la tabla a verificar de eventos de cambio.
- Arrancar el Zookeeper.
- Arrancar un broker (nodo).
- Definir un topic (tema) con una única partición y sin replicación.
- Utilizar un consumidor "por línea de comandos".
- Utilizar el microservicio.
- Almacenar los datos en una base de datos no relacional.
- Visualizar las transacciones en un tablero de control.

5.2.1 DETECTOR DE CAMBIOS

Dentro del primer componente se encuentra el proceso de Captura de Cambios de los Datos (CDC). El primer paso para habilitar la CDC se realiza mediante la ejecución del procedimiento sys.sp_cdc_enable_db que se encuentra dentro de la base de datos a habilitar bajo el rol de administrador, por ejemplo:

```
USE BASEDEDATOS
GO
EXEC sys.sp_cdc_enable_db
GO
```

Una vez que la base de datos está habilitada para CDC, el siguiente paso es habilitar cada tabla (origen) de la cual se desea seguir los cambios que se hagan sobre ella. Esto se hace utilizando el procedimiento almacenado sys.sp_cdc_enable_table.

```
USE BASEDEDATOS
GO
EXEC sys.sp_cdc_enable_table
@Source_schema = N'dbo,
@Source_name = N'TABLA ,
@Role_name = N'CDCRole '
```

Parámetros a utilizar

@Source_schema - es el nombre de esquema de la tabla de origen.

@Source_name - es el nombre de la tabla de origen que se desea habilitar para CDC.

@Role_name - es la función de seguridad que se crea cuando los CDC está activado. Este rol puede ser ignorado, o se puede utilizar para asignar permisos a usuarios específicos, para que puedan acceder a los datos utilizando las funciones de los CDC.

En la figura 21 y 22 se puede observar el resultado de la ejecución del script de activación de CDC (Anexo 1) para el caso de estudio de la institución financiera. En las figuras se muestra el resultado de la activación tanto en la base de datos y la tabla respectivamente donde se almacenan las transacciones de tarjetas de débito.

Results Messages

	name	is_cdc_enabled
1	master	0
2	tempdb	0
3	model	0
4	msdb	0
5	STAG	0
6	Sentinel	0
7	SSISDB	0
8	STAG2	1

Figura 19 Base de datos activada con CDC

	name	is_tracked_by_cdc
1	SwTrans_old	1
2	systranschemas	0
3	change_tables	0
4	ddl_history	0
5	lsn_time_mapping	0
6	captured_columns	0
7	index_columns	0
8	dbo_SwTrans_old_CT	0

Figura 20 Tabla para capturar las transacciones de tarjetas de débito con opción CDC activada

En el Anexo 1 se detalla los scripts utilizados para activar CDC en el proceso de captura de cambios sobre una base de datos SQL SERVER 2019.

5.2.2 NÚCLEO

Una vez se encuentra habilitado el proceso CDC dentro de la tabla/tablas a escuchar los eventos de cambio se configura el componente principal llamado “Núcleo”. Al ser este componente el que maneja todo el proceso de ingestión, limpieza, y almacenamiento de datos en tiempo real, los mecanismos o herramientas que son aplicadas se describen con detalle a continuación.

Para la presente implementación se utilizó la última versión de Apache Kafka como bróker de mensajería (descrito en el capítulo 3) y obtenida desde la página web oficial https://kafka.apache.org/downloads.

Una vez descomprimido el archivo, se procedió a descargarlo en un directorio elegido. La carpeta elegida pasa a ser el directorio de instalación, el cual se referencia como KAFKA_HOME.

Para entornos Unix

Todos los ejecutables .sh se encuentran bajo el directorio %KAFKA_HOME%\bin

Todos los archivos de configuración de Apache Kafka se encuentran en el directorio %KAFKA_HOME%\config.

Para entornos Windows

Todos los ejecutables .bat se encuentran bajo el directorio %KAFKA_HOME%\bin\windows

Todos los archivos de configuración de Apache Kafka se encuentran en el directorio %KAFKA_HOME%\config\windows

Requisitos previos

Se requiere tener instalado una versión de Java acorde con la versión descargada de la plataforma.

Se aconseja utilizar Java 8.

5.2.2.1 CONFIGURACIÓN DE APACHE ZOOKEPER

.Zookeeper ofrece un servicio para la coordinación de procesos distribuidos, permitiendo la gestión de grupos, protocolos de presencia y elección de líder de los nodos para el proceso de replicación que ofrece Kafka. Siendo Zookeeper lo primero que se requiere ejecutar para empezar a trabajar con Kafka a continuación se describe la configuración definida y los parámetros a utilizar.

El archivo por defecto es: %KAFKA_HOME%\config\zookeeper.properties

Parámetros de configuración

- El puerto por defecto es el 2181 (propiedad "clientPort")
- El directorio de datos por defecto es "/tmp/data" (propiedad "dataDir")

Se aconseja modificar “dataDir=/tmp/zookeeper” por otra ruta para que NO se elimine cada cierto tiempo.

Iniciar Apache Zookeper

Desde %KAFKA_HOME%\bin donde se descomprimió el archivo Kafka se ejecuta la siguiente línea (para distribuciones UNIX) para inicializar Zookeper teniendo como resultado la Figura 23.

./zookeeper-server-start.sh %KAFKA_HOME%\config\zookeeper.properties

```
[2020-10-17 15:21:41,315] INFO minSessionTimeout set to 6000 (org.apache.zookeeper.server.ZooKeeper
Server)
[2020-10-17 15:21:41,315] INFO maxSessionTimeout set to 60000 (org.apache.zookeeper.server.ZooKeepe
rServer)
[2020-10-17 15:21:41,316] INFO Created server with tickTime 3000 minSessionTimeout 6000 maxSessionT
imeout 60000 datadir /opt/kafka/zookeper/tmp/version-2 snapdir /opt/kafka/zookeper/tmp/version-2 (o
rg.apache.zookeeper.server.ZooKeeperServer)
[2020-10-17 15:21:41,322] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server con
nection factory (org.apache.zookeeper.server.ServerCnxnFactory)
[2020-10-17 15:21:41,324] INFO Configuring NIO connection handler with 10s sessionless connection t
imeout, 2 selector thread(s), 24 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.se
rver.NIOServerCnxnFactory)
[2020-10-17 15:21:41,326] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NI
OServerCnxnFactory)
[2020-10-17 15:21:41,335] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKD
atabase)
[2020-10-17 15:21:41,501] INFO Reading snapshot /opt/kafka/zookeper/tmp/version-2/snapshot.630 (org
.apache.zookeeper.server.persistence.FileSnap)
[2020-10-17 15:21:41,562] INFO Snapshotting: 0x643 to /opt/kafka/zookeper/tmp/version-2/snapshot.64
3 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2020-10-17 15:21:41,575] INFO Using checkIntervalMs=60000 maxPerMinute=10000 (org.apache.zookeeper
```

Figura 21 Inicio Zookeper

Con esto, Zookeeper estará conectado y escuchando en el puerto **2181** definido en el archivo de configuración (zookeper.properties).

5.2.2.2 CONFIGURACIÓN DE DEBEZIUM

El conector Debezium captura los cambios a nivel de fila que se producen en los esquemas de una base de datos.

Hay varios tipos de conectores Debezium para diferentes bases de datos relacionales y no relacionales, entre las más utilizadas se encuentran:

- MySql
- Postgres
- MongoDB
- Sql Server
- Oracle
- DB2
- Cassandra

Todos estos conectores pueden ser encontrados en la página oficial de Debezium https://debezium.io/documentation/reference/install.html

Para más información de la configuración de Debezium con el conector SQL SERVER (base de datos utilizado por la institución financiera) se puede consultar el Anexo 2.

5.2.2.3 CONFIGURACIÓN DE SERVER APACHE KAFKA

Una vez arrancado Zookeeper se procede a configurar los nodos de Apache Kafka para su inicialización. Los nodos de Kafka son el corazón del clúster y actúan como canalizaciones donde se almacenan y distribuyen los datos.

El **archivo por defecto** es: %KAFKA_HOME%\config\server.properties

En el archivo de configuración se pueden modificar los siguientes parámetros de configuración para que se conecte a la instancia de Zookeeper (ver Figura 24).

Parámetros de configuración

broker.id=Identificador único del broker/nodo

log.dirs=/opt/kafka/kafka_2.13-2.5.0/logs

zookeeper.connect=localhost:2181 (dirección de zookeper)

```
############################# Log Basics #############################

# A comma separated list of directories under which to store log files
log.dirs=/opt/kafka/kafka_2.13-2.5.0/logs

# The default number of log partitions per topic. More partitions allow greater
# parallelism for consumption, but this will also result in more files across
# the brokers.
num.partitions=1

# The number of threads per data directory to be used for log recovery at startup and flushing at shutdown.
# This value is recommended to be increased for installations with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################# Internal Topic Settings  #############################
# The replication factor for the group metadata internal topics "__consumer_offsets" and "__transaction_state"
# For anything other than development testing, a value greater than 1 is recommended to ensure availability such as 3.
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
```

Figura 22 server.properties

Iniciar nodo Kafka

Desde el directorio %KAFKA_HOME%\bin se ejecuta la siguiente línea para inicializar un nodo de Kafka.

./kafka-server-start.sh %KAFKA_HOME%\config\server.properties

Las propiedades definidas en el archivo (server.properties) como el número de particiones y el factor de replicación son establecidas en 1 por defecto. A continuación, se explica el proceso para personalizar un tema bajo un factor de replicación y un número de particiones.

5.2.2.4 GESTIÓN DE LOS TEMAS (TOPICS)

Para la gestión de temas se proporciona dentro de Apache Kafka el siguiente comando **kafka-topics**.

CREAR TEMA

Permite crear un tema con una configuración específica.

./kafka-topics.sh --create --zookeeper 127.0.0.1:2181 --replication-factor 1 --partitions 1 --topic integracionserver.dbo.SwTrans_old

Parámetros de configuración

--create: Indica que la acción a realizar es la creación

--zookeeper: Establece la dirección del Zookeeper con la que trabajará

--partitions: Define cuántas particiones habrá en un tema

--replication-factor: Número de copias del tema en un Kafka cluster

--topic: Establece el nombre del tema

BORRAR TEMA

Permite eliminar un tema creado.

./kafka-topics.sh --delete --zookeeper 127.0.0.1:2181 --topic integracionserver.dbo.SwTrans

Parámetros de configuración

--delete: Indica que la acción a realizar es la eliminación

--zookeeper: Establece la dirección del Zookeeper con la que trabajara

--topic: Establece el nombre del tema (En este caso " integracionserver.dbo.SwTrans")

LISTAR TEMAS

Muestra todos los temas que se encuentran creados dentro de Zookeeper

./kafka-topics.sh --list --bootstrap-server localhost:9092

Parámetros de configuración

--list: Indica que la acción a realizar es el listado de temas.

--zookeeper: Establece la dirección del Zookeeper con la que trabajara.

5.2.2.5 CONFIGURACIÓN AUTÓNOMA DE KAFKA

Los conectores y las tareas son unidades lógicas de trabajo y se ejecutan como un proceso. Este proceso se denomina trabajador en Kafka Connect. Hay dos modos para ejecutar trabajadores: modo autónomo y modo distribuido. Para las pruebas realizadas en este trabajo se utilizó el modo autónomo para probar el funcionamiento de la arquitectura propuesta.

El modo autónomo (standalone) es útil para desarrollar y probar Kafka Connect en una máquina local. También se puede utilizar para entornos que suelen utilizar agentes únicos (por ejemplo, enviar registros de un servidor Web a Kafka).

A continuación, se muestra un comando de ejemplo que inicia un trabajador en modo independiente desde la carpeta principal %KAFKA_HOME%\bin.

Iniciar trabajador Kafka

El siguiente comando permite iniciar un trabajador en modo autónomo.

```
./connect-standalone.sh worker.properties connector.properties
```

El primer parámetro (`worker.properties`) es el archivo de propiedades de configuración del trabajador (Figura 25). Este archivo le da control sobre configuraciones como el clúster de Kafka para usar en caso de que sea requerido y el formato de serialización.

```
offset.storage.file.filename=/tmp/connect211.offsets
bootstrap.servers=127.0.0.1:9092
offset.flush.interval.ms=1000
rest.port=10082
rest.host.name=127.0.0.1
rest.advertised.port=10082
rest.advertised.host.name=127.0.0.1
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
plugin.path=/opt/kafka/plugins/
#If kafka is TLS authenticated, uncomment below lines.
#security.protocol=SSL
#ssl.truststore.location=/tmp/kafka.client.truststore.jks
#producer.security.protocol=SSL
#producer.ssl.truststore.location=/tmp/kafka.client.truststore.jks
```

Figura 23 worker.properties

El segundo parámetro (connector.properties) es el archivo de propiedades de configuración del conector como se puede observar en la Figura 26, mantiene los datos de conexión a una base de datos, es en este punto que se realiza la respectiva integración entre los plugins Debezium agregados anteriormente del tipo de base de datos que se desea utilizar del proceso de captura de cambios y la relación con el trabajador. En este archivo adicional se encuentra el nombre del tema donde se publicarán los datos de entrada de la fuente de datos bajo el parámetro **database.history.kafka.topic.**

```
name=sql-server-connection
connector.class=io.debezium.connector.sqlserver.SqlServerConnector
database.hostname=******
database.port=**
database.user=***
database.password=******
database.dbname=*****
database.server.name=integracionserver
table.whitelist=*******
database.history.kafka.bootstrap.servers=127.0.0.1:9092
#integracionserver.dbo.Tokens
database.history.kafka.topic=integracionserver.dbo.SwTrans_old
binary.handling.mode=base64
```

Figura 24 connector.properties

Es importante indicar que el nombre de los temas de Kafka tiene la siguiente estructura: serverName.schemaName.tableName, donde serverName es el nombre lógico del conector como se especifica con la database.server.name propiedad de configuración en el trabajador de

Kafka, schemaName es el nombre del esquema donde ocurrió la operación y tableName es el nombre de la tabla de la base de datos en la que ocurrió la operación.

Los eventos generados dentro de una transacción se detallan a continuación.

op: Identificador del tipo de evento.

I: Inserción

U: Actualización

D: Eliminación

R: Lectura

Las acciones que se producen durante la captura de los datos son las siguientes:

- Cuando son eventos de inserción y lectura se va considerar que el objeto before no contenga información y solo se componga en el objeto after.
- Cuando es el evento de eliminación, el objeto after estará en null.
- Cuando es el evento de actualización, en el objeto before estará la información antes de la modificación y en el objeto after por la cual estamos modificando.

Para más información de los archivos de configuración empleados para el proceso de tarjetas de débito de la institución financiera se puede consultar el Anexo 3.

5.2.2.6 CONSUMIDORES

Para el presente proyecto de tesis se describen dos tipos de consumidores en la suscripción hacia el tema que mantiene los flujos de transacciones capturadas en el proceso de Detector de cambios.

El primer consumidor es una herramienta que se encuentra dentro de Apache Kafka y visualiza en modo consola todos los registros capturados y sigue presentando resultados a medida que se escriben más registros en el tema en tiempo real pudiendo inicializarse por línea de comandos.

El segundo consumidor es una aplicación propia (microservicio) desarrollada bajo lenguaje Java que captura los flujos de datos generados por Debezium, genera el proceso de ETL, y almacena los resultados en la base de datos no relacional; todo en tiempo real.

5.2.2.6.1 CONSUMIDOR EN CONSOLA

Iniciar Consumidor

Dentro de la carpeta principal %KAFKA_HOME%\bin se ejecuta el siguiente comando que permite iniciar un consumidor suscrito al tema creado desde consola.

./kafka-console-consumer.sh --bootstrap-server localhost:9092 –topic integracionserver.dbo.SwTrans_old –from-beginning

Parámetros de configuración

- **--bootstrap-server: Establece** la dirección de los nodos o trabajadores.
- **--topic: Establece** el nombre del tema (En este caso " integracionserver.dbo.SwTrans_old ")
- **--from-beginning:** Mostrará el contenido del tema desde el inicio de los datos recibidos en el tema

5.2.2.6.2 CONSUMIDOR JAVA (MICROSERVICIO)

El microservicio desarrollado en lenguaje JAVA permite realizar la captura de los flujos de mensajes Kafka generados en Debezium a través del Conector Apache Kafka añadido en el proyecto MAVEN y el envío de los datos transformados hacia la base de datos no relacional.

Este microservicio es adaptable para diferentes temas dado que cuenta con una estructura que permite modificar el tema a escuchar a través de un archivo de configuración.

El código del microservicio desarrollado puede ser consultado en el Anexo 4 donde se presenta las librerías utilizadas y configuradas en este proyecto.

Por lo tanto, lo que hasta ahora era una cantidad de información sin utilidad se ha convertido en un conjunto de datos limpios y transformados que se pueden almacenar como documento JSON en la última fase del ETL: la fase de carga.

5.2.2.7 CARGA

Para el almacenamiento no relacional y servidor de búsqueda en tiempo real se descargó la última versión de Elasticsearch como herramienta seleccionada en el capítulo 3 para este objetivo. La versión obtenida en el momento de descarga es la 7.8.

Una vez descargado y descomprimido el archivo de Elasticsearch en la carpeta principal Kafka_home se procede a inicializarlo a través del siguiente comando desde la carpeta bin.

Iniciar Elasticsearch

Se ejecuta el siguiente comando para iniciar la base de datos no relacional seleccionada para este proceso:

./elasticsearch.sh &

El puerto por default donde se levanta el proceso es el siguiente http://localhost:9200/ teniendo el resultado que se presenta en la Figura 27.

```
{
  "name" : "linux-8rkv",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "seERD3rIRG65LhlFy5Ge1w",
  "version" : {
    "number" : "7.8.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "757314695644ea9a1dc2fecd26d1a43856725e65",
    "build_date" : "2020-06-14T19:35:50.234439Z",
    "build_snapshot" : false,
    "lucene_version" : "8.5.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```


Figura 25 Inicio Elasticsearch

5.2.3 TABLERO DE CONTROL

Dentro de este proceso se permite visualizar a través de gráficas en un tablero o panel como se encuentra el comportamiento de una organización de acuerdo a los indicadores de rendimiento definidos (KPI por sus siglas en inglés), permitiendo así a los usuarios tomar decisiones desde diferentes perspectivas del negocio.

5.2.3.1 INDICADORES CLAVES DE RENDIMIENTO (KPI)

Los KPI brindan información relevante del desempeño de una organización, ofreciendo a las empresas información si están o no encaminadas hacia sus objetivos establecidos.

Además, estos indicadores son especialmente útiles para la toma de decisiones cuando son utilizados de manera habitual pudiendo marcar una gran diferencia en el éxito de una empresa.

El objetivo de la presente tesis fue presentar una visión de los siguientes indicadores requeridos en la institución financiera:

- Los países con más uso de la tarjeta de débito.
- Los comercios y servicios digitales con más demanda con tarjeta de débito como medio de pago.

- Montos estimados del consumo de este servicio.
- Número de transacciones de compras con tarjeta de débito realizadas durante periodos de tiempo.

5.2.3.2 KIBANA

Para el tablero de control necesario para visualizar las gráficas interactivas en tiempo real la herramienta a utilizar es Kibana. Se descargó la última versión de Kibana Versión kibana-7.8.0-linux-x86_64.tar.gz y se colocó la misma descomprimida dentro del Kafka_home.

Iniciar Kibana

Para inicializar Kibana se accede a la carpeta bin y se ejecuta el siguiente comando.

./kibana.sh &

El puerto por default donde se levanta el proceso es el siguiente http://localhost:5601/ teniendo el resultado que se presenta en la Figura 28.

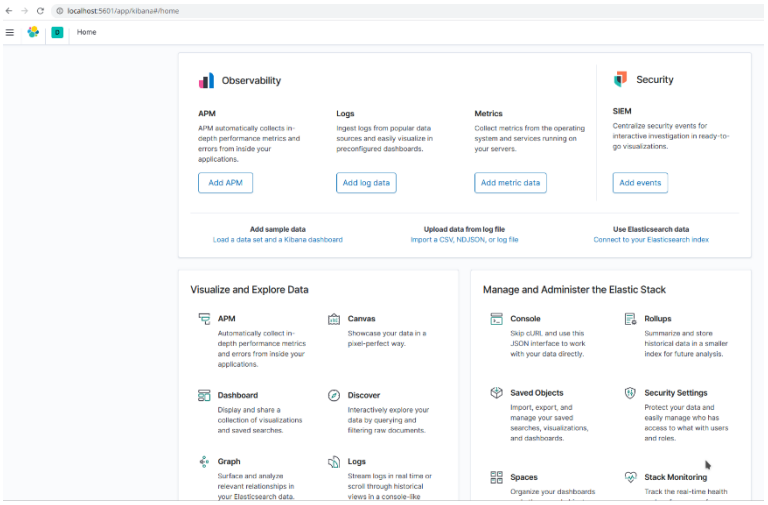

Figura 26 Kibana

En el Anexo 6 se puede revisar el proceso de configuración para la visualización de las transacciones de las tarjetas de débito propuesto como caso de estudio en la implementación de la arquitectura.

5.3 ANÁLISIS DE RESULTADOS

Merece la pena indicar que las pruebas realizadas a continuación fueron ejecutadas sobre ambientes de no producción en la institución financiera. Es importante recalcar además que, para simular el flujo masivo de transacciones, inicialmente se dio lectura de la tabla de una base de datos que mantiene las transacciones de las tarjetas de débito con corte al 1 de septiembre por lo que se tomó las transacciones del periodo del 01 a 31 agosto de 2021 como datos para el estudio. Posteriormente se creó un script que actúa simulando un productor de transacciones para demostrar su comportamiento en tiempo real como si fuera un procesamiento en puntos de venta o transacciones de comercio electrónico. Lo que se pretende con este script es demostrar el funcionamiento en tiempo real de la arquitectura propuesta.

Es importante resaltar que por motivo de acuerdos de confidencialidad de los datos solicitado por la institución financiera donde se aplicó la arquitectura propuesta, la información no puede ser expuesta en un repositorio público dado que contiene datos sensibles como números de cuenta, nombres, comercios, consumos, etc., de los socios evitando violar la privacidad de este tipo de información. Ahora bien, para demostrar el funcionamiento de la arquitectura planteada, la entidad

financiera accedió a generar una réplica de la base de datos de producción a una de pruebas, donde se presentan a continuación los resultados y gráficas obtenidas del procesamiento de la tabla de transacciones de tarjetas de débito durante un rango de tiempo seleccionado tomado como muestra.

De la tabla de transacciones de tarjetas de débito durante el periodo de prueba, se tienen los siguientes resultados:

1) *El top 5 de los países con más uso de esta forma de pago*. Como se aprecia en la figura, Ecuador encabeza esta lista al ser el país donde se utiliza mayoritariamente este medio de pago de tarjeta de débito de la institución financiera en comercios locales. Seguido se encuentra Estados Unidos e Inglaterra (Figura 29).

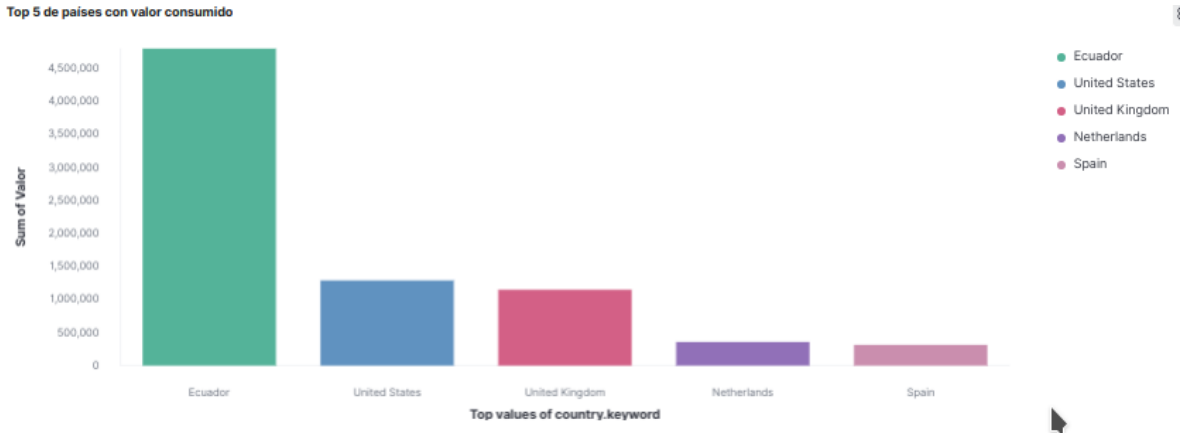

Figura 27 Top 5 de países con mayor consumo de compras con tarjeta de débito

2) El monto total de procesamiento en transacciones de tarjetas de débito como forma de pago en comercios y puntos de venta (POS) durante el mes de agosto 2021. La Figura 30 permite a los directivos visualizar el total procesado ya sea en POS (puntos de venta) o comercio electrónico con este medio de pago y la eficiencia de su uso entre los socios de la institución financiera.

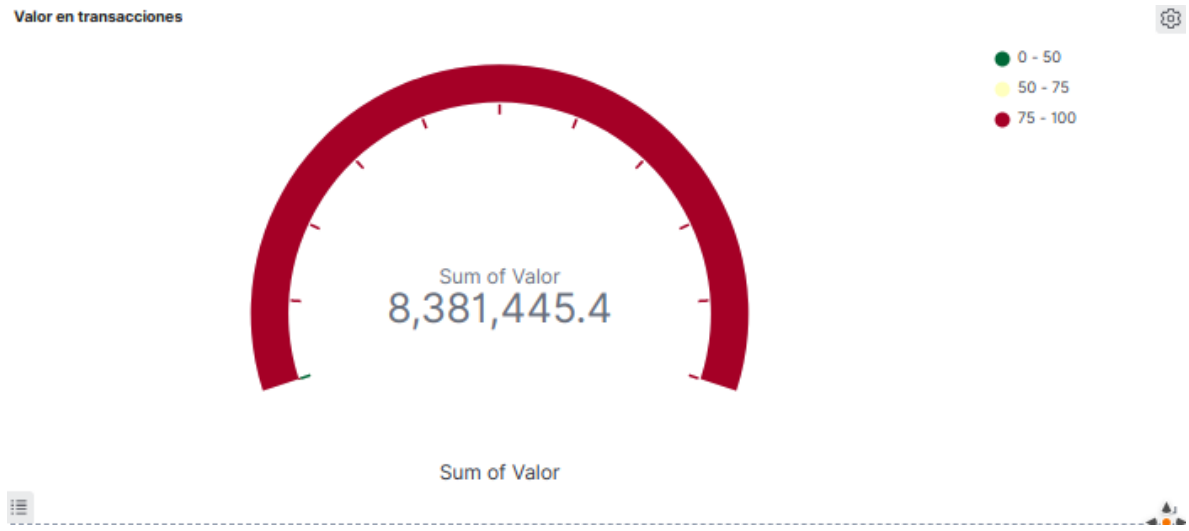

Figura 28 Monto total en compras con tarjeta de débito

3) El número de transacciones realizadas en comercios y puntos de venta (POS) durante el mes de agosto 2021. Al igual que la gráfica anterior permite dar una visión en tiempo real de la cantidad de transacciones que se realizan con tarjeta de débito (Figura 31) permitiendo en este tablero de control también poder agregar filtros y determinar la cantidad de transacciones efectuadas entre días semanas meses, etc.

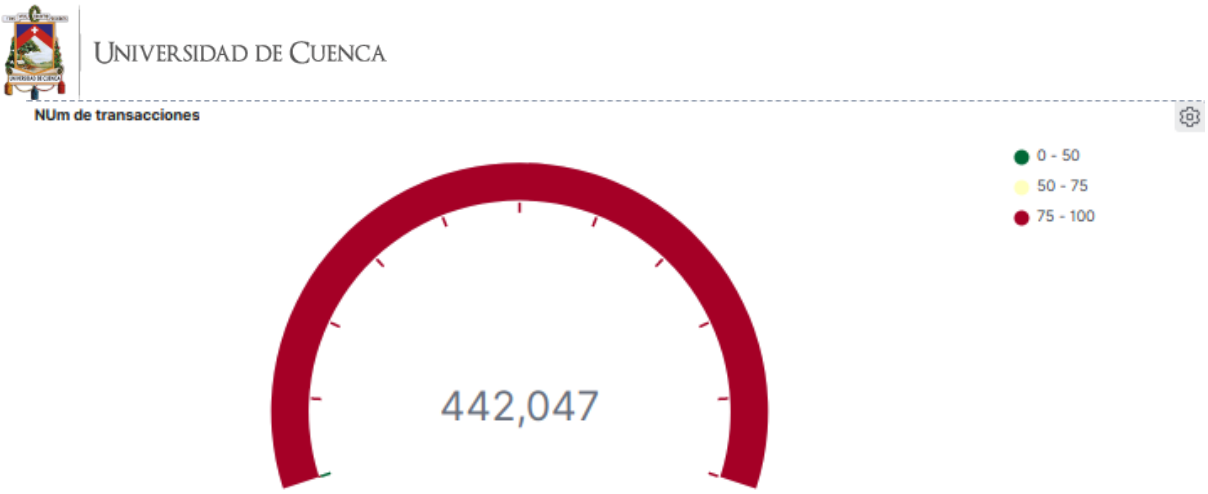

Figura 29 Número de transacciones realizadas con tarjeta de débito

4) El top 5 de comercios a nivel mundial donde la forma de pago con tarjeta de débito de la institución financiera es utilizada. Este indicador permite identificar los tipos de comercio donde se han efectuado compras con tarjeta de débito, generando una visión al área de mercadeo de la institución financiera para fortalecer o mejorar su uso a través de promociones, descuentos u otros que permitan ya sea nuevos comercios afiliarse al uso de los POS con los que cuenta la institución financiera generando un ingreso adicional para la empresas y por otro lado atraer nuevos socios que contraten el servicio de tarjeta de débito también agregando un valor a los ingresos de la entidad por el chip con la tarjeta que se emite. En la Figura 32 liderando se encuentran las Tiendas minoristas diversas y especializadas. Seguida se encuentra los Medios de productos digitales: Juegos. Las Tiendas de Comestibles también se encuentran en este listado. Para más detalle de los diferentes tipos de comercios que se utilizan actualmente dentro de los procesos de tarjetas de crédito/débito se puede acceder a la siguiente página de la franquicia VISA https://usa.visa.com/dam/VCOM/download/merchants/verified-by-visa-acquirer-merchant-implementation-guide.pdf.

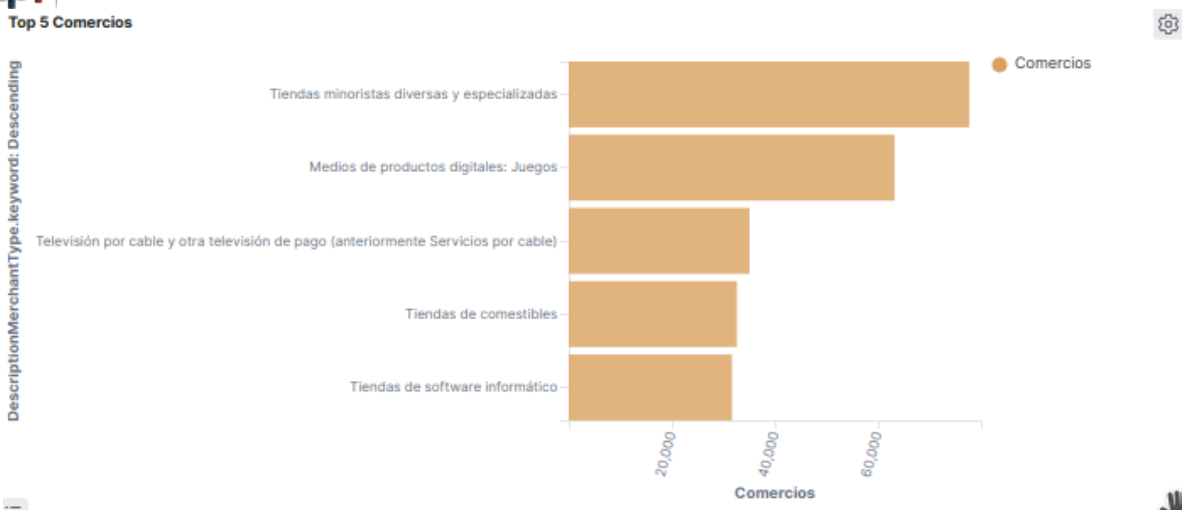

Figura 30 Top 5 de comercios con mayor número de compras con tarjeta de débito

6) Porcentajes a nivel de comercios de acuerdo al número de transacciones.

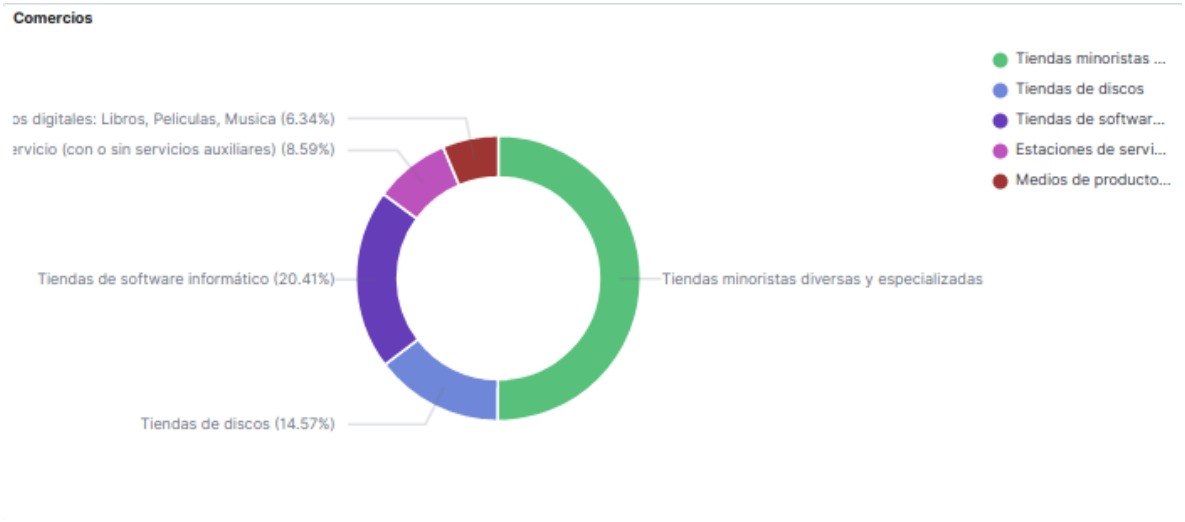

Figura 31 Porcentajes a nivel de comercios de acuerdo al número de transacciones

.

Una vez presentado el tablero de control a nivel general es posible generar además los gráficos drill and down que permiten ver los datos con mayor nivel de detalle y representan uno de los objetivos de la presente tesis. Por ejemplo, al seleccionar el país Ecuador en la figura 29 se presenta una tabla resumen donde se puede visualizar los 10 tipos de comercios en donde se han efectuado las compras de mayor a menor número de transacciones.

A continuación, en la tabla 9 se presenta este resumen de los tipos de comercios y al pulsar sobre cada uno se despliegan la descripción de los lugares de comercio como se puede apreciar en las tablas 10, 11, 12.

Ecuador

Top 50 unusual terms in DescriptionMerchantType.keyword	Count
Tiendas de comestibles	2,721
Estaciones de servicio (con o sin servicios auxiliares)	1,337
Droguerías y farmacias	914
Servicios de fax	646
Grandes almacenes	507
Lugares para comer y restaurantes	479
Restaurantes de comida rápida	394
Ferreterías	212
Tiendas de cosméticos	96
Drogas, propietarios de drogas y artículos diversos del farmacéutico	91
	8.804

Tabla 9 Top 10 de tipos de comercio en Ecuador

country.keyword: Ecuador × + Add filter

Tiendas de comestibles: Comercio

Tipo de Comercio	Count
ALMACEN GERARDO ORTIZ	226
GERARDO ORTIZ E HIJOS CI	120
GERARDO ORTIZ CIA L DEL	114
CORALRIO	129
MI COMISARIATO	100
HYPERMARKET MACHALA	107
GERARDO ORTIZ CIA LTDA	60
SUPERMAXI EL VERGEL LC 4	65
HYPERMARKET NORTE	53
SUPERMAXI DON BOSCO LC 4	51

Export: Raw Formatted

Tabla 10 Top 10 de lugares de comercio para Tiendas de Comestibles

Droguerías y farmacias: Comercio

Tipo de Comercio	Count
FYBECA	125
CORPDESFA	83
DIFARE SA	114
SANA SANA	65
PMZ FUXION BIOTECH EC	36
OMNILIFE DEL ECUADOR	40
FARMACIA POPULAR UNIDAS	27
FARMACIAS MIA	41
ORIFLAME DEL ECUADOR I	12
FARMASOL TOTORACOCHA	10

Tabla 11 Top 10 de lugares de comercio para Droguerías y Farmacias

Grandes almacenes: Comercio

Tipo de Comercio	Count
TIA S A	312
T VENTAS	25
ETAFASHION 9 DE OCTUBRE	13
CENTER PLAZA SAN BLAS	9
JUAN MARCET CUENCA	10
ETAFASHION MACHALA	9
PMZ PAY BILL BY TCK	6
SUPER STOCK	7
MERCANTIL TOSI S.A.	5
ETAFASHION MANTA	4

Tabla 12 Top 10 de lugares de comercio en Grandes Almacenes

Por su parte en Estados Unidos e Inglaterra las transacciones donde se han realizado mayor consumo son referente a comercios que ofrecen productos digitales como Juegos y en Tiendas de software Informático (Tabla 13). Algunos de los lugares de comercio se detallan en la Tabla 14 y 15 respectivamente.

Estados Unidos

Top 50 unusual terms in DescriptionMerchantType.keyword	Count
Medios de productos digitales: Juegos	2,825
Tiendas de software informático	2,764
Tiendas de discos	1,762
Servicios de programación informática, diseño de sistemas integrados y procesamiento de datos	748
Medios de productos digitales: Aplicaciones (Excluidos Juegos)	601
Marketing directo: comerciante de continuidad / suscripción	405
Servicios comerciales, no clasificados en otra parte	328
Librerías	155
Mercadeo directo - Comerciante de teleservicios entrantes	51

Tabla 13 Top 10 de tipos de comercio en Estados Unidos

Medios de productos digitales: Juegos: Comercio

Tipo de Comercio	Count
Google	2,596
PLAYSTATION NETWORK	237
PlaystationNetwork	49
NINTENDO *AMERICAUS	10
Xsolla *Twitch	6
Microsoft	19
Xsolla *Roblox	3
EPC*FORTNITE	7

Tabla 14 Top 10 de lugares de comercio en Medios de productos digitales como Juegos

Tiendas de software informático: Comercio

Tipo de Comercio	Count
Google	3,050
DIGITALOCEAN COM	11
DISCORD* GIFT-NITRO	3
DISCORD* NITROMONTHLY	3
WIX*Wix.Com, Inc.	3

Tabla 15 Top 10 de lugares de comercio en Tiendas de software informático

MAPA DE CALOR

Un mapa de calor permite representar a través de gráficas las zonas donde existe una mayor o menor presencia de condiciones definidas para determinar el éxito o fracaso de un determinado estudio. En el caso del uso de las tarjetas de débito y como se puede ver en la Figura 34, se aprecia los países donde su uso es mayoritario. A partir de esta gráfica y generando un zoom hacia cada uno de los países se puede observar las ciudades donde se han realizado transacciones, permitiendo así potencializar el medio de pago y determinar zonas donde se pueda emplear estrategias de expansión en el uso de las tarjetas de débito, generando mayor eficacia para la captación de nuevos socios.

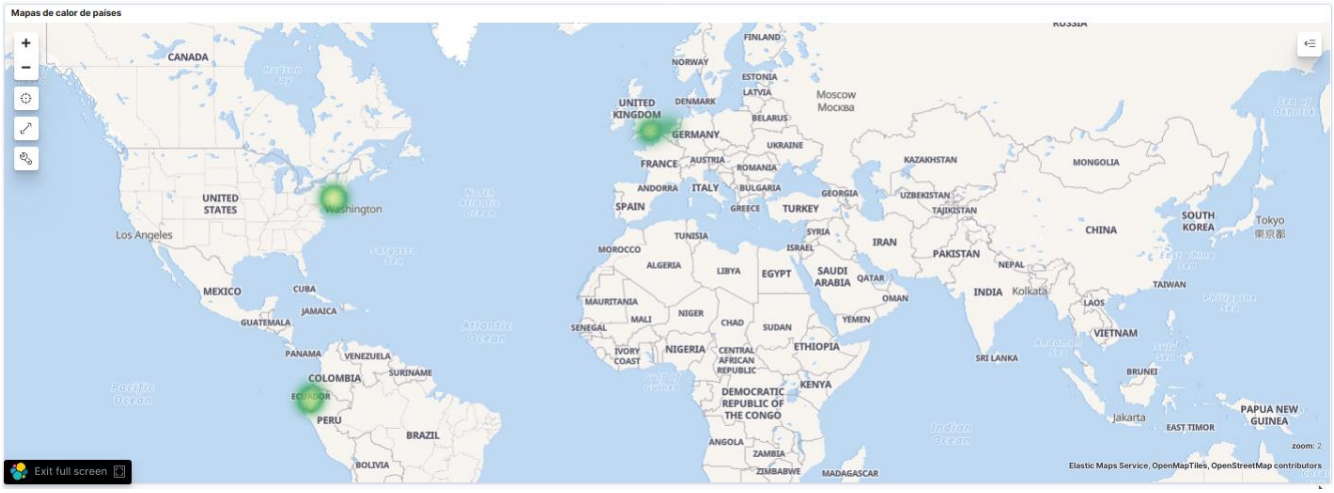

Figura 32 Mapa de calor de tarjetas de débito

CAPÍTULO 6 CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se han planteado algunos objetivos, mismos que han sido los pilares para el desarrollo del proyecto. En este sentido, este capítulo revisa el cumplimiento de cada objetivo y los hallazgos obtenidos; así mismo, se presentan los posibles trabajos futuros y el articulo aceptado referente al proyecto propuesto.

6.1 CONCLUSIONES

Considerando los objetivos planteados inicialmente, a continuación, se describen las conclusiones obtenidas por cada uno de ellos.

6.1.1 OBJETIVO GENERAL

El objetivo general de este trabajo fue *diseñar e implementar microservicios para la generación de un modelo de análisis de transacciones masivas de tarjetas de débito para una institución financiera.*

En lo que corresponde al objetivo general se puede concluir que éste fue cumplido en su totalidad. En el capítulo 5, se presentó las pruebas realizadas en el diseño de la arquitectura propuesta bajo una simulación de transmisión de transacciones generadas en POS y canales electrónicos en tiempo real hacia una herramienta de inteligencia de negocios. En consecuencia, la solución propuesta presenta los siguientes aportes:

- Se efectuó una revisión de trabajos de investigación relacionados a modelos de análisis de transacciones masivas en tiempo real enfocados a casos de estudio en el sector financiero. La revisión efectuada generó como conclusión que la mayor parte de propuestas enfocadas a procesamiento en tiempo real hace uso de herramientas tanto libre como de pago pero que NO contienen dentro de su funcionalidad el uso de microservicios. En la propuesta presentada en este trabajo se usaron microservicios para limpiar y convertir la información y permitir su visualización en herramientas de inteligencia de negocios que aporten valor a las organizaciones.

- Fue posible determinar del análisis de la literatura que, el uso de herramientas de código abierto (open source) pueden gestionar de manera eficiente miles de registros en tiempo real, aportando velocidad, rendimiento y optimización en búsquedas de grandes volúmenes de datos gracias a la tecnología NoSQL.

- La arquitectura propuesta está inspirada en el diseño de un modelo Kappa que permite la transmisión de flujos de datos en tiempo real. El diseño propuesto en este trabajo integra en tres (3) componentes claves denominados: i) Detector de cambios. ii) Núcleo. y iii) Tablero de Control, todas las funciones de un sistema informático que requiere gestionar grandes volúmenes de datos tiempo real.

- Para soportar el funcionamiento de cada uno de los componentes de la arquitectura, se realizó un análisis de diferentes herramientas de código abierto que permitan un correcto funcionamiento de la plataforma y una adecuada interacción entre cada componente. Además, en la fase de diseño de la arquitectura, se integró un modelo basado en microservicios a través de la gestión de eventos que usa Apache Kafka. También se diseñó el tablero de control que permita presentar el estado actual de diferentes transacciones realizadas en tiempo real.

- Fue posible determinar que la plataforma de mensajería y transmisión de datos que provee Apache Kafka, resulta muy útil dentro del ambiente financiero ya que permite una gestión eficiente de la información, bajos tiempos de latencia, y un escalamiento horizontal que brinda la posibilidad de manejar grandes volúmenes de transacciones y contar con una alta disponibilidad gracias al enfoque de tolerancia a fallas que maneja esta tecnología.

- En la fase de implementación, se describió con suficiente nivel de detalle las instalaciones/configuraciones realizadas sobre la arquitectura. La implementación permitió demostrar que la arquitectura puede ser usada para gestionar uno de los servicios más utilizados dentro de una institución financiera; el uso de tarjetas de débito para compras en línea o en puntos de venta. Además, en los resultados descritos en el Capítulo 5, se establecieron algunos indicadores, los cuales puedan ser aplicados para conocer el funcionamiento de la institución. Por ejemplo, la cantidad de transacciones, comercios y negocios donde se realizan habitualmente las compras con tarjetas de débito. Esta información puede resultar muy útil para la toma de decisiones.

Para lograr el objetivo general, se plantearon seis objetivos específicos, los cuales se analizan a continuación.

6.1.2 OBJETIVO ESPECÍFICO 1

Diseñar e implementar microservicios para capturar, transformar y cargar el flujo de datos en tiempo real de transacciones de tarjetas de débito de una institución financiera.

Este objetivo se cumplió en su totalidad al diseñar e implementar en un ambiente de pruebas una arquitectura que hace uso de tecnologías como microservicios y un bróker de mensajería para manejo de flujos de eventos en tiempo real. La arquitectura propuesta se diseñó en base al análisis de otro tipo de arquitecturas de tiempo real, integrando los componentes o elementos existentes en la literatura en tres componentes funcionales que interactúan entre sí y se describen a continuación:

a) Detector de Cambios: Este componente fue diseñado para que a través de una configuración sencilla se pueda activar el seguimiento o captura de los datos que están cambiado sobre un almacén de datos mediante logs transaccionales. Esto permite que, a diferencia de otros enfoques de captura de cambios esta solución no sea intrusiva, puesto que de forma automática detecta los cambios que se producen en el repositorio de datos y transmite los eventos generados a otras aplicaciones para poder actuar de manera inmediata con la información.

b) Núcleo: componente principal de la arquitectura propuesta dado que permite actuar como un puente entre los datos relacionales y su envío hacia una base de datos no relacional. Este componente fue diseñado en base a la arquitectura Kappa que permite la transmisión en tiempo real de la información, su funcionalidad se basa en el uso de un microservicio que genera en un solo mecanismo todo el proceso de limpieza, transformación, y carga de los datos a diferencia de otras arquitecturas que presenta estos procesos en elementos independientes. El microservicio desarrollado fue diseñado para que este sea independiente, escalable y parametrizable permitiendo almacenar en la base de datos NoSQL, información de calidad.

c) Tablero de Control: Este componente presenta en tiempo real los datos de las transacciones del medio de pago seleccionado almacenadas en la base de datos NoSQL a través de visualizaciones gráficas de inteligencia de negocios. Los indicadores del funcionamiento para este caso de estudio fueron diseñados a medida con el personal de la institución financiera, buscando presentar la información en visualizaciones simples pero que generen una visión completa del funcionamiento de las tarjetas de débito. Para presentar la información se seleccionaron los tipos de gráficas que permitan a través de su diseño presentar los datos de forma resumida como una mejor alternativa para potenciar el medio de pago de este caso de estudio. Por ejemplo, para presentar las zonas geográficas donde se utiliza las tarjetas de débito de la institución financiera se utilizó un mapa de calor, para desglosar la información de tipos de comercios y lugares donde se efectuó su uso se diseñó una tabla que permita una navegación bajando un nivel de detalle (drill and down).

6.1.3 OBJETIVO ESPECÍFICO 2

Crear y configurar una base de datos NoSQL para almacenar la información de las transacciones de tarjetas de débito de una institución financiera.

En el caso de este trabajo se optó por el uso de una base de datos NoSQL, pues este tipo de gestor de datos permite un diseño flexible de la información, escalabilidad para poder agregar de una manera eficiente y rápida nuevos recursos, y optimización de consultas para grandes volúmenes de

transacciones como las generadas dentro de una institución financiera. Ahora bien, con el fin de cumplir este objetivo, se inició determinando los parámetros que debería tener la herramienta de código libre para el almacenamiento NoSQL necesaria para la arquitectura propuesta. Con estos parámetros se efectuó un análisis de las herramientas disponibles que aporten además el manejo de información en tiempo real. Una vez seleccionada la herramienta a utilizar se procedió a instanciar la misma para su uso dentro de la comunicación del microservicio.

Es importante añadir que el uso de una base de datos NoSQL trajo consigo algunos problemas a resolver. Por ejemplo, para el proceso de conversión de datos utilizado dentro del microservicio es necesario manejar la relación entre los códigos de comercios, países, tipos de comercios y su respectiva descripción. En una base de datos relacional, este proceso puede ser solventado a través de la combinación de tablas (usando un campo común entre ellas) que permita obtener la información necesaria, pero con tiempos significativos de procesamiento. Sin embargo, en una base de datos NoSQL no siempre es posible el uso de este tipo de operaciones, principalmente porque su uso puede disminuir significativa el rendimiento. La solución propuesta en este trabajo, fue la creación de índices independientes que permiten gracias a la funcionalidad de búsquedas de texto optimizar el rendimiento en consultas y la recuperación de la información.

6.1.4 OBJETIVO ESPECÍFICO 3

Evaluar y seleccionar las técnicas de filtrado y de análisis de datos, aplicables a la información que permita la extracción de información relevante para la toma de decisiones.

Con el fin de cumplir este objetivo, se realizó inicialmente un análisis de las técnicas de filtrado de datos que se podrían utilizar para este estudio, usando la información actual de las transacciones de tarjetas de débito que llegan desde los sistemas origen. Para cumplir este proceso se efectuó una lluvia de ideas junto con el personal de tecnología de la entidad financiera, buscando definir los requisitos que necesitaba cumplir las gráficas de inteligencia de negocios, esto es, presentar información relevante una vez aplicada las técnicas de filtrado de datos. A continuación, se procedió a seleccionar y desarrollar las técnicas dentro del microservicio para generar información que sirva de entrada a la base de datos NoSQL y aporte valor para la toma de decisiones. Entre las técnicas de filtrado o reducción de datos que se analizaron dentro de la literatura se encuentran: transformación de datos y normalización, integración, limpieza de ruido e imputación de valores perdidos, sin embargo, para la arquitectura propuesta en este estudio, se implementaron únicamente tareas pertenecientes a los procesos de transformación de datos y limpieza de ruido:

- Eliminación de registros que contengan valores vacíos o nulos
- Eliminación de registros de transacciones realizadas en cajeros automáticos generadas originalmente en los flujos de eventos y que llegan a la base de datos que almacenan los datos del medio de pago de tarjetas de débito.

- Conversión de datos a través de una librería que detecta las tramas ISO8583, el estándar manejado en el uso de las tarjetas de crédito/débito y decodifica a campos necesarios que generen valor a la información.

Finalmente, los datos limpios y transformados fueron almacenados en la base de datos NoSQL para su uso en el tablero de control.

6.1.5 OBJETIVO ESPECÍFICO 4

Identificar los indicadores claves para la obtención de información que será utilizada en la elaboración de cuadros de mando a nivel estratégico.

Para cumplir este objetivo se recurrió al uso de entrevistas con el personal de operaciones de medios de pago y tecnología dentro de la institución financiera, identificando las métricas e indicadores a presentar dentro del tablero de control para el proyecto propuesto. Los indicadores de rendimiento seleccionados se encuentran bajo el modelo SMART (Especifico, Medible, Alcanzable, Relevante, Tiempo Libre) buscando medir el desempeño de uno de las formas de pago seleccionadas dentro de la institución financiera. El número de indicadores para este estudio además fue establecido desglosando con el personal los objetivos que se desean medir dentro del funcionamiento de las tarjetas de débito. Si bien para otros entornos o formas de pago se pueden llegar a definir nuevos KPIS bajo el mismo contexto utilizado es importante en términos simples que los indicadores contribuyan a dar una visión general a las organizaciones si están en el camino para lograr los objetivos estratégicos planteados.

Todos los indicadores que dan cumplimiento a este objetivo específico están descritos en el capítulo 5 en la sección de análisis de resultados.

6.1.6 OBJETIVO ESPECÍFICO 5

Crear un informe con resultados de los indicadores seleccionados.

Se puede concluir que el objetivo se ha logrado cumplir, pues para crear el informe se ha implementado un tablero de control que presenta los resultados obtenidos durante el procesamiento de la información gestionada en tiempo real en la institución financiera. Merece la pena destacar que actualmente el acceso a la información sobre uso de los medios de pagos involucra altos tiempos de procesamiento, procesos específicos sobre las bases de datos, o tareas manuales con un alto grado de complejidad y propensa a errores. Sin embargo, la arquitectura propuesta ha traído una mejora significativa, dado que la información se presenta a través de un informe en tiempo real, mostrando los resultados de una manera ágil y eficiente para la toma de decisiones. Además, la arquitectura es suficientemente flexible como para permitir incorporar nuevos indicadores (que involucren otros

medios de pago o servicios) que permitan la toma de acciones en diferentes departamentos de la organización.

Se espera que el uso de propuestas como las planteadas en este trabajo puedan ayudar a conocer mejor el estado situacional de aquellas instituciones que manejan este tipo de transacciones. Por ejemplo, uno de los indicadores solicitados para este estudio fue conocer el número de transacciones y montos que se están utilizando con una tarjeta de débito. Esta información podría ser usada por el departamento de marketing para impulsar o promover la afiliación del servicio entre los socios de la institución financiera a través de promociones o incentivos. Otro ejemplo del funcionamiento de un informe basado en indicadores gestionados con datos en tiempo real, es la obtención de los tipos de comercio en donde se efectúan las compras con el medio de pago, otorgando a la entidad oportunidades para la creación de nuevos servicios que fidelicen a los socios y maximicen las ganancias de la organización. Esta misma información puede alertar sobre un posible fraude que se produzca con este tipo de transacciones.

6.1.7 OBJETIVO ESPECÍFICO 6

Diseñar representaciones gráficas de inteligencia de negocios que permitan visualizar el comportamiento de los indicadores.

Hoy en día, la combinación de herramientas de patrones de mensajería (procesamiento en tiempo real) e inteligencia de negocios aplicado a sectores donde el flujo de datos es relevante y continuo es de gran interés para las empresas. Toda la información procesada con esta combinación de técnicas tiene un valor positivo, porque facilita identificar mejor al cliente, proporcionando productos con más calidad, detectar problemas de los servicios ofertados en una organización, lugares donde se pueden introducir nuevos productos y mejora la proyección de los resultados a través de mecanismos de geo posicionamiento. Es así que este objetivo se ha cumplido en su totalidad, mediante la creación de una dashboard o panel de control que visualiza los indicadores diseñados en el Objetivo 4 mediante gráficas resumen, que permiten conocer el estado actual del funcionamiento del servicio o medio de pago de las tarjetas de débito (usado en este caso de estudio- Ver Anexo 6).

Los tipos de gráficas al igual que los indicadores fueron diseñados luego de un proceso de entrevista realizado al personal de la organización financiera. La meta fue diseñar gráficas que muestren la información de una manera accesible, resumida y comprensible para la toma de decisiones. De ello resulta necesario concluir que la integración entre herramientas de captura de flujos de datos en tiempo real y gráficas de inteligencia de negocios es factible. Este trabajo es una muestra que una arquitectura flexible puede facilitar la instalación, configuración y diseño de tableros de control que se puedan adaptar a las necesidades de cada proyecto o entorno.

Finalmente, se concluye que la arquitectura propuesta se puede tener en cuenta en el futuro para nuevos servicios o entornos como se describe a continuación.

6.2 TRABAJOS FUTUROS

El diseño propuesto en este trabajo de titulación estuvo orientado a probar que la integración de diferentes herramientas y tecnologías es factible para procesar datos en tiempo real y ejecutar la toma de decisiones. Como una prueba de concepto, se diseñó e implementó una arquitectura que involucra un único nodo y una única partición. Además, para probar el funcionamiento de la arquitectura se simuló un escenario de pruebas que incluyó la simulación de flujo de eventos generados en transacciones de tarjetas de débito en sistemas origen como POS (Puntos de venta) y comercio electrónico. Esta propuesta puede ser el comienzo de las futuras implementaciones que permitan la comunicación en tiempo real de bases de datos relacionales y no relacionales; siendo expandibles no solo a la creación de soluciones orientadas a institución financieras (como este caso de estudio) sino hacia otros entornos que requieren dentro de sus herramientas tecnológicas una forma de poder realizar visualizaciones del funcionamiento de la organización a través de inteligencia de negocios en tiempo real. Se plantea además como trabajos futuros, adaptar el diseño presentando en un clúster, que permita el uso de varios nodos de Apache Kafka en un ambiente de producción bajo entornos que manejan grandes volúmenes de datos, potenciando así las características principales de esta tecnología como son la tolerancia a fallas y el escalamiento horizontal para medir su rendimiento. Además, es importante recalcar que al manejar una arquitectura basada en microservicio de acuerdo a la arquitectura presentada, se podría plantear como trabajo futuro poder adaptar el diseño a nuevos microservicios que interactúen con diferentes contextos o entornos para la generación de información útil para toma de decisiones en tiempo real.

6.3 DIFUSIÓN DE RESULTADOS

Como parte de los requisitos para el proceso de graduación de la maestría en “Gestión Estratégica a de Tecnologías de la Información segunda cohorte”, se ha realizado el aporte de un artículo científico presentado a continuación.

6.3.1 ACEPTACIÓN ARTÍCULO CIENTÍFICO

Como parte del trabajo realizado se elaboró el artículo “**Proposal of a real time microservice architecture applied to business intelligence for the financial sector**”, el cual fue aceptado y presentado en la conferencia “International Conference on Applied Technologies ICAT 2021”. El artículo presenta el diseño propuesto de una arquitectura para la transmisión de transacciones

generadas en una institución financiera en tiempo real a través de un microservicio para la visualización en tableros de control de inteligencia de negocios.

ANEXO 1: CONFIGURACIÓN CDC EN SQL SERVER 2019

La configuración siguiente es la realizada para el proceso de captura de cambios en los datos para la base de datos y tabla respectiva de procesamiento de tarjetas de débito dentro de la institución financiera.

Código 1 Script para activación de base de datos y tabla de transacciones de tarjetas de débito

```
/*
SCRIPTS PARA HABILITAR CDC SQL SERVER - MANEJO DE TARJETAS DE DEBITO
*/
-- ====
-- Enable Database for CDC
-- ====
USE STAG2
GO
-- Activar change data capture

EXEC sys.sp_cdc_enable_db
GO
--Crear PK
ALTER TABLE dbo.SwTrans_old ADD IdTransaccion INT IDENTITY(1,1)
ALTER TABLE dbo.SwTrans_old
ADD CONSTRAINT PKIDSwTrans_old
PRIMARY KEY(IdTransaccion);
```

```
--Activar Tabla

USE STAG2

GO

EXEC sys.sp_cdc_enable_table

@source_schema = N'dbo', --esquema de la base de datos

@source_name = N'SwTrans_old', --nombre de la tabla

@role_name = N'cdc_Admin',

@capture_instance = 'dbo_SwTrans_old',

@filegroup_name = N'PRIMARY',

@supports_net_changes = 1

GO
```

En el caso que se desee desactivar dicha funcionalidad CDC se puede ejecutar el siguiente script (Código 2).

Código 2 Script para desactivación de base de datos y tabla de transacciones

```
--Desactivar captura de datos sobre la base de datos


EXEC sys.sp_cdc_disable_db

GO


--Desactivar captura de datos sobre la tabla


USE STAG

GO
```

```
EXEC sys.sp_cdc_disable_table
@source_schema = N'dbo',
@source_name = N'TOken',
@capture_instance = N'dbo_SwTrans_old'
GO
```

Para realizar la comprobación y rastreo de los eventos generados en los datos capturados se puede aplicar el código 3.

Código 3 Script para comprobación de funciones habilitadas bajo CDC

```
--Listar bases de datos para determinar si se encuentran activas
SELECT name, is_cdc_enabled FROM sys.databases;
--Para ver si una tabla esta habilitada
SELECT name, is_tracked_by_cdc FROM sys.tables;
--Ver cambios realizados
declare @begin_lsn binary(10), @end_lsn binary(10)
-- get the first LSN for customer changes
select @begin_lsn = sys.fn_cdc_get_min_lsn('dbo_SwTrans')
-- get the last LSN for customer changes
select @end_lsn = sys.fn_cdc_get_max_lsn()
-- get net changes; group changes in the range by the pk
select * from cdc.fn_cdc_get_all_changes_dbo_SwTrans(
@begin_lsn, @end_lsn, 'all');
--Resumen de todo lo que se encuentra activado dentro de la tabla
EXECUTE sys.sp_cdc_help_change_data_capture
@source_schema = 'dbo',
```

```
@source_name = 'SwTrans_old';
GO
```

Dentro del proceso de activación de captura de cambios en la base de datos SQL SERVER se presentó un problema que se encuentra detallado a continuación.

Problema Encontrado

Uno de los problemas generados al intentar activar CDC para la base de datos SQL SERVER 2019 sobre la sobre la cual se va a trabajar para capturar las transacciones es que requiere de permisos de propietario de la base de datos para ejecutar la acción de activación de CDC:

Inconveniente:

El error que se produce al ejecutar el comando de habilitación de CDC es el siguiente:

The error returned was 15517:

'Cannot execute as the database principal because

the principal "dbo" does not exist,

this type of principal cannot be impersonated,

or you do not have permission.

Solución:

Para resolver el problema, se solicita cambiar la base de datos a la cuenta que maneje privilegios de administrador y luego habilitar CDC como se muestra a continuación.

```
EXEC sp_changedbowner 'sa'
EXEC sys.sp_cdc_enable_db
```

ANEXO 2: DEBEZIUM CONECTOR SQL SERVER

El conector Debezium SQL Server captura los cambios a nivel de fila que se producen en los esquemas de una base de datos de SQL Server. Para la implementación de este trabajo de tesis se descargó el siguiente conector disponible **debezium-connector-sqlserver-1.2.0.CR1-plugin.tar.gz**

Se inicia copiando los jars que se encuentran dentro del conector descomprimido Debezium en una nueva carpeta (Figura 35) que mantendrá los plugins necesarios para el funcionamiento de la integración de manera que se tiene el siguiente resultado en la carpeta /opt/kafka/plugins/ creada.

Esta carpeta de plugins será referenciada desde el nodo Kafka para la captura de los datos y el envío en flujo de mensajes Kafka.

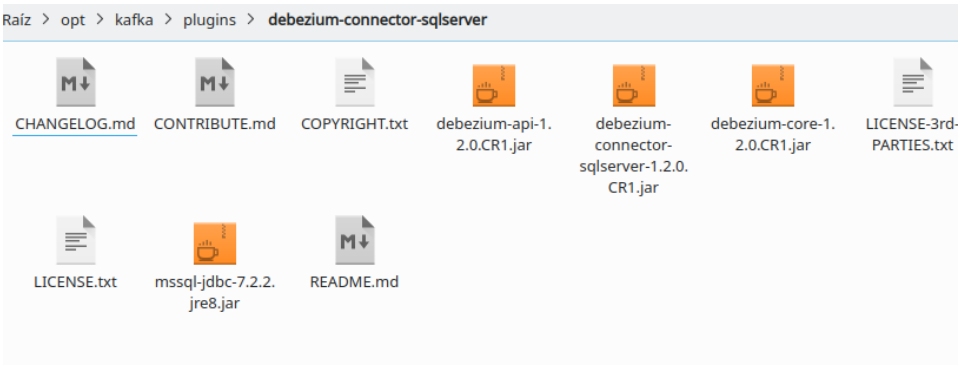

Figura 33 Contenido carpeta conector debezium sql server

Dentro del proceso de captura de los datos y conversión a flujos de mensajes Kafka proporcionado por Debezium se detectó que no se genera correctamente el formato del campo varbinay que contiene la trama ISO8586 de la tarjeta de débito.

Problema Encontrado

DECODIFICACIÓN CAMPOS VARBINARY SQL SERVER 2018 – DEBEZIUM

Inconveniente

Uno de los campos requeridos dentro de la propuesta planteada para este trabajo dentro de la base de datos de transacciones de tarjetas de débito es el campo ORGDATA. Este campo contiene

información adicional de la tarjeta de débito. Como se describió anteriormente, el procesamiento de tarjetas de débito/crédito se la hace a través de tramas ISO8583. Esta trama ISO8583 se encuentra almacenada en el campo ORGDATA dentro de la base de datos SQL SERVER en formato varbinary. Cuando se procedió a realizar su respectiva captura en el consumidor JAVA (microservicio), la codificación de la información del campo no correspondía a un formato claro.

Solución

Para verificar la codificación establecida dentro de la información del campo que se presenta en el consumidor JAVA, fue necesario entrar al código fuente del conector Debezium. De esta manera se detectó que dentro de sus métodos se realiza la codificación del texto varbinary en formato base 64 y hexadecimal. Aplicando así un decodificador online (Figura 36) se pudo capturar la data correcta almacenada en la base de datos relacional y proceder a implementar este tipo de decodificación dentro del microservicio.

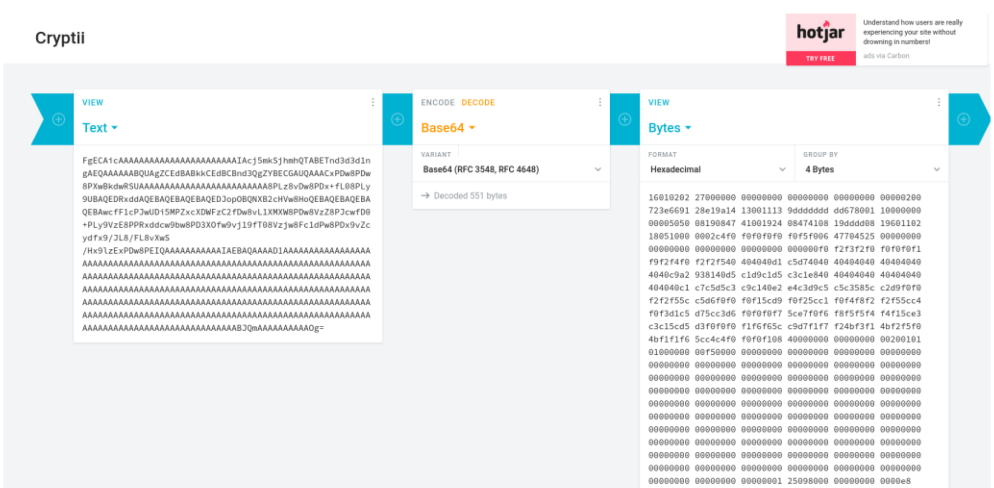

Figura 34 Decodificador online

ANEXO 3: CONFIGURACIÓN DE APACHE KAFKA

Para la configuración del bróker de mensajería como se presentó en la sección de Apache Kafka se requiere dos archivos de configuración que permiten generar la comunicación entre Debezium y Apache Kafka para capturar los eventos de cambio. A continuación se presenta en las figuras 37 y 38 el archivo del trabajador de un nodo Kafka (worker.properties) y el conector hacia la base de datos con CDC (connector.properties) respectivamente.

```
offset.storage.file.filename=/tmp/connect211.offsets
bootstrap.servers=127.0.0.1:9092
offset.flush.interval.ms=1000
rest.port=10082
rest.host.name=127.0.0.1
rest.advertised.port=10082
rest.advertised.host.name=127.0.0.1
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
plugin.path=/opt/kafka/plugins/
#If kafka is TLS authenticated, uncomment below lines.
#security.protocol=SSL
#ssl.truststore.location=/tmp/kafka.client.truststore.jks
#producer.security.protocol=SSL
#producer.ssl.truststore.location=/tmp/kafka.client.truststore.jks
```

Figura 35 worker.properties

```
name=sql-server-connection
connector.class=io.debezium.connector.sqlserver.SqlServerConnector
database.hostname******
database.port=**
database.user=***
database.password******
database.dbname=*****
database.server.name=integracionserver
table.whitelist=*******
database.history.kafka.bootstrap.servers=127.0.0.1:9092
#integracionserver.dbo.Tokens
database.history.kafka.topic=integracionserver.dbo.SwTrans_old
binary.handling.mode=base64
#transforms convert
```

Figura 36 connector.properties

En el archivo worker.properties se estableció las propiedades key.converter.schemas.enable y value.converter.schemas.enable en false(el valor predeterminado es true), esto conlleva que solo se transmiten los datos de las transacciones, sin los datos del esquema de la base de datos reduciendo así la sobrecarga de información para aplicaciones que no necesitan manejar el esquema. Por ejemplo:

Un tema Kafka que escucha los datos enviados por CDC a través de Debezium usando el conector de origen, tiene el siguiente formato:

```
{
  "Message": {
    "schema": {
      "type": "struct",
      "fields": [
        ...
      ],
      "optional": true,
      "name": ""
    },
    "payload": {
      "op": "u",
      "ts_ms": 1465491411815,
      "before": {
        "id": 1004,
        "first_name": "Name1"
       },
      "after": {
        "id": 1004,
        "first_name": "Name2"
      },
      "source": {
        "db": "****",
        "table": "*****",
        ...
        "query": "*****"
      }
    }
  }
}
```

Donde a través de la siguiente configuración empleada en la Figura 39 se obtiene la información únicamente del payload o carga útil donde se mantienen los datos de las operaciones (lectura, inserción, modificación, eliminación) de los registros de la tabla de datos de las transacciones de tarjetas de débito.

```
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter
```

Figura 37 Configuración worker.properties

Teniendo así el siguiente resultado obtenido de una de las transacciones de tarjetas de débito:

{"before":null,"after":{"FechaSwitch":1583884800000,"HoraSwitch":"170201595","Protocolo":"0","MsgId":"0311044062200000420847","TieneResp":1,"Tarjeta":"000477044XXXXXX2225","NormalOReverso":0,"Token":"190423235160312729","Transaccion":0,"TipoCtaOrg":0,"TipoCtaDest":0,"Valor":29.0,"TerminalType":1,"TerminalID":"ZXFACMGL","TerminalDate":1583884800000,"TerminalTime":"170201","FechaContable":1583884800000,"FechaRespuesta":1583884800000,"HoraRespuesta":"170202735","RespCode":62,"OrgLink":"CREDIMAT","DestLink":"JEP","FechaEmisor":1583884800000,"FechaReplica":null,"MTI":"100","Moneda":"840","Pais":"372","AcquirerCode":"420847","IssuerCode":"477044","DispIDTran":"ZXFACMGL","LugarTran":"SHOPIFY* 76343063
+353800808523IE","MerchantType":"5734","PosEntryMode":"012","PosConditionCode":"59","ValorParcial":"AA==","ReferenceNumber":"440622","LecturaDeBanda":0,"TransConPin":0,"NumAutoriza":"","monedaOrigen":"840","valorOrigen":"BGzQ","tasaCambio":"JxA=","codigoEntidad":"001","valorSurcharge":"AA==","ProtocoloOut":"0","SystemCodResp":1048,"codigoEntidadOrigen":"000","IndicadorEMV":"N","RedOrigen":3,"RedDestino":0,"IDDispositivo":"ZXFACMGL","IDMovimiento":"AA==","Milisegundos":1157,"FechaNotify":null,"Comision":"AA==","ServiceCode":"","AccountDebit":"","AccountCrebit":"","NUT":"","FechaExpiracion":"","RRN":"007122440622","Id_Transaccion":8939095},"source":{"version":"1.2.0.CR1","connector":"sqlserver","name":"integracionserver","ts_ms":1602967033562,"snapshot":"last","db":"STAG","schema":"dbo","table":"SwTrans","change_lsn":null,"commit_lsn":"00000281:00004ad8:0001","event_serial_no":null},"op":"r","ts_ms":1602967033562,"transaction":null}

ANEXO 4: MICROSERVICIO PARA TARJETAS DE DÉBITO

El consumidor desarrollado bajo lenguaje de programación Java es un microservicio que permite escuchar los registros insertados en la base de datos a través del proceso CDC que escucha las transacciones de tarjetas de débito y llevarlos hacía la base de datos no relacional Elasticsearch. Con esta premisa se inicia creando un proyecto Maven donde se agregaron las siguientes dependencias para agregar Apache Kafka bajo el archivo pom (Código 4).

Código 4 Dependencia de Maven para el conector de Apache Kafka

```
<dependencies>
    <dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-streams</artifactId>
  <version>1.0.2</version>
</dependency>
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
```

Se crearon además los siguientes archivos de propiedades dentro del proyecto que manejan los datos necesarios para conexiones tanto de Kafka como Elasticsearch, así con los campos sobre los cuales se realizara la conversión al tipo correspondiente bajo la premisa de extracción, transformación y carga. El parámetro topic del archivo kafka.properties mantiene el tema al cual se está suscribiendo el microservicio para escuchar los eventos generados en el trabajador Kafka (worker.properties) y que estarán sujetos al proceso ETL.

kafka.properties

topic=integracionserver.dbo.SwTrans_old

bootstrapserver=localhost:9092

grupo=SwTrans_old

readoptions=c,r

fields=MerchantType:elastic;OrgLink:string;DestLink:string;MTI:int;Valor:float;FechaSwitch:long;IdTransaccion:int;IssuerCode:string;LugarTran:elastic;Pais:elastic;PosEntryCode:string;PosConditionCode:string;Milisegundos:int;OrgData:iso8583

fieldsconversionelastic=Pais

fieldsconversionsdatetime=FechaSwitch

fieldsconversionstime=HoraSwitch

formatdatetime=dd/MM/yyyy HH:mm:ss

elasticsearch.properties

ip=localhost

port=9200

index=debitc

document=_doc

iso8583.properties

fieldsiso8583=PAN:0;ProcCode:1

El consumidor obtiene datos en paralelo y para optimizar el proceso utiliza un puntero que marca donde empiezan los mensajes. Además, está diseñado para ejecutarse en un hilo y con una comunicación asíncrona.

El código dentro del consumidor (Código 5) comienza definiendo el comportamiento de la captura de los registros y el formato de deserializacion, es decir, para interpretar/deserializar las claves y valores del mensaje se utiliza StringDeserializer. Además para la propiedad auto.offset.reset que permite determinar si comenzar desde el principio del tema o el final de la captura. Para esta propuesta fue definido desde el inicio dado que se va a empezar realizando una lectura de todo lo que se ha cargado previamente en la tabla de transacciones y así poder generar una mejor toma de decisiones debido a la cantidad de información.

Código 5 Propiedades de Configuración de un consumidor Kafka

```
Properties configProperties = new Properties();

    configProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, this.bootstrap);

    configProperties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);

    configProperties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);

    configProperties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    configProperties.put(ConsumerConfig.GROUP_ID_CONFIG, this.groupid);

    configProperties.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");
```

El código 6 define como se inicializa el proceso de lectura de registros a través de clases Kafka que permiten adquirir la información proveniente del flujo de datos.

Class KafkaConsumer<K,V>: Se utiliza para inicializar el Consumidor Kafka. Requiere de un conjunto de parámetros Key-Value que serán donde se guardan tanto el número de identificación de cada mensaje, así como su contenido.

Class ConsumerRecords<K,V>: Se encarga de "recibir" los mensajes provenientes del flujo de datos.

Class ConsumerRecord<K,V>: Esta clase construye una lista de Mensajes de acuerdo a cada iteración.

Código 6 Consumidor Kafka para procesamiento de registros

```
kafkaConsumer = new KafkaConsumer<String, String>(configProperties);

kafkaConsumer.subscribe(Collections.singletonList(this.topicName)); //Toma el topic o tema kafka del cual desea obtener las peticiones

    /*

     * Al producir, si no especifica explícitamente una partición, el productor seleccionará una automáticamente de su tema.


                                En su consumidor, si está suscrito a su tema, puede buscar el inicio de todas las particiones a las que su consumidor está asignado actualmente usando:

     */

kafkaConsumer.seekToBeginning(kafkaConsumer.assignment());


while (true)

{

ConsumerRecords<String,String> records = kafkaConsumer.poll(100);


for (ConsumerRecord<String, String> record : records)

{

 record.key(), record.value(), record.partition(), record.offset());
```

TRANSFORMACIONES

Dentro de este proceso se evita información que no aporte valor para el objetivo del proyecto como datos vacíos, nulos o transacciones de cajeros automáticos (ATMs) dado que el propósito de los indicadores para esta tesis (kpis) están enfocados en el manejo de comercios y lugares donde se utiliza la tarjeta de débito como medio de pago tomando, así como fuente de datos los registros generados por POS (Puntos de Venta) y comercio electrónico.

- **ÍNDICE PARA MANEJO DE CÓDIGOS DE PAÍS**

Para poder generar la correspondencia entre los códigos de país que se tiene actualmente en los registros y su respectivo significado, se generó el siguiente procedimiento.

Dentro de la documentación para procesos VISA y MASTERCARD se utiliza ISO 3166 international standard utilizado en toda la industria de las tecnologías de información por sistemas informáticos y software para facilitar la identificación de nombres de países.

Tomando en consideración lo antes planteado se procedió a crear un archivo JSON (Figura 40) que englobe los siguientes campos.

- City
- city_ascii
- lat
- lng
- country
- iso2
- iso3

Archivo JSON

```
{"index":{}}
{"city": "Cairo", "city_ascii": "Cairo","location": "30.05,31.25", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Alexandria", "city_ascii": "Alexandria","location": "31.2,29.95", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Giza", "city_ascii": "Giza","location": "30.01,31.19", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Ismailia", "city_ascii": "Ismailia","location": "30.5903,32.26", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Port Said", "city_ascii": "Port Said","location": "31.26,32.29", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Luxor", "city_ascii": "Luxor","location": "25.7,32.65", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Sūhāj", "city_ascii": "Suhaj","location": "26.5504,31.7", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Al Manşūrah", "city_ascii": "Al Mansurah","location": "31.0504,31.38", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Suez", "city_ascii": "Suez","location": "30.005,32.5499", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
{"index":{}}
{"city": "Damanhūr", "city_ascii": "Damanhur","location": "31.0504,30.47", "country": "Egypt", "iso2": "EG", "iso3": "EGY", "code": "818"}
```


Figura 38 Archivo JSON de países

A partir de este archivo se procede a crear un índice que contiene la estructura de los campos de los diferentes países que se encuentran dentro de la norma ISO 3166, se agregó adicional las coordenadas de latitud y longitud de cada uno para la posterior relación en la gráfica de mapa de calor a visualizar en el tablero de control.

Crear índice

Para crear el índice respectivo se procedió a ejecutar la siguiente línea de comando ejecutada desde consola.

curl-H "Content-Type: application/x-ndjson" -XPOST localhost:9200/countries/country/_bulk --data-binary @paises.json

- **ÍNDICE PARA MANEJO DE TIPOS DE COMERCIO**

Este índice permite a través de los códigos de tipos de comercio obtener la respectiva descripción a través de un archivo JSON creado con los detalles descritos en la página oficial de VISA (https://usa.visa.com/content/dam/VCOM/download/merchants/) como se aprecia en la Figura 41.

Archivo JSON

```
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4814", "Description_EN": "Fax services", "Description_ES": "Servicios de fax"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4815", "Description_EN": "VisaPhone", "Description_ES": "VisaPhone"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4821", "Description_EN": "Telegraph services", "Description_ES": "Servicios de telégrafo"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4829", "Description_EN": "Money Orders - Wire Transfer", "Description_ES": "Giros postales - Transferencia bancaria"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4899", "Description_EN": "Cable and other pay television (previously Cable Services)", "Description_ES": "Televisión por cable y otra televisión de pago (anteriormente Servicios por cable)"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "4900", "Description_EN": "Electric, Gas, Sanitary and Water Utilities", "Description_ES": "Servicios de electricidad, gas, sanitarios y agua"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5013", "Description_EN": "Motor vehicle supplies and new parts", "Description_ES": "Suministros de vehículos de motor y piezas nuevas"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5021", "Description_EN": "Office and Commercial Furniture", "Description_ES": "Mobiliario de oficina y comercial"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5039", "Description_EN": "Construction Materials, Not Elsewhere Classified", "Description_ES": "Materiales de construcción, no clasificados bajo otros epígrafes"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5044", "Description_EN": "Office, Photographic, Photocopy, and Microfilm Equipment", "Description_ES": "Equipos de oficina, fotográficos, de fotocopias y microfilmes"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5045", "Description_EN": "Computers, Computer Peripheral Equipment, Software", "Description_ES": "Computadoras, Equipo periférico de computadora, Software"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5046", "Description_EN": "Commercial Equipment, Not Elsewhere Classified", "Description_ES": "Equipo comercial, no incluido en otras categorías"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5047", "Description_EN": "Medical, Dental Ophthalmic, Hospital Equipment and Supplies", "Description_ES": "Equipos y suministros médicos, dentales, oftálmicos y hospitalarios"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5051", "Description_EN": "Metal Service Centers and Offices", "Description_ES": "Centros y oficinas de servicios metálicos"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5065", "Description_EN": "Electrical Parts and Equipment", "Description_ES": "Piezas y equipos eléctricos"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5072", "Description_EN": "Hardware Equipment and Supplies", "Description_ES": "Equipo y suministros de hardware"}
{"index":{}}
{"OrgLink": "VISA", "MerchantTypeCode": "5074", "Description_EN": "Plumbing and Heating Equipment and Supplies", "Description_ES": "Equipos y suministros de fontanería y calefacción"}
```


Figura 39 Archivo JSON de tipos de comercio

Crea índice

Para crear el índice respectivo se procedió a ejecutar la siguiente línea de comando ejecutada desde consola.

curl -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/business/merchanttype/_bulk --data-binary @merchanttype.json

- **ÍNDICE PARA MANEJO DE EXPRESIONES**

Este índice es creado con un archivo JSON (Figura 42) que permite manejar la relación de los lugares de comercios donde se realizaron las compras con tarjetas de débito y su respectiva expresión que permita asociar por ejemplo todas las transacciones de medio digitales realizadas en diferentes lugares en un solo tipo de comercio.

Archivo JSON

```
{"index":{}}
{"name": "Netflix", "expression": "*netflix*"}
{"index":{}}
{"name": "Netflix", "expression": "Netflix.*"}
{"index":{}}
{"name": "Netflix", "expression": "Netflix.*"}
{"index":{}}
{"name": "Netflix", "expression": "NETFLIX.*"}
{"index":{}}
{"name": "Netflix", "expression": "*.NETFLIX.*"}
{"index":{}}
{"name": "Netflix", "expression": "NETFLIX.COM"}
{"index":{}}
{"name": "Zoom", "expression": "Zoom.*"}
{"index":{}}
{"name": "Zoom", "expression": "*.zoom.*"}
{"index":{}}
{"name": "Zoom", "expression": "Zoom.*"}
{"index":{}}
{"name": "Google", "expression": "GOOGLE.*"}
{"index":{}}
{"name": "Google", "expression": "Google.*"}
{"index":{}}
{"name": "Spotify", "expression": "Spotify.*"}
{"index":{}}
{"name": "Spotify", "expression": "SPOTIFY.*"}
{"index":{}}
{"name": "Apple", "expression": "Apple.*"}
{"index":{}}
{"name": "Apple", "expression": "APPLE "}
{"index":{}}
{"name": "Apple", "expression": "APPLE *"}
{"index":{}}
{"name": "Apple", "expression": "APPLE*"}
{"index":{}}
{"name": "Apple", "expression": "APPLE.COM*"}
```

Figura 40 Archivo JSON de expresiones para determinar el nombre del comercio

Crear índice

Para crear el índice respectivo se procedió a ejecutar la siguiente línea de comando ejecutada desde consola.

curl -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/expressions/expression/_bulk --data-binary @merchanttype.json

MANEJO DE TRAMAS ISO8583

A través del archivo de configuración iso8583.properties que se encuentra dentro del microservicio se definen los tipos de campos que se desea obtener del procesamiento de la trama para transacciones de tarjetas. Se implementa una librería Java añadida en el microservicio, que permite realizar la conversión de una trama bajo el estándar ISO8583 en información que permita aportar datos adicionales de la transacción al registro final a almacenarse en la base de datos no relacional. A continuación, se presenta parte del código (Código 7) implementado dentro del consumidor Java. Este tipo de implementación se la realizo para el proceso de autorización de compras para la red VISA. La figura 43 muestra el resultado de colocar la trama iso8583 dentro de una aplicación ya desarrollada denominada ISOTESTER facilitada por un proveedor de la institución financiera, donde se puede comprobar que los resultados de los campos que se obtiene son los mismos que los que se generan dentro del consumidor desarrollado (Figura 44).

Código 7 Transformación trama ISO8583

```
if(orglink.trim().equals("VISA") && destlink.trim().equals("XXX") && mti==100) //SOLO PARA TRAMAS DE AUTORIZACION VISA
                                        {
                                                log.info("ENTRA A PROCESAR TRAMA ISO8583");
                                                String valorcampo=valorobject.get(campo)!=null?valorobject.get(campo).getAsString():"";
                                                byte[] decoded = BaseEncoding.base64().decode(valorcampo);
                                                String tramaEntradaISO8583 = Hex.encodeHexString(decoded).toUpperCase();

        System.out.println("hexadecimal"+tramaEntradaISO8583.toUpperCase());
                                                IsoMensaje respvisa=p1.readMensajeFromString(tramaEntradaISO8583,44,"respTramaVisa.xml",false);
                                                        for(String fieldiso:fieldsiso8583)
                                                        {
                                                                String[] datosfieldiso=fieldiso.split(":");
                                                                        String decodedfield=respvisa.getValues().get(Integer.parseInt(datosfieldiso[1])).
```

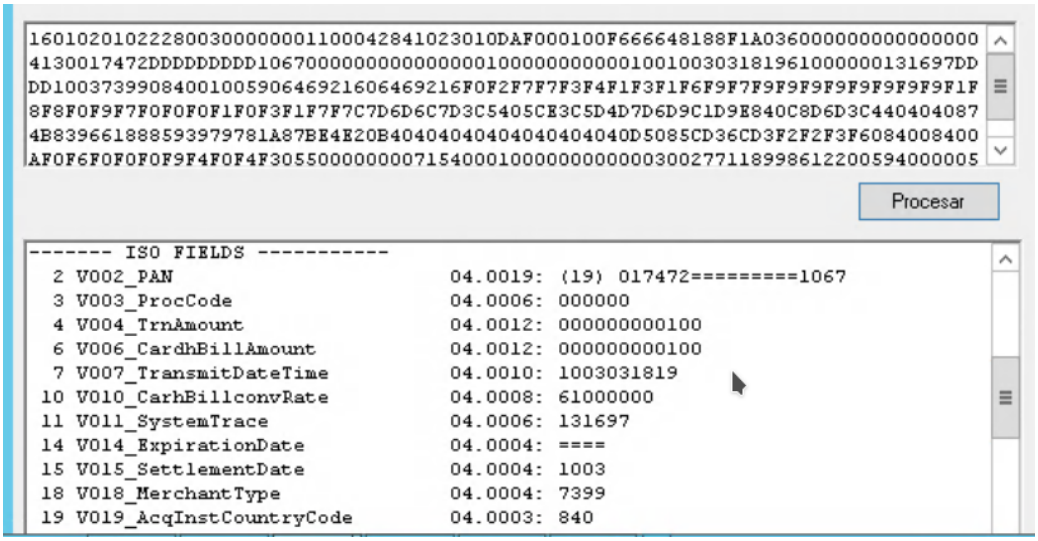

Figura 41 Resultado de trama iso8583 en isotester

```
0100
F   2(LLVARHEX): 130017472DDDDDDDDD1067 -> '0017472DDDDDDDDD1067'
F   3(NUMERIC): 000000 -> '0'
F   4(MONTO): 000000000100 -> '1.0'
F   6(MONTO): 000000000100 -> '1.0'
F   7(DATE10): 1003031819 -> '1003031819'
F  10(NUMERIC): 61000000 -> '61000000'
F  11(NUMERIC): 131697 -> '131697'
F  14(DATE4): DDDD -> 'DDDD'
F  15(NUMERIC): 1003 -> '1003'
F  18(NUMERIC): 7399 -> '7399'
F  19(NUMERIC): 0840 -> '840'
F  22(NUMERIC): 0100 -> '100'
F  25(NUMERIC): 59 -> '59'
```

Figura 42 Trama iso8583 en microservicio

ANEXO 5: ALMACENAMIENTO EN ELASTICSEARCH

Una vez se han tratado los datos obtenidos en la cola de mensajes se envían a Elasticsearch para su almacenamiento. Se incluye así en el proyecto la dependencia Maven asociada a la versión de Elasticsearch inicializada como se indica en el código 8.

Código 8 Dependencia de Maven para el conector de Apache Kafka

```
<dependency>
   <groupId>org.elasticsearch.client</groupId>
   <artifactId>transport</artifactId>
   <version>7.9.1</version>
</dependency>
```

Dentro del microservicio desarrollado se implementó una clase que maneja el proceso de envió de las transacciones finales una vez han pasado el proceso de limpieza y transformación de datos hacia el almacenamiento en Elasticsearch como se presenta en el código 9.

Código 9. Carga desde microservicio a Elasticsearch

```
public void bulkSend(List<Record> records, String index, String type) {
Bulk.Builder bulkBuilder = new Bulk.Builder()
.defaultIndex(index)
.defaultType(type);

for (Record bulkItem : records)
```

```
{
JsonObject dataList = bulkItem.getDataList();
boolean isInsert = bulkItem.getBehaviour().equals(Constants.insertedFlagValue);
if(isInsert) {
String idtransaccion=dataList.get("IdTransaccion").getAsString();
log.info("IDTRANSACCION"+idtransaccion);
bulkBuilder.addAction(new Index.Builder(dataList).id(idtransaccion).index(index).build());
}
}
try
{
BulkResult execute = client.execute(bulkBuilder.build());
if (!execute.isSucceeded()) {
System.err.println("Failed to add metric(s) to ES: " + execute.getErrorMessage());
}

} catch (IOException e) {
e.printStackTrace();
log.error(e.toString());
}
}
```

Por último, en la figura 45 se muestra una transacción de una tarjeta de débito almacenada en la base de datos no relacional con sus respectivos campos.

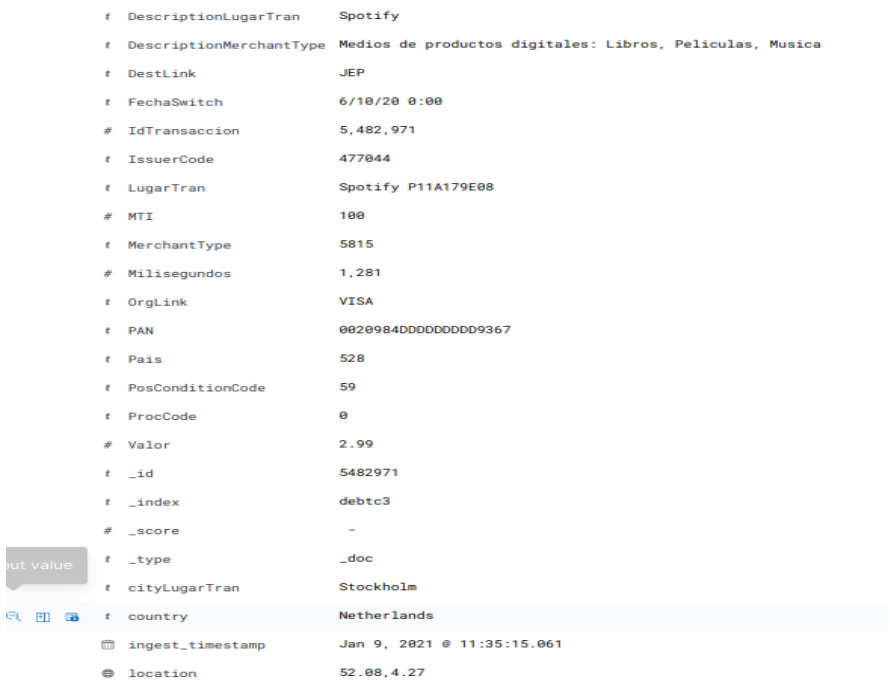

Figura 43 Campos almacenados en Elasticsearch

Para más información del almacenamiento no relacional para el proceso de tarjetas de débito de la institución financiera se puede observar en el Anexo 5.

ANEXO 6: TABLERO DE CONTROL DE TARJETAS DE DÉBITO

Kibana inicialmente requiere de un patrón de índice para acceder a los datos de Elasticsearch. Se seleccionan los datos que se utilizarán y permite a su vez definir las propiedades de los campos.

Se define en primera instancia para el índice **my_index** el formato de los campos, es decir se ha creado un campo location que será tipo geopoint para definir la latitud y longitud. Además, se ha creado un campo adicional tipo timestamp para determinar la fecha y hora en que se receptaron cada una de las transacciones.

CREAR TIMESTAMP Y GEOPOINT

Para crear una marca de tiempo de los documentos que estarán dentro del índice que crearemos donde se almacenará la información una vez se ha realizado la respectiva limpieza. Una marca de tiempo permite determinar la fecha y hora en que llegaron las transacciones a almacenarse. Se ejecuta las siguientes líneas de código para agregar así esta marca de tiempo al documento a indexarse.

La herramienta adecuada para configurar el mapeo de los datos es la consola, que se encuentra en el menú “Dev Tool” > “Console” en la interfaz de Kibana donde se inserta por orden los siguientes mapeos utilizando una petición PUT:

```
PUT my_index
{
  "settings": {
    "default_pipeline": "my_timestamp_pipeline"
  }
}

PUT _ingest/pipeline/my_timestamp_pipeline
{
  "description": "Adds a field to a document with the time of ingestion",
  "processors": [
    {
      "set": {
        "field": "ingest_timestamp",
        "value": "{{_ingest.timestamp}}"
      }
```

```
    }
  ]
}

PUT my_index
{
  "mappings": {
    "properties": {
      "location": {
        "type": "geo_point"
      }
    }
  },
  "settings": {
    "default_pipeline": "my_timestamp_pipeline"
  }

}
```

Crear un patrón de índice

Para que Kibana sepa qué datos ha de procesar, hay que crear los patrones para los índices en este caso “my_index”. Para definir primero haremos lo siguiente:

Abre el menú “Management” (Administración) y clic en “Index Patterns” (Patrones de índice). Con el primer patrón de índice, la página “Create index pattern” (Crear patrón de índice) se abre automáticamente, aunque también puede abrirse con el botón del mismo nombre.

Escribe "my_index*" en el campo "Index pattern" y, a continuación, pincha en "Next step" (Paso siguiente). En este momento se selecciona "**@timestamp**" en el menú desplegable "**Time Filter field name**" (Nombre de campo de filtro de tiempo), ya que estos registros contienen datos de secuencias cronológicas de la marca de tiempo definida anteriormente. A continuación, clica en "**Create index pattern**" como aprecia en la Figura 46.

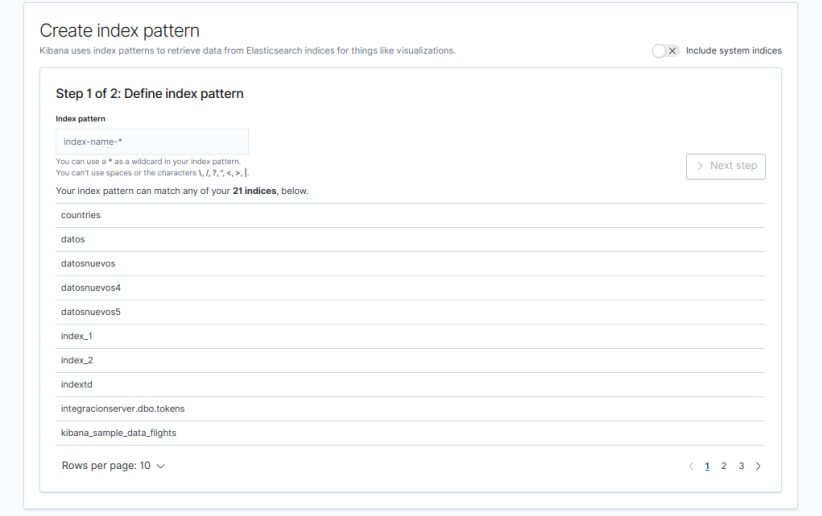

Figura 44 Patrón de índice en Kibana

Una vez se ha configurado el índice del cual capturar la información en la sección Discover de Kibana, se puede acceder a verificar las transacciones en tiempo real que se están almacenando sobre Elasticsearch (Figura 47). A través de la siguiente gráfica de barras y de acuerdo a la configuración necesaria se puede permitir visualizar la información dentro de dicha gráfica por rangos de tiempo, originando gracias a esta herramienta de monitoreo verificar picos de transacciones durante lapsos de seg/min/horas/días.

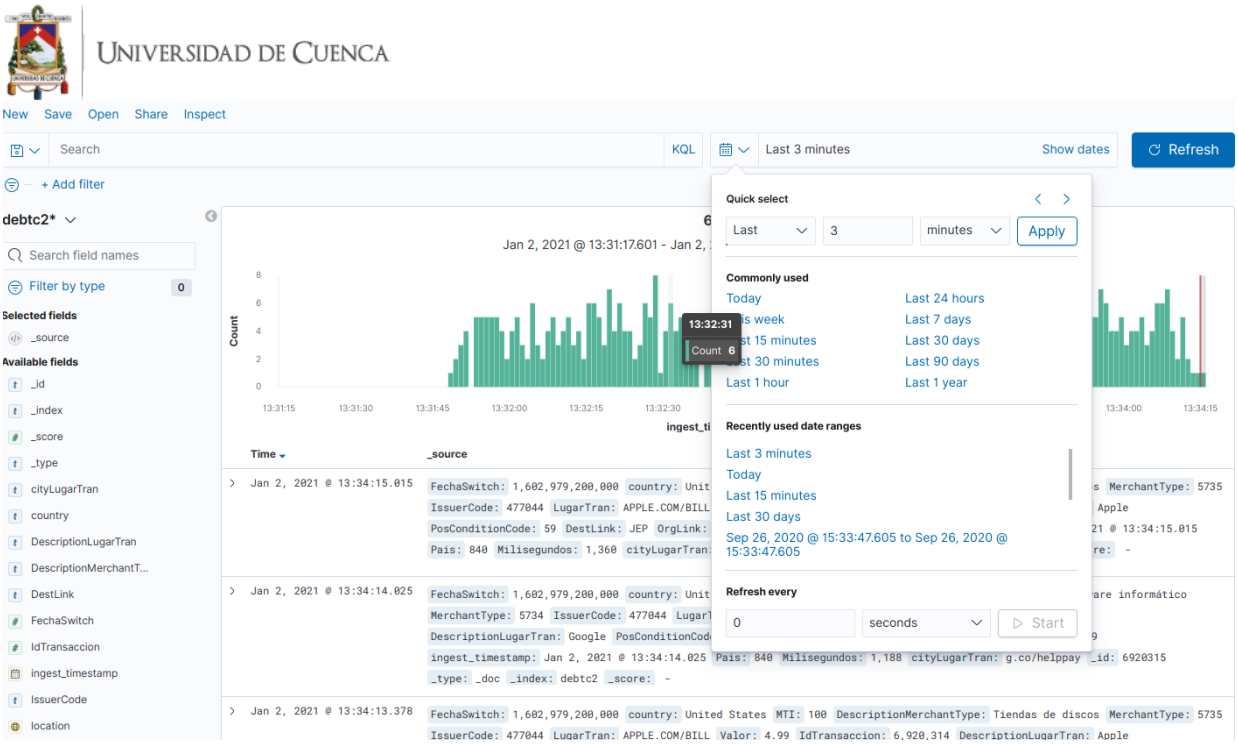

Figura 45 Visualización de transacciones en tiempo real en Kibana

Finalmente se presenta en la Figura 48 el tablero de control diseñado para la institución financiera que aporte una visualización general del uso de las tarjetas de débito como medio de pago en establecimientos y comercio electrónico y un análisis de los resultados obtenidos.

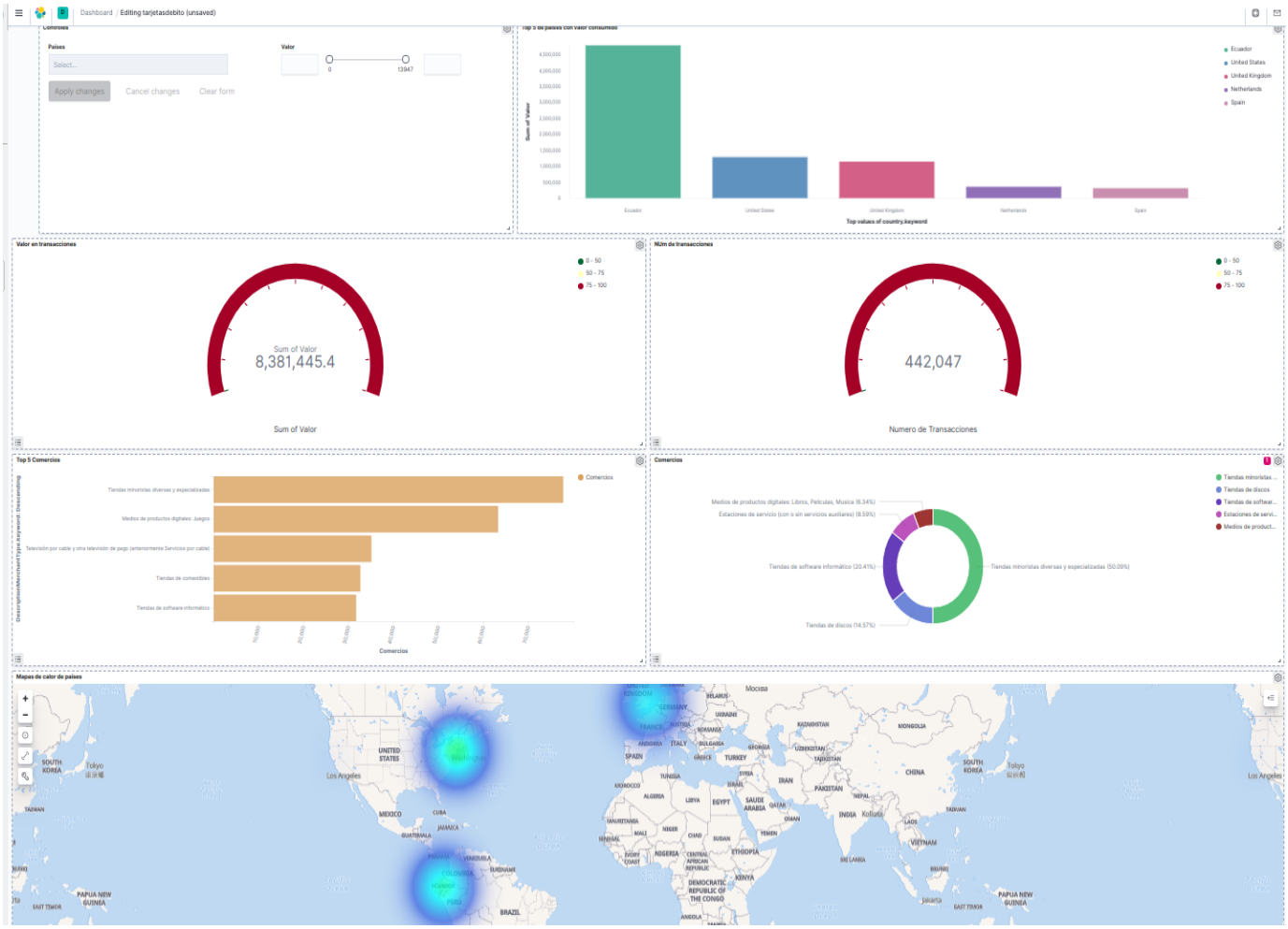

Figura 46 Tablero de control de tarjetas de débito de la institución financiera

Durante la interacción entre Kibana y Elasticsearch para visualizar en tiempo real el flujo de la información se encontraron los siguientes inconvenientes para búsqueda de la información de manera ágil y eficiente.

Problemas Encontrados

JOINS EN ELASTICSEARCH

Inconveniente

Elasticsearch no admite uniones entre índices como SQL. Se realizó el estudio sobre este punto dado que es necesario poder consultas los países, mercados o negocios donde se generan cada una de las transacciones. Dado que la información que se obtiene de la data de una tarjeta de débito contiene únicamente los códigos, se precisa realizar un cruce de información para obtener las etiquetas respectivas.

Solución

Se crearon consultas propias de Elasticsearch dentro del consumidor Java (microservicio). A través de consultas must match para búsquedas de las etiquetas respectivas de los códigos tanto de países, negocios o tipos de mercado.

MANEJO DE MARCAS DE TIEMPO EN ELASTICSEARCH

Inconveniente

Para poder observar la transaccionalidad, uno de los elementos claves es generar marcas de tiempo para todos los documentos que se van a agregar al índice. Elasticsearch solía admitir la adición automática de marcas de tiempo a los documentos que se indexan, pero desaprobó esta característica en la versión 2.0.0

Solución

Se debe crear un campo de fecha regular con la marca de tiempo actual desde el lado de la aplicación e invocar al mismo desde el índice creado.

Ejemplo

```
PUT _ingest/pipeline/indexed_at
{
  "description": "Adds indexed_at timestamp to documents",
  "processors": [
    {
      "set": {
        "field": "_source.indexed_at",
        "value": "{{_ingest.timestamp}}"
      }
    }
  ]
```

BIBLIOGRAFÍA

1. Andreas Andreakis, I. P. (2019). *DBLog: A Generic Change-Data-Capture Framework*. Obtenido de https://netflixtechblog.com/dblog-a-generic-change-data-capture-framework-69351fb9099b
2. Basantes Espinoza Gabriela Paola, López Galarza Daniel Eduardo. (2012). Estudio de la aplicación de Inteligencia de Negocios en los procesos académicos. Caso de estudio "Universidad Politecnica Salesiana".
3. Bathla Gourav, Rani Rinkle, Aggarwal Himanshu. (2018). Comparative study of NoSQL databases for big data storage.
4. Benson, K. (2019). Implementing Publish-Subscribe Pattern in a. *Master's Thesis*.
5. Bucchiarone Antonio, Dragoni Nicola, Dustdar Schahram, Larsen Stephan, Mazzara Manuel. (2018). From Monolithic to Microservices: An experience report. *IEEE Software*, 50-55.
6. Bucur Vlad, Stan Ovidiu, Miclea Liviu. (2020). n Analysis of the Implementation of Kafka in High-Frequency Electronic Trading Environments. *International Journal of Modeling and Optimization*, 52-56.
7. Díaz de la Paz, Lisandra & García Mendoza, Juan Luis. (2015). Técnicas para capturar cambios en los datos y mantener actualizado un almacén de datos. *Revista Cubana de Ciencias Informáticas.*
8. Fernández, C. Ó. (2017). Procesamiento de flujos de eventos en un entorno distribuido y análisis de comportamiento: arquitectura e implementación. *Ingeniería Informática*.
9. Fikri Noussair, Rida Mohamed, Abghour Noureddine, Moussaid Khalid, El Omri Amina. (2019). An adaptive and real-time based architecture for financial data integration. *Journal of Big Data*, 6,97.
10. Filip Mezera, Jiří Křupka. (2017). Selected business intelligence methods for decision-making support in a finance institution.
11. Freire, J. F. (2015). *Módulo de Business Intelligence para el Sistema Financiero Financial de la empresa.* Ambato.
12. Garrido, F. F. (2017). Arquitectura Big Data de ingesta en Real Time. *Master en inteligencia de negocio y Big Data*.
13. Guo Fu, Y. Z. (2020). A Fair Comparison of Message Queuing Systems. *IEEE Access*.
14. Hojjat Jafarpour, Rohan Desai, Damian Guy. (2019). KSQL: Streaming SQL Engine for Apache Kafka. *Industry and Applications Paper*.
15. Ichinose Ayae, Takefusa Atsuko, Nakada Hidemoto, Oguchi Masato. (2017). A study of a video analysis framework using Kafka and spark streaming.
16. Jácome, D. F. (2020). Arquitectura de análisis de datos generados por el Internet de las cosas (IoT) en tiempo real.
17. Javed, S. (2017). Apache Kafka: Real Time Implementation with Kafka Architecture Review. International Journal of Advanced Science and Technology.
18. Jay Kreps,Neha Narkhede,Jun Rao. (2011). Kafka: a Distributed Messaging System for Log Processing.
19. Jebaraj Anand, Chaojie Yang, Koona Manoj, Ratnayake Dilanka. (2020). Business Analytics in Banking: A comparative study between IBM and Tableau.

20. Libs, O. S. (2020). *68 Open Source Cdc Software Projects*. Obtenido de https://opensourcelibs.com/libs/cdc
21. Madrid, V. (2018). *Aprendiendo Apache Kafka*. Obtenido de EN MI LOCAL FUNCIONA Technical thoughts, stories and ideas: https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-2-2/
22. Mary Julieth Murillo Junco, Gustavo Cáceres Castellanos. (2013). Business intelligence and financial decision-making: a theoretical approach. *LOGOS CIENCIA & TECNOLOGÍA*, 119 – 138.
23. Microsoft. (2021). *Real time processing*. Obtenido de https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/real-time-processing
24. Minsait. (2021). *X Edición del Informe de Tendencias en Medios de Pago*. Obtenido de https://www.minsait.com/es/actualidad/insights/tendencias-en-medios-de-pago
25. Montero, J. M. (12 de Junio de 2016). *Detección de fraude bancario en tiempo real utilizando tecnologías de procesamiento distribuido*. Obtenido de https://eprints.ucm.es/38647/1/Memoria%20TFM%20Detección%20Fraude_FINAL.pdf
26. Nicola Dragoni,Schahram Dustdar, Stephan T. Larsen,Manuel Mazzara.(2017). Microservices: Migration of a Mission Critical System.
27. Ouda, G. (2016). Application of Business Intelligence in the Financial Services Industry.
28. Philippe Dobbelaere,Kyumars Sheykh Esmaili. (2017). Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations. *11th ACM International Conference*.
29. Raje, S. N. (2019). Perfomance Comparison of message queue methods. *Master of Science in Computer Science*.
30. Ríos Carrión Pamela Mishelle, Bermeo Pazmiño Katina Vanessa, Narváez Zurita Cecilia Ivonne. (2021). *Inteligencia de negocios como estrategia para la toma de decisiones en una empresa financiera*. Obtenido de https://doi.org/10.35381/cm.v7i12.438
31. Ríos, D. F. (2020). Diseño de un prototipo de una arquitectura basada en microservicios para la integración de aplicaciones web altamente transaccionales. Caso: entidades financieras.
32. Salvador, J. V. (2020). Análisis de la arquitectura dirigida por eventos (eda).
33. Smid Antonin, Wang Ruolin, Černý Tom. (2019). Case study on data communication in microservice architecture.
34. stratebi. (2020). Herramientas Change-Data-Capture: Debezium y DBLog. *Stratebi open business intelligence*.
35. Takashi, S. (2020). *Developing a stream processor with Apache Kafka*. Obtenido de https://developer.ibm.com/tutorials/developing-a-streams-processor-with-apache-kafka/
36. TAT, T. K. (julio de 2019). *Arquitectura Big Data para la ejecución de Reglas de Negocio*. Recuperado el 22 de abril de 2020, de https://repositori.udl.cat/bitstream/handle/10459.1/67910/tkint.pdf

37. TORRALBA, P. P. (2021). *¿Qué son los procesos ETL?* Obtenido de https://www.iebschool.com/blog/que-son-los-procesos-etl-big-data/
38. Universo, E. (2021). *Aumenta consumo con tarjetas de débito y baja el uso de las de crédito en Ecuador*. Obtenido de https://www.eluniverso.com/noticias/economia/aumenta-consumo-con-tarjetas-de-debito-y-baja-el-uso-de-las-de-credito-en-ecuador-nota/
39. Valle, V. M. (2019). *Comunicando microservicios con Apache Kafka*. Obtenido de Paradigma: https://www.paradigmadigital.com/dev/comunicacion-microservicios-apache-kafka/
40. Vlad Bucur, Ovidiu Stan, Liviu Micle. (2020). An Analysis of the Implementation of Kafka in High-Frequency Electronic Trading Environments. *International Journal of Modeling and Optimization*.
41. Wieringa Roel J. (2014). Design Science Methodology for Information Systems and Software Engineering. *Springer*.
42. Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, Engelbert Mephu Nguifo. (2018). A Comparative Study on Streaming Frameworks for Big Data. *Latin America Data Science Workshop*.
43. Kreps, J. (2014). *Questioning the Lambda Architecture*. Obtenido de https://www.oreilly.com/radar/questioning-the-lambda-architecture/