

**UNIVERSIDAD DE SANTIAGO DE CHILE**  
**FACULTAD DE INGENIERÍA**  
**DEPARTAMENTO DE INGENIERÍA INFORMÁTICA**



## **Laboratorio 2- Apache Kafka y Apache Spark**

Integrante: Luis González  
Curso: Sistemas Distribuidos  
Profesor: Manuel Manríquez Badilla  
Ayudante: Ariel Madariaga

24 de Agosto de 2024

# Tabla de contenidos

<b>1. Introducción</b>	<b>1</b>
<b>2. Caso de estudio</b>	<b>2</b>
2.1. Investigación del caso de estudio . . . . .	2
<b>3. Arquitectura propuesta</b>	<b>4</b>
3.1. Generación de Datos . . . . .	4
3.1.1. Transmisor_datos_kafka.py . . . . .	4
3.2. Kafka Cluster . . . . .	5
3.2.1. Brokers . . . . .	5
3.2.2. DataTopic . . . . .	5
3.3. Spark Cluster . . . . .	6
3.3.1. Spark Master . . . . .	6
3.3.2. Workers (Worker 1 y Worker 2) . . . . .	6
3.3.3. notification.py . . . . .	7
3.4. Notificación . . . . .	8
<b>4. Metodología</b>	<b>9</b>
4.0.1. Generación de Datos . . . . .	9
4.0.2. Creación del JSON de Datos de Usuario . . . . .	10
4.0.3. Simulación y Envío de Datos a Kafka . . . . .	12
4.0.4. Recepción y Procesamiento de Datos con Spark . . . . .	13
<b>5. Resultados y discusión</b>	<b>15</b>
5.1. Resultados Obtenidos: Mensajes No Enviados (Tiempo de compra nulo) . . .	15
5.2. Resultados Obtenidos: Mensajes Enviados con Tiempo de Compra Menor a 30 minutos . . . . .	16
5.3. Resultados Obtenidos: Mensajes Enviados con Tiempo de Compra Mayor a 30 minutos . . . . .	17
5.4. Discusión de Resultados . . . . .	18

<b>6. Conclusión</b>	<b>19</b>
6.1. Limitaciones y Trabajo Futuro . . . . .	19
<b>Bibliografía</b>	<b>21</b>

# 1. Introducción

En la era digital actual, la integración de software con procesos de negocio es cada vez más crucial para empresas que buscan mantenerse competitivas en un mercado globalizado. La complejidad inherente a las arquitecturas de software modernas exige soluciones que no solo respondan a las necesidades funcionales del negocio, sino que también manejen eficazmente grandes volúmenes de datos y garanticen la consistencia y disponibilidad del sistema.

Este laboratorio se enfoca en el uso de tecnologías de procesamiento distribuido para abordar estos desafíos. En particular, se exploran dos tecnologías clave: Apache Kafka y Apache Spark. Apache Kafka es una plataforma de transmisión de datos distribuida que facilita la gestión de flujos de datos en tiempo real mediante la implementación de un sistema de mensajería altamente escalable y confiable. Apache Spark, por otro lado, es un motor de procesamiento de datos en clúster que permite la realización de análisis de datos de gran escala con alta velocidad y eficiencia.

El objetivo principal de este laboratorio es comprender y aplicar estas tecnologías para desarrollar un estudio de caso que permita abordar un problema real y significativo en el contexto de un sistema distribuido. A través del análisis y la implementación de modelos de consistencia y políticas de replicación, se busca demostrar cómo estas tecnologías pueden ser utilizadas para resolver problemas complejos en un entorno de procesamiento distribuido.

## 2. Caso de estudio

El caso de estudio seleccionado para este análisis es el colapso del e-commerce de Falabella durante eventos de alta demanda como el Cyberday. Durante estos eventos, el sitio web experimenta un aumento masivo en el tráfico, lo que frecuentemente resulta en fallas del sistema, afectando tanto la experiencia del usuario como las operaciones de la empresa. Este escenario representa un desafío significativo en términos de escalabilidad, consistencia de datos y manejo de cargas distribuidas.

El contexto funcional del problema involucra la necesidad de un sistema distribuido que no solo soporte un gran número de transacciones simultáneas, sino que también mantenga la integridad y consistencia de los datos en tiempo real. La caída del sistema en estos eventos sugiere deficiencias en la arquitectura actual, especialmente en lo que respecta al manejo de la carga y la replicación de datos. Por lo tanto, la propuesta arquitectónica se enfocará en mejorar la resiliencia y la capacidad de procesamiento del sistema durante picos de demanda, utilizando Apache Kafka para la gestión eficiente de mensajes y Apache Spark para el procesamiento de datos en clústeres distribuidos.

### 2.1. Investigación del caso de estudio

Según el Sernac Hora de Noticias (2023) (Servicio Nacional del Consumidor), existieron muchas demandas a Falabella en un periodo de tiempo donde el problema principal eran las compras en línea a través de su sitio web, es por esto que esta entidad abrió un formulario para los consumidores afectados para que entregarán un mayor detalle de lo sucedido. Esta empresa junto con Paris, enfrentaron una demanda colectiva al no presentar una propuesta de compensación.

Estos problemas y demandas presentaron una pérdida millonaria para las dos empresas, además de que el prestigio de ambas decayó por un periodo de tiempo. Como se muestra en la imagen 1, Falabella tuvo que devolver a los usuarios una cantidad enorme de dinero.

	<b>ACUERDO</b>			
	<b>MONTO</b>	<b>TRANSACCIONES</b>	<b>\$/Trans</b>	<b>Clientes</b>
INCUMPLIMIENTO FECHA DE ENTREGA	\$ 67.789.450	10.913	\$ 6.212	10.627
CANCELACIÓN DE ÓRDENES POR QUIEBRE D	\$ 73.163.500	7.992	\$ 9.155	7.849
GIFT CARD ENTREGADA Y NO COBRADA	\$ 123.500	26	\$ 4.750	26
COSTO DE RECLAMACIÓN	\$ 149.460.245	18.931	\$ 7.895	18.182
COMPENSACIÓN COMPLEMENTARIA	\$ 151.196.791	18.931	\$ 7.987	18.182
<b>SUB TOTAL</b>	\$ 441.733.486	18.931	\$ 23.334	18.182
COMPENSACIÓN ENTREGADA	\$ 46.362.600	4.056	\$ 11.431	
<b>TOTAL FINAL</b>	\$ 395.370.886	18.931	\$ 20.885	

Figura 1: Devolución efectuada por Falabella

### **3. Arquitectura propuesta**

La arquitectura del sistema propuesto se basa en un flujo de datos que se realiza de manera distribuida usando la herramienta de Apache Kafka para la transmisión y gestión de compras realizadas por usuarios en tiempo real y de manera simultanea:

#### **3.1. Generación de Datos**

Para hacer el enfoque de una arquitectura como la mencionada anteriormente, primero se deben generar los datos correspondientes, para ello se emplea el concepto de Productor(Producer) y Consumidor(Consumer). El Productor es el encargado de generar los mensajes con respectivas condiciones y el Consumidor es el encargado de leer estos mensajes.

##### **3.1.1. Transmisor\_datos\_kafka.py**

Este script se encarga de generar los datos correspondientes a una "persona" que realiza una compra en un sistema de comercio electrónico. El objeto "persona" contiene toda la información relevante, como nombre, ID de la persona, nombre, apellido, telefono, los detalles de la compra (Número de folio), incluyendo un tiempo de compra, etc. Si el tiempo de compra es null o vacío, el número de folio del artículo comprado no se genera, ya que la transacción no es contabilizada y a esa persona no le llega el mensaje que corrobora la compra. Además existe la condición que verifica que los procesos esten funcionando correctamente mediante un mensaje que será diferente si el tiempo en que se demoró en realizar una transacción o compra es mayor a 30 min o si es menor a 30 min. Cabe destacar que esto puede ser pasado de un broker a otro para así distribuir de mejor manera los datos, en este caso solo se envia un mensaje diferente esperando el feedback del usuario, solo para temas de prueba.

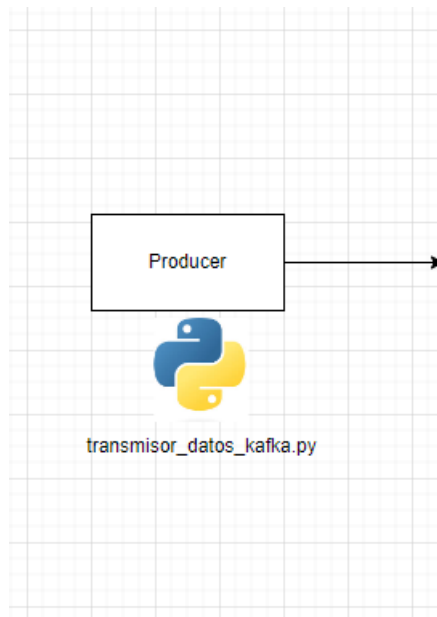


Figura 2: Diagrama Producer trasmisor de datos

## 3.2. Kafka Cluster

Es un un grupo de servidores que actua como intermediarios que trabajan juntos para manejar de la mejor manera los flujos de datos entrantes y salientes de un sistema kafka, cabe destacar que estos servidores trabajan en procesos separados.Boltic (2024)

### 3.2.1. Brokers

El clúster de Kafka está compuesto por tres brokers (Broker 1, Broker 2, y Broker 3) que gestionan la distribución de los mensajes y aseguran la durabilidad de los datos. Figura 3

### 3.2.2. DataTopic

Es el topic donde se almacenan los mensajes generados por el productor. Estos mensajes contienen la información de la persona y su transacción válida. Figura 3



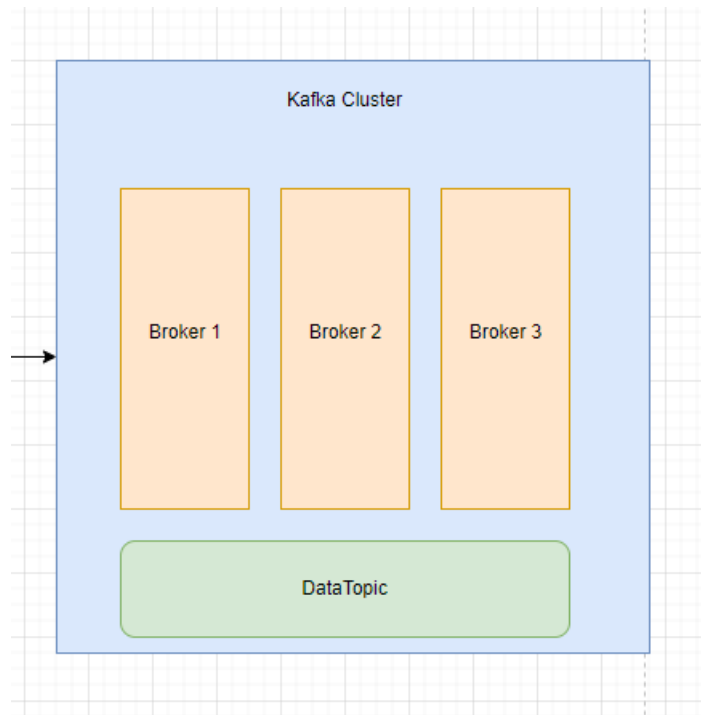


Figura 3: Diagrama Kafka cluster

### 3.3. Spark Cluster

” El administrador de clúster o Clúster Manager en Spark hace referencia a la comunicación del driver con el backend para adquirir recursos físicos y poder ejecutar los executors.” KeepCoding (2024)

#### 3.3.1. Spark Master

Este componente orquesta las tareas de procesamiento distribuidas entre los trabajadores del clúster.

#### 3.3.2. Workers (Worker 1 y Worker 2)

Los workers se encargan de procesar los datos recibidos del DataTopic de Kafka. Cada worker aplica los esquemas de consistencia y maneja las réplicas según lo diseñado. En este caso, se enfoca en validar los datos y notificar aquellas transacciones que cumplen con los criterios especificados (transacciones válidas con un tiempo de compra registrado)

y procesan el mensaje correspondiente si es menor o mayor a 30 min lo que se demoró el usuario en hacer la compra.

### 3.3.3. notification.py

Este script es el encargado de procesar las transacciones validadas por los workers y generar notificaciones para las diferentes condiciones. Las notificaciones simulan un sistema de alerta en tiempo real (por consola) que informa sobre las transacciones procesadas.

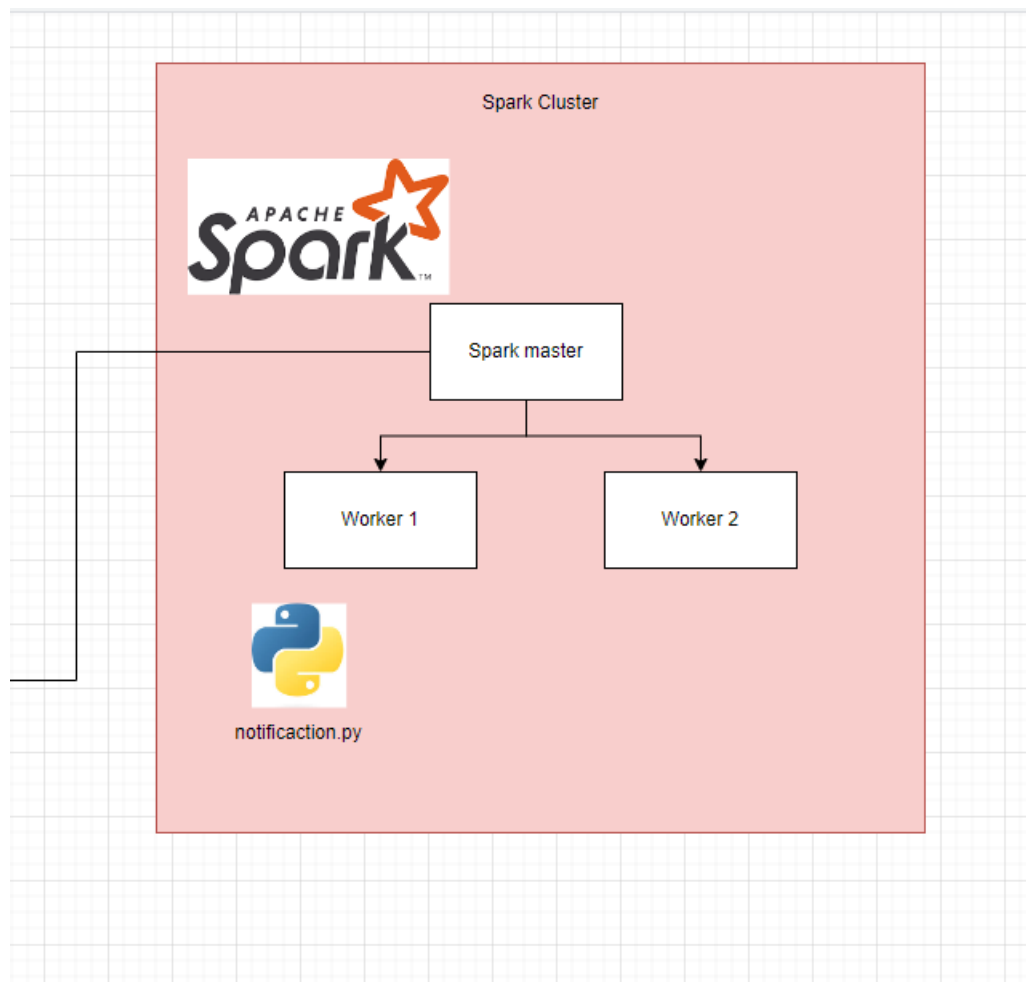


Figura 4: Diagrama Spark cluster

### 3.4. Notificación

Una vez que los datos son procesados por los workers del clúster Spark, se generan notificaciones en tiempo real. Estas notificaciones informan sobre las transacciones validadas y en este caso por temas de aprendizaje se muestran a través de la consola, permitiendo el monitoreo de aquellas personas que tuvieron un problema en la compra del producto, demorándose más de 30 min en terminar de hacer las acciones para adquirirlo.

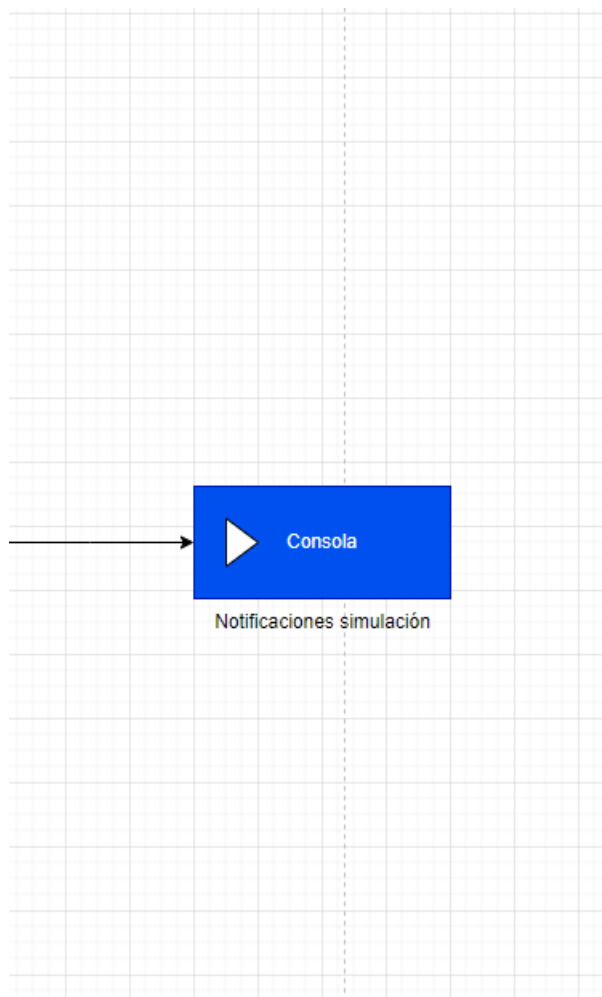


Figura 5: Diagrama notificación por consola

Finalmente el diagrama completo de la arquitectura escogida para resolver esta problemática es la presentada en la Figura 6.

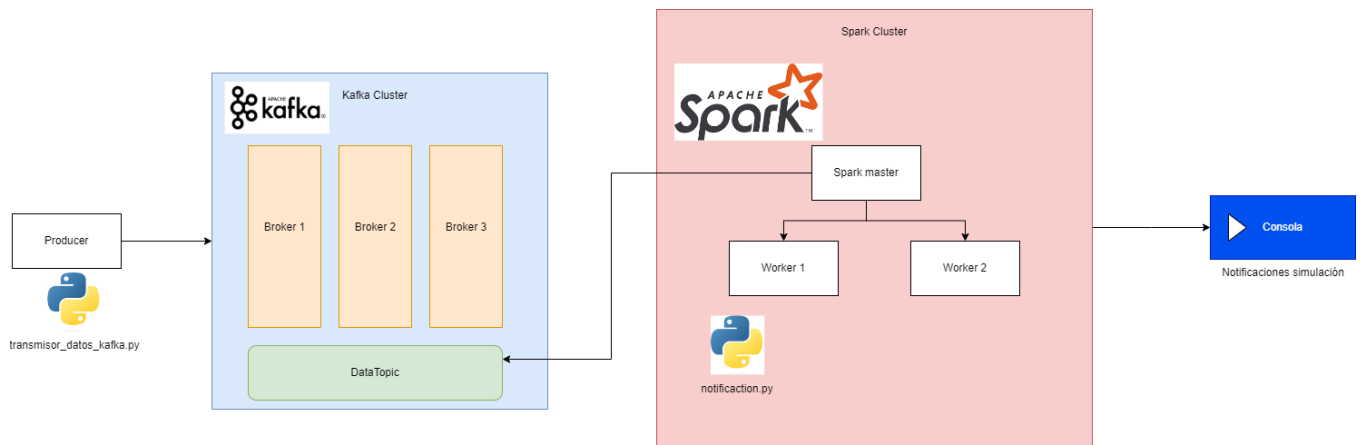


Figura 6: Diagrama completo

## 4. Metodología

La metodología implementada para la simulación de la recepción de datos desde un e-commerce se realizó utilizando Python, con un enfoque en la generación de datos aleatorios representativos de transacciones de usuarios. Se emplearon varias clases dedicadas a la generación de distintos atributos de los usuarios, como nombre, apellido, edad, y otros detalles necesarios para simular la interacción de un usuario con la plataforma de e-commerce. A continuación, se detallan los pasos y configuraciones utilizados para obtener los resultados:

### 4.0.1. Generación de Datos

Para poder simular los datos que llegan desde la plataforma, se utilizaron diferentes clases para que se puedan representar de la mejor manera a los usuarios que adquirieron un producto:

- **GeneradorId:** Genera un identificador único para cada usuario.
- **GeneradorNombre:** Genera un nombre aleatorio leído desde un archivo txt.
- **GeneradorApellido:** Genera un apellido aleatorio leído desde un archivo txt.
- **GeneradorEdad:** Genera una edad dentro de un rango específico.
- **GeneradorNumeroTelefonico:** Genera un número telefónico válido.

- **GeneradorPais:** Lee un archivo txt donde se contienen países aleatorios.
- **GeneradorCiudad:** Lee un archivo txt donde se contienen ciudades aleatorias.
- **GeneradorDireccion:** Lee un archivo txt donde se contienen direcciones aleatorias.
- **GeneradorUbicacion:** Genera una ubicación basada en una lista de ciudades, direcciones y países basado en lo leído por las clases `GeneradorPais`, `GeneradorCiudad` y `GeneradorDireccion`.
- **GeneradorProductoVisto:** Simula la visualización de productos por parte del usuario leyendo los productos desde un txt donde se generan de forma aleatoria.
- **GeneradorTAPS:** Genera un tiempo de compra asociado a la transacción, el cual puede ser nulo.
- **GeneradorFolio:** Genera un folio de transacción solo si el tiempo de compra (TAPS) es válido y los guarda en un txt.

Estos generadores se implementaron para garantizar que cada ejecución del script produjera un conjunto único de datos de usuario, reflejando posibles interacciones en un escenario de alta demanda, como un evento de CyberDay.

#### 4.0.2. Creación del JSON de Datos de Usuario

La clase `GeneradorJSONDatosPersonas` integra todos los generadores mencionados anteriormente para construir un objeto JSON que representa a un usuario con todos sus atributos, incluyendo:

- **ID del usuario**
- **Nombre y apellido**
- **Edad**
- **Email**
- **Número telefónico**

- Ubicación (ciudad, dirección, país)
- Producto comprado
- Folio (si aplica)

En el siguiente fragmento se puede apreciar de mejor manera como se genera este json con los datos de la persona:

```
class GeneradorJSONDatosPersonas:
    def generar_json_datos_personas(self):
        gen_id = GeneradorId()
        id = gen_id.generar_id()

        gen_nombre = GeneradorNombre()
        nombre = gen_nombre.generar_nombre()

        gen_apellido = GeneradorApellido()
        apellido = gen_apellido.generar_apellido()

        gen_edad = GeneradorEdad()
        edad = gen_edad.generar_edad()

        gen_telefono = GeneradorNumeroTelefonico()
        telefono = gen_telefono.generar_numero_telefonico()

        gen_ubicacion = GeneradorUbicacion()
        ubicacion = gen_ubicacion.generar_ubicacion()

        gen_producto_visto = GeneradorProductoVisto()
        producto_visto = gen_producto_visto.generar_producto_visto()

        gen_taps = GeneradorTAPS()
```

```

taps = gen_taps.generar_taps()

# Generamos el folio utilizando la nueva lógica dentro de GeneradorFolio
gen_folio = GeneradorFolio()
folio = gen_folio.generar_folio(taps)

email = f"{nombre}.{apellido}@mail.com"

datos_persona = {
    "id": id,
    "nombre": nombre,
    "apellido": apellido,
    "edad": edad,
    "email": email,
    "telefono": telefono,
    "ubicacion": ubicacion,
    "productovisto": producto_visto,
    "taps": taps,
    "folio": folio
}

datos_persona_json = json.dumps(datos_persona)
return datos_persona_json

```

#### 4.0.3. Simulación y Envío de Datos a Kafka

El archivo `transmisor_datos_kafka.py` actúa como un productor de mensajes para Apache Kafka. En este script, se invoca el método `generar_json_datos_personas()` para crear el JSON con los datos del usuario, y se envía el mensaje al topic correspondiente en Kafka utilizando el siguiente proceso:

**#Instanciación del Productor Kafka**

```

producer = KafkaProducer(bootstrap_servers=bootstrap_servers)
#Luego la generación y envío de datos:
datos_persona = gen_datos_persona.generar_json_datos_personas()
producer.send(topic_name, value=str(datos_persona).encode('utf-8'))
#Posteriormente el cierre del productor:
producer.close()

```

El script está diseñado para enviar datos al cluster de Kafka cada 2 segundos, simulando un flujo constante de transacciones durante un evento de alta demanda. Los datos generados se guardan en archivos de texto para su posterior análisis, incluyendo nombres, ciudades, direcciones, folios, etc.

#### 4.0.4. Recepción y Procesamiento de Datos con Spark

Los datos producidos por `transmisor_datos_kafka.py` son recepcionados y procesados utilizando un script consumer denominado `notification.py`. Este script se conecta al cluster de Kafka y utiliza Apache Spark para procesar en tiempo real los datos recibidos:

- **Sesión de Spark:** Se crea una sesión de Spark configurada para recibir y procesar datos en modo streaming.

```

spark = SparkSession\
    .builder\
    .appName("KafkaIntegration")\
    .master("local[3]")\
    .config("spark.sql.shuffle.partitions", 3)\
    .getOrCreate()

```

- **Recepción de Datos desde Kafka:** Se define un `DataFrame` para recibir los datos del topic de Kafka.

```

streaming_df = spark.readStream\

```



```

.format("kafka")\
.option("kafka.bootstrap.servers", "kafka1-p:9092, kafka2-p:9092, kafka3-
.option("subscribe", "DataTopic")\
.option("startingOffsets", "earliest")\
.load()

```

- **Procesamiento de Datos:** Se define un esquema para los datos recibidos y se procesan utilizando una consulta SQL para seleccionar los clientes que compraron el producto.

```

parsed_df = streaming_df\
    .select(F.col("value").cast(StringType()).alias("value"))\
    .withColumn("input", F.from_json("value", schema))\
    .select("input.*", "input.ubicacion.*")\
    .drop("ubicacion")

```

```

clientes_notificados_df = clientes_notificar(parsed_df)

```

- **Envío de Notificaciones:** Se configura la consulta para ejecutar cada 9 segundos, enviando notificaciones a los clientes basadas en su tiempo de compra (Tcompra).

```

query = clientes_notificados_df.writeStream\
    .outputMode("append") \
    .trigger(processingTime="9 seconds") \
    .foreach(lambda row: enviar_notificacion(row.nombre, row.productovisto, r
    .start()

```

Este proceso garantiza que los datos de transacciones simuladas sean procesados en tiempo real, permitiendo enviar notificaciones personalizadas a los usuarios en función de su comportamiento de compra en la plataforma. La consulta SQL dentro de Spark filtra y selecciona

solo aquellos usuarios relevantes para la notificación, asegurando un procesamiento eficiente y escalable en un entorno de producción. Además de facilitar el proceso de captación de errores dado que esto se puede reorganizar y ajustar para que no solo envíe un mensaje, si no que se haga un monitoreo completo del por que se demoró el usuario tanto en realizar la compra.

## **5. Resultados y discusión**

Durante la simulación de la recepción y procesamiento de datos, se observaron diferentes comportamientos en función del tiempo de compra (taps) registrado para cada usuario. A continuación, se discuten los resultados obtenidos y se analizan las posibles razones detrás de estos comportamientos, dando el enfoque a la problemática planteada y recopilando los datos necesarios para optimizar el proceso.

### **5.1. Resultados Obtenidos: Mensajes No Enviados (Tiempo de compra nulo)**

Desde la plataforma, se envían los datos a la maqueta establecida donde se recopila el tiempo de compra del usuario, esto quiere decir que es el momento cuando esta ingresa a la plataforma y luego confirma la compra. Cuando este tiempo es nulo, quiere decir que el usuario no completó la compra y no se procesan los datos, ya que no se considera apropiado que el usuario se le notifique que el artículo fue visto o que no fue comprado. En la figura 7 se puede apreciar que "Jhon" solo estuvo en la plataforma y no compró nada, de esta forma estas personas se pueden procesar a través de otro Broker y darle prioridad a los que si compraron.

```
{
  "id": "062292fd7b7e3e688f29418e",
  "nombre": "Scarlett",
  "apellido": "Morris",
  "edad": 44,
  "email": "Scarlett.Morris@mail.com",
  "telefono": "+70 976-874-0232",
  "ubicacion": {
    "direccion": "123 Abbey Road",
    "ciudad": "Lakewood",
    "pais": "Inverness"
  },
  "productovisto": "Colch\u00f3n de espuma viscoel\u00e1stica",
  "taps": 17.8,
  "folio": "SRZ-5509"
}

{
  "id": "6b28427ff85faa10eb5321dc",
  "nombre": "John",
  "apellido": "Harrison",
  "edad": 63,
  "email": "John.Harrison@mail.com",
  "telefono": "+94 585-941-9707",
  "ubicacion": {
    "direccion": "789 Via Milano",
    "ciudad": "Rosewood",
    "pais": "Queenshire"
  },
  "productovisto": "Kit de maquillaje profesional",
  "taps": 0,
  "folio": null
}

{
  "id": "4458865aac12aaecbd462e45",
  "nombre": "Eduardo",
  "apellido": "Wright",
  "edad": 39,
  "email": "Eduardo.Wright@mail.com",
  "telefono": "+43 276-162-7851",
  "ubicacion": {
    "direccion": "123 Oak Street",
    "ciudad": "Willow Creek",
    "pais": "Hartford"
  },
  "productovisto": "Reloj inteligente",
  "taps": 18.1,
  "folio": "FMR-8760"
}

{
  "id": "73ea7cf767df0272481549db",
  "nombre": "Luis",
  "apellido": "Reed",
  "edad": 66,
  "email": "Luis.Reed@mail.com",
  "telefono": "+89 954-043-1254",
  "ubicacion": {
    "direccion": "456 Calle Granada",
    "ciudad": "Northville",
    "pais": "Hartford"
  },
  "productovisto": "Barra de sonido para TV",
  "taps": 0,
  "folio": null
}

"taps": 27.9,
"folio": "ALG-2482"
}

^CTraceback (most recent call last):
  File "transmisor_datos_kafka.py", line 27, in <module>
    time.sleep(2) # Esperar 2 segundos antes de enviar el pr\u00f3ximo
[appuser@267b03429bb4 producer]$ i
KeyboardInterrupt
```

Estimad@ Scarlett,

¡Esperamos que est\u00e9s bien! Quer\u00edamos recordarte que has adquirido el producto 'Colch\u00f3n de espuma viscoel\u00e1stica' en nuestra tienda en l\u00ednea de Falabella. Tu n\u00famero de folio es: SRZ-5509, donde te demoraste un tiempo corto en adquirirlo, te invitamos a seguir comprando en nuestra platadorma.

Saludos.

Estimad@ Eduardo,

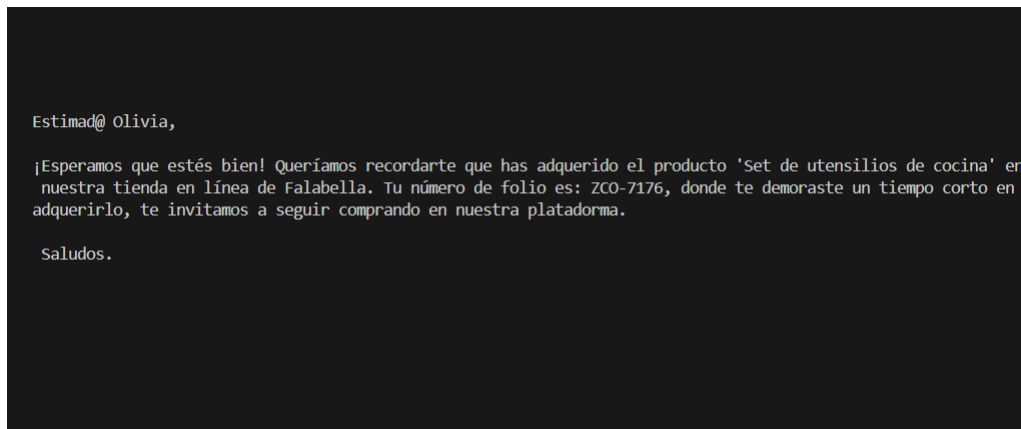
¡Esperamos que est\u00e9s bien! Quer\u00edamos recordarte que has adquirido el producto 'Reloj inteligente' en nuestra tienda en l\u00ednea de Falabella. Tu n\u00famero de folio es: FMR-8760, donde te demoraste un tiempo corto en adquirirlo, te invitamos a seguir comprando en nuestra platadorma.

Saludos.

Figura 7: Ejemplos de no compra

## 5.2. Resultados Obtenidos: Mensajes Enviados con Tiempo de Compra Menor a 30 minutos

Cuando se tienen compras menores a 30 min donde el usuario no tuvo inconvenientes para realizar la transacci\u00f3n, el mensaje que aparece es uno el cual muestra que el periodo de tiempo de la compra fue corto y con esto se busca incentivar al usuario a continuar interactuando con la plataforma, este mensaje es mas directo y se orienta a la repetici\u00f3n de la compra como se logra apreciar en la figura 8.

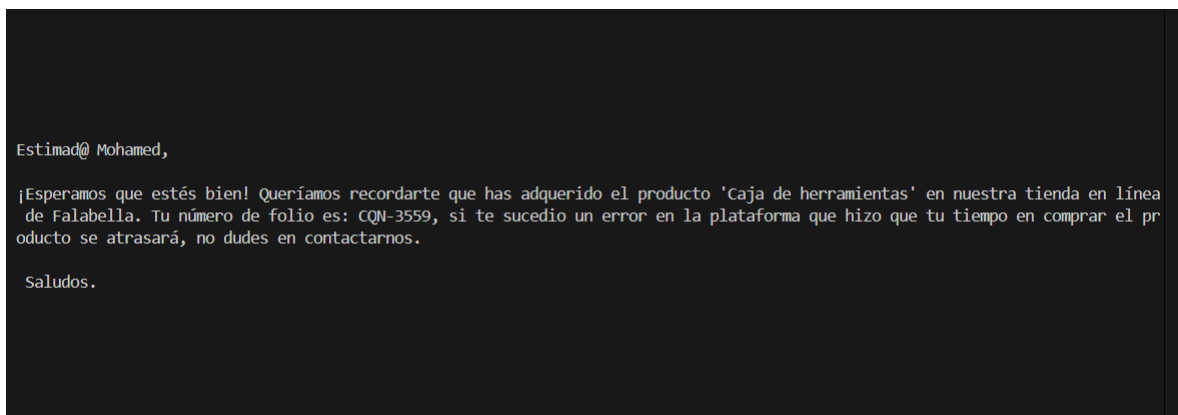


```
Estimad@ Olivia,  
  
¡Esperamos que estés bien! Queríamos recordarte que has adquirido el producto 'Set de utensilios de cocina' en  
nuestra tienda en línea de Falabella. Tu número de folio es: ZCO-7176, donde te demoraste un tiempo corto en  
adquierirlo, te invitamos a seguir comprando en nuestra plataforma.  
  
Saludos.
```

Figura 8: Corto tiempo de compra

### 5.3. Resultados Obtenidos: Mensajes Enviados con Tiempo de Compra Mayor a 30 minutos

Por ultimo se tienen las compras mayores a 30 minutos donde se envia un mensaje diferente que busca que el usuario pueda explicar los problemas que tuvo contactando directamente al servicio, esto es con el fin de generar confianza con el usuario y que cualquier inconveniente generado será atendido a la brevedad para mejorar la plataforma. Actualmente esto solo esta en forma de mensaje, dado que es una maqueta y generar una aplicación que procesará los datos puede conllevar meses incluso años, pero la idea basica de esto es que este tipo de problemas se procesen por otro Broker replicando el servicio y que se distribuya la carga de mejor manera.



```
Estimad@ Mohamed,  
  
¡Esperamos que estés bien! Queríamos recordarte que has adquirido el producto 'Caja de herramientas' en nuestra tienda en línea  
de Falabella. Tu número de folio es: CQN-3559, si te sucedio un error en la plataforma que hizo que tu tiempo en comprar el pr  
oducto se atrasará, no dudes en contactarnos.  
  
Saludos.
```

Figura 9: Alto tiempo de compra

## 5.4. Discusión de Resultados

Estos resultados destacan la importancia de personalizar las comunicaciones con los usuarios en función de su comportamiento específico en la plataforma. La implementación de un sistema que distingue entre diferentes tiempos de compra permite ajustar las estrategias de notificación para maximizar el impacto positivo en la experiencia del usuario y así hacer que la plataforma sea mucho mas eficiente a la hora de un aumento masivo de trafico de datos que pueden llevar a perdidas millonarias para la empresa, provocando que los sistemas colapsen y no sean levantados nuevamente.

La implementación de este sistema tiene costos asociados, como el desarrollo de lógica específica para el manejo de diferentes casos de uso y el procesamiento en tiempo real de los datos. Sin embargo, los beneficios potenciales, como la mejora en la retención de clientes y el aumento de las conversiones, justifican estos costos. Al adaptar las notificaciones a las circunstancias individuales de cada usuario, se pueden mejorar significativamente los índices de satisfacción y fidelización. Por otra parte existen limitaciones en cuanto al enfoque de esta solución la cual es la dependencia de la calidad de los datos recibidos. Si el tiempo de compra no se registra correctamente o es nulo en casos donde no debería serlo, el sistema podría no comportarse como se espera. Para mitigar esto, es crucial contar con mecanismos de validación y corrección de datos en tiempo real.

En futuras implementaciones, se podría considerar la integración de análisis predictivo para anticipar el comportamiento del usuario basado en patrones históricos, permitiendo una personalización aún más avanzada de las notificaciones.

## 6. Conclusión

En este informe, hemos explorado la implementación de una arquitectura distribuida utilizando Apache Kafka y Apache Spark para gestionar y procesar datos en tiempo real en un entorno de comercio electrónico. A partir de la simulación de escenarios como el colapso de la plataforma de Falabella durante eventos de alta demanda, se ha demostrado la capacidad de estas tecnologías para mejorar la escalabilidad y resiliencia de los sistemas de procesamiento de datos.

Una de las principales ventajas de la solución propuesta es su capacidad de escalar horizontalmente. Apache Kafka permite manejar grandes volúmenes de datos mediante la distribución eficiente de los mensajes entre múltiples brokers, mientras que Apache Spark distribuye el procesamiento de datos entre varios workers, lo que permite gestionar la carga incluso en picos de demanda extrema. Esta escalabilidad no solo mejora el rendimiento del sistema, sino que también garantiza la disponibilidad continua del servicio, un aspecto crítico en el contexto de comercio electrónico.

La implementación de una arquitectura distribuida con Apache Kafka y Apache Spark implica costos organizacionales relacionados con la infraestructura y la administración de clústeres. Sin embargo, estos costos pueden ser justificados por la reducción en tiempos de inactividad y la mejora en la experiencia del usuario, lo que se traduce en una mayor satisfacción del cliente y una posible reducción en pérdidas financieras durante eventos de alta demanda.

Las principales ventajas de esta arquitectura incluyen la alta disponibilidad, la resiliencia ante fallas, y la capacidad de manejar grandes volúmenes de datos en tiempo real. Sin embargo, también existen desventajas, como la complejidad en la configuración y mantenimiento de los clústeres, así como la necesidad de una infraestructura robusta que pueda soportar el tráfico de datos a gran escala.

### 6.1. Limitaciones y Trabajo Futuro

A pesar de las ventajas mencionadas, existen limitaciones en la implementación actual. Una de ellas es la dependencia de una infraestructura física o en la nube que pueda

escalar adecuadamente para soportar los clústeres de Kafka y Spark. Además, la latencia introducida por la replicación de datos y la comunicación entre brokers y workers puede impactar el rendimiento en situaciones de extrema carga.

Para trabajos futuros, se sugiere explorar la optimización de la replicación de datos en Kafka para reducir la latencia, así como la integración de herramientas de monitoreo avanzadas para prever y mitigar problemas antes de que impacten al usuario final. También sería beneficioso investigar la posibilidad de implementar arquitecturas híbridas que combinen lo mejor de las soluciones en la nube y on-premise para maximizar la flexibilidad y reducir costos.

# Bibliografía

Boltic (2024). Kafka clusters. <https://www.boltic.io/blog/kafka-clusters>. Accessed: 2024-08-23.

Hora de Noticias (2023). SERNAC llama a todos los clientes de Falabella y París a aportar datos para comenzar demanda colectiva. Último acceso: 23 de agosto de 2024.

KeepCoding (2024). Cluster manager en apache spark: Qué es y cómo funciona. <https://keepcoding.io/blog/cluster-manager-spark/>. Último acceso: 24 de agosto de 2024.