PROYECTO FIN DE GRADO

TÍTULO: Arquitectura Data-Driven: Análisis, Diseño e Implementación mediante plataforma Apache Kafka

AUTOR/A: Yanjun Chen

TITULACIÓN: Telemática

DIRECTOR/A: Javier Poveda Barbero

TUTOR/A: Pablo Ramírez Ledesma

DEPARTAMENTO: DTE

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Carlos Felipe Rueda Frías

TUTOR/A: Pablo Ramírez Ledesma

SECRETARIO/A: Aurelio Berges García

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Agradecimientos

Agradecimientos

Agradecimientos

En primer lugar, quiero expresar mi más profundo agradecimiento a la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la Universidad Politécnica de Madrid por concederme el honor de estudiar en esta escuela.

Expreso mi más sincero agradecimiento a Javier Poveda Barbero, Faustino Tello Caballo, los tutores de la empresa, y a Pablo Ramírez Ledesma, tutor académico, por sus invaluable guía, apoyo y paciencia durante el desarrollo de este proyecto. Su experiencia y conocimientos han sido fundamentales para la culminación exitosa de este trabajo.

A mis profesores y profesoras, quienes, a lo largo de estos años, han contribuido significativamente a mi formación académica y profesional. Su dedicación y compromiso con la enseñanza me han inspirado a seguir adelante y superar los desafíos.

A mis compañeros de clase, por su compañerismo y apoyo constante. Juntos hemos enfrentado los retos académicos y compartido innumerables momentos de aprendizaje y crecimiento personal.

A mi familia, por su amor incondicional y su apoyo en cada etapa de mi vida. Sus palabras de aliento y su fe en mis capacidades me han motivado a seguir adelante, incluso en los momentos más difíciles.

Finalmente, quiero agradecer a CRIDA A.I.E. por su colaboración y apoyo durante el desarrollo de este proyecto.

A todos ustedes, gracias por hacer posible la realización de este sueño.

Agradecimientos

Abstract

Abstract

In this new era digital age, organizations generate and collect vast amounts of data from a variety of sources and turning this data into valuable, actionable information is critical to making strategic decisions, improving operational efficiency, and generating positive business impact. Data-Driven architecture, based on technologies such as Apache Kafka, provides a robust framework for large-scale data management and analysis. As a result of the project, this architecture can ingest and process large volumes of data from various sources in real time and can store and manage them in a scalable and secure way and then analyze and process this data to extract valuable information and visualizations that help the strategic decision making of the different departments of the organization.

Resumen

En esta nueva era digital, las organizaciones generan y recopilan vastas cantidades de datos a partir de diversas fuentes y convertir estos datos en información valiosa y procesable es fundamental para tomar decisiones estratégicas, mejorar la eficiencia operativa y generar un impacto positivo en el negocio. Una arquitectura Data-Driven, basada en tecnologías como Apache Kafka, ofrece un marco robusto para la gestión y el análisis de datos a gran escala. Como resultado del proyecto, esta arquitectura puede ingerir y procesar grandes volúmenes de datos de diversas fuentes en tiempo real pudiendo almacenarlos y gestionarlos de manera escalable y segura para posteriormente poder analizar y procesar estos datos para extraer información valiosa y realizar visualizaciones que ayudan la toma de decisiones estratégicas de los diferentes departamentos de la organización.

Keywords; Kafka; Zookeeper; Topics; Change Data Capture (CDC); Brocker.

Abstract

Contenido

Contenido

Contenido

Contenido

Contenido

Lista de Acrónimos

Lista de Acrónimos

Tabla 1: Lista de Acrónimos

Acrónimo	Definición completa
CDC	*Change Data Capture*
SQL	*Structured Query Language*
IoT	*Internet of Things*
TI	*Tecnología de la información*
API	*Application Programming Interface*
JDK	*Java Development Kit*
IDE	*Integrated Development Enviroment*
YAML	*YAML Ain't Markup Language*
JSON	*JavaScript Object Notation*
AES	*Advanced Encryption Standard*
JAR	*Java Archive*
SSL	*Secure Sockets Layer*
TLS	*Trasport Layer Security*
RBAC	*Role-Based Access Control*
GDPR	*General Data Protection Regulation*
HIPAA	*Health Insurance Portability and Accountability Act*

Lista de Acrónimos

Índice de Graficas

Índice de Graficas

Índice de Graficas

Índice de Tablas

Índice de Tablas

Índice de Tablas

Índice de Códigos

Índice de Códigos

Índice de Códigos

1. Introducción a las arquitecturas Data-Driven

1. Introducción a las arquitecturas Data-Driven

Las arquitecturas Data-Driven son fundamentales en el panorama actual, donde los datos se han convertido en un activo crítico para las organizaciones. Estas arquitecturas se centran en la recopilación, procesamiento, análisis y utilización de datos para la toma de decisiones informadas y la generación de valor.

Actualmente, la toma de decisiones basadas en datos se ha convertido en un factor fundamental para el éxito de las organizaciones, muchas empresas pueden aprovechar el poder de sus datos para obtener información en tiempo real y tomar decisiones ágiles para obtener una ventaja competitiva significativa. En este trabajo se propone un proyecto que implementa una arquitectura Data-Driven para la obtención y el análisis de datos en tiempo real, con el objetivo de impulsar la toma de decisiones estratégicas en la organización. Se trata de una arquitectura diseñada para apoyar a las empresas que tratan mucho con los datos para que puedan analizar aquellos datos lo antes posible y con eficiencia.

1.1 Motivación

Tras la aparición de las plataformas de colas como IBM MQ que proporciona una base de mensajería universal para la conectividad robusta y el tratamiento de los datos no en tiempo real, pocos años después desarrollaron se desarrollaron plataformas como Apache Kafka que permite a las empresas recibir y producir datos en tiempo real que benefician a la organización.

En la actualidad predomina la automatización de estos datos y que las organizaciones a partir de estas automatizaciones obtengan directamente el resultado final para el apoyo de las decisiones, todo el proceso es automático, simple y sencillo.

1.2 Definición

Una arquitectura Data-Driven es un marco de trabajo tecnológico que permite a las organizaciones recopilar, almacenar, procesar y analizar datos de diversas fuentes para obtener información significativa y tomar decisiones basadas en datos. La toma de estos datos aporta posteriormente a las decisiones de una organización, lo cual se centra en los objetivos de negocio y de los entornos digitales de esos objetivos.

La arquitectura Data-Driven es una arquitectura totalmente automatizada, ligera y simple que, mediante el aprovechamiento de los datos como un activo estratégico, impulsa la toma de decisiones con el objetivo de identificar los clientes más potentes, reducir consumo, reducir costo y optimizar cadena de tecnologías, entre otros objetivos.

1.3 Funcionalidad

Como se indica en la Fig. 1, esta arquitectura, nos aporta la capacidad de integrar los beneficios internos de una organización con sus objetivos tanto internos como externos y los beneficios externos.

1. Introducción a las arquitecturas Data-Driven

Igual que todas las aplicaciones, la Arquitectura Data-Driven tiene unas funcionalidades y debe tener la capacidad de:

- Recopilación de datos desde diferentes fuentes, como aplicación, sistemas remotos, etc.
- Con los datos recopilados almacenarlos en un sistema de almacenamiento diseñados para manejar grandes volúmenes de datos como almacenamiento en la nube o sistemas de Archivos Distribuidos, etc.
- Con los datos almacenados, procesarlos y transformarlos para ser analizado, dentro de esto lleva limpieza de datos, integración, transformación y enriquecimiento de datos.
- En la fase de análisis, los datos procesados se analizan utilizando técnicas como análisis estadístico, aprendizaje automático, inteligencia artificial, minería de datos y entre otras para generación de información útil.
- Después de todas estas fases, la información derivada ya es comprensible y visualmente atractiva para facilitar la compresión y toma de decisión.

Beneficios de utilizar una arquitectura Data-Driven:

- Permite a las organizaciones tomar decisiones basadas en datos en lugar de suposiciones.
- Facilita la identificación de áreas de mejora y optimización en los procesos.
- Ayuda a identificar nuevas oportunidades de negocio, tendencias del mercado y necesidades del cliente.
- Permite a la personalización y segmentación de productos, servicios y experiencias basadas en el comportamiento y las preferencias del cliente.
- Facilita la compresión del comportamiento y las preferencias del cliente para ofrecer experiencias personalizadas y satisfactorias.
- Los procesos se automatizan en base a los datos, liberando tiempo y recursos para tareas más estratégicas.
- Se reduce el tiempo y el coste de la toma de decisiones.
- Se pueden realizar pruebas y experimentos de forma más rápida y eficiente.
- Permite a las empresas tomar decisiones más transparentes y responsables.

Desafíos de una Arquitectura Data-Driven:

- La gestión de grandes volúmenes de datos de diferentes fuentes puede ser compleja y desafiante.
- La seguridad y la privacidad de los datos son preocupaciones importantes en las arquitecturas Data-Driven, especialmente cuando se trata de datos sensibles.
- Garantizar la calidad de los datos es crucial para obtener resultados precisos y confiables del análisis.
- Adoptar una arquitectura Data-Driven puede requerir cambios culturales dentro de la organización, donde los datos se convierten en el fundamento de las tomas de decisiones.

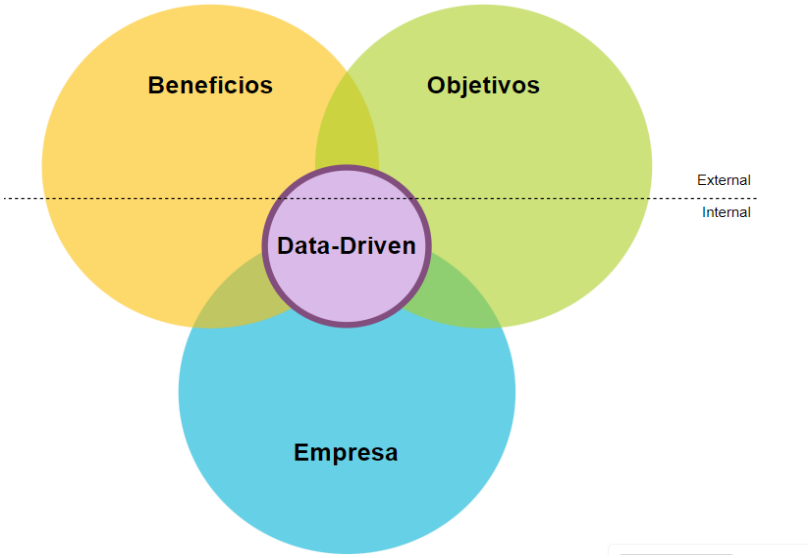


Fig. 1: Introducción de la Arquitectura Data-Driven.

Fuente. Elaboración propia.

1.4 Resumen:

En este proyecto se va a desarrollar una arquitectura que analiza y transforma datos de una máquina fuente a otra máquina remota mediante la plataforma de Apache Kafka. Una vez obtenidos los datos en la máquina remota se procede a transformaciones de estos datos mediante un programa escrito en Java, que posteriormente podrán observar en el Anexo, para descartar informaciones irrelevantes y quedar con las informaciones importantes para el apoyo de las decisiones que tomará la empresa.

2. Características de la Arquitectura Data-Driven

2. Características de la Arquitectura Data-Driven

La arquitectura Data-Driven como se ha dicho en la introducción tiende a ser una tecnología innovadora, eficiente y fácil de usar. Como todos los proyectos, la Arquitectura Data-Driven también tienes unos objetivos, alcances y características que se presenta a continuación.

2.1 Objetivos

Esta arquitectura tiene unos objetivos claros y precisos que llevara a cabo durante esta memoria y que se puede ver en la Figura 2 a continuación.

1. **Facilitar el acceso y la gestión de datos:**
    - **Organizar los datos de forma eficiente**: La arquitectura debe clasificar y estructurar los datos de manera meticulosa para que su búsqueda, análisis y uso sean lo más sencillos y eficientes posible incluyendo la creación de categorías lógicas, la utilización de esquemas consistentes y la implementación de mecanismos de indexación que permitan encontrar los datos de manera rápida y precisa. Además, es fundamental mantener una documentación exhaustiva sobre la estructura y organización de los datos, para que todos los usuarios puedan entender y utilizar el sistema de manera efectiva.
    - **Implementar tecnologías de acceso**: Es esencial emplear herramientas y tecnologías que permitan a los usuarios acceder a los datos de manera rápida y sencilla, sin requerir conocimientos técnicos avanzados que debe incluir la implementación de interfaces de usuario intuitivas, sistemas de gestión de bases de datos accesibles, y el uso de APIs que faciliten la integración de los datos con otras aplicaciones, es importante considerar la capacitación y el soporte continuo para los usuarios, asegurando que puedan aprovechar al máximo las tecnologías disponibles.
    - **Garantizar la calidad de los datos**: La arquitectura debe asegurar que los datos sean precisos, completos y confiables para que las decisiones basadas en ellos sean informadas y efectivas para la implementación de controles de calidad rigurosos, incluyendo validaciones automáticas y manuales, auditorías regulares de los datos y mecanismos de retroalimentación que permitan a los usuarios reportar y corregir errores, es crucial establecer políticas claras para la gestión de datos, incluyendo la definición de responsabilidades y procedimientos para la actualización y mantenimiento de la información, asegurando así su integridad y relevancia a lo largo del tiempo.
2. **Apoyar el análisis y la toma de decisiones:**
    - **Proporcionar herramientas de análisis:** La arquitectura debe ofrecer una variedad de herramientas para analizar los datos y obtener información útil para el negocio. Las herramientas deben ser accesibles para usuarios con diferentes niveles de habilidad técnica permitiendo que todos los miembros de la organización puedan manejarla sin dificultad y generar informe detallada que apoye a la toma de decisiones estratégicas.

2. Características de la Arquitectura Data-Driven

- **Integrar los datos con los procesos de negocio:** Los datos deben estar disponibles para ser utilizados en los diferentes procesos de la empresa, como la producción, el marketing o las ventas. Esto quiere decir que el proyecto debe permitir transferencia fluida de datos entre diferentes departamentos y aplicaciones eliminando silos de información y facilitando una visión unificada del negocio.
- **Fomentar una cultura basada en datos:** La arquitectura debe promover el uso de datos para tomar decisiones en todos los niveles de la organización, esto conlleva a que la organización debe de tener una formación a sus empleados para adaptarse a esta nueva cultura basada en datos.

3. **Asegurar la escalabilidad y la flexibilidad:**
   - **Adaptarse al crecimiento de los datos:** Una de las capacidades esenciales que debe tener una plataforma Data-Driven es la habilidad de adaptarse al constante crecimiento del volumen y la variedad de datos. En este contexto, la implementación de Apache Kafka es fundamental debido a su diseño robusto para manejar flujos de datos en tiempo real y a gran escala permitiendo la ingesta continua de datos de múltiples fuentes, escalando horizontalmente sin perder eficiencia.
   - **Evolucionar con las necesidades del negocio:** Otra característica clave de una plataforma Data-Driven es su capacidad para evolucionar conforme cambian las necesidades del negocio. En este sentido, Apache Kafka se destaca por su flexibilidad y adaptabilidad ya que permite a las organizaciones ajustar y reconfigurar sus pipelines de datos de manera dinámica, integrando nuevas fuentes de datos, ajustando flujos existentes y adaptándose rápidamente a nuevas demandas del mercado incluyendo la capacidad de implementar nuevas tecnologías y herramientas de análisis, asegurando que la plataforma pueda seguir siendo relevante y efectiva a medida que el negocio crece y evoluciona.
   - **Reducir los costes de almacenamiento y procesamiento:** La reducción de costes de almacenamiento y procesamiento es un objetivo crítico para cualquier plataforma Data-Driven. Apache Kafka contribuye significativamente a este objetivo mediante su diseño eficiente y escalable permitiendo la retención de grandes volúmenes de datos en sus clústeres, optimizando el uso del almacenamiento mediante la compactación de logs y la gestión eficiente de la retención de datos. Además, Kafka es altamente eficiente en términos de procesamiento de datos, lo que reduce la necesidad de recursos computacionales excesivos con la capacidad para manejar flujos de datos en tiempo real minimiza la latencia y mejora el rendimiento general del sistema, lo que se traduce en una reducción de costes operativos.
4. **Proteger la seguridad y la privacidad de los datos:**
   - **Implementar medidas de seguridad:** Nuestra plataforma Data-Driven debe implementar robustas medidas de seguridad para proteger los datos contra accesos no autorizados, robos o daños. Apache Kafka, con su sólida arquitectura de seguridad, permite asegurar los datos a través de varias capas de protección que incluye la autenticación mediante protocolos como Kerberos y SSL/TLS para cifrar la comunicación entre los productores, consumidores y el clúster de Kafka. Además, Kafka ofrece control de acceso basado en roles (RBAC), lo que

2. Características de la Arquitectura Data-Driven

permite definir permisos específicos para diferentes usuarios y aplicaciones, asegurando que solo el personal autorizado pueda acceder y manipular los datos sensibles.

- **Garantizar el cumplimiento de las normativas:** La arquitectura de una plataforma Data-Driven debe cumplir con todas las leyes y regulaciones relacionadas con la privacidad de los datos, nuestra arquitectura cumple este cumplimiento mediante capacidades avanzadas de auditoría y monitoreo pudiendo configurar la arquitectura para registrar todas las actividades y accesos a los datos, lo que proporciona un rastro de auditoría completo que es esencial para cumplir con normativas como GDPR, HIPAA y otras legislaciones de privacidad. Además, las políticas de retención y eliminación de datos de Kafka aseguran que los datos se gestionen de acuerdo con los requisitos legales, eliminando datos cuando ya no son necesarios o cuando se requiere por ley.
- **Concienciar a los usuarios sobre la importancia de la seguridad de los datos:** Promover el uso responsable de los datos por parte de los usuarios es fundamental para una plataforma Data-Driven efectiva, esta arquitectura no solo proporciona las herramientas técnicas para asegurar los datos, sino que también puede ser una parte integral de los programas de concienciación y capacitación en seguridad de datos dentro de la organización. Implementaciones de políticas claras de uso de datos y ofrecer capacitación regular a los empleados sobre las mejores prácticas en seguridad de datos ayuda a fomentar una cultura de responsabilidad y vigilancia.

Fig. 2: Esquema de Objetivos de una Arquitectura Data-Driven.

Fuente. Elaboración propia.

2. Características de la Arquitectura Data-Driven

2.2 Alcance

Como alcance se ha establecido 3 puntos como se puede apreciar en la figura 3 de la página siguiente:

1. **Creación de nuevos productos y servicios**
   - Los datos pueden usarse para identificar las necesidades y preferencias de los clientes.
   - Las empresas pueden usar esta información para desarrollar productos y servicios que satisfagan mejor las demandas del mercado ayudando a las empresas a aumentar sus ingresos y a obtener una ventaja competitiva.
2. **Aumento de la agilidad y la capacidad de respuesta**
   - Las arquitecturas Data-Driven permiten a las empresas adaptarse rápidamente a los cambios en el mercado y en las necesidades de los clientes.
   - Los datos proporcionan información en tiempo real que permite a las empresas tomar decisiones ágiles y estratégicas.
3. **Conectividad entre datos fuentes y publicación de los datos procesados a los clientes**
   - La arquitectura Data-Driven debe de ser capaz de captar de diversas fuentes y procesar estos datos para posteriormente ser consumidos.
   - Los datos procesados deben de ser capaces de ser compartidos a sus clientes.

Fig. 3: Esquema de Alcance de la Arquitectura Data-Driven.

Fuente. Elaboración propia.

2. Características de la Arquitectura Data-Driven

2.3 Política de acceso y seguridad

1. **Implementación de medidas de seguridad técnicas**
   - Se debe implementar la encriptación de los datos utilizando encriptación simétrica y algoritmos de encriptación avanzados, como por ejemplo el algoritmo de encriptación AES (Advanced Encryption Standard) para garantizar la confidencialidad y la integridad de la información asegurando que solo los usuarios autorizados puedan acceder a la información.
2. **Implementar medidas de seguridad físicas**
   - Acceso a los servidores físico y dispositivos de almacenamiento restringido para reducir el riesgo de accesos no autorizados o de manipulaciones físicas que puedan comprometer la seguridad de los datos.
   - Realizar copias de seguridad y almacenándolas en ubicaciones seguras para protegerse contra eventos catastróficos no controlados, es recomendable realizar copias de seguridad cada periodo de tiempo según la importancia de los datos.

2.4 Capacitación y soporte

- Capacitación/Formación
  - Identificar las necesidades de capacitación de los diferentes grupos de usuarios, como los productores de fuentes, los analistas de datos y los usuarios finales.
  - Desarrollar programas de capacitación que brinden a los usuarios los conocimientos y habilidades necesarios para utilizar la arquitectura Data-Driven de manera efectiva.
  - Ofrecer diferentes formatos de capacitación, para adaptarse a las diferentes necesidades de los usuarios.
  - Brindar continuo a los usuarios a través de recursos como documentación y tutoriales.
- Soporte
  - Establecer un equipo de soporte dedicado a ayudar a los usuarios con la arquitectura Data-Driven.
  - Proporcionar diferentes canales de soporte, como teléfono, correo electrónico y chat en línea, para que los usuarios puedan obtener ayuda de la manera más conveniente para ellos.
  - Documentar problemas y las soluciones para que los usuarios puedan encontrar información útil en caso de que experimenten problemas similares.
  - Monitorizar el uso de la arquitectura Data-Driven para identificar áreas donde se puede mejorar la capacitación y el soporte.

2.5 Flujo de trabajo

Todas las arquitecturas tienen un flujo de funcionamiento, la arquitectura Data-Driven igual que el resto de las arquitecturas también tiene su propio flujo de trabajo como se puede ver en la figura 4 y que se describe a continuación:

1. **Estudio y planificación**

2. Características de la Arquitectura Data-Driven

En esta etapa, se debe de realizar un estudio de los datos que se van a recolectar y la planificación de los procesos tanto de análisis, preparación de los datos como la transformación y almacenamiento de los resultados.

2. **Recopilación de datos**
   En esta etapa, se deben identificar y recopilar los datos necesarios para alcanzar los objetivos. Los datos pueden provenir de diversas fuentes, como sistemas internos, bases de datos externas, sensores, dispositivos móviles, etc. En nuestro escenario los datos recopilados vienen en un base de datos.

3. **Limpieza y transformación de datos**
   En esta etapa, se deben limpiar y transformar los datos para que sean aptos para el análisis. Esto implica eliminar datos duplicados o incorrectos, normalizar los datos y convertirlos a un formato adecuado.

4. **Almacenamiento de datos**
   En esta etapa, se deben almacenar los datos de forma segura y accesible. Se pueden utilizar diferentes tecnologías de almacenamiento, como bases de datos, data lakes o data Warehouse, en nuestro caso H2.

5. **Análisis y preparación de datos**
   En esta etapa, se deben analizar los datos para obtener información útil para la toma de decisiones. Se pueden utilizar diferentes técnicas de análisis, como visualización de datos, machine learning, análisis estadístico, etc.

6. **Publicación y Toma de decisiones**
   En esta etapa, se deben tomar decisiones basadas en la información obtenida del análisis de datos. Las decisiones deben ser relevantes para los objetivos definidos en la primera etapa.

7. **Implementación de las decisiones**
   En esta etapa, se deben implementar las decisiones tomadas en la etapa anterior. Esto puede implicar cambios en los procesos de negocio, la tecnología o la organización.

8. **Monitorización, evaluación y almacenamientos de las decisiones junto a los datos**
   En esta etapa, se debe monitorizar el impacto de las decisiones tomadas y evaluar el éxito de la arquitectura Data-Driven. Es importante realizar ajustes en la arquitectura de forma regular para asegurar que se siguen alcanzando los objetivos.

9. **Reutilización de los Datos**
   Una vez que los datos se hayan almacenado con las decisiones, estos datos pueden tener un segundo uso dependiendo de la aplicación que tiene esos datos sacándolo del lugar almacenado.

2. Características de la Arquitectura Data-Driven

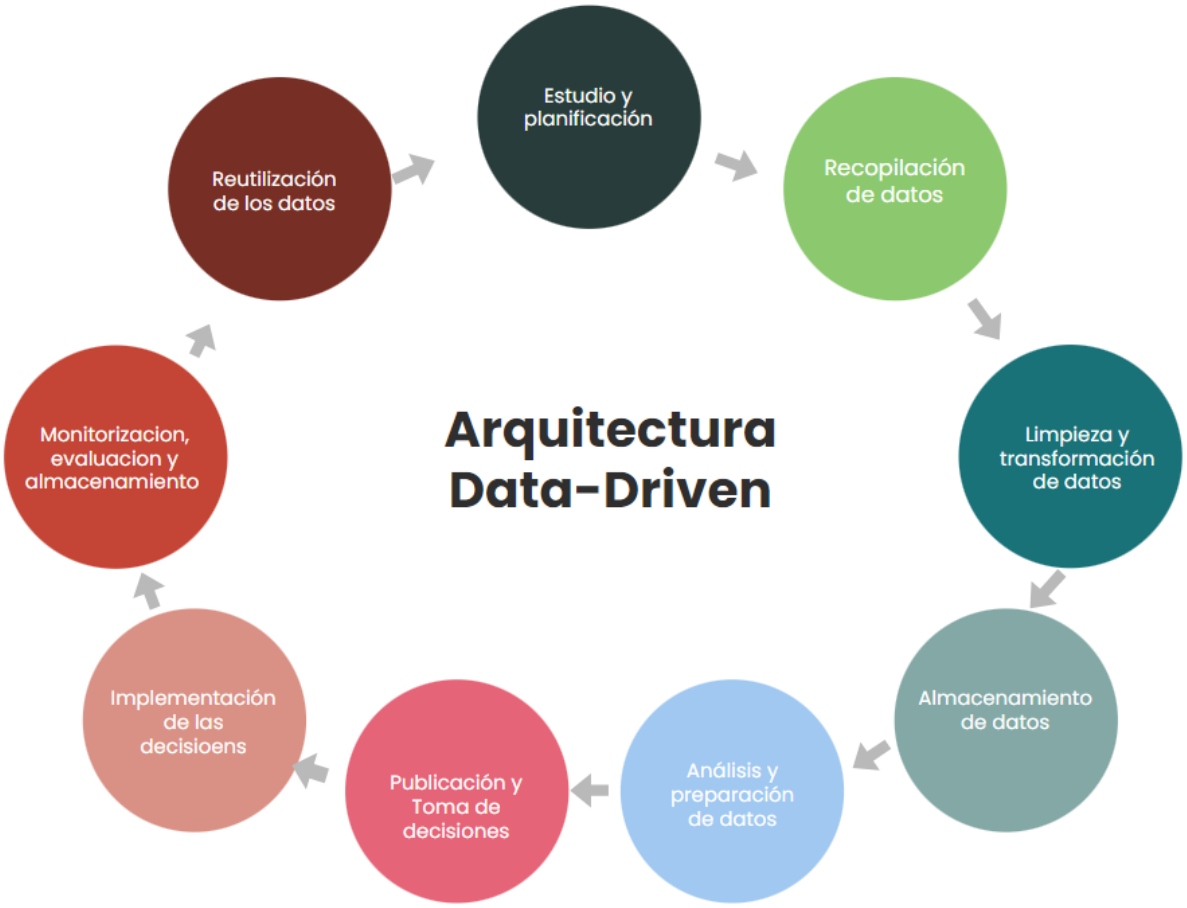


Fig. 4: Flujo de funcionamiento de la Arquitectura Data-Driven.

Fuente. Elaboración propia.

3. Uso en aplicaciones

3. Uso en aplicaciones

La arquitectura Data-Driven está desarrollada para ser utilizada en diversas áreas de tecnología, como ejemplo se anotan unas posibles áreas de aplicación como se puede apreciar en la figura 6.

1. **Procesamiento de Flujos de Datos en Tiempo Real**
   - **Análisis de logs**: La recopilación y análisis exhaustivo de logs en tiempo real constituye una herramienta indispensable para las operaciones de TI, permitiendo la detección temprana de errores, fallos de seguridad y otras anomalías críticas. Este proceso implica la monitorización continua y el análisis detallado de los registros generados por sistemas y aplicaciones, facilitando así la identificación y resolución rápida de problemas que podrían afectar la estabilidad y seguridad de las infraestructuras tecnológicas. La capacidad de reaccionar ante incidentes en tiempo real mejora significativamente la resiliencia y la eficiencia operativa de las organizaciones.
   - **Análisis de redes sociales**: La recopilación y análisis de datos de redes sociales en tiempo real es crucial para las estrategias de marketing digital, ofreciendo perspectivas valiosas sobre tendencias actuales, comportamiento del usuario y la eficacia de campañas publicitarias. Al entender cómo interactúan los usuarios con el contenido y qué temas generan mayor compromiso, las marcas pueden adaptar sus mensajes y campañas para resonar mejor con su público objetivo, mejorando así su impacto en el mercado y su conexión con los consumidores.
   - **Detección de fraude**: La detección de transacciones fraudulentas en tiempo real mediante el análisis de patrones de comportamiento es fundamental para proteger la integridad financiera de las instituciones y la seguridad de los consumidores. Utilizando tecnologías avanzadas de aprendizaje automático y análisis de datos, las empresas pueden identificar actividades sospechosas casi instantáneamente, permitiendo una respuesta rápida para mitigar potenciales pérdidas. Esta capacidad de intervención temprana es esencial en un entorno económico donde el fraude y la ciberdelincuencia están en constante evolución.
   - **Recomendación de productos**: Proporcionar recomendaciones de productos personalizadas a los usuarios en tiempo real se ha convertido en una práctica estándar para mejorar la experiencia de compra online. Este proceso utiliza algoritmos avanzados que analizan el historial de compras y el comportamiento de navegación de los usuarios para ofrecer sugerencias que sean altamente relevantes y atractivas para cada individuo. Implementar este tipo de recomendaciones personalizadas no solo incrementa la probabilidad de conversión, sino que también fomenta una mayor lealtad del cliente al demostrar un entendimiento profundo y cuidadoso de sus preferencias y necesidades individuales.
   - **Personalización de contenido**: La personalización del contenido que se muestra a los usuarios en tiempo real es esencial para captar y mantener la atención del público en el saturado entorno digital actual. Mediante el análisis de intereses y preferencias, es posible ajustar dinámicamente el contenido que se ofrece a cada usuario, desde artículos y videos hasta publicidad y ofertas promocionales. Este enfoque no solo mejora la relevancia del contenido, sino que también mejora

3. Uso en aplicaciones

la experiencia del usuario, haciéndola más agradable y eficaz en términos de compromiso y satisfacción del usuario.

- **Análisis del comportamiento del cliente**: Monitorizar el comportamiento del cliente en tiempo real es crucial para comprender de manera exhaustiva sus necesidades y preferencias. Este análisis implica recoger y evaluar datos sobre cómo los clientes interactúan con servicios y productos en diversos canales. La información obtenida permite a las empresas ajustar sus estrategias de marketing, desarrollo de productos y servicios al cliente, asegurando que estas áreas estén alineadas con las expectativas y comportamientos reales de sus consumidores. Así, el análisis detallado del comportamiento del cliente facilita una toma de decisiones más informada y orientada al usuario, lo cual es vital para el éxito en el competitivo mercado actual.

2. **Integración de Datos**
   - **Integración de datos de diferentes fuentes**: La integración eficaz de datos procedentes de múltiples fuentes en tiempo real es esencial para construir una visión holística y unificada de la información disponible. Este proceso implica combinar datos de diversas procedencias —ya sean internas o externas— para crear una única vista que refleje todas las facetas relevantes del negocio o del fenómeno estudiado. La capacidad de amalgamar esta información de manera cohesiva y en tiempo real permite a las organizaciones tomar decisiones basadas en un compendio completo y actualizado de datos, lo cual es crucial en entornos empresariales que demandan respuestas rápidas y basadas en evidencias claras y abarcadoras.
   - **Sincronización de datos entre diferentes sistemas**: Mantener una sincronización de datos entre diversos sistemas en tiempo real es fundamental para garantizar la coherencia y la exactitud de la información a lo largo de toda la organización. Este proceso asegura que todos los sistemas relevantes dentro de la infraestructura de TI compartan una base de datos consistente y actualizada, lo que elimina discrepancias que podrían llevar a decisiones erróneas o a ineficiencias operativas. La sincronización efectiva en tiempo real es especialmente crítica en entornos donde los procesos de negocio dependen de la precisión y la inmediatez de la información, tales como en las finanzas, la logística y la gestión de cadenas de suministro.

3. **Internet de las Cosas (IoT)**
   - **Recopilación y análisis de datos de dispositivos IoT:** La capacidad de recoger y analizar datos procedentes de dispositivos IoT en tiempo real es fundamental para monitorizar de manera continua el estado y el funcionamiento de estos dispositivos. Este proceso no solo facilita una visión detallada del rendimiento y la salud de los dispositivos conectados, sino que también permite la toma de decisiones informadas y ajustadas en tiempo real, basadas en datos precisos y actualizados.
   - **Control de dispositivos IoT**: La gestión eficiente de dispositivos IoT también implica el control directo y en tiempo real de estos aparatos a través de la infraestructura proporcionada por Apache Kafka. Este método permite el envío de comandos específicos a los dispositivos, facilitando una interacción dinámica y una respuesta rápida a las necesidades emergentes o a los cambios en el entorno operativo. La integración de Kafka como herramienta de comunicación subraya la

3. Uso en aplicaciones

importancia de contar con un sistema robusto y escalable que pueda manejar de forma efectiva las cargas de trabajo generadas por la gestión de múltiples dispositivos IoT simultáneamente.

4. **Machine Learning e Inteligencia Artificial**
   - **Entrenamiento de modelos de Machine Learning**: La tarea de entrenar modelos de aprendizaje automático en tiempo real se ha vuelto esencial en la era de los grandes datos. Utilizando datos en flujo continuo, o Streaming, es posible ajustar y optimizar modelos de manera continua, lo que permite que estos sistemas aprendan y se adapten dinámicamente a nuevos patrones o cambios en los datos existentes. Este enfoque proactivo en el entrenamiento permite a las organizaciones mantener sus modelos a la vanguardia de la precisión y la relevancia, asegurando que las decisiones basadas en estos modelos sean tan informativas y efectivas como sea posible.
   - **Realización de inferencias en tiempo real**: Realizar inferencias utilizando modelos de Machine Learning previamente entrenados en un entorno de tiempo real es otro componente crítico en la cadena de valor del análisis de datos. Esta capacidad de aplicar modelos a datos en vivo para obtener resultados inmediatos transforma la manera en que las empresas pueden responder a eventos críticos en tiempo real. Al implementar inferencias en el momento adecuado, las empresas no solo mejoran la eficiencia operativa, sino que también potencian la toma de decisiones estratégicas, lo que a su vez puede traducirse en una ventaja competitiva significativa en mercados altamente dinámicos.

3. Uso en aplicaciones

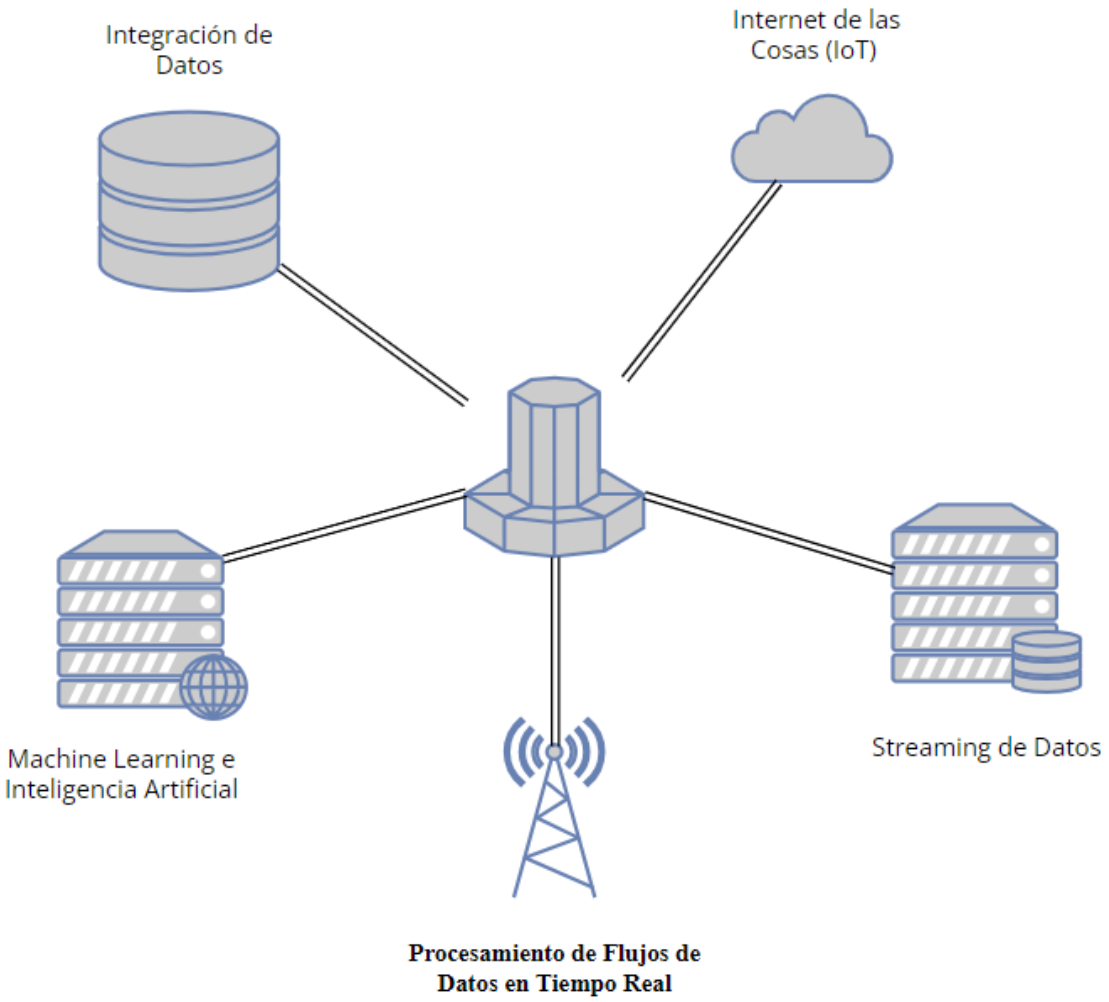


Fig. 5: Aplicación de uso que puede tener la Arquitectura Data-Driven.

Fuente. Elaboración propia.

4. Estado del Arte

4. Estado del Arte

Para la selección de la arquitectura de nuestro proyecto, se ha llevado a cabo una exhaustiva evaluación de diversas tecnologías que podrían ser útiles para alcanzar nuestros objetivos. Entre las opciones consideradas, Apache Kafka destaca como la piedra angular de nuestra infraestructura puesto que nuestro proyecto está diseñado alrededor de una arquitectura que se basa en las capacidades y el ecosistema de Apache Kafka. Esto se ha decidido así después de realizar un estudio comparativo entre las diferentes plataformas.

4.1 API GATEWAY

Un API Gateway es un componente crítico en las arquitecturas de microservicios y aplicaciones modernas que actúa como intermediario entre los clientes (como navegadores web y aplicaciones móviles) y los servicios backend que proporcionan la lógica de negocio y los datos, el API Gateway funciona como un punto de entrada único para todas las llamadas de API, (APIKey) ofreciendo una interfaz simplificada a los consumidores de la API mientras encapsula la complejidad de interactuar con varios microservicios.

Éste maneja todas las solicitudes entrantes y las dirige al servicio apropiado dentro del backend implicando resolver las rutas y posiblemente realizar algún nivel de agregación de servicios para responder a la solicitud de manera efectiva simplificando la arquitectura cliente, ya que los clientes necesitan conocer sólo la dirección del API Gateway y no la de los servicios individuales. Para conseguir estos procedimientos el API Gateway debe de asegurar que los usuarios estén autenticado y autorizados proporcionando tokens o claves API.

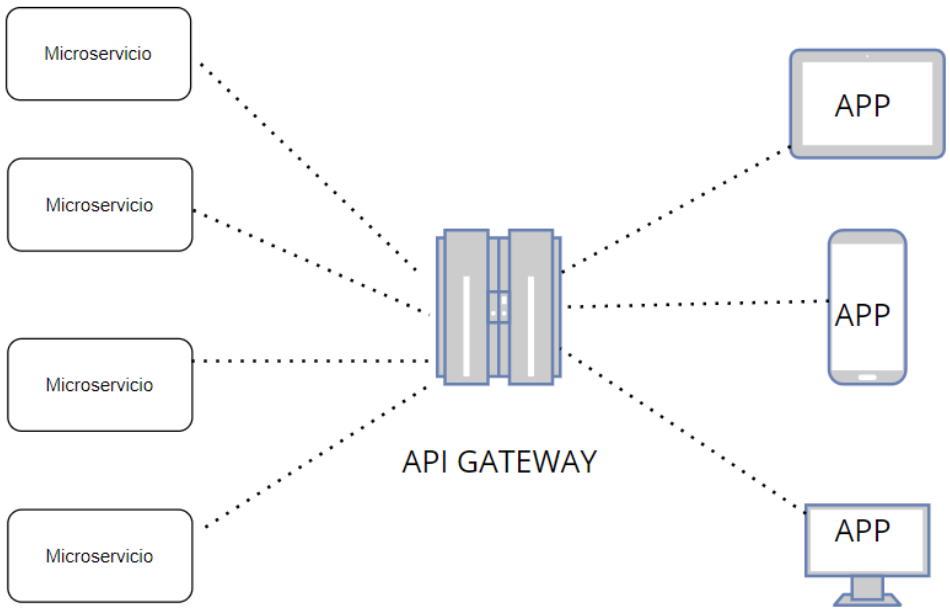


Fig. 6: Esquema de la arquitectura de microservicios

Fuente. Elaboración propia.

4. Estado del Arte

4.1.1 Kong

Kong es un ejemplo de una API Gateway, se trata de una plataforma de administración de API (interfaz de programación de aplicaciones) de códigos abiertos originalmente basada en NGINX con una herramienta de servicio web y proxy que permite ser utilizado para manejar las solicitudes de API en microservicios y arquitecturas orientadas a servicios, el Kong funciona como un intermediario entre los clientes que realizan solicitudes y los servicios que las procesan, gestionando y optimizando las comunicaciones de forma segura y escalable. [1]

En la figura 7 muestra como un cliente puede conectar a una aplicación mediante Kong Gateway. El cliente pide al Kong el servicio que se quiere conectar mediante el PATH, el Kong buscar por los servicios y hace la petición a la aplicación, una vez conseguido se establece la conexión entre la aplicación y el cliente.

El Kong se puede instalar en cualquier infraestructura o sistema, incluso en un Docker. Después de tenerlo instalado es necesario configurar a las necesidades mediante un archivo de configuración (Kong.conf). Para más detalles sobre cómo configurar el Kong puede ver el *Anexo I*.

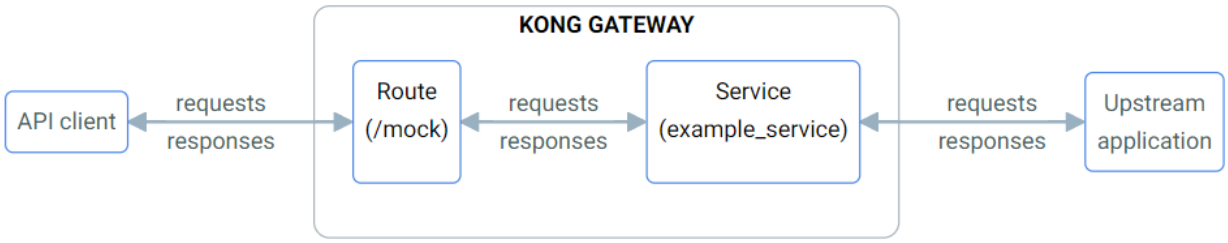


Fig. 7: Arquitectura de un servidor Kong

Fuente. [1].

4.2 SpringBoot

Spring Boot es un marco de trabajo para el desarrollo de aplicaciones en Java que simplifica el desarrollo de aplicaciones.

Pasa usar SpringBoot es necesario tener instalado un JDK (Java Development Kit) un IDE (Entorno de desarrollo integrado) como por ejemplo Eclipse o IntelliJ IDEA y un gestor de dependencias como Maven o Gradle. Para inicializar un proyecto Spring Boot se puede utilizar Spring Initializr, una herramienta online para generar nuevos proyectos Spring Boot.

4.3 Fichero Yaml

Un archivo YAML (YAML Ain't Markup Language) es un formato de serialización de datos legible por humanos que se utiliza comúnmente para la configuración de aplicaciones y la representación de datos

4. Estado del Arte

estructurados muy similar a JSON (JavaScript Object Notation) pero diseñado para ser más legible y fácil de escribir para humanos.

4.4 Arquitecturas.

El diseño de una arquitectura Data-Driven puede variar significativamente dependiendo de los requisitos específicos de la organización, el tipo de datos que se manejan, las capacidades técnicas disponibles y otros factores. Aquí mostramos los diseños más comunes.

- Arquitectura centralizada
  - Todos los datos son recopilados, almacenados y procesados por un único sistema centralizado.
  - Los datos pueden ser almacenados en una base de datos central como un Data Warehouse.
  - Simplifica la gestión y el almacenamiento de la arquitectura, facilita la integración de los datos y el análisis, proporciona una visión única y coherente de los datos de la organización.
  - Pero puede ser difícil de escalar para manejar grandes volúmenes de datos, puede convertirse en un cuello de botella para el rendimiento y la disponibilidad. No es adecuado para entornos con necesidades de datos distribuidos.

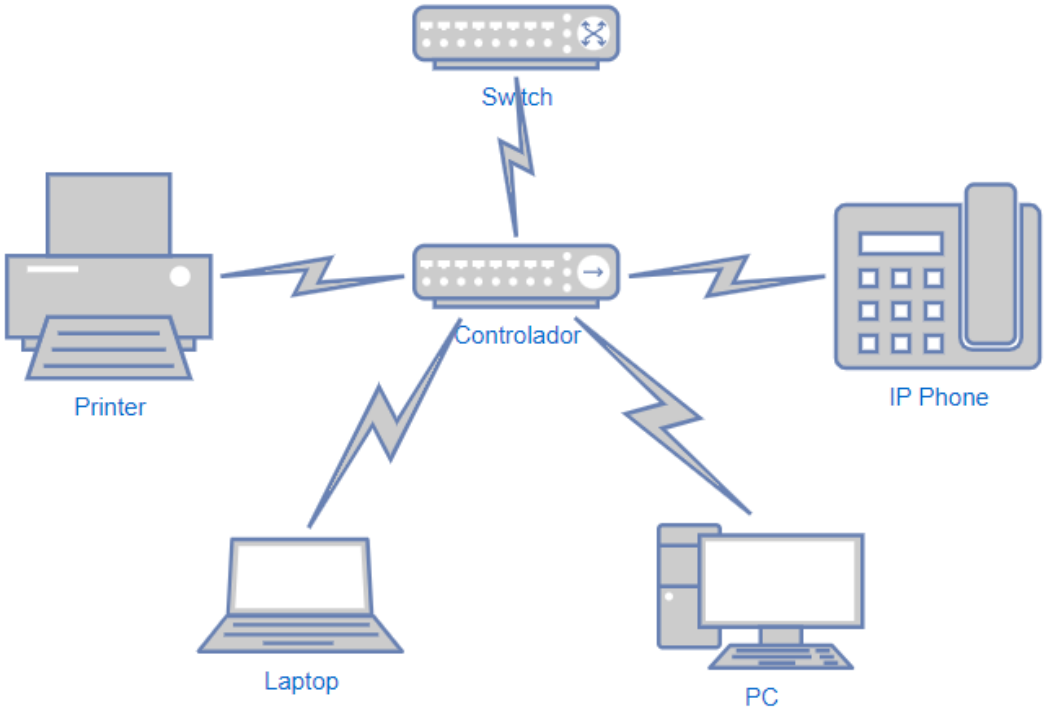


Fig. 8: Esquema de la arquitectura centralizada.

Fuente. Elaboración propia.

4. Estado del Arte

- Arquitectura Distribuida
  - En esta arquitectura, los datos se distribuyen y procesan en múltiples sistemas o nodos distribuidos.
  - Utiliza tecnologías como Hadoop, Spark y sistemas de bases de datos distribuidas.
  - Puede manejar grandes volúmenes de datos y escalar horizontalmente según sea necesario con la capacidad de manejar grandes volúmenes de datos poca tolerancia a fallos y alta disponibilidad.
  - Puede ser más complejo de diseñar, implementar y mantener ya que requiere habilidades técnicas avanzadas para la gestión y administración junto a mayor costo de infraestructura y mantenimiento.

Fig. 9: Esquema de la arquitectura distribuida.

Fuente. [2].

- Arquitectura de Microservicios
  - Divide la aplicación en varios servicios independientes, cada uno con su propia base de datos y lógica de negocio.
  - Los datos se distribuyen entre los diferentes servicios y se comunican a través de APIs.
  - Permite la flexibilidad y la escalabilidad al desarrollar, implementar y escalar servicios de forma independiente.

4. Estado del Arte

- Flexibilidad y escalabilidad al desarrollar y desplegar servicios. Facilita la integración con sistemas existentes y la adopción de nuevas tecnologías. Permite la colaboración y la autonomía de los equipos de desarrollo.
- Existe una complejidad añadida en la gestión de múltiples servicios y comunicación entre ellos que requiere una sólida estrategia de gestión de datos para mantener la coherencia y la integridad de los datos como resultado, puede aumentar la complejidad operativa y de mantenimiento.

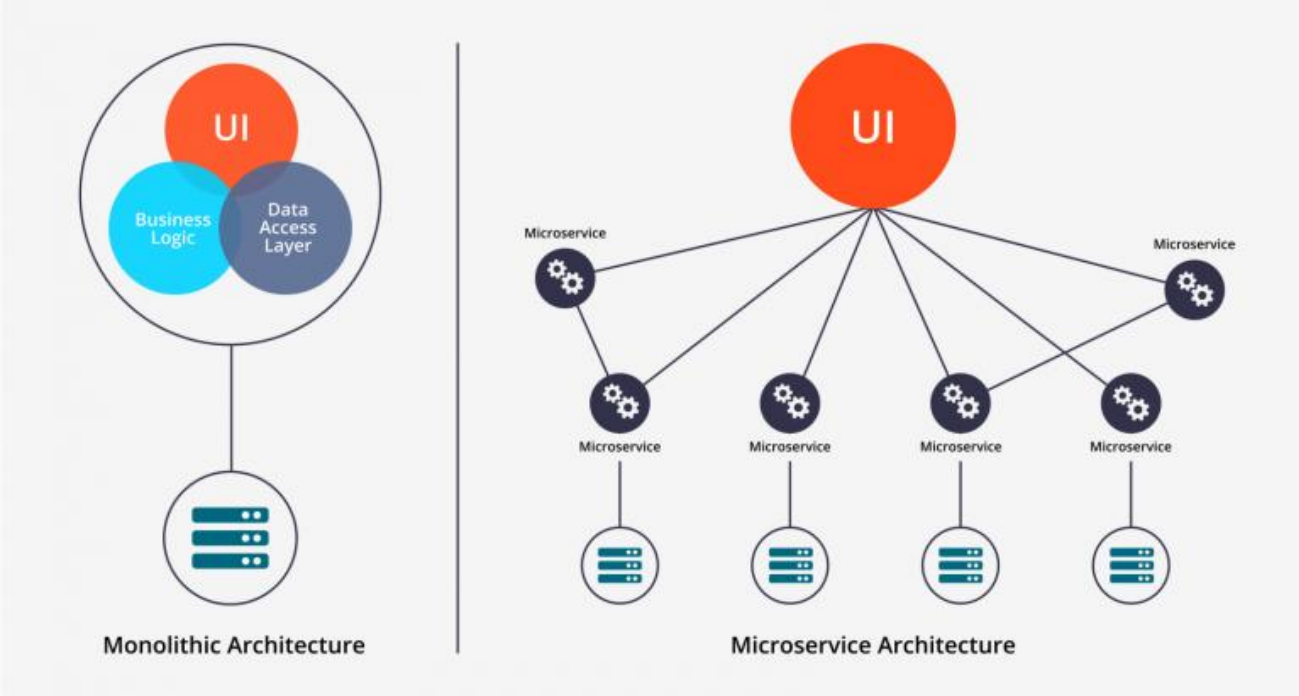


Fig. 10: Esquema comparativa entre la arquitectura de microservicios y la arquitectura monolítica.

Fuente. [3].

- Arquitectura de Event-Driven
  - Los sistemas reaccionan a eventos en tiempo real y procesan los datos en función de estos eventos. Utiliza tecnologías como Apache Kafka para la gestión de eventos y el procesamiento de streams de datos. Facilita la construcción de sistemas.
  - Procesamiento de datos en tiempo real y toma de decisiones ágil. Escalabilidad para manejar flujos de datos de alta velocidad. Facilita la detección de patrones, tendencias y anomalías en tiempo real.
  - Requiere una infraestructura robusta y escalable para manejar eventos de alta velocidad. Puede ser más complejo de diseñar y desarrollar en comparación con enfoques por lote. Necesita una cuidadosa planificación y diseño para garantizar la coherencia y la integridad de los datos.
  - Bases datos dedicadas a la gestión de eventos.

4. Estado del Arte

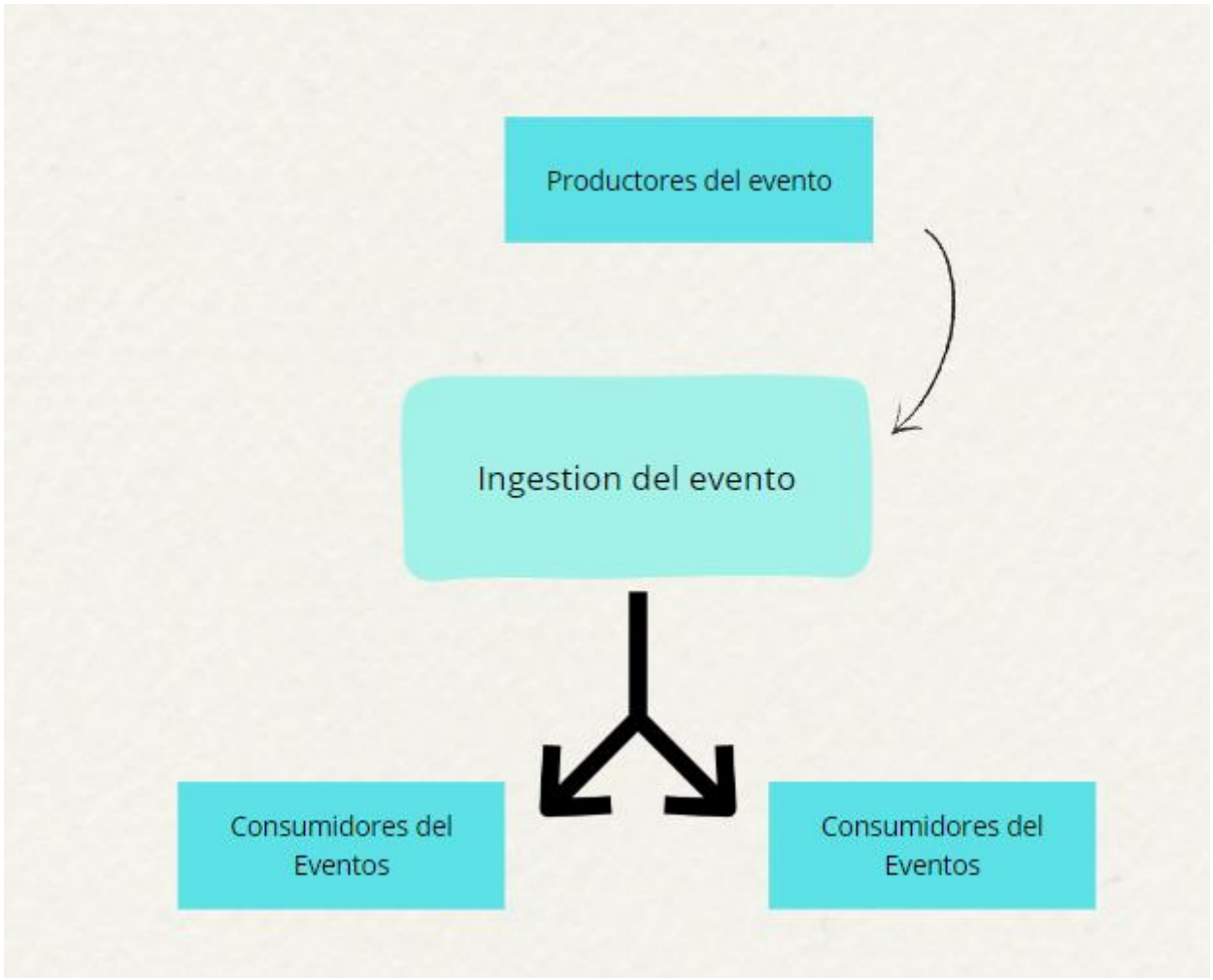


Fig. 11: Esquema de la arquitectura Event-Driven.

Fuente. Elaboración propia.

El enfoque propuesto en este proyecto se caracteriza por ser una arquitectura híbrida. Esta arquitectura combina la arquitectura centralizada junto a la arquitectura de microservicios para el mejor control y monitorización de los datos, gracias a la arquitectura de microservicios podemos descomponer una arquitectura enorme difícil de controlar en partes más minimizadas y mayor control y depuración de errores y por último, integración con la arquitectura Event-Driven para el manejo de la mensajería de datos y aprovecha múltiples enfoques o paradigmas para abordar eficazmente de las arquitecturas mencionadas anteriormente y resuelve los desafíos específicos que se presentan en el contexto del proyecto.

4.5 Data WareHouse

Un Data Warehouse es un almacén de datos con sistema de almacenamiento, recuperamiento y análisis de grandes volúmenes de datos provenientes de diversas fuentes dentro de una organización con la principal

4. Estado del Arte

función de centralizar y consolidar grandes cantidades de datos de múltiples orígenes, transformarlos en homogéneo y analizable con el fin de consultarlas y apoyar la toma de decisiones.

Como se ha visto en el apartado anterior, el proyecto tendrá una metodología hibrida por lo tanto la selección del sistema de almacenamiento tendrá una arquitectura centralizada y distribuida, pero para el escenario que vamos a simular usaremos SQL Server por las conectividades y las ventajas que ofrece.

- Bases de Datos Centralizados:
  - **MySQL**: Es un sistema de gestión de bases de datos relacional de código abierto y gratuito. Organización en tablas con filas y columnas. [4]
  - **PostgreSQL**: Es un base de datos relacional de código abierto y gratuito. Organización en tablas con filas y columnas.[5]
  - **Oracle**: Es un base de datos relacional comercial desarrollado por Oracle. Organización en tablas con filas y columnas.[6]
  - **SQL Server**: Sistema de base de datos relacional desarrollado por Microsoft. SQL Server organiza los datos en tablas. [7]

Tabla 2: Comparativas de Base de datos Centralizadas.

Característica	Tipo	Modelo de Datos	Lenguaje de Consulta	Coste	Escalabilidad
MySQL	Relacional	Tablas	SQL	FREE	Vertical y horizontal
PostgreSQL	Relacional	Tablas	SQL	FREE	Vertical y horizontal
Oracle	Relacional	Tablas	SQL	Comercial	Vertical y horizontal
SQL Server	Relacional	Tablas	T-SQL	FREE	Vertical y Horizontal

Fuente. De elaboración propia inspirada en [4][5][6][7].

- Bases de Datos Distribuidas:
  - **MongoDB**: Sistema de base de datos NoSQL de código abierto y gratuito. Organización en documentación JSON.[8]
  - **Cassandra**: Sistema de base de datos NoSQL de código abierto y gratuito. Cassandra almacena los datos en formato clave y valor.[9]
  - **Redis**: Sistema de base de datos NoSQL de código abierto y gratuito. Se almacena los datos en formato clave y valor. [10]
  - **HDFS**: Es un marco de trabajo que utiliza sistema de archivos distribuido diseñado para ejecutarse en hardware estándar que permite almacenar y gestionar grandes volúmenes de datos de manera eficiente y fiable creado por Apache Hadoop. [11]
  - **Hbase**: Sistema de base de datos NoSQL de códigos abierto y gratuito que almacena de forma clave-valor con la propiedad de almacenamiento subyacente utilizando

4. Estado del Arte

HDFS y el optimizado máximo de las operaciones para el acceso rápido de lectura y escritura de datos. [12]

Tabla 3: Comparativas de Base de datos Distribuidos.

Característica	Tipo	Modelo de Datos	Lenguaje de Consulta	Coste	Escalabilidad
MongoDB	Documento	JSON	-	FREE	Horizontal
Cassandra	NoSQL	Clave-Valor	CQL	FREE	Horizontal
Redis	NoSQL	Clave-Valor	-	FREE	En memoria
HDFS	NoSQL	Archivos distribuidos	-	FREE	Horizontal
Hbase	NoSQL	Clave-Valor	Hbase Shell	FREE	Horizontal

Fuente. De elaboración propia inspirada en [8][9][10][11][12].

4.6 Plataformas de Colaciones de Datos

Aunque el proyecto se titula arquitectura de Data-Driven basada en una plataforma de Apache Kafka, la elección es debida a las comparaciones y valoraciones que se han realizado y que se indican en la tabla 4 de abajo.

Se han encontrado siguientes plataformas:

- **Apache Kafka**: Plataforma totalmente de código abierto y escalable para la distribución de los datos en tiempo real y en paralelo. El funcionamiento de Kafka utiliza un modelo de publicación-suscripción, donde los datos son almacenados en Topics que a su vez pueden estar divididos en particiones, con cada una de las particiones ordenadas e inmutables, gracias a esto, esta plataforma es altamente escalable requiriendo alto nivel de almacenamiento. [13]
- **Amazon Kinesis**: Plataforma con servicio de Streaming de datos no gratuito ofrecido por Amazon Web Service. [14]
- **Google Cloud Pub/Sub**: Servicio de mensajería escalable no gratuito para enviar y recibir mensajes entre aplicaciones y servicios. Tiene funcionamiento parecido a correo electrónico. [15]
- **Confluent Cloud**: Plataforma de Streaming de datos gestionado por Apache Kafka.[16]
- **IBM Event Streams**: Plataforma de Streaming totalmente gestionado en Apache Kafka. Servicio manejado en IBM Cloud. [17]
- **Apache Pulsar**: Es una plataforma de Streaming de mensajes todo en uno, es de código abierto, tiene una arquitectura parecida a Apache Kafka. En comparación con Apache Kafka es que tiene menor latencia, puede procesar millones de mensajes con una latencia inferior a 1ms. El funcionamiento del Pulsar es similar al Apache Kafka, también utiliza un modelo de publicación-suscripción a diferencia de que el Pulsar separa el servicio de almacenamiento del servicio de mensajería utilizando un diseño

4. Estado del Arte

de capas que ofrece mayor flexibilidad y eficiencia, es decir Pulsar utiliza un almacenamiento basado en libros (Bookeepers) que no depende del almacenamiento en disco local. [18]

- **IBM MQ:** IBM MQ está diseñado para asegurar de la entrega de los mensajes utilizando colas para manejar los mensajes y que éste sea procesado solo una vez. No está diseñado para Streaming por lo que no se puede procesar datos en tiempo real.[19]

4. Estado del Arte

Tabla 4: Comparativas de Plataformas de colaciones de Datos.

Plataforma	Características	Streaming SI/NO	Uso	Ventaja
Apache Kafka	- Código abierto - Escalable - Alta disponibilidad - Soporta múltiples lenguajes	SI	- Requiere experiencia técnica - Puede ser complejo	- Bajo costo - Flexible - Comunidad activa
Amazon Kinesis	- Servicio gestionado por AWS - Fácil de usar - Escalable automáticamente - Integración con otros servicios AWS	SI	- Mayor costo - Menor flexibilidad - Menor control	- Simplicidad de uso - Alta disponibilidad - Soporte para diferentes tipos de datos
Google Cloud Pub/Sub	- Servicio gestionado por GCP. Alta escalabilidad y rendimiento - Integración con otros servicios GCP	SI	- Mayor costo - Menor flexibilidad - Menor control	- Fácil de usar - Escalable automáticamente - Alta disponibilidad - Soporte para diferentes tipos de datos
Confluent Cloud	-Servicio basado en Apache Kafka -Completamente gestionado -Escalable -Seguridad y confiabilidad empresarial	SI	- Mayor costo - Menor flexibilidad - Menor control	- Facilidad de uso - Alta disponibilidad - Soporte para diferentes tipos de datos
IBM MQ	-Sistema de encolamiento de mensaje -Escalable -Seguridad	NO	-Menor costo -Menor control -Mayor flexibilidad	-Fácil de usar -Alta disponibilidad -Escalabilidad alta
IBM Event Streams	-Servicio totalmente gestionado -Altamente escalable y seguro -Integración con otros servicios de IBM Cloud	SI	- Mayor costo - Menor flexibilidad - Menor control	- Fácil de usar - Alta disponibilidad - Soporte para diferentes tipos de datos
Apache Pulsar	-Código abierto - Escalable -Soporta múltiples lenguajes - Escalabilidad horizontal - Seguridad	SI	- Fácil de usar - Menor Coste	- Bajo costo - Flexible -Comunidad activa - Moderna

Fuente. De elaboración propia inspirada en [13][14][15][16][17][18][19].

Apache Kafka y Apache Pulsar son plataformas más orientados al Streaming de eventos en tiempo real mientras que IBM MQ está más centrado en la mensajería confiable y transaccional.

4. Estado del Arte

Realizada las comparaciones, se decide utilizar Apache Kafka por ser de Open-Source, de tiempo real y más sencillo de usar en comparación con otras plataformas y tecnologías.

4.7 Tecnologías para Apache Kafka

4.7.1 Conectores

Los conectores son aplicaciones de Apache Kafka para conectar aplicaciones terceras con Kafka y mantener una relación entre esa aplicación.[20] Se han encontrado 2 conectores principalmente para conexión entre 2 tecnologías para reparticiones de mensajes.

- **Debezium Connector**: Debezium es un proyecto que ofrece conectores de Kafka para capturar cambios en bases de datos en tiempo real. Debezium proporcionan conectores específicos para diferentes tipos de bases de datos que pueden capturar eventos de cambio en las tablas de bases de datos y enviarlos a Kafka como flujos de eventos. Esto permite que las aplicaciones consuman estos eventos para análisis en tiempo real, auditoría, replicación de datos, entre otros. [21]
- **Apache Kafka Connect con JDBC Connector**: Apache Kafka Connect es un marco de trabajo distribuido para la integración de sistemas externos con Kafka. Puedes utilizar el conector JDBC de Kafka Connect para leer datos de bases de datos y publicarlos en Kafka, así como para consumir mensajes de Kafka y escribirlos en bases de datos. [22][23]

Tabla 5: Comparativas de Conectores.

Característica	Debezium Connector	JDBC Connector
Tipo de conexión	Basado en cambios	Basado en consultas
Captura de datos	Captura cambios en tiempo real	Captura datos a través de consultas periódicas
Latencia	Baja latencia	Latencia variable, dependiendo de la frecuencia de las consultas
Escalabilidad	Altamente escalable	Escalabilidad limitada por la base de datos
Compatibilidad	Compatible con una amplia gama de bases de datos	Compatible con cualquier base de datos que tenga un controlador JDBC
Complejidad	Más complejo de configurar y administrar	Más simple de configurar y administrar
Costo	Puede ser más costoso	Generalmente menos costoso

Fuente. De elaboración propia inspirada en [21][22][23].

Se ha elegido Debezium Connector porque se acerca más a lo que nuestro objetivo se pide ya que tiene la capacidad de capturar los cambios realizado en una tabla de un BBDD, extraerlo y publicarlo en Kafka.

4. Estado del Arte

4.7.2 Librerías.

- **KSQLDB**: Librería de consulta para Apache Kafka.
- **Kafka Streams:** Biblioteca para desarrollar aplicaciones en tiempo real. Permite transformaciones de los datos en formato claro y a petición de usuario.[24]

Tabla 6: Comparativas de librerías para Apache Kafka.

Criterio	KSQLDB	Kafka Streams
Tipo de procesamiento	SQL	Java/Scala
Facilidad de uso	Alta	Media
Escalabilidad	Alta	Alta
Flexibilidad	Media	Alta
Costo	Bajo	Bajo
Dependencias	Apache Kafka	Apache Kafka
Curva de aprendizaje	Baja	Media

Fuente. De elaboración propia inspirada en [24].

Se utilizará KafkaStream para implementación de recepción de datos en tiempo real puesto que hay posibilidad de transformación de los datos utilizando esta tecnología mientras que KSQLDB solo puedes realizar consultas a Kafka y no tienes capacidad de realizar transformaciones de esas mismas.

4. Estado del Arte

4.7.3 Tecnologías para monitorización.

Para el proceso de monitorización de los datos recibido en Streaming se han encontrado siguientes tecnologías:

- **Kafka-UI**: Es una web versátil, rápido y fácil de usar diseñada para monitorización de Apache Kafka. Es una aplicación totalmente gratuita. [25]
- **KAFKAdeck**: Herramienta de monitorización avanzada para Apache Kafka. Ofrece amplia gama de funcionalidades, pero requiere de pago. [26]

Tabla 7: Comparativas de Interfaces de usuario para Apache Kafka.

Criterio	Kafka UI	KAFKAdeck
Tipo de visualización	Gráficos y tablas	Gráficos y tablas
Funcionalidades	Monitoreo de temas, grupos de consumidores, flujos de datos y métricas de Kafka.	Monitoreo de temas, grupos de consumidores, flujos de datos, métricas de Kafka y análisis de topología.
Facilidad de uso	Interfaz intuitiva y fácil de usar.	Interfaz más compleja y requiere algunos conocimientos técnicos.
Escalabilidad	Soporta grandes volúmenes de datos.	Soporta grandes volúmenes de datos.
Costo	Gratis	Gratis y de pago (versión Enterprise con más funcionalidades).
Dependencias	Apache Kafka	Apache Kafka
Curva de aprendizaje	Baja	Media

Fuente. De elaboración propia inspirada en [24] [25].

Se va a utilizar Kafka UI para nuestro escenario ya que es de código abierto.

4. Estado del Arte

4.8 Tecnologías para ejecución de servicios en Linux

- **JAR**: Para el uso de un JAR, requiere tener descargado un JDK hay que tener JAR generado por Spring Boot.
- **Docker**: Es una plataforma open source que puede correr aplicaciones manejando infraestructuras en un ambiente llamado contenedor otorgando seguridad y escalabilidad. [27]
- **Systemctl**: Es una herramienta de línea de comandos para administrar servicios. Requiere un archivo.service para su ejecución. [28]

Tabla 8: Comparativas de Ejecución de Servicios en Linux.

Criterio	Ejecutar JAR directamente	Docker	Systemctl
Simplicidad	Muy simple, solo requiere el JAR generado por Spring Boot.	Requiere conocimientos básicos de Docker y la creación de una imagen.	Requiere conocimientos básicos de Systemd y la creación de un archivo de unidad.
Portabilidad	Baja, no es ideal para diferentes entornos.	Alta, la imagen se puede ejecutar en cualquier plataforma que tenga Docker instalado.	Media, requiere adaptar el archivo de unidad para diferentes sistemas.
Control	Bajo, no ofrece control sobre el proceso ni gestión de errores.	Alto, permite controlar el proceso, reinicios y monitorización.	Alto, permite controlar el proceso, reinicios, monitorización y configuración de dependencias.
Escalabilidad	Baja, no es ideal para escalar horizontalmente.	Alta, se pueden escalar los containers horizontalmente en un cluster.	Media, la escalabilidad depende del sistema de gestión de servicios utilizado junto con Systemctl.
Costo	Bajo, solo requiere el JAR y la máquina virtual donde se ejecuta.	Medio, requiere una máquina virtual o un servicio de Docker para ejecutar la imagen.	Medio, puede requerir una máquina virtual o un servidor dedicado para Systemd.

Fuente. De elaboración propia inspirada en [26][27].

Para nuestro escenario vamos a usar tanto Docker como Systemctl, Docker para creación de imágenes tener nuestro interfaz de usuario y Systemctl para correr los servicios implementados posteriormente para transformaciones de datos.

4. Estado del Arte

4.9 Tecnologías para ejecución de servicios en Windows

- **NSSM:** Tecnología de monitorización de los servicios que ayuda a los programas reiniciar en caso de caída.
- **WinSW:** Una herramienta para administrar servicios en Windows. Contiene interfaz gráfica y permite supervisar el estado de los servicios y el reinicio en caso de caída. [29]
- **SrvMgr:** Una herramienta grafica para administrar servicios de Windows. Herramienta nativa de Windows.

Tabla 9: Comparativas de Tecnologías para la ejecución automática de servicios en Windows.

Criterio	NSSM	WinSW	SrvMgr
Interfaz	Línea de comandos	Gráfica (opcional)	Gráfica
Instalación	Requerida	Requerida	Requerida
Portabilidad	Media (requiere .NET)	Baja (Windows)	Baja (Windows)
Características	Básica (monitoreo, reinicio automático)	Avanzadas (monitoreo, reinicio automático)	Básica
Curva de aprendizaje	Media	Baja	Baja

Fuente. De elaboración propia inspirada en [29].

Para nuestro escenario se ha elegido NSSM por tener conocimiento previo de su uso ya que se estaría más familiarizado.

5. Formación e introducción en Apache Kafka

5. Formación e introducción en Apache Kafka

Apache Kafka es una plataforma de transmisión de datos Open-Source que se utiliza para la construcción de sistemas de mensajería y procesamiento de datos como eventos en tiempo real.[30]

5.1 Arquitectura de Kafka:

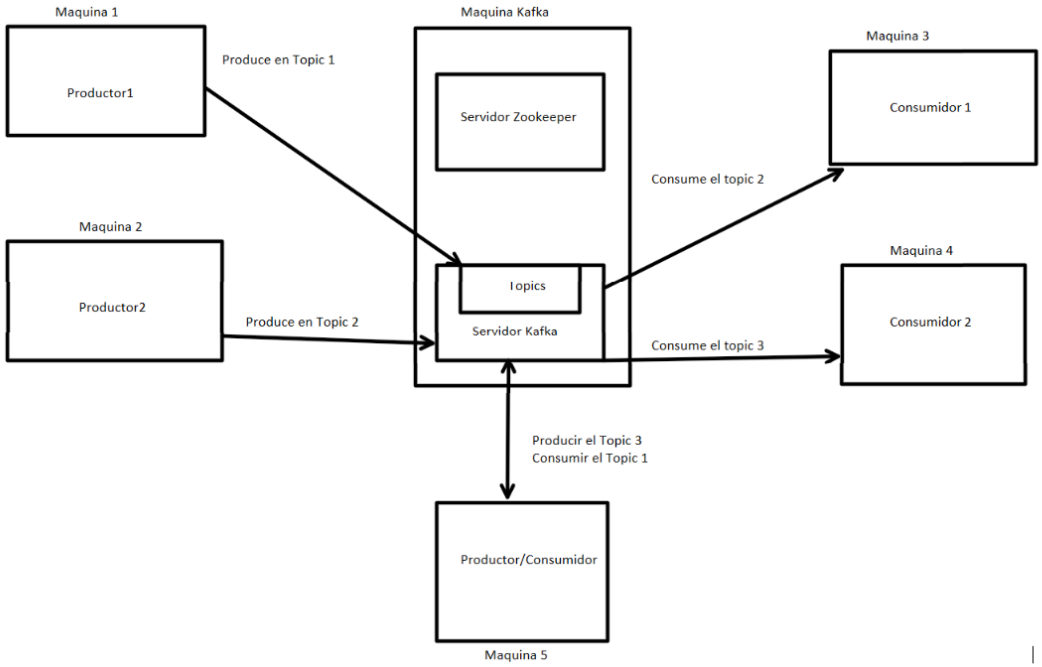


Fig. 12: Esquema de la arquitectura de Kafka.

Fuente. Elaboración propia inspirada en [30].

Para entender el funcionamiento del Apache Kafka primero tenemos que introducir los componentes que componen el Apache Kafka.

- **Servidor Zookeeper**: Software que proporciona un servicio de coordinación de alto rendimiento para distribuir aplicaciones, además es el gestor/manejador de los clústeres, controla la plataforma mediante un almacén de clave-valor. En el zookeeper es el primer elemento en arrancar y es donde se intercambia metadatos con los brokers, productores y consumidores proporcionando una vista sincronizada de la configuración de clúster.
- **Servidor Kafka (Broker):** Instancia que almacenan y sirven de los datos, gestiona las replicaciones y reparticiones. Aquí tienen guardados los Topics y las particiones de estas topics. Crea un topic cuando hay un productor que solicita producir, esta sirve los mensajes producidos cuando un consumidor se suscribe. Los brokers están en comunicación frecuente con el Zookeeper y cada broker tienen su

5. Formación e introducción en Apache Kafka

propia características hardware y su configuración. Cada broker puede contener la partición leader que es la partición que coordina los topics.

- **Clúster:** Agrupaciones de uno o más números de Brokers y el coordinador Zookeeper. Tiene como objetivo guardar conjunto de registros en las categorías llamadas topics donde esos registros se dividen denominando particiones y cada una de las particiones es de propiedad de un sólo broker.
- **Productores**: Tipo de cliente de Kafka que se encargan de procesos que envían mensajes. Solo pueden producir a un topic en concreto.
- **Consumidores**: Tipo de cliente de Kafka que se encargan de procesos de lectura de mensajes. Un consumidor se puede suscribir a varios topics.
- **Topics (Temas):** Canales de comunicación la cual se realiza el streams de datos. Dentro del topic contiene todos los mensajes desde que se produjo el primer mensaje hasta que sea borrado. Los mensajes se guardan en logs y lo almacena de forma que los mensajes puedan trabar asíncronamente y no en tiempo real para prevenir perdida de datos.
- **Mensajes:** Mensaje es la unidad de datos con la que trabaja Apache Kafka, compuesta por Arrays de bytes que se representa con Clave, valor y un timestamp. En Apache Kafka los mensajes son denominados records o messages, estos no tienen una estructura o formato específico para Kafka, pero pueden cambiar con el tipo de esquema usada. Cada mensaje está compuesto por un clave (key) no obligatorio.
- **Particiones:** Las particiones son subdivisiones de un Topic, es decir, una unidad de almacenamiento y gestión de datos dentro del propio Topic. Dentro de cada partición se establece un orden, es decir, que cada partición tiene su propio offset, gracia a las particiones aumenta la eficiencia y rendimiento del sistema y apoya a la distribución de carga.

Una vez explicado los componentes que forman Kafka, podemos entrar a la explicación de la figura 12. La máquina 1 y 2 se posicionan como el productor de mensajería de Kafka publicando sus mensajes a un Topic de la Máquina Kafka, las máquinas 3 y 4 se posicionan como consumidores que se suscriben a la máquina Kafka para recibir mensajes dependiendo del Topic que quieran suscribirse, una máquina puede ser tanto productor como consumidor que es el caso de la máquina 5.

5.2 Funcionamiento

A continuación, se procede a realizar una explicación breve del funcionamiento del Apache Kafka. Varios productores se proceden a emitir mensajes en el mismo Topic como se puede en observar en la *Fig. 13*, el topic guarda los mensajes emitidos por el productor en forma de cola y espera a ser leído por un consumidor, estos datos se guardan solo un tiempo determinado definido por el usuario y una vez pasado el tiempo, ese dato se borra y no es recuperable. Cuando un consumidor se suscribe al Topic y quiere recibir los mensajes de ese Topic, puede indicar desde que orden empieza a recibir los mensajes, eso ayuda a que el consumidor no reciba mensajes duplicados.

5. Formación e introducción en Apache Kafka

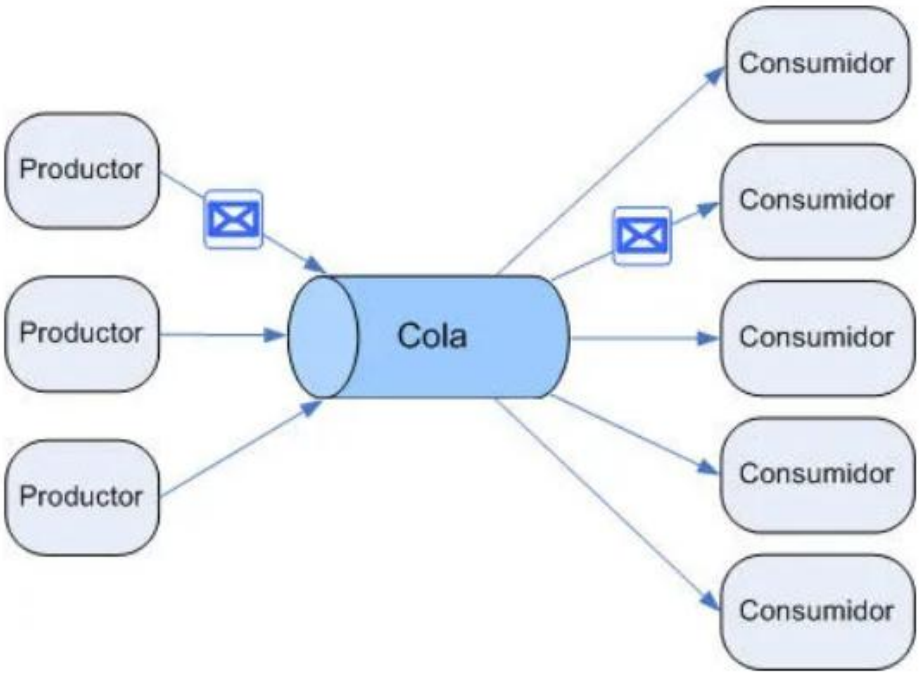


Fig. 13: Esquema de producción y consumición de Kafka [31]

Fuente. [31].

5.2.1 Como reconocer un mensaje

Cada evento producido contiene un "record", indica que se ha producido un evento saltando en la consola del consumidor en caso de estar leyendo el topic. [32][33]

Un record contiene:

- Event Key: Es un componente opcional del registro en Kafka que sirve para identificar de manera única a cada mensaje dentro de las particiones. El cálculo para saber a qué partición pertenece un clave es mediante un Hash y los mensajes que llevan el mismo clave se agrupan a la misma partición.
- Event Value: Es el contenido real del mensaje o el dato que se está transmitiendo en secuencia de bytes.
- Event timestamp: Es una marca de tiempo real asociado a cada mensaje, esto dependiendo de la configuración del sistema puede ser la marca del tiempo del momento en que el productor se ha realizado el envío del mensaje o puede ser la marca del tiempo del momento que el mensaje ha sido aceptado por el broker.

5.2.2 Clientes de Kafka

Los productores son aplicaciones cliente que publican (escriben) eventos en Kafka, y los consumidores son aquellos que se suscriben (leen y procesan) estos eventos. En Kafka, los productores y consumidores están completamente desacoplados y son agnósticos entre sí, lo cual es un elemento de diseño clave para lograr la alta

5. Formación e introducción en Apache Kafka

escalabilidad por la que Kafka es conocido. Por ejemplo, los productores nunca necesitan esperar a los consumidores y esto proporciona varias garantías, como la capacidad de procesar eventos exactamente una vez.

Los eventos están organizados y almacenados de manera duradera en topics. De manera simplificada, un topic es similar a una carpeta en un sistema de archivos, y los eventos son los archivos en esa carpeta. Los topics en Kafka siempre son multi-productor y multi-suscriptor: un Topic puede tener cero, uno o varios productores que escriben eventos en él, así como cero, uno o varios consumidores que se suscriben a estos eventos. Los eventos en un Topic pueden ser leídos tantas veces como sea necesario; a diferencia de los sistemas de mensajería tradicionales, los eventos no se eliminan después de su consumo, en su lugar, defines por cuánto tiempo Kafka debe retener tus eventos a través de una configuración por Topic, después de lo cual los eventos antiguos serán descartados. El rendimiento de Kafka es efectivamente constante con respecto al tamaño de los datos, por lo que almacenar datos durante mucho tiempo es perfectamente aceptable.

Los Topics están particionados, lo que significa que un Topic se distribuye en un número de "contenedores" ubicados en diferentes brokers de Kafka. Esta colocación distribuida de tus datos es muy importante para la escalabilidad porque permite a las aplicaciones cliente leer y escribir los datos desde/hacia muchos brokers al mismo tiempo. Cuando se publica un nuevo evento en un Topic, en realidad se agrega a una de las particiones del Topic y los eventos con la misma clave de evento (por ejemplo, un ID de cliente o vehículo) se escriben en la misma partición, y Kafka garantiza que cualquier consumidor de una partición de un Topic dado siempre leerá los eventos de esa partición exactamente en el mismo orden en que fueron escritos.

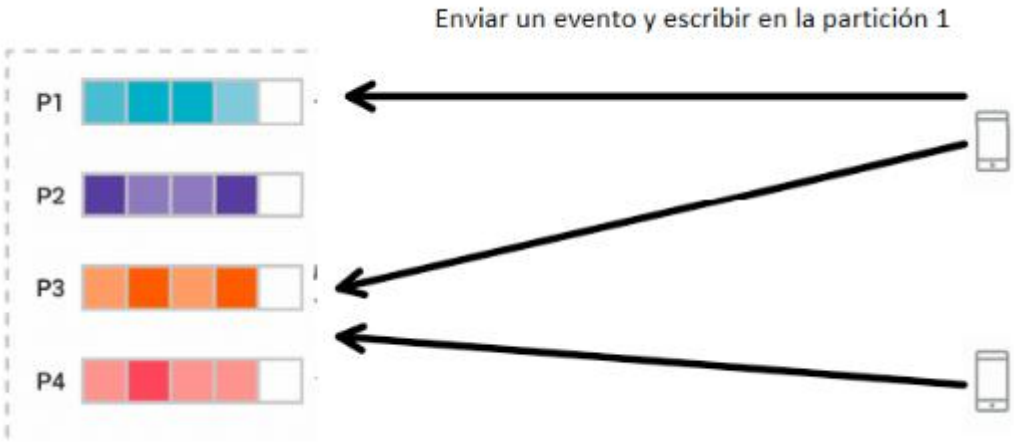


Fig. 14: Esquema de particiones. [33].

Fuente. [33].

En la *Fig. 14* se trata de un ejemplo que tiene cuatro particiones P1-P4. Dos clientes productores diferentes están publicando, independientemente entre sí, nuevos eventos en el Topic escribiendo eventos a través de la red en las particiones del Topic. Los eventos con la misma clave (indicados por su color en la figura) se escriben en la misma partición y se puede observar que ambos productores pueden escribir en la misma partición si es necesario.

5. Formación e introducción en Apache Kafka

5.3 Ejecución y configuración de Kafka

Todos los procesos de producción y de consumición proporcionan gran capacidad de distribución, escalabilidad, elasticidad, tolerancia y seguridad. Además, Kafka se puede desplegar en máquinas virtuales y en contenedores, así como desplegarlo en la nube. En una misma máquina se podría desplegar tantas consolas de Broker como queramos, pero la tolerancia de fallo sería mucho más alta, debido a que, si la máquina se cae, todos los servidores Brokers y sus respectivos Zookeeper dejarán de estar en funcionamiento y hará perdidas a la empresa. Para ello se debe desplegar en 2 consolas de 2 máquinas diferentes donde una es para el servidor Zookeeper y otra es para el servidor Broker para proporcional mayor escalabilidad, seguridad y tolerancia a error gracias a que la conexión del Zookeeper y Broker se establece por un fichero de configuración en la cual se determina que Servidor suscribir y que puertos escuchar y como es el acceso a estos servidores. El despliegue de los servidores Brokers y de los Zookeeper se consiguen mediante unos archivos de configuración que vienen por defecto cuando instalas el Apache Kafka, estas configuraciones se debe modificar dependiendo de la necesidad de cada uno para saber detalles avanzados de como configurar y ejecutar el Apache Kafka se pueden visitar el *Anexo II*.

5.4 Conexión de Kafka con terceras máquinas

Conseguir la conexión entre Kafka y otros sistemas requiere utilizar conectores llamados Kafka Connect. Kafka Connect es una herramienta para transmitir datos de manera escalable y confiable entre Apache Kafka y otros sistemas. Facilita definir rápidamente conectores que mueven grandes colecciones de datos dentro y fuera de Kafka. Kafka Connect puede ingresar bases de datos enteras o recolectar métricas de todos sus servidores de aplicaciones en temas de Kafka, lo que hace que los datos estén disponibles para el procesamiento de transmisión con baja latencia. Un trabajo de exportación puede enviar datos desde temas de Kafka a sistemas de almacenamiento secundario y sistemas de consulta, o a sistemas de lotes para análisis fuera de línea.

6. Análisis de deficiencias y beneficios de la plataforma Apache Kafka y la Arquitectura Data-Driven y la propuesta de posibles mejoras

6. Análisis de deficiencias y beneficios de la plataforma Apache Kafka y la Arquitectura Data-Driven y la propuesta de posibles mejoras

6.1 Beneficios de utilizar una arquitectura Data-Driven

- Permite a las organizaciones tomar decisiones basadas en datos en lugar de suposiciones.
- Facilita la identificación de áreas de mejora y optimización en los procesos.
- Ayuda a identificar nuevas oportunidades de negocio, tendencias del mercado y necesidades del cliente.
- Permite a la personalización y segmentación de productos, servicios y experiencias basadas en el comportamiento y las preferencias del cliente.
- Facilita la compresión del comportamiento y las preferencias del cliente para ofrecer experiencias personalizadas y satisfactorias.
- **Personalización de la experiencia del cliente:**
  - Los datos pueden usarse para comprender mejor el comportamiento individual de los clientes.
  - Las empresas pueden usar esta información para ofrecer experiencias personalizadas a cada cliente. Esto puede aumentar la satisfacción del cliente y la fidelidad a la marca.
- Los procesos se automatizan en base a los datos, liberando tiempo y recursos para tareas más estratégicas.
- Se reduce el tiempo y el coste de la toma de decisiones.
- Se pueden realizar pruebas y experimentos de forma más rápida y eficiente.
- Permite a las empresas tomar decisiones más transparentes y responsables.
- Toma de decisiones más inteligentes:
  - Los datos proporcionan información valiosa sobre el mercado, los clientes, la competencia y las operaciones internas.
  - Esta información permite a las analistas tomar decisiones más informadas y estratégicas.
  - Las empresas pueden reducir el riesgo y aumentar la probabilidad de éxito al basar sus decisiones en datos concretos.

6.2 Beneficios de utilizar Apache Kafka

- **Simplificación:** Las herramientas de gestión simplifican la configuración, el mantenimiento y el monitoreo de Kafka.
- **Escalabilidad:** La arquitectura de microservicios facilita la escalabilidad horizontal y vertical.
- **Seguridad:** Las medidas de seguridad protegen la plataforma de accesos no autorizados y fugas de datos.
- **Visibilidad:** Las herramientas de monitoreo permiten identificar y solucionar problemas de forma rápida y eficiente.
- **Eficiencia:** Procesamiento de datos y detección de anomalías en tiempo real otorgando velocidad y menor latencia.

6. Análisis de deficiencias y beneficios de la plataforma Apache Kafka y la Arquitectura Data-Driven y la propuesta de posibles mejoras

- **Fluidez:** Facilita la comunicación y el intercambio de datos entre diferentes sistemas.
- **Agilidad en el Desarrollo de Aplicaciones:** Streaming de datos ayuda a reducir el tiempo de lanzamiento al mercado de nuevos productos y servicios.

Fig. 15: Esquema de Ventajas de la arquitectura Data-Driven.

Fuente. Elaboración propia.

6.3 Deficiencias de Apache Kafka

- **Complejidad:** La configuración y el mantenimiento de una arquitectura Data-Driven con Apache Kafka puede ser complejo, especialmente para equipos pequeños o sin experiencia en la plataforma.
- **Gestión de Esquemas de Datos:** Esta plataforma por sí no gestiona esquemas de datos, por lo que puede llevar a problemas de incompatibilidad entre los datos producidos y consumidos, especialmente en sistemas complejos y en evolución.
- **Monitoreo:** El monitoreo es esencial para garantizar el correcto funcionamiento de la arquitectura Data-Driven. Kafka ofrece herramientas de monitoreo básicas, pero es posible que se necesiten herramientas adicionales para obtener una visibilidad completa del sistema.

6. Análisis de deficiencias y beneficios de la plataforma Apache Kafka y la Arquitectura Data-Driven y la propuesta de posibles mejoras

- **Latencia en el Procesamiento de Streaming:** Dependiendo de las cargas, puede haber ligera latencia. Pero en cargas muy grandes puede tardas horas en volver a Streaming.
- **Compatibilidad con diferentes tipos de datos:** En un principio Kafka envía datos de forma estructurada en Json. No es posible transmitir imágenes ni videos ni audios.
- **Dificultad en la depuración de errores:** Requieren herramientas de depuración específicas para identificar y solucionar problemas de forma eficiente.

6.4 Propuestas de Mejora a las Deficiencias de utilizar Apache Kafka

- **Adoptar una arquitectura de microservicios:** Para la complejidad de Apache Kafka se puede descomponer la arquitectura en microservicios independientes facilita la escalabilidad y la gestión de errores.
- **Utilizar herramientas de gestión:** Existen herramientas de gestión como Confluent Control Center que simplifican la configuración, el mantenimiento y el monitoreo de Kafka para la gestión de esquemas de datos y que puedan solucionar incompatibilidades entres los datos producidos y consumidos. Los programas mencionados son de pago.
- **Utilizar herramientas de monitoreo:** Implementar herramientas de monitoreo como Prometheus y Grafana para obtener una visibilidad completa del sistema. Pero gracias a los desarrolladores de GITHUB han desarrollado una herramienta de monitoreo open-source para visualizar la mensajería producida en Kafka y monitorizar estas.
- **Crear Arquitectura Robusta:** Para minimizar carga de datos y que produzca alta latencia, intentar hacer que nuestra arquitectura sea robusta, minimizar errores para que el Streaming no se caiga y que la cola de carga siempre este en control.
- **Utilizar herramientas de depuración:** Para la depuración de Kafka se han implementado herramientas como Kafka JMX y Kafka Profiler.

6.5 Desafíos de una Arquitectura Data-Driven

- La gestión de grandes volúmenes de datos de diferentes fuentes puede ser compleja y desafiante.
- La seguridad y la privacidad de los datos son preocupaciones importantes en las arquitecturas Data-Driven, especialmente cuando se trata de datos sensibles.
- Garantizar la calidad de los datos es crucial para obtener resultados precisos y confiables del análisis.
- Adoptar una arquitectura Data-Driven puede requerir cambios culturales dentro de la organización, donde los datos se convierten en el fundamento de las tomas de decisiones.

6.6 Propuestas de Mejora a los desafíos de una Arquitectura Data-Driven

- Una posible solución para la gestión de grandes volúmenes de datos seria utilizar plataformas diseñadas para manejo de grandes volúmenes de datos de manera eficiente y descomponer aplicaciones monolíticas en microservicios para una mejor gestión y escalabilidad de los datos.

6. Análisis de deficiencias y beneficios de la plataforma Apache Kafka y la Arquitectura Data-Driven y la propuesta de posibles mejoras

- Para la seguridad y la privacidad se propone encriptar los datos utilizando claves simétricas, establecer un control de acceso con autenticación multifactorial para que los datos solo sean accedidos por el personal y el enmascaramiento de los datos.
- Para conseguir que los datos recibidos al final de ciclo sean confiable y preciso se propone el monitoreo continuo en las diferentes fases de la arquitectura tanto en el Streaming en Kafka como al final en la publicación de los datos.
- Para el tema del cambio cultural en la organización se propone ofrecer programas de capacitación y educación para empleados en el uso y análisis de los datos.

7. Caso de uso

7. Caso de uso

En este proyecto se va a realizar un caso de uso para la arquitectura definida: la ingesta y el procesamiento de los datos en tiempo real y monitorización, para posteriormente integrarlos en las aplicaciones y poder realizar análisis a partir de los datos.

Nuestro escenario tendrá una Máquina Virtual que le denominaremos Máquina A, y una segunda Máquina Virtual que le denominaremos Máquina B.

La Máquina A contiene un sistema operativos Windows 10. Lleva instalado base de datos SQL Server y los ficheros necesarios para el funcionamiento correcto del Apache Kafka, además, para la automatización de los arranques de los servidores zookeeper, brokers y Connector se ha instalado NSSM como se ha definido en el Estado de Arte anteriormente.

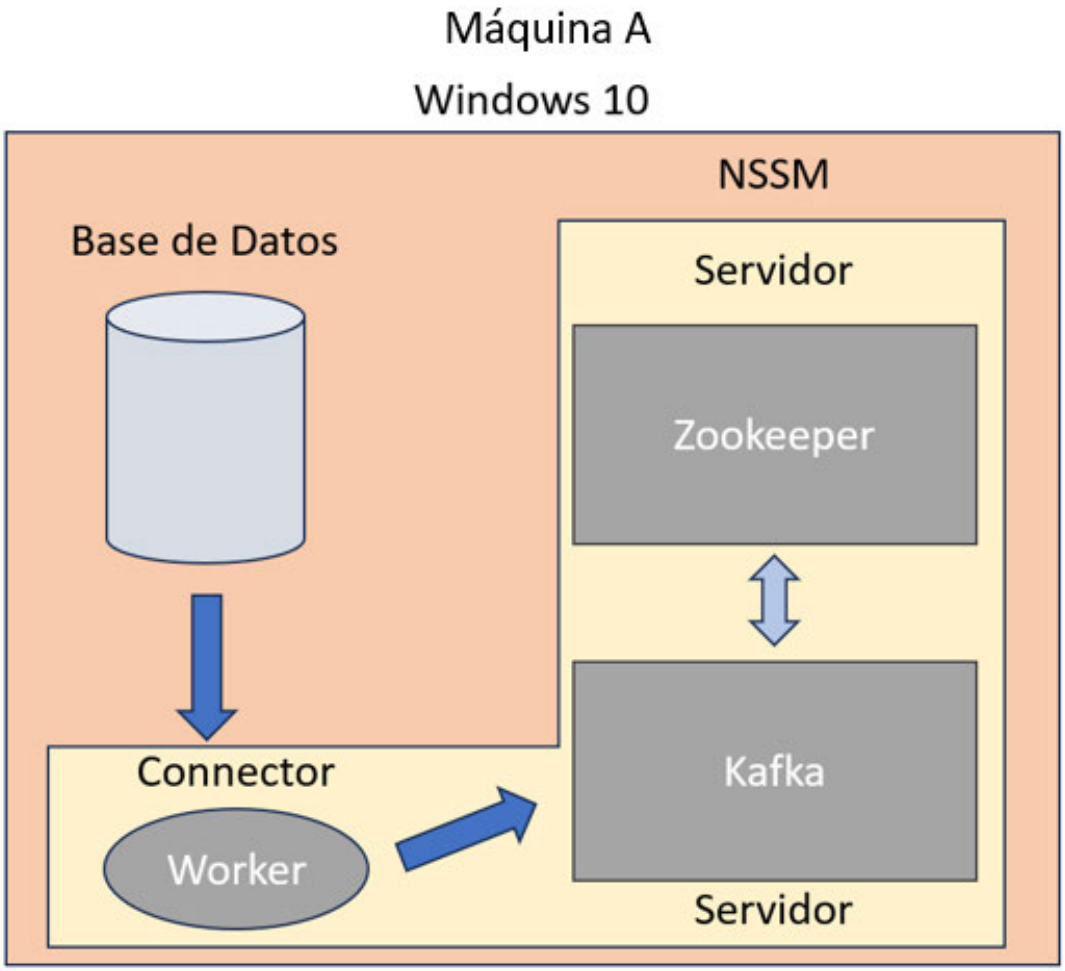


Fig. 16: Máquina A.

Fuente. Elaboración propia.

7. Caso de uso

La Máquina B contiene un sistema operativo Linux. Se ha definido en esta máquina los servicios necesarios para la automatización de los arranques de los servidores necesarios para Kafka mediante el uso del *Systemctl*. Los servicios de implementación propia para las transformaciones también se han automatizado utilizando *Systemctl*. En esta máquina también tiene instalada Docker para contener una imagen de UI de Kafka saliendo por el puerto 8080/tcp.

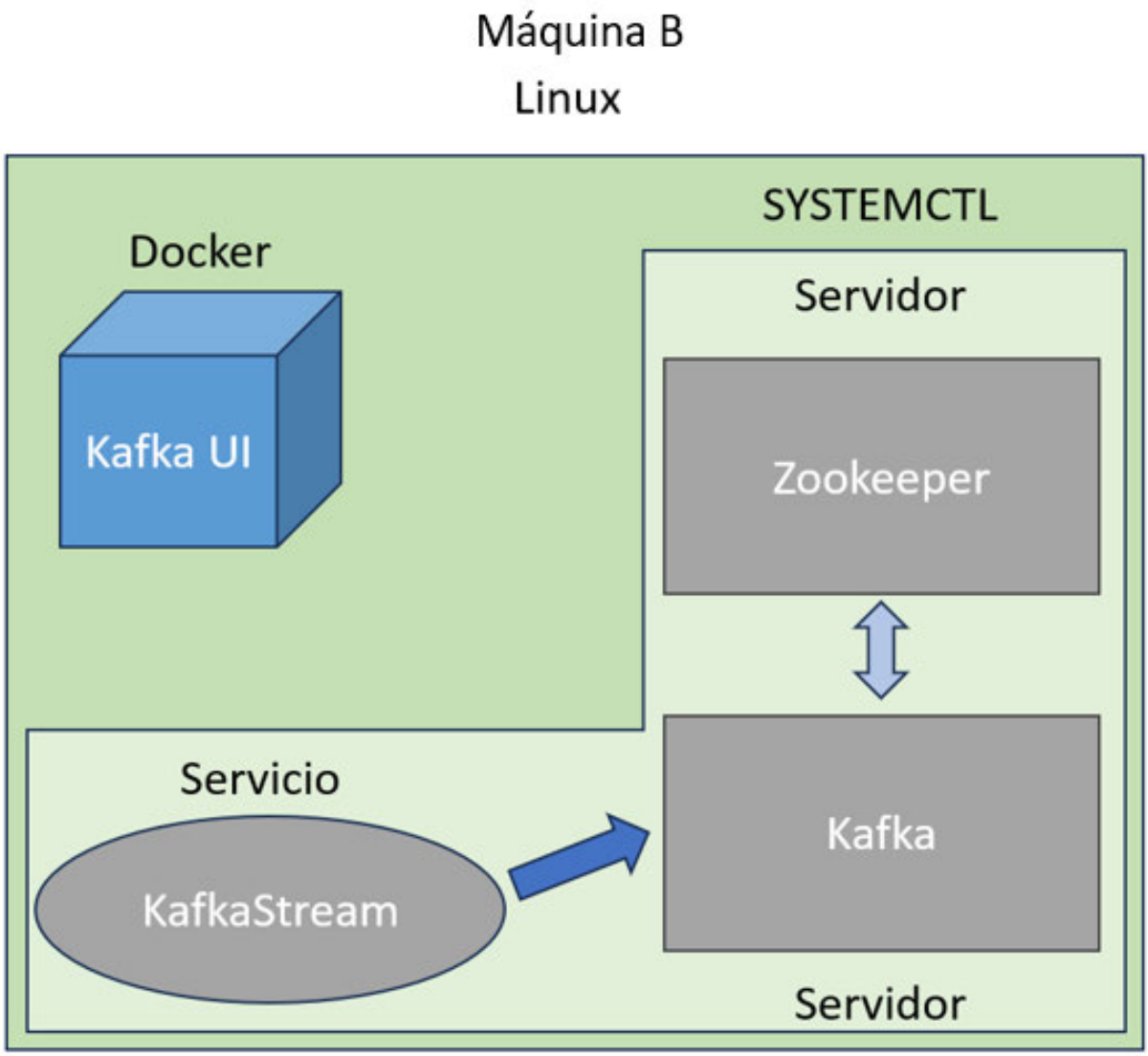


Fig. 17: Máquina B.

Fuente. Elaboración propia.

7. Caso de uso

7.1 Despliegue de la arquitectura

El apartado sobre el despliegue de la arquitectura para escenario tiene como objetivo describir en detalle cómo se llevará a cabo la instalación, configuración y puesta en marcha de la solución propuesta que es fundamental para demostrar la viabilidad y la implementación practica de la arquitectura diseñada.

7.1.1 Arquitectura del escenario propuesto

Las instalaciones y las configuraciones de las tecnologías definidas que se describen a continuación:

Despliegue de la Máquina A

- Instalar la última versión de Kafka: Kafka_2.12-3.61.
- Instalar la última versión Conector Debezium SQL Server Connector: Debezium-Connector-2.5.
- Crear un servicio para Zookeeper y levantarlo.
- Crear un servicio para Kafka y levantarlo.
- Crear un servicio para Worker con el conector y levantarlo.
- Despliegue de Base de Datos
    - Instalar en un servidor remoto SQL Server Management Studio 19.
    - Instalar una instancia de SQL Server DEVELOPER 2022.
    - Crear una base de Datos.
    - Crear una tabla con las columnas necesarias.
    - Activar tanto para base de dato como para las tablas la opción de CDC.

Despliegue de la Máquina B

- Instalar la última versión de Kafka: Kafka_2.12-3.61.
- Crear un servicio para Zookeeper y levantarlo.
- Crear un servicio para Kafka y levantarlo.
- Implementar un servicio mediante uso de librería de Kafka para realizar transformaciones de los datos leídos. Implementación del servicio mediante SpringBoot.
- Crear un servicio para KStream y levantarlo.

Como se ha definido en el apartado anterior, el escenario de este proyecto contiene una base de datos SQL Server que almacena la primera información fuente, esta información es extraída hacia el Servidor Kafka mediante el uso de Kafka Connect. Para nuestro escenario, el Kafka Connect que se ha utilizado es el Debezium Connector for SQL Server. Debezium como se ha mencionado en el apartado del *Estado del Arte* es una plataforma de cambio de datos que facilita la CDC (Change Data Capture), captura de eventos de cambio de datos, del SQL Server y los entrega en tiempo real al Apache Kafka. En la *fig. 17* Kafka Connect es denominado Worker que extrae información del SQL Server y publica en un topic del servidor de la Máquina A. En la Máquina B reside un microservicio denominado KafkaStream que consume del topic que se ha publicado el Worker y realiza transformaciones de aquellos datos para posteriormente publicarlo en un topic de

7. Caso de uso

la máquina B. Un detalle de la configuración importante a tener en cuenta en la Máquina A es que dentro del SQL Server debe estar activado el SQL Server Agent y otorgar permiso de CDC a las bases de datos necesarios y sus tablas para ver sus configuraciones en *Anexo IV*.

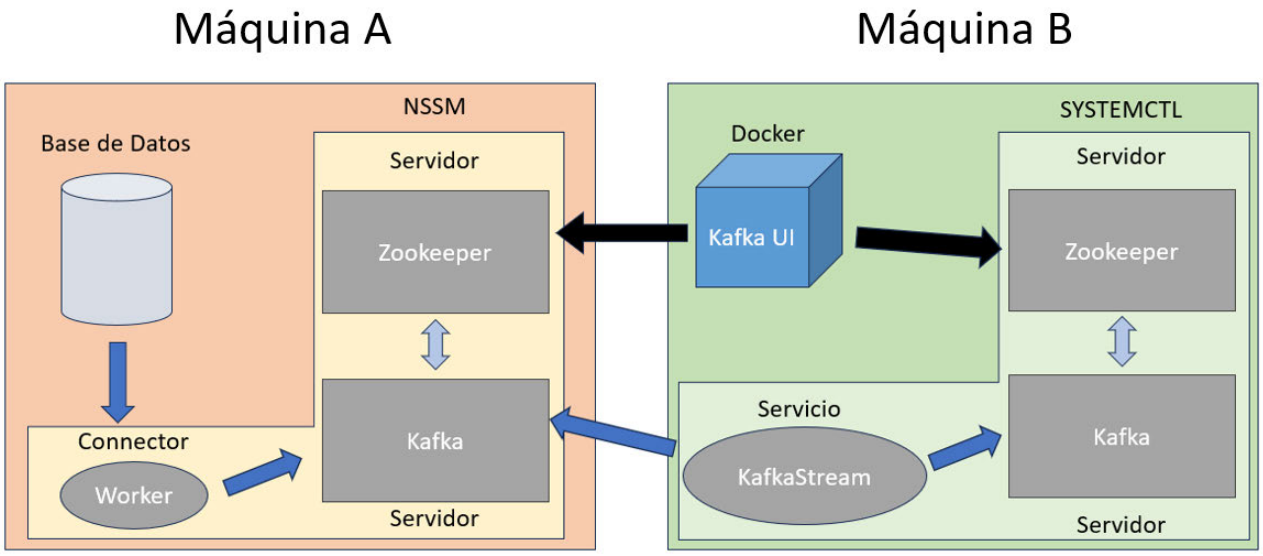


Fig. 18: Arquitectura del escenario propuesto.

Fuente. Elaboración propia.

- Worker: Es un proceso de ejecución que se encarga de administrar y ejecutar los conectores Kafka, transfiriendo datos entre Kafka y los sistemas externos. En este escenario está recopilando los cambios que se ha realizado en una tabla de un base de datos y publicando en un Topic del servidor Kafka Local.
- KafkaStream: Es un microservicio en ejecución que conecta dos servidores Kafka y mediante transformaciones convierte los datos enormes en datos más reducidos desechando datos innecesarios.

Una vez que el servidor Kafka de la Máquina A tenga creado el Topic de los mensajes que va a publicar el Worker, el microservicio de la Máquina B se conectará al servidor Kafka de la Máquina A suministrando el Topic creado por el Worker y lo publica en el servidor Kafka de la Máquina B.

7.1.2 Prueba de fallos

En este apartado se va a definir el fallo y error de nuestro caso de uso, comentando que sucede en casos de caídas de internet, caídas de una máquina cuando está conectado a otra máquina, etc.

- Cuando el consumidor se cae por cualquier motivo y no tiene establecido la propiedad de “auto_reset”, y vuelve a conectarse a la máquina recibe datos duplicados. La solución es establecer la propiedad definido anteriormente o establecer un commit sincronizado.
- Cuando un productor está realizando producción de mensajes y el consumidor ya está conectado a ese topic que produce el productor y el servidor Zookeeper se ha caído, resulta que en el servidor broker

7. Caso de uso

salta el error mientras que el consumidor puede recibir mensajes sin problema, pero una vez que se sale del Topic ya no es capaz de volver a conectar.

- Cuando el consumidor está consumiendo mensajes de un topic y el broker se cae, el consumidor directamente deja de consumir y saltan errores hasta que vuelvan a establecer conexión con el broker.

Se presentarán figuras de las pruebas comentadas anteriormente en el *Anexo IV*

7.1.3 Automatización de la Arquitectura

Teniendo la arquitectura definida, en este apartado se debe llegar a conseguir la automatización de todo el proceso, desde que llega información a la base de datos, luego esa información es extraída por el conector hasta que llega al servidor Kafka de la Máquina B después de ser transformada.

Para la automatización se ha creado un ejecutable basado en jar de java en la Máquina A para que inserte informaciones a la base de datos SQL Server periódicamente.

Para prevenir caídas tanto de Worker, como servidor zookeeper y Kafka en la Máquina A, se ha utilizado NSSM como había definido en el apartado del *Estado del Arte* instalando servicios del Worker, zookeeper y Kafka dentro del sistema. Gracias a esto cuando haya caídas de cualquier tipo de problemas, los servidores y el Worker pueden levantarse de nuevo automáticamente proporcionando siempre entrega de eventos al servidor Kafka.

De la misma manera, en la Máquina B para prevenir caídas, se ha utilizado *Systemctl* como había definido en el apartado del *Estado del Arte* estableciendo tanto los microservicios como los servidores como servicios de monitorizado por el *Systemctl* para que en caso de tener problemáticas pueda ser reiniciado automáticamente y que el microservicio KafkaStream este siempre realizando transformaciones de los datos y guardándolos en el servidor Kafka de la Máquina B. Para tener más detalle del microservicio KafkaStream visualice *Anexo V*.

7.1.4 Monitorización de la Arquitectura

Conseguido esta parte de automatización, requiere una monitorización de todo el proceso para estar seguro de que el funcionamiento esta todo correcto, para ello se va a instalar dentro de un Docker una imagen de Kafka UI utilizando un fichero Yaml que define qué es lo que tiene que recoger. La información de qué topics tienen guardado cada uno de los brokers tanto de la Máquina A como la información de la Máquina B lo muestra en una página web como se puede apreciar en la *Fig. 18*. En nuestro caso la página está controlada por la Máquina B. Para mayor detalle de como levantar el Docker e instalación de la imagen visualice *Anexo VI*.[25]

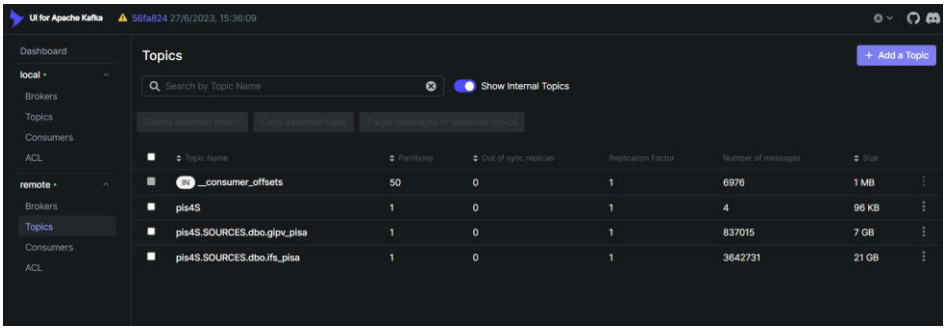


Fig. 19: Captura de pantalla del Kafka UI.

Fuente. Elaboración propia.

7. Caso de uso

7.1.5 Beneficios que se podría obtener a través de este Caso de Uso

Otro beneficio que puede traer esta Arquitectura es en la parte de monitorización que visualizando la UI de Kafka en el apartado de estadísticas podemos realizar un análisis del consumo y utilización de discos de cada topic por cada cliente.

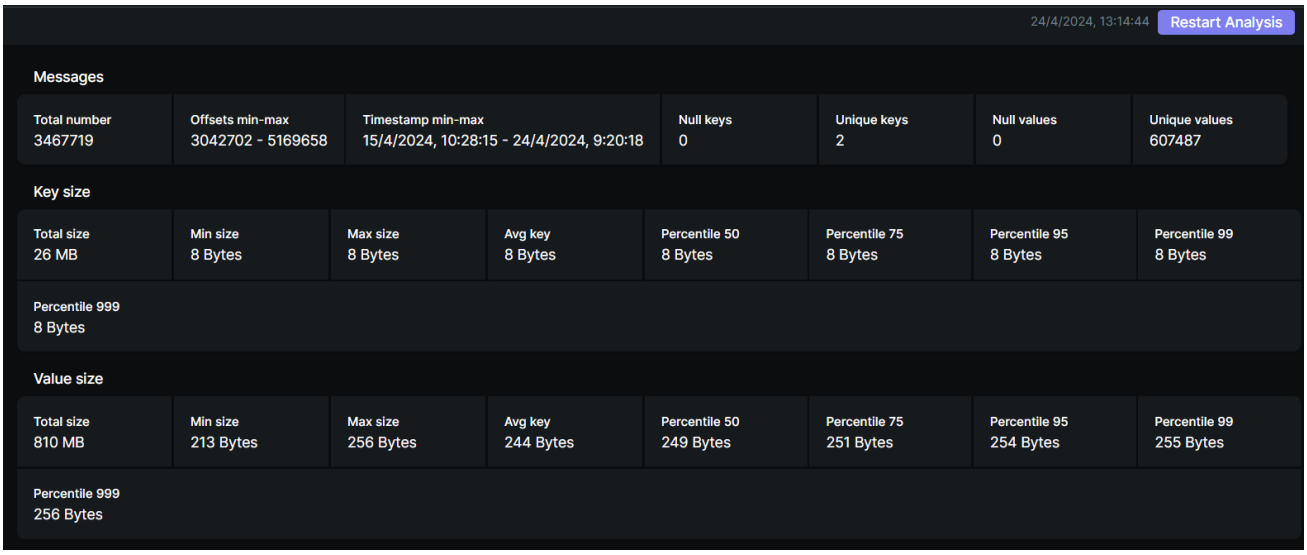


Fig. 20: Estadísticas de un Topic.

Fuente. Elaboración propia.

Como ejemplo, en la figura anterior (*Fig. 19*) se ha calculado unas estadísticas de este topic mostrando los mensajes totales que se han pasado por este Topic y el tiempo (Timestamp por la que se ha pasado), también se puede visualizar el tamaño de los valores. Este Topic tiene 3 particiones, como se aprecia en la *Fig. 20*.

7. Caso de uso

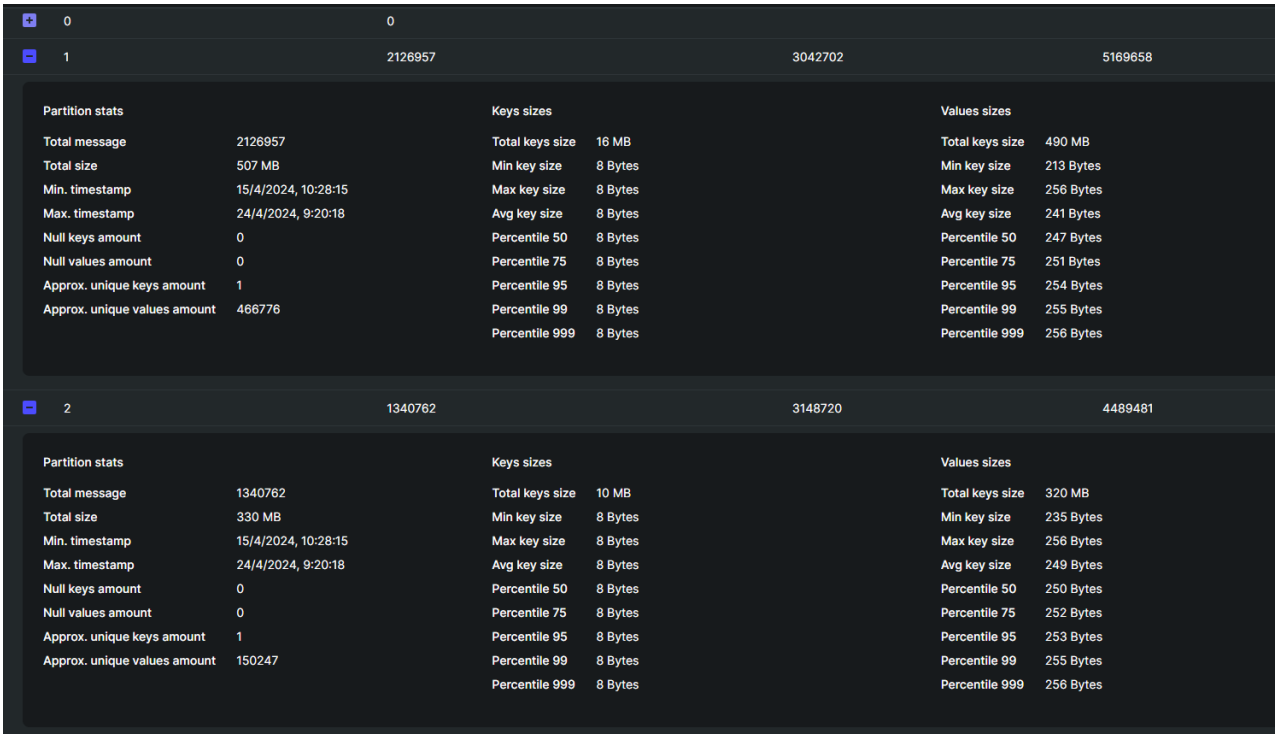


Fig. 21: Estadísticas de un Topic con particiones.

Fuente. Elaboración propia.

Dentro de cada partición, aparece el tamaño de cada mensaje y el número total de los mensajes y el timestamp de cada una de las particiones.

Con esto podemos comercializar nuestro Topic o nuestros eventos de forma que si un consumidor consume hasta una cantidad determinada de eventos tendrá que pagar una cuota.

De la misma manera si nuestro Productor utiliza un número determinado de almacenamiento de nuestra Kafka también tendría que pagar una cuota.

Podemos monitorizar los clientes que se están consumiendo el topic dentro de la pestaña de “Consumers” como aparece en la figura de abajo.

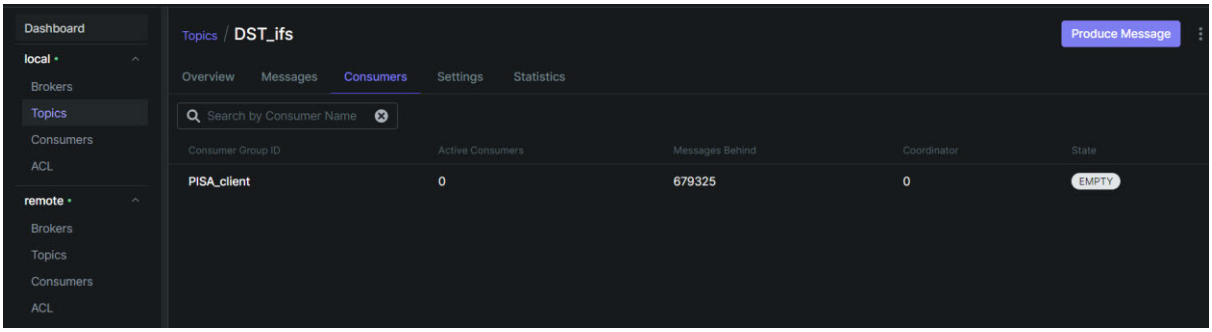


Fig. 22: Consumidores de un Topic.

Fuente. Elaboración propia.

7. Caso de uso

La arquitectura Data-Driven como otras arquitecturas también tiene sus pros y sus contras. Y como cualquier tecnología, sus deficiencias deben ser consideradas.

7.1.6 Problemática del escenario

Uno de los desafíos más significativos que enfrentamos en este escenario radica en la implementación de la funcionalidad de Captura de Datos Cambiantes (CDC) en la base de datos. Al activar esta opción, el sistema automáticamente genera tablas auxiliares que funcionan como colas temporales para almacenar eventos antes de su transmisión al servidor Kafka. Estas tablas del sistema consumen un volumen considerable de espacio de almacenamiento, acumulando todos los eventos registrados a lo largo del tiempo. Dicha acumulación de datos solo puede ser controlada mediante una configuración que permite la eliminación automática de la información cada cierto día, dado que no se ofrece la posibilidad de eliminar los registros manualmente. Esta limitación es inherente al diseño de la base de datos y representa una restricción considerable en la gestión de recursos de almacenamiento.

Adicionalmente, enfrentamos complicaciones en la utilización del servidor Kafka cuando opera sobre sistemas Windows. En estos sistemas, el proceso de eliminación de mensajes de un tópico puede provocar la caída del servidor. Esto se debe a dificultades en el manejo de los archivos de registro (logs), que, en teoría, deberían haber sido eliminados. Una solución provisional que hemos implementado consiste en la eliminación manual de estos ficheros de registros. Continuamos investigando alternativas más eficientes y automáticas para resolver esta problemática, con el objetivo de optimizar la estabilidad y funcionalidad del sistema en entornos Windows.

8. Presupuesto

8. Presupuesto

Dependiendo del tamaño y complejidad que se quiera utilizar con esta arquitectura habrá una gran variación en los presupuestos. Durante el escenario presentado el único presupuesto se corresponde con el trabajo del alumnado de alrededor de medio año. Llevándolo al mundo empresarial un proyecto podría requerir unos presupuestos estimados como el que NO se puede apreciar en la siguiente tabla:

Tabla 10: Presupuesto de la arquitectura Data-Driven.

Fase	Costo Estimado (€)	Descripción
Análisis y diseño	€50.000 - €150.000	- Consultores - Herramientas de modelado de datos
Implementación	€30.000 - €200.000	- Plataforma Apache Kafka - Servidores y almacenamiento - Integraciones con sistemas existentes
Análisis y visualización de datos	€10.000 - €40.000	- Herramientas de análisis de datos - Cuadros de mando e informes
Costos continuos	€15.000 - €40.000 por año	- Licencias de software - Mantenimiento y soporte - Personal
Servidores físicos	€10.000 - €20.000 por año	- Hardware - Seguridad de los servidores
Total	**€11.000 - €450.000 o más**	**Incluye todos los costos mencionados anteriormente**

9. Impacto del proyecto

9. Impacto del proyecto

La arquitectura que utiliza Apache Kafka para la recopilación, el procesamiento, el análisis y la utilización de los datos puede tener un amplio espectro de implicaciones que abarcan diferentes áreas como:

1. Sociales:
   - Mejora de la toma de decisiones públicas: Los gobiernos pueden utilizar los datos para abordar problemas sociales como la pobreza, la educación, la salud y la seguridad pública de manera más eficaz.
   - Empoderamiento ciudadano: Los ciudadanos pueden acceder a información y servicios públicos más relevantes y personalizados.
   - Promoción de la transparencia y la rendición de cuentas: Los datos pueden utilizarse para monitorear el desempeño del gobierno y garantizar la transparencia en la toma de decisiones.
2. Salud y seguridad:
   - Mejora de la atención médica: Los datos de salud pueden utilizarse para diagnosticar enfermedades con mayor precisión, desarrollar tratamientos personalizados y mejorar la prevención de enfermedades.
   - Reducción de accidentes y delitos: Los datos pueden utilizarse para identificar patrones de delincuencia y accidentes, y desarrollar estrategias para prevenirlos.
   - Protección del medio ambiente: Los datos pueden utilizarse para monitorear la calidad del aire y el agua, y para desarrollar estrategias para proteger el medio ambiente.
3. Ambientales:
   - Reducción del consumo de energía: Los datos pueden utilizarse para optimizar el uso de energía en edificios, transporte y otros sectores.
   - Gestión sostenible de recursos: Los datos pueden utilizarse para gestionar de manera más sostenible los recursos naturales como el agua, la tierra y los bosques.
   - Mitigación del cambio climático: Los datos pueden utilizarse para comprender mejor los efectos del cambio climático y desarrollar estrategias para mitigarlo.
4. Económicas:
   - Aumento de la productividad: Los datos pueden utilizarse para optimizar procesos comerciales, mejorar la eficiencia y aumentar la productividad.
   - Desarrollo de nuevos productos y servicios: Los datos pueden utilizarse para comprender mejor las necesidades de los clientes y desarrollar nuevos productos y servicios que satisfagan esas necesidades.

9. Impacto del proyecto

- Creación de nuevos empleos: La industria de la analítica de datos está en rápido crecimiento y está creando nuevos empleos en diversos sectores.

5. Tecnológicas:
   - Avance de la inteligencia artificial: Los datos son esenciales para el desarrollo de la inteligencia artificial, que tiene el potencial de revolucionar muchos sectores de la economía.
   - Mejora de la ciberseguridad: Los datos pueden utilizarse para detectar y prevenir ataques cibernéticos.
   - Desarrollo de nuevas tecnologías: Los datos pueden utilizarse para desarrollar nuevas tecnologías en diversas áreas, como la robótica, la automatización y la Internet de las cosas.
6. Industriales:
   - Optimización de la cadena de suministro: Los datos pueden utilizarse para optimizar la cadena de suministro, reduciendo costos y mejorando la eficiencia.
   - Desarrollo de nuevos modelos de negocio: Los datos pueden utilizarse para desarrollar nuevos modelos de negocio que sean más innovadores y eficientes.
   - Mejora de la competitividad: Las empresas que utilizan datos de manera efectiva pueden obtener una ventaja competitiva sobre sus rivales.

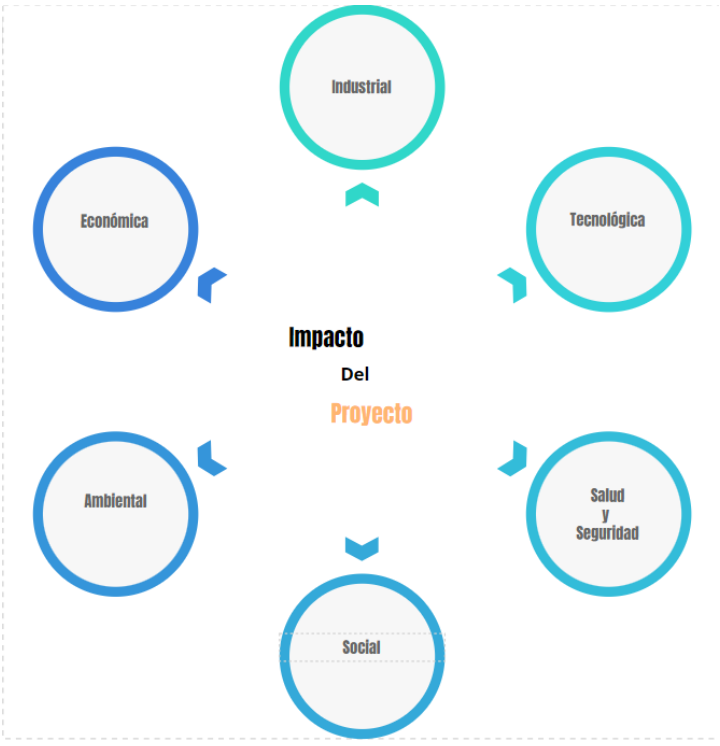


Fig. 23: Impacto del proyecto.

Fuente. Elaboración propia.

10. Trabajos futuros y líneas de mejora

10. Trabajos futuros y líneas de mejora

Un posible trabajo futuro es utilizar Apache Pulsar como plataforma de diseño e implementación para nuestra área de recopilación, procesamiento y transformación de datos puesto que se trata de una plataforma moderna y con bajas latencias.

Además, implementar las comunicaciones entre productor con el consumidor encriptadas, es decir, tener nuestros datos aún más protegidos a posibles amenazas externas utilizando criptografía hibrida, ya que es la más segura.

Otro caso de línea de mejora es utilizar todos los brokers el sistema operativo Linux ya que es menos problemático, y utilizar programas de depuración para monitorizar mejor los fallos y los bugs.

Luego, mejorar la calidad de los datos desarrollando herramientas para procesos automáticos que identifiquen y corrijan los datos incorrectos o inconsistentes en el tiempo real.

Por otro lado, buscar una alternativa de escalabilidad en la parte del almacenamiento de los datos enfocado a que no requiera tanta memoria para ello.

Como última mejora propuesta, integrar nuestra arquitectura con Machine Learning y AI incorporando modelos avanzados de aprendizaje automático y la inteligencia artificial con la funcionalidad de realizar una toma de decisión para el cliente.

11. Conclusiones

11. Conclusiones

Las arquitecturas Data-Driven que utilizan Apache Kafka para la recopilación, el procesamiento, la transformación, y el análisis de datos ofrecen una amplia gama de beneficios para las empresas y la sociedad en general.

Mediante la implementación de la arquitectura propuesta, se ha logrado optimizar el acceso y la gestión de los datos, organizándolos de manera más eficiente y garantizando su calidad. Asimismo, estos datos han sido fundamentales para respaldar el análisis y la toma de decisiones estratégicas, al integrarlos de manera efectiva con los procesos de negocio. De esta forma, se han cumplido los objetivos establecidos para este proyecto, asegurando una alineación con las metas planteadas y potenciando la capacidad de la organización para responder a sus necesidades de manera ágil y precisa.

Esta arquitectura ha logrado integrar bases de datos con servidores Kafka para la lectura y escritura de datos, así como el almacenamiento de estos en una base de datos original o secundaria. Esta integración permite a las organizaciones tomar decisiones más eficientes y generar valor a partir de los resultados obtenidos. La optimización en la toma de decisiones y la reducción de costes contribuyen a un incremento en los ingresos y a una mejora en la satisfacción de los clientes.

Además, la arquitectura ha facilitado el análisis de estadísticas de datos en tiempo real mediante la interfaz gráfica Kafka UI, lo que permite a las organizaciones aprovechar estas estadísticas para obtener beneficios estratégicos y operativos.

Finalmente, con el caso de uso propuesto, podemos contemplar que lo más complicado de la arquitectura es la configuración de cada una de las herramientas y la conexión entre ellas.

Sin esta arquitectura, la obtención de los resultados para las decisiones a tomar por parte de las empresas y compañías pueden resultar afectadas, por ejemplo, por la tardanza o la precisión de los resultados y gracias a esta arquitectura las organizaciones podrán minimizar recursos para utilizarlos en otras áreas más necesarias.

12. Referencias

12. Referencias

[1] KongHQ. (s. f.). *Kong Configuration File - Kong Gateway | Kong Docs*. Kong Docs. https://docs.konghq.com/gateway/latest/production/kong-conf/

[2] 1.6 Diseño de software de arquitectura distribuida. (2016c, mayo 31). Arquitecturas de Software. https://ingsoftwareisc.wordpress.com/2016/05/27/1-6-diseno-de-software-de-arquitectura-distribuida/

[3] Hiberus. (2023, 28 noviembre). *De una arquitectura tradicional a una arquitectura microservicios*. Blog de Hiberus. https://www.hiberus.com/crecemos-contigo/de-una-arquitectura-tradicional-a-microservicios/

[4] MySQL: MySQL Documentation. (s. f.). https://dev.mysql.com/doc/

[5] *PostgreSQL 16.2 documentation*. (2024, 8 febrero). PostgreSQL Documentation. https://www.postgresql.org/docs/current/

[6] *Oracle Database Online Documentation 11G*. (2006, 8 diciembre). Copyright 2011, 2014, Oracle And/Or Its Affiliates. https://docs.oracle.com/cd/E11882_01/index.htm

[7] rwestMSFT. (2024, 21 febrero). *¿Qué es SQL Server? - SQL Server*. Microsoft Learn. https://learn.microsoft.com/es-es/sql/sql-server/what-is-sql-server?view=sql-server-ver16

[8] *MongoDB documentation*. (s. f.). MongoDB Documentation. https://www.mongodb.com/docs/

[9] *Welcome to Apache Cassandra's documentation! | Apache Cassandra Documentation*. (s. f.). Apache Cassandra. https://cassandra.apache.org/doc/latest/

[10] *Redis - the real-time data platform*. (2024, 9 abril). Redis. https://redis.io/

[11] *¿Qué es HDFS? Sistema de archivos distribuido Apache Hadoop | IBM*. (s. f.). https://www.ibm.com/es-es/topics/hdfs

[12] *Apache HBase – Apache HBaseTM Home*. (s. f.). https://hbase.apache.org/

[13] *Apache Kafka*. (s. f.-b). Apache Kafka. https://kafka.apache.org/documentation/

[14] *Servicio de datos de streaming administrado - Preguntas frecuentes de Amazon Kinesis Data Streams - AWS*. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/kinesis/data-streams/?p=pm&c=aa&pd=kinesis&z=4

[15] *Documentación de PUB/SuB | Documentación de Cloud PUB/SuB | Google Cloud*. (s. f.). Google Cloud. https://cloud.google.com/pubsub/docs?hl=es-419

[16] Quick *Start for Confluent Cloud | Confluent Documentation*. (s. f.). https://docs.confluent.io/cloud/current/get-started/index.html

[17] *IBM Cloud Docs*. (s. f.). https://cloud.ibm.com/docs/EventStreams?topic=EventStreams-getting-started

[18] *Apache Pulsar | Apache Pulsar*. (s. f.). https://pulsar.apache.org/

[19] *IBM MQ 9.1*. (s. f.). https://www.ibm.com/docs/es/ibm-mq/9.1?topic=mq-introduction

[20] Fiz, J. M. (2022, 18 mayo). ¿Cómo usar Kafka Connect? *Paradigma Digital*. https://www.paradigmadigital.com/dev/kafka-connect-para-conectarlos-todos/

[21] *Debezium connector for SQL Server: Debezium Documentation*. (s. f.).

[22] *JDBC Source Connector for Confluent Platform | Confluent Documentation*. (s. f.). https://docs.confluent.io/kafka-connectors/jdbc/current/source-connector/overview.html

[23] *Lesson: JDBC Introduction (The JavaTM Tutorials > JDBC Database Access)*. (s. f.). https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

[24] *Apache Kafka*. (s. f.). Apache Kafka. https://kafka.apache.org/documentation/streams/

[25] *About | UI for Apache Kafka*. (s. f.-b). https://docs.kafka-ui.provectus.io/

[26] *kadeck | Kafka UI - Your team's Apache Kafka tool belt*. (s. f.). https://www.kadeck.com/?utm_source=ga&utm_medium=eu-ui&gad_source=1&gclid=CjwKCAjw8diwBhAbEiwA7i_sJa4zGr5A3SoHRcR2_VgUsV547kg09QYe7zBRAUR0HkHEJyMK17qQIhoCZk0QAvD_BwE

[27] *«Docker overview»*. (2023, 22 noviembre). Docker Documentation. https://docs.docker.com/get-started/overview/

[28] *systemctl*. (s. f.). https://www.freedesktop.org/software/systemd/man/latest/systemctl.html

[29] Winsw. (s. f.). *GitHub - winsw/winsw: A wrapper executable that can run any executable as a Windows service, in a permissive license.* GitHub. https://github.com/winsw/winsw

[30] *¿Qué es Apache Kafka?* (s. f.). https://www.redhat.com/es/topics/integration/what-is-apache-kafka

[31] Madrid, V. (2021, 18 mayo). *Aprendiendo Apache Kafka (Parte 2): Conceptos básicos*. Enmilocalfunciona. https://www.enmilocalfunciona.io/aprendiendo-apache-kafka-parte-2-2/[23] *Lesson: JDBC Introduction (The JavaTM Tutorials > JDBC Database Access)*. (s. f.). https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html

[32] Fernandez, O. (2024, 1 marzo). *Apache Kafka: Introducción*. Aprender BIG DATA. https://aprenderbigdata.com/introduccion-apache-kafka/

[33] *Introduction to Apache Kafka | Confluent Documentation*. (s. f.). https://docs.confluent.io/kafka/introduction.html

Anexo I: Configuración y Ejecución del Kong

Anexo I: Configuración y Ejecución del Kong

Lo primero de todo es obtener el Kong, que lo podemos conseguir mediante la página oficial de Kong [1].

Kong Gateway lleva instalado con un fichero de configuración por defecto, puedes realizar cambios utilizando ese fichero como plantilla.

Las configuraciones importantes que se deben tener en cuenta:

- Proxy_listen: Define en qué dirección y puerto Kong aceptará las solicitudes entrantes.
- Admin_listen: Define las direcciones y puertos donde Kong expondrá su interfaz de administración permitiendo realizar configuraciones al sistema y consultar métricas.
- Database: Especifica sistema de almacenamiento del Kong. Nuestro caso SQL Server.
- Sql_server_host: nombre del servidor.
- sql_server_port : puerto del sql server.
- sql_server_user : usuario del sql server.
- sql_server_password : contraseña del sql server
- sql_server_database : nombre de la base de datos.
- Client_ssl: define si se va a querer seguridad o no para los clientes del Kong.
- Plugins: Define las extensiones del Kong.
- Mem_cache_size: Define el tamaño de la memoria caché que Kong utilizará.

Para confirmar que tu configuración está bien correcta puedes hacerlo con

```
kong check /etc/kong/kong.config
```

Código 1: Comprobación de la configuración

Una vez obtenido las configuraciones es iniciarlo con el comando

```
kong start –config /etc/kong/kong.config
```

Código 2: Comando de inicio

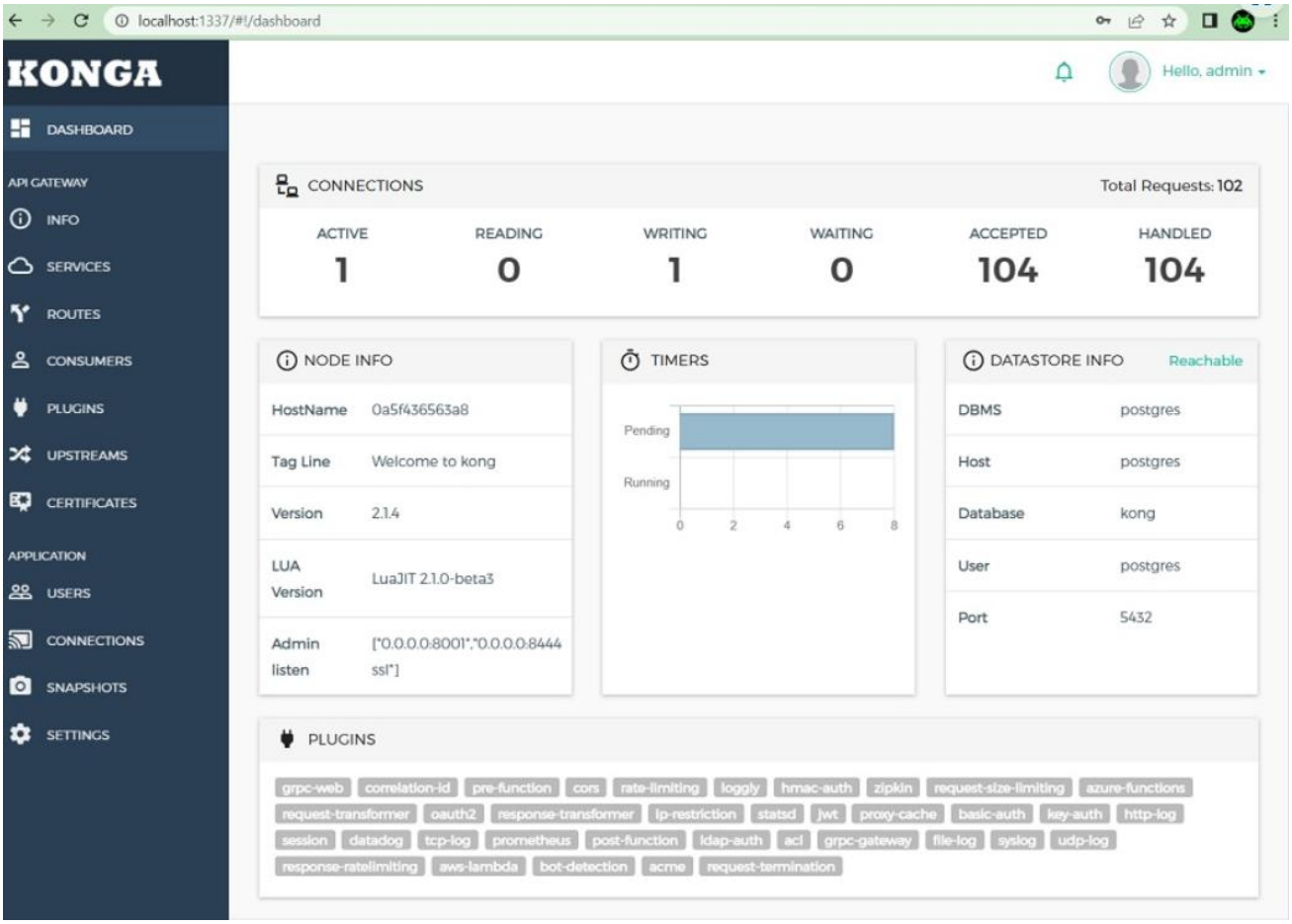


Fig. 24: Representación UI del Kong.

Fuente. [1].

Anexo II: Configuración y Ejecución del servidor Kafka

Anexo II: Configuración y Ejecución del servidor Kafka

Para obtener un correcto funcionamiento con el servidor de Apache Kafka se debe tener unas configuraciones y unos comandos de ejecución.

- Ejecutar el servidor de zookeeper con su configuración.
  - Path/zookeeper-server-start config/zookeeper.properties
  - Dentro del fichero "zookeeper.properties":
    - dataDir= "Path para guardar Snapshot".
    - clientPort= "Puerto por la cual se conecta un servidor Kafka.
- Ejecutar el servidor de Kafka con su configuración.
  - Path/Kafka-server-start config/server.properties
  - Dentro del fichero "server.properties".
    - broker.id= "Id único del broker"
    - listeners= "Formato://IP:Puerto para la conexión de un cliente".
    - log.dirs= "Directorio que guarda los logs".
    - num.network.threads=3
    - num.io.threads=8
    - socket.send.buffer.bytes=102400
    - socket.receive.buffer.bytes=102400
    - socket.request.max.bytes=104857600
    - num.partitions=1
    - num.recovery.threads.per.data.dir=1
    - offsets.topic.replication.factor=1
    - transaction.state.log.replication.factor=1
    - transaction.state.log.min.isr=1
    - log.retention.hours=168
    - zookeeper.connect=localhost:2181
    - zookeeper.connection.timeout.ms=18000
    - group.initial.rebalance.delay.ms=0

Creación de los Topics

La creación de los topics se puede realizar de manera automática si en la configuración del servidor no tiene desactivado

```
auto.create.topics.enable=false
```

En caso de la creación automática, el topic tendría una partición con un factor de replicación 1, tiempo de retención de los topic por defecto de 7 días.
En caso de tener la configuración de creación automática de los topic desactivada, la creación de los topics se debe realizar de manera manual, mediante comandos:

Anexo II: Configuración y Ejecución del servidor Kafka

En Windows:

```
$|kafka|>.\bin\windows\kafka-topics.bat --create --topic "nombre del topic" --bootstrap-server "localhost":9092
```

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- "nombre del topic": Debe indicar el nombre del topic que se quiera publicar.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a almacenar el topic.

En Linux:

```
$|kafka|>.\bin\windows\kafka-topics.bat --create --topic "nombre del topic" --bootstrap-server "localhost":9092
```

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- "nombre del topic": Debe indicar el nombre del topic que se quiera publicar.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a almacenar el topic.

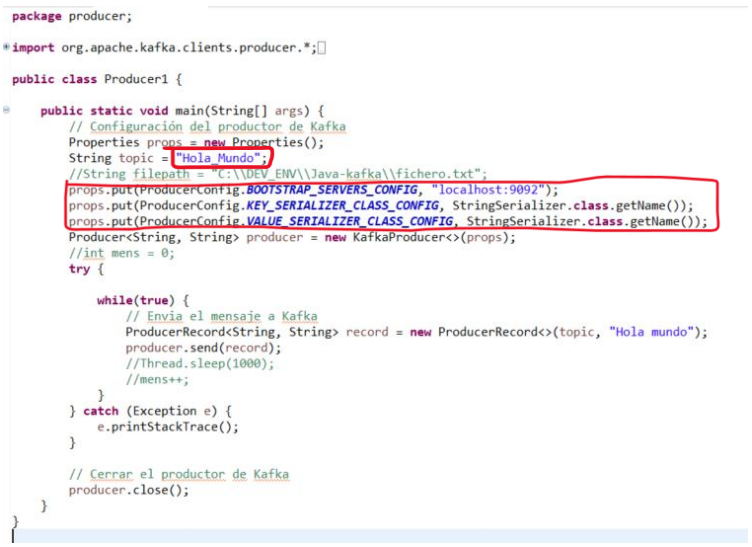
Tanto para Windows como Linux se puede especificar el factor de replicaciones y el número de particiones mediante la siguiente configuración:

- "--replication-factor X": Donde X indica el número del factor de replicación.
- "--partitions X" : Donde X indica el número de particiones.

Ejecución del productor y el consumidor

- Ejecutar un productor en el entorno de Apache Kafka utilizando Java Maven requiere unas librerías especificas del Apache Kafka denominada "***kafka-client***". En la *fig. 24* se han resaltado con el color rojo las propiedades imprescindibles para conseguir de forma exitosa la ejecución del productor.

Anexo II: Configuración y Ejecución del servidor Kafka



```java
package producer;

import org.apache.kafka.clients.producer.*;

public class Producer1 {

    public static void main(String[] args) {
        // Configuración del productor de Kafka
        Properties props = new Properties();
        String topic = "Hola_Mundo";
        //String filepath = "C:\\DEV_ENV\\Java-kafka\\fichero.txt";
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        Producer<String, String> producer = new KafkaProducer<>(props);
        //int mens = 0;
        try {

            while(true) {
                // Envía el mensaje a Kafka
                ProducerRecord<String, String> record = new ProducerRecord<>(topic, "Hola mundo");
                producer.send(record);
                //Thread.sleep(1000);
                //mens++;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Cerrar el productor de Kafka
        producer.close();
    }
}
```

Fig. 25: Captura de pantalla del productor de mensaje.

Fuente. Elaboración propia.

Para ejecutar un consumidor en el entorno de script en Windows o en Linux requiere el siguiente comando:

Windows:

```
$|kafka|>.\bin\windows\kafka-console-producer.bat --topic "nombre del topic" --bootstrap-server "localhost":9092
```

Código 3: código de lanzamiento del Broker en Windows

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- "nombre del topic": Debe indicar el nombre del topic que se quiera publicar.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a publicar.

Linux:

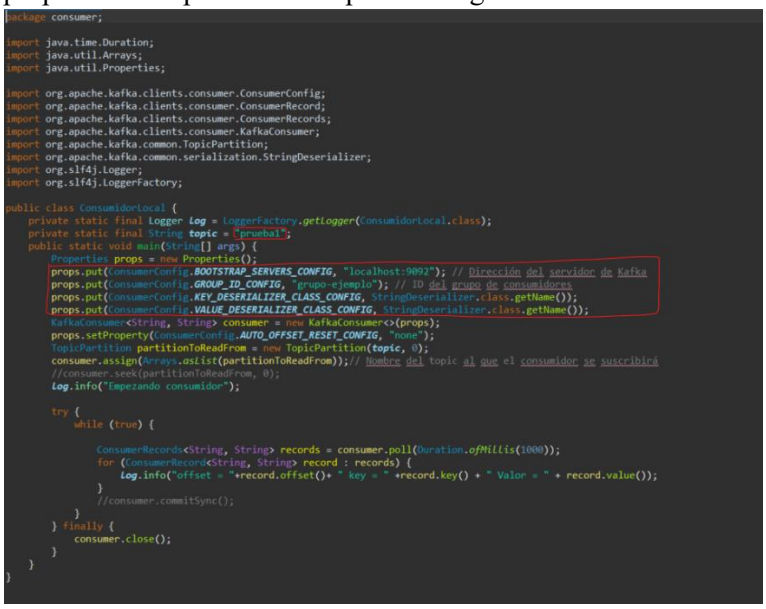
```
$|kafka|>.\bin\kafka-console-producer.sh --topic "nombre del topic" --bootstrap-server "localhost":9092
```

Código 4: código de lanzamiento del Broker en Windows

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- "nombre del topic": Debe indicar el nombre del topic que se quiera publicar.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a publicar.

Anexo II: Configuración y Ejecución del servidor Kafka

- Ejecutar un consumidor en el entorno de Apache Kafka utilizando Java Maven requiere unas librerías especificas del Apache Kafka denominada "***kafka-client***". En la *fig. 25* se han resaltado con el color rojo las propiedades imprescindibles para conseguir de forma exitosa la ejecución del consumidor.



```
package consumer;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ConsumidorLocal {
    private static final Logger log = LoggerFactory.getLogger(ConsumidorLocal.class);
    private static final String topic = "prueba1";
    public static void main(String[] args) {
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092"); // Dirección del servidor de Kafka
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "grupo-ejemplo"); // ID del grupo de consumidores
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
        props.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "none");
        TopicPartition partitionToReadFrom = new TopicPartition(topic, 0);
        consumer.assign(Arrays.asList(partitionToReadFrom));// Nombre del topic al que el consumidor se suscribirá
        //consumer.seek(partitionToReadFrom, 0);
        log.info("Empezando consumidor");

        try {
            while (true) {

                ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(1000));
                for (ConsumerRecord<String, String> record : records) {
                    log.info("offset = "+record.offset()+ " key = " +record.key() + " Valor = " + record.value());
                }
                //consumer.commitSync();
            }
        } finally {
            consumer.close();
        }
    }
}
```

Fig. 26: Captura de pantalla del consumidor del mensaje.

Fuente. Elaboración propia.

Para ejecutar un consumidor en el entorno de script en Windows o en Linux requiere el siguiente comando:

Windows:

```
$|kafka|>.\bin\windows\kafka-console-consumer.bat --topic "nombre del topic" --from-beginning --bootstrap-server "localhost":9092
```

Código 5: código de lanzamiento del consumidor en Windows.

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- –from-beginning: Es la configuración del consumidor indicando lectura del topic desde el offset 0.
- "nombre del topic": Debe indicar el nombre del topic que se quiera suscribir.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a suscribir.

Linux:

```
$|kafka|>.\bin \kafka-console-consumer.sh --topic "nombre del topic" --from-beginning --bootstrap-server "localhost":9092
```

Código 6: código de lanzamiento del consumidor en Linux.

- $|kafka|: Es el path hasta llegar directorio donde tienes instalado Apache Kafka.
- –from-beginning: Es la configuración del consumidor indicando lectura del topic desde el offset 0.

- "nombre del topic": Debe indicar el nombre del topic que se quiera suscribir.
- "localhost":9092: Debe indicar la dirección IP del servidor y el puerto por la cual se va a suscribir.

```
[main] INFO consumer.ConsumidorLocal - offset = 256703596 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703597 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703598 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703599 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703600 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703601 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703602 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703603 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703604 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703605 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703606 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703607 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703608 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703609 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703610 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703611 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703612 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703613 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703614 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703615 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703616 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703617 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703618 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703619 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703620 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703621 key = null Valor = Hola mundo
[main] INFO consumer.ConsumidorLocal - offset = 256703622 key = null Valor = Hola mundo
```

Fig. 27: Captura de pantalla de los mensajes consumidos.

Fuente. Elaboración propia.

Anexo III: Configuración y Ejecución del servidor Kafka

Anexo III: Configuración y Ejecución del servidor Kafka

Para conseguir el escenario debemos de tener unas configuraciones en nuestros SQL Server. Estas configuraciones son obligatorias para el correcto funcionamiento de nuestro escenario.

- La principal configuración es activar el SQL Server Agent.
- Después, mediante T-SQL escribir siguiente código para tener activado CDC como se ve en la *Fig. 27* cambiando lo destacado en rojo por la base de datos de uso.

```
-- ================================
-- Enable Database for CDC Template
-- ================================
USE Kafka
GO

EXEC sys.sp_cdc_enable_db
GO
```

Fig. 28: Captura de la configuración CDC para Base de Datos.

Fuente. Elaboración propia.

- Luego, mediante T-SQL activar permiso CDC para los bases de datos que quieras recoger eventos mediante códigos de la fig. 28. En la figura, las configuraciones resaltadas son las configuraciones que se deben cambiar dependiendo de la base de datos, tabla, esquema…
    - SOURCES: Base de datos que se va a recoger el CDC.
    - @source_schema: Nombre del esquema de la tabla que se activará el CDC.
    - @source_name: Nombre de la tabla que se activará el CDC.
    - @filegroup_name: Nombre del grupo lógico, son grupos de ficheros que organizan los ficheros de la base de datos según su función o tipo de datos. Por defecto es “PRIMARY”.

```
-- ====================================================
-- Enable a Table Specifying Filegroup Option Template
-- ====================================================
USE SOURCES
GO

EXEC sys.sp_cdc_enable_table
    @source_schema = N'dbo',
    @source_name   = N'Ejemplo',
    @role_name     = Null,
    @filegroup_name = N'PRIMARY',
    @supports_net_changes = 0
GO
```

Fig. 29: Captura de la configuración CDC para una tabla.

Fuente. Elaboración propia.

Anexo III: Configuración y Ejecución del servidor Kafka

- Por último, nuestro Worker también necesita estar configurado tal que cuando sea llamado por el Worker, éste sepa a que base de datos recurrir y a qué tablas coger eventos para ello, nuestro Conector debe de ser un fichero properties como de la *fig. 29*.

```
name=sqlserver-connector
connector.class=io.debezium.connector.sqlserver.SqlServerConnector
database.hostname=10.232.172.141
database.user=sa
database.password=
database.instance=SQL2022DEVELOP
database.names=Kafka
table.include.list=dbo.Ejemplo
topic.prefix=sqlserver
schema.history.internal.kafka.bootstrap.servers=localhost:9092
schema.history.internal.kafka.topic=sqlserver
database.encrypt=false
```

Fig. 30: Captura de la configuración para el Connector.

Fuente. Elaboración propia.

  - name: Nombre del Worker.
  - connector.class: Es el jar que utiliza nuestro conector para extraer eventos del SQL Server.
  - database.hostname: La dirección IP o el nombre del host que reside el SQL Server.
  - database.user: El nombre de usuario del SQL Server.
  - database.password: Contraseña del usuario definido anteriormente.
  - database.instancia: Instancia del SQL Server.
  - database.names: Nombres de las bases de datos que tiene que extraer eventos.
  - table.include.list: Listas de tablas que tienen permiso de CDC de las bases de datos definido anteriormente.
  - topic.prefix: Nombre del topic de los logs de este connector.
  - schema.history.internal.kafka.bootstrap.servers: Servidor Kafka que se conectara para publicaciones de Topics.
  - schema.history.internal.kafka.topic: Nombre del topic que recurrirá en caso de caída.
  - database.encrypt: Necesidad de encriptar o no los datos.

El conector es muy específico con los datos numéricos por lo que hay que añadir una propiedad

  - decimal.handling.mode=string

En caso de tenemos que extraer informaciones como el TimeStamp o fechas.

Al tener esto configurado, SQL Server puede estar realizando transformaciones en la tabla “Ejemplo” y ser captado por el Worker y producirlo en el Topic “Kafka.dbo.Ejemplo”. [4]

Anexo IV: Pruebas de fallos

Anexo IV: Pruebas de fallos

- Cuando se cae el consumidor

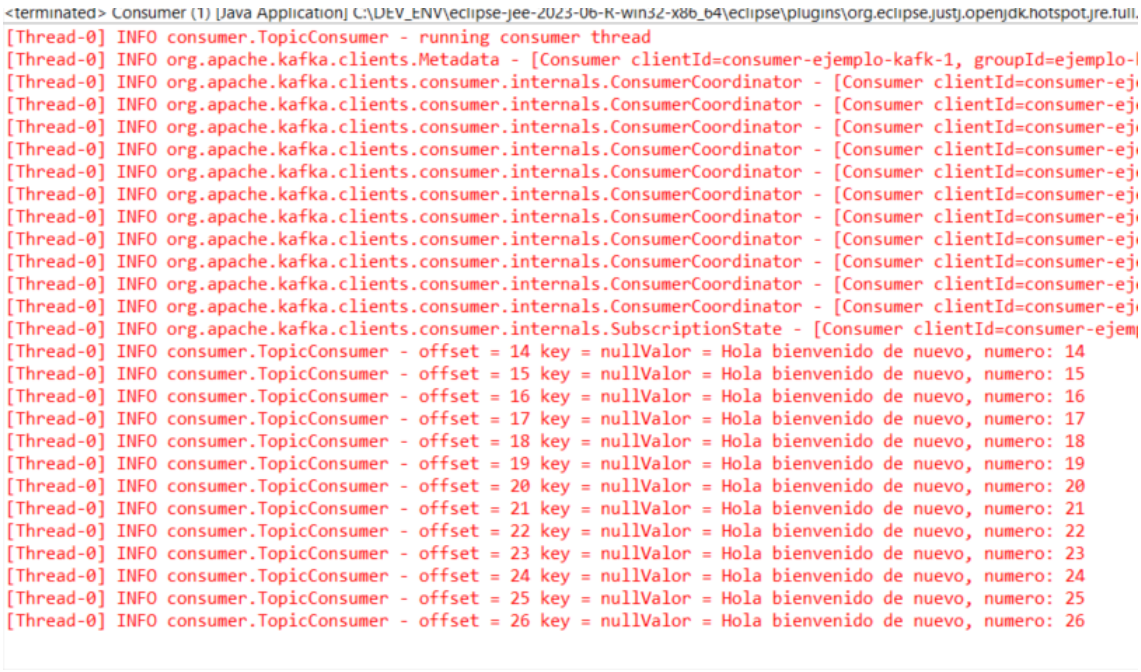


Fig. 31: Captura de mensaje antes de la caída del consumidor.

Fuente. Elaboración propia.

Fig. 32: Captura de mensaje después de establecimiento de conexión con el topic.

Fuente. Elaboración propia.

- Caída del Zookeeper

Anexo IV: Pruebas de fallos

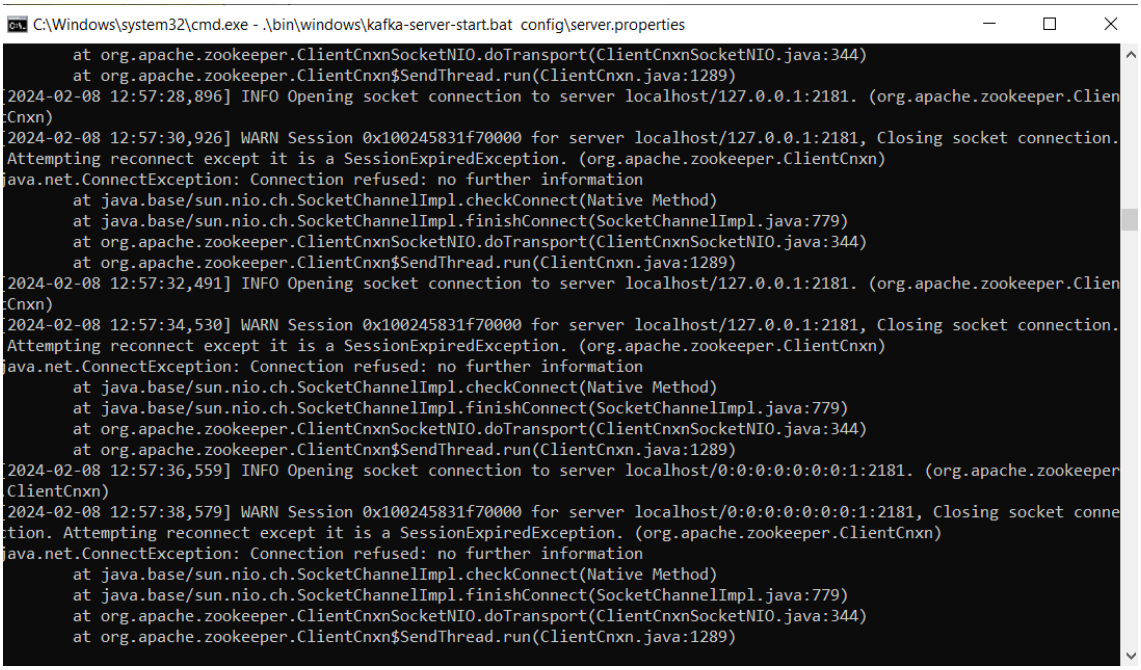


Fig. 33: Captura del error del servidor broker por la caída del Zookeeper.

Fuente. Elaboración propia.

Fig. 34: Captura del mensaje por la caída del Zookeeper.

Fuente. Elaboración propia.

- Caída del Broker

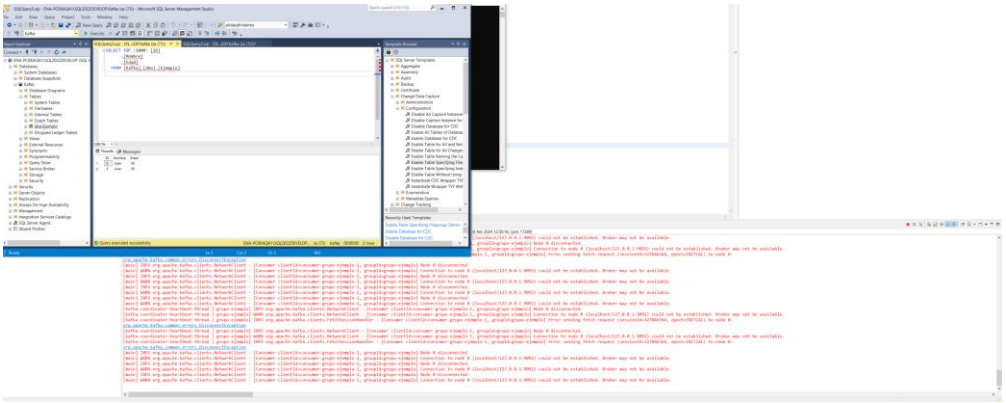


Fig. 35: Captura del mensaje por la caída del Broker.

Fuente. Elaboración propia.

Anexo V: Configuración y Ejecución del microservicio

Anexo V: Configuración y Ejecución del microservicio

El microservicio de la Máquina B es un programa en SpringBoot utilizando la librería KafkaStream. Se había mencionado en el *Estado del Arte* sobre KafkaStream y para entrar en detalle de esta librería se puede visitar a la referencia [8].

Lo primero es inicializar un archivo SpringBoot para Maven de forma para posteriormente insertar unas dependencias Maven para este microservicio. Las dependencias necesarias para el Streaming y transformaciones de los datos son el KafkaStream [8] y Jackson-databind que es una librería que trata con los Json.

En el código principal igual que productor y el consumidor, es necesario establecer unas propiedades como aparece en la *Fig. 30*. La parte resaltada indica las 2 direcciones IP donde uno es para suscribirse al topic y el otro es para publicación después de realizar transformaciones.

```
Properties props = new Properties();
props.put(StreamsConfig.APPLICATION_ID_CONFIG, "filtering-Cr1d4as-as");
props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092,192.168.25.115:9092");
props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
```

Fig. 36: Propiedades necesarias para KafkaStream.

Fuente. Elaboración propia.

Antes de pasar a las transformaciones es necesario indicar al KafkaStream que topic se debe suscribirse para la transformación de ello resaltado en la *Fig. 31*. Luego, para la parte de transformaciones se ha pasado primero por un filtro que descarta datos en caso de que el valor es nulo, como aparece en la *Fig. 31*.

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> sourceStream = builder.stream("Prueba");
KStream<String, String> filtradoStream = sourceStream.filter((key,values) ->{
        try {
            if(values == null) {
                return false;
            }
            ObjectMapper mapper = new ObjectMapper();
            JsonNode jsonNode = mapper.readTree(values);
            JsonNode payload = jsonNode.get("payload");
            JsonNode after = payload.get("after");
            if( after.isMissingNode() || after.isNull()) {
               return false;
            }else {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
            return false; // No se emite el registro en caso de error
        }
});
```

Fig. 37: Configuraciones de transformaciones de KafkaStream.

Anexo VI: Configuración y Ejecución de Kafka UI en Docker

Fuente. Elaboración propia.

Siguiente paso es el código de transformaciones que utilizando la librería de Jackson-databind se puede conseguir lectura de valores de Json y convertirlos en otro formato de Json. Resaltando en la *Fig. 32* el método para la transformación de los mensajes.

```
KStream<String, String> filtradoStream = sourceStream.mapValues(values->{
    ObjectMapper mapper = new ObjectMapper();
    JsonNode newNode = null;
    try {
        ObjectMapper newMapper = new ObjectMapper();
        JsonNode root = mapper.readTree(values);
        newNode = newMapper.createObjectNode()
                    .put("mes", "MES")
                    .put("dia","DIA")
                    .put("hora","HORA")
                    .put("ts","TimeStamp")
                    .put("recordInfos","Informacion");
    } catch (JsonProcessingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return newNode.toPrettyString();

});
```

Fig. 38: Transformaciones de KafkaStream.

Fuente. Elaboración propia.

Por último, se debe definir a que topic de salida se debe publicar los datos transformados como aparece resaltado en la *Fig. 33*.

```
filteredStream.to("DST_LEMD",Produced.with(Serdes.String(), Serdes.String()));
KafkaStreams streams = new KafkaStreams(builder.build(), props);
streams.cleanUp();
streams.start();
```

Fig. 39: Propiedades necesarias para KafkaStream.

Fuente. Elaboración propia.

Anexo VI: Configuración y Ejecución de Kafka UI en Docker

El entorno del sistema que se encuentra Kafka UI es dentro de un Linux (Máquina B) por lo que el primer paso es instalar Docker mediante comando,

Anexo VI: Configuración y Ejecución de Kafka UI en Docker

```
Sudo apt install Docker
```

Después de tener Docker instalado debemos de crear un archivo para guardar nuestro fichero Yaml donde le indicas la imagen que necesitas resaltada en la *Fig. 34*. La parte de ***image*** es necesario poner la imagen diseñado por Kafka UI [20] como indica la Fig. 34. Dentro del ***environment*** es necesario indicarle en nombre, el servidor IP y el puerto de escucha para Kafka y el servidor IP y el puerto de escucha para Zookeeper.

```
version: '3'
services:
  kafka-ui:
    image: provectuslabs/kafka-ui:latest
    ports:
      - "8080:8080"
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: "Máquina B:9092"
      KAFKA_CLUSTERS_0_ZOOKEEPER: "zookeeper:2181"
      KAFKA_CLUSTERS_1_NAME: remote
      KAFKA_CLUSTERS_1_BOOTSTRAPSERVERS: "Máquina A:9092"
      KAFKA_CLUSTERS_1_ZOOKEEPER: "Máquina A:2181"
```

Fig. 40: Fichero Yaml para Kafka UI.

Fuente. Elaboración propia.

Por último, solo necesitamos levantar la imagen dentro del Docker con el comando

```
Docker-compose up -d
```

Código 7: código para levantar el Docker y la UI de kafka.