UD 1. Estrategias de ingestión y almacenamiento de datos en Big Data

Curso de Especialización en IA y BD

5075. Big Data aplicado

Prof. D. Ricardo Mariscal Quintero

Índice General

Índice de Figuras

Información previa

Esta unidad didáctica trabaja la adquisición de la competencia "Gestiona soluciones a problemas propuestos, utilizando sistemas de almacenamiento y herramientas asociadas al centro de datos. (RA 1)".

Los objetivos a cubrir, durante todo el desarrollo de la presente unidad, son:

- Determinar los procedimientos y mecanismos para la ingestión de datos. (CE b)
- Determinar el formato de datos adecuado para el almacenamiento. (CE c)

Los contenidos que se van a desarrollar serán:

- Introducción a la ingestión de datos.
- Herramientas y mecanismos para la ingestión de datos.
- Formatos de datos para almacenamiento en Big Data.
- Diseño de pipelines de ingestión.

Actividades previas

Sin haber revisado aún los contenidos de la unidad, investiga para responder a las siguientes preguntas:

1. Explica el concepto de ingestión de datos en Big Data y diferencia entre ingestión en tiempo real y procesamiento por lotes.

2. ¿Cuál es la importancia de la ingestión de datos en el ciclo de vida de los datos en Big Data? ¿Qué consecuencias puede tener una mala gestión de este proceso?

3. Compara las principales diferencias entre el proceso ETL tradicional y la ingestión de datos en entornos de Big Data.

4. ¿Qué desafíos pueden surgir al manejar grandes volúmenes de datos en entornos de Big Data? Detalla al menos tres y posibles soluciones.

5. Describe las características clave de las herramientas de ingestión como Apache NiFi y Apache Kafka. ¿En qué casos se utilizaría cada una?

6. ¿Qué criterios deberías considerar al seleccionar una herramienta de ingestión para un proyecto de Big Data? Menciona al menos tres factores y su relevancia.

7. Analiza las diferencias entre el almacenamiento en formato de texto plano (como CSV o JSON) y el formato columnar (como Parquet o ORC) en términos de rendimiento y eficiencia.

8. Explica el proceso de ingestión de datos en tiempo real con Kafka y menciona al menos dos casos de uso donde este método sea esencial.

9. ¿Qué son los pipelines de ingestión en Big Data y cuáles son sus componentes clave? Explica la función de cada uno.

10. ¿Cuáles son las ventajas y desventajas de utilizar el procesamiento híbrido (combinando lotes y tiempo real) en la ingestión de datos?

11. Menciona tres tipos de fuentes de datos en Big Data y explica cómo varían en términos de estructura, frecuencia y formato de datos.

12. ¿Qué problemas pueden surgir en la integración de datos de fuentes heterogéneas y cómo podrían solucionarse mediante la normalización y el data wrangling?

13. Describe el impacto del formato de datos en el rendimiento y la escalabilidad de un sistema de almacenamiento en Big Data. ¿Cómo influye en la velocidad de procesamiento y consultas?

14. Explica el concepto de "backpressure" en Apache NiFi. ¿Cómo ayuda a evitar la sobrecarga del sistema durante la ingestión de datos?

15. Define los conceptos de escalabilidad horizontal y tolerancia a fallos en el contexto de la ingestión de datos. ¿Por qué son críticos para entornos de Big Data?

1. Introducción a la ingestión de datos

La ingestión de datos es un proceso fundamental en el ecosistema de Big Data, ya que se refiere al método mediante el cual los datos son recopilados, transferidos, y almacenados en un sistema centralizado para su posterior procesamiento y análisis. En este contexto, la ingestión de datos debe ser capaz de manejar grandes volúmenes de información provenientes de diversas fuentes, en distintos formatos, y con diferentes grados de estructura. Este proceso es crucial para garantizar que los datos estén disponibles de manera eficiente y oportuna para los sistemas de análisis de Big Data, permitiendo así la toma de decisiones informadas.

1.1. Definición y concepto de ingestión de datos

1.1.1. ¿Qué es la ingestión de datos en Big Data?

La **ingestión de datos** en el contexto de Big Data se refiere al proceso de capturar y transferir datos desde diversas fuentes hacia un sistema de almacenamiento centralizado, como un lago de datos, un almacén de datos o una base de datos distribuida. Este proceso es clave para la integración de grandes volúmenes de datos que provienen de fuentes heterogéneas, como sensores IoT, redes sociales, sistemas transaccionales, archivos de log, entre otros.

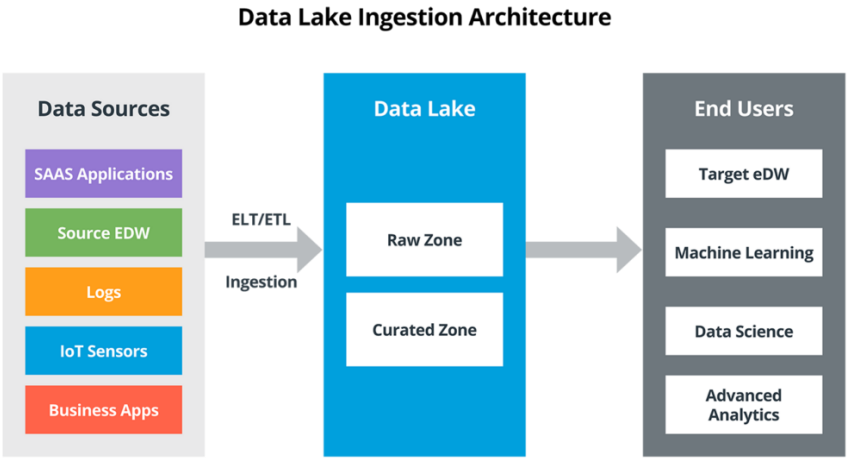

Figura 1. Fase inicial de ingestión de datos (Extracción)

Fuente: dataintegration.info

En un entorno de Big Data, la ingestión de datos se distingue por la necesidad de manejar características propias de los datos masivos, conocidas como las 3V:

- **Volumen:** La capacidad de manejar grandes cantidades de datos, desde terabytes hasta petabytes o más.
- **Velocidad:** La rapidez con la que los datos son generados y deben ser procesados, lo que requiere un sistema de ingestión que pueda operar en tiempo real o casi en tiempo real.
- **Variedad:** La diversidad de formatos de datos que deben ser ingeridos, que pueden ser estructurados, semiestructurados o no estructurados.

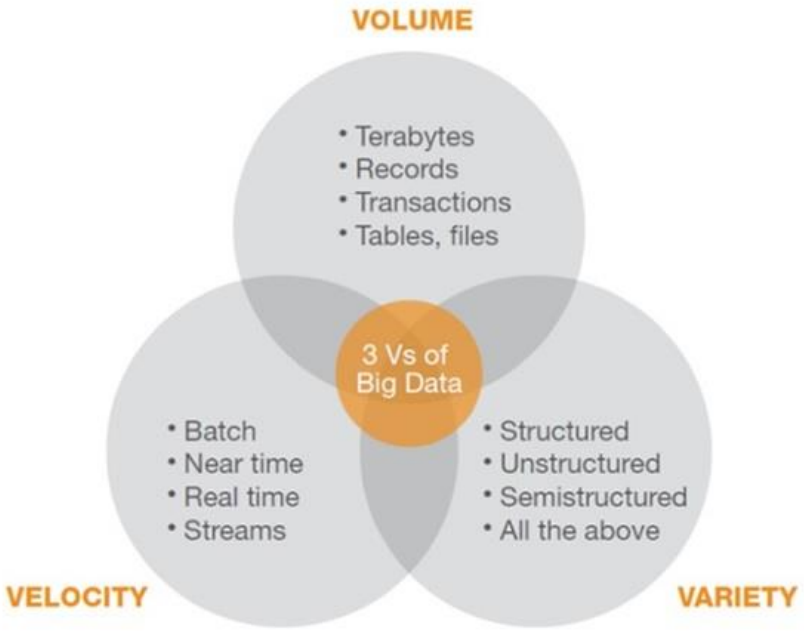

Figura 2. Vs del Big Data

Fuente: indracompany.com

El proceso de ingestión es la primera fase del ciclo de vida de los datos en un sistema de Big Data. Una vez que los datos son ingeridos, pueden ser almacenados, transformados, procesados y analizados para extraer información valiosa que apoye la toma de decisiones en tiempo real o en procesos de análisis más complejos.

1.1.2. Diferencia entre ingestión de datos y ETL tradicional

La ingestión de datos en Big Data se diferencia del proceso tradicional de **ETL (Extract, Transform, Load)** en varios aspectos clave:

- **ETL tradicional:**
    - **Extract (Extracción):** Datos se extraen de una o más fuentes.
    - **Transform (Transformación):** Los datos se limpian, se transforman, y se estructuran de acuerdo con las necesidades del análisis o del sistema que los consumirá.
    - **Load (Carga):** Los datos transformados se cargan en un almacén de datos, típicamente relacional, donde se analizan.

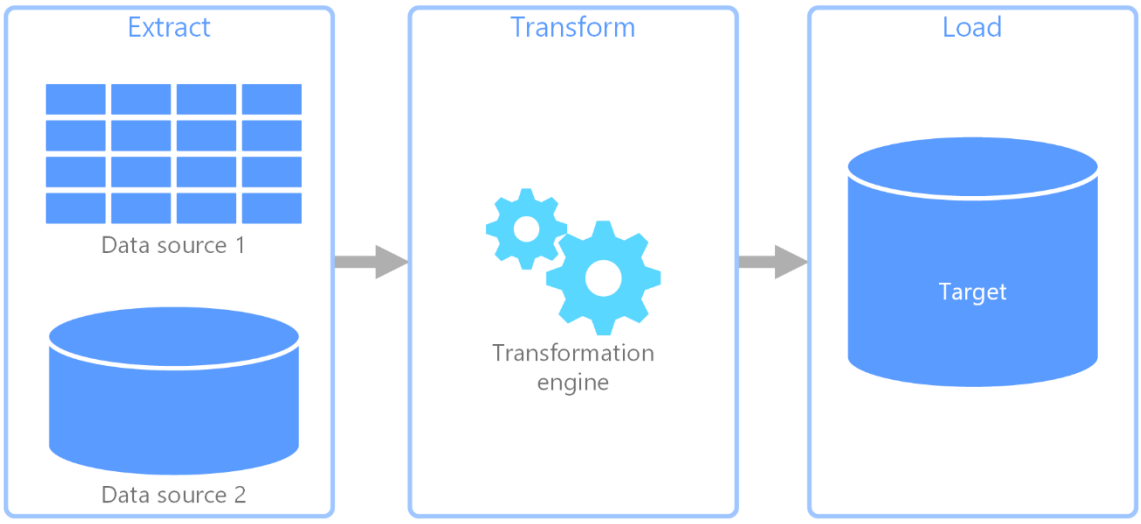

Figura 3. Proceso ETL

Fuente: learn.microsoft.com

- **Ingestión de datos en Big Data:**
    - **Directo al almacenamiento:** En Big Data, los datos se ingieren y almacenan de inmediato, a menudo sin una transformación previa significativa. Esto permite manejar volúmenes masivos de datos sin necesidad de procesarlos en tiempo real.

- **Flexibilidad en el procesamiento:** A diferencia del ETL, que requiere definir el esquema y el procesamiento antes de la carga, la ingestión en Big Data permite almacenar los datos en su forma bruta, permitiendo su procesamiento o análisis según sea necesario, en diferentes momentos.

- **Capacidad para tiempo real:** Además del procesamiento por lotes, la ingestión de datos en Big Data permite la captación y procesamiento en tiempo real, lo que es esencial para aplicaciones como la monitorización en tiempo real, análisis de eventos, y respuestas automatizadas a cambios en los datos.

- **Diversidad de Fuentes y Formatos:** Mientras que ETL tradicional se enfoca en fuentes de datos estructuradas (como bases de datos relacionales), la ingestión en Big Data debe manejar una mayor variedad de datos, incluyendo semiestructurados (XML, JSON) y no estructurados (imágenes, texto libre).

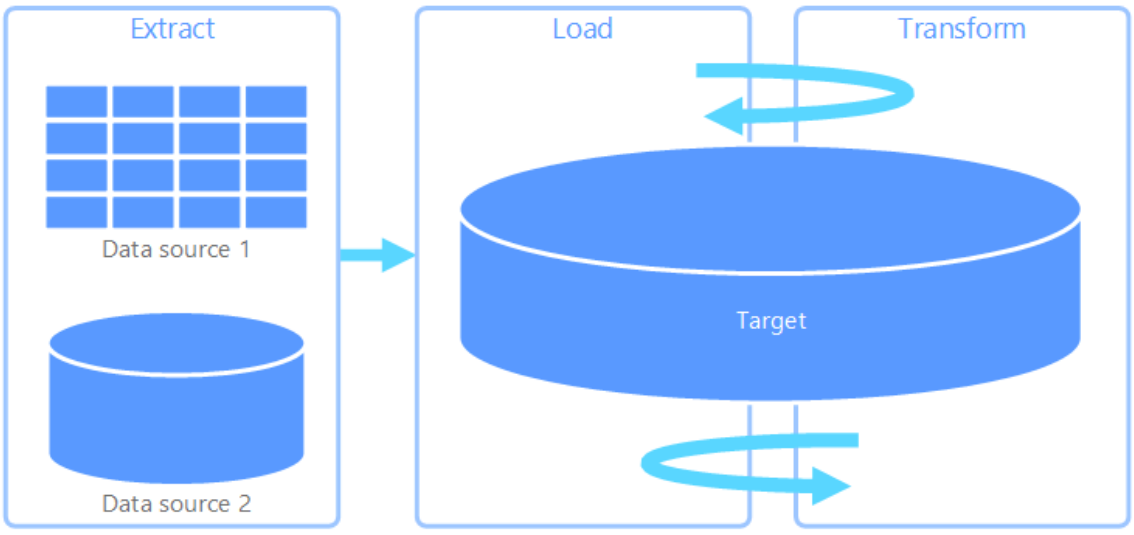

Figura 4. Proceso ELT

Fuente: learn.microsoft.com

1.1.3. Relevancia de la ingestión en entornos de Big Data

La ingestión de datos es crucial en entornos de Big Data por varias razones:

- **Manejo eficiente de volúmenes masivos de datos:** En entornos de Big Data, los sistemas deben ser capaces de manejar cantidades masivas de datos que se generan continuamente. La ingestión eficiente asegura que estos datos se capturen y almacenen sin pérdida de información y con la rapidez necesaria para análisis inmediatos o futuros.

- **Flexibilidad y adaptabilidad:** La capacidad de ingerir datos de múltiples fuentes y en diferentes formatos sin necesidad de una transformación previa permite que las organizaciones sean más flexibles. Esto significa que pueden almacenar grandes cantidades de datos sin preocuparse por cómo los utilizarán más adelante, permitiendo que los datos brutos estén disponibles para una variedad de usos.

- **Soporte para análisis en tiempo real:** La ingestión de datos en tiempo real permite a las organizaciones tomar decisiones informadas rápidamente, basadas en datos actuales. Esto es especialmente importante en sectores como el comercio, finanzas, y seguridad, donde el tiempo de respuesta es crítico.

- **Escalabilidad:** A medida que las necesidades de una organización crecen, su infraestructura de datos también debe escalar. Los sistemas de ingestión de datos en Big Data están diseñados para escalar horizontalmente, lo que significa que pueden manejar más datos simplemente añadiendo más recursos, sin necesidad de cambiar la arquitectura fundamental.

- **Integración de datos heterogéneos:** En Big Data, los datos pueden provenir de una amplia variedad de fuentes, incluyendo redes sociales, dispositivos IoT, logs de servidores, bases de datos transaccionales, y más. La ingestión de datos permite integrar estos datos en un solo sistema, lo que facilita su análisis conjunto y la obtención de insights más completos.

En conclusión, la ingestión de datos en entornos de Big Data es esencial para asegurar que las organizaciones puedan capturar, almacenar, y utilizar de manera eficiente y efectiva los vastos volúmenes de datos generados, permitiendo que los análisis y la toma de decisiones estén basados en información completa y actualizada.

1.2 Tipos de ingestión de datos

La ingestión de datos en Big Data puede llevarse a cabo de diferentes maneras, dependiendo de la naturaleza de los datos, la velocidad con la que son generados y las necesidades específicas del negocio. Los tres principales tipos de ingestión de datos son el **Batch Processing (procesamiento por lotes)**, el **Stream Processing (procesamiento en tiempo real)**, y la **Ingestión Híbrida**, que combina los dos métodos anteriores para aprovechar lo mejor de ambos mundos.

1.2.1. Batch processing (Procesamiento por lotes)

El **Batch processing**, o procesamiento por lotes, es un método en el que los datos se recopilan y se procesan en bloques (o lotes) a intervalos regulares. Este enfoque es adecuado para escenarios en los que los datos no necesitan ser procesados inmediatamente y donde la latencia no es un factor crítico.

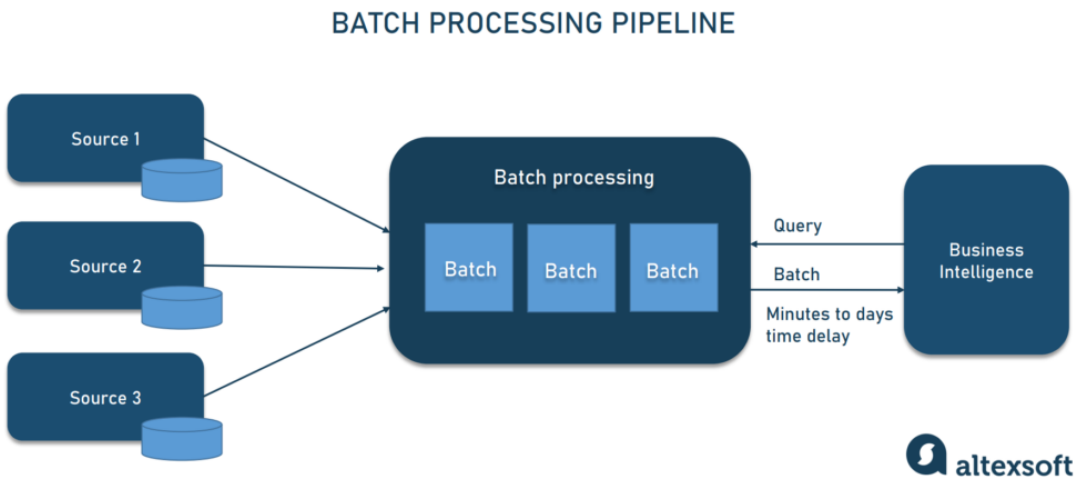

Figura 5. Batch processing

Fuente: altensoft.com

Características:

- **Volumen grande de datos:** El procesamiento por lotes es ideal para manejar grandes volúmenes de datos que se generan durante un periodo de tiempo y se procesan todos juntos en un solo bloque.
- **Latencia alta:** Debido a que los datos se procesan en bloques, la latencia es generalmente alta, lo que significa que hay un retraso entre la generación de los datos y su procesamiento.
- **Eficiencia en costos:** Como los datos se procesan en bloques, los recursos pueden ser utilizados de manera más eficiente, lo que puede reducir los costos de computación.
- **Simplicidad:** Los procesos por lotes tienden a ser más simples de implementar y gestionar, ya que no requieren infraestructura en tiempo real.

Casos de uso:

- **Análisis de ventas diarias:** Una tienda puede recopilar todos los datos de ventas realizados durante el día y procesarlos en un lote durante la noche para generar informes de ventas.
- **Procesamiento de nóminas:** Las empresas suelen procesar los salarios de los empleados en lotes al final del mes o cada quincena.
- **Generación de informes periódicos:** Organizaciones que necesitan generar informes semanales o mensuales suelen utilizar el procesamiento por lotes para agregar y analizar los datos.

1.2.2. Stream processing (Procesamiento en tiempo real)

El **Stream processing**, o procesamiento en tiempo real, es un método en el que los datos se procesan inmediatamente a medida que se generan. Este enfoque es adecuado para escenarios donde la rapidez en la toma de decisiones es crucial y donde se necesita una respuesta inmediata a los eventos que ocurren.

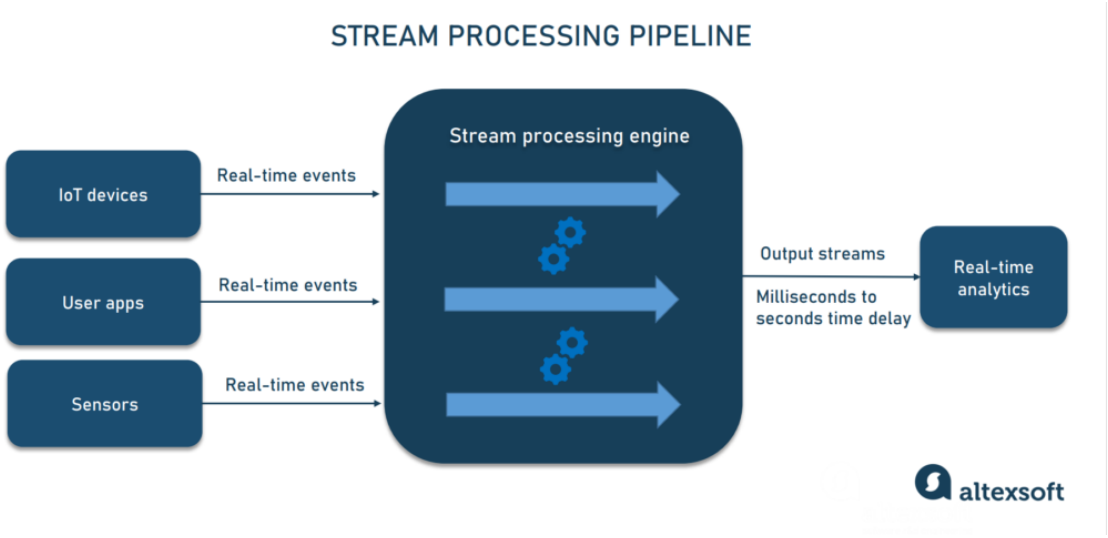

Figura 6. Stream processing

Fuente: altensoft.com

Características:

- **Baja latencia:** El procesamiento en tiempo real ofrece baja latencia, lo que significa que los datos son procesados casi instantáneamente después de ser generados.
- **Flujo continuo de datos:** Los datos se procesan como un flujo continuo, lo que permite a las organizaciones reaccionar a los eventos en tiempo real.
- **Infraestructura compleja:** A diferencia del procesamiento por lotes, el procesamiento en tiempo real requiere una infraestructura más compleja para manejar la alta disponibilidad y la escalabilidad.

- **Escalabilidad horizontal:** Los sistemas de procesamiento en tiempo real suelen estar diseñados para escalar horizontalmente, añadiendo más nodos al clúster para manejar mayores flujos de datos.

Casos de uso:

- **Monitoreo de sensores IoT:** En industrias como la manufactura o la agricultura, los sensores IoT pueden generar datos en tiempo real que deben ser monitoreados para detectar anomalías o condiciones específicas.
- **Procesamiento de transacciones financieras:** Los sistemas de trading o pagos electrónicos requieren procesamiento en tiempo real para detectar fraudes o gestionar transacciones instantáneamente.
- **Analítica de redes sociales:** Las empresas pueden procesar menciones de su marca en redes sociales en tiempo real para responder rápidamente a los comentarios de los clientes o gestionar crisis de reputación.

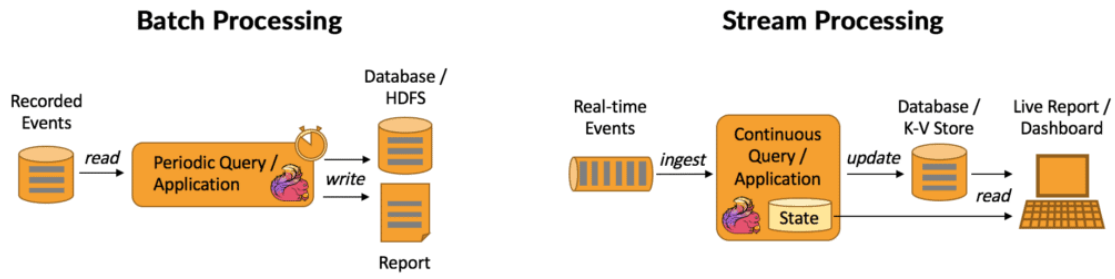

Figura 7. Batch processing vs. Stream processing

Fuente: k21academy.com

1.2.3. Ingestión Híbrida

La **ingestión híbrida** combina lo mejor de ambos mundos, permitiendo a las organizaciones procesar datos tanto en tiempo real como en lotes según sea necesario. En este enfoque, ciertos tipos de datos o eventos críticos se procesan en tiempo real, mientras que otros se almacenan y se procesan en lotes más grandes y menos frecuentes.

Características:

- **Flexibilidad:** Permite a las organizaciones elegir qué datos procesar en tiempo real y cuáles en lotes, según las necesidades del negocio.
- **Optimización de recursos:** El enfoque híbrido permite utilizar los recursos de manera más eficiente, procesando solo los datos críticos en tiempo real y dejando otros datos para procesamiento por lotes.
- **Mejor toma de decisiones:** Proporciona una visión integral del sistema al combinar datos históricos procesados por lotes con datos actuales procesados en tiempo real.

Casos de uso:

- **Monitoreo de sistemas informáticos:** Un centro de datos podría utilizar procesamiento en tiempo real para alertas de seguridad y fallos críticos, mientras que el procesamiento por lotes podría usarse para el análisis de rendimiento semanal.
- **Análisis de tráfico web:** Un sitio web puede procesar en tiempo real las visitas y acciones del usuario para personalizar la experiencia en tiempo real, mientras que los datos de clics y conversiones se procesan en lotes para análisis de marketing más detallado.

1.3 Desafíos en la ingestión de datos

La ingestión de datos en Big Data es un proceso complejo que enfrenta varios desafíos inherentes debido a la naturaleza masiva y diversa de los datos involucrados. Abordar estos desafíos es crucial para asegurar que los datos ingresados en el sistema sean útiles, precisos y estén disponibles para análisis en el momento adecuado. A continuación, se describen tres de los principales desafíos que se deben considerar al diseñar y gestionar pipelines de ingestión de datos en un entorno de Big Data.

1.3.1. Manejo de grandes volúmenes de datos

Uno de los desafíos más significativos en la ingestión de datos en entornos de Big Data es la capacidad de manejar grandes volúmenes de datos de manera eficiente y efectiva. Los sistemas de Big Data suelen trabajar con cantidades enormes de datos que pueden ir desde terabytes hasta petabytes, generados de manera continua o en ráfagas. Estos volúmenes imponen una carga considerable sobre la infraestructura, tanto en términos de almacenamiento como de procesamiento.

Aspectos a considerar:

- **Escalabilidad:** El sistema de ingestión debe ser capaz de escalar horizontalmente, es decir, añadir más nodos al clúster para manejar un mayor volumen de datos sin disminuir el rendimiento.
- **Velocidad de ingestión:** Es necesario garantizar que el sistema pueda ingerir los datos tan rápido como se generan, especialmente en aplicaciones donde la latencia debe ser mínima.
- **Costos:** Manejar grandes volúmenes de datos también puede implicar costos elevados en términos de almacenamiento y computación. Es fundamental optimizar los recursos para minimizar estos costos.

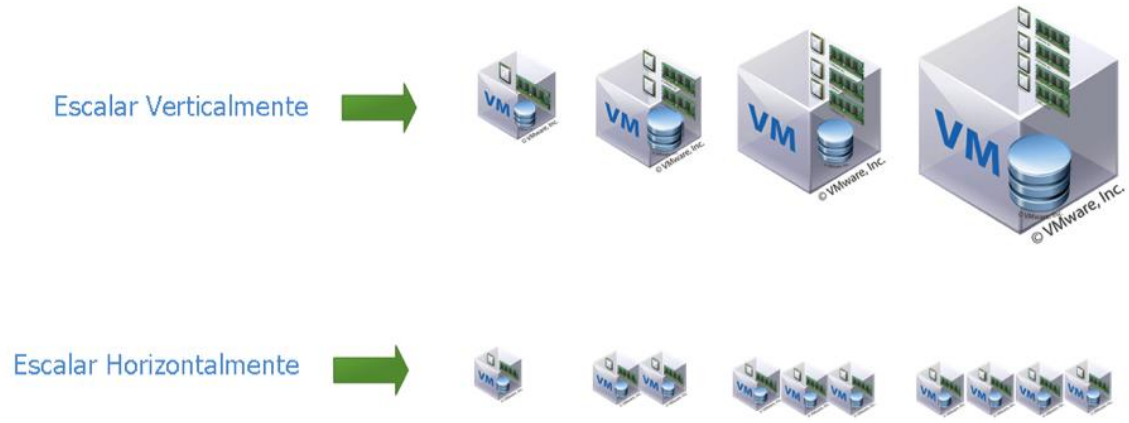

Figura 8. Tipos de escalado

Fuente: blogs.vmware.com

Estrategias para superar el desafío:

- **Uso de sistemas distribuidos:** Implementar sistemas de almacenamiento y procesamiento distribuidos como HDFS (Hadoop Distributed File System) que permiten manejar grandes volúmenes de datos distribuyendo la carga entre múltiples nodos.

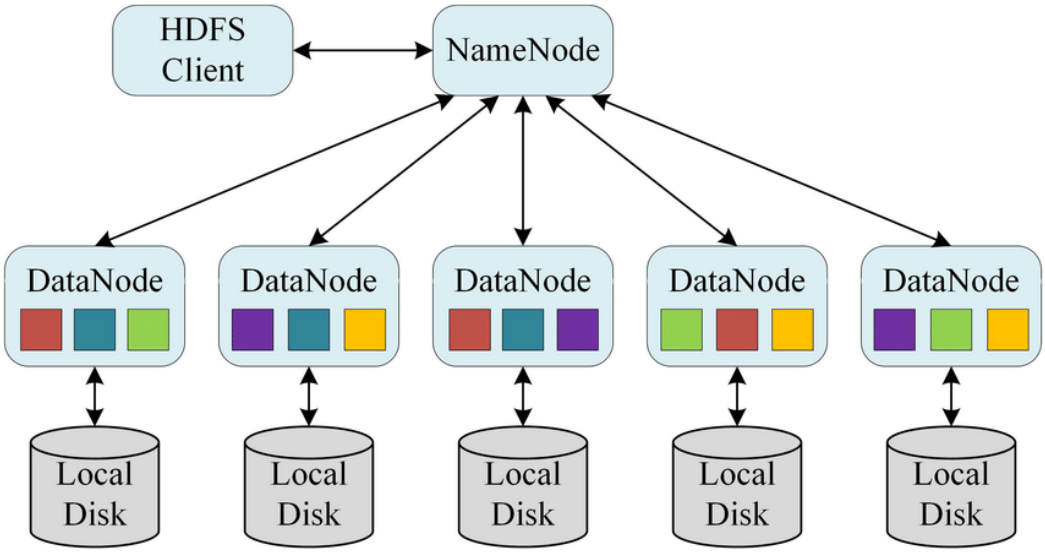

Figura 9. Sistemas distribuidos – HDFS

Fuente: researchgate.net

- **Compresión de datos:** Utilizar técnicas de compresión para reducir el tamaño de los datos y minimizar el uso de almacenamiento y ancho de banda durante la ingestión.

- **Pipeline optimizado:** Diseñar pipelines de ingestión que puedan manejar la entrada de datos de manera paralela y asíncrona, asegurando que el sistema no se sature bajo carga pesada.

1.3.2. Integración de fuentes de datos heterogéneas

Otro desafío clave en la ingestión de datos es la integración de fuentes de datos heterogéneas. En un entorno de Big Data, los datos provienen de múltiples fuentes con diferentes formatos, estructuras y frecuencias de generación. Esto incluye datos estructurados (como bases de datos relacionales), semiestructurados (como archivos JSON o XML) y no estructurados (como imágenes, videos o texto libre).

Figura 10. Tipos de datos

Fuente: es.slideshare.net

Aspectos a considerar:

- **Compatibilidad de formatos:** La necesidad de manejar diferentes formatos de datos y convertirlos a un formato común o compatible para su análisis.

- **Frecuencia de datos:** Algunas fuentes generan datos de manera continua, mientras que otras pueden hacerlo a intervalos regulares o esporádicamente, lo que complica la sincronización de la ingestión.

- **Semántica de los datos:** Diferencias en cómo los datos son organizados y definidos entre las fuentes pueden llevar a inconsistencias si no se manejan adecuadamente durante la ingestión.

Estrategias para superar el desafío:

- **Uso de herramientas de ingestión flexible:** Herramientas como Apache NiFi o Apache Kafka pueden manejar datos de múltiples fuentes y formatos, permitiendo una integración más sencilla.

- **Normalización y estándares de datos:** Implementar un proceso de normalización que estandarice los datos a un formato común antes de que se almacenen en el sistema.

    - **Propósito:** El objetivo de la **normalización** es asegurar que los datos de diferentes fuentes o sistemas se ajusten a un **formato común y estandarizado** antes de ser almacenados o procesados.

    - **Enfoque:** En este proceso, la tarea principal es convertir los datos para que cumplan con ciertos **estándares predefinidos**. Por ejemplo, si tienes datos de fechas en diferentes formatos (dd/mm/yyyy, mm/dd/yyyy, etc.), los transformarías todos a un único formato común (por ejemplo, ISO 8601: yyyy-mm-dd).

    - **Resultado final:** Un conjunto de datos **uniformes** y **consistentes**, independientemente de su origen, para que se integren fácilmente en los sistemas de almacenamiento o procesamiento.

- **Data Wrangling y transformación:** Utilizar herramientas de transformación de datos (como Apache Spark) para limpiar y organizar los datos de diferentes fuentes antes de almacenarlos.

- **Propósito: Data Wrangling** (también llamado **preparación de datos**) implica **limpiar, transformar y reorganizar** los datos para que sean más utilizables en análisis o almacenamiento. Implica más que solo la normalización; puede incluir operaciones como la **limpieza de datos**, la **eliminación de duplicados**, la **corrección de errores**, la **integración de múltiples fuentes de datos**, y la **transformación de formatos**.

- **Enfoque:** Aquí, el foco está en preparar los datos para un **análisis** o **almacenamiento posterior**. Herramientas como **Apache Spark** se utilizan para manejar grandes volúmenes de datos y realizar operaciones complejas como la transformación y limpieza.

- **Resultado final:** Un conjunto de datos que ha sido **limpiado**, **transformado**, y posiblemente **reorganizado**, para hacerlo más adecuado para el análisis y uso inmediato en procesos de Machine Learning, análisis de datos o almacenamiento eficiente.

Por lo anterior, tras analizar las estrategias anteriores, resaltaremos los siguientes aspectos:

- **Normalización:** Se enfoca en asegurar que los datos estén en un formato común y sigan ciertos estándares específicos. Esto es más un proceso de **estandarización**.

- **Data Wrangling y transformación:** Es un proceso más amplio que abarca la **limpieza**, **transformación** y **reorganización** de datos de diferentes fuentes, preparándolos para su análisis o almacenamiento. Incluye normalización, pero también involucra otras tareas, como la **eliminación de errores** o la **integración de datos** de diversas fuentes.

1.3.3. Garantía de calidad y consistencia de los datos

La calidad y consistencia de los datos son fundamentales para garantizar que los análisis y decisiones basados en esos datos sean precisos y confiables. La ingestión de datos en un entorno de Big Data presenta desafíos específicos en términos de asegurar que los datos sean completos, precisos, y estén libres de errores o duplicados.

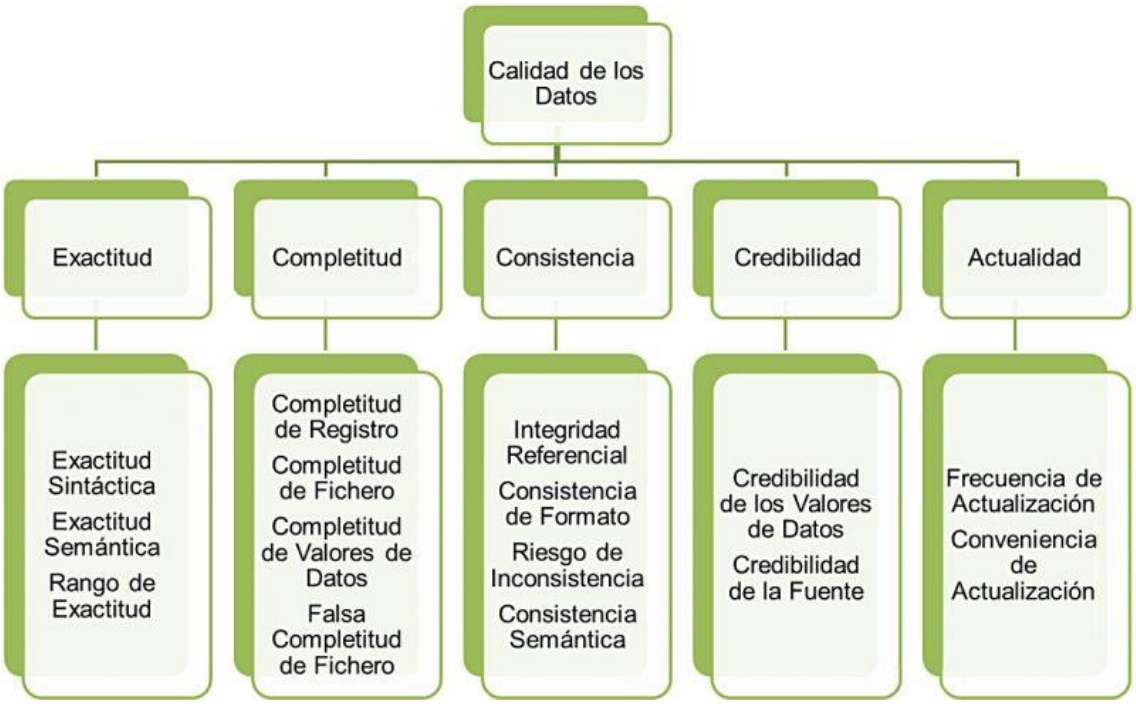

Figura 11. Calidad de los datos

Fuente: revista.aenor.com

Aspectos a considerar:

- **Calidad de los datos:** Incluye la exactitud, completitud, validez, y puntualidad de los datos ingeridos.
- **Consistencia de los datos:** Asegurar que los datos no estén duplicados o conflictivos, especialmente cuando se ingieren desde múltiples fuentes.
- **Data Governance:** La gobernanza de datos es clave para definir y aplicar políticas de calidad de datos que aseguren que todos los datos ingresados cumplan con los estándares requeridos.

Figura 12. Gobierno del dato

Fuente: ifgeekthen.nttdata.com

Estrategias para superar el desafío:

- **Validación de datos en el punto de ingestión:** Implementar reglas de validación en el pipeline de ingestión para detectar y corregir errores en los datos antes de que sean almacenados.
- **Deduplicación y limpieza de datos:** Utilizar técnicas de deduplicación para eliminar entradas de datos redundantes y aplicar procesos de limpieza para corregir o eliminar datos erróneos o incompletos.
- **Monitorización de calidad de datos:** Implementar herramientas de monitorización que alerten sobre posibles problemas de calidad o consistencia en los datos durante la ingestión.

2. Herramientas y mecanismos para la ingestión de datos

La selección de herramientas adecuadas para la ingestión de datos es crucial en cualquier estrategia de Big Data, ya que influye directamente en la eficiencia y la eficacia con la que se procesan grandes volúmenes de datos de diversas fuentes. Existen múltiples herramientas diseñadas para facilitar la ingestión de datos, cada una con funcionalidades específicas que las hacen más adecuadas para ciertos tipos de flujos de trabajo. A continuación, se presenta una visión general de las principales herramientas utilizadas en Big Data para la ingestión de datos, seguida de una exploración detallada de algunas de las herramientas más relevantes como Apache NiFi, Apache Kafka, y Apache Flume.

2.1 Visión general de herramientas de ingestión

Las herramientas de ingestión de datos son esenciales para capturar, transformar y cargar datos desde múltiples fuentes en sistemas de almacenamiento centralizados. Estas herramientas permiten gestionar grandes volúmenes de datos, procesar datos en tiempo real, y asegurar que los datos estén disponibles para análisis y otras aplicaciones de Big Data. Algunas de las herramientas más comunes incluyen:

- **Apache NiFi:** Una herramienta flexible y poderosa para el flujo de datos que facilita la automatización y gestión de la ingestión de datos desde múltiples fuentes.
- **Apache Kafka:** Un sistema de mensajería distribuido que permite la ingestión de datos en tiempo real, con un enfoque en la transmisión de datos de alta velocidad y baja latencia.
- **Apache Flume:** Una herramienta especializada en la ingestión de datos de logs, diseñada para transportar grandes volúmenes de datos desde múltiples fuentes hacia un sistema de almacenamiento centralizado.
- **Logstash:** Parte de la pila ELK (Elasticsearch, Logstash, Kibana), se utiliza para la ingestión, transformación y almacenamiento de datos en tiempo real.

- **AWS Kinesis:** Un servicio de Amazon Web Services que permite la ingestión y el procesamiento en tiempo real de grandes flujos de datos.

2.1.1. Criterios para seleccionar herramientas de ingestión

Al seleccionar una herramienta de ingestión de datos, es fundamental considerar varios factores que determinarán su idoneidad para un caso de uso específico. Estos criterios incluyen:

- **Volumen de datos:** La capacidad de la herramienta para manejar grandes volúmenes de datos es crucial. Por ejemplo, Kafka es ideal para flujos de datos de alta velocidad, mientras que NiFi es más adecuado para flujos de datos de múltiples fuentes heterogéneas.

- **Velocidad de ingestión:** Dependiendo de si los datos necesitan ser procesados en tiempo real o en lotes, se elegirán diferentes herramientas. Kafka es excelente para tiempo real, mientras que herramientas como NiFi pueden manejar ambos.

- **Integración con otros sistemas:** La facilidad con la que la herramienta se integra con otros componentes del ecosistema de Big Data, como HDFS, S3, o bases de datos relacionales, es clave. NiFi, por ejemplo, es altamente integrable con una amplia gama de sistemas.

- **Facilidad de uso y configuración:** Algunas herramientas, como NiFi, ofrecen interfaces gráficas que facilitan la configuración y gestión de los flujos de datos, mientras que otras, como Kafka, requieren un mayor conocimiento técnico.

- **Escalabilidad:** Es importante que la herramienta pueda escalar horizontalmente para manejar el aumento en el volumen y velocidad de los datos sin comprometer el rendimiento.

2.2 Apache NiFi

Apache NiFi es una herramienta diseñada para automatizar el flujo de datos entre sistemas distribuidos. Es especialmente poderosa para la integración de fuentes de datos heterogéneas y para la creación de pipelines de datos complejos que incluyen filtrado, transformación y enruteo.

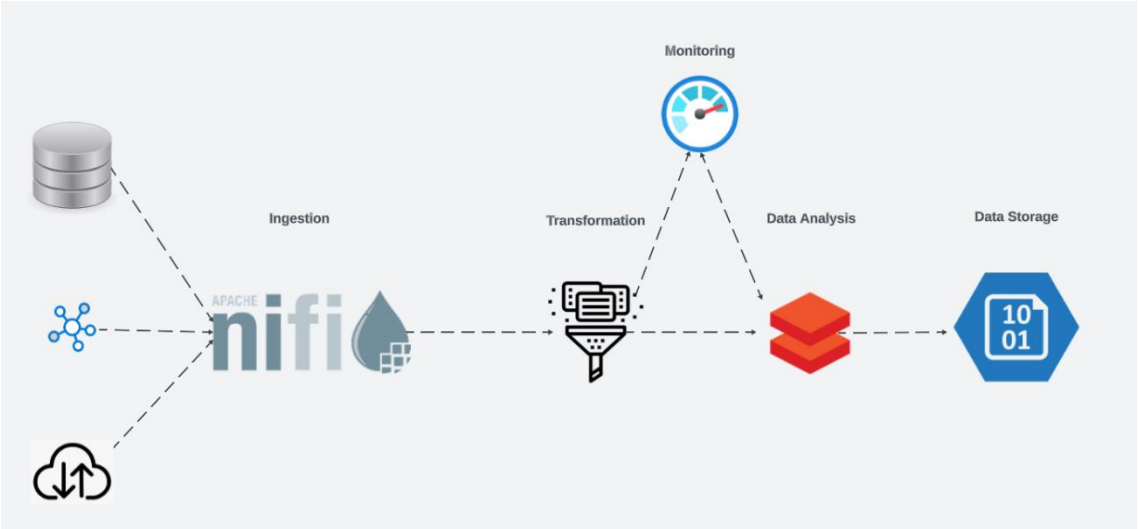

Figura 13. Apache NiFi

Fuente: miro.medium.com

Funcionalidades clave:

- **Flujo de datos visual:** NiFi permite la creación de flujos de datos a través de una interfaz gráfica de usuario (GUI), facilitando el diseño y la gestión de pipelines sin necesidad de escribir código.
- **Integración de múltiples fuentes:** Soporta una amplia gama de fuentes de datos, incluyendo archivos locales, bases de datos, APIs REST, servicios en la nube, y más.
- **Transformación y enruteo de datos:** NiFi permite la transformación de datos en tiempo real mediante operaciones de filtrado, agregación y manipulación de datos.

- **Control de Flujo:** Ofrece capacidades avanzadas de control de flujo de datos, como el backpressure, que permite regular el flujo de datos para evitar la sobrecarga del sistema.

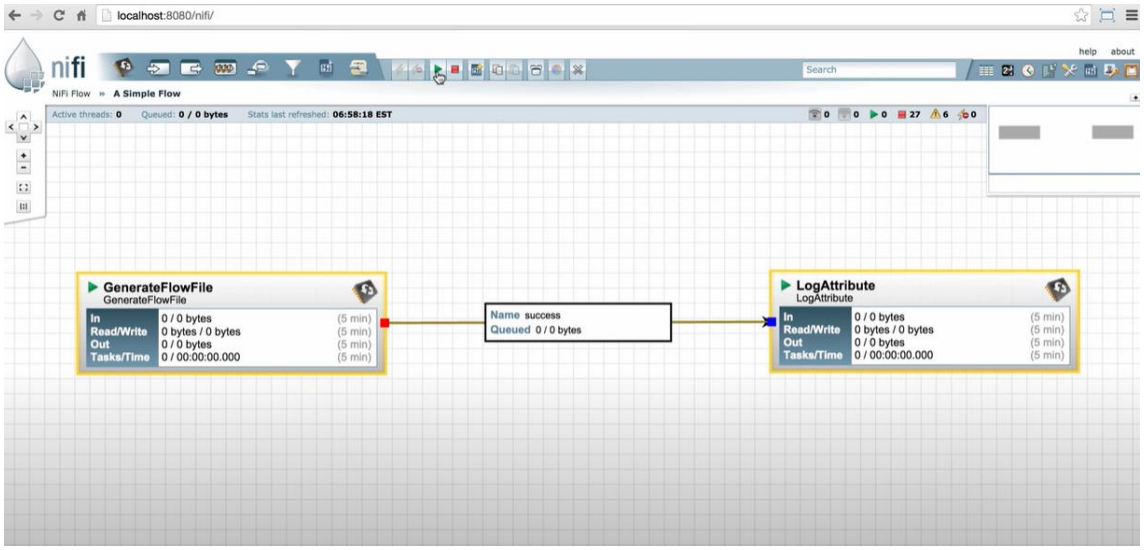

Figura 14. Creación de un flujo de datos

Fuente: todobi.com

2.3 Apache Kafka

Apache Kafka es una plataforma de transmisión de datos en tiempo real que facilita la ingestión de grandes volúmenes de datos con baja latencia. Kafka se basa en el concepto de "streams" de datos, donde los datos se publican como mensajes en un tema (topic) y son consumidos por diferentes aplicaciones.

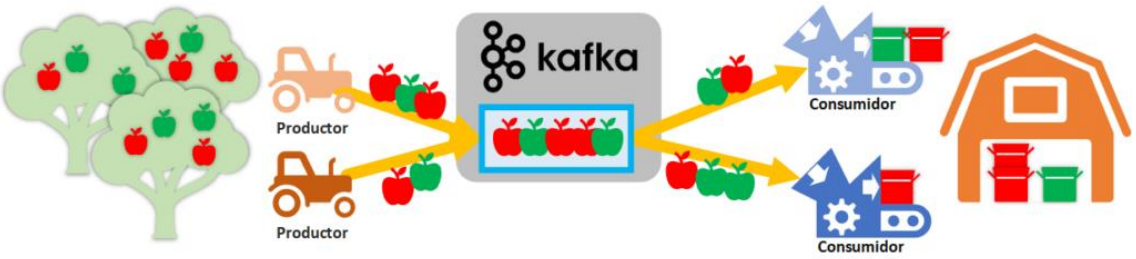

Figura 15. Apache Kafka

Fuente: jarroba.com

Conceptos fundamentales:

- **Mensajes:** Un mensaje es la unidad de datos que se publica y se consume en Kafka. Cada mensaje se almacena con una clave opcional y un valor.
- **Temas:** Un tema es un canal de comunicación en Kafka donde los productores publican mensajes y los consumidores los suscriben.
- **Brokers:** Un broker es un servidor en el clúster de Kafka que almacena los datos y sirve las solicitudes de los clientes. Un clúster de Kafka puede consistir en múltiples brokers para distribuir la carga.

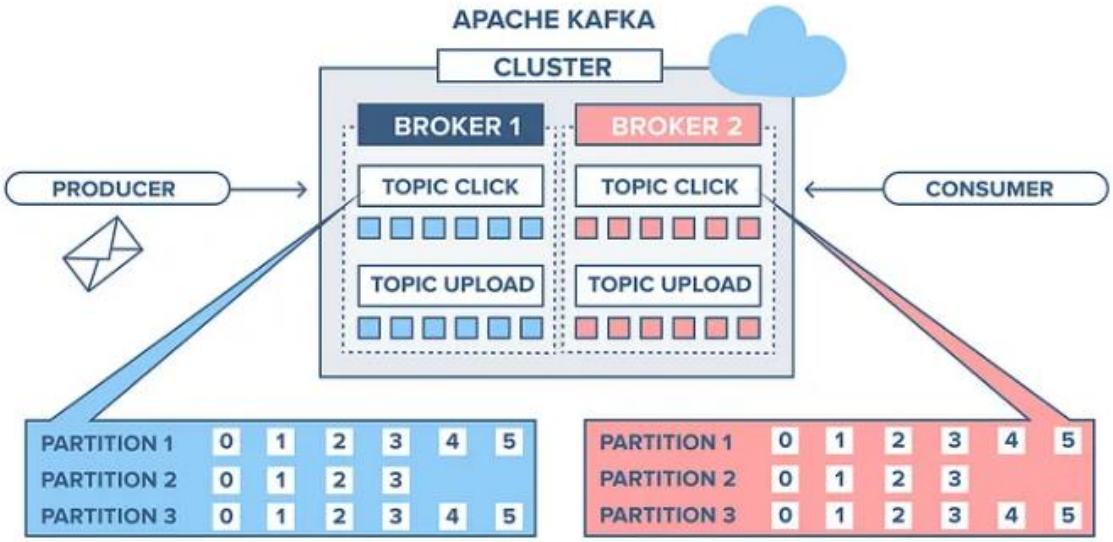

Figura 16. Arquitectura de Apache Kafka

Fuente: médium.com

2.3.1. Ingestión en tiempo real con Kafka

Kafka es ideal para la ingestión de datos en tiempo real debido a su capacidad para manejar flujos de datos de alta velocidad y baja latencia.

Características de Kafka:

- **Alto rendimiento:** Kafka está optimizado para manejar millones de mensajes por segundo.

- **Persistencia y replicación:** Los mensajes en Kafka se almacenan en el disco y se replican entre brokers para asegurar la durabilidad y disponibilidad.
- **Escalabilidad:** Kafka puede escalar horizontalmente añadiendo más brokers al clúster.

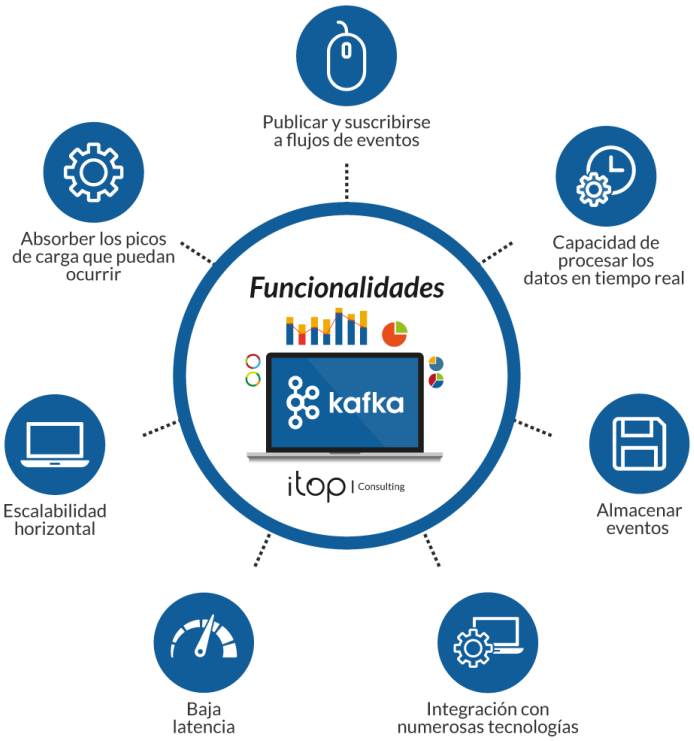

Figura 17. Características de Apache Kafka

Fuente: itop.es

Casos de Uso:

- **Monitoreo de infraestructura:** Kafka puede ingerir logs y métricas en tiempo real desde múltiples servidores para el monitoreo continuo.
- **Analítica de Clickstream:** Empresas de comercio electrónico utilizan Kafka para capturar y procesar datos de clickstream en tiempo real, proporcionando insights sobre el comportamiento del usuario.

2.4 Apache Flume

Apache Flume es una herramienta específica para la ingestión de grandes volúmenes de datos de logs. Está diseñado para recolectar, agregar y mover datos de múltiples fuentes hacia un destino centralizado.

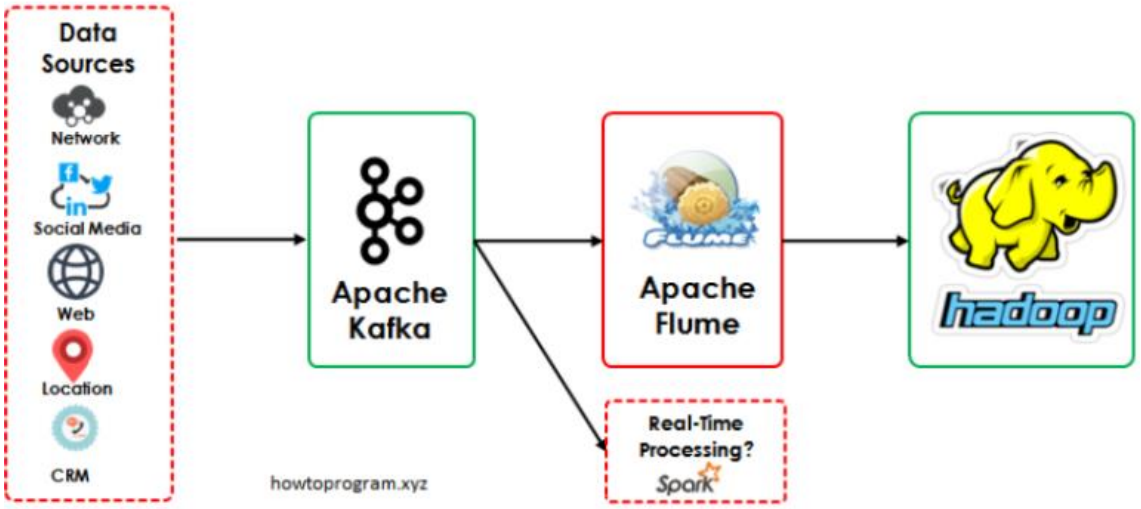

Figura 18. Apache Flume con Apache Kafka

Fuente: keepcoding.io

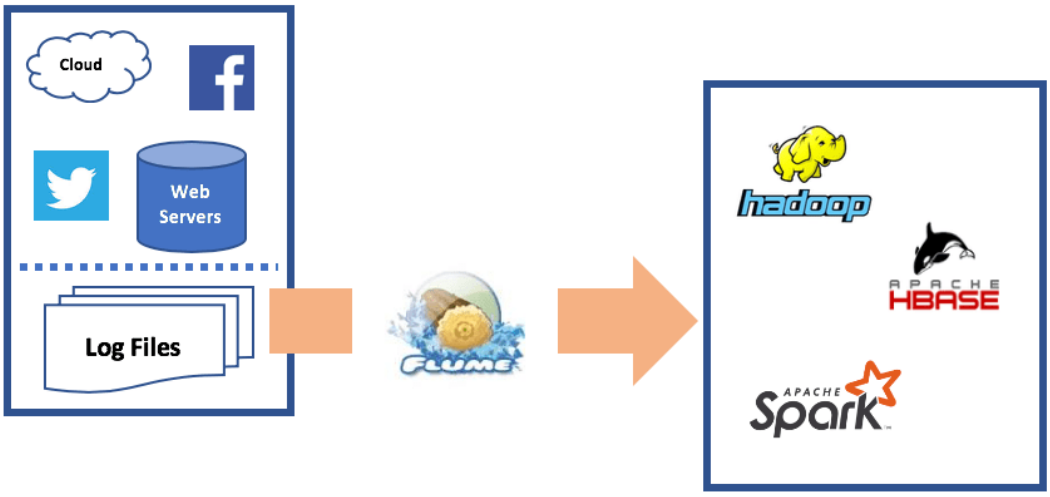

Figura 19. Apache Flume

Fuente: logz.io

Características de Flume:

- **Ingestión escalable:** Flume puede manejar la ingestión de grandes volúmenes de logs generados en un entorno distribuido.
- **Arquitectura simple y robusta:** Basado en una arquitectura de agente que facilita la recolección y transmisión de datos.
- **Manejo de Fallos:** Incluye mecanismos para manejar fallos de red y congestión de datos sin pérdida de datos.

Casos de uso:

- **Ingestión de logs de servidores web:** Flume es comúnmente utilizado para recoger y mover logs de servidores web hacia un almacén de datos o un lago de datos para análisis posterior.
- **Monitoreo de aplicaciones:** Flume puede integrar los logs de aplicaciones distribuidas en un sistema centralizado de monitoreo.

2.4.1. Arquitectura de Flume: agentes, fuentes, canales, sumideros

La arquitectura de Flume está basada en agentes que facilitan la transferencia de datos desde la fuente hasta el destino final.

Componentes de Flume:

- **Agentes:** Son los procesos que ejecutan Flume. Un agente puede tener múltiples fuentes, canales y sumideros.
- **Fuentes:** Son los puntos de entrada de los datos en un agente. Ejemplos de fuentes incluyen archivos de logs, syslog, o JDBC.
- **Canales:** Son las vías a través de las cuales los datos se transfieren dentro del agente, actuando como buffers temporales.
- **Sumideros (Sinks):** Son los destinos finales donde se almacenan o procesan los datos, como HDFS, HBase, o ElasticSearch.

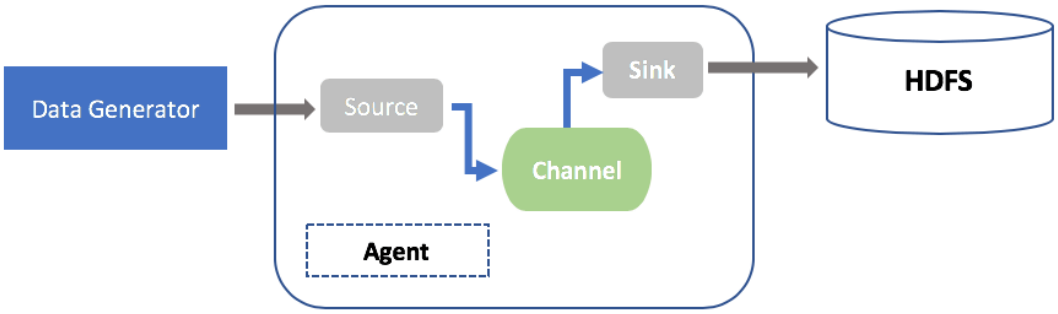

Figura 20. Componentes de Apache Flume

Fuente: logz.io

2.5 Integración con sistemas de almacenamiento

Una vez que los datos son ingeridos, deben ser almacenados de manera eficiente para permitir su procesamiento y análisis posterior. Las herramientas de ingestión de datos se integran con diferentes sistemas de almacenamiento según las necesidades del negocio.

Sistemas de almacenamiento comunes:

- **HDFS (Hadoop Distributed File System):** Un sistema de archivos distribuido que permite el almacenamiento escalable y distribuido de grandes volúmenes de datos.
- **Amazon S3:** Un servicio de almacenamiento en la nube que permite almacenar y recuperar cualquier cantidad de datos en cualquier momento.
- **Bases de datos NoSQL:** Bases de datos como HBase o Cassandra que están diseñadas para manejar grandes volúmenes de datos estructurados y no estructurados.

Integración:

- Las herramientas como Apache NiFi, Kafka, y Flume tienen conectores nativos que permiten integrar estos flujos de datos directamente con HDFS, S3, y otros sistemas de almacenamiento.

- Se pueden configurar pipelines para que los datos ingresados en tiempo real o por lotes se almacenen automáticamente en el sistema de almacenamiento deseado.

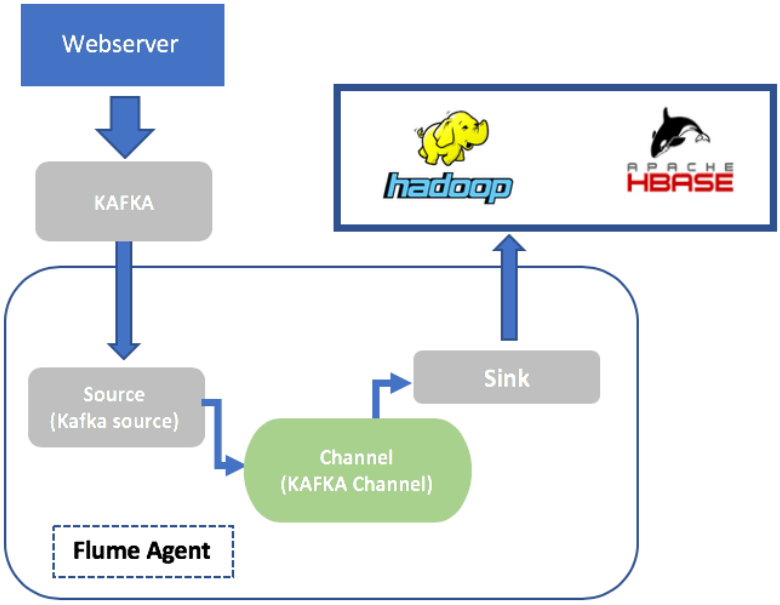

Figura 21. Integración Kafka, Flume y HDFS

Fuente: logz.io

3. Formatos de datos para almacenamiento en Big Data

En el contexto de Big Data, la elección del formato de almacenamiento de datos es crucial, ya que impacta directamente en la eficiencia del procesamiento, el rendimiento de las consultas, y la capacidad de manejo de grandes volúmenes de datos. Cada formato de datos tiene sus propias características, ventajas y limitaciones, lo que hace esencial seleccionar el formato adecuado para cada caso de uso específico. A continuación, se exploran en detalle los principales formatos de datos utilizados en Big Data, su importancia, y cómo elegir el más adecuado para diferentes escenarios.

3.1 Importancia de los formatos de datos en Big Data

3.1.1. Impacto del formato en el almacenamiento y procesamiento

El formato en el que se almacenan los datos tiene un impacto significativo en varios aspectos del manejo de Big Data:

- **Rendimiento de consultas:** Algunos formatos están optimizados para consultas rápidas y eficientes, lo que es crucial en escenarios donde la velocidad de acceso a los datos es importante. Por ejemplo, los formatos columnares como Parquet y ORC permiten realizar consultas más rápidas en grandes volúmenes de datos.

- **Uso de almacenamiento:** Los diferentes formatos de datos varían en su eficiencia de almacenamiento. Formatos como Avro y Parquet suelen ser más compactos debido a su capacidad de compresión, lo que reduce el espacio en disco necesario para almacenar grandes conjuntos de datos.

- **Velocidad de procesamiento:** El formato de los datos afecta la velocidad con la que se pueden procesar y analizar. Los formatos optimizados, como los formatos binarios y columnar, permiten un procesamiento más rápido, especialmente en operaciones de lectura intensiva.

- **Compatibilidad y portabilidad:** Algunos formatos, como JSON y CSV, son más fáciles de compartir entre diferentes sistemas debido a su simplicidad y

compatibilidad universal, aunque pueden no ser tan eficientes en términos de almacenamiento y procesamiento.

3.1.2. Consideraciones para elegir el formato adecuado

Al seleccionar un formato de datos para almacenamiento en Big Data, es esencial considerar varios factores:

- **Tipo de datos:** Si los datos son estructurados, semiestructurados o no estructurados, influye en la elección del formato. Por ejemplo, JSON es ideal para datos semiestructurados, mientras que Parquet es preferible para datos tabulares estructurados.
- **Requisitos de consultas:** Si se anticipa que se realizarán consultas frecuentes y complejas, los formatos columnares como Parquet o ORC son más adecuados.
- **Tamaño del conjunto de datos:** Para grandes volúmenes de datos, los formatos que soportan compresión, como Avro y Parquet, pueden reducir significativamente los requisitos de almacenamiento.
- **Compatibilidad con herramientas:** La integración con las herramientas y sistemas existentes también es un factor importante. Por ejemplo, CSV es ampliamente soportado por la mayoría de las herramientas, pero puede no ser el más eficiente para grandes volúmenes de datos.

3.2 Formatos de texto plano

3.2.1. CSV, JSON, XML

Los formatos de texto plano, como CSV, JSON y XML, son ampliamente utilizados debido a su simplicidad y compatibilidad universal. Sin embargo, también presentan ciertas limitaciones que deben tenerse en cuenta.

CSV (Comma-Separated Values):

- **Características:** Es uno de los formatos más simples y comunes para almacenar datos tabulares. Cada línea en un archivo CSV representa un registro, y los valores dentro de cada registro están separados por comas.

- **Usos:** Ideal para datos tabulares pequeños a medianos que requieren una estructura mínima. Es ampliamente utilizado en transferencias de datos entre sistemas diferentes.

- **Limitaciones:** No es adecuado para datos complejos o jerárquicos, y carece de capacidades para manejar grandes volúmenes de datos eficientemente. Además, carece de metadatos integrados para describir los datos.

```
Date,Weekday,Region,Employee,Item, Units , Unit Cost , Total
15-Dec-21,Wednesday,Central,Jones,Pen Set,700, $1.99 ," $1,393.00 "
16-Dec-21,Thursday,West,Kivell,Binder,85, $19.99 ," $1,699.15 "
17-Dec-21,Friday,Central,Howard,Pen & Pencil,62, $4.99 , $309.38
18-Dec-21,Saturday,East,Gill,Pen,58, $19.99 ," $1,159.42 "
19-Dec-21,Sunday,East,Anderson,Binder,10, $4.99 , $49.90
20-Dec-21,Monday,East,Anderson,Pen Set,19, $2.99 , $56.81
21-Dec-21,Tuesday,East,Anderson,Pen Set,6, $1.99 , $11.94
22-Dec-21,Wednesday,Central,Howard,Pen & Pencil,10, $4.99 , $49.90
23-Dec-21,Thursday,West,Wilson,Paper,39, $1.99 , $77.61
24-Dec-21,Friday,West,Wilson,Binder,1, $8.99 , $8.99
25-Dec-21,Saturday,West,Wilson,Pen & Pencil,80, $4.99 , $399.20
26-Dec-21,Sunday,West,Wilson,Binder,51, $1.99 , $101.49
27-Dec-21,Monday,West,Wilson,Binder,10, $19.99 , $199.90
28-Dec-21,Tuesday,West,Wilson,Pen Set,15, $4.99 , $74.85
29-Dec-21,Wednesday,West,Wilson,Desk,31, $125.00 ," $3,875.00 "
30-Dec-21,Thursday,Central,Jones,Pen Set,46, $15.99 , $735.54
31-Dec-21,Friday,West,Kivell,Binder,61, $8.99 , $548.39
1-Jan-22,Saturday,Central,Jones,Pen,90, $8.99 , $809.10
```

Figura 22. CSV

Fuente: syncfusion.com

JSON (JavaScript Object Notation):

- **Características:** Es un formato semiestructurado que utiliza una sintaxis basada en texto legible por humanos para representar objetos y arrays. JSON es ideal para datos jerárquicos.
- **Usos:** Comúnmente utilizado para API web, transmisión de datos entre servidores y aplicaciones web, y almacenamiento de configuraciones.
- **Limitaciones:** Aunque es más flexible que CSV, JSON puede volverse ineficiente para grandes volúmenes de datos debido a su naturaleza verbosa y a la falta de capacidades de compresión.


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
```


Figura 23. JSON

Fuente: help.apple.com

XML (eXtensible Markup Language):

- **Características:** Similar a JSON, XML es un formato basado en texto que utiliza etiquetas para definir la estructura de los datos. Es ampliamente utilizado en aplicaciones que requieren interoperabilidad entre diferentes sistemas.
- **Usos:** Ideal para documentos donde la estructura y el formato son importantes, como en servicios web SOAP.
- **Limitaciones:** XML es más verboso que JSON y CSV, lo que lo hace menos eficiente en términos de almacenamiento y procesamiento, especialmente para grandes volúmenes de datos.

Figura 24. XML vs. JSON

Fuente: dongee.com

3.2.2. Ventajas y Limitaciones de los Formatos de Texto Plano

Ventajas:

- **Simplicidad y compatibilidad:** Los formatos de texto plano son fáciles de leer, escribir y compartir, y son compatibles con la mayoría de las herramientas y sistemas.
- **Portabilidad:** Debido a su naturaleza basada en texto, los formatos de texto plano son altamente portátiles y pueden ser fácilmente transferidos entre diferentes plataformas.

Limitaciones:

- **Eficiencia de almacenamiento:** Estos formatos suelen ser más grandes y menos eficientes en términos de espacio en disco, especialmente cuando se comparan con formatos binarios o columnar.
- **Rendimiento:** El rendimiento de las consultas y el procesamiento es generalmente más lento en comparación con formatos más optimizados, como Parquet o ORC.
- **Falta de metadatos:** A menudo carecen de capacidades para manejar metadatos, lo que puede complicar la gestión de datos a gran escala.

3.3. Formatos Columnares

Los formatos columnares, como Apache Parquet y ORC (Optimized Row Columnar), están diseñados para mejorar la eficiencia en el almacenamiento y procesamiento de grandes volúmenes de datos tabulares. A diferencia de los formatos de texto plano, los formatos columnares almacenan datos en columnas en lugar de filas, lo que permite optimizar las consultas y la compresión de datos.

3.3.1. Historia y desarrollo de Parquet y ORC

1. **Apache Parquet**:
    - **Desarrollado primero**: Parquet fue desarrollado en 2013 por **Twitter** y **Cloudera** en colaboración.
    - **Propósito**: Su objetivo fue crear un formato de almacenamiento columnar abierto para entornos de Big Data que ofreciera compresión y eficiencia de lectura para cargas de trabajo analíticas.
    - **Amplia adopción**: Se integró rápidamente con múltiples herramientas de procesamiento de Big Data como **Apache Spark**, **Apache Drill**, **Presto e Impala**, además de **Hadoop**.

2. **Apache ORC (Optimized Row Columnar)**:

    - **Desarrollado después de Parquet**: ORC fue creado por **Hortonworks** en 2013 como parte del proyecto **Apache Hive**.

    - **Propósito**: ORC fue diseñado para ser una alternativa más optimizada y eficiente dentro del ecosistema **Hive** y **Hadoop**, con el objetivo de mejorar la eficiencia de almacenamiento y consulta de grandes volúmenes de datos.

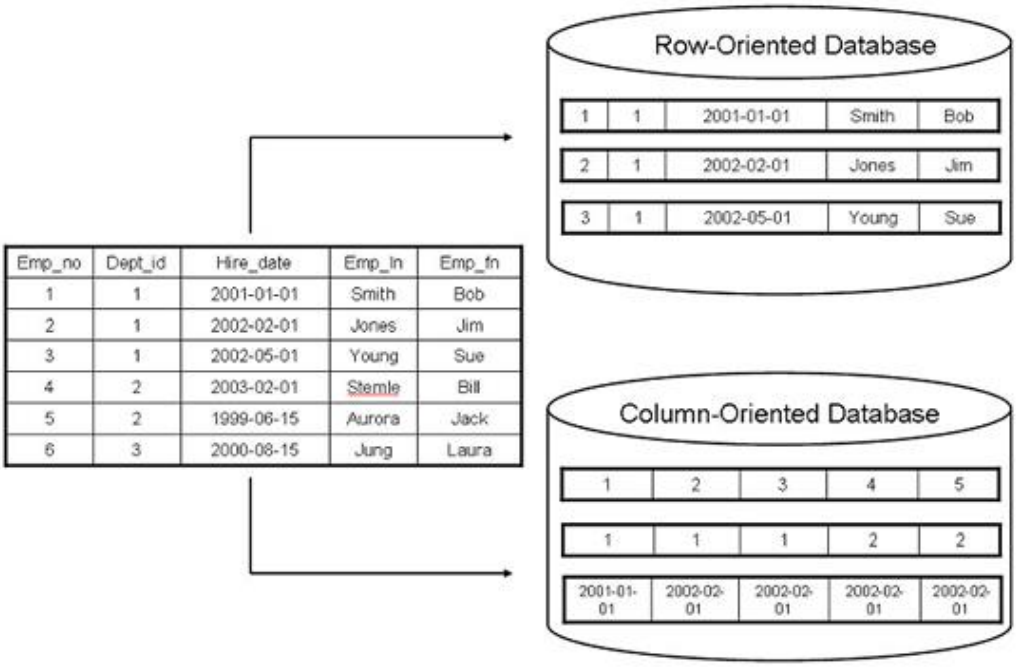

Figura 25. Orientado a filas vs. Orientado a columnas

Fuente: softwareengineeringdaily.com

3.3.2. Diseño y estructura de datos columnares

- **Parquet:** Es un formato de almacenamiento columnar de código abierto que soporta compresión y codificación eficiente de datos. Parquet está optimizado para cargas de trabajo de análisis de Big Data y es compatible con muchas herramientas de procesamiento, como Apache Spark, Hive y Presto.

- **ORC:** Similar a Parquet, ORC es un formato de almacenamiento columnar diseñado para cargas de trabajo de procesamiento de datos intensivo. Es especialmente optimizado para Apache Hive y proporciona una compresión más eficiente y capacidades avanzadas de análisis.

3.3.3. Ventajas de los formatos columnares

- **Optimización de consultas:** Al almacenar datos en columnas, estos formatos permiten la lectura selectiva de columnas, lo que reduce la cantidad de datos que deben ser leídos del disco y mejora el rendimiento de las consultas.
- **Compresión eficiente:** Los datos almacenados en columnas similares tienden a ser más repetitivos, lo que permite una compresión más efectiva y reduce el espacio de almacenamiento necesario.
- **Compatibilidad con herramientas de Big Data:** Parquet y ORC son ampliamente compatibles con herramientas de procesamiento de Big Data como Apache Spark, Hive e Impala, lo que facilita su integración en pipelines de datos.

Row storage	
Row 1	1
	US
	Free
Row 2	2
	UK
	Paid
Row 3	3
	ES
	Paid

Column storage	
user_id	1
	2
	3
country	US
	UK
	ES
subscription_type	Free
	Paid
	Paid

Figura 26. Optimización de consultas

Fuente: aitor-medrano.github.io

Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	$5.75
Data stored in Apache Parquet format*	**130 GB**	**6.78 seconds**	**2.51 GB**	**$0.01**
Savings / Speedup	**87% less with Parquet**	**34x faster**	**99% less data scanned**	**99.7% savings**

Figura 27. CSV vs. Parquet

Fuente: blog.openbridge.com

Dataset	Columns	Size on Amazon S3	Data Scanned	Cost
Data stored as CSV file	4	4TB	4TB	**$20** (4TB x $5/TB)
Data stored as GZIP CSV file	4	1TB	1TB	**$5** (1TB x $5/TB)
Data stored as Parquet file	4	1TB	**.25TB**	**$1.25** (.25TB x $5/TB)

Figura 28. Dataset en varios formatos

Fuente: blog.openbridge.com

3.3.4. Ejemplo de uso en un análisis de datos

Imagina que tienes un dataset enorme con millones de filas y cientos de columnas, pero solo te interesa hacer un análisis sobre el campo "edad" y "salario". Si los datos estuvieran en formato **CSV** o **JSON**, tendrías que leer todo el archivo para obtener estas columnas. Sin embargo, si están almacenados en formato **Parquet** o **ORC**, el sistema puede leer solo las columnas "edad" y "salario" de manera mucho más eficiente.

Ventajas clave del acceso columnar:

- **Consultas más rápidas:** Puedes leer solo las columnas necesarias.
- **Menor uso de recursos:** Se reduce la cantidad de datos que se deben mover desde el disco hacia la memoria.
- **Mayor compresión:** La compresión es más efectiva porque los datos de una columna tienden a ser más homogéneos. En una columna, todos los valores suelen ser del mismo tipo de datos (por ejemplo, una columna "Edad" solo contendrá números enteros, y una columna "Nombre" solo contendrá texto). Debido a que

los valores dentro de una columna tienden a ser más **similares** o **homogéneos** (es decir, son más parecidos entre sí), las técnicas de compresión son más eficientes.

3.4. Formato de fila

3.4.1. Avro

Avro es un formato binario diseñado para la serialización de datos, lo que lo hace ideal para la transmisión de datos entre sistemas y la persistencia de grandes volúmenes de datos.

```
{
  "type": "record",
  "name": "User",
  "fields": [
    {"name": "name", "type": "string"},
    {"name": "age", "type": "int"},
    {"name": "email", "type": ["null", "string"], "default": null}
  ]
}
```

Figura 29. Esquema Avro en JSON (incluido en el binario)

Fuente: researchgate.net

Características de Avro:

- **Formato binario:** Avro utiliza un formato binario compacto que permite la serialización eficiente de datos. Esto reduce significativamente el tamaño de los datos y mejora el rendimiento de la transmisión.
- **Esquema auto-descriptivo:** Los datos en Avro se almacenan junto con su esquema, lo que facilita la lectura y escritura de datos en diferentes sistemas sin necesidad de preacordar el esquema.

- **Compatibilidad con sistemas distribuidos:** Avro es ampliamente utilizado en sistemas distribuidos, como Apache Kafka y Hadoop, para la serialización y deserialización de datos.

Avro vs. Formatos de texto plano

Avro está diseñado para ser **más eficiente** que los formatos de texto plano, como **CSV** o **JSON**, al ofrecer las siguientes ventajas:

- **Compactación binaria:** Los datos se almacenan en binario, lo que los hace más pequeños y rápidos de transmitir.
- **Esquema embebido:** El esquema se almacena junto con los datos, lo que facilita la interoperabilidad y evolución de los datos, algo que los formatos de texto plano no proporcionan.
- **Manejo eficiente de grandes volúmenes:** Debido a su formato binario, Avro es más adecuado para manejar **grandes volúmenes de datos** y para trabajar en entornos distribuidos, donde la transmisión de datos rápida y compacta es crítica.

3.5. Comparativa de formatos

3.5.1. Análisis de rendimiento y eficiencia en diferentes escenarios

Al comparar diferentes formatos de datos, es esencial considerar el rendimiento y la eficiencia en escenarios específicos:

- **Rendimiento de consultas:** Los formatos columnares, como Parquet y ORC, suelen ofrecer un mejor rendimiento en consultas analíticas debido a su capacidad para leer solo las columnas necesarias.
- **Uso de almacenamiento:** Formatos como Avro y Parquet suelen ser más compactos debido a su capacidad de compresión, lo que los hace más eficientes en términos de uso de almacenamiento.

- **Facilidad de integración:** Formatos como CSV y JSON son más fáciles de integrar y transferir entre diferentes sistemas debido a su simplicidad, pero pueden no ser los más eficientes en términos de rendimiento.

Característica	**Parquet**	**ORC**	**Avro**
Estructura	**Columnar**	**Columnar**	**De Fila**
Uso Principal	Análisis de grandes volúmenes de datos	Optimización en Hadoop/Hive	Transmisión de datos, serialización
Lectura	Columnas específicas, eficiente. Es decir, más eficiente en lectura selectiva.	Más rápido en lectura secuencial cuando se trabaja con grandes conjuntos de datos en columnas	Lectura completa de registros
Compresión	Muy buena	Generalmente (para ciertos tipos de datos) mejor que Parquet	Compacto, pero sin foco en compresión
Casos de Uso	Big Data Analytics, Data Warehouses	Hadoop/Hive, Big Data	Kafka, Ingesta de Datos, Mensajería
Evolución de Esquema	Limitada	Limitada	Flexible, ideal para cambios

3.5.2. Estudio de casos: Selección del formato adecuado según el caso de uso

Caso de uso 1: Análisis de datos tabulares

- **Formato recomendado:** Parquet o ORC.
- **Razón:** Estos formatos están optimizados para datos tabulares y ofrecen un rendimiento superior en consultas analíticas, especialmente cuando se necesitan leer grandes volúmenes de datos.

Caso de uso 2: Transmisión de datos entre sistemas

- **Formato recomendado:** Avro.
- **Razón:** Avro es ideal para la serialización de datos y la transmisión entre sistemas distribuidos debido a su formato binario compacto y su soporte de esquema auto-descriptivo.

Caso de uso 3: Intercambio de datos entre aplicaciones

- **Formato recomendado:** JSON o CSV.
- **Razón:** Estos formatos son ampliamente soportados y fáciles de leer y escribir, lo que los hace adecuados para el intercambio de datos entre diferentes aplicaciones o servicios web.

4. Diseño de pipelines de ingestión

El diseño de pipelines de ingestión de datos es una parte fundamental en cualquier arquitectura de Big Data. Un pipeline de datos bien diseñado asegura que los datos se capturen, procesen y almacenen de manera eficiente y fiable, desde su origen hasta su destino final. A continuación, se exploran en detalle los conceptos, principios de diseño, herramientas de automatización, y estrategias de monitorización necesarias para construir pipelines de ingestión robustos y escalables.

4.1 Concepto de pipelines de ingestión

4.1.1. Definición y propósito de un pipeline de datos

Un **pipeline de datos** es un conjunto de procesos automatizados que facilitan el movimiento, procesamiento y almacenamiento de datos desde diversas fuentes hasta un destino final, como un almacén de datos, lago de datos, o sistema de análisis. El propósito principal de un pipeline de datos es asegurar que los datos fluyan de manera continua y eficiente, desde su punto de origen hasta su destino, a la vez que se aplican las transformaciones y validaciones necesarias en el camino.

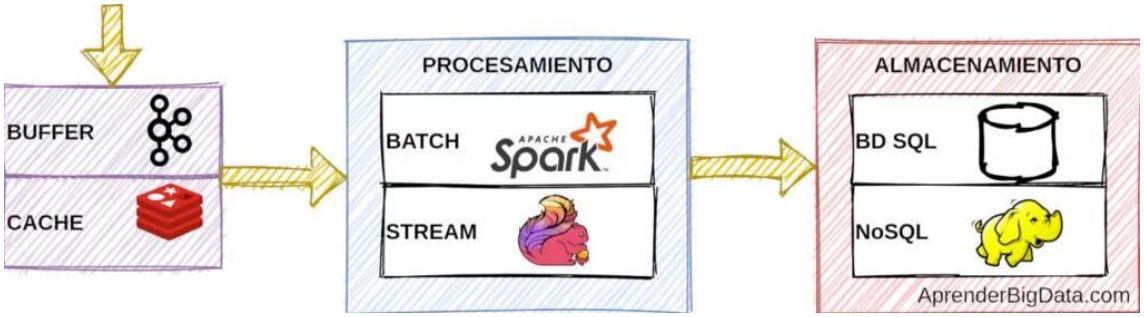

Figura 30. Pipeline de datos

Fuente: aprenderbigdata.com

Propósitos clave:

- **Automatización del flujo de datos:** Automatizar la captura, procesamiento y almacenamiento de datos para minimizar la intervención manual y asegurar que los datos estén siempre actualizados.
- **Transformación y limpieza de datos:** Aplicar transformaciones y limpieza de datos en tiempo real o por lotes para asegurar la calidad y consistencia antes de su almacenamiento o análisis.
- **Escalabilidad y resiliencia:** Diseñar el pipeline para manejar volúmenes crecientes de datos y ser resiliente frente a fallos o interrupciones.

4.1.2. Componentes clave de un pipeline de ingestión

Un pipeline de ingestión de datos generalmente incluye varios componentes clave, cada uno de los cuales desempeña un rol específico en el proceso de movimiento y transformación de datos:

- **Fuente de Datos:** El origen de los datos, que puede ser una base de datos, un servicio web, un archivo de registro, sensores IoT, entre otros.
- **Ingestión:** El proceso de capturar datos desde la fuente y moverlos al pipeline. Herramientas como Apache NiFi, Kafka, o Flume se utilizan comúnmente para esta tarea.
- **Procesamiento:** Transformación, agregación, y limpieza de datos en tiempo real o por lotes. Apache Spark y Flink son ejemplos de herramientas utilizadas para el procesamiento de datos.
- **Almacenamiento:** El destino final donde los datos procesados se almacenan para su análisis posterior. Ejemplos incluyen HDFS, Amazon S3, y bases de datos NoSQL como Cassandra o HBase.
- **Orquestación:** La coordinación y automatización de las diferentes etapas del pipeline, que asegura que las tareas se ejecuten en el orden correcto y con las

dependencias adecuadas. Apache Airflow es una herramienta comúnmente utilizada para la orquestación.

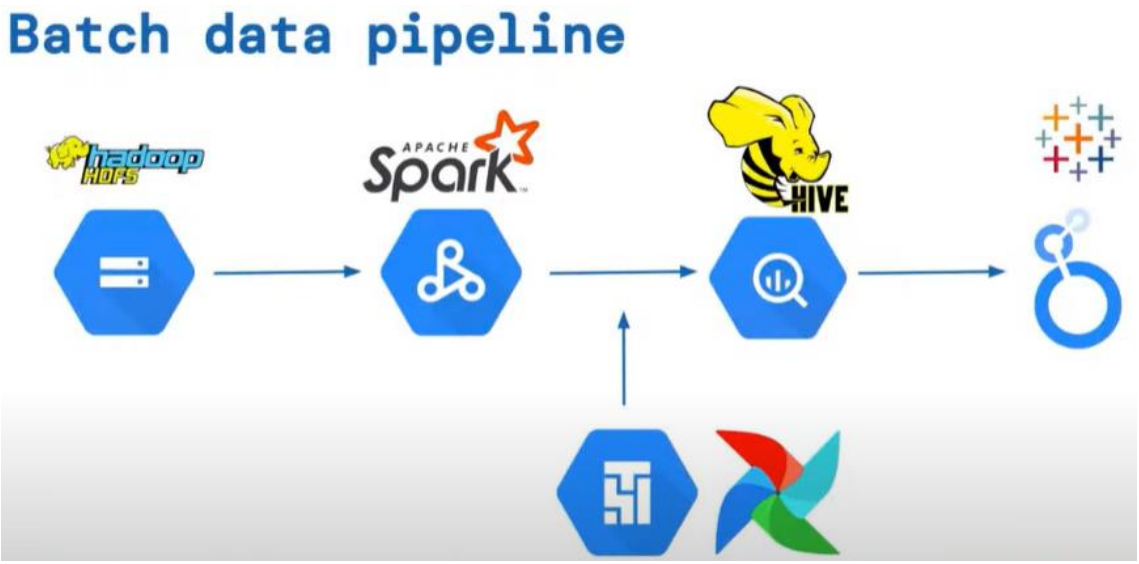

Figura 31. Componentes clave de un pipeline de datos

Fuente: escueladedatosvivos.ai

4.2 Diseño de pipelines eficientes

4.2.1. Principios de diseño: modularidad, escalabilidad y tolerancia a fallos

El diseño eficiente de pipelines de datos requiere la aplicación de varios principios clave que aseguran la robustez, escalabilidad y mantenibilidad del sistema:

- **Modularidad:** Un pipeline modular se divide en componentes independientes que pueden ser desarrollados, desplegados y mantenidos de manera autónoma. Esto permite que las diferentes partes del pipeline se actualicen o escalen sin afectar a las demás.

- **Escalabilidad:** Es esencial que el pipeline pueda escalar horizontalmente para manejar un volumen creciente de datos. Esto implica diseñar los componentes del pipeline para que puedan funcionar en paralelo y distribuir la carga entre múltiples nodos.

- **Tolerancia a fallos:** Un pipeline bien diseñado debe ser resiliente frente a fallos, lo que significa que debe ser capaz de manejar errores en cualquier parte del pipeline sin interrumpir el flujo general de datos. Esto puede incluir la implementación de mecanismos de retry, backups, y failover.

4.2.2. Ejemplo de diseño: Pipeline para ingestión y procesamiento de datos de redes sociales

Un ejemplo de un pipeline diseñado para la ingestión y procesamiento de datos de redes sociales podría incluir los siguientes pasos:

1. **Fuente de datos:** APIs de redes sociales como Twitter o Facebook, donde los datos se capturan en tiempo real.

2. **Ingestión:** Uso de Apache NiFi o Kafka para captar y mover los datos desde las APIs hacia el sistema de procesamiento.

3. **Procesamiento:** Apache Spark se utiliza para transformar y analizar los datos en tiempo real, por ejemplo, para identificar tendencias o analizar el sentimiento.

4. **Almacenamiento:** Los resultados procesados se almacenan en HDFS o S3 para su análisis posterior y generación de informes.

5. **Orquestación:** Apache Airflow se utiliza para coordinar el proceso, asegurando que cada tarea se complete antes de que la siguiente comience, y gestionando las dependencias entre tareas.

Bibliografía

1. J. G. Fernández, *Big Data y análisis de datos masivos: Introducción y aplicaciones prácticas*, Ediciones Pirámide, 2018.
2. A. Holmes, *Data Lakes: The Next Big Thing in Big Data*, O'Reilly Media, 2020.
3. M. V. López, "Procesamiento de datos en tiempo real con Apache Kafka", *Revista Tecnología y Sociedad*, vol. 18, no. 2, pp. 35-45, 2021.
4. J. Kreps, N. Narkhede, y N. Rao, "Kafka: A Distributed Messaging System for Log Processing", en *Proceedings of the NetDB Conference*, 2011.
5. A. L. García y R. Hernández, *Introducción a Apache NiFi: Gestión y Automatización de Flujos de Datos*, Alfaomega, 2022.
6. T. White, *Hadoop: The Definitive Guide*, 4th ed., O'Reilly Media, 2015.
7. J. Dean y S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.
8. R. C. González y P. Martínez, "Optimización de Pipelines de Ingesta de Datos para Big Data", *Revista Iberoamericana de Computación*, vol. 22, no. 3, pp. 79-89, 2020.
9. P. Zikopoulos y C. Eaton, *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data*, McGraw-Hill, 2011.
10. E. Sammer, *Hadoop Operations: A Guide for Developers and Administrators*, O'Reilly Media, 2012.