


Sesión: Big Data

 por Kibernetum Capacitación S.A.

QUÉ ES BIG DATA

No es un software.

No es una tecnología específica.

Big Data es un concepto que se refiere al manejo y análisis de conjuntos de datos tan grandes y complejos que no pueden ser gestionados eficientemente con herramientas tradicionales. Su relevancia ha crecido exponencialmente debido al aumento de la generación de datos desde diversas fuentes: redes sociales, dispositivos IoT, sensores, transacciones digitales, entre otros.

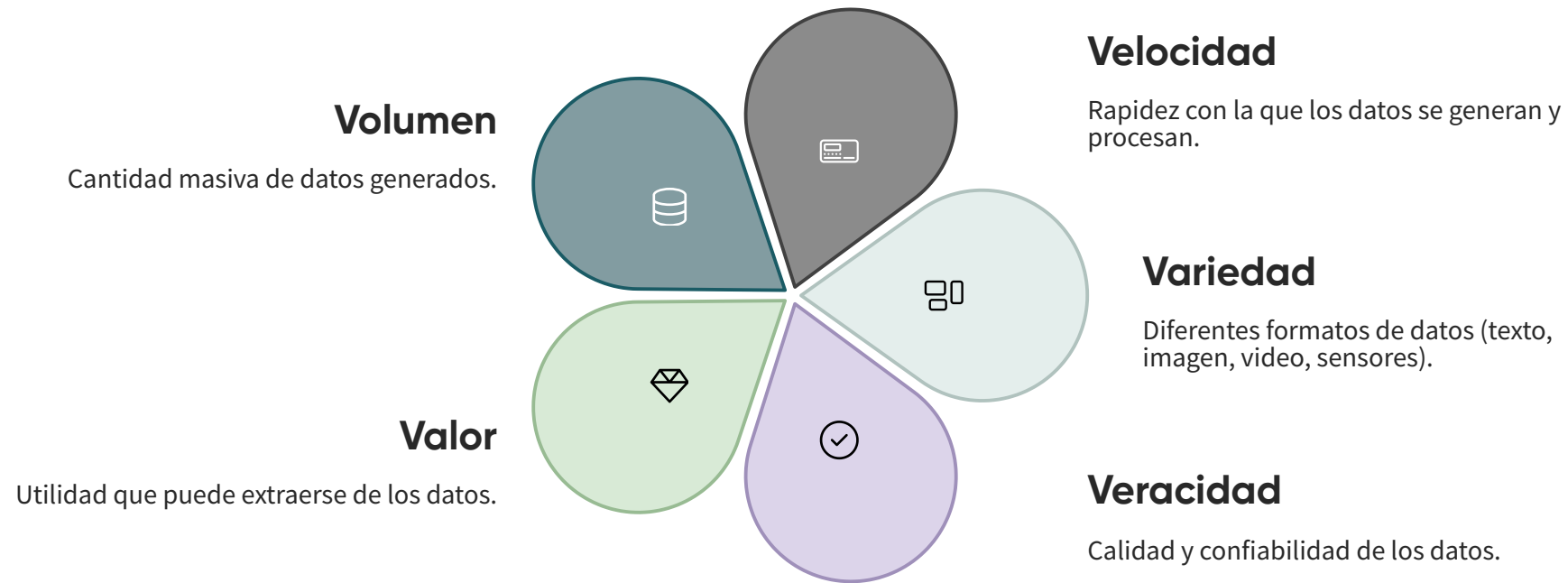
No hay una definición formal para el volumen de datos necesario para ser considerado dentro de Big Data.

No hay una definición formal para el volumen de datos necesario para ser considerado dentro de Big Data



Las 5V's de Big Data

Las características que definen el Big Data se conocen como las 5V:



Qué problemas se pueden resolver con Big Data

Big Data permite abordar problemas complejos como:



Análisis de sentimientos en redes sociales

Procesamiento de millones de publicaciones para entender tendencias y opiniones.



Predicción de fallas en máquinas industriales

Monitoreo constante de sensores para anticipar problemas antes de que ocurran.



Recomendaciones personalizadas

Sistemas como los de Netflix o Amazon que sugieren productos basados en comportamientos previos.



Prevención de fraude financiero

Detección de patrones anómalos en transacciones bancarias en tiempo real.

Ejemplo práctico: Una compañía de transporte usa sensores en sus camiones para predecir fallos mecánicos analizando millones de registros de temperatura, presión y velocidad.

EVOLUCIÓN DE LAS TECNOLOGÍAS DE BIG DATA

En sus inicios, los sistemas tradicionales de gestión de datos estaban basados en bases de datos relacionales (RDBMS) como MySQL, Oracle o SQL Server. Estas tecnologías funcionaban bien cuando los datos eran estructurados, estaban organizados en tablas y se podían almacenar en un solo servidor.

Sin embargo, a medida que internet, redes sociales, sensores IoT y dispositivos móviles comenzaron a generar datos en formatos no estructurados (texto libre, imágenes, video, logs), en volúmenes masivos y en tiempo real, estos sistemas comenzaron a presentar limitaciones importantes:

- Dificultades para escalar horizontalmente (es decir, distribuir datos y procesamiento entre varios servidores).
- Alta latencia en consultas sobre grandes volúmenes.
- Incompatibilidad con formatos de datos no estructurados o semiestructurados.
- Costos crecientes de almacenamiento y mantenimiento.

Esto impulsó una evolución tecnológica que puede resumirse en tres grandes etapas:

Etapas 1: Sistemas Relacionales (Tradicionales)



- Basados en el modelo relacional y lenguaje SQL.
- Escalabilidad vertical (mejoras de hardware).
- Alta dependencia de la estructura fija de datos.



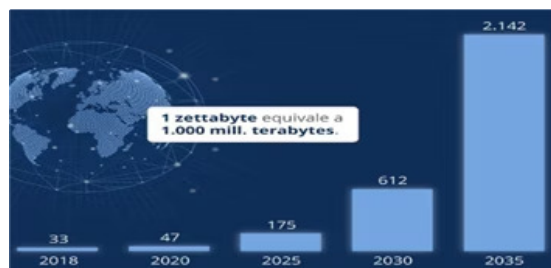
Etapas 2: Tecnologías NoSQL y Hadoop

- Aparecen sistemas como MongoDB, Cassandra, y Hadoop con HDFS y MapReduce.
- Soporte para datos no estructurados y distribución del procesamiento.
- Escalabilidad horizontal y tolerancia a fallos.
- Hadoop introduce el paradigma "almacenar primero, procesar después".

Etapas 3: Procesamiento en memoria y tiempo real



- Emergen frameworks como Apache Spark, que permiten procesamiento distribuido en memoria, mucho más rápido que MapReduce.
- Aparecen soluciones de streaming como Apache Kafka y Spark Streaming.
- Las arquitecturas Lambda y Kappa permiten combinar procesamiento batch y en tiempo real.



SISTEMAS LOCALES Y DISTRIBUIDOS

El procesamiento de datos puede realizarse en dos grandes tipos de sistemas: locales y distribuidos. Ambos tienen características, ventajas y limitaciones particulares, y la elección entre uno u otro depende del volumen de datos, la complejidad del procesamiento y los requerimientos de escalabilidad.

Sistemas Locales

Un sistema local se basa en un solo servidor o computadora que ejecuta todo el procesamiento y almacenamiento de datos. Estos sistemas son adecuados para volúmenes de datos pequeños o moderados y tareas que no requieren paralelismo masivo.

Características:

- Todo el cómputo y almacenamiento ocurre en una única máquina.
- Limitados por la capacidad física del hardware (memoria, CPU, disco).
- Riesgo de punto único de falla.
- Menor complejidad de configuración.

Tecnologías comunes:

- Bases de datos relacionales como MySQL, PostgreSQL, SQLite.
- Lenguajes y entornos locales como Python en notebooks o scripts locales.

Ejemplo práctico: Una pequeña empresa usa Excel y una base de datos en MySQL para gestionar ventas diarias.

Sistemas Distribuidos

Los sistemas distribuidos reparten la carga de procesamiento y almacenamiento entre varios nodos interconectados (servidores o máquinas). Son ideales para manejar grandes volúmenes de datos y permiten escalabilidad horizontal.

Características:

- Procesamiento paralelo en múltiples nodos.
- Mayor tolerancia a fallos (si un nodo falla, otros continúan).
- Alta escalabilidad y rendimiento.
- Requiere mayor complejidad en configuración y mantenimiento.

Tecnologías comunes:

- Apache Hadoop (almacenamiento HDFS y procesamiento MapReduce).
- Apache Spark (procesamiento distribuido en memoria).
- Cassandra y MongoDB (bases de datos NoSQL distribuidas).
- Kubernetes para orquestación de contenedores distribuidos.

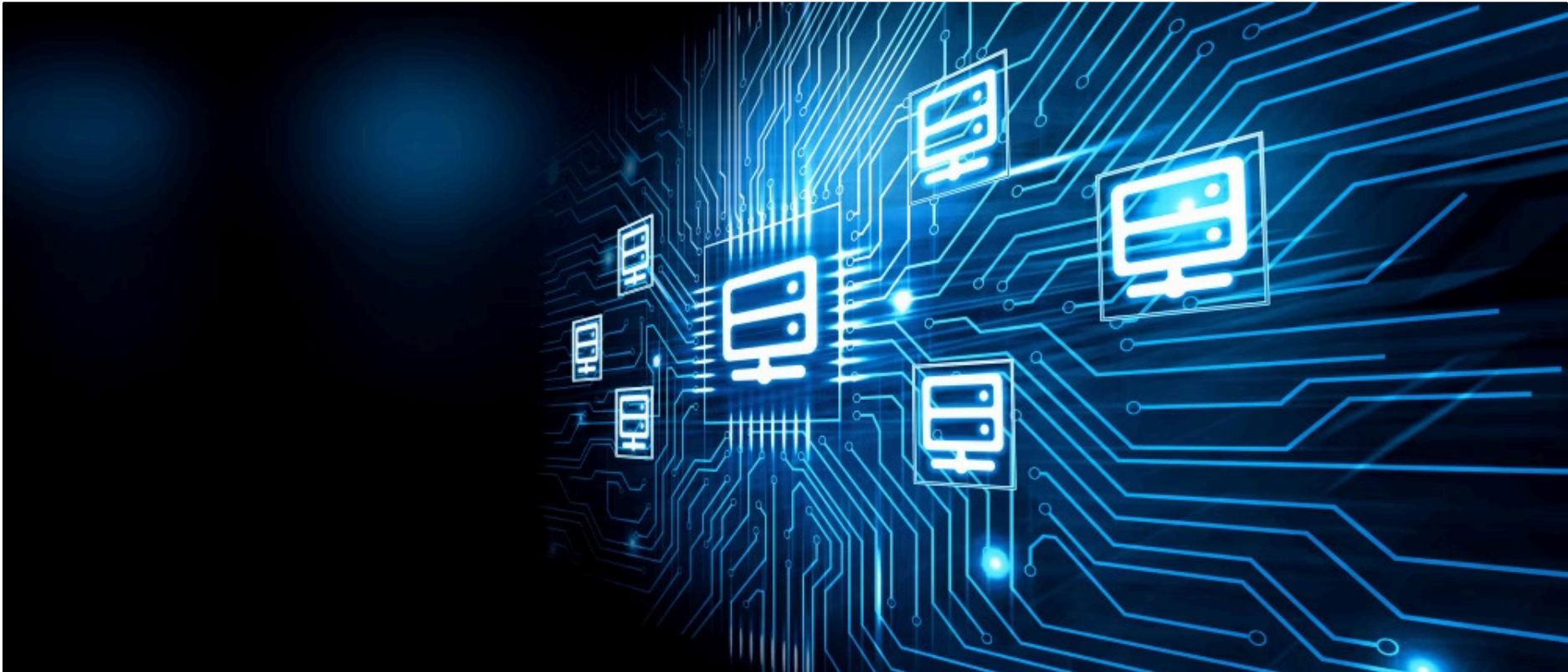
Ejemplo práctico: Netflix utiliza sistemas distribuidos para procesar datos de visualización, comportamiento de usuarios y generar recomendaciones en tiempo real usando Apache Spark y Cassandra.

TECNOLOGÍAS DE BIG DATA

En un mundo donde los datos crecen a un ritmo vertiginoso, las tecnologías tradicionales de almacenamiento y procesamiento se han vuelto insuficientes. La solución a esta problemática ha venido de la mano del ecosistema de tecnologías Big Data, diseñadas para manejar altos volúmenes, velocidades y variedades de datos en distintos entornos: corporativos, industriales, financieros, científicos, entre otros.

Estas tecnologías no solo permiten almacenar grandes volúmenes de datos, sino también analizarlos en tiempo real, extraer conocimiento valioso y tomar decisiones de forma rápida y escalable.

A continuación, exploramos las principales tecnologías del ecosistema Big Data.



NOSQL

NoSQL ("Not Only SQL") es un conjunto de bases de datos no relacionales que surgen como alternativa a las limitaciones de las bases de datos SQL tradicionales, especialmente ante los desafíos del Big Data.

A diferencia de las bases de datos relacionales, que requieren esquemas fijos y estructuras tabulares, las NoSQL permiten almacenar y consultar datos en estructuras más flexibles como documentos, pares clave-valor, columnas anchas o grafos.

Ventajas principales

- Escalabilidad horizontal: Se pueden agregar nodos fácilmente para manejar más datos.
- Alta disponibilidad y tolerancia a fallos: Clústeres distribuidos garantizan continuidad del servicio.
- Esquema flexible: No requiere estructuras fijas, ideal para datos heterogéneos.
- Rendimiento optimizado: Especialmente en lecturas y escrituras masivas.

Tipos de bases de datos NoSQL

- Documentales (MongoDB, CouchDB): Almacenan datos como documentos JSON o BSON.
- Clave-valor (Redis, Riak): Cada dato es accesible por una clave única.
- Columnares (Apache Cassandra, HBase): Eficientes en análisis de grandes volúmenes.
- Grafos (Neo4j, Amazon Neptune): Representan relaciones complejas entre entidades.

Escenarios de uso

- Aplicaciones web de alta concurrencia.
- Almacenamiento de logs y registros de sensores.
- Análisis de redes sociales y patrones de conexión.
- Recomendadores personalizados (productos, contenidos).

Estructura general de un documento NoSQL (MongoDB):

Este tipo de estructura permite almacenar múltiples niveles de información sin relaciones rígidas ni necesidad de normalización, como en las bases de datos SQL.

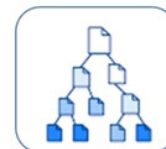
```
{
  "usuario": "jhondoe",
  "correo": "jhondoe@email.com",
  "compras": [
    {"producto": "Laptop", "precio": 950},
    {"producto": "Mouse", "precio": 25}
  ]
}
```

Ventajas principales:

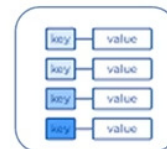
- Escalabilidad horizontal.
- Flexibilidad para manejar datos estructurados, semiestructurados o no estructurados.
- Mayor rendimiento en consultas específicas.

Escenarios de uso:

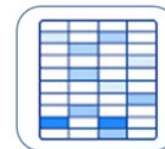
- Aplicaciones web con millones de usuarios.
- Sistemas de recomendación en tiempo real.
- Almacenamiento de registros de sensores IoT.



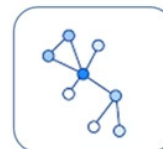
Document Store



Key-Value Store



Wide-Column Store



Graph Store

REAL TIME

El procesamiento en tiempo real se refiere a la capacidad de analizar, transformar y actuar sobre los datos al mismo tiempo que son generados. A diferencia del procesamiento batch, que trabaja con lotes de datos almacenados previamente, el enfoque en tiempo real permite obtener insights inmediatos y ejecutar acciones automáticas.

Problemas que resuelve

- Alta latencia en la toma de decisiones: permite actuar inmediatamente ante eventos críticos (como fraudes o fallos).
- Procesamiento secuencial lento: reemplaza ciclos largos de procesamiento batch.
- Incapacidad para detectar patrones fugaces: captura eventos que podrían perderse en el tiempo si no se procesan al instante.

Casos de uso

- Detección de fraudes financieros: identificar transacciones sospechosas en milisegundos.
- Sistemas de recomendación dinámica: actualizar sugerencias de productos o contenidos en tiempo real.
- Monitoreo industrial e IoT: responder a alertas o anomalías en sensores.
- Análisis de redes sociales: detectar tendencias, hashtags o eventos virales al momento.
- Automatización de decisiones: como frenar una compra, enviar una alerta o redirigir tráfico de red.

Tecnologías destacadas



Apache Kafka

Sistema de mensajería distribuida para ingesta masiva de datos.



Apache Flink

Procesamiento de streams con muy baja latencia y tolerancia a fallos.



Apache Storm

Arquitectura ligera para procesamiento de flujos en tiempo real.



Apache Spark Structured Streaming

Integración poderosa con el ecosistema Spark para análisis en memoria.



Amazon Kinesis / Google Cloud Dataflow

Opciones de streaming en la nube.

Ejemplos de uso:

- Detección de fraudes bancarios.
- Monitoreo de redes sociales.
- Sistemas de alerta en salud y transporte.



Apache Kafka



elasticsearch



Amazon
DynamoDB



Amazon S3



Apache Cassandra



Amazon Kinesis
Data Streams



ALMACENAMIENTO DISTRIBUIDO

El almacenamiento distribuido es una arquitectura que permite guardar datos en múltiples nodos o servidores conectados en red, en lugar de centralizar toda la información en un único lugar. Esta estrategia es esencial en entornos Big Data, donde los volúmenes de información superan las capacidades de los sistemas tradicionales.

¿Qué es y para qué sirve?

Permite dividir y replicar datos entre varios equipos para asegurar disponibilidad, rendimiento y resistencia a fallos. Cada nodo almacena una parte de los datos (o una copia), lo que permite trabajar en paralelo y acceder a la información de manera eficiente. También se usa este enfoque cuando las fuentes de datos son variadas.

Problemas que resuelve

- Límites de capacidad de los servidores únicos.
- Cuellos de botella en lectura/escritura de datos.
- Pérdida de datos por fallos físicos o errores de hardware.
- Dificultad para escalar almacenamiento conforme crecen los datos.

Casos de uso

- Plataformas de streaming de video (Netflix, YouTube).
- Sistemas de respaldo y almacenamiento empresarial.
- Almacenamiento de datos históricos de sensores IoT.
- Análisis de logs de navegación o transacciones bancarias.

Tecnologías disponibles



HDFS (Hadoop Distributed File System)

Arquitectura de almacenamiento tolerante a fallos, núcleo del ecosistema Hadoop.



Amazon S3

Almacenamiento escalable basado en la nube, con alta disponibilidad y durabilidad.



Google Cloud Storage / Azure Blob Storage

Soluciones similares en la nube pública.



Ceph / GlusterFS

Alternativas de código abierto para almacenamiento distribuido.

Ventajas:

- Escalabilidad horizontal: Se pueden añadir nodos para incrementar la capacidad.
- Alta disponibilidad: Datos replicados en múltiples nodos.
- Tolerancia a fallos: El sistema sigue operando incluso si algunos nodos fallan.
- Acceso concurrente: Múltiples usuarios y procesos pueden leer y escribir datos simultáneamente.

En resumen, el almacenamiento distribuido es una base fundamental en cualquier arquitectura de Big Data moderna, asegurando que la información pueda ser almacenada, accedida y recuperada de forma segura y eficiente incluso en entornos de gran escala.

Este tipo de almacenamiento divide y distribuye los datos entre varios nodos para permitir una mayor capacidad, redundancia y acceso concurrente.



PROCESAMIENTO DISTRIBUIDO

El procesamiento distribuido es una estrategia fundamental en entornos Big Data que consiste en dividir una tarea o conjunto de datos entre múltiples nodos (servidores o máquinas), permitiendo que trabajen en paralelo para alcanzar mayor eficiencia, escalabilidad y tolerancia a fallos.

¿Qué es y para qué sirve?

Consiste en ejecutar procesos simultáneamente en varios nodos interconectados. En lugar de centralizar el procesamiento en un solo servidor (limitado por su capacidad de cómputo y memoria), se reparten las cargas de trabajo, lo que mejora el rendimiento general.

Sirve para:

- Analizar grandes volúmenes de datos en menos tiempo.
- Ejecutar tareas complejas como algoritmos de aprendizaje automático, simulaciones o análisis de logs masivos.
- Asegurar la continuidad del servicio si uno o varios nodos fallan.

Problemas que resuelve

- Cuellos de botella en procesamiento secuencial.
- Limitaciones de hardware en servidores únicos.
- Falta de disponibilidad ante caídas de servicio.
- Procesos lentos en contextos de grandes volúmenes de datos.

Principales herramientas

- Apache Hadoop (MapReduce): Modelo batch, tolerante a fallos.
- Apache Spark: Procesamiento en memoria, rápido y versátil.

Ventajas

- Mayor rendimiento para tareas complejas.
- Reducción del tiempo de procesamiento.
- Capacidad de trabajar con conjuntos de datos masivos.

Ejemplo práctico:

Una empresa desea analizar los logs de acceso web de millones de usuarios para detectar patrones de navegación. Usando Apache Spark, divide los archivos entre múltiples nodos, que procesan los datos en paralelo, reduciendo el tiempo de ejecución de horas a minutos.

FRAMEWORKS

Los frameworks de Big Data son conjuntos integrados de herramientas, bibliotecas y servicios que permiten diseñar, implementar y ejecutar procesos de almacenamiento, transformación, análisis y visualización de grandes volúmenes de datos. Estos entornos proporcionan una base estandarizada y escalable para trabajar con arquitecturas distribuidas de forma eficiente.

¿Qué son y para qué sirven?

Son plataformas que agrupan múltiples componentes tecnológicos para facilitar el desarrollo de soluciones de análisis de datos a gran escala.

Sirven para:

- Orquestrar flujos de datos complejos.
- Automatizar tareas de procesamiento.
- Integrar fuentes de datos variadas.
- Ejecutar modelos de machine learning o algoritmos de análisis.



Apache Hadoop

Framework pionero en procesamiento batch y almacenamiento distribuido con HDFS.



Apache Spark

Plataforma para procesamiento distribuido en memoria, ideal para tareas batch, streaming, ML y SQL.



Apache Flink

Framework para procesamiento de datos en flujo (streaming) con baja latencia.



Apache Beam

Modelo unificado para flujos batch y streaming, compatible con motores como Flink y Dataflow.



Databricks

Plataforma comercial basada en Spark, con herramientas colaborativas para ciencia de datos.

Escenarios de aplicación



Ciencia de datos y machine learning

Entrenamiento de modelos sobre datasets distribuidos.



Procesamiento de datos en tiempo real

Análisis continuo de eventos como clics, transacciones o sensores.



ETL distribuidas

Extracción, transformación y carga de grandes volúmenes de datos desde múltiples fuentes.



Análisis financiero y detección de fraudes

Integración con sistemas en línea para actuar en tiempo real.

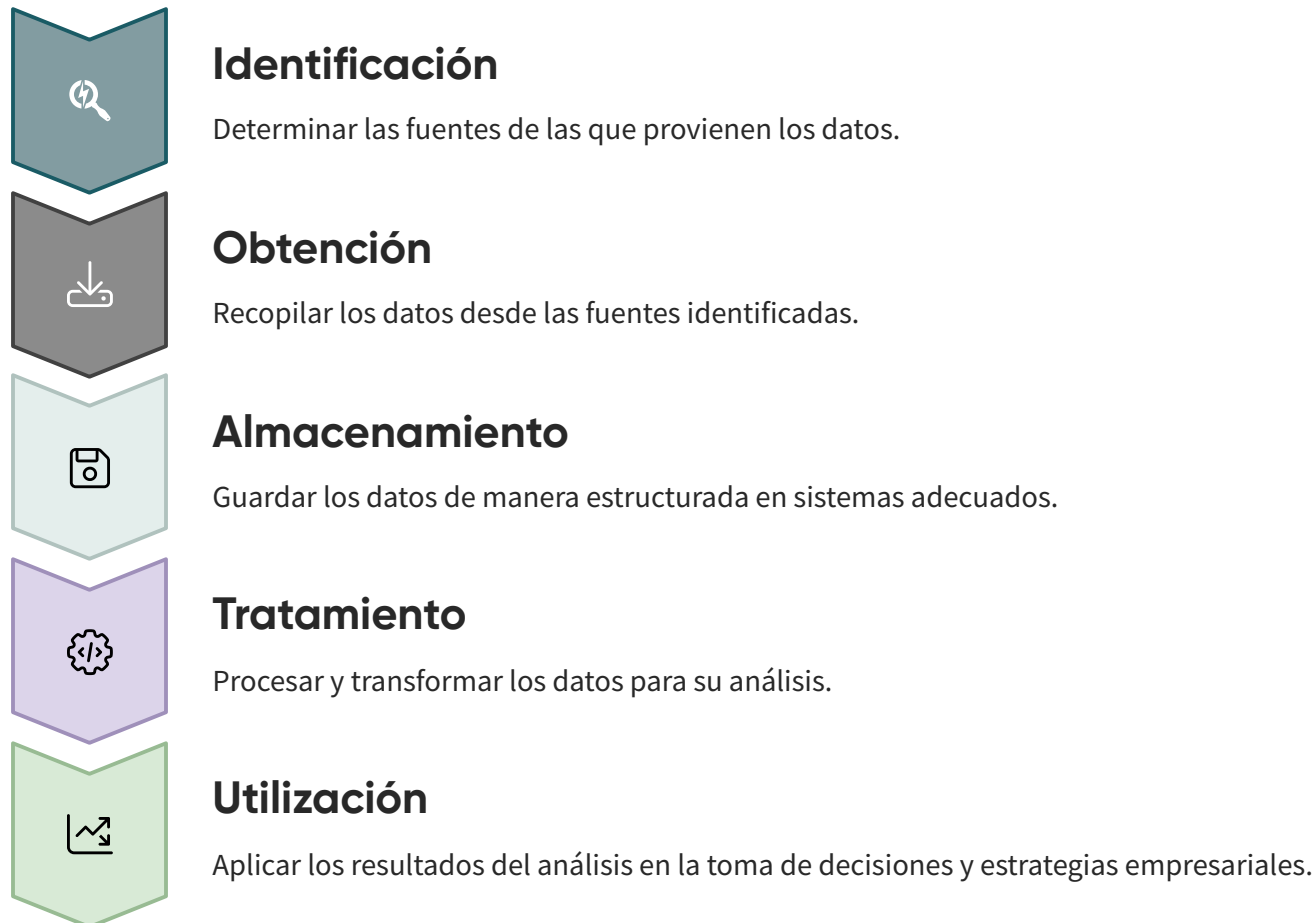
En resumen, los frameworks Big Data son esenciales para construir soluciones escalables, automatizadas y robustas que permiten extraer valor a partir de datos masivos en diversos sectores e industrias.

Los frameworks son plataformas o entornos de trabajo que integran diversas tecnologías y herramientas para facilitar el desarrollo, procesamiento y análisis de Big Data.

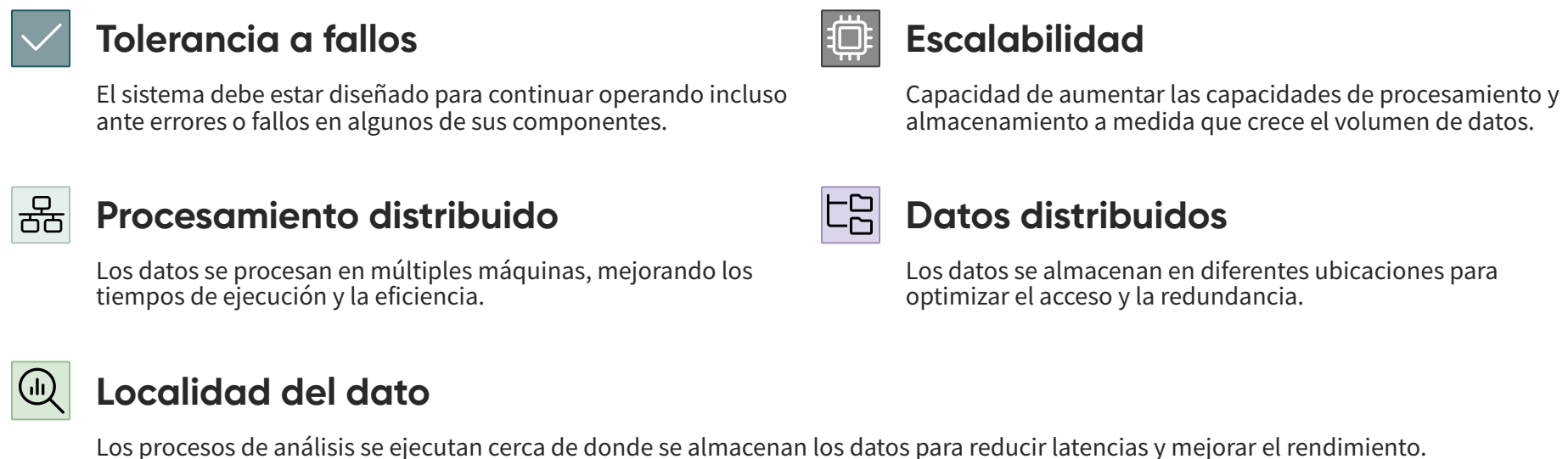
ARQUITECTURAS TÍPICAS DE BIG DATA

La arquitectura Big Data se refiere al conjunto de tecnologías, procesos y recursos diseñados para manejar volúmenes masivos de datos que no pueden ser gestionados mediante herramientas tradicionales. Estos datos provienen de diversas fuentes y se presentan en múltiples formatos, lo que requiere componentes específicos como almacenamiento distribuido, procesamiento paralelo y herramientas de análisis avanzado.

Procesos clave en la arquitectura Big Data



Características principales de la arquitectura Big Data



Principales tipologías de arquitectura Big Data

Antes de iniciar un proyecto de Big Data, es esencial seleccionar la arquitectura adecuada según las necesidades y casos de uso específicos.



Big Data On-Premise

Esta arquitectura implica que la infraestructura de Big Data se aloja en las instalaciones propias de la empresa, utilizando hardware de su propiedad dedicado exclusivamente al procesamiento de datos.

Ventajas:

- Control total: Al estar local, la empresa tiene un control completo sobre los datos y su seguridad.

Desventajas:

- Coste elevado: Requiere una inversión significativa en infraestructura y conocimientos especializados para su implementación y mantenimiento.



Big Data en la Nube

Consiste en implementar soluciones y procesos de Big Data en entornos basados en la nube, aprovechando los servicios y recursos ofrecidos por proveedores especializados.

Ventajas:

- Escalabilidad: Permite ajustar recursos según la demanda.
- Flexibilidad: Facilita la adaptación a diferentes necesidades y cargas de trabajo.
- Accesibilidad: Posibilita el acceso a los datos desde cualquier ubicación.
- Reducción de costes operativos: Disminuye la necesidad de invertir en infraestructura física y su mantenimiento.

Desventajas:

- Dependencia del proveedor: La empresa confía en terceros para la gestión y seguridad de sus datos.



Importancia de la arquitectura Big Data

La arquitectura Big Data es esencial para la toma de decisiones y la innovación en diversos sectores. Su capacidad para gestionar grandes volúmenes de datos, procesarlos y extraer información valiosa impulsa avances en inteligencia artificial, análisis predictivo y personalización de servicios, otorgando a las empresas ventajas competitivas significativas.

HADOOP Y SPARK

Hadoop



Hadoop es un framework de software de código abierto desarrollado por la Apache Software Foundation que permite el almacenamiento distribuido y el procesamiento de grandes volúmenes de datos utilizando un conjunto de computadoras conectadas en red (clúster).

Componentes principales:

- HDFS (Hadoop Distributed File System): sistema de archivos distribuido.
- MapReduce: modelo de programación para el procesamiento paralelo de datos.
- YARN: sistema de gestión de recursos y planificación de tareas.
- Common: bibliotecas y utilidades compartidas.

¿Qué problemas resuelve?

- **Escalabilidad:** permite trabajar con petabytes de datos simplemente agregando más nodos al clúster.
- **Fallo de hardware:** detecta fallos automáticamente y replica los datos para evitar pérdidas.
- **Procesamiento paralelo:** divide las tareas en pequeños fragmentos que se ejecutan en paralelo.
- **Costo:** puede ejecutarse en hardware común (commodity hardware), reduciendo costos frente a soluciones comerciales.
- **Flexibilidad de datos:** puede manejar diferentes tipos de datos (texto, imágenes, videos, logs, etc.).

Ventajas:

- Escalabilidad horizontal
- Tolerancia a fallos
- Costo-efectivo
- Código abierto
- Flexibilidad

Desventajas de Hadoop:

- **Curva de aprendizaje pronunciada:** especialmente para MapReduce.
- **No es en tiempo real:** Hadoop está pensado para procesamiento **por lotes**, no para respuestas inmediatas.
- **Problemas con datos pequeños:** no es eficiente con volúmenes de datos pequeños.
- **Requiere una buena configuración y administración del clúster.**
- **Competencia con nuevas tecnologías:** como Apache Spark, que es más rápido para ciertas tareas.

Ejemplos de uso de Hadoop:

- Análisis de logs web de millones de usuarios.
- Procesamiento de datos genómicos.
- Sistemas de recomendación para e-commerce.
- Análisis de redes sociales.
- Detección de fraudes en bancos.

Apache Spark



Apache Spark es un motor de procesamiento de datos de código abierto diseñado para ser rápido, escalable y versátil, permitiendo procesar grandes volúmenes de datos en memoria (in-memory), lo que lo hace mucho más rápido que sistemas como Hadoop MapReduce.

Características principales:

- Procesamiento en memoria
- Soporte para múltiples tipos de procesamiento (batch, streaming, ML, SQL)
- APIs en Python, Scala, Java y R
- Integración con diversas fuentes de datos.

¿Qué problemas resuelve Spark?

- Procesamiento lento de Hadoop MapReduce: Spark puede ser hasta 100 veces más rápido, gracias al procesamiento en memoria.
- Procesamiento en tiempo real: con Spark Streaming se pueden procesar datos conforme van llegando.
- Unificación de tareas: permite realizar procesamiento batch, streaming, ML y SQL con una sola plataforma.
- Eficiencia en trabajos iterativos y algoritmos de ML: al trabajar en memoria, evita escribir/leer datos del disco constantemente.

Facilidad de uso: proporciona APIs en Python, Scala, Java y R, y una interfaz interactiva para exploración de datos.

Ventajas:

- Velocidad (hasta 100 veces más rápido que MapReduce)
- Versatilidad
- Facilidad de uso
- Amplio ecosistema

Desventajas de Apache Spark:

- Alto consumo de memoria: el procesamiento en memoria requiere gran cantidad de RAM. Complejidad en configuración y despliegue en producción.
- No reemplaza completamente a bases de datos: no está optimizado para operaciones OLTP o consultas transaccionales.
- Curva de aprendizaje inicial en clústeres grandes.
- La versión de Spark Streaming original no es completamente "real-time" (usa micro-batching).

Ejemplos de uso de Spark:

- Detección de fraudes bancarios en tiempo real.
- Recomendaciones personalizadas en plataformas de e-commerce.
- Análisis de logs de millones de usuarios web.
- Modelos de predicción del clima o consumo energético.
- Procesamiento de grandes volúmenes de datos de sensores en sistemas IoT.

Característica	Hadoop MapReduce	Apache Spark
Velocidad	Lento (procesamiento en disco)	Rápido (procesamiento en memoria)
Streaming	No	Sí
Machine Learning	Limitado	Incluye MLlib
Complejidad	Más bajo nivel	APIs de alto nivel
Escenarios ideales	Procesos batch grandes	Batch + Streaming + ML

OTRAS TECNOLOGÍAS

Además de los frameworks y plataformas más conocidas como Hadoop y Spark, el ecosistema Big Data incluye una gran variedad de herramientas especializadas que cumplen funciones específicas dentro del flujo de trabajo de datos. Estas tecnologías permiten extender las capacidades del sistema y adaptarlo a necesidades concretas como ingestión, orquestación, análisis avanzado, visualización, entre otras.

Tecnologías de ingesta y transmisión de datos

- **Apache Kafka:** Plataforma distribuida para manejar flujos de datos en tiempo real, ideal para conectar múltiples sistemas.
- **Apache NiFi:** Herramienta de integración de datos orientada a flujos, con interfaz visual y control de versiones.
- **Flume:** Diseñada para recolectar, agregar y mover grandes cantidades de datos de logs.

Orquestación y automatización

- **Apache Airflow:** Framework para definir, programar y monitorear flujos de trabajo (pipelines) complejos.
- **Luigi:** Sistema similar a Airflow, desarrollado por Spotify para tareas de ETL.

Visualización de datos

- **Kibana:** Visualización de datos almacenados en Elasticsearch, útil para monitoreo y análisis en tiempo real.
- **Grafana:** Dashboard potente y flexible para métricas en tiempo real.
- **Tableau / Power BI:** Plataformas comerciales ampliamente usadas en entornos empresariales.

Análisis avanzado y machine learning

- **TensorFlow:** Librería de código abierto para aprendizaje profundo.
- **Scikit-learn:** Librería de Python para machine learning clásico y modelado estadístico.
- **MLflow:** Herramienta para gestionar el ciclo de vida de los modelos de machine learning.

Servicios en la nube (Big Data as a Service)

- **Google BigQuery:** Plataforma de análisis de datos masivos en la nube.
- **Amazon Redshift:** Data warehouse escalable para análisis complejos.
- **Azure Synapse Analytics:** Solución de análisis integrada para grandes volúmenes.

ACTIVIDAD PRÁCTICA GUIADA

TÍTULO: Realizar un cuadro comparativo entre **Hadoop y Apache Spark**

OBJETIVO: Comprender y analizar las principales características de las tecnologías **Hadoop y Apache Spark**

INSTRUCCIONES: Elabora un cuadro comparativo en el cual analices las principales características de Hadoop y Apache Spark

Característica	Hadoop MapReduce	Apache Spark
Velocidad	Lento (procesamiento en disco)	Rápido (procesamiento en memoria)
Streaming	No	Sí
Machine Learning	Limitado	Incluye MLlib
Complejidad	Más bajo nivel	APIs de alto nivel
Escenarios ideales	Procesos batch grandes	Batch + Streaming + ML

CASO PRÁCTICO: DETECCIÓN DE FRAUDES

Escenario: Una entidad financiera desea detectar transacciones potencialmente Fraudulentas en tiempo real para evitar afectar a sus clientes. Cada segundo, se generan miles de operaciones a través de tarjetas de crédito, transferencias electrónicas, apps móviles y cajeros automáticos en todo el país.

Aplicación de las 5V's en este caso

Aplicación 5V's	Descripción en este caso
Volumen	Miles de transacciones por segundo, provenientes de todo el país.
Velocidad	Se requiere análisis en tiempo real, tan pronto ocurre la transacción.
Variedad	Datos estructurados (monto, hora) y semiestructurados (dispositivo, ubicación geográfica).
Veracidad	Necesidad de datos precisos y confiables para evitar falsos positivos.
Valor	Detectar fraudes rápidamente protege a clientes y reduce pérdidas económicas.

¿Por qué este problema no podría resolverse eficientemente con una arquitectura local?

- Las bases de datos relacionales no pueden escalar para soportar el volumen y la velocidad requerida.
- Un único servidor tendría cuellos de botella y puntos de falla.
- No ofrecen análisis en tiempo real (solo procesamiento batch).
- No permiten integración eficiente con múltiples fuentes de datos heterogéneos.

Este caso demuestra la necesidad de implementar una arquitectura distribuida con procesamiento en tiempo real, utilizando tecnologías como Apache Kafka para la ingesta de datos, Apache Spark Streaming para el procesamiento y algoritmos de machine learning para la detección de patrones fraudulentos.