Ingeniería de Datos. Conferencias

Lic. Niley González

2024 - 2025

1

Conferencia 1

Introducción a la Ingeniería de Datos. Metodologías para la Ciencia de Datos

¿Cómo definirían ustedes la ciencia de datos?

Pausa para que los estudiantes compartan sus ideas.

Diferentes autores han intentado definir el campo de la ciencia de datos; algunas de las prespectivas son:

- campo multidisciplinario que combina áreas como la informática, las matemáticas y la estadística. Su objetivo principal es utilizar métodos y técnicas científicas para extraer conocimiento y valor de grandes volúmenes de datos, estructurados o no estructurados. (2019)
- campo interdisciplinario que integra disciplinas como las ciencias de la computación, el aprendizaje automático, las matemáticas y las estadísticas. (2021)
- tema multidisciplinario cuyo propósito es descubrir conocimiento para apoyar la toma de decisiones en diversos contextos empresariales.(2020)

En general, hay un consenso en que la ciencia de datos es un campo multidisciplinario o interdisciplinario que se nutre de áreas como la informática, la estadística y las ciencias de la computación. Su enfoque principal es el estudio de los datos, con el propósito de extraer conocimiento y valor a partir de ellos.

Ahora, ¿qué entienden ustedes por una metodología?

Pausa para que los estudiantes compartan sus ideas.

Una metodología puede entenderse como una **estrategia, guía o conjunto de pautas** que nos ayudan a desarrollar un proceso o actividad de manera estructurada. A diferencia de las herramientas o tecnologías específicas, las metodologías no están ligadas a un software o hardware en particular. En cambio, proporcionan un **marco de trabajo** que nos indica cómo proceder de manera sistemática para alcanzar nuestros objetivos.

¿Por qué es importante esto?
En el contexto de la ciencia de datos, las metodologías ayudan a abordar problemas complejos de manera organizada, intentando que cada paso del proceso esté bien definido y alineado con los objetivos del proyecto.

1.1 Metodologías en Ciencia de Datos

A continuación se presentan varias metodologías de la Ciencia de Datos, analizando críticamente su estructura, enfoque y aplicabilidad en diferentes contextos. El objetivo es comprender cómo cada metodología se adapta -o no- a distintos escenarios organizacionales, técnicos y de complejidad. A través de este análisis, descubriremos por qué ninguna metodología puede aplicarse universalmente a todas las circunstancias, y cómo su implementación rígida y sin adaptación puede llevar a resultados subóptimos.

1.1.1 KDD (Knowledge Discovery in Databases)

Definido como un proceso interactivo e iterativo de 5 fases para descubrir conocimiento por sus creadores en 1996:

- **Selección**: Los datos cambian de acuerdo con los objetivos del proceso. Se establece un grupo de datos con el que proceder.
- **Procesamiento/Limpieza**: En la fase de data cleaning se examina la calidad de los datos y se manejan situaciones como datos con parámetros faltantes/nulos, datos duplicados, etc.
- **Transformación/Reducción**: Convertir datos pre-procesados en utilizables, identificando características importantes según los objetivos del proceso.

- **Minería de Datos**: Búsqueda de patrones de interés mediante técnicas como clasificación, regresión, clustering, correlaciones, etc.

- **Interpretación/Evaluación**: Fase final, de consolidación del conocimiento encontrado en los datos. Se preparan los resultados para documentación y toma de decisiones. Los datos se han transformado en visualizaciones para facilitar la evaluación del resultado depurado.

1.1.2 CRISP-DM (Cross-Industry Standard Process for Data Mining)

Publicada en 1999 con el objetivo de estandarizar los procesos de minería de datos en diversos sectores, se ha consolidado como la metodología más utilizada en proyectos de minería de datos, análisis y ciencia de datos:

- **Comprensión Empresarial**: se centra en entender los objetivos del proyecto, para luego ser evaluados y descubrir si los datos son aptos para cumplir con los objetivos y producir un plan de proyecto.

- **Comprensión de Datos**: Recopilación, descripción y exploración de los datos iniciales.

- **Preparación de Datos**: 5 tareas: selección de datos, limpieza de datos, construcción de datos, integración de datos y ajuste del formato.

- **Modelado**: Construcción y evaluación de varios modelos con diferentes técnicas algorítmicas. Se determina que algoritmos probar (por ejemplo, regresión, red neuronal); se realiza un diseño de experimentos; construir el modelo y evaluar el modelo.

- **Evaluación**: Se evalúa y revisa la creación de modelos respecto a los objetivos comerciales. Para ello se evaluan los resultados, se revisan los procesos y se determinan los próximos pasos.

- **Implementación**: Despliegue, seguimiento, mantenimiento y revisión final de los resultados obtenidos.

CRISP-DM presenta un proceso iterativo estructurado, definido y documentado. Es una metodología empleada como referencia por otras metodologías.

1.1.3 SEMMA (Sample, Explore, Modify, Model, Assess)

Propuesta para manejo de grandes volúmenes de datos:

- **Muestreo**: Selección de una muestra representativa de datos del problema que está investigando. La forma correcta de obtener una muestra es la selección aleatoria.

- **Exploración**: Exploración de información útil, con la finalidad de sintetizar el problema y mejorar la eficiencia del modelo.

- **Modificación**: Manipulación de los datos con base en la investigación realizada para que los datos ingresados al modelo estén definidos y en un formato adecuado.

- **Modelado**: Modelado de datos, con el propósito de establecer una relación entre las variables explicativas y el objeto de estudio.

- **Evaluación**: Validación comparativa de los resultados, a través del análisis de los modelos, comparado con otros modelos estadísticos o una nueva muestra poblacional.

1.1.4 RAMSYS (Rapid collaborative data Mining System)

Desarrollada por Steve Moyle en 2002. Es una metodología que apoya proyectos de minería de datos, por ello amplía el método CRISP-DM.
Define su metodología en tres roles:

- **Modeladores**: encargados de probar la viabilidad de las hipótesis y generar nuevos conocimientos.

- **Data Master**: responsable de mantener la versión actual de la base de datos, las transformaciones y la información sobre los datos, como metadatos e información sobre la calidad de los datos.

- **Comité de Dirección**: responsable de establecer los desafíos del proyecto, definir criterios, recibir y seleccionar las presentaciones.

1.1.5 TDSP (Team Data Science Process)

Metodología de Microsoft de 2017. Es de cierta forma una combinación de Scrum y CRISP-DM. El ciclo de vida de TDSP se compone de cinco etapas principales:

- **Comprensión empresarial**: Se definen los objetivos y se identifican las fuentes de datos.
- **Adquisición y comprensión de datos**: Se incorporan los datos y se determina si se puede responder a la pregunta planteada (combina efectivamente la Comprensión de los Datos y la Limpieza de los Datos de CRISP-DM).
- **Modelado**: Ingeniería de características (feature engineering) y entrenamiento de modelos (model training). Combina Modelado y Evaluación de CRISP-DM).
- **Implementación**: Implementar en un entorno de producción.
- **Aceptación del cliente**: Validación por parte del cliente de si el sistema satisface las necesidades del negocio (una fase no cubierta explícitamente por CRISP-DM).

TDSP aborda la debilidad de CRISP-DM en cuanto a la falta de definición del equipo, definiendo seis roles:

- Arquitecto de soluciones
- Project Manager
- Ingeniero de datos
- Científico de datos
- Desarrollador de aplicaciones
- Líder de proyecto (Project lead)

1.1.6 Conclusiones del tema

A lo largo de esta conferencia, hemos explorado diversas metodologías utilizadas en la Ciencia de Datos. Aunque cada una tiene sus particularidades, todas comparten elementos comunes:

- **Comprensión del Negocio**: Definir el problema empresarial y se identifican los objetivos del análisis. El equipo de ciencia de datos debe trabajar en estrecha colaboración con los clientes para entender el problema y definir los objetivos.
- **Comprensión de los Datos**: Identificar y recopilar los datos requerido para el análisis. Exploración de los datos para entender su estructura, calidad y completitud.
- **Preparación de los Datos**: Limpiar, transformar y preparar los datos para garantizar que estén en el formato y calidad adecuados para el análisis.
- **Modelado de Datos**: Seleccionar técnicas de modelado adecuadas para analizar los datos e implementar modelos predictivos. Esta etapa también involucra la selección de algoritmos, ajuste de parámetros y validación del modelo.
- **Evaluación**: Evaluar el rendimiento del modelo y su capacidad para resolver el problema empresarial. Utilizando métricas de evaluación adecuadas y realizando mejoras al modelo si es necesario.
- **Despliegue**: Desplegar el modelo en un entorno de producción, integrándolo en los procesos de la negocio y asegurando su correcto funcionamiento.
- **Monitoreo y Mantenimiento**: Supervisar el rendimiento del modelo en producción y realizar ajustes para mantener su efectividad.

En resumen, una metodología de Ciencia de Datos es un enfoque estructurado que combina comprensión del negocio, manejo de datos, modelado y evaluación para transformar datos en soluciones efectivas. Sin embargo, es crucial recordar que:

- **No existe una metodología universal**: Cada proyecto tiene características únicas que pueden requerir adaptaciones o combinaciones de enfoques.
- **La flexibilidad es clave**: Las metodologías no deben aplicarse de manera rígida, sino como guías que permitan ajustarse a las necesidades específicas del problema.
- **La colaboración es esencial**: La comunicación entre científicos de datos, ingenieros y stakeholders es fundamental para alinear objetivos y garantizar resultados útiles.
- **El ciclo nunca termina**: La Ciencia de Datos es un proceso iterativo, donde el monitoreo y la mejora continua son parte integral del éxito.

1.2 Ingeniería de Datos

Para concluir vamos a definir en que consiste el trabajo de un ingeniero de datos.

En el libro *Fundamentals of Data Engineering* de *Joe Reis and Matt Housley* se define el término como:

La Ingeniería de Datos consiste en el desarrollo, interpretación y mantenimiento de sistemas y procesos que toman datos crudos y producen información consistente de alta calidad que soporta downstream casos de uso como analíticas y machine learning.

Un ingeniero de datos maneja el ciclo de vida de la ingeniería de datos que abarca el proceso que comienza en obtener los datos de las fuentes y termina sirviéndolos, después de procesados para los casos de uso.

En este curso estaremos viendo la ingeniería de datos como proceso integral que abarca la recolección, almacenamiento, procesamiento, y disponibilidad de datos.

1.2.1 Breve historia de la evolución de la ingeniería de datos

Los comienzos: 1980 a 2000 de Data Warehouse a la Web.
El nacimiento de la ingeniería de datos tiene sus raíces en data warehousing[1].
Que data desde la década de 1970 y toma forma en los 80s cuando se crea el término data warehouse. Luego surge el Structured Query Language (SQL). Mientras crecían los nacientes sistemas de datos, los negocios necesitaban dedicar herramientas a reportar y a business intelligence (BI). Surgiendo roles como BI ingeniero[2], desarrolladores ETL[3] e ingenieros de data warehouse. Precursores de lo que se considera actualmente los ingenieros de datos. También se popularizó el internet a mediados de los 90s, surgiendo una ola de compañías centradas en la web.

Los inicios de los 2000: El nacimiento de la ingeniería de datos contemporánea.
Las grandes compañías sobrevivientes como Yahoo, Google y Amazon crecerían

[1] Centralized repository that aggregates data from various sources to support data analysis, data mining and artificial intelligence

[2] A business intelligence engineer designs, implements, and maintains systems used to collect and analyze business intelligence data.

[3] Responsible for designing, building, managing, and maintaining ETL (Extract, Transform, Load) processes.

hasta convertirse en gigantes tecnológicos. La necesidad de sistemas escalables, con alta disponibilidad, confiables y cost-effective llevó a innovaciones en sistemas distribuidos y almacenamiento, marcando el inicio de la era del 'big data'. La combinación del surgimiento de nuevos algoritmos y metodologías como: 'Google File System', 'MapReduce', Apache Hadoop, Amazon Elastic Compute Cloud y su apertura como servicio al público a través de Amazon Web Services (AWS) creó una nueva era en el manejo y tratamiento de datos. Nacía la era de ingeniero de big data.
Mientras AWS se volvió muy rentable otras compañías lanzaros sus propios ecosistemas en la nube: Google Cloud, Microsoft Azure, DigitalOcean. La nube es discutiblemente una de las innovaciones más significativas del siglo 21; iniciando una revolución en la forma en que el software y las aplicaciones de datos se desarrollan y despliegan

Finales de los 2000 y la década de 2010: The Big Data Engineering Era
Surgen herramientas open source que democratizan el acceso a tecnologías de big data; que ya no estarían limitadas solo a las grandes compañías.
También comienza la transformación de procesamiento en batch a streaming a partir de eventos.
Ocurre una explosión de herramientas de manejo de datos. Y los ingenieros de bog data debían ser proficientes en desarrollo de software y configuración de infraestructuras de bajo nivel.
Big data engineers se centraban en manejar sistemas de datos de gran escala, pero la complejidad y el costo de mantener estas nuevas herramientas impulsaron a optar por simplificaciones.
El término "big data" perdió su brillo mientras se volvían más accesibles las herramientas para procesarlos; los ingenieros de big data engineers pasan a ser simplemente ingenieros de datos.

2020s: Ingeniería por el ciclo de vida de los datos.
El rol de los ingenieros de datos se encuentra en rápida evolución. La tendencia está siendo centrarse en herramientas descentralizadas, modulares y abstractas. Las tendencias populares a principios de la década de 2020 incluyen el modern data stack (MDS), que representa una colección de productos "listos para usar", tanto de código abierto como de terceros, ensamblados para facilitar el trabajo de los analistas. Al mismo tiempo, las fuentes de datos y los formatos de datos están creciendo tanto en variedad como en tamaño. La ingeniería de datos es, cada vez más, una disciplina de interconexión, que conecta varias tecnologías como si fueran piezas de LEGO, para servir a los objetivos empresariales finales.

1.2.2 Relación entre la Ingeniería de Datos y la Ciencia de Datos

La ingeniería de datos se sitúa upstream de la ciencia de datos, lo que significa que los ingenieros de datos proporcionan las entradas utilizadas por los científicos de datos.
Para muchos lo más interesante de la ciencia de datos es construir y optimizar modelos de Machine Learning; la realidad es que se estima que entre el 70% y el 80% del tiempo se dedica a recopilar, limpiar y procesar datos.
Además; usualmente los científicos de datos no están entrenados para diseñar sistemas de datos de grado de producción, y terminan haciendo este trabajo improvisadamente porque carecen del soporte y los recursos de un ingeniero de datos.
En un mundo ideal, los científicos de datos deberían dedicar más del 90% de su tiempo al análisis, la experimentación y el ML. Esto se logra cuando los ingenieros de datos se centran en construir una base sólida para que los científicos de datos tengan éxito.

1.2.3 Habilidades del Ingeniero de Datos

El conjunto de habilidades de un ingeniero de datos debe abarcar las ideas subyacentes de la ingeniería de datos: seguridad, gestión de datos, DataOps, arquitectura de datos e ingeniería de software. Se requiere un entendimeinto de como evaluar herramientas de datos y como estas encajan en el ciclo de vida de la ingeniería de datos.

Un ingeniero de datos maneja una gran cantidad de piezas móviles complejas y debe optimizar constantemente a lo largo de los ejes de costo, agilidad, escalabilidad, simplicidad, reutilización e interoperabilidad. También se espera que el ingeniero de datos cree arquitecturas de datos ágiles que evolucionen a medida que surjan nuevas tendencias.

Por definición, un ingeniero de datos debe comprender tanto los datos como la tecnología. Con respecto a los datos, esto implica conocer varias de las mejores prácticas en torno a la gestión de datos. En el extremo tecnológico, un ingeniero de datos debe estar al tanto de varias opciones de herramientas, su interrelación y sus trade-offs.

2

Conferencia 2

El ciclo de vida de la ingeniería de datos.

2.1 El Ciclo de Vida de la Ingeniería de Datos (Data Engineering Lifecycle)

El ciclo de vida de la ingeniería de datos o Data Engineering Lifecycle.

Estudiaremos este ciclo de forma agnóstica a un tipo de software o hardware específico. La idea es alejarnos de las tecnologías y centrarnos en los datos y el propósito que deben servir.

El ciclo de vida de la ingeniería de datos comprende las siguientes etapas que transforman datos crudos en un producto final con valor listo para ser consumido por los analistas, científicos de datos, ingenieros de ML y otros:

- Generación
- Almacenamiento
- Ingesta
- Transformación
- Serving

Además de las etapas, el ciclo tiene una noción de ideas subyacentes, críticas a o largo de todo el ciclo de vida. Estas incluyen seguridad, gestión de datos, DataOps, arquitectura de datos, orquestación e ingeniería de software.

El almacenamiento ocurre a lo largo de todo el ciclo a medida que los datos fluyen desde el inicio hasta el final.
En general, las etapas intermedias (almacenamiento, ingesta, transformación) pueden mezclarse un poco. Varias etapas del ciclo de vida pueden repetirse, ocurrir fuera de orden, superponerse o entrelazarse.

El ciclo de vida de la ingeniería de datos es un subconjunto de todo el ciclo de vida de los datos.

Un ingeniero de datos tiene varios objetivos de alto nivel a lo largo del ciclo de vida de los datos: producir un Return on Investment(ROI) y reducir los costos, reducir el riesgo y maximizar el valor y la utilidad de los datos.

2.1.1 Generación

Un *sistema de origen* es el origen de los datos utilizados en el ciclo de vida de la ingeniería de datos. Por ejemplo, un sistema de origen podría ser un dispositivo IoT, una cola de mensajes de aplicación o una base de datos transaccional.

Un ingeniero de datos consume datos de un sistema de origen, pero normalmente no posee ni controla el sistema de origen en sí. El ingeniero de datos necesita tener un entendimiento funcional de cómo funcionan los sistemas de origen, la forma en que generan los datos, la frecuencia y la velocidad de los datos, y la variedad de datos que se generan.
También necesitan mantener una línea de comunicación abierta con los propietarios del sistema de origen sobre los cambios que podrían interrumpir los pipelines y los análisis.

Las fuentes producen datos consumidos por sistemas posteriores, incluidas hojas de cálculo generadas por humanos, sensores IoT y aplicaciones web y móviles. Cada fuente tiene su volumen y cadencia únicos de generación de datos.

Ejemplos de sistemas de origen:

- Dispositivos IoT
- Terminales de tarjetas de crédito
- Operaciones de la bolsa
- Sensores de telescopios
- Spreadsheets

Tipos de sistemas de origen:

- Archivos y datos no estructurados (Excel, CSV, JSON, XML, TXT). Estos archivos tienen sus peculiaridades y pueden ser estructurados (Excel, CSV), semi-estructurados (JSON, XML, CSV) o no estructurados (TXT, CSV).

- APIs
- Bases de Datos: Pueden ser relacionales(SQL), o no relacionales como: Document stores, Wide-column, Graph databases,
- Application Databases (online transaction processing (OLTP) system) : Una base de datos de aplicaciones almacena el estado de una aplicación. Un ejemplo típico es una base de datos que almacena los saldos de cuentas bancarias. A medida que se producen transacciones y pagos de los clientes, la aplicación actualiza los saldos de las cuentas bancarias. Normalmente, una base de datos de aplicaciones es un sistema de procesamiento de transacciones en línea (OLTP): una base de datos que lee y escribe registros de datos individuales a una alta velocidad.
- Fuentes de Datos de Terceros: Debido a que la tecnología se ha integrado en muchas empresas (y agencias gubernamentales), estas buscan ofrecer sus datos a clientes y usuarios. El acceso directo a datos de terceros se realiza comúnmente a través de APIs, mediante el intercambio de datos en una plataforma en la nube o mediante la descarga de datos.
- Colas de Mensajes y Plataformas de Streaming de Eventos: Las arquitecturas basadas en eventos (event-driven) son cada vez más populares en software. Además, las aplicaciones que integran analítica en tiempo real (data apps) se benefician de las arquitecturas basadas en eventos, ya que los eventos disparan acciones en la aplicación y alimentan el análisis en tiempo real.

Ahora se presentan preguntas para evaluar sistemas de origen que los ingenieros de datos deben considerar:

- ¿Cuáles son las características esenciales de la fuente de datos? ¿Es una aplicación? ¿Un enjambre de dispositivos IoT?
- ¿Cómo se persisten los datos en el sistema de origen? ¿Los datos se persisten a largo plazo, o son temporales y se eliminan rápidamente?
- ¿A qué velocidad se generan los datos? ¿Cuántos eventos por segundo? ¿Cuántos gigabytes por hora?
- ¿Qué nivel de consistencia pueden esperar los ingenieros de datos de los datos de salida? Si se están ejecutando comprobaciones de calidad de datos contra los datos de salida, ¿con qué frecuencia se producen inconsistencias de datos: valores nulos donde no se esperan, formato deficiente, etc.?
- ¿Con qué frecuencia se producen errores?

- ¿Los datos contendrán duplicados?
- ¿Algunos valores de datos llegarán tarde, posiblemente mucho más tarde que otros mensajes producidos simultáneamente?
- ¿Cuál es el esquema de los datos ingeridos? ¿Los ingenieros de datos necesitarán unir varias tablas o incluso varios sistemas para obtener una imagen completa de los datos?
- Si hay cambios en el esquema (por ejemplo, se agrega una nueva columna), ¿cómo se aborda esto y se comunica a las partes interesadas en el resto del procesamiento?
- ¿Con qué frecuencia se deben extraer los datos del sistema de origen?
- Para los sistemas con estado (por ejemplo, una base de datos que rastrea la información de la cuenta del cliente), ¿se proporcionan los datos como instantáneas periódicas o eventos de actualización de la captura de datos de cambio (change data capture CDC)? ¿Cuál es la lógica de cómo se realizan los cambios y cómo se rastrean estos en la base de datos de origen?
- ¿Quién/qué es el proveedor de datos que transmitirá los datos para el consumo posterior?
- ¿Leer de una fuente de datos afectará su rendimiento?
- ¿El sistema de origen tiene dependencias anteriores? ¿Cuáles son las características de estos sistemas anteriores?
- ¿Existen controles de calidad de datos para verificar datos tardíos o faltantes?

2.1.2 Almacenamiento

Whether data is needed seconds, minutes, days, months, or years later, it must persist in storage until systems are ready to consume it for further processing and transmission.

El proceso de elección de almacenamiento es clave para el éxito en el resto del ciclo de vida de los datos, y también es una de las etapas más complicadas. Primero, las arquitecturas de datos en la nube a menudo se aprovechan de varias alternativas de almacenamiento. Segundo, pocas herramientas de almacenamiento de datos funcionan puramente como almacenamiento, y muchas admiten complejas queries de transformación de datos; incluso el almacenamiento basado en objetos

pueden admitir potentes capacidades de consulta, por ejemplo, Amazon S3 Select. Tercero, si bien el almacenamiento es una etapa del ciclo de vida de la ingeniería de datos, con frecuencia toca otras etapas, como la ingesta, la transformación y el serving.

El almacenamiento se extiende a lo largo de todo el ciclo de vida de la ingeniería de datos, y a menudo en múltiples lugares del pipeline de datos. En muchos sentidos, la forma en que se almacenan los datos impacta en cómo se utilizan en todas las etapas del ciclo de vida de la ingeniería de datos. Por ejemplo, los almacenes de datos en la nube pueden almacenar datos, procesar datos en pipelines y servirlos a los analistas.

Sistemas de Almacenamiento:

- Almacenamiento de Archivos (File Storage): Estos sistemas organizan los archivos en una estructura de árbol de directorios.

- Almacenamiento de Objetos (Object Storage): Contiene objetos de todos los tamaños y formas. El término "almacenamiento de objetos" puede ser confuso debido a los múltiples significados de "objeto" en informática. En este contexto, se refiere a una construcción especializada similar a un archivo. Puede ser cualquier tipo de archivo: TXT, CSV, JSON, imágenes, videos o audio. Amazon S3, Azure Blob Storage y Google Cloud Storage (GCS) son almacenes de objetos ampliamente utilizados. Además, muchos almacenes de datos en la nube (y un número creciente de bases de datos) utilizan el almacenamiento de objetos como su capa de almacenamiento, y los data lakes en la nube generalmente se basan en almacenes de objetos.

- Sistemas de Almacenamiento Basados en Caché y Memoria: Ofrecen una excelente latencia y velocidades de transferencia. Sin embargo, tradicional son extremadamente vulnerable a la pérdida de datos, ya que un corte de energía, incluso de un segundo, puede borrar los datos. Los sistemas de almacenamiento basados en RAM generalmente se centran en aplicaciones de almacenamiento en caché, presentando datos para un acceso rápido y un alto ancho de banda. Estos sistemas de caché ultrarrápidos son útiles cuando los ingenieros de datos necesitan servir datos con una latencia de recuperación ultrarrápida.

Abstracciones de Almacenamiento:

- Almacén de Datos (Data Warehouse): El término "almacén de datos" se refiere a plataformas tecnológicas (por ejemplo, Google BigQuery y Teradata), una arquitectura para la centralización de datos y un patrón organizativo dentro de una empresa.

- Lago de Datos (Data Lake): Originalmente concebido como un almacén masivo donde los datos se conservaban en forma bruta y sin procesar.

- Casa del Lago de Datos (Data Lakehouse): Una arquitectura que combina aspectos del almacén de datos y el lago de datos. Tal como se concibe generalmente, la "lakehouse" almacena datos en el almacenamiento de objetos al igual que un lago. Sin embargo, la "lakehouse" agrega a esta disposición características diseñadas para optimizar la gestión de datos y crear una experiencia de ingeniería similar a la de un almacén de datos. Esto significa un soporte robusto para tablas y esquemas y características para gestionar actualizaciones y eliminaciones incrementales. Las "lakehouses" típicamente también soportan el historial de la tabla y la reversión (rollback); esto se logra conservando versiones antiguas de archivos y metadatos.

- Plataformas de Datos (Data Platforms): Algo nuevo.

Estas son algunas preguntas claves al elegir un sistema de almacenamiento para data warehouse, un data lakehouse, una base de datos o un almacenamiento de objetos(object storage):

- ¿Es este sistema de almacenamiento compatible con las velocidades de lectura y escritura requeridas por la arquitectura?

- ¿El almacenamiento creará un cuello de botella para los procesos posteriores?

- ¿Entienden cómo funciona esta tecnología de almacenamiento? ¿Está utilizando el sistema de almacenamiento de manera óptima o cometiendo actos antinaturales?

- ¿Este sistema de almacenamiento manejará la prevista escala futura? Debe considerar todos los límites de capacidad en el sistema de almacenamiento: almacenamiento total disponible, tasa de operación de lectura, volumen de escritura, etc.

- ¿Los usuarios y procesos posteriores podrán recuperar datos en el service-level agreement (SLA) requerido?

- ¿Se está capturando metadatos sobre la evolución del esquema, los flujos de datos, el linaje de datos, etc.? Los metadatos tienen un impacto significativo en la utilidad de los datos. Los metadatos representan una inversión en el futuro, mejorando drásticamente la detectabilidad y el conocimiento institucional para agilizar futuros proyectos y cambios de arquitectura.

- ¿Es esta una solución de almacenamiento pura (almacenamiento de objetos) o admite patrones de consulta complejos (por ejemplo, un data warehouse en la nube)?
- ¿El sistema de almacenamiento es independiente del esquema (almacenamiento de objetos)? ¿Esquema flexible (Cassandra)? ¿Esquema forzado (un data warehouse en la nube)?
- ¿Cómo está rastreando los datos maestros, la calidad de los datos de registros dorados y el linaje de datos para la gobernanza de datos?
- ¿Cómo está manejando el cumplimiento normativo y la gobernanza de los datos? Por ejemplo, ¿puede almacenar sus datos en ciertas ubicaciones geográficas pero no en otras?

2.1.3 Ingesta

Usualmente, los sistemas de origen y la ingesta representan los cuellos de botella más significativos en el ciclo de vida de la ingeniería de datos. Los sistemas de origen están normalmente fuera de su control directo y podrían dejar de responder aleatoriamente o proveer datos de baja calidad. O, su servicio de ingesta de datos podría misteriosamente dejar de funcionar por muchas razones.

Al prepararse para diseñar o construir un sistema, aquí hay algunas preguntas primarias sobre la etapa de ingesta:

- ¿Cuáles son los casos de uso para los datos que estoy ingiriendo? ¿Puedo re-utilizar estos datos en lugar de crear múltiples versiones del mismo conjunto de datos?
- ¿Los sistemas que generan e ingieren estos datos son fiables, y los datos están disponibles cuando los necesito?
- ¿Cuál es el destino de los datos después de la ingesta?
- ¿Con qué frecuencia necesitaré acceder a los datos?
- ¿En qué volumen llegarán típicamente los datos?
- ¿En qué formato están los datos? ¿Pueden mis sistemas de almacenamiento y transformación posteriores manejar este formato?
- ¿Están los datos iniciales en buen estado para su inmediato uso posterior? Si es así, ¿por cuánto tiempo, y qué podría causar que se vuelvan inutilizables?

- Si los datos provienen de una fuente de *streaming*, ¿necesitan ser transformados antes de llegar a su destino? ¿Sería apropiada una transformación dentro del propio *stream*?

Lotes (*Batch*) vs. *Streaming*

Virtualmente todos los datos con los que tratamos son inherentemente *streaming*. Los datos son casi siempre producidos y actualizados continuamente en su origen. La ingesta por lotes es simplemente una forma especializada y conveniente de procesar este *stream* en grandes trozos—por ejemplo, manejar el valor de un día completo de datos en un solo lote.

La ingesta de *streaming* nos permite proveer datos a los sistemas posteriores—ya sean otras aplicaciones, bases de datos, o sistemas de analítica—de una forma continua y en tiempo real.

La elección depende en gran medida del caso de uso y las expectativas de puntualidad de los datos.

Las siguientes son algunas preguntas que debe hacerse al determinar si la ingesta de *streaming* es una opción apropiada sobre la ingesta por lotes:

- Si se ingieren los datos en tiempo real, ¿pueden los sistemas de almacenamiento posteriores manejar la tasa de flujo de datos?
- Es necesaria una ingesta de datos en tiempo real de mili-segundos? ¿O funcionaría un enfoque de micro-lotes, acumulando e ingiriendo datos, por ejemplo, cada minuto?
- ¿Cuáles son mis casos de uso para la ingesta de *streaming*? ¿Qué beneficios específicos obtengo al implementar *streaming*? Si obtengo datos en tiempo real, ¿qué acciones puedo tomar sobre esos datos que serían una mejora con respecto al lote?
- ¿Mi enfoque de priorizar *streaming* costará más en términos de tiempo, dinero, mantenimiento, tiempo de inactividad y costo de oportunidad que simplemente hacer lotes?
- ¿Son mi *pipeline* y sistema de *streaming* confiables y redundantes si falla la infraestructura?
- ¿Qué herramientas son las más apropiadas para el caso de uso? ¿Debería usar un servicio gestionado (Amazon Kinesis, Google Cloud Pub/Sub, Google Cloud Dataflow) o levantar mis propias instancias de Kafka, Flink, Spark, Pulsar, etc.? Si hago lo último, ¿quién lo administrará? ¿Cuáles son los costos y las ventajas y desventajas?

- Si estoy implementando un modelo de ML, ¿qué beneficios tengo con las predicciones en línea y posiblemente el entrenamiento continuo?
- ¿Estoy obteniendo datos de una instancia de producción en vivo? Si es así, ¿cuál es el impacto de mi proceso de ingesta en este sistema de origen?

Push vs. Pull

En el modelo de ingesta de datos de *push* un sistema de origen envía datos hacia un destino; ya sea una base de datos, un almacén de objetos o un sistema de archivos. En el modelo de *pull* los datos se recuperan del sistema de origen.

El proceso de extracción, transformación y carga (ETL) comúnmente utilizado en flujos de trabajo de ingesta orientados a lotes, La parte de extracción (E) de ETL aclara que estamos lidiando con un modelo de ingesta de tipo *pull*.

Con la ingesta de *streaming*, los datos evitan una base de datos *backend* y se envían (*push*) directamente a un punto final, típicamente con datos almacenados en búfer por una plataforma de *event-streaming*. Este patrón es útil con flotas de sensores IoT que emiten datos de sensores. En lugar de depender de una base de datos para mantener el estado actual, simplemente pensamos en cada lectura registrada como un evento. Este patrón también está creciendo en popularidad en aplicaciones de software, ya que simplifica el procesamiento en tiempo real, permite a los desarrolladores de aplicaciones adaptar sus mensajes para análisis posteriores, y simplifica enormemente la vida de los ingenieros de datos.

2.1.4 Transformación

La siguiente etapa en el ciclo de vida de la ingeniería de datos es la transformación, lo que significa que los datos deben ser modificados desde su forma original a algo útil para los casos de uso posteriores.
Típicamente, la etapa de transformación es donde los datos comienzan a crear valor para el consumo por parte de los usuarios posteriores.
Inmediatamente después de la ingesta, las transformaciones básicas mapean los datos a los tipos correctos (cambiando los datos de tipo cadena ingeridos a tipos numéricos y de fecha, por ejemplo), colocando los registros en formatos estándar y eliminando los incorrectos.
Las etapas posteriores de la transformación pueden transformar el esquema de datos y aplicar normalización.
En etapas posteriores, podemos aplicar agregaciones a gran escala para la elaboración de informes o la *featurización* de datos para los procesos de aprendizaje automático (ML).

Consideraciones clave:

- ¿Cuál es el costo y el retorno de la inversión (ROI) de la transformación? ¿Cuál es el valor comercial asociado?
- ¿Es la transformación tan simple y auto-aislada como sea posible?
- ¿Qué reglas de negocio las transformaciones soportan?

Se pueden transformar los datos en lotes o en *streaming* en tiempo real. Las transformaciones por lotes son abrumadoramente populares, pero dada la creciente popularidad de las soluciones de procesamiento en *streaming* y el aumento general en la cantidad de datos en *streaming*, se espera que la popularidad de las transformaciones en *streaming* continúe creciendo.

La transformación a menudo está entrelazada con otras fases del ciclo de vida. Típicamente, los datos se transforman en los sistemas de origen o en tiempo real durante la ingesta. Por ejemplo, un sistema de origen puede agregar una marca de tiempo de evento a un registro antes de reenviarlo a un proceso de ingesta. O un registro dentro de un pipeline de *streaming* puede ser "enriquecido" con campos y cálculos adicionales antes de ser enviado a un data warehouse.

Lógica de Negocio La lógica de negocio es un importante impulsor de la transformación de datos, a menudo en el modelado de datos. Los datos traducen la lógica de negocio en elementos reutilizables.

El *featuring* de datos para ML es otro proceso de transformación de datos. Tiene como objetivo extraer y mejorar las características de los datos que son útiles para el entrenamiento de modelos de ML. Puede ser un arte oscuro, que combina el conocimiento del dominio (para identificar qué características podrían ser importantes para la predicción) con una amplia experiencia en ciencia de datos. El punto principal es que una vez que los científicos de datos determinan cómo *featurize* los datos, los procesos de *featuring* pueden ser automatizados por los ingenieros de datos en la etapa de transformación de un pipeline de datos.

Algunos de los elementos fundamentales de la etapa de transformación son: Preparación de datos, manipulación y limpieza de datos; consultas, modelado de datos.

Puede ocurrir en lotes o en stream.

Transformaciones por Lotes (Batch) se ejecutan en fragmentos discretos de datos, y pueden programarse a intervalos fijos (e.g., diario, por hora, o cada 15 minutos) para soportar informes, análisis y modelos de ML.
Un patrón de transformación extendido desde los inicios de las bases de datos relacionales es el ETL por lotes. El ETL tradicional se basa en un sistema de transformación externo para extraer, transformar y limpiar los datos, preparándolos para un esquema objetivo; como un almacén de datos, donde se pueden realizar

análisis de negocio. La fase de extracción solía ser un cuello de botella importante, limitando la velocidad a la que se podían extraer los datos.
Una evolución popular del ETL es el ELT. A medida que los almacenes de datos han crecido en rendimiento y capacidad de almacenamiento, se ha vuelto común simplemente extraer los datos en bruto de un sistema fuente, importarlos al almacén de datos con una transformación mínima, y luego limpiarlos y transformarlos directamente en el sistema del almacén. Una segunda noción de ELT, ligeramente diferente, se popularizó con la aparición de los data lakes. En esta versión, los datos no se transforman al cargarlos. De hecho, se pueden cargar cantidades masivas de datos sin preparación ni plan alguno. La suposición es que el paso de transformación ocurrirá en un momento futuro indeterminado.
La data wrangling toma datos desordenados y mal formados y los convierte en datos útiles y limpios. Generalmente, este es un proceso de transformación por lotes.

Transformaciones en Streaming los datos se procesan continuamente a medida que llegan.
Change Data Capture (CDC): La captura de datos modificados (CDC) es un método para extraer cada evento de cambio (inserción, actualización, eliminación) que ocurre en una base de datos. La CDC se utiliza con frecuencia para replicar entre bases de datos casi en tiempo real o para crear un flujo de eventos para procesamientos posteriores.
El enfoque del fast-follower: Las bases de datos de producción generalmente no están equipadas para manejar cargas de trabajo de producción y ejecutar simultáneamente grandes escaneos analíticos sobre cantidades significativas de datos. Ejecutar tales consultas puede ralentizar la aplicación de producción o incluso provocar que se bloquee. Uno de los patrones de consulta de streaming más antiguos consiste simplemente en consultar la base de datos de análisis, recuperando resultados estadísticos y agregaciones con un ligero retraso con respecto a la base de datos de producción.

2.1.5 Serving

Ahora que los datos han sido ingeridos, almacenados y transformados en estructuras coherentes y útiles, es hora de obtener valor de los datos. "Obtener valor" de los datos significa diferentes cosas para diferentes usuarios.

Analítica

La analítica es el núcleo de la mayoría de los esfuerzos de datos. Una vez que sus datos están almacenados y transformados, está listo para generar informes o *dashboards* y realizar análisis *ad hoc* sobre los datos. Mientras que la mayor parte de la analítica solía abarcar la inteligencia empresarial (BI), ahora incluye otras facetas como la analítica operativa y la analítica integrada.

La Inteligencia Empresarial (BI) aprovecha los datos recopilados para comprender el rendimiento pasado y presente de una empresa.
La Analítica Operativa se centra en la información en tiempo real sobre las operaciones de negocio, impulsando la acción inmediata. Piense en vistas de inventario en vivo o *dashboards* en tiempo real que monitorean la salud del sitio web o de la aplicación. A diferencia de la BI, la analítica operativa enfatiza el presente, enfocándose menos en las tendencias históricas.
La Analítica Integrada (Embedded Analytics), o analítica orientada al cliente, es la práctica de integrar capacidades analíticas directamente en un producto o plataforma de *software*. Si bien aparentemente similar a la BI, la analítica integrada presenta desafíos únicos. Servir analítica a una gran base de clientes requiere tasas de solicitud significativamente mayores, lo que exige sistemas analíticos escalables y robustos. Lo más crítico es que el control de acceso se vuelve primordial. Cada cliente debe ver solo sus datos, y cualquier fuga de datos es una grave violación de la confianza con consecuencias potencialmente catastróficas.

Machine Learning

Algunas consideraciones para la fase de servicio de datos específicas para ML:

- ¿Son los datos de calidad suficiente para realizar una ingeniería de features fiable? Los requisitos y las evaluaciones de calidad se desarrollan en estrecha colaboración con los equipos que consumen los datos.

- ¿Son los datos localizables? ¿Pueden los científicos de datos y los ingenieros de ML encontrar fácilmente datos valiosos?

- ¿Dónde están los límites técnicos y organizacionales entre la ingeniería de datos y la ingeniería de ML? Esta cuestión organizativa tiene importantes implicaciones arquitectónicas.

- ¿El conjunto de datos representa adecuadamente el ground truth? ¿Está sesgado injustamente?

ETL Inversa

El ETL inverso toma los datos procesados del lado de salida del ciclo de vida de la ingeniería de datos y los devuelve a los sistemas de origen. El ETL inverso nos permite tomar analíticas, modelos puntuados, etc., y devolverlos a los sistemas de producción o plataformas SaaS.

2.1.6 Principales Corrientes Subyacentes en el Ciclo de Vida de la Ingeniería de Datos

Seguridad

La seguridad debe ser una prioridad para los ingenieros de datos. Deben comprender tanto la seguridad de los datos como la seguridad del acceso, ejerciendo el principio de privilegio mínimo. El principio de privilegio mínimo significa dar a un usuario o sistema acceso solo a los datos y recursos esenciales para realizar una función prevista.

La seguridad de los datos también se trata de sincronización: proporcionar acceso a los datos exactamente a las personas y sistemas que necesitan acceder a ellos y solo durante el tiempo necesario para realizar su trabajo.

Gestión de Datos

Los ingenieros de datos gestionan el ciclo de vida de los datos, y la gestión de datos abarca el conjunto de mejores prácticas que los ingenieros de datos utilizarán para llevar a cabo esta tarea, tanto técnica como estratégicamente.

DataOps

DataOps mapea las mejores prácticas de la metodología Agile, DevOps y el control estadístico de procesos (SPC) a los datos. Mientras que DevOps tiene como objetivo mejorar la publicación y la calidad de los productos de *software*, DataOps hace lo mismo para los productos de datos.

DataOps tiene tres elementos técnicos centrales: automatización, monitorización y observabilidad, y respuesta a incidentes.

La automatización de DataOps tiene un marco de trabajo y un flujo de trabajo similares a los de DevOps, que consisten en la gestión de cambios (control de versiones de entorno, código y datos), integración continua/despliegue continuo (CI/CD) y configuración como código.

Data Architecture

Una arquitectura de datos define los planos del sistema de datos de una organización. Es usualmente trabajo para el arquitecto de datos. Típicamente es un rol separado del ingeniero de datos, pero este debe implementar los diseños y proveer feedback

Orquestación

La orquestación es el proceso de coordinar muchos trabajos para que se ejecuten de la manera más rápida y eficiente posible en una cadencia programada. Un motor de orquestación construye metadatos sobre las dependencias de los trabajos, generalmente en la forma de un grafo acíclico dirigido (DAG). El DAG se puede ejecutar una vez o programarse para que se ejecute a un intervalo fijo de diariamente, semanalmente, cada hora, cada cinco minutos, etc.

Ingeniería de Software

Algunas áreas comunes de la ingeniería de software que se aplican al ciclo de vida de la ingeniería de datos.

Código de Procesamiento de Datos Central (Core Data Processing Code): Escribir código eficiente y comprobable (Spark, SQL, etc.) para la ingesta, transformación y servicio de datos es fundamental para todas las etapas del ciclo de vida de la ingeniería de datos. Las metodologías de prueba adecuadas (pruebas unitarias, de regresión, de integración, de extremo a extremo y de humo) garantizan la calidad y confiabilidad de los datos.

Desarrollo de Frameworks de Código Abierto (Open Source Frameworks): Los ingenieros de datos a menudo contribuyen y aprovechan las herramientas de código abierto, por lo que comprender su desarrollo y contribuir con mejoras es valioso.

Streaming (Procesamiento en Flujo Continuo): El procesamiento de datos en tiempo real requiere habilidades y herramientas especializadas. Las tecnologías de streaming son cada vez más importantes en todas las etapas del ciclo de vida de la ingeniería de datos.

Infraestructura como Código (IaC - Infrastructure as Code): La gestión de la infraestructura (servidores, bases de datos, etc.) a través de código garantiza implementaciones coherentes y repetibles, lo cual es fundamental para las prácticas de DataOps en todo el ciclo de vida de la ingeniería de datos.

Pipelines como Código (Pipelines as Code): Definir flujos de trabajo de datos y dependencias mediante código permite la orquestación automatizada. Los ingenieros de datos usan código (típicamente Python) para declarar las tareas de

datos y las dependencias entre ellas. El motor de orquestación interpreta estas instrucciones para ejecutar los pasos utilizando los recursos disponibles.

Desarrollo de Código Personalizado: Los ingenieros de datos a menudo necesitan escribir código personalizado para manejar escenarios específicos, integrar sistemas y resolver problemas más allá de las capacidades de las herramientas estándar. Son esenciales las sólidas habilidades de ingeniería de software (uso de API, transformación de datos, manejo de excepciones, etc).

3

Conferencia 3

Tecnologías de la Ingeniería de Datos: Almacenamiento e Ingestión

Abordaremos conceptos generales y haremos un recorrido por las principales tecnologías y herramientas utilizadas en la industria. El objetivo es que comprendan el panorama general, se familiaricen con los casos de uso clave y, sobre todo, identifiquen qué tecnologías merecen su atención.

3.1 Almacenamiento

3.1.1 Tipos de almacenamiento de datos:

Bases de Datos Relacionales

Una base de datos relacional es una colección de información que organiza los datos en tablas (o relaciones). Los datos se almacenan en filas y columnascon una clave única que identifica cada fila. Generalmente, cada tabla/relación representa un "tipo de entidad" (como cliente o producto). Las filas representan instancias de ese tipo de entidad (como "Lee" o "silla") y las columnas representan valores atribuidos a esa instancia (como dirección o precio). Se representan relaciones como conexiones lógicas entre tablas, establecidas en función de sus interacciones. Cada tabla tiene un esquema predefinido que fuerza a todos los elementos de la tabla a cumplirlos. Utilizan SQL, o variaciones de SQL como lenguaje estándar, que permite crear consultas complejas que abarquen varias tablas.

Escalan verticalmente, o sea, para mejorar el rendimiento hay que añadir más recursos (CPU/RAM) al servidor. El enfoque de escalado vertical aumenta la capacidad de una sola máquina incrementando los recursos en el mismo servidor lógico. Esto implica añadir recursos como memoria, almacenamiento y potencia de procesamiento al software existente, mejorando así su rendimiento.

Garantizan el cumplimiento del modelo relacional mediante el concepto de ACID.

En los sistemas de bases de datos, ACID (Atomicidad, Consistencia, Aislamiento(isolation), Durabilidad) se refiere a un conjunto estándar de propiedades que garantizan que las transacciones de la base de datos se procesen de forma fiable. ACID se preocupa especialmente de cómo una base de datos se recupera de cualquier fallo que pueda ocurrir durante el procesamiento de una transacción. Un DBMS compatible con ACID asegura que los datos en la base de datos permanezcan precisos y consistentes a pesar de cualquier fallo de este tipo.

Atomicidad significa que se garantiza que o bien toda la transacción tiene éxito, o bien ninguna parte de ella lo tiene. No se obtiene una parte exitosa y otra que no lo es. Si una parte de la transacción falla, toda la transacción falla. Con la atomicidad, es "todo o nada".

Consistencia asegura que se garantiza que todos los datos serán consistentes. Todos los datos serán válidos de acuerdo con todas las reglas definidas, incluyendo cualquier restricción, cascada y disparador (trigger) que se haya aplicado en la base de datos.

Aislamiento garantiza que todas las transacciones ocurrirán de forma aislada. Ninguna transacción se verá afectada por ninguna otra transacción. Por lo tanto, una transacción no puede leer datos de ninguna otra transacción que aún no se haya completado.

Durabilidad significa que, una vez que una transacción se ha confirmado (committed), permanecerá en el sistema, incluso si hay una caída del sistema inmediatamente después de la transacción. Cualquier cambio de la transacción debe almacenarse permanentemente. Si el sistema le dice al usuario que la transacción ha tenido éxito, la transacción debe, de hecho, haber tenido éxito.

RDBMS (Relational Database Management System)=SGBD(R)

- Oracle Database: Su primera versión data a 1979. Fue el primer RDBMS SQL disponible comercialmente.
- MySQL: MySQL se ofrece en dos ediciones diferentes: MySQL Community Server, de código abierto, y Enterprise Server, de propietario. Aunque comenzó como una alternativa de gama baja a bases de datos propietarias más potentes, ha evolucionado gradualmente para dar soporte a necesidades

de mayor escala. Todavía se utiliza más comúnmente en implementaciones de un solo servidor, de pequeña a mediana escala. P o como un servidor de bases de datos independiente. Gran parte del atractivo de MySQL se origina en su relativa simplicidad y facilidad de uso, que se ve facilitada por un ecosistema de herramientas de código abierto. En el rango medio, MySQL se puede escalar desplegándolo en hardware más potente, como un servidor multiprocesador con gigabytes de memoria.

- SQLite: motor de base de datos relacional gratuito y de código abierto. No es una aplicación independiente; más bien, es una biblioteca que los desarrolladores de software incrustan en sus aplicaciones. SQLite fue diseñado para permitir que el programa se opere sin instalar un sistema de gestión de bases de datos o requerir un administrador de bases de datos. Viene instalado default en la mayoría de los sistemas operativos y navegadores (browsers). Debido al diseño server-less, las aplicaciones SQLite requieren menos configuración que las bases de datos cliente-servidor. SQLite se conoce como de "zero-configuration"("configuración cero") porque las tareas de configuración, como la gestión de servicios, los scripts de inicio y el control de acceso basado en contraseñas o GRANT, son innecesarias. SQLite almacena toda la base de datos, que consta de definiciones, tablas, índices y datos, como un único archivo multiplataforma, lo que permite que varios procesos o subprocesos accedan a la misma base de datos simultáneamente.

- PostgreSQL: es uno de los sistemas de gestión de bases de datos objeto-relacionales de propósito general más avanzados y es de código abierto. Sus características más destacadas incluyen: Herencia de tablas, Replicación asíncrona, Capacidad definir nuevos tipos, así como soporte de muchos tipos que incluyen: Numéricos de precisión arbitraria, Arrays (de longitud variable y de cualquier tipo de datos hasta 1 GB de tamaño), Direcciones IPv4 e IPv6, Identificador único universal (UUID), JSON y un JSONB binario más rápido.
  Se pueden crear e instalar extensiones o plugins que funcionan como características integradas. A lo largo de los años, se ha desarrollado un rico ecosistema de extensiones que cubren desde la optimización del rendimiento hasta tipos de datos especializados. Ejemplos son:
  PostGIS: introduce tipos de datos adicionales para la gestión geo-espacial.
  pgcrypto: agrega funciones criptográficas para el cifrado, el hashing y más.
  pgvector: Agrega soporte para operaciones vectoriales en Postgres, lo que permite la búsqueda de similitud, la búsqueda del vecino más cercano y más.

Bases de Datos No Relacionale (NoSQL)

NoSQL es un enfoque al diseño de bases de datos que se centra en proporcionar un mecanismo para el almacenamiento y la recuperación de datos que se modelan utilizando medios distintos a las relaciones tabulares empleadas en las bases de datos relacionales.
Las motivaciones para este enfoque incluyen la simplicidad del diseño, un escalado "horizontal" más sencillo a clústeres de máquinas, un control más preciso sobre la disponibilidad y la limitación del desajuste de impedancia objeto-relacional. Las estructuras de datos utilizadas por las bases de datos NoSQL permiten que algunas operaciones sean más rápidas y se consideran "más flexibles" que las tablas de las bases de datos relacionales.
Se clasifican en cuatro categorías principales:

- Key-value stores(clave-valor): utilizan el array asociativo (también llamado mapa o diccionario) como su modelo de datos fundamental. En este modelo, los datos se representan como una colección de pares clave-valor, de tal manera que cada clave posible aparece como máximo una vez en la colección. Ejemplos: Redis, Couchbase, DynamoDB.

- Column-family stores(de columnas): Utiliza tablas, filas y columnas, pero a diferencia de una base de datos relacional, los nombres y el formato de las columnas pueden variar de fila a fila en la misma tabla. Un almacén de columnas anchas se puede interpretar como un store clave-valor bidimensional. Ejemplo: Cassandra.

- Document databases (de documentos): El concepto central de un almacén de documentos es el de un "documento". Las codificaciones en uso incluyen XML, YAML y JSON, y formas binarias como BSON. Se accede a los documentos en la base de datos a través de una clave única que representa ese documento. Las colecciones podrían considerarse análogas a las tablas y los documentos análogos a los registros(filas). Pero son diferentes: cada registro en una tabla tiene la misma secuencia de campos, mientras que los documentos en una colección pueden tener campos que son completamente diferentes. Ejemplo: MongoDB.

- Graph databases (de grafos): Las bases de datos de grafos están diseñadas para datos cuyas relaciones están bien representadas como un grafo, que consiste en elementos conectados por un número finito de relaciones. Ejemplos de datos incluyen relaciones sociales, enlaces de transporte público, mapas de carreteras, topologías de redes, etc. Ejemplo: Neo4j.

Actualmente, existen dos tipos principales de sistemas de procesamiento de datos críticos: el Procesamiento de Transacciones en Línea (OLTP) y el Procesamiento Analítico en Línea (OLAP). Cada uno tiene un propósito diferente y, técnicamente, deberían estar separados en cualquier organización.
OLTP es prácticamente lo que su nombre indica: una base de datos responsable del procesamiento de transacciones operativas. Debe ser rápido, dinámico y tener capacidad de respuesta ante fallos en la base de datos con un plan de respaldo.
OLAP es diferente: su propósito es guardar datos históricos y mantener los procesos de Extracción, Transformación y Carga (ETL) que se utilizan para el análisis de datos. El sistema OLAP es fundamental para las decisiones comerciales críticas, ya que ofrece datos relacionados con el rendimiento diario y la estabilidad organizacional a largo plazo.

Data Warehouse

Los Data Warehouse son almacenes de datos son repositorios centrales de datos integrados, provenientes de fuentes diversas. Almacenan datos actuales e históricos organizados para optimizar el análisis de datos, la generación de informes y, sobre todo, la obtención de información clave a partir de la integración de datos. Son bases de datos especializadas diseñadas para almacenar y analizar grandes volúmenes de datos estructurados y semiestructurados para fines de inteligencia empresarial (BI) y analítica.

- PostgreSQL: Una característica excelente de PostgreSQL es su capacidad para ser utilizado tanto para OLTP como para OLAP. Esto facilita que las bases de datos que utilizan OLAP para almacenar los datos se comuniquen con las bases de datos que utilizan OLTP para crear los datos más recientes. Esta puede ser la razón principal por la que PostgreSQL es tan popular.

- Snowflake: Fundada en 2012, Snowflake es un Data Warehouse basado en la nube, conocido por su escalabilidad y flexibilidad. La arquitectura nativa de la nube de Snowflake aprovecha los beneficios de proveedores como AWS, Azure y Google Cloud. La plataforma ofrece funciones de seguridad integradas, incluyendo cifrado y control de acceso, lo que la hace adecuada para organizaciones con necesidades estrictas de seguridad y cumplimiento normativo. En general, Snowflake proporciona una solución robusta para almacenar, gestionar y analizar grandes conjuntos de datos en la nube.

- Amazon Redshift: Amazon Redshift es un servicio de almacenamiento de datos rápido y totalmente administrado en la nube, que permite a las empresas ejecutar consultas analíticas complejas sobre grandes volúmenes de

datos, minimizando así los retrasos y garantizando un sólido apoyo para la toma de decisiones en todas las organizaciones. Fue lanzado en 2013, creado para solucionar los problemas asociados con el almacenamiento de datos tradicional en las instalaciones, como la escalabilidad, el costo y la complejidad. Redshift se integra perfectamente con Amazon S3, Amazon RDS, AWS Glue y mucho más para crear un ecosistema de datos.

- Google BigQuery: Google BigQuery es un Data Warehouse nativo de la nube. Es un almacén de datos en la nube sin servidor, altamente escalable y rentable que permite realizar consultas súper rápidas a escala de petabytes utilizando la potencia de procesamiento de la infraestructura de Google. Su arquitectura sin servidor le permite operar a escala y velocidad para proporcionar análisis SQL increíblemente rápidos sobre grandes conjuntos de datos.

Data Lake

Repositorios centralizados que almacenan grandes cantidades de datos sin procesar en su formato nativo, lo que permite a las organizaciones realizar análisis avanzados, aprendizaje automático y exploración de datos. Tecnologías como Apache Hadoop, Apache Spark y AWS Glue se utilizan comúnmente para construir y administrar lagos de datos.

3.1.2 Tipos de almacenamiento de datos:

Almacenamiento en Archivos(File Storage)

Dada la siguiente información como punto de partida, y manteniendo el estilo de escritura, profundiza en la explicación de los tipos de almacenamiento.

Almacenamiento en Bloques(Block Storage)

Divide los datos en bloques de tamaño fijo y los almacena sin metadatos adicionales. Cada bloque tiene una dirección única y puede ser gestionado independientemente, lo que lo hace ideal para bases de datos y aplicaciones de alto rendimiento que requieren baja latencia.

Características clave:

Bajo nivel: Los bloques son manejados directamente por el sistema de almacenamiento.

Alto rendimiento: Ideal para bases de datos (ej. Oracle, SQL Server) y máquinas virtuales.

Flexibilidad: Permite configuraciones avanzadas como RAID y snapshots.

Almacenamiento de Objetos(Object Storage)

El almacenamiento de objetos es una arquitectura de almacenamiento de datos en la que los datos se almacenan y gestionan como unidades autocontenidas llamadas objetos. Cada objeto contiene una clave, datos y metadatos opcionales. Dado que el almacenamiento de archivos y bloques utiliza jerarquías, el acceso a los datos se ralentiza a medida que los almacenes de datos crecen desde gigabytes y terabytes hasta petabytes e incluso más.
Cada objeto se almacena con sus metadatos, que pueden ser bastante detallados. Pueden incluir información como políticas específicas de privacidad y seguridad, reglas de acceso e incluso especificaciones, por ejemplo, sobre dónde se grabó un videoclip o quién creó los datos.
Cada unidad tiene un identificador único o clave, que permite encontrarlas sin importar dónde estén almacenadas en un sistema distribuido. Los objetos se almacenan en un único pool sin una jerarquía de carpetas o directorios.

Ejemplos:

- Amazon S3
- Azure Blob
- Google Cloud Storage

Los sistemas de archivos distribuidos como Hadoop Distributed File System (HDFS) y Google File System (GFS) proporcionan soluciones de almacenamiento escalables y tolerantes a fallos para entornos de computación distribuida. Están optimizados para almacenar y procesar grandes conjuntos de datos a través de múltiples nodos en un clúster de computación distribuida, soportando el procesamiento de datos en paralelo y la tolerancia a fallos.

3.2 Ingesta

La ingesta de datos es el proceso de mover datos (especialmente datos no estructurados) desde una o más fuentes a un sistema de almacenamiento para su posterior procesamiento y análisis.
Es la primera etapa en un pipeline de ingeniería de datos donde los datos provenientes de varias fuentes comienzan su recorrido.

3.2.1 APIs

Una de las formas más frecuentes de ingestión de datos es a través de Application Programming Interface. En general, se accede a las APIs a través de la web utilizando una URL. Dentro de la dirección web, se especifica la información que se desea. Para saber cómo formatear la dirección web, es necesario leer la documentación de la API. Algunas APIs también requieren que envíes credenciales de inicio de sesión como parte de tu solicitud. La biblioteca 'requests' de Python hace que trabajar con APIs sea relativamente sencillo.

A menudo se considera que las ingestas de datos mediante APIs son más rápidas y escalables en ciertas situaciones, y pueden soportar cambios variables en los atributos. Esta funcionalidad puede funcionar con procesamiento en tiempo real o por lotes, lo que la convierte en una herramienta valiosa para muchas industrias diferentes.

Postman

Herramientas de ingesta de datos

- Airbyte: Es una herramienta de ingesta de datos de código abierto que se centra en la extracción y carga de datos, desarrollada para le proceso de ETL. Facilita la configuración de pipelines y mantiene un flujo seguro a lo largo de todo el pipeline. Puede proporcionar acceso tanto a datos brutos como normalizados e integra más de 120 conectores de datos.

- Amazon Kinesis: Ayuda en la ingesta de datos en tiempo real de diversas fuentes, y a procesar y analizar los datos a medida que llegan. Además, ofrece diferentes capacidades, como Kinesis Data Streams, Kinesis Data Firehose y más, para la ingesta de datos en streaming a cualquier escala de forma rentable.

- Apache Kafka: Es una plataforma de streaming de eventos distribuida de código abierto. El streaming de eventos captura datos en tiempo real de fuentes de eventos como dispositivos móviles, sensores, bases de datos, servicios en la nube y aplicaciones de software. Kafka almacena los datos de forma duradera para su posterior recuperación para procesamiento y análisis. Sus otras capacidades principales incluyen alto rendimiento, escalabilidad y alta disponibilidad.

- Apache NiFi: Herramienta visual para automatizar los flujos de datos entre sistemas.

4

Integración de Datos

Data integration refers to the process of combining data from multiple sources into a unified, coherent format that can be used for various analytical, operational, and decision-making purposes This process is essential in today's digital landscape, where organizations gather data from a wide range of sources, including databases, apps, spreadsheets, cloud services, APIs, and more

Solutions for combining data from disparate sources into a unified view, often involving middleware or virtual data layers.

5

Conferencia 4

Tecnologías de la Ingeniería de Datos: Transformación y Servicio

5.1 Transformación

Data transformation involves converting raw data into a format suitable for analysis and consumption. This stage is crucial for adding value to data and making it useful for downstream processes.

Key Operations in Data Transformation

Normalization: Modifying data scales, such as scaling values from 0 to 1, to enable comparisons.

Standardization: Transforming data to have a unit variance and zero mean, often required before using machine learning methods.

Encoding: Transforming categorical data into numerical representations using label or one-hot encoding.

Discretization: Converting continuous data into discrete bins, facilitating analysis and enhancing model performance.

Attribute Generation: Creating new variables from existing data, such as deriving an 'age' variable from a date of birth.

Revising: Ensuring that the data supports its intended usage by deleting duplicates, standardizing the data collection, and purifying it.

Manipulation: Creating new values from existing ones or changing the state of data through

Techniques and Tools

Programmatic Transformation: Automating transformation operations using scripts or programming languages such as Python, R, or SQL.

ETL Tools: Tools for extracting, transforming, and loading data (ETL) are designed to handle complex data transformation requirements in large-scale environments.

Normalization/Standardization: Scikit-learn in Python provides functions for normalization and standardization such as MinMaxScaler and StandardScaler.

Encoding Categorical Variables: Pandas library in Python provides the $get_dummies function foro$
$hotencoding. For labelencoding, LabelEncoder is provided by Scikit-learn.$

Imputation: Missing values in the dataset are filled using statistical methods like the fillna method in the Pandas Library. Additionally, missing data can be imputed using mean, median, or mode using Scikit-learn's SimpleImputer.

Feature Engineering: New features are developed by combining old ones. Pandas functions such as apply, map, and transform are used to generate new features

Tools:

- dbt (data build tool)
- Talend: ETL/ELT platform for integrating and transforming data.

Stream Processing

- Apache Flink: Framework for stateful computations over unbounded/bounded data streams.
- Apache Kafka Streams: Library for building real-time stream processing applications.

Batch Processing

- Apache Spark: Engine for large-scale batch data processing and analytics.
- Apache Hadoop : Framework for distributed processing of big datasets.

Transformation tools:

Data Processing, Data Transformation, Data Serialization Formatting, API Management

– YAML engineering - Airflow (orchestration) - Prefect - Terraform and Pulumi are IaC tools - Docker and Kubernetes are popular containerization tools. - Prefect and Luigi are used for automating and managing complex data workflows. - dbt (data build tool) and Metabase are used for data transformation and analytics.

5.2 Generación

5.3 Almacenamiento

Containerization Tools: - Docker - Kubernetes Data Warehouse Tools: - Snowflake - PostgreSQL - BigQuery - Amazon Redshift

- Apache Kafka Data Lakes: - Amazon S3 - Azure Data Lake - Hadoop Distributed File System (HDFS)

5.4 Ingesta

- Segment - Fivetran: is a comprehensive ETL tool - Stitch - Apache Kafka - Airbyte

5.5 Transformación

- dbt (data build tool)

Batch: - Apache Spark - Apache Hadoop

Streaming: - Apache Kafka - Apache Flink

5.6 Serving

Analytics Engineering Tools - Metabase - Power BI Visualization: - Tableau - Looker

Bibliografía:

- El libro "Fundamentals of Data Engineering" va de : the data engineering lifecycle: data generation, storage, ingestion, transformation, and serving. Since the dawn of data, we've seen the rise and fall of innumerable specific technologies and vendor products, but the data engineering lifecycle stages have remained essentially unchanged.