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 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.

[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.

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 reutilizar 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 agrega-

ciones 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

Conferencia 4

Tecnologías de la Ingeniería de Datos: Transformación y Servicio

4.1 Transformación

La transformación de datos es el proceso de convertir, limpiar y estructurar los datos brutos en un formato adecuado para análisis y modelado. Esta etapa es crucial en los pipelines de datos ya que mejora la calidad, consistencia y utilidad de los datos. Preparándolos para su consumo en aplicaciones de business intelligence y machine learning.

Resaltar que ninguna simplemente se modifican los datos originales, sino que se crea un nuevo espacio para contener los datos "limpios y procesados".

4.1.1 Ejemplos de operaciones de transformación de datos:

Limpieza

La limpieza de datos es el proceso de detectar y rectificar errores e inconsistencias en los datos. Es parte esencial del procesamiento de los datos, y casi en el 100% de los casos es necesario, a diferencia de otras operaciones que veremos a continuación.

Ejemplos:

- Remover datos duplicados: Registros idénticos (por fallas en la generación o ingestión)

```
ANTES:                          DESPUÉS:
ID  Producto  Valor              ID  Producto  Valor
1   Laptop    $1000              1   Laptop    $1000
1   Laptop    $1000      →       2   Mouse     $20
2   Mouse     $20
```

- Detectar valores atípicos: Ejemplo en temperaturas (°C):

```
Datos: [22, 23, 21, 25, 19, 28, 31, -273, 24]
Outlier: -273 (posiblemente error de sensor)
```

- Valores inválidos para el tipo: Ejemplo: Fechas con formato incorrecto o fuera de rango lógico.

```
    ANTES:                                DESPUÉS:
Paciente  Fecha_Nacimiento          Paciente  Fecha_Nacimiento
A         1990-05-15                A         1990-05-15
B         2025-13-02          →     B         NaN (fecha futura/mes inválido)
C         15/07/1985                C         1985-07-15 (formato apropiado)
```

- Atributo con valores de tipo incorrecto: Símbolos mezclados en campos numéricos.

```
ANTES:                              DESPUÉS:
Producto  Precio                    Producto  Precio
X         $120.50                   X         120.50
Y         95,99€            →       Y         95.99
Z         "1,200"                   Z         1200.00
```

Algunas herramientas que ayudan a la visualización y limpieza de datos incluyen pandas(Python),

Normalización

Proceso de escalar valores numéricos a un rango común (generalmente [0,1]) sin distorsionar las diferencias. Es relevante cuando los datos serán utilizados por algún algoritmo sensible a la escala; así todos los features tienen la misma escala y no se induce un sesgo en ese sentido (KNN, Redes Neuronales).

Min-Max Normalization: transforms the original data linearly.

$$X_{\text{norm}} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

Ejemplos:
Unificación de lecturas de sensores con diferentes rangos:

```
ANTES (Raw values):                 DESPUÉS (Normalizado):
Sensor_A: [200, 400, 600]           Sensor_A: [0.0, 0.5, 1.0]
Sensor_B: [10, 20, 30]        →     Sensor_B: [0.0, 0.5, 1.0]
(Rango A: 200-600)                  (Ambos en [0,1])
(Rango B: 10-30)
```

La biblioteca Scikit-learn proveé herramientas para el escalado/normalización automático.

Estandarización

Transformación similiar a la Normalización, que ajusta los datos para tener media 0 y desviación estándar 1 (Z-score).
Z-Score Normalization: zero-mean normalization.

$$Z = \frac{X - \mu}{\sigma}$$

Ejemplos:
Comparar puntuaciones en distintas escalas:

```
ANTES (Puntajes crudos):            DESPUÉS (Z-scores):
FICO: [650, 720, 580]               FICO: [0.23, 1.12, -1.35]
Vantage: [550, 600, 450]      →     Vantage: [0.23, 1.12, -1.35]
(mu=650, sigma=62.5)                (Comparables directamente)
(mu=550, sigma=44.7)
```

Herramientas: Scikit-learn (StandardScaler), TensorFlow Transform.

Encoding

Conversión de datos categóricos a formato numérico para procesamiento algorítmico.
Los datos pueden ser binarios excluyentes, ordinales o nominales (sin orden).
Los datos binarios pueden pasar por un encoding que los transforma en 0 y 1.
Para los datos ordinales usualmente se utiliza Label Encoding o Ordinal Encoding. Sin embargo los datos categóricos nominales, no se puede asumir que tienen algún orden y forzar uno inevitablemente introducirá un sesgo. En ese caso se sugiere utilizar One-Hot Encoding; que consiste en poner cada valor posible como una columna y asignar 0 o 1 indicando si el registro tenía esa clasificación.
Otra forma de codificar es Frequency Encoding. Que transforma el valor que era string en una fracción que indica la frecuencia de la categoría en la columna. Hay otros encoding techniques como Binary Encoding, Hash Encoding, Mean/Target Encoding,

Las herramientas usuales incluyen Pandas, Scikit-learn

One-Hot Encoding: Para categorías sin orden. Ejemplo colores:

```
ANTES:          DESPUÉS (One-Hot):
Color           Rojo  Verde  Azul
Rojo     →      1     0      0
Verde           0     1      0
Azul            0     0      1
```

Label Encoding: Para categorías ordinales. Ejemplo tallas:

```
ANTES:          DESPUÉS:
Talla           Talla_Encoded
S        →      0
M               1
L               2
```

Discretizar

Transformación de variables continuas en intervalos discretos.
Esto puede ser relevante al trabajar con tiempo, temperatura, mediciones, etc. Algunos de los métodos comunes son:

- Equal Width Binning: divide el rango en n intervalos de igual tamaño.
- Equal Frequency Binning: divide los datos de forma que cada intervalo tenga la misma cantidad de data points.
- Clusters: Dividir en clusters que implican similitud con algoritmos como K-Means. Todos los datos de un cluster se tratan como una única categoría.
- etc

Ejemplo: Agrupar edades para segmentación demográfica:

```
ANTES (Edad):       DESPUÉS (Bins):
[15,                ["0-18",
 25,           →     "19-35",
 70,                 "60+",
 32]                 "19-35"]
```

Herramientas: Pandas (cut/qcut), KNIME, RapidMiner.

Generación de nuevos atributos

Creación de características derivadas mediante operaciones matemáticas o lógicas.
Ejemplo: Calcular el Índice de Masa Corporal (IMC) a partir de peso y altura. Derivar la edad de la fecha de nacimiento.

Imputación

Técnicas para manejar valores faltantes mediante estimación o sustitución. Eso incluye técnicas estadísticas como remplazar por la media, mediana o moda; algoritmos de ML de regresión, KNN, u otros más específicos; utilizar distribuciones probabilísticas; entre muchos otros que dependen del tipo de datos y los casos de uso finales.

Ejemplo: Rellenar valores nulos en ingresos con la mediana del grupo demográfico.

Herramientas: Scikit-learn (SimpleImputer), DataWrangler, IBM SPSS.

Procesamiento de texto

Transformación de texto no estructurado a formato analizable. Puede incluir procesos como Tokenización, lematización y vectorización ya sea con TF-IDF o con embeddings.

Ejemplo: Tokenización, lematización y vectorización (TF-IDF) de reseñas de clientes.

Herramientas: NLTK, SpaCy, Transformers, TensorFlow Text.

4.1.2 ETL y ELT en el Procesamiento de Datos

ETL (Extract-Transform-Load): Nace en los 70s con sistemas mainframe y consolida su modelo en los 90s con data warehouses. Diseñado para hardware limitado: transformación *antes* de cargar para reducir la carga en los sistemas destino.
ELT (Extract-Load-Transform): Emerge en 2010s con Big Data y tecnologías cloud (Snowflake, BigQuery, Redshift). Habilitado por escalamiento elástico y procesamiento distribuido (Spark, columnar storage). Respuesta a necesidades de data lakes y análisis exploratorio con datos crudos.

ETL: Transformación en Tránsito

Transformación en **staging area** externa (servidores ETL). Caso de uso: Datos estructurados con reglas de negocio complejas (ej. consolidación financiera)

Usar ETL cuando: Requieres validación estricta pre-carga (ej. datos regulados: HIPAA, PCI-DSS). Los sistemas destino tienen capacidad computacional limitada. Existen transformaciones complejas independientes del almacenamiento.

ELT: Transformación en Destino

Transformación dentro del **sistema destino** (data warehouse/lake). Caso de uso: Análisis ad-hoc sobre datos semi-estructurados (ej. logs de apps móviles).

Usar ELT cuando: Necesitas retener datos crudos (ej. auditorías forenses). Trabajas con datos no estructurados/semi-estructurados. Requieres escalamiento horizontal (Big Data ¿ 1TB).

4.1.3 Views

Las vistas son representaciones virtuales de datos que se definen mediante consultas SQL almacenadas. No almacenan datos físicamente. Proporcionan una capa de abstracción sobre tablas base. Y no necesitan actualizarse ante nuevos registros.

Usar vistas para: Capas de presentación (BI tools), Transformaciones ligeras (filtros, joins simples), Control de acceso granular.

This stored query can target both tables and other views (you can create a view that queries other views). This stored query (view definition) represents part of the database, but it doesn't store any physical data! This is the first important difference to a "regular" table – views don't store the data, which means that every time you need the data from the view, the underlying stored query will be executed against the database. Since views are being run each time you "call" them, they will always pick the relevant data from the underlying tables. That means you don't need to worry if something changed in the underlying table (deleted/updated rows), as you will always get the actual data from the tables.

4.1.4 dbt (data build tool)

Es un framework open-source que permite ejecutar transformaciones de datos **dentro del data warehouse** mediante SQL, gestionar pipelines como código versionado (Git), automatizar testing, documentación y despliegues.
Promueve el principio de "Analytics engineering as software development".
dbt es la T en ELT. No extrae o carga datos, sino que transforma los datos en el Data Warehouse para ser consumidos por herramientas de BI.
dbt transforma mediante views/materialized views

Tiene integraciones nativas con:

- Warehouses: Snowflake, BigQuery, Redshift, Databricks
- Ingestion: Airbyte, Fivetran, Meltano
- BI: Metabase, Mode, Preset

The utility of dbt that makes it superior to other approaches originates from the combination of Jinja templates with SQL, and re-usable components or models.

Se puede utilizar dbt combinado con spark

4.1.5 Spark(https://spark.apache.org)

Se ha convertido en uno de los frameworks de procesamiento de grandes volúmenes de datos más destacados de la industria. Spark está diseñado para trabajar con una amplia gama de fuentes de datos, como HDFS, Apache Cassandra, Apache HBase y Amazon S3.

Unifica el procesamiento de tus datos en lotes y en tiempo real (streaming), utilizando tu lenguaje preferido: Python, SQL, Scala, Java o R.

Realiza Análisis Exploratorio de Datos (EDA) en datos a escala de petabytes sin tener que recurrir al submuestreo(downsampling) gracias a su escalamiento horizontal. Se ejecuta más rápido que la mayoría de los almacenes de datos. Tiene MLlib, biblioteca integrada de machine learning para tareas como clasificación, regresión y clustering.

- Pyspark

4.2 Serving

Fase final del pipeline donde los datos transformados se ponen a disposición de los usuarios finales, ya sea BI, dashboards, APIs, microservicios, ML o sistemas externos. Idealmente el servicio es con

acceso seguro, baja latencia y formatos consumibles.

4.2.1 Desplegar un API

Esto permite entregar datos a clientes en varias modalidades: web, móvil, IoT, otro servicio de analítica. Permite consultar datos en formato streaming en tiempo real. Además de permitir un control preciso y granular sobre qué datos exponer, a cada tipo de usuario.

Retos comunes:
Si el volumen de datos(+100) es grande es necesario implementar paginación, straming, etc.
Establecer protocolos de seguridad con JWT, OAuth2, Rate limiting.

Frameworks Principales

- **FastAPI**: Async, auto-documentación OpenAPI Ejemplo endpoint:

```
@app.get("/sales/{region}")
async def get_sales(region: str,
                    start: date,
                    end: date):
    return db.query(Sales)
        .filter(Sales.region == region)
        .filter(Sales.date.between(start, end))
```

- **Flask-RESTful**: Ligero, ideal para prototipos, Extensible con Flask-SQLAlchemy
- **Django REST Framework**: ORM integrado, autenticación robusta, Ideal para CRUD complejos

Librerías Complementarias

- **Pydantic**: Validación y serialización de modelos.
- **Celery**: Procesamiento asíncrono de requests pesados.
- **Authlib**: Implementación OAuth2/JWT.
- **Redis**: Caché de respuestas frecuentes.

Patrones de Diseño

- **Modelo de Paginación**:

```
{
  "data": [...],
  "pagination": {
    "total": 1000,
    "page": 2,
    "per_page": 50,
    "next": "/api/v1/sales?page=3"
  }
}
```

- **Versionado Semántico**:

  - `/api/v1/sales`
  - `/api/v2/sales`

- **Circuit Breaker**: Usando Tenacity

```
@retry(stop=stop_after_attempt(3),
      wait=wait_exponential(multiplier=1))
def fetch_external_data():
    ...
```

Buenas prácticas: Documentación Automática, Monitoreo, Pruebas automáticas, Logging.

4.2.2 Herramientas de Análisis y Visualización

Tableau

Plataforma líder de visualización empresarial con capacidades drag-and-drop y soporte para big data. Tableau es una herramienta de Inteligencia de Negocios (BI) para analizar datos visualmente. Los usuarios pueden crear y distribuir dashboards interactivos y compartibles que representan tendencias, variaciones y densidad en los datos en forma de gráficos y diagramas. Tableau puede conectarse a archivos, fuentes relacionales y de Big Data para adquirir y procesar datos.

Soporta, con Tableau Embedding API, añadir (embed) visualizaciones a una página web.

Power BI

(free tier on desktop) Microsoft Power BI es una plataforma de visualización de datos e informes. Un dashboard de Power BI permite reportar y visualizar datos en una amplia gama de estilos, incluyendo gráficos, mapas, diagramas, diagramas de dispersión y más. Allows to also create highly customizable visualizations with R and Python. Gran soporte para mapas.

Power BI en sí está compuesto por varias aplicaciones interrelacionadas: Power BI Desktop, Power BI Pro, Power BI Premium, Power BI Mobile,
Power BI Embedded: Power BI White-label para proporcionar paneles y analíticas orientadas al cliente en tus propias aplicaciones.
Power BI Report Server: Solución local incluida en Premium; puede ser migrada a la nube.

Looker

Looker es una plataforma de computación en la nube que proporciona inteligencia empresarial (BI) y análisis de datos. Integración de datos: Looker tiene la capacidad de establecer conexiones con varias fuentes de datos basadas en la nube, como los data warehouse Snowflake, Amazon Redshift y Google BigQuery. Debido a esto, los usuarios pueden examinar los datos almacenados en estas plataformas sin tener que copiarlos o reubicarlos. Visualización: Looker ofrece una variedad de opciones de visualización para transmitir eficazmente información valiosa de los datos. Para comunicar mediciones, tendencias y patrones, los usuarios pueden crear paneles, gráficos y diagramas.

Metabase

Metabase es una plataforma de inteligencia empresarial de código abierto. Puede usar Metabase para hacer preguntas sobre sus datos, o incrustar Metabase en su aplicación para permitir que sus clientes exploren sus propios datos.

5

Conferencia 5

Tecnologías de la Ingeniería de Datos: Tecnologías complementarias

5.1 Gestión de Datos

Las herramientas de data governance son soluciones de software diseñadas para ayudar a las empresas a gestionar sus datos de manera más efectiva. Estas herramientas respaldan procesos que aseguran que los datos sean precisos, seguros y estén en línea con los requisitos regulatorios. Pueden incluir capacidades tales como:

- **Catalogación de datos:** el proceso de crear un inventario exhaustivo de sus activos de datos. Una herramienta de gobierno de datos con sólidas capacidades de catalogación facilita la búsqueda, comprensión y uso de datos en toda su organización.
- **Seguimiento del linaje de datos:** comprender de dónde provienen sus datos, cómo se transforman y a dónde van es crucial. Las funciones de linaje de datos le permiten rastrear el flujo de datos a través de los sistemas, lo que garantiza la transparencia y la confianza.
- **Gestión del cumplimiento normativo:** muchas industrias están sujetas a regulaciones estrictas como GDPR, HIPAA o CCPA. Las herramientas de gobierno de datos que ofrecen gestión del cumplimiento normativo pueden ayudar a su organización a mantener y demostrar el cumplimiento de estas regulaciones.
- **Control de calidad de los datos:** las herramientas de gobierno de datos deben tener controles de calidad de datos integrados que garanticen la precisión, integridad y consistencia de sus datos. Estas herramientas deben proporcionar alertas automatizadas cuando surjan problemas de calidad de los datos.
- **Controles de acceso basados en roles:** la seguridad de los datos es un componente clave del gobierno. Asegúrese de que su herramienta admita controles de acceso basados en roles para que solo el personal autorizado tenga acceso a datos confidenciales.

5.2 Orquestación de Datos

En el ámbito de la ingeniería de datos, el proceso de transformar datos brutos en información valiosa implica la utilización de pipelines y flujos de trabajo de datos. Estos conceptos forman la columna vertebral de la gestión eficiente de datos y permiten a los ingenieros de datos manejar tareas complejas de procesamiento de datos sin problemas.

Los pipelines de datos sirven como el tejido conectivo dentro del ecosistema de la ingeniería de datos. Son responsables de orquestar el flujo de datos desde su origen hasta su destino, manejando varias etapas del procesamiento de datos a lo largo del camino. Los ingenieros de datos diseñan e implementan pipelines para asegurar el movimiento fluido de los datos, incluyendo tareas como la ingestión, integración, transformación y carga de datos.

Apache Airflow: Es una herramienta de código abierto ampliamente utilizada para la creación, programación y monitoreo de flujos de trabajo complejos. Airflow permite a los equipos de DataOps diseñar pipelines de datos de manera flexible y modular, integrando distintas fuentes y destinos de datos. Es ideal para la automatización de tareas y facilita la colaboración entre equipos.

Airflow proporciona un framework de Python flexible y escalable que permite a los ingenieros de datos crear, programar y monitorizar sus pipelines de datos.
Un DAG representa un conjunto de tareas y sus dependencias, donde la dirección de las flechas indica el flujo de datos o ejecución.
La parte "Dirigido" significa que cada tarea tiene una relación definida con una o más tareas, formando una cadena de dependencias. Esto asegura que las tareas se ejecuten en el orden correcto, basándose en sus dependencias.
La parte "Acíclico" significa que no hay bucles o ciclos en el grafo. En otras palabras, las tareas no pueden tener dependencias circulares que conducirían a bucles infinitos. Esto asegura que los flujos de trabajo puedan completarse exitosamente sin bucles infinitos o dependencias no resueltas.

Prefect: Similar a Airflow, permite la orquestación y automatización de flujos de trabajo de datos; así como el monitoreo de pipelines con una interfaz intuitiva para equipos de datos.

Prefect is an open-source orchestration tool for data engineering. It is a Python-based tool that allows you to define, schedule, and monitor your data pipelines. Prefect is a great tool for data engineers and data scientists who want to automate their data pipelines.

5.3 DataOps

La creciente complejidad de los datos en las organizaciones modernas ha dado lugar a la necesidad de prácticas más ágiles y eficientes. DataOps surge como una respuesta para mejorar la gestión de datos, optimizar flujos de trabajo y fomentar una mayor colaboración entre equipos de TI y analistas de datos. DataOps es una metodología que combina los principios de DevOps con la gestión de datos. Su objetivo principal es optimizar el ciclo de vida de los datos, desde la integración y almacenamiento hasta su análisis y entrega, asegurando la calidad y confiabilidad. La adopción de herramientas adecuadas es crucial para implementar DataOps con éxito y maximizar los beneficios de esta metodología.

Talend: Plataforma para integración, transformación y migración de datos en tiempo real, con enfoque en automatización y calidad de datos para mantener altos estándares y cumplimiento normativo.

5.3.1 Containerization

La contenerización es una práctica de desarrollo de software que empaqueta una aplicación y sus dependencias en un único contenedor. El contenedor está aislado del sistema host, permitiendo que la aplicación se ejecute en cualquier infraestructura que soporte contenedores, independientemente del sistema operativo subyacente.
Inspirados por la idea de las máquinas virtuales, los contenedores proporcionan entornos ligeros y aislados que empaquetan las aplicaciones con sus dependencias. Esta innovación asegura consistencia, portabilidad y eficiencia. Con el tiempo, herramientas como Docker y Kubernetes se han convertido en estándares de la industria para la gestión de aplicaciones contenerizadas, permitiendo una escalabilidad y orquestación sin problemas.

Los contenedores pueden moverse fácilmente de un host a otro, lo que facilita la migración de una solución de ingeniería de datos a una infraestructura diferente si es necesario.
Los contenedores proporcionan un entorno consistente para que las aplicaciones se ejecuten, asegurando que la solución de ingeniería de datos se comporte de la misma manera independientemente del entorno en el que se despliegue.

Docker: Plataforma de código abierto que permite la automatización del despliegue, escalado y gestión de aplicaciones en contenedores. Kubernetes facilita la creación de entornos escalables y resilientes para gestionar grandes volúmenes de datos y flujos de trabajo. Con Kubernetes, los equipos pueden orquestar los recursos necesarios de manera eficiente, garantizando que las aplicaciones de datos operen de manera fluida.

Kubernetes: Plataforma para automatizar despliegue, escalado y gestión de aplicaciones en contenedores, usada para crear entornos escalables y resilientes en proyectos de ingeniería de datos y DataOps

6

Conferencia 6

Integración de Sistemas en la Ingeniería de Datos

6.1 System Integration

Definición:
La Integración de Sistemas es el proceso de ingeniería que consiste en integrar hardware, software, redes y fuentes de datos dispares en un sistema unificado y cohesivo que opera como una sola entidad funcional. Esto implica establecer canales de comunicación y protocolos de intercambio de datos entre sistemas previamente aislados. El objetivo es lograr que estos sistemas trabajen juntos sin problemas para que puedan compartir información y procesos de manera más eficiente. La integración de sistemas se centra típicamente en mejorar la funcionalidad, el flujo de datos y la eficiencia operativa.

Con los años, la integración de sistemas ha evolucionado. Si bien antes se trataba principalmente de conectar servidores on-premise (locales), la integración actual a menudo se realiza en la nube. Ya sea a través de APIs o buses de servicio empresarial, la integración moderna ayuda a que diferentes sistemas y herramientas trabajen juntos de manera eficiente en un entorno basado en la nube.

6.1.1 Tipos de Integración de Sistemas y Ejemplos

Integración de Datos (Data Integration)

La integración de datos es un tipo de integración de sistemas enfocada en consolidar información de diversas fuentes, como plataformas, servicios y bases de datos, en una visión unificada.

Ejemplo: Starbucks utiliza la integración de datos para combinar información de su aplicación móvil, sistemas de punto de venta (POS) y comentarios de clientes. Esta integración permite promociones personalizadas, seguimiento de inventario en tiempo real y una mejor experiencia del cliente mediante análisis integrales.

Integración de Aplicaciones (Application Integration)

La integración de aplicaciones se refiere al proceso de conectar diferentes aplicaciones empresariales para que funcionen de manera eficiente. Esto es crucial para optimizar flujos de trabajo y procesos de negocio, permitiendo que los datos se muevan entre aplicaciones automáticamente sin intervención manual.

Ejemplo: Integrar herramientas como Slack y Jira para que las actualizaciones de proyectos se sincronicen en tiempo real entre plataformas de comunicación y gestión de proyectos.

Integración de Sistemas Legacy (Legacy System Integration)

Este tipo de integración conecta sistemas antiguos (legacy) con tecnologías modernas, permitiendo seguir utilizando datos existentes junto con herramientas nuevas.

Ejemplo: Un hospital que conecta un antiguo sistema de gestión de pacientes con un nuevo sistema de Registro Electrónico de Salud (EHR, por sus siglas en inglés). Esto permite a los médicos acceder a toda la información del paciente desde ambos sistemas sin cambiar su rutina.

Integración de Aplicaciones Empresariales (Enterprise Application Integration, EAI)

La EAI unifica diferentes aplicaciones de software dentro de una empresa para que funcionen de manera cohesionada. Conecta diversas aplicaciones, permitiéndoles compartir e intercambiar datos en tiempo real. Resuelve el problema de aplicaciones empresariales dispares que almacenan datos por separado, integrando todo en un sistema unificado.

Ejemplo: Una empresa utiliza sistemas separados para ventas, servicio al cliente e inventario. La EAI enlaza estos sistemas, permitiéndoles compartir información y comunicarse en tiempo real. Esta integración hace que las operaciones sean más fluidas y eficientes al eliminar la transferencia manual de datos.

Integración Empresa a Empresa (Business-to-Business, B2B Integration)

La integración B2B conecta los sistemas de diferentes empresas para agilizar sus procesos colaborativos. Automatiza el intercambio de transacciones y documentos, como órdenes y facturas, entre negocios.

Este flujo eficiente de información conduce a una cooperación más eficaz, transacciones más rápidas y menos errores, simplificando las interacciones con proveedores, clientes y socios.

Intercambio de Datos Electrónicos (Electronic Data Interchange, EDI)

La integración de documentos electrónicos (EDI) es un tipo específico de integración centrado en el intercambio de documentos comerciales entre sistemas, frecuentemente entre empresas (B2B). El EDI automatiza y protege la transferencia de documentos, reduciendo errores manuales y mejorando la eficiencia. En industrias como la salud y las finanzas, donde la confidencialidad de los datos es crítica, el EDI garantiza un manejo seguro de documentos.

Ejemplo: Imagina una empresa que utiliza sistemas separados para ventas, servicio al cliente e inventario. Cada sistema tiene sus propios datos y procesos, lo que dificulta tener una visión completa de lo que ocurre en el negocio. La EAI integra estos sistemas, permitiéndoles compartir información y comunicarse en tiempo real.

6.1.2 Integración de Sistemas. Conectando sistemas

APIs (Interfaces de Programación de Aplicaciones)

Las APIs son un mecanismo fundamental para permitir la interoperabilidad entre sistemas de software. Una API define un conjunto de rutinas, protocolos y herramientas para construir aplicaciones. Actúa esencialmente como intermediario, proporcionando una interfaz estandarizada mediante la cual las aplicaciones pueden solicitar servicios e intercambiar datos. Esta capa de abstracción protege a los desarrolladores de la complejidad subyacente de los sistemas que interactúan.

Las APIs especifican formatos de datos (ej. JSON, XML) y protocolos de comunicación (ej. REST, SOAP, gRPC) para la interacción. Las APIs RESTful, en particular, utilizan métodos HTTP (GET, POST, PUT, DELETE) para realizar operaciones sobre recursos identificados por URLs. La seguridad es una consideración crítica, abordada frecuentemente mediante mecanismos de autenticación (ej. claves API, OAuth) y autorización.

Las APIs soportan un enfoque de diseño modular, permitiendo que los sistemas evolucionen independientemente mientras mantienen la interoperabilidad.

Middleware

El middleware abarca una amplia categoría de software que facilita la comunicación y gestión de datos entre aplicaciones distribuidas. Proporciona una capa de abstracción que oculta la heterogeneidad de los sistemas subyacentes, simplificando los esfuerzos de integración.

Las soluciones de middleware ofrecen diversos servicios, incluyendo colas de mensajes (ej. RabbitMQ, Apache Kafka), gestión de transacciones, transformación de datos y seguridad. Las colas de mensajes permiten comunicación asíncrona, desacoplando aplicaciones y mejorando la escalabilidad. Los motores de transformación de datos convierten información entre diferentes formatos y esquemas, asegurando compatibilidad entre sistemas.

El middleware desacopla aplicaciones, permitiéndoles operar independientemente y mejorando la resiliencia del sistema. Las colas de mensajes y otros servicios de middleware mejoran la escalabilidad del sistema distribuyendo cargas de trabajo y gestionando comunicación asíncrona.

Webhooks

Los webhooks son un mecanismo para permitir comunicación en tiempo real entre aplicaciones. En lugar de consultar constantemente (polling) un servidor por actualizaciones, una aplicación puede registrar un webhook para recibir notificaciones cuando ocurren eventos específicos en otro sistema.

Un webhook es básicamente un callback HTTP definido por el usuario. Cuando ocurre un evento predefinido en un sistema fuente, este envía una petición HTTP POST a una URL especificada (el endpoint del webhook) conteniendo datos relacionados con el evento. La aplicación receptora puede entonces procesar los datos y tomar las acciones correspondientes.

Los webhooks proporcionan actualizaciones en tiempo real, permitiendo respuestas inmediatas a eventos. Eliminan la necesidad de polling constante, reduciendo latencia y mejorando la capacidad de respuesta del sistema. Son más eficientes en recursos que el polling, ya que solo transmiten datos cuando ocurren eventos.

EDI (Intercambio Electrónico de Datos)

El EDI es un método estandarizado para intercambiar documentos comerciales electrónicamente entre organizaciones. Reemplaza procesos tradicionales basados en papel con transmisión automatizada de datos, optimizando operaciones comerciales.

Los estándares EDI (ej. ANSI X12, UN/EDIFACT) definen el formato y estructura de documentos comerciales como órdenes de compra, facturas y avisos de envío. Las transacciones EDI típicamente se transmiten sobre redes seguras usando protocolos como AS2 (Applicability Statement 2). Se utiliza software de traducción para convertir datos entre formatos EDI y formatos internos de aplicaciones.

El EDI elimina la entrada manual de datos, reduciendo errores y mejorando la precisión de la información.

Snaps de SnapLogic

Los Snaps de SnapLogic son conectores preconstruidos diseñados para simplificar la integración con aplicaciones comúnmente utilizadas. Estos conectores ofrecen una interfaz amigable, permitiendo a los usuarios establecer conexiones sin necesidad de programación personalizada extensa.

Los Snaps abstraen las complejidades de las APIs subyacentes, proporcionando un enfoque visual y basado en configuración para la integración. Típicamente soportan operaciones comunes para interactuar con aplicaciones objetivo, como recuperación de datos, creación de datos y modificación de datos.

6.1.3 Enfoques de Integración de Sistemas

Las metodologías de integración de sistemas definen cómo los datos y procesos se interconectan entre subsistemas, donde cada enfoque ofrece ventajas y limitaciones distintas según escalabilidad, comple-

jidad y requisitos funcionales. A continuación analizamos los modelos de integración predominantes:

Modelo de Integración Vertical (Basado en Silos)

Arquitectura jerárquica donde los subsistemas se apilan en capas funcionales, con componentes básicos en la base y procesos de alto nivel en la cima. Esto crea silos aislados optimizados para flujos de trabajo específicos.

Ventajas:

- *Simplicidad:* Configuración sencilla y directa
- *Aislamiento Funcional:* Ideal para procesos específicos y aislados
- La integración vertical es más fácil de gestionar y solucionar dentro del silo

Desventajas:

- Difícil escalabilidad debido a su estructura rígida
- Cada nueva función requiere su propio silo, generando posibles ineficiencias
- Flexibilidad limitada para compartir datos entre funciones

Ejemplo: Procesamiento de pedidos en e-commerce - los subsistemas de ventas, producción y logística se encadenan linealmente sin intercambio lateral de datos.

Modelo de Integración Horizontal (Hub-and-Spoke)

Arquitectura centralizada que emplea una capa intermediaria (ej. Enterprise Service Bus, ESB) para enrutar datos entre subsistemas, desacoplando emisores y receptores.

Ventajas:

- *Modularidad:* Alta flexibilidad para reemplazar o actualizar sistemas individuales
- *Gestión Unificada:* Administración centralizada que reduce la complejidad de comunicación entre sistemas
- Soporta amplia gama de funciones y aplicaciones

Desventajas:

- Configuración y mantenimiento más complejos
- Puede ser costoso por requerir una capa adicional de integración
- *Punto Único de Falla:* Riesgo de un único punto de falla si la capa central tiene problemas

Ejemplo: Ecosistemas centrados en CRM (ej. Salesforce) como hubs para subsistemas de marketing, inventario y logística.

Modelo de Integración Punto a Punto

Enfoque más simple de conectividad donde se vinculan dos sistemas directamente entre sí. Conexiones directas por pares entre subsistemas, típicamente para transferencia unidireccional de datos o activación de eventos.

Ventajas:

- Implementación simple y rápida
- Permite comunicación directa entre sistemas con mínima sobrecarga

- Bajo costo y esfuerzo inicial

Desventajas:

- Escalabilidad limitada, ya que cada nueva conexión añade complejidad
- Se vuelve difícil de gestionar al crecer el número de conexiones
- No es adecuado para integraciones a gran escala con muchos sistemas

Ejemplo: Formularios de contacto en sitios web que envían datos directamente a un CRM como Salesforce.

Modelo de Integración en Estrella

La integración en estrella conecta cada sistema directamente con los demás, creando una red de conexiones punto a punto. Modelo híbrido que combina conexiones punto a punto selectivas entre subsistemas críticos, formando una malla parcial.

Ventajas:

- *Eficiencia Dirigida:* Permite rutas de datos prioritarias sin interconectividad completa
- *Flexibilidad:* Permite personalizar conexiones según necesidades específicas
- Máxima flexibilidad para compartir datos entre sistemas
- Mejora funcionalidad al permitir múltiples rutas para flujo de datos

Desventajas:

- Creciente complejidad al añadir más sistemas
- Requiere alto esfuerzo de mantenimiento para gestionar todas las conexiones
- Difícil escalabilidad, ya que cada nuevo sistema incrementa exponencialmente el número de conexiones necesarias

Ejemplo: Un cliente realiza un pedido en un sitio web. El sitio envía la orden al sistema de ventas, que a su vez envía la información al departamento de envíos y logística. Simultáneamente, el sitio registra al cliente en el sistema CRM de la empresa, que envía una notificación a un representante de ventas en Slack. Múltiples aplicaciones están conectadas y se comunican entre sí, pero no todas están interconectadas entre ellas.

7

Conferencia 7

Integración de Datos en la Ingeniería de Datos

Es parte de Data Mangement (o Gestión de Datos), una de las corrientes subyacentes del ciclo de vida de la ingeniería de datos.

7.1 Integración de Datos

La integración de datos es un proceso sistemático que implica la combinación de datos provenientes de fuentes heterogéneas en un formato unificado, coherente y estandarizado. Este proceso es crítico en entornos computacionales y empresariales modernos, donde los datos se generan y almacenan en diversas plataformas, incluyendo bases de datos relacionales, aplicaciones, hojas de cálculo, servicios en la nube y interfaces de programación de aplicaciones (APIs). El objetivo principal de la integración de datos es facilitar el procesamiento analítico eficiente, los flujos de trabajo operacionales y la toma de decisiones basada en datos.

Este proceso implica extraer, transformar y cargar datos en un repositorio central o base de datos para proporcionar una visión integral de los datos y permitir una toma de decisiones informada.

La integración de datos trasciende la mera agregación; requiere la reconciliación de diferencias estructurales, semánticas y sintácticas entre los conjuntos de datos fuente. A menudo se emplean soluciones middleware avanzadas o capas de datos virtuales para orquestar este proceso, garantizando una interoperabilidad sin problemas. El ecosistema unificado de datos resultante permite análisis de alta precisión, inteligencia operativa en tiempo real y conocimientos estratégicos para el negocio.

7.1.1 Proceso de integración de datos

1. Inicio con Descubrimiento y Perfilado de Datos

Esta fase implica un análisis profundo del panorama de datos, identificando las múltiples fuentes que alimentan el ecosistema informativo de la organización. Aquí, el enfoque está en comprender los matices de cada fuente de datos—su estructura, calidad y peculiaridades—sentando las bases para una integración sin fisuras.

2. Extracción de Datos con Precisión

Este paso crítico requiere extraer datos de sus repositorios nativos, lo que demanda precisión y sensibilidad para evitar interrumpir la funcionalidad de los sistemas fuente. El desafío radica en reunir eficientemente los datos de sus silos, asegurando que estén preparados para la fase de transformación subsiguiente.

3. Transformación de Datos para Uniformidad

El corazón del proceso de integración de datos reside en la transformación de datos. En esta etapa, los diversos elementos de datos son estandarizados, depurados y armonizados, asegurando que "hablen" un lenguaje común. Esta transformación es pivotal, convirtiendo una cacofonía de datos dispares en un conjunto de datos armonioso listo para la integración. Un ejemplo ilustrativo podría ser un gigante del comercio electrónico integrando datos de interacción con clientes en múltiples canales. Aquí, datos de interacciones en redes sociales, visitas al sitio web e historiales de compras se someten a normalización y deduplicación, volviéndose analizables como un todo coherente.

4. Carga de Datos en un Repositorio Unificado

Esta fase implica transferir los datos refinados a un repositorio central, como un data warehouse, donde se almacenan, acceden y analizan. La carga debe ejecutarse manteniendo la integridad de los datos y optimizando para accesos futuros.

5. Garantía de Integridad mediante Validación

6. Mantenimiento de Relevancia con Sincronización de Datos

Validación y aseguramiento de calidad de datos. Este paso confirma que los datos integrados son precisos, consistentes y de alta calidad. Se aplican controles de validación rigurosos para detectar y rectificar cualquier anomalía, asegurando la fiabilidad de los datos para la toma de decisiones.

7. Facilitación de Acceso y Consumo de Datos

La culminación del proceso de integración de datos asegura que los datos integrados sean fácilmente accesibles para análisis e inteligencia de negocio. Esto implica disponibilizar los datos en formatos y a través de canales que soporten un consumo, análisis y reporte eficientes.

7.1.2 Tipos de Integración de Datos

Existen varios métodos de integración de datos, cada uno adecuado para diferentes casos de uso. Los principales tipos son los siguientes:

Integración Manual de Datos

La integración manual de datos implica la extracción, transformación y carga (ETL) de datos de fuentes dispares mediante intervención humana. Este enfoque, aunque conceptualmente simple, requiere una inversión significativa de recursos humanos. Si bien es intensivo en recursos, puede ser una solución efectiva para proyectos de integración a pequeña escala o especializados donde las herramientas automatizadas no son factibles o rentables.

El proceso central implica recolectar manualmente datos de diversas fuentes (por ejemplo, formularios en línea, encuestas a clientes, feeds de redes sociales). Típicamente, esto incluye técnicas manuales como copiar y pegar datos en software de hojas de cálculo. Los datos recolectados luego se transforman y limpian, usualmente utilizando software de hojas de cálculo o herramientas similares, para garantizar consistencia y precisión. Finalmente, los datos transformados se cargan en un repositorio central o base de datos, potencialmente usando scripts ETL simples o soluciones personalizadas.

Las organizaciones pueden emplear integración manual cuando la complejidad o sensibilidad de los datos requiere supervisión humana. Aplicaciones comunes incluyen segmentación de clientes, detección de fraudes y cumplimiento normativo. Por ejemplo, combinar manualmente datos exportados de diferentes bases de datos dentro de un entorno de hojas de cálculo representa un ejemplo típico, aunque potencialmente propenso a errores.

La definición fundamental de integración manual de datos es el proceso de combinar datos de múltiples fuentes sin el uso de herramientas o software automatizado. Esta dependencia de intervención manual en cada paso hace que el proceso sea lento y susceptible a errores.

Ventajas:

- **Simplicidad y Accesibilidad:** No se requieren herramientas o software especializados. El software de hojas de cálculo, ampliamente disponible, provee una plataforma accesible para la integración manual de datos.
- **Flexibilidad y Personalización:** Permite un alto grado de personalización y flexibilidad, sin las limitaciones de herramientas automatizadas, permitiendo transformaciones a medida y manejo de matices únicos en los datos.
- **Adecuación para Proyectos Pequeños:** Efectivo para proyectos a pequeña escala con fuentes de datos limitadas y volúmenes bajos, donde la infraestructura de automatización puede no justificarse.

Desventajas:

- **Consumo de Tiempo:** Proceso intensivo en mano de obra que requiere esfuerzo manual significativo en cada etapa del proceso ETL.
- **Propensión a Errores:** La intervención humana introduce riesgos de errores durante la extracción, transformación y carga de datos.
- **Ineficiencia a Escala:** Se vuelve cada vez más impráctico e ineficiente a medida que aumentan el volumen de datos, la complejidad de las fuentes y las transformaciones requeridas. La escalabilidad es severamente limitada.
- **Escalabilidad Limitada:** No es adecuado para proyectos de integración de datos a gran escala debido a limitaciones inherentes para manejar grandes conjuntos de datos y transformaciones complejas.

En resumen, la integración manual de datos presenta una opción viable para proyectos pequeños donde no se necesitan herramientas especializadas y los volúmenes de datos son manejables. Sin embargo, las limitaciones inherentes del proceso en cuanto a escalabilidad, eficiencia y propensión a errores requieren una consideración cuidadosa antes de emplearlo para esfuerzos de integración de datos complejos o a gran escala. Decidir entre automatizar la integración de datos o continuar con métodos manuales requiere sopesar cuidadosamente estas compensaciones.

Integración de Datos mediante Middleware

Las soluciones de middleware son intermediarios esenciales en escenarios de integración complejos, facilitando la comunicación y transformación de datos entre sistemas dispares. Actúan como una capa de software que puentea la brecha entre diversas aplicaciones, permitiendo el intercambio y traducción de datos sin intervención manual. Estas soluciones comúnmente proveen sincronización en tiempo real, procesamiento automatizado y entrega garantizada de datos. Las organizaciones las utilizan para impulsar la Integración de Aplicaciones Empresariales (EAI) e interacciones sistema-a-sistema.

Considere un escenario donde un sistema almacena datos de clientes en formato XML, mientras otro espera JSON. El middleware permite la traducción entre estos formatos, automatizando el proceso de transformación en lugar de requerir codificación manual. Herramientas como ETL (Extraer, Transformar, Cargar) simplifican aún más la integración al automatizar estos procesos.

La integración mediante middleware utiliza software especializado para integrar datos de fuentes diversas. El middleware extrae datos, los transforma a un formato común y los carga en un repositorio central o base de datos. Gestiona el flujo de datos entre sistemas, asegurando procesamiento y almacenamiento eficientes.

Diferentes tipos de middleware atienden necesidades específicas:

- **Middleware orientado a mensajes:** Enfocado en comunicación asíncrona usando colas de mensajes.
- **Middleware para bases de datos:** Provee una interfaz unificada para acceder a diferentes bases.
- **Middleware de integración de aplicaciones:** Conecta y orquesta aplicaciones mediante APIs y servicios.

Cada tipo ofrece capacidades únicas para distintos proyectos.
Ventajas:

- **Eficiencia mejorada:** Centraliza la gestión de flujos, optimizando el proceso.
- **Escalabilidad:** Plataforma flexible para integrar numerosas fuentes.
- **Simplificación:** Interfaz común para acceso y manipulación de datos.
- **Calidad de datos:** Incluye manejo de errores y limpieza.

Desventajas:

- **Complejidad:** Requiere integración de software adicional.
- **Costo:** Necesita habilidades especializadas para implementación y mantenimiento.
- **Dependencia:** Crea dependencia de la plataforma.
- **Retos de integración:** Requiere coordinación cuidadosa con infraestructuras existentes.

En conclusión, el middleware puede mejorar significativamente la eficiencia y escalabilidad de la integración, aunque su adopción debe evaluarse cuidadosamente contra sus potenciales desventajas.

Almacenamiento de Datos (Data Warehousing)

El data warehousing implica la extracción, transformación y carga (ETL) de datos desde fuentes heterogéneas hacia un repositorio centralizado (data warehouse). Este entorno estructurado está optimizado para consultas sistemáticas, reportes y análisis avanzado.

Un data warehouse provee una visión histórica e integrada de los datos organizacionales, facilitando análisis de tendencias y toma de decisiones informada. A diferencia de bases transaccionales, están diseñados para rendimiento analítico mediante esquemas desnormalizados, índices optimizados y almacenamiento columnar que acelera consultas sobre grandes volúmenes.

Son cruciales para organizaciones que necesitan:

- **Análisis de clientes:** Comportamiento, segmentación y campañas personalizadas.
- **Reportes financieros:** Estados consolidados, KPIs y cumplimiento regulatorio.
- **Modelado predictivo:** Pronósticos, evaluación de riesgos y optimización.

Por ejemplo, con datos de ventas fragmentados en múltiples bases operacionales, un data warehouse los consolida permitiendo consultas históricas complejas. Esta agregación habilita análisis que serían ineficientes en bases tradicionales. Además, integrar datos de ventas, marketing y CRM permite reportes unificados con visión holística del negocio.

Integración Basada en Aplicaciones

La integración de datos basada en aplicaciones es un paradigma de diseño donde las funcionalidades de integración se incorporan directamente dentro de una aplicación, permitiendo conectividad y sincronización de datos entre diferentes sistemas en tiempo real. Este enfoque provee capacidades intrínsecas de integración, eliminando la necesidad de herramientas o capas separadas.

A diferencia de métodos tradicionales que dependen de mecanismos externos, esta integración otorga a las aplicaciones control directo sobre la conectividad y armonización de datos. Al incorporar la lógica de integración directamente en la aplicación, las organizaciones logran mayor agilidad, eficiencia e interoperabilidad. Esta integración más estrecha permite flujos de trabajo personalizados y control detallado sobre las transformaciones de datos.

Por ejemplo, un sistema CRM que se integra directamente con herramientas de email marketing ilustra este enfoque. La sincronización en tiempo real mejora la precisión de los datos y optimiza los esfuerzos de marketing.

Además, este método facilita la incorporación de diversos patrones de integración (procesamiento por lotes o streaming en tiempo real) dentro de la arquitectura de la aplicación. Esto permite obtener insights más profundos, fomentando la innovación y maximizando el valor de los sistemas organizacionales. La integración en la nube es un ejemplo claro, unificando datos de múltiples fuentes cloud para mejorar la toma de decisiones.

Ventajas:

- **Simplicidad:** Los usuarios interactúan directamente con la aplicación, sin necesidad de entender herramientas complejas de integración.
- **Rendimiento:** La integración embebida puede ser más eficiente que capas separadas, especialmente con grandes volúmenes de datos.
- **Personalización:** Permite adaptar la lógica de integración a necesidades específicas de la aplicación.

Desventajas:

- **Complejidad:** Implementar integración dentro de aplicaciones puede ser complejo, especialmente con fuentes diversas.
- **Interoperabilidad:** Puede limitar el intercambio de datos entre sistemas diversos comparado con capas de integración independientes.
- **Mantenimiento:** Actualizar la lógica de integración embebida puede ser costoso conforme evoluciona la aplicación.

En resumen, este enfoque ofrece beneficios de simplicidad y rendimiento, pero presenta desafíos en complejidad y mantenimiento.

Federación de Datos (Virtualización)

La federación de datos, o integración virtual, permite acceder y consultar datos en tiempo real a través de fuentes heterogéneas sin migración física. En lugar de consolidar datos, crea una capa virtual que presenta una vista lógica unificada de datos distribuidos en diferentes sistemas.

Esta capa virtual abstrae las fuentes subyacentes. Por ejemplo, datos de clientes en un CRM, ventas en un ERP e inventario en otra base pueden consultarse mediante una interfaz unificada. Aplicaciones típicas incluyen análisis avanzado de clientes, motores de recomendación y detección de fraudes. Como ilustración, permite consultar múltiples bases como si fueran una única base de datos unificada.

Características clave:

- Acceso en tiempo real sin replicación física

- Vista lógica unificada de múltiples fuentes
- Elimina necesidad de procesos ETL tradicionales
- Ideal para escenarios que requieren datos actualizados

Beneficios principales:

- Reduce la redundancia de datos
- Minimiza la latencia en el acceso
- Mantiene datos en sus sistemas fuente
- Proporciona una visión integral

7.1.3 Enfoques de Integración de Datos

Integración por Lotes (Batch Integration)

La integración por lotes (Batch Integration) es un elemento fundamental del proceso de integración de datos, diseñado para escenarios donde los datos en tiempo real no son primordiales. Este enfoque, integral al proceso de integración de datos, implica la agregación de datos en intervalos predeterminados, optimizando así la eficiencia y minimizando la carga en los sistemas de origen. A pesar de su simplicidad, el método de integración por lotes (Batch Integration) dentro del proceso de integración de datos puede introducir retrasos en la disponibilidad de los datos, lo que plantea desafíos para la toma de decisiones sensibles al tiempo.

Integración en Tiempo Real (Real-time Integration)

El proceso de integración de datos se ve significativamente mejorado por la integración en tiempo real (Real-time Integration), que garantiza la disponibilidad inmediata de los datos, un requisito crítico para las operaciones que exigen datos actualizados al minuto. Esta faceta del proceso de integración de datos es esencial para permitir la toma de decisiones dinámicas y la capacidad de respuesta operativa. Sin embargo, se debe considerar la complejidad y la intensidad de recursos de la integración en tiempo real (Real-time Integration) dentro del proceso de integración de datos.

Integración Híbrida (Hybrid Integration)

La integración híbrida (Hybrid Integration) representa una combinación estratégica dentro del proceso de integración de datos, combinando las fortalezas de los métodos por lotes (Batch) y en tiempo real (Real-time) para ofrecer una solución versátil adaptada a diversas necesidades comerciales. Este enfoque enriquece el proceso de integración de datos al proporcionar flexibilidad, permitiendo a las organizaciones equilibrar la inmediatez y la eficiencia en función de condiciones específicas de integración de datos.

Integración Basada en la Nube (Cloud-based Integration)

La computación en la nube ha introducido la integración basada en la nube (Cloud-based Integration) en el proceso de integración de datos, ofreciendo escalabilidad, rentabilidad y accesibilidad remota. Este enfoque se ha vuelto cada vez más relevante en la integración de datos, especialmente para las organizaciones con equipos geográficamente dispersos que requieren acceso a datos integrados. La seguridad de los datos y la dependencia de Internet son desafíos inherentes dentro de este enfoque de integración de datos.

7.1.4 Técnicas de Integración de Datos

La integración de datos eficaz es crucial para consolidar la información de fuentes dispares. Se encuentran disponibles varias técnicas, cada una con sus propias fortalezas y debilidades.

ETL (Extract, Transform, Load)

ETL (Extract, Transform, Load) es un proceso de integración de datos bien establecido que consta de tres fases principales:

- **Extract (Extracción):** Los datos se recuperan de varios sistemas de origen.
- **Transform (Transformación):** Los datos extraídos se someten a limpieza, formato y transformación para garantizar la coherencia y la idoneidad para el sistema de destino. Esto puede incluir la estandarización, la deduplicación y el enriquecimiento.
- **Load (Carga):** Los datos transformados se cargan en el sistema de destino designado, como un almacén de datos (data warehouse).

Las herramientas ETL (Extract, Transform, Load) facilitan el movimiento y la transformación de datos entre sistemas. Algunos ejemplos son Rivery Data Integration, Talend Data Integration, Informatica PowerCenter y Oracle Data Integrator. Un ejemplo práctico implica extraer información de clientes de una plataforma de comercio electrónico, estandarizar los formatos de las direcciones y cargarla en un almacén de datos (data warehouse) centralizado. Este proceso garantiza la precisión y la coherencia de los datos para la presentación de informes y el análisis.

ELT (Extract, Load, Transform)

ELT (Extract, Load, Transform) representa un enfoque más moderno de la integración de datos. A diferencia de ETL (Extract, Transform, Load), ELT invierte el orden de los pasos de transformación y carga:

- **Extract (Extracción):** Los datos se recuperan de varios sistemas de origen.
- **Load (Carga):** Los datos brutos se cargan directamente en el sistema de destino, a menudo un lago de datos (data lake) o un almacén de datos (data warehouse) que ofrece importantes recursos computacionales.
- **Transform (Transformación):** La transformación de datos se realiza dentro del sistema de destino utilizando sus capacidades de procesamiento. Esto a menudo aprovecha tecnologías como SQL u otros frameworks de procesamiento de datos.

Un ejemplo de ELT (Extract, Load, Transform) implica cargar datos de registro (log data) sin procesar en un lago de datos (data lake) y luego usar consultas SQL para transformar y analizar los datos. Este enfoque aprovecha la potencia de procesamiento del lago de datos (data lake) para una manipulación eficiente de los datos.

Comparación de Estrategias de Integración

La selección de una estrategia de integración adecuada depende de requisitos específicos, incluidos el volumen de datos, la tolerancia a la latencia y la complejidad de los datos. La Tabla 7.1 resume varias estrategias de integración de datos y sus características.
Batch ETL es adecuado para escenarios donde la latencia de los datos no es crítica. Procesa los datos en lotes, lo que lo hace rentable para grandes volúmenes.
Real-time ETL proporciona integración de datos casi en tiempo real, crucial para aplicaciones que requieren información actualizada al minuto. Sin embargo, es más complejo y potencialmente costoso de implementar.

Table 7.1: Comparación de Estrategias de Integración de Datos

Estrategia de Integración	**Adecuada para**	**Cómo Funciona**	**Ventajas**	**Desventajas**
Batch ETL	Integración no en tiempo real	Extrae, Transforma, Carga en lotes	Rentable, adecuado para grandes volúmenes de datos	No en tiempo real, latencia de datos potencial
Real-time ETL	Integración casi en tiempo real	Extracción, transformación y carga continuas	Datos actualizados al minuto, crítico para la toma de decisiones en tiempo real	Más complejo y potencialmente caro
Change Data Capture (CDC)	Sincronización de datos en tiempo real	Captura y replica los cambios del sistema de origen	Actualizaciones en tiempo real, minimiza la latencia de los datos	Configuración compleja, uso intensivo de recursos
Data Federation / Virtualization	Fuentes de datos heterogéneas	Proporciona una vista unificada sin integrar físicamente los datos	Minimiza la duplicación de datos, simplifica el acceso	Desafíos de rendimiento para consultas complejas, depende del rendimiento del sistema subyacente
Data Replication	Sincronización de datos distribuidos	Copia datos entre sistemas, mantiene la sincronización	Garantiza la coherencia de los datos en todas las ubicaciones	Uso intensivo de recursos, posibles conflictos de datos
Integración Basada en API (API-Based Integration)	Servicios de terceros / aplicaciones en la nube	Conecta sistemas a través de APIs para el intercambio de datos	Eficiente para servicios en la nube y socios externos	Control limitado sobre las APIs de terceros, puede ser necesario un desarrollo personalizado, depende de la disponibilidad y la estabilidad de la API

Change Data Capture (CDC) se centra en la sincronización de datos en tiempo real capturando y replicando los cambios en los sistemas de origen. Minimiza la latencia de los datos, pero requiere una configuración compleja y un uso intensivo de recursos.

Data Federation/Virtualization ofrece una vista unificada de los datos de fuentes heterogéneas sin integración física. Si bien minimiza la duplicación de datos y simplifica el acceso, el rendimiento puede ser un desafío para las consultas complejas. Su rendimiento depende en gran medida de las capacidades de los sistemas subyacentes.

Data Replication garantiza la coherencia de los datos en diferentes ubicaciones copiando datos entre sistemas. Si bien proporciona coherencia de datos, requiere muchos recursos y puede provocar conflictos de datos si no se gestiona con cuidado.

Integración Basada en API (API-Based Integration) utiliza APIs para el intercambio de datos, particularmente útil para la integración con servicios de terceros y aplicaciones en la nube. Si bien es eficiente para tales servicios, el control sobre las APIs de terceros es limitado y puede ser necesario un desarrollo personalizado. La dependencia de la disponibilidad y la estabilidad de la API introduce posibles vulnerabilidades.

7.1.5 Herramientas y Tecnologías para la Integración de Datos

Podemos simplificar el proceso de integración de datos utilizando diversas herramientas y tecnologías, las cuales son:

Herramientas ETL

Las herramientas ETL automatizan los procesos de extract, transform, and load (extraer, transformar y cargar), haciendo que la integración de datos sea más eficiente.

Por ejemplo, Talend, Apache NiFi e Informatica son herramientas ETL populares utilizadas para optimizar la integración de datos.

Soluciones de Data Warehousing

Las soluciones de Data Warehousing proporcionan un repositorio central para los datos integrados, lo que permite una consulta y análisis organizados.

Amazon Redshift, Google BigQuery y Snowflake son soluciones de Data Warehousing ampliamente utilizadas.

Herramientas de Virtualización de Datos

Las herramientas de virtualización crean una capa de datos virtual, permitiendo el acceso en tiempo real a los datos integrados.

Por ejemplo, Denodo, IBM Data Virtualization y Red Hat JBoss Data Virtualization son ejemplos de herramientas de virtualización de datos.

Enterprise Information Integration (EII)

Las herramientas EII proporcionan una vista unificada de los datos de fuentes dispares. Permiten a los usuarios acceder y consultar datos sin tener que mover o copiar los datos. Las soluciones EII populares incluyen WebSphere Information Integrator de IBM, TIBCO ActiveMatrix BusinessWorks y Microsoft SQL Server Integration Services.

7.1.6 Mejores Prácticas para la Integración de Datos

Para asegurar una integración de datos exitosa, siga estas mejores prácticas:

Definir Objetivos Claros

Defina claramente sus objetivos de integración de datos, tales como mejorar la precisión de los datos, mejorar la toma de decisiones u optimizar las operaciones.

Establecer el objetivo de integrar los datos de ventas y clientes para obtener mejores conocimientos del cliente es un ejemplo de un objetivo claro.

Elegir las Herramientas Adecuadas

Seleccione las herramientas que se ajusten a sus necesidades de integración, considerando factores como el volumen de datos, la complejidad y los requisitos en tiempo real.

Por ejemplo, usar una herramienta ETL para el procesamiento por lotes de grandes conjuntos de datos y una herramienta de virtualización de datos para el acceso a datos en tiempo real puede optimizar la integración de datos.

Asegurar la Calidad de los Datos

Implemente controles de calidad de los datos para garantizar la precisión y la consistencia de los datos integrados.

Por ejemplo, utilizar reglas de validación de datos para comprobar si hay duplicados y valores faltantes asegura una alta calidad de los datos.

Mantener la Seguridad de los Datos

Asegúrese de que los procesos de integración de datos actúen de acuerdo con las regulaciones de seguridad y privacidad de los datos, protegiendo la información confidencial.

Por ejemplo, cifrar los datos durante la transferencia y asegurar el cumplimiento con GDPR son críticos para mantener la seguridad de los datos.

Monitorear y Optimizar

Monitoree regularmente los procesos de integración de datos y optimícelos para obtener rendimiento y eficiencia.

Por ejemplo, utilizar herramientas de monitoreo del rendimiento para identificar cuellos de botella y mejorar la velocidad de procesamiento de datos puede mejorar la eficiencia.

7.1.7 Desafíos en la Integración de Datos

La integración de datos puede presentar varios desafíos, incluyendo:

- Data Silos (Silos de Datos): Los datos almacenados en sistemas aislados pueden ser difíciles de integrar, lo que lleva a vistas de datos incompletas o inconsistentes.
  Ejemplo: Diferentes departamentos que utilizan bases de datos separadas sin una estrategia unificada de integración de datos pueden crear silos de datos.

- Data Quality Issues (Problemas de Calidad de los Datos): La mala calidad de los datos puede llevar a análisis y toma de decisiones inexactas, socavando el valor de los datos integrados.
  Ejemplo: Formatos de datos inconsistentes y registros duplicados pueden causar errores en los informes.

- Complex Data Transformation (Transformación Compleja de Datos): Los procesos complejos de transformación de datos pueden llevar mucho tiempo y requerir habilidades especializadas.
  Ejemplo: Convertir datos de varios formatos y estructuras a un formato común para la integración puede ser un desafío.

- Scalability (Escalabilidad): Integrar grandes volúmenes de datos de múltiples fuentes puede ser un desafío, requiriendo soluciones escalables.
  Ejemplo: Manejar la integración de datos transaccionales de alta frecuencia de sistemas de comercio electrónico y financieros exige soluciones de integración de datos escalables.

8

Conferencia 8

DevOps: Introducción

8.1 Definición de DevOps y Fundamentos

DevOps es un enfoque transformador para el desarrollo y la entrega de software que fusiona filosofías culturales, prácticas y herramientas para mejorar la capacidad de una organización para entregar aplicaciones y servicios a alta velocidad. Al combinar los términos 'Development' (Dev) y 'Operations' (Ops), DevOps representa un cambio cultural que fomenta la colaboración entre equipos tradicionalmente aislados, permitiendo una evolución y mejora más rápidas de los productos en comparación con los procesos tradicionales de desarrollo de software y gestión de infraestructura. Este ritmo acelerado permite a las organizaciones servir mejor a sus clientes y mantener una ventaja competitiva en el mercado.

DevOps se define como 'un conjunto de prácticas destinadas a reducir el tiempo entre la confirmación de un cambio en un sistema y la puesta en producción normal del cambio, al tiempo que se garantiza una alta calidad'. En la práctica, se caracteriza por la propiedad compartida, la automatización del flujo de trabajo y la retroalimentación rápida. DevOps, particularmente a través de la entrega continua, se adhiere al principio de 'Bring the pain forward', abordando los desafíos al principio del proceso mediante la automatización y la detección temprana de problemas.

8.1.1 Principios Fundamentales

En su base, DevOps enfatiza:

- **Colaboración:** Romper las barreras entre los equipos de desarrollo y operaciones para permitir una comunicación fluida y la propiedad compartida del ciclo de vida del software.

- **Automatización:** Aprovechar las herramientas para automatizar cada fase del ciclo de vida del desarrollo de software (SDLC), incluyendo la planificación, la codificación, las pruebas, el despliegue y la monitorización, para mejorar la eficiencia, la consistencia y la reducción de errores.

- **Entrega Continua:** Implementar prácticas como la integración continua y la entrega continua (CI/CD) para permitir versiones rápidas, fiables y repetibles.

- **Ciclos de Retroalimentación:** Incorporar mecanismos de retroalimentación rápida para fomentar la mejora iterativa y una toma de decisiones más rápida.

8.1.2 Ciclo de Vida de DevOps

1. Planificación (Plan) - Determinación de requisitos y análisis de los objetivos de negocio. Se diseñan hojas de ruta de proyectos para optimizar el impacto empresarial y producir el resultado deseado. Crear una hoja de ruta del proyecto para maximizar el valor empresarial y entregar el producto deseado.

2. Código (Code) - El código fuente se desarrolla en entornos de control de versiones (por ejemplo, Git, GitHub, GitLab).

3. Construcción (Build) - Después de que los programadores hayan completado sus tareas, envían el código a la fuente de código común.

4. Prueba (Test) - Para asegurar la integridad del software, el producto se entrega primero a la plataforma de prueba para ejecutar varios tipos de pruebas, como pruebas de aceptabilidad del usuario, pruebas de seguridad, comprobación de la integración, pruebas de velocidad, etc.

5. Lanzamiento (Release) - En este punto, la compilación está preparada para ser desplegada en el entorno operativo. El departamento de DevOps prepara actualizaciones o envía varias versiones a producción cuando la compilación satisface todas las comprobaciones basadas en las demandas de la organización.

6. Despliegue (Deploy) - En este punto, Infrastructure-as-Code ayuda a crear la infraestructura operativa y, posteriormente, publica la compilación utilizando varias herramientas del ciclo de vida de DevOps.

7. Operación (Operate) - Esta versión ahora es conveniente para que los usuarios la utilicen. El departamento de gestión se encarga de la configuración del servidor y el despliegue en este punto.

8. Monitorización (Monitor) - El flujo de trabajo de DevOps se observa en este nivel dependiendo de los datos recogidos del comportamiento del consumidor, la eficiencia de la aplicación y otras fuentes. La capacidad de observar el entorno completo ayuda a los equipos a identificar los cuellos de botella que afectan al rendimiento de los equipos de producción y operaciones.

DevOps sigue técnicas positivas que consisten en código, construcción, prueba, lanzamiento, despliegue, operación, visualización y planificación. El ciclo de vida de DevOps sigue una serie de fases como el desarrollo continuo, la integración continua, las pruebas continuas, la monitorización continua y la retroalimentación continua.

7 Cs de DevOps

- **Continuous Development**: La fase de desarrollo continuo comienza con la obtención de requisitos y la alineación de las partes interesadas, seguida del mantenimiento de un backlog de producto derivado de los comentarios de los clientes. Este backlog se descompone en lanzamientos y hitos iterativos para permitir el desarrollo incremental de software. Una vez finalizados los requisitos, el desarrollo comienza a través de commits de código modulares y con control de versiones. Este cambio de paradigma del desarrollo monolítico a la codificación iterativa permite una rápida adaptación a los cambios de requisitos y a las optimizaciones de rendimiento.

  El proceso de CDv enfatiza las contribuciones atómicas de código que se someten a una validación inmediata a través de las etapas posteriores del pipeline. Este enfoque mejora la calidad del código a través de puntos de integración frecuentes, permite la detección temprana de defectos y facilita el enfoque del desarrollador en unidades funcionales discretas. La implementación técnica típicamente emplea sistemas de control de versiones distribuidos (por ejemplo, Git) con estrategias de branching como GitFlow para mantener la integridad del código durante flujos de desarrollo paralelos.

- **Continuous Integration**: CI constituye la práctica de desarrollo de software en la que los desarrolladores fusionan regularmente sus cambios de código en un repositorio central, tras lo cual se ejecutan builds y pruebas automatizadas. Esta práctica incorpora análisis estático de código, pruebas unitarias y verificación de la construcción a través de pipelines automatizados. Cada commit desencadena una nueva instancia de build en entornos efímeros, lo que garantiza el aislamiento y la reproducibilidad.

  Los objetivos clave de la integración continua son encontrar y abordar los errores más rápidamente, mejorar la calidad del software y reducir el tiempo necesario para validar y lanzar nuevas actualizaciones de software.

  Las implementaciones modernas aprovechan los entornos de construcción contenerizados (por ejemplo, Docker) para garantizar la coherencia entre las estaciones de trabajo de desarrollo y las plataformas de CI.

- **Continuous Testing**: CT integra la verificación de calidad en todo el pipeline de entrega a través de suites de pruebas automatizadas ejecutadas en entornos similares a los de producción. La metodología de la pirámide de pruebas guía la implementación, priorizando las pruebas unitarias (70-80%), las pruebas de integración (10-20%) y las pruebas de extremo a extremo (5-10%). Las plataformas de orquestación de contenedores (por ejemplo, Kubernetes) facilitan la ejecución paralela de pruebas en múltiples configuraciones de entorno.

  Los frameworks de automatización de pruebas (Selenium, Cypress, etc.) se integran con los sistemas de CI para proporcionar una retroalimentación inmediata sobre los impactos de la regresión. Las implementaciones avanzadas incorporan pruebas de mutación y principios de ingeniería del caos para validar la resiliencia del sistema. Las pruebas fallidas activan automáticamente tickets de defectos y mecanismos de rollback del pipeline.

- **Continuous Deployment/Delivery (CD)**: Los cambios de código se construyen, prueban y preparan automáticamente para su lanzamiento a producción. Se expande sobre la integración continua desplegando todos los cambios de código en un entorno de pruebas y/o un entorno de producción después de la fase de construcción. Cuando la entrega continua se implementa correctamente, los desarrolladores siempre tendrán un artefacto de construcción listo para ser desplegado que ha pasado por un proceso de prueba estandarizado.

  El Despliegue Continuo es el proceso de despliegue mediante la automatización del lanzamiento de cambios de código en el entorno de producción. Los cambios de código se ejecutan a través de una serie de pruebas automatizadas y, una vez que las superan, se envían inmediatamente a los usuarios del software.

  La Entrega Continua mantiene puertas de aprobación manuales pre-producción mientras automatiza todas las etapas precedentes. Envía nuevas compilaciones a un entorno de pruebas de control de calidad (QA) automatizado para buscar errores y otros problemas, y una vez que lo supera, se aprueba para su despliegue.

- **Continuous Monitoring**: CM implementa la observabilidad a través de la recolección de métricas (Prometheus), el rastreo distribuido (Jaeger) y la agregación de logs (ELK Stack). Los datos de telemetría se introducen en algoritmos de detección de anomalías que desencadenan flujos de trabajo de auto-remediación. Las métricas clave incluyen:

  - Infraestructura: Utilización de CPU/Memoria, I/O de red
  - Aplicación: Latencia, tasas de error, rendimiento
  - Negocio: Tasas de conversión, adopción de funcionalidades

  Las herramientas de visualización (Grafana) sintetizan los datos de monitorización en dashboards accionables. Las reglas de alerta siguen el framework SLO/SLI para mantener los objetivos de calidad del servicio.

- **Continuous Feedback**: CF establece canales de comunicación bidireccionales entre los usuarios y los equipos de desarrollo a través de:
  - Estructurada: Telemetría in-app, encuestas NPS
  - No estructurada: Análisis de sentimiento de tickets de soporte, redes sociales

  Los ciclos de retroalimentación se integran con los product backlogs a través de la generación automatizada de tickets (Jira, Azure DevOps). Las pruebas multivariante (A/B, canary) proporcionan datos de rendimiento cuantitativos para guiar las mejoras iterativas.

- **Continuous Operations**: CO asegura la disponibilidad de la aplicación a través de:
  - Despliegues sin tiempo de inactividad (actualizaciones rolling, rotación de clústeres)
  - Sistemas de auto-recuperación (políticas de reinicio de pods, autoescalado horizontal)
  - Recuperación ante desastres (geo-replicación, flujos de trabajo de backup/restore)

  Las plataformas de orquestación de contenedores (Kubernetes) implementan la gestión declarativa del estado para mantener los niveles de servicio deseados. Las herramientas de ingeniería del caos (Gremlin) validan proactivamente la resiliencia del sistema en condiciones de fallo.

8.2 Prácticas DevOps. Parte 1

8.2.1 CI/CD

El pipeline CI/CD es esencialmente un flujo de trabajo que proporciona una vía a través de la cual los equipos DevOps automatizan el proceso de entrega de software. En ausencia de un pipeline automatizado, los equipos tendrían que configurar su flujo de trabajo para que se realizara manualmente, lo cual consume mucho tiempo y es propenso a errores. El pipeline CI/CD elimina los errores manuales, estandariza los ciclos de retroalimentación de los desarrolladores y aumenta la velocidad de las iteraciones del producto.

Etapas:

- Etapa de código fuente (Source stage): Los desarrolladores verifican y trabajan localmente en los códigos fuente de la aplicación almacenados en un repositorio central, como GitHub. Luego, crean una nueva rama (branch) para la función o el error que desean corregir, ejecutando pruebas localmente en su entorno de desarrollo, antes de volver a confirmarlo (commit) en el repositorio de código fuente.

- Etapa de construcción (Build stage): El nuevo código confirmado en el repositorio de código fuente desencadena la etapa de construcción, en la que los códigos de la rama se recopilan y compilan para construir una instancia ejecutable de la versión. Los desarrolladores pueden construir y probar sus versiones varias veces al día para detectar errores en el código y recibir alertas si una compilación falla, o sobre otros problemas, para que el equipo pueda implementar una solución lo antes posible. Una vez que el código está libre de errores, pasa a la siguiente etapa.

- Etapa de prueba (Test stage): La compilación pasa por varias pruebas para validar el código y asegurarse de que está actuando como debería. Lo más común es que estas incluyan pruebas unitarias (unit tests), donde la aplicación se divide en pequeñas unidades de código y se prueban individualmente, y pruebas de aceptación del usuario (UAT), en las que las personas usan la aplicación para determinar si el código requiere más cambios antes de que se implemente en producción. Pruebas de carga, seguridad y otras pruebas continuas se pueden realizar en esta etapa.

- Etapa de implementación (Deployment stage): Cuando la aplicación está lista para producción, existen muchas estrategias de implementación diferentes entre las que elegir. La mejor elección depende del tipo de aplicación, la cantidad de reformas que ha sufrido, el entorno de destino y otros factores. Los siguientes son enfoques de implementación modernos para aplicaciones en la nube.

  - Implementación gradual (Rolling deployment): Este método entrega la aplicación actualizada de forma incremental hasta que todos los destinos tengan la versión actualizada. Las implementaciones graduales plantean menos riesgo de tiempo de inactividad y son fáciles de revertir, pero requieren que los servicios admitan tanto la versión nueva como la anterior de la aplicación.
  - Implementación azul-verde (Blue-green deployment): En este método, los desarrolladores ejecutan dos versiones de la aplicación en paralelo en infraestructuras separadas. La última versión estable de la aplicación se ejecuta en el entorno de producción (azul) y la nueva versión se ejecuta en un entorno de preproducción (verde). La versión verde de la aplicación se prueba en cuanto a funcionalidad y rendimiento, y a medida que pasa, el tráfico se traslada del entorno azul al verde, que se convierte en el nuevo entorno de producción. Las implementaciones azul-verde son rápidas y fáciles de implementar, pero pueden ser costosas si el entorno de producción replicado es particularmente complejo.
  - Implementación canario (Canary deployment): En este modelo de implementación, los desarrolladores lanzan una aplicación a un pequeño subconjunto de usuarios que la usan y brindan comentarios. Una vez que se confirma que la aplicación actualizada funciona correctamente, se implementa para los usuarios restantes. La estrategia permite a los desarrolladores probar dos versiones de una aplicación con usuarios reales lado a lado y permite actualizaciones y reversiones sin tiempo de inactividad.

  En cada etapa del pipeline, el equipo de desarrollo recibe alertas sobre errores para que puedan abordar el problema de inmediato. Los cambios de código pasan por el pipeline nuevamente, por lo que solo el código libre de errores se implementa en producción.

Buenas prácticas:

- Utilizar un entorno consistente: Un pipeline de CI/CD fiable siempre producirá la misma salida para una entrada particular. Pero no se puede tener un pipeline fiable si cada ejecución modifica el entorno del pipeline, por lo que se debe iniciar cada workflow desde el mismo entorno aislado.

- Aplicar analíticas de pipeline: La analítica de pipeline visualiza la telemetría de sus procesos de CI/CD, expandiendo la visibilidad e incrementando las capacidades de medición, para que sepa qué tan bien está funcionando su delivery pipeline.

- Commitear temprano y seguido: Commitear cambios temprano y frecuentemente permite a los equipos sacar el máximo provecho de CI/CD. Específicamente, dado que los commits atómicos son pequeños cambios realizados rápidamente, son más fáciles de probar, más fáciles de validar, más fáciles de revertir y proporcionan la mejor base para una iteración más rápida. Los commits frecuentes también aseguran que los cambios no se perderán en casos extremos de pérdida de la máquina o negligencia del desarrollador.

- Construir una sola vez: Los pipelines de CI/CD exitosos típicamente incluyen el proceso de build como el primer paso en el ciclo de CI/CD, y el paso se ejecuta solo una vez con la salida resultante utilizada a través de todo el pipeline. Reutilizar una sola build previene inconsistencias recién introducidas a medida que se mueve a través de las etapas del pipeline. Por ejemplo, si el proceso de deployment involucra desplegar a un entorno de pruebas, entorno de staging y entorno de producción, la automatización a menudo reconstruirá los assets cada vez. Esto significa que el binario o binarios desplegados a producción no eran realmente los mismos que los desplegados a staging y puede que no sean idénticos. Cuando esto sucede, todas las pruebas deben ejecutarse nuevamente para validar los cambios o no podemos tener la confianza de que funcionarán cuando se pongan en producción.

- Implementar control de versiones: Un sistema de control de versiones es esencial para almacenar el código fuente, rastrear los cambios de código y revertir a deployments anteriores cuando sea necesario. Asegúrese de aplicar y usar el control de versiones para scripts, documentación, librerías y configuraciones para su aplicación.
- Desarrollar incrementalmente: Trabajar en pequeñas iteraciones — dividiendo la funcionalidad en sub-funcionalidades más pequeñas y usando feature flags únicos para cada una — facilitará el aislamiento de problemas y reducirá el riesgo de problemas de integración cuando la funcionalidad se envíe a producción.
- Monitorear sus pipelines: Correlacione los datos de su pipeline con otras métricas para lograr un rendimiento general óptimo de sus aplicaciones.

8.2.2 Infrastructure as Code (IaC)

La Infraestructura como Código (IaC) es una práctica en la que la infraestructura se aprovisiona y se gestiona utilizando código y técnicas de desarrollo de software, como el control de versiones y la integración continua. Permite a los desarrolladores y administradores de sistemas interactuar con la infraestructura de forma programática y a escala, en lugar de tener que configurar y configurar los recursos manualmente. La infraestructura de IT gestionada por este proceso comprende tanto el equipo físico, como los servidores bare-metal, como las máquinas virtuales y los recursos de configuración asociados. Por lo tanto, los ingenieros pueden interactuar con la infraestructura utilizando herramientas basadas en código y tratar la infraestructura de una manera similar a como tratan el código de la aplicación. Debido a que están definidas por código, la infraestructura y los servidores se pueden implementar rápidamente utilizando patrones estandarizados, actualizarse con los últimos parches y versiones, o duplicarse de forma repetible.

Características de IaC

- Automatización: IAC automatiza el aprovisionamiento y la configuración de la infraestructura, reduciendo los errores manuales y ahorrando tiempo.
- Repetibilidad: Los scripts IAC se pueden usar repetidamente, lo que facilita la recreación de la misma infraestructura en múltiples entornos.
- Control de Versiones: El código IAC se almacena en sistemas de control de versiones como Git, lo que facilita el seguimiento de los cambios, la reversión a versiones anteriores y la colaboración con otros.
- Escalabilidad: IAC facilita el escalado de la infraestructura hacia arriba o hacia abajo, agregando o eliminando recursos según sea necesario.
- Transparencia: IAC hace que la infraestructura sea transparente y comprensible, ya que el código define los componentes de la infraestructura y sus relaciones.
- Seguridad Mejorada: IAC ayuda a garantizar que la infraestructura esté configurada de manera consistente y segura, reduciendo el riesgo de vulnerabilidades de seguridad.

Casos de Uso de IaC

- Aprovisionamiento de Máquinas Virtuales (VMs): Usando IAC, puede escribir código para aprovisionar VMs en un entorno de cloud computing, y especificar el número de VMs, el sistema operativo y el software requerido.
- Despliegue de una Red: Puede usar IAC para desplegar una red, especificar la topología de la red, crear subredes y configurar grupos de seguridad.
- Configuración de una Base de Datos: Puede escribir código para configurar una base de datos, especificar el motor de la base de datos, configurar usuarios y definir el esquema.

- Despliegue de una Aplicación Web: Puede usar IAC para desplegar una aplicación web, especificar el servidor web, configurar el servidor de aplicaciones y configurar el balanceo de carga.

- Gestión de Registros DNS: Puede usar IAC para gestionar registros del Domain Name System (DNS), automatizar la creación y eliminación de registros y garantizar la coherencia en múltiples entornos.

Herramientas:

- Terraform: Una herramienta de código abierto ampliamente utilizada de HashiCorp que automatiza la gestión de recursos en la nube y on-premises.

- Ansible: Una herramienta de automatización simple y sin agentes para gestionar la configuración, el aprovisionamiento y los deployments.

- Kubernetes: Una plataforma para automatizar el deployment, el escalado y la gestión de aplicaciones containerizadas.

8.2.3 Monitoring and Logging

La monitorización es el proceso de recopilar datos sobre la salud y el rendimiento de una aplicación o infraestructura.

El logging es el proceso de registrar los eventos que ocurren dentro de una aplicación o infraestructura. Estos eventos pueden incluir mensajes de error, advertencias y otros mensajes del sistema.

Las organizaciones monitorizan métricas y logs para ver cómo el rendimiento de la aplicación y la infraestructura impacta la experiencia del usuario final de su producto. Al capturar, categorizar y luego analizar los datos y los logs generados por las aplicaciones y la infraestructura, las organizaciones comprenden cómo los cambios o las actualizaciones impactan a los usuarios, arrojando información sobre las causas raíz de los problemas o los cambios inesperados. La monitorización activa se vuelve cada vez más importante a medida que los servicios deben estar disponibles las 24 horas del día, los 7 días de la semana y a medida que aumenta la frecuencia de actualización de las aplicaciones y la infraestructura. La creación de alertas o la realización de análisis en tiempo real de estos datos también ayuda a las organizaciones a monitorear sus servicios de manera más proactiva.

La monitorización y el logging en DevOps brindan visibilidad en tiempo real del estado del sistema, lo que permite a los equipos identificar y abordar los problemas de inmediato. Al monitorizar continuamente métricas clave como los tiempos de respuesta, la utilización de CPU y memoria y las tasas de error, los equipos obtienen una comprensión integral del comportamiento del sistema y pueden detectar y resolver de forma proactiva posibles cuellos de botella o fallas. La monitorización y el logging también ayudan en la planificación de la capacidad y la optimización de los recursos al identificar las tendencias de rendimiento y pronosticar los requisitos futuros.

La monitorización y el logging permiten una resolución de problemas eficaz. Cuando surgen problemas, los logs y las métricas detallados ayudan a identificar la causa raíz, lo que reduce el tiempo medio de resolución (MTTR). Los logs sirven como una valiosa fuente de información, capturando eventos del sistema, errores y actividades del usuario. Se pueden utilizar para rastrear transacciones, reconstruir eventos que conduzcan a fallas y proporcionar un contexto valioso durante el análisis post-mortem.

La monitorización y el logging contribuyen a la toma de decisiones basada en datos y a la mejora continua. Al analizar los datos históricos y las tendencias, los equipos pueden identificar áreas de optimización, mejorar el rendimiento del sistema y tomar decisiones informadas con respecto al escalado de la capacidad, la priorización de funciones y las actualizaciones de la infraestructura.

Herramientas Populares:

- Prometheus es una solución de monitorización de código abierto ampliamente adoptada, conocida por su escalabilidad, flexibilidad y sólidas capacidades de integración.

- Grafana es una popular plataforma de visualización de código abierto que se integra a la perfección con varias fuentes de datos, incluido Prometheus. Permite a los usuarios crear paneles y gráficos interactivos y enriquecidos, proporcionando información en tiempo real sobre el rendimiento y las métricas del sistema.

- ELK Stack (Elasticsearch, Logstash, Kibana): El ELK Stack, ahora comúnmente conocido como Elastic Stack, es una poderosa combinación de herramientas de código abierto para la gestión y el análisis de logs. Consta de Elasticsearch, un motor de búsqueda y análisis distribuido; Logstash, un pipeline versátil de procesamiento de datos; y Kibana, una plataforma de visualización basada en web. El Elastic Stack permite la recopilación, el almacenamiento y el análisis centralizados de logs de diversas fuentes. Logstash permite la ingestión, el análisis sintáctico y la transformación de logs antes de enviar los datos a Elasticsearch para su indexación y almacenamiento.

9

Conferencia 9

DevOps: Prácticas avanzadas de DevOps

9.1 Prácticas DevOps. Parte 2

9.1.1 Microservicios

Microservices es un estilo arquitectónico y un enfoque de diseño para desarrollar aplicaciones de software como una colección de servicios pequeños e independientes que trabajan juntos para formar un sistema completo. A diferencia de las aplicaciones monolíticas tradicionales, donde todos los componentes y funcionalidades están integrados de manera estrecha en una única base de código unificada, los microservicios dividen la aplicación en múltiples servicios desacoplados. Cada uno de estos servicios se enfoca en una capacidad o función específica del negocio, lo que hace que el sistema sea más modular, flexible y fácil de gestionar.

En su esencia, un microservicio es una unidad pequeña y autónoma de software que se ejecuta en su propio proceso. Esta independencia significa que cada microservicio puede ser desarrollado, desplegado, actualizado y escalado de forma separada respecto a los demás. La comunicación entre microservicios generalmente ocurre a través de protocolos ligeros, siendo los más comunes las APIs basadas en HTTP, que proporcionan interfaces bien definidas para la interacción. Este enfoque permite que diferentes microservicios sean escritos en distintos lenguajes de programación o frameworks, según lo que mejor se adapte a los requisitos particulares de cada servicio, además de permitir el reemplazo de microservicios sin causar interrupciones.

Uno de los principios clave de los microservicios es que cada servicio está limitado a un único propósito o capacidad del negocio. Por ejemplo, en una aplicación de comercio electrónico, microservicios separados podrían encargarse de la autenticación de usuarios, la gestión del catálogo de productos, el procesamiento de pedidos y la gestión de pagos. Esta clara separación de responsabilidades ayuda a los equipos a enfocarse en funcionalidades específicas, mejorando la velocidad y calidad del desarrollo.

Cada microservicio opera de manera autónoma con su propia lógica de negocio, base de datos y API. La arquitectura de microservicios ofrece flexibilidad para actualizaciones plug-and-play, lo que simplifica la escalabilidad de componentes específicos dentro de una aplicación sin afectar al sistema completo. Los contenedores son el medio principal para desplegar microservicios. Herramientas de DevOps como Kubernetes y Docker distribuyen de manera eficiente la potencia de procesamiento y otros recursos entre los microservicios.

Los microservicios aplican un ejemplo del principio open/closed: están abiertos para la extensión (usando las interfaces que exponen) y están cerrados para la modificación (cada uno se implementa y versiona de manera independiente).

Microservices ofrecen muchas **ventajas** sobre las arquitecturas monolíticas:

1. Aceleran la escalabilidad

Los equipos de DevOps pueden introducir nuevos componentes sin causar tiempo de inactividad, gracias a la operación independiente de cada servicio dentro de la arquitectura de microservicios. Pueden elegir el mejor lenguaje o tecnología para cada servicio sin preocuparse por problemas de compatibilidad. El despliegue de servicios en múltiples servidores puede mitigar el impacto en el rendimiento de componentes individuales y ayudar a las empresas a evitar el vendor lock-in (dependencia de un proveedor). Cada servicio es escalable de forma independiente según la demanda. Esto significa que cada servicio puede manejar una carga incrementada sin afectar negativamente el rendimiento de los otros servicios o de la aplicación en general.

2. Mejoran el aislamiento de fallos

   La arquitectura de microservicios está compartimentada: si un servicio encuentra una falla o error, este no se propaga a todo el sistema. Los microservicios son autónomos, por lo que la falla de un servicio no afecta significativamente a toda la aplicación.

3. Aumentan la productividad del equipo

   La arquitectura de microservicios permite que equipos pequeños y enfocados se concentren en el desarrollo, despliegue y mantenimiento de un servicio específico sin verse abrumados por las complejidades de todo el sistema. Fomenta un sentido de propiedad y especialización dentro de los equipos, permitiendo que los miembros especializados tomen decisiones informadas, iterando rápidamente y manteniendo una alta calidad del servicio en su dominio.

4. Reducción del tiempo de despliegue

   En arquitecturas monolíticas, cualquier cambio requiere redeplegar toda la aplicación. La arquitectura de microservicios permite lanzamientos más rápidos porque cada servicio evoluciona y se despliega de forma independiente, reduciendo el riesgo y el tiempo asociados a coordinar cambios en toda la aplicación. Este desacoplamiento mejora la agilidad, permitiendo implementar actualizaciones o correcciones rápidamente con mínima interrupción al sistema completo. Con este estilo arquitectónico, es posible lograr integración y despliegue continuo (CI/CD) en aplicaciones grandes y complejas, lo que mejora la velocidad de llegada al mercado.

5. Incrementan la eficiencia en costos

   La arquitectura de microservicios optimiza la asignación de recursos y el mantenimiento porque los equipos trabajan sobre servicios pequeños y bien definidos. Los esfuerzos se localizan en servicios específicos, reduciendo costos totales de desarrollo y mantenimiento. Los equipos se enfocan en funcionalidades concretas, asegurando que los recursos se utilicen eficientemente sin redundancias ni capacidad innecesaria.

   - Modularidad: Cada servicio representa una funcionalidad específica del negocio (por ejemplo, productos, usuarios, checkout, carrito de compras), haciendo el sistema modular. Los equipos autónomos pueden iterar cambios rápidamente y mantener y desarrollar cada servicio independientemente. - Eliminación de puntos únicos de fallo (SPOFs): Se asegura que los problemas en un servicio no colapsen ni afecten otras partes de la aplicación. - Escalabilidad independiente: Los microservicios individuales pueden escalarse por separado para proveer mayor disponibilidad y capacidad. En una aplicación monolítica, si se modifica una parte, es necesario redeplegar toda la aplicación. - Extensión de funcionalidades: Los equipos de DevOps pueden agregar nuevos microservicios sin afectar innecesariamente otras partes de la aplicación. Esto permite que distintos desarrolladores trabajen simultáneamente en diferentes partes sin interferencias, y que los cambios en una parte no afecten al resto.

Desventajas de los microservicios:

- **Complejidad aumentada:** Debido a que los microservicios están distribuidos, gestionar la comunicación entre servicios puede ser un desafío. Los desarrolladores pueden tener que escribir código adicional para asegurar una comunicación fluida entre los módulos. Al descomponer la aplicación en microservicios más pequeños e independientes, se crea un sistema distribuido que

debe funcionar correctamente con los demás servicios y manejar los datos de manera consistente. Gestionar esta dispersión de software se vuelve complicado sin las herramientas adecuadas.

- **Retos en despliegue y versionado:** Coordinar los despliegues y gestionar el control de versiones entre múltiples servicios puede ser complejo, lo que puede provocar problemas de compatibilidad. El despliegue de microservicios a menudo requiere el uso de herramientas como Open DevOps o Compass, así como técnicas de orquestación para automatizar los despliegues y garantizar la tolerancia a fallos.

- **Complejidad en las pruebas:** Probar microservicios implica escenarios complejos, especialmente al realizar pruebas de integración entre varios servicios. Orquestar estas pruebas puede ser complicado. La naturaleza distribuida y las interdependencias entre servicios hacen que las pruebas unitarias, de integración y end-to-end sean mucho más intrincadas. Se necesitarán herramientas y técnicas especializadas en microservicios para llevar a cabo las pruebas adecuadamente.

- **Dificultades en el Debug:** Debugging una aplicación que contiene múltiples microservicios, cada uno con su propio conjunto de logs, puede ser exigente. Un solo proceso de negocio puede ejecutarse simultáneamente en varias máquinas, lo que aumenta la complejidad.

- **Retos en la gestión de datos:** La consistencia de datos y las transacciones a través de múltiples servicios pueden ser complejas. La arquitectura de microservicios requiere una gestión y coordinación cuidadosa de los datos para mantener la integridad.

- **Seguridad:** La naturaleza distribuida y las interdependencias entre servicios también impactan en la seguridad. Es fundamental implementar mecanismos robustos para proteger la comunicación, autenticar y autorizar cada servicio, y asegurar los datos en tránsito y en reposo. Además, la complejidad del sistema exige herramientas especializadas para monitorear y responder a posibles vulnerabilidades o ataques.

Patrones de Diseño de Microservicios

Los microservicios cuentan con algunos patrones de diseño fundamentales, y comprender DevOps es una excelente manera de captar las ventajas inherentes a estos patrones. Estos métodos probados ofrecen soluciones a desafíos comunes, como dirigir el tráfico a través de un API gateway o evitar sobrecargas con un circuit breaker. Cada patrón tiene su propio método para resolver problemas específicos de los microservicios. Algunos de los patrones de diseño de microservicios más comunes incluyen los siguientes:

API gateway

Un API gateway funciona como la puerta principal para todas las interacciones de los clientes con los microservicios. Agrega las solicitudes provenientes de distintos clientes, las dirige a los microservicios correspondientes y compila las respuestas. Este patrón simplifica la experiencia del cliente, ofreciendo una interfaz unificada que integra los servicios individuales.

Circuit breaker

El patrón circuit breaker actúa como un interruptor de seguridad para la red. Cuando una llamada a un servicio falla repetidamente, el circuit breaker se activa, previniendo una mayor carga y posibles fallos en todo el sistema. Posteriormente, realiza verificaciones periódicas para detectar si el problema se ha resuelto, permitiendo una recuperación controlada. Este patrón mejora la resiliencia del sistema al manejar las interrupciones de servicio de manera elegante.

Event sourcing

El event sourcing registra los cambios en el estado de un sistema como una serie de eventos. En lugar de almacenar únicamente el estado actual, este patrón guarda la secuencia completa de acciones que llevaron a dicho estado. Este enfoque proporciona una auditoría confiable y puede simplificar transacciones complejas y la recuperación de errores. Además, permite reproducir los eventos para reconstruir estados pasados.

CQRS (Command Query Responsibility Segregation)

CQRS consiste en dividir las operaciones de la base de datos en comandos (que modifican datos) y consultas (que leen datos).

Esta separación optimiza el almacenamiento y el rendimiento de datos al escalar de forma independiente las cargas de trabajo de lectura y escritura. Es especialmente útil en sistemas donde la naturaleza y el volumen de lecturas y escrituras varían significativamente.

Saga

El patrón saga aborda transacciones complejas que involucran múltiples microservicios dividiéndolas en operaciones más pequeñas y manejables. Cada servicio se encarga de su parte de la transacción. Si un paso falla, la saga inicia acciones compensatorias para mitigar el impacto del fallo en las operaciones previas.

Este método permite realizar transacciones distribuidas manteniendo la autonomía de cada microservicio, lo que ayuda a la consistencia de datos sin un acoplamiento estricto.

Bulkhead

Nombrado en referencia a los compartimentos estancos de un barco, el patrón bulkhead limita los fallos a un solo servicio o a un grupo de servicios. Aísla partes de la aplicación de modo que si una sección se sobrecarga o falla, las demás continúan funcionando. Este aislamiento mejora la tolerancia a fallos y la resiliencia del sistema.

Database-per-service

Con el patrón database-per-service, cada microservicio tiene una base de datos dedicada. Esto evita que las llamadas a la base de datos de un servicio afecten a otros. Garantiza un acoplamiento débil y una alta cohesión, haciendo que los servicios sean más resilientes a cambios y más fáciles de escalar y mantener.

9.1.2 Gestión de Configuración(Configuration Management)

Los datos de configuración históricamente han sido difíciles de manejar y con facilidad pueden convertirse en un aspecto secundario. No son realmente código, por lo que no se incluyen inmediatamente en un sistema de control de versiones, y tampoco son datos de primera clase, por lo que no se almacenan en una base de datos principal. La administración tradicional y a pequeña escala de sistemas generalmente se realiza mediante una colección de scripts y procesos ad-hoc.

En los primeros años del desarrollo de aplicaciones en internet, los recursos de hardware y la administración de sistemas se realizaban principalmente de forma manual. Los administradores de sistemas gestionaban los datos de configuración mientras aprovisionaban y administraban manualmente los recursos de hardware basándose en dichos datos.

Un sistema de gestión de configuración consta de varios componentes. Los sistemas gestionados pueden incluir servidores, almacenamiento, redes y software. Estos son los objetivos del sistema de gestión de configuración. El objetivo es mantener estos sistemas en estados conocidos y determinados. Otro aspecto de un sistema de gestión de configuración es la descripción del estado deseado para los sistemas. El tercer aspecto principal es el software de automatización, responsable de asegurar que los sistemas y software objetivo se mantengan en el estado deseado.

Los desarrolladores y administradores de sistemas utilizan código para automatizar la configuración del sistema operativo y del host, tareas operativas y más. El uso de código hace que los cambios de configuración sean repetibles y estandarizados. Esto libera a los desarrolladores y administradores de sistemas de configurar manualmente los sistemas operativos, aplicaciones del sistema o software de servidor.

La práctica de manejar metódicamente los ajustes en la infraestructura, los componentes vinculados y las configuraciones de una aplicación se conoce como gestión de configuración. Desde variables de entorno y conexiones a bases de datos hasta configuraciones de servidores y parámetros de aplicaciones, estas opciones lo abarcan todo. Para preservar la estabilidad y prevenir problemas causados por configuraciones incorrectas en un entorno DevOps, donde los despliegues rápidos y frecuentes son la norma, la gestión de configuración se vuelve esencial.

La *Configuración como código* (*Configuration-as-a-Code*, CaaC), al igual que *Infrastructure as Code* (IaaC), define la configuración de servidores o cualquier recurso computacional. Nuevamente, como en IaaC, este código se envía a un sistema de control de versiones como parte de la canalización de despliegue de software. Esto configura automáticamente la infraestructura relevante para que esté lista para desarrollar y probar el software en cuestión.

Para definir la configuración, se obtienen los parámetros que establecen los ajustes que permitirán que el software funcione según lo esperado.

Herramientas relevantes: Ansible, Chef, Puppet

Una función importante de la gestión de configuración es definir el estado de cada sistema. Al orquestar estos procesos con una plataforma, las organizaciones pueden asegurar la consistencia a través de sistemas integrados y aumentar la eficiencia. El resultado es que las empresas pueden escalar más fácilmente sin contratar personal adicional de gestión de TI. Las compañías que de otro modo no tendrían los recursos pueden crecer implementando un enfoque DevOps.

La gestión de configuración está estrechamente asociada con la gestión de cambios, y como resultado, a veces se confunden ambos términos. La gestión de configuración se describe más fácilmente como la automatización, gestión y mantenimiento de las configuraciones en cada estado, mientras que la gestión de cambios es el proceso mediante el cual las configuraciones se redefinen y modifican para satisfacer las condiciones de nuevas necesidades y circunstancias dinámicas.

La Importancia de la Gestión de Configuración en DevOps

Consistencia: DevOps hace especial hincapié en mantener la consistencia entre distintos entornos, como desarrollo, pruebas y producción. Al garantizar que estos entornos sean idénticos, una adecuada gestión de configuración reduce la posibilidad de comportamientos inesperados causados por inconsistencias en la configuración.

Reproducibilidad: Con una buena gestión de configuración, los desarrolladores pueden replicar fácilmente entornos específicos o instancias de aplicaciones. Esto es esencial para pruebas, depuración y resolución de problemas.

Escalabilidad: La gestión de configuración es fundamental para escalar recursos según sea necesario a medida que las aplicaciones crecen. Permite aprovisionar nuevas instancias con configuraciones estandarizadas de manera eficiente.

Control de Versiones: Los equipos pueden rastrear cambios y volver a configuraciones anteriores cuando sea necesario gracias al control de versiones, una funcionalidad que suelen ofrecer las herramientas de gestión de configuración.

Reducción de Errores: Los errores humanos en la configuración pueden provocar fallos en el sistema o vulnerabilidades de seguridad. La automatización y las herramientas de gestión de configuración mitigan estos riesgos.

Mejores Prácticas para la Gestión de Configuración en DevOps

Utilizar herramientas de gestión de configuración: Herramientas como Ansible, Puppet y Chef ofrecen capacidades de automatización para gestionar configuraciones en servidores y servicios. Ayudan a garantizar consistencia y reducir errores manuales.

Infraestructura Inmutable: Considere emplear infraestructura inmutable, donde los componentes se reemplazan en lugar de actualizarse. Esto reduce la deriva de configuración y garantiza consistencia.

Control de Versiones: Almacene archivos de configuración en sistemas como Git que ofrezcan control de versiones. Esto permite colaboración entre equipos y proporciona un historial de cambios.

Variables Específicas por Entorno: Utilice variables de entorno para almacenar valores de configuración que varíen entre entornos. Esto facilita adaptar despliegues a distintas etapas.

Integración en CI/CD: Incorpore la gestión de configuración en sus pipelines de integración y entrega continua (CI/CD). Pruebas automatizadas pueden verificar la exactitud de las configuraciones antes del despliegue.

Auditorías Regulares: Realice auditorías periódicas de configuración para identificar ajustes obsoletos o inseguros. Esto fomenta el cumplimiento y la seguridad.

Gestión Segura de Secretos: Emplee herramientas seguras de gestión de secretos para manejar datos sensibles, como API keys y contraseñas. Evite codificar secretos directamente en archivos de configuración.

9.1.3 Políticas como Código (Policy as Code)

Policy as Code (PaC) es un enfoque para definir y gestionar políticas de seguridad, cumplimiento y desarrollo mediante código, almacenado y administrado en un motor de políticas centralizado. Estas políticas se aplican programáticamente dentro de un pipeline de CI/CD (Integración/Entrega/Despliegue Continuo), permitiendo la automatización de flujos de trabajo de seguridad.

Ejemplos de Políticas

- **Seguridad**:
  - Requisitos de complejidad de contraseñas
  - Estándares de cifrado para datos en reposo y en tránsito
  - Listas de control de acceso a redes (ACLs)
- **Cumplimiento**:
  - Requisitos de registro y auditoría de accesos
  - Restricciones geográficas de almacenamiento de datos
  - Programas de retención y eliminación de datos
- **Desarrollo**:
  - Requisitos de revisión de código (code review)
  - Restricciones de commits directos en ramas principales
  - Escaneo de vulnerabilidades
  - Convenciones de nomenclatura

Implementación Técnica:

Las políticas se escriben en lenguajes de alto nivel (YAML, Python) compatibles con las herramientas de la organización.

Un *motor de políticas* (policy engine) procesa estas reglas mediante consultas y devuelve decisiones en tiempo real.

La ejecución se realiza mediante llamadas API al pipeline de CI, permitiendo pruebas de seguridad sin interrumpir builds existentes.

Consideraciones Clave:

Dependencias:

- ¿Pueden las pruebas de seguridad interrumpir el *build* o despliegue?
- ¿Qué tipos de hallazgos deben escalarse a sistemas de seguimiento de incidencias?

Cambios en el código:

- When was the change committed?
- What is the magnitude of the change?

- Does this warrant additional testing or manual code review?

Business criticality of application being tested:

- ¿La aplicación maneja datos sensibles?
- Riesgos asociados a tiempo de inactividad
- Superficie de ataque de la aplicación

Beneficios:

En el contexto de pruebas de seguridad de aplicaciones, las organizaciones pueden aprovechar las políticas como código para definir las condiciones de cuándo probar, qué herramienta de prueba debe usarse y si es necesario realizar pruebas. Al codificar estos parámetros, los equipos de seguridad pueden simplificar la coordinación de múltiples herramientas AST y lograr precisión en sus flujos de trabajo de pruebas. Esto permite la aplicación consistente y automatizada de políticas de seguridad, y en última instancia, la capacidad de lograr mejor calidad de software sin comprometer la velocidad de desarrollo.

Más específicamente, la aplicación de políticas como código ayuda de estas formas importantes:

- Acelera las pruebas de seguridad. Con la aplicación automatizada de políticas, las pruebas de seguridad pueden activarse sin intervención manual y solo cuando sea necesario.
- Aumenta la eficiencia. Al eliminar la aplicación manual de políticas de la ecuación, las políticas pueden actualizarse y compartirse dinámicamente, eliminando elementos humanos innecesarios que ralentizan el proceso.
- Ayuda con el control de versiones y mejora la visibilidad. Las partes interesadas pueden ver fácilmente lo que está sucediendo en sus operaciones, y el control de versiones automatizado permite actualizaciones perfectas o la eliminación de actualizaciones en caso de problemas asociados con nuevas versiones.
- Minimiza errores y permite la validación. Con políticas automatizadas implementadas, se evitan errores causados por la participación humana. Adicionalmente, cuando las políticas se escriben en código, es fácil ejecutar actividades de validación y garantizar precisión.

9.2 Seguridad en DevOps (DevSecOps)

DevSecOps es una evolución de DevOps que incorpora la seguridad en todos los aspectos del proceso. El objetivo es abordar los problemas de seguridad desde el inicio mismo del proyecto. En este marco, no solo todo el equipo asume responsabilidad sobre el aseguramiento de calidad e integración del código, sino también sobre la seguridad. En la práctica, esto significa que los equipos discuten las implicaciones de seguridad durante la planificación y comienzan a probar vulnerabilidades en entornos de desarrollo, en lugar de esperar hasta el final. Este enfoque también se conoce como "shift left security".

DevSecOps es una práctica de seguridad de aplicaciones (AppSec) que introduce la seguridad temprano en el ciclo de vida del desarrollo de software (SDLC). Al integrar equipos de seguridad en el ciclo de entrega de software, DevSecOps amplía la colaboración entre los equipos de desarrollo y operaciones. Esto convierte la seguridad en una responsabilidad compartida y requiere cambios en la cultura, procesos y herramientas entre estos grupos funcionales. Todos los involucrados en el SDLC tienen un rol que desempeñar para incorporar seguridad en el flujo de trabajo CI/CD (Integración Continua/Entrega Continua) de DevOps.

La incorporación continua de seguridad a lo largo del SDLC ayuda a los equipos DevOps a entregar aplicaciones seguras con velocidad y calidad. Cuanto antes se incluya la seguridad en el flujo de trabajo, más rápido se podrán identificar y remediar debilidades y vulnerabilidades de seguridad. Este concepto

a veces se denomina "shift left" porque traslada las pruebas de seguridad hacia los desarrolladores, permitiéndoles corregir problemas de seguridad en su código mientras desarrollan, en lugar de esperar hasta el final del ciclo, como tradicionalmente se hacía. Por el contrario, DevSecOps abarca todo el SDLC, desde planificación y diseño hasta codificación, construcción, pruebas y lanzamiento, con bucles de retroalimentación continua e insights en tiempo real.

La seguridad es una parte clave de DevOps. Pero, ¿cómo sabe un equipo si un sistema es seguro? ¿Es realmente posible entregar un servicio completamente seguro?

Lamentablemente, la respuesta es no. DevSecOps es un esfuerzo continuo y permanente que requiere la atención de todos, tanto en desarrollo como en operaciones de TI. Si bien el trabajo nunca está realmente terminado, las prácticas que los equipos emplean para prevenir y manejar brechas pueden ayudar a producir sistemas lo más seguros y resilientes posible.

Policy-as-code ayuda a superar estos obstáculos para DevSecOps mediante:

- Proporcionar bucles de retroalimentación continua para desarrolladores. Las políticas pueden aplicarse mediante integración API para comunicar directamente actividades críticas de seguridad a los desarrolladores a través de tickets de Jira o notificaciones de Slack.

- Automatizar la toma de decisiones. Codificar las condiciones que activan eventos de seguridad basados en umbrales predefinidos para riesgo de aplicación, cambios de código y dependencias ayuda enormemente a reducir la fricción en la estandarización de AppSec para entornos ágiles. Las políticas como código eliminan la intervención manual que normalmente se requeriría para determinar si se debe probar y qué prueba aplicar.

10

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.