



DATASPHERE

W O R K S H O P

DATA FOUNDATIONS

INGENIERÍA DE DATOS Y SNOWFLAKE EN ACCIÓN

MÓDULO 1: FUNDAMENTOS EN INGENIERÍA DE DATOS Y CONCEPTOS MODERNOS



Nelson Zepeda

nelson.zepeda@datasphere.tech

Mayo 2025

WWW.DATASPHERE.TECH

Acerca de Datasphere

Datasphere es un proveedor pionero en soluciones avanzadas de análisis y IA, diseñadas para transformar organizaciones en empresas impulsadas por datos.

Fundada en **2019**, está posicionada para aprovechar la creciente demanda de soluciones basadas en datos, utilizando herramientas analíticas modernas y enfoques centrados en el cliente, ayudamos a transformar modelos de negocio y lograr objetivos estratégicos.

Datasphere ha desarrollado con éxito proyectos en El Salvador, Honduras, Bolivia, Tel Aviv, EE. UU. y Sudáfrica.



+60 Proyectos Exitosos
+6 Mercados

Nuestro Stack



www.datasphere.tech

<https://www.linkedin.com/company/datasphere-consulting/>

Agenda

- **¿Qué es una Estrategia de Datos?**
 - **Enfoques Arquitectónicos**
 - **Patrones Arquitectónicos**
 - **Bases MPP**
-

¿Qué es una Estrategia de Datos?

Una estrategia de datos es un **plan integral** que **define objetivos, establece metodologías y herramientas** necesarias para recolectar, procesar y analizar conjuntos de datos.

Data Governance

Procesos claros, políticas firmes y responsables definidos para proteger y optimizar el uso de los datos.

Data Architecture

garantiza que los datos estén seguros, accesibles, bien estructurados y listos para su análisis eficiente, permitiendo escalabilidad y flexibilidad

Data Collection

ETL / ELT

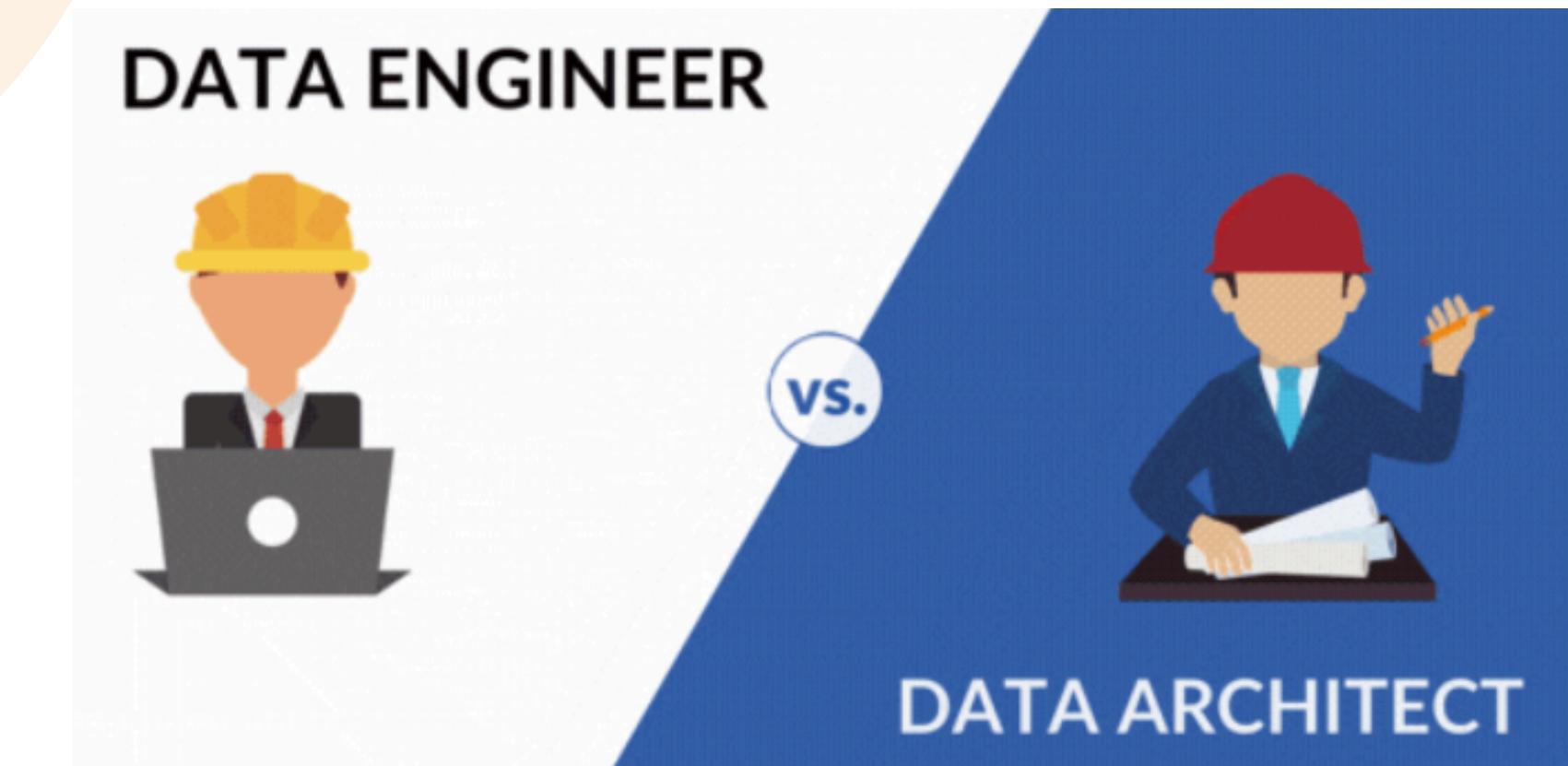
Data Analysis

Permite descubrir patrones, tendencias y correlaciones para generar insights que apoyen decisiones estratégicas: Descriptive Analytics, Predictive Analytics, Prescriptive Analytics

Data Security

protege los datos frente a accesos no autorizados, garantiza su confidencialidad y minimiza riesgos de brechas mediante medidas como encriptación, autenticación y monitoreo.

Ingeniero de Datos vs Arquitecto de Datos

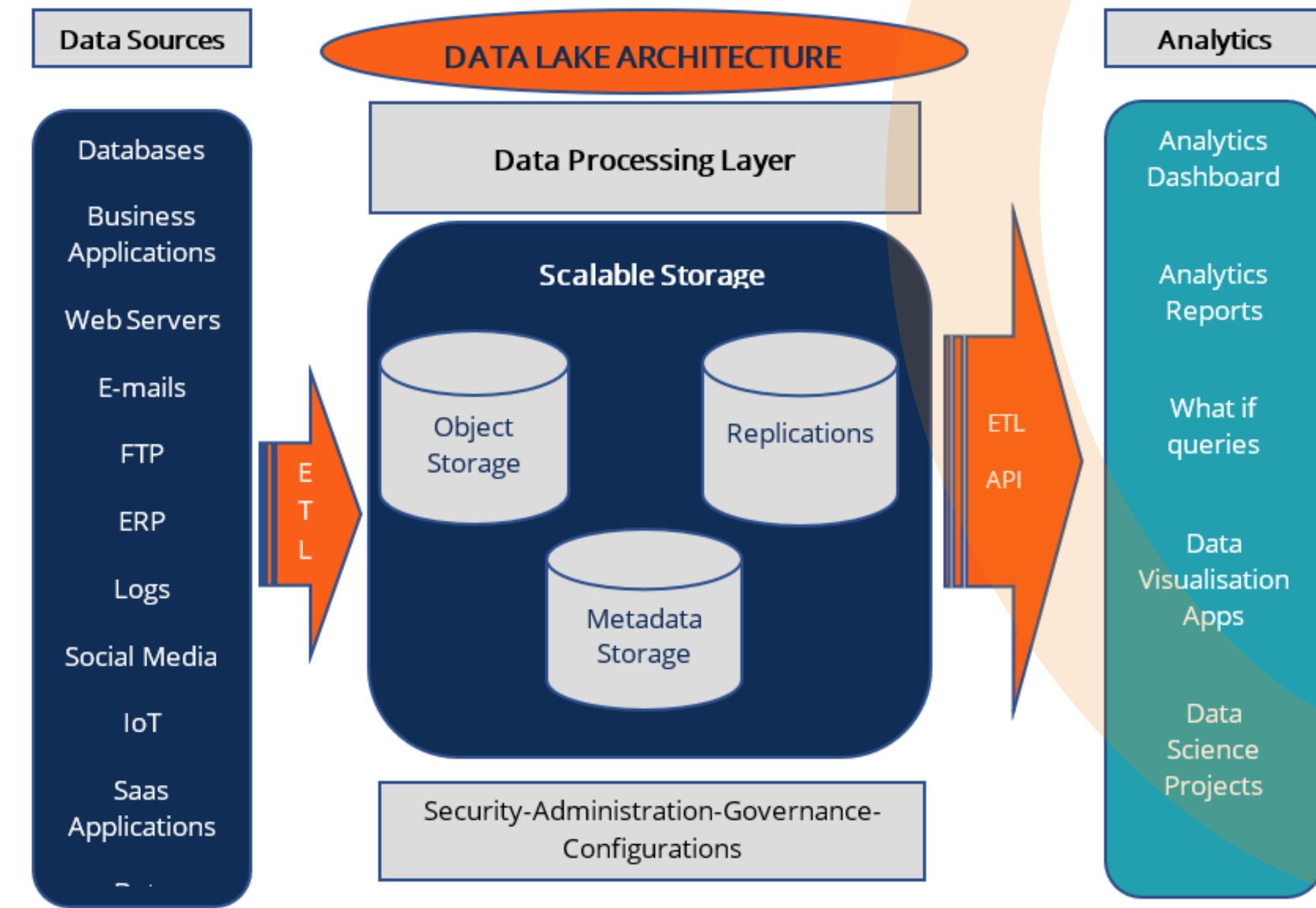


Un **Ingeniero de Datos** es el responsable de diseñar, construir y mantener los pipelines de datos que permiten a analistas, científicos de datos y otros usuarios acceder a datos de alta calidad, de forma oportuna y segura.

Un **Data Architect** es un profesional senior de tecnología que diseña, desarrolla y gestiona la arquitectura de bases de datos y sistemas de datos. Su rol combina habilidades técnicas avanzadas en diseño de sistemas con una visión estratégica para apoyar los objetivos de gestión y análisis de datos de la organización.

Enfoques Arquitectónicos

Data Lake



Un Data Lake es esencialmente un gran "**storage**" (almacenamiento) donde puedes guardar datos tal como llegan, y tú decides cómo organizarlos..

el objetivo es:

- Centralizar toda la data, aunque no sea estructurada.
- Tenerla disponible para otros procesos

El Data Lake no es sólo para hacer consultas SQL, es un espacio donde los datos viven hasta que alguien o algo los necesite, en cualquier forma.

Almacenamiento masivo y escalable: Un sistema para guardar grandes volúmenes de datos, de cualquier tipo y tamaño. Ejemplos: AWS S3, Azure Data Lake Storage, Google Cloud Storage, o incluso almacenamiento on-premise (como un clúster HDFS).

Organización lógica de datos: Definir una estructura de carpetas, zonas y metadatos para no convertir el Data Lake en un "data swamp".

Herramientas de ingestión y acceso: Métodos para subir datos y leerlos cuando se necesiten.

¿Qué necesitas para tener un Data Lake?



AWS Lake Formation



Azure Data Lake Storage Gen2

BigLake



databricks

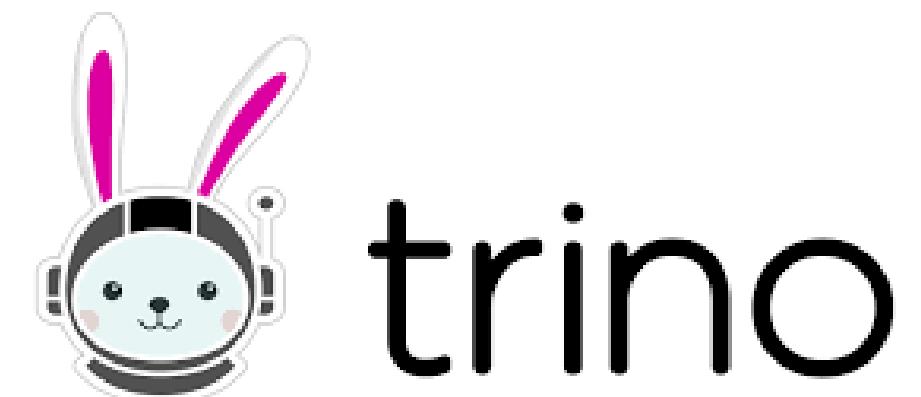
CLOUDERA

Open-source



Motores de procesamiento distribuido:

- Apache Spark: Procesamiento paralelo para ETL, análisis y Machine Learning.
- Presto: Consultas SQL distribuidas de alta velocidad sobre múltiples fuentes de datos.
- Trino: Evolución de Presto, más potente para cargas analíticas modernas.



Línea de tiempo: Evolución del Data Lake

- 2006 – Yahoo! desarrolla Hadoop
- 2008 – Aparece Hive en Facebook
- 2009 – Spark inicia en UC Berkeley
- 2012 – Presto nace en Facebook
- 2014 – Data Lake como término popularizado por Pentaho
- 2015 – MinIO lanzado
- 2016 – Apache Hudi nace en Uber
- 2018 – Netflix lanza Apache Iceberg
- 2019 – Databricks lanza Delta Lake
- 2019 – Cloudera + Hortonworks lanzan Cloudera Data Platform (CDP)
- 2020 – Presto se convierte en Trino
- 2022 – Google presenta BigLake
- 2025 – Consolidación del modelo Lakehouse: Modelos como Databricks Lakehouse, BigLake.



Cuando se almacenan datos en un Data Lake, los archivos siguen un esquema: un conjunto de columnas con nombres y tipos definidos.

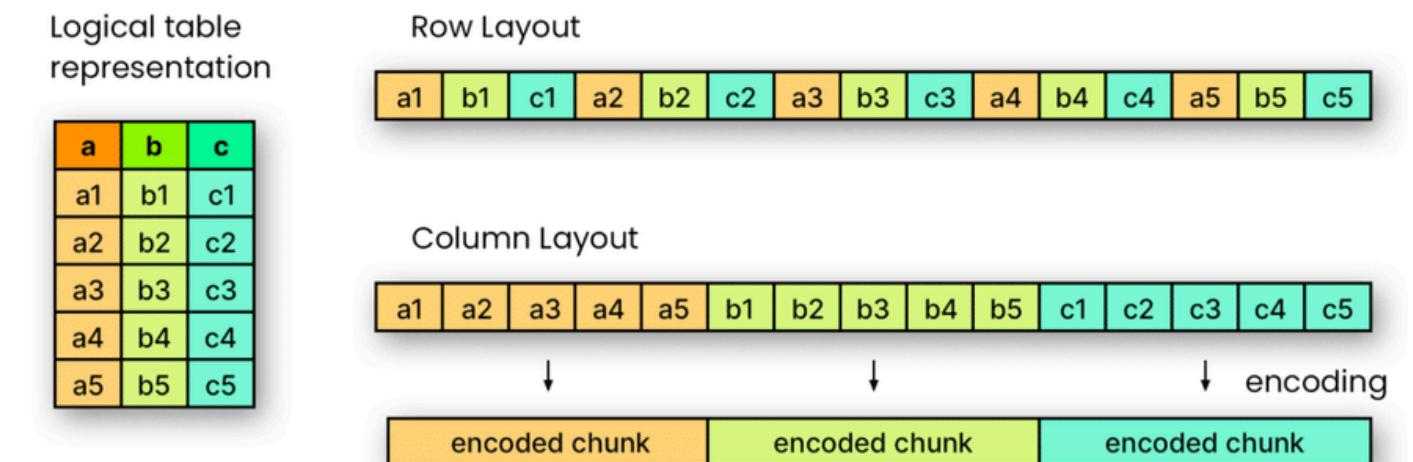
Sin embargo, en entornos reales:

- Las fuentes cambian (nuevas columnas, columnas eliminadas, cambios de tipo).
- Los datos se escriben con versiones distintas del esquema (ej. día 1 tiene 10 columnas, día 15 ya tiene 12).

El schema evolution permite que el sistema:

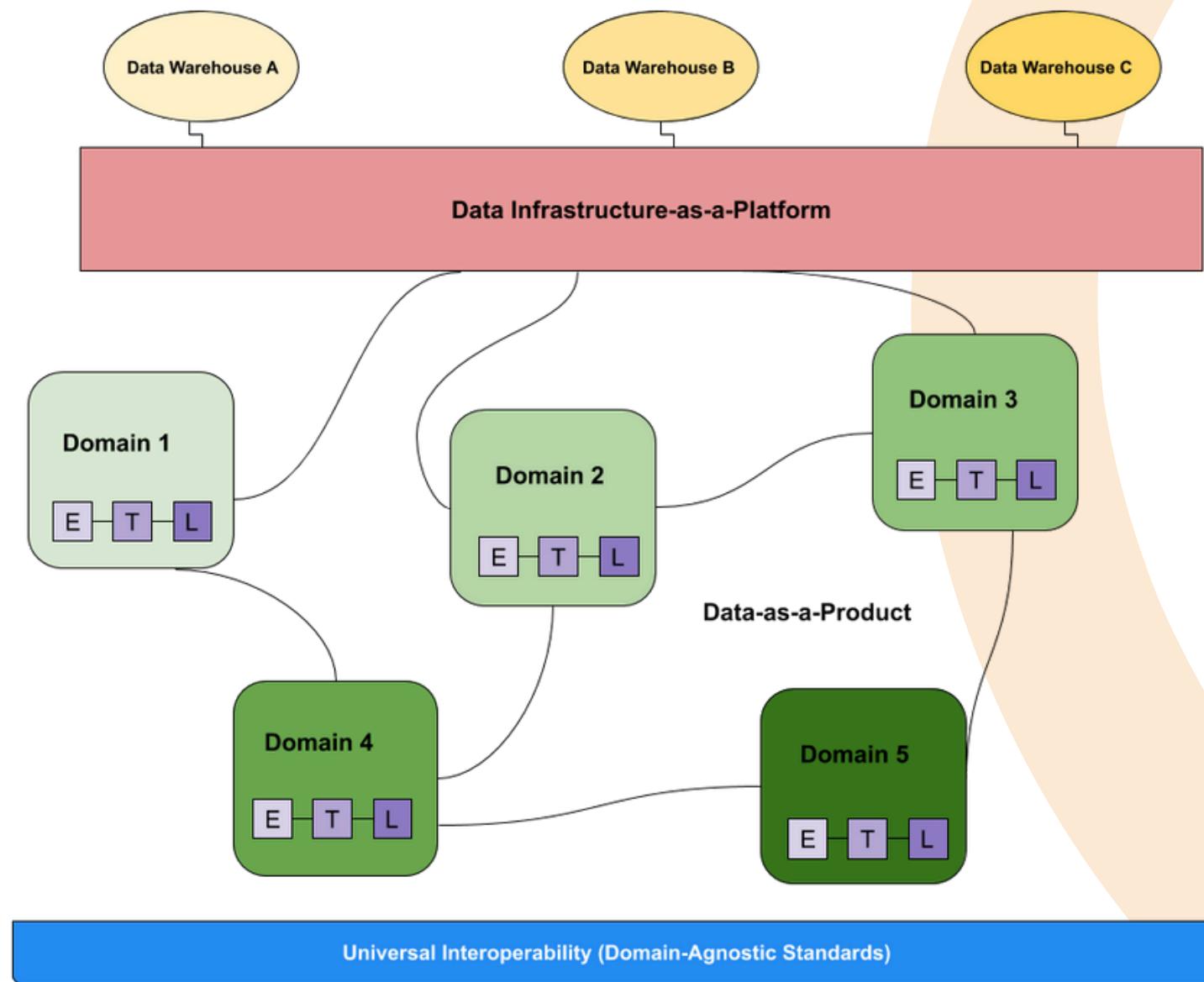
1. Reconozca los cambios automáticamente.
2. Permita lectura y escritura con distintas versiones del esquema.
3. Adapte consultas sin fallar ante registros con columnas nuevas o faltantes.

Schema Evolution



Parquet es un formato columnar, comprimido y autocontenido, diseñado para el análisis eficiente de grandes volúmenes de datos en arquitecturas distribuidas y modernas de almacenamiento.

Data mesh



El Data Mesh es un enfoque arquitectónico que promueve la descentralización de la gestión y el control de los datos. En lugar de concentrar todos los datos en un único repositorio centralizado, como en los almacenes de datos tradicionales, el Data Mesh propone dividir los datos en múltiples dominios, donde cada dominio es gestionado de forma independiente por equipos responsables.

Cada equipo trata sus datos como un producto, garantizando calidad, accesibilidad y gobernanza bajo una infraestructura común de autoservicio.

¿Qué implica tratar los datos como productos?

- Tener un dueño
- Ser usable y accesible
- Ofrecer calidad garantizada
- Estar monitorizados y versionados
- Seguridad y cumplimiento

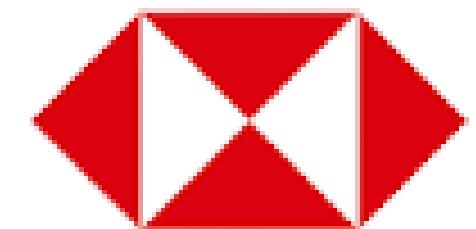
Cada dominio es dueño de sus propios productos de datos, y debe diseñarlos con la misma seriedad, calidad y enfoque al usuario que si estuviera lanzando un producto al mercado.

Netflix, Zalando, Intuit, JPMC (JP Morgan Chase), HSBC, Spotify, Booking.com, ThoughtWorks



zalando

Booking.com

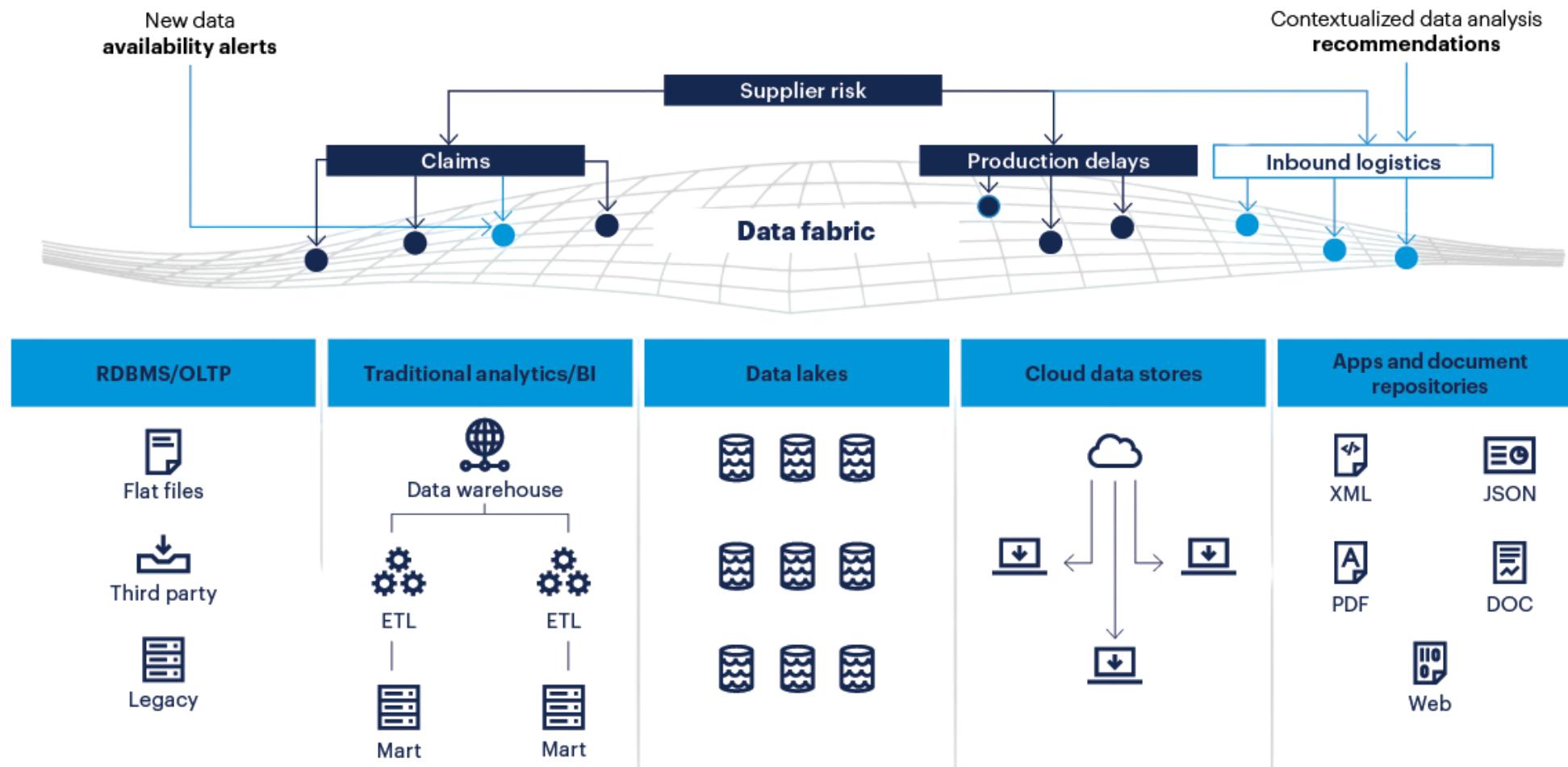


HSBC



Spotify®

Data Fabric Is an Integrated Layer of Connected Data



Source: Gartner
 © 2024 Gartner, Inc. and/or its affiliates. All rights reserved. 2737624

Gartner®

Data Fabric

Data Fabric es un enfoque que permite conectar, gestionar y acceder a todos los datos en tiempo real, a través de diferentes sistemas y aplicaciones. Facilita la creación de una única fuente de verdad, democratizando y automatizando los procesos de gestión de datos.

Un Data Fabric unifica, limpia, enriquece y asegura los datos, haciéndolos accesibles y listos para su uso en análisis, inteligencia artificial y machine learning, incluso en arquitecturas distribuidas y complejas.

Componentes de Data Fabric

Conectores de Datos

Enlazan diferentes fuentes de datos (bases de datos, aplicaciones, sensores, APIs) hacia una vista unificada.

Gestión de Datos

Organización, calidad, gobernanza y seguridad de los datos.

Incluye integración de datos, reglas de uso, protección contra accesos no autorizados.

Modelado de Datos y Capa Semántica

Creación de modelos de datos unificados.

Definición de un lenguaje común para describir y entender los datos en toda la organización.

Procesamiento y Analítica de Datos

Almacenamiento (Data Warehousing).

Procesamiento en tiempo real (Data Streaming).

Visualización de datos (dashboards, reportes interactivos).

Automatización de la Gestión de Datos

Automatización de tareas como integración, gobernanza y seguridad usando machine learning y reglas inteligentes.

Mejora la calidad, reduce errores y optimiza el uso de datos.

L'ORÉAL

vodafone

SIEMENS

citibank

American Airlines

Carrefour

ING BANK

Data warehouse



Un Data Warehouse (DW) es un almacén centralizado de datos estructurados, diseñado para consultas analíticas rápidas y complejas, donde los datos vienen de múltiples sistemas operacionales y se integran de forma consistente, limpia y optimizada.

¿Cuándo es ideal un Data Warehouse?

- Cuando se requieren reportes financieros, regulatorios o históricos
- Cuando las fuentes de datos son estables y bien definidas
- Cuando la latencia no es crítica
- Cuando la gobernanza de datos es prioritaria

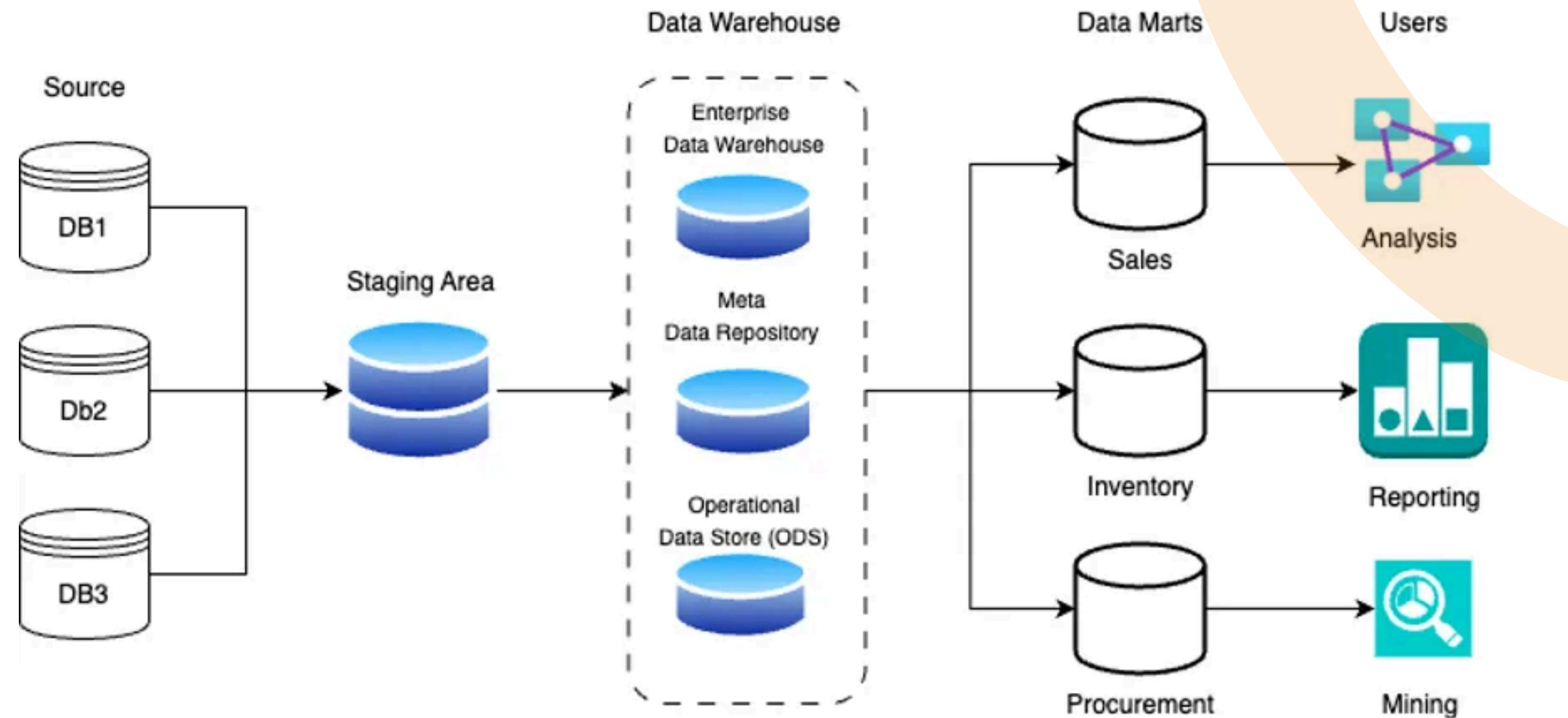
Inmon

Bill Inmon, conocido como "el padre del Data Warehouse", propuso en los años 90 que un Data Warehouse debe ser una única fuente centralizada de datos integrados, diseñada para soportar decisiones estratégicas.

El enfoque de Inmon se resume así:

Primero modelas los datos a nivel empresarial (modelo normalizado).

Luego creas Data Marts para áreas específicas si es necesario, pero los Data Marts derivan del Data Warehouse.



3NF

La 3NF o Tercera Forma Normal es un principio de diseño de bases de datos que busca eliminar redundancias y dependencias innecesarias entre los datos, para asegurar que cada dato esté almacenado de forma limpia, organizada y consistente.

Nombre Cliente	Dirección	Producto Comprado	Precio Producto
Ana López	Calle 1, Madrid	Laptop	1000
Ana López	Calle 1, Madrid	Mouse	25
Juan Pérez	Calle 2, Sevilla	Laptop	1000
Juan Pérez	Calle 2, Sevilla	Teclado	50

Problema:

en esta tabla, si Ana cambia de dirección, hay que actualizar cada fila donde ella aparece.

Si el precio de la Laptop cambia, tienes que actualizarlo en cada fila donde alguien la haya comprado.

La normalización divide esa lista en varias tablas pequeñas y organizadas, evitando repetir la misma información una y otra vez.

Tabla Cliente

ID Cliente	Nombre Cliente
1	Ana López
2	Juan Pérez

Tabla Orden

ID Orden	ID Cliente
1	1
2	1
3	2
4	2

Primera Forma Normal (1NF)

Asegura que cada campo tenga un solo valor atómico (sin listas ni múltiples datos en un mismo campo).

Segunda Forma Normal (2NF)

Elimina dependencias parciales: cada atributo no clave debe depender de toda la clave primaria, no solo de una parte.

Tercera Forma Normal (3NF)

Elimina dependencias transitivas: cada atributo no clave debe depender directamente de la clave primaria, no a través de otro atributo.

Cuarta Forma Normal (4NF)

Elimina dependencias multivaluadas: evita que una fila tenga múltiples listas de valores independientes

Tabla Producto

ID Producto	Producto
1	Laptop
2	Mouse
3	Teclado

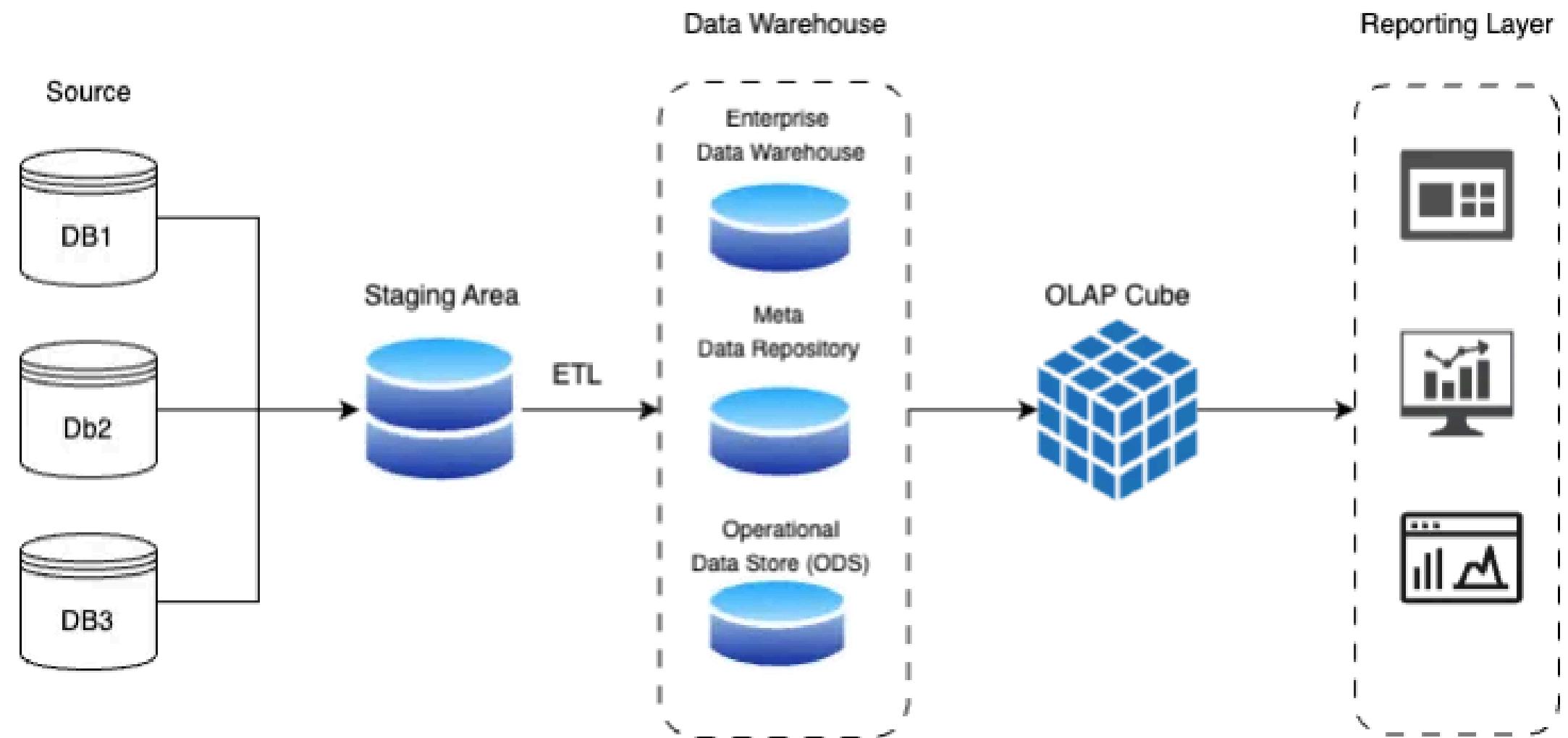
Kimball

¿Qué es el enfoque Kimball en Data Warehousing?

Ralph Kimball propuso un enfoque diferente al de Inmon:

Construir primero Data Marts individuales, modelados dimensionalmente (esquemas estrella), y luego integrarlos para formar un Data Warehouse.

Empiezas pequeño (por área o proceso) y entregas valor rápido.



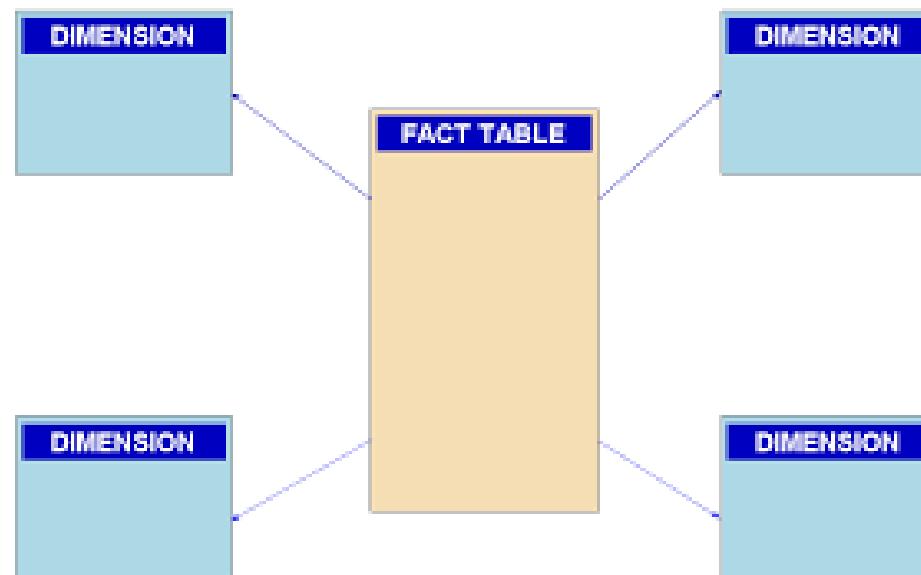
Desnormalización

La desnormalización es el proceso de volver a combinar o duplicar datos en una base de datos para hacerla más rápida y fácil de consultar, sacrificando un poco de la pureza y eficiencia en almacenamiento que se logra con la normalización.

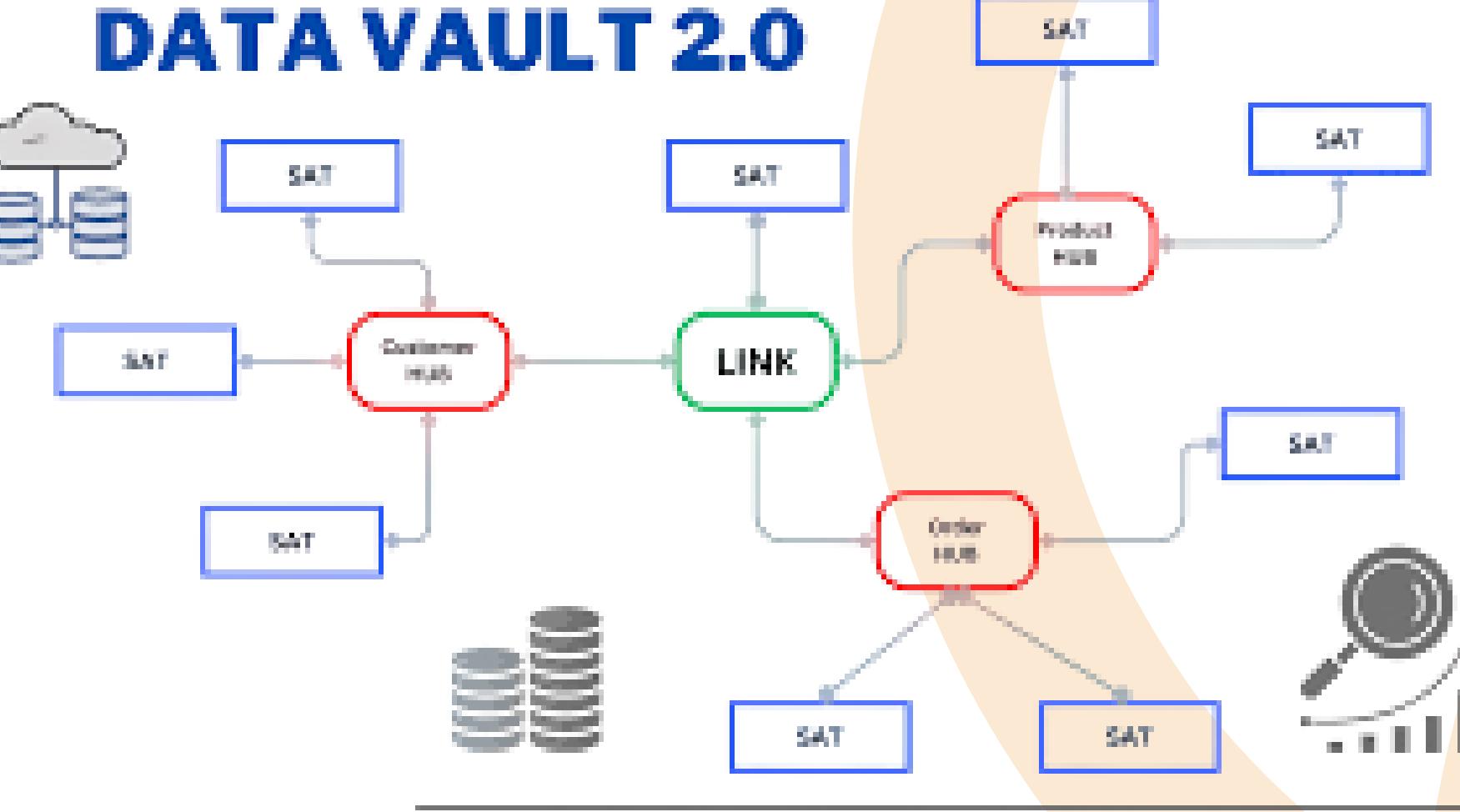
En Kimball, los datos se desnormalizan a propósito en esquemas dimensionales (estrella o copo de nieve) para optimizar la velocidad de acceso y la facilidad de análisis.

Por eso modela los datos usando esquemas dimensionales como:

- Esquema Estrella (Star Schema)
- Esquema Copo de Nieve (Snowflake Schema) (un poco más normalizado, pero sigue siendo fácil de consultar)



Data vault



Data Vault es un enfoque de modelado de datos para Data Warehouses que combina lo mejor de Inmon (rigor e integración empresarial) y Kimball (agilidad y flexibilidad), pero además lo hace mucho más escalable y adaptado a ambientes de cambio constante.

Fue creado por Dan Linstedt en los años 2000.

Alta flexibilidad para cambios de negocio.
Alta trazabilidad histórica.
Modelo orientado a carga masiva y resiliente al cambio.

Hub

Representa una entidad de negocio central.
 Contiene solo la clave única de la entidad (por ejemplo, ID Cliente, ID Producto).
 Ejemplos: Hub_Cliente, Hub_Producto.

Link

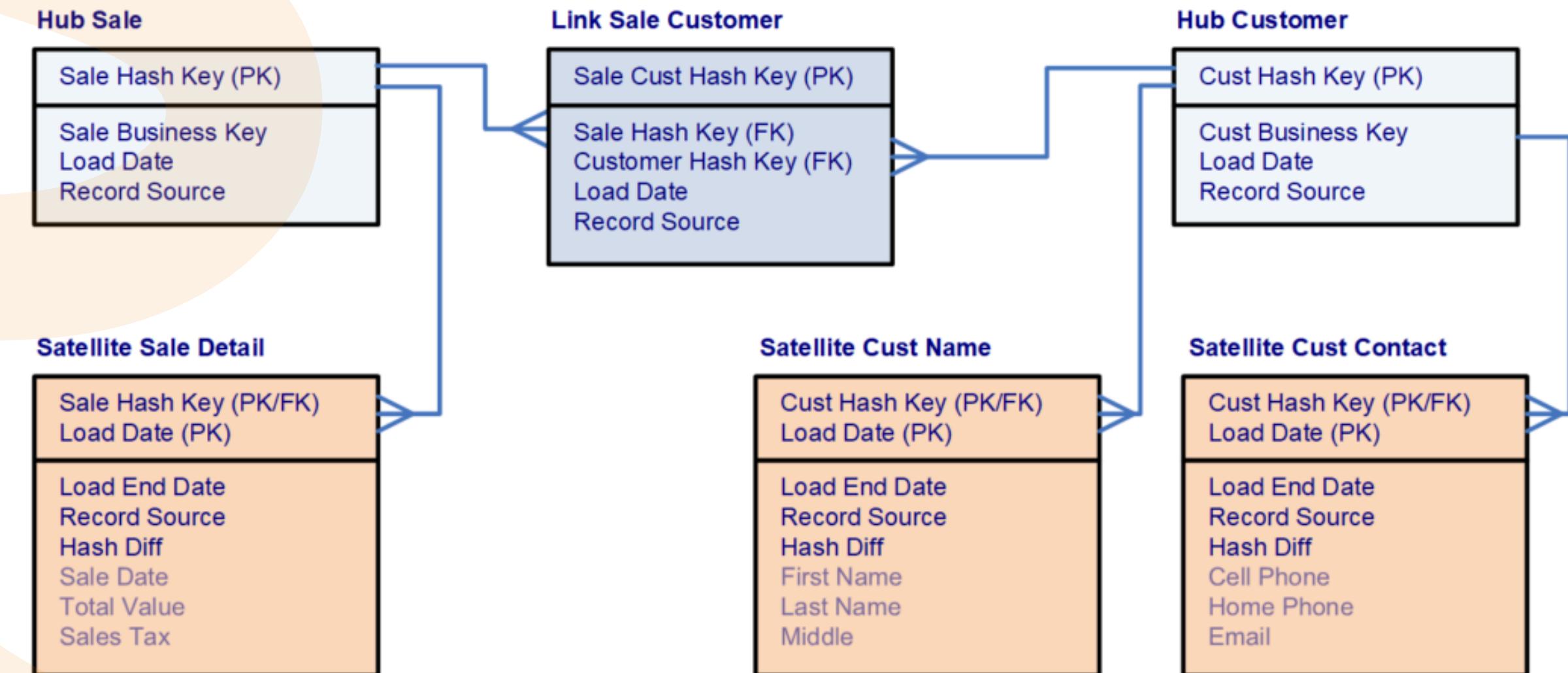
Representa una relación entre dos o más Hubs.
 Guarda las llaves primarias de los Hubs relacionados.

Ejemplos: Link_Cliente_Orden,
 Link_Empleado_Departamento.

Satellite

Contiene atributos descriptivos o detalles asociados a un Hub o Link.
 Permite historizar cambios en los datos (ej: cambios de nombre, dirección, estado).

Ejemplos: Satellite_Cliente_Detalle,
 Satellite_Orden_Detalle.



Componentes

Otros enfoques arquitectónicos

Data Lakehouse

Combina lo mejor de los Data Lakes y Data Warehouses: almacenamiento flexible + consultas transaccionales y de alta calidad.

Event-Driven Architecture (EDA) aplicada a datos

Tratar cada cambio en los datos como un "evento" que puede ser capturado y procesado en tiempo real.

Data Virtualization

Consultar múltiples fuentes de datos como si fueran una sola, sin replicarlas, usando una capa de abstracción.

Streaming Data Architecture

Procesar y analizar datos en movimiento (streaming) mientras se generan, no sólo en batch.

Patrones Arquitectónicos

Un patrón arquitectónico es una solución probada y reutilizable a un problema común en el diseño de sistemas complejos.

No es una tecnología específica.

No es una implementación exacta.

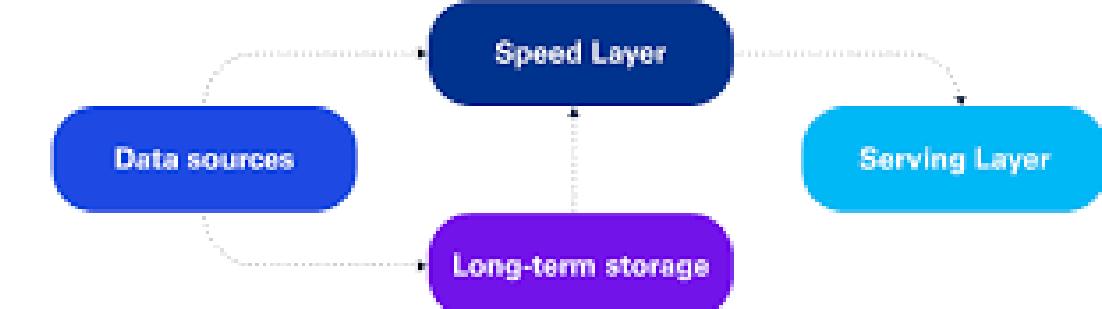
Patrones

Lambda Architecture

Combina procesamiento batch y procesamiento en tiempo real.

Usa dos capas separadas (batch layer + speed layer).

Ideal cuando necesitas alta precisión histórica y actualizaciones rápidas.



Kappa Architecture

Todo se procesa como flujos de datos en tiempo real (streaming).

Elimina la necesidad de un batch separado.

Ideal para aplicaciones 100% en tiempo real, más simple que Lambda.

Medallón Architecture

Organiza los datos en capas de calidad: Bronze (raw), Silver (limpio), Gold (curado para negocio).

Mejora la calidad de datos, la trazabilidad y facilita la recuperación ante errores.

Lambda

Supongamos que somos el equipo de ingeniería de una empresa startup que está desarrollando un juego móvil.

Cada mañana, Bob, el CEO, nos envía un mensaje por Slack diciendo:

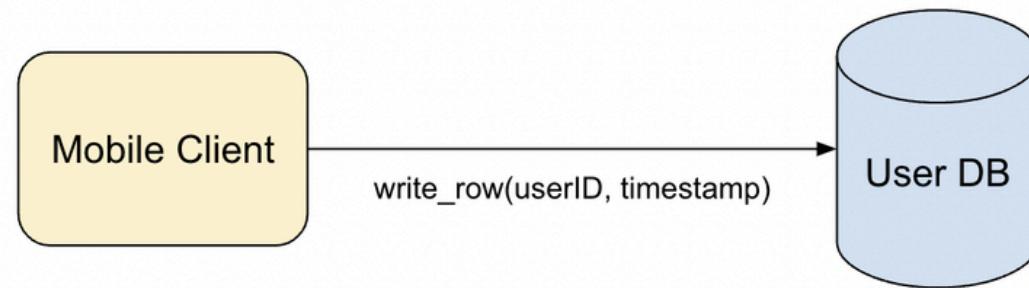
Tenemos una arquitectura simple donde, al iniciar el juego, nuestro cliente del juego registra el ID de usuario del jugador y la marca de tiempo en nuestra base de datos SQL.



Richard Sun
Software Engineer at Google



Bob (CEO) 8:26 AM
how many players were there yesterday?



```
SELECT COUNT(DISTINCT player_id) FROM player_table WHERE DATE(`timestamp`) = DATE_SUB(CUR_DATE(), INTERVAL 1 DAY)
```

Le respondemos a Bob - "21,367 Jugadores!"

Esta arquitectura es inicialmente lo suficientemente escalable para nuestra base de jugadores relativamente pequeña. Sin embargo, poco a poco, nuestro juego se convierte en un éxito masivo y explota en popularidad. Ganamos millones de jugadores en todo el mundo y estamos muy emocionados por el potencial de crecimiento continuo.

Desafortunadamente, nuestra base de datos SQL comienza a **colapsar** debido al aumento de la carga. El CPU alcanza **picos del 100%** y la confiabilidad **disminuye**: comenzamos a perder datos. Para soportar esta carga, consideramos alquilar nodos SQL más potentes (escalado vertical) y fragmentar nuestra base de datos (escalado horizontal). Sin embargo, hacemos algunos cálculos rápidos y nos damos cuenta de que cualquiera de las dos opciones resultaría en **costos de infraestructura prohibitivamente altos** para soportar la escala que necesitamos.



Decidimos crear algo mas robusto

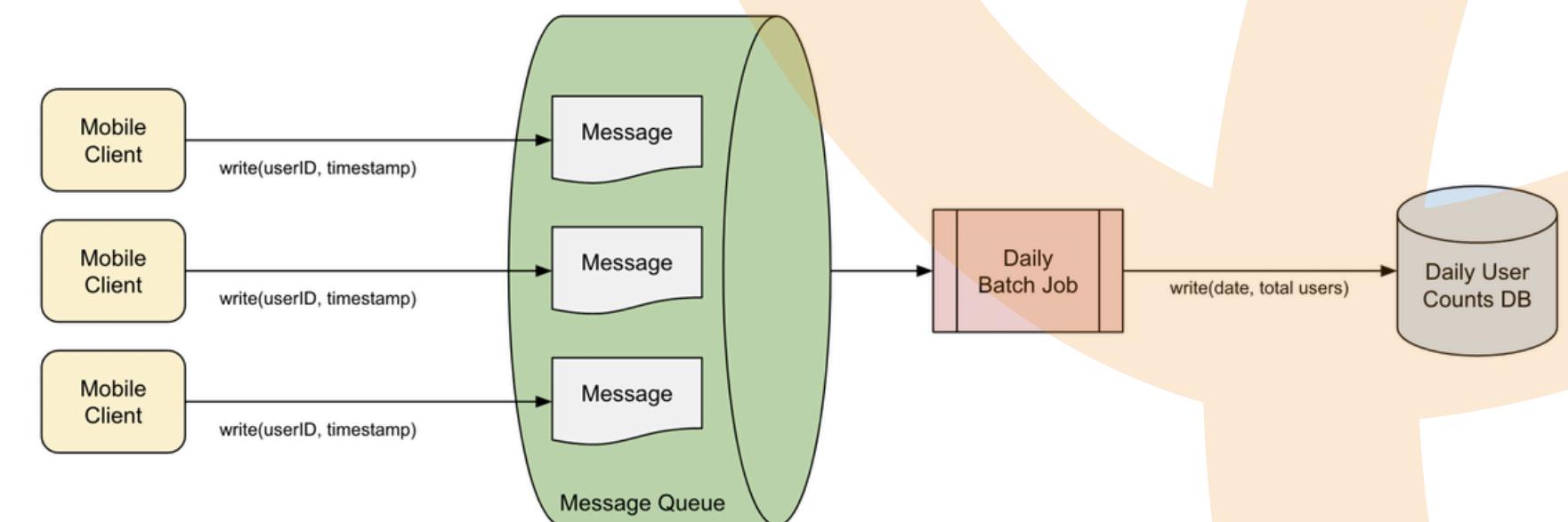
Capa Batch

Para manejar nuestro volumen creciente de solicitudes, necesitamos una forma más eficiente de registrar los IDs de usuario y las horas. Decidimos escribir en una cola de mensajes en su lugar, lo que puede soportar de manera eficiente un alto número de solicitudes de escritura. Después de escribir en la cola de mensajes procedemos a calcular el número de jugadores únicos por día mediante un trabajo batch que se ejecuta al final de cada día.

Nuestro trabajo lee todos los mensajes escritos el día anterior, cuenta los IDs de usuario únicos y escribe el resultado en una base de datos. El job tarda algunas horas en completarse y los resultados deberían estar listos por la mañana.



Bob (CEO) 8:26 AM
how many players were there yesterday?



Revisamos el job de ayer y vimos que se completó. Consultamos el resultado y le respondemos a Bob: "¡20,340,911 jugadores!"

Nuestro sistema funciona como se espera y el número de jugadores sigue creciendo, la vida es buena.



Un día, recibimos un mensaje diferente de Bob en Slack:



Bob (CEO) 4:24 PM

roughly how many ppl played today so far??

Sin embargo, solo tenemos los números de **ayer**. Rápidamente lanzamos una ejecución manual del trabajo batch para los datos de hoy. Después de unas horas, cuando el trabajo se completa, respondemos a Bob con el número. Sin embargo, Bob ya ha terminado su jornada laboral y está desconectado.



Para poder manejar preguntas como esta de manera más rápida en el futuro, decidimos construir un sistema en **tiempo real**.

Brevemente reconsideramos nuestro diseño anterior de una base de datos **SQL** que pudiera ser consultada rápidamente, pero recordamos las limitaciones de escalabilidad.

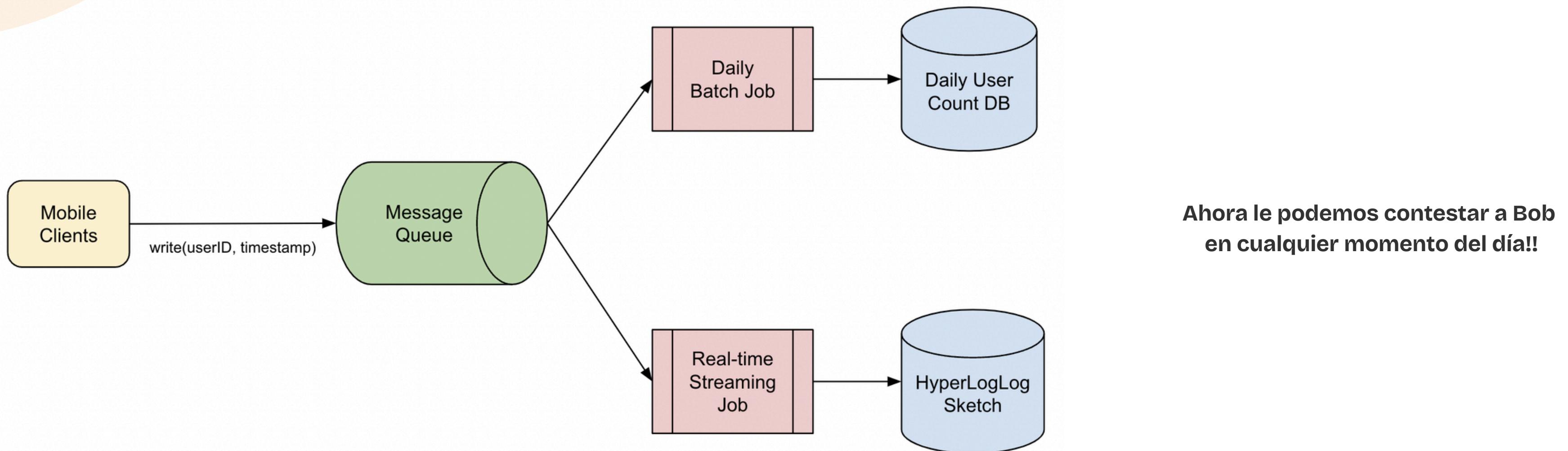
También consideramos ejecutar el trabajo batch cada pocos minutos para tener siempre datos actualizados, pero el trabajo batch es **costoso** y solo podemos permitirnos unas **pocas ejecuciones por día**.

Recordamos haber aprendido sobre el streaming de datos y cómo puede ser utilizado para manejar datos en tiempo real, así que decidimos construir un trabajo de streaming. Configuramos nuestro trabajo de streaming y lo agregamos como un consumidor adicional de nuestra cola de mensajes, donde constantemente recupera los últimos IDs de usuario de la cola.

Consideramos hacer que el trabajo de streaming mantuviera un conjunto de IDs de usuario únicos para llevar la cuenta distinta, pero eso requeriría memoria proporcional al número de jugadores —demasiado costoso.



Como a Bob le basta con un número aproximado en tiempo real, decidimos usar un contador de cardinalidad probabilístico, como **HyperLogLog**, que puede aproximar el número de valores únicos con un alto grado de precisión y utilizando solo una fracción de la memoria.



Serving Layer

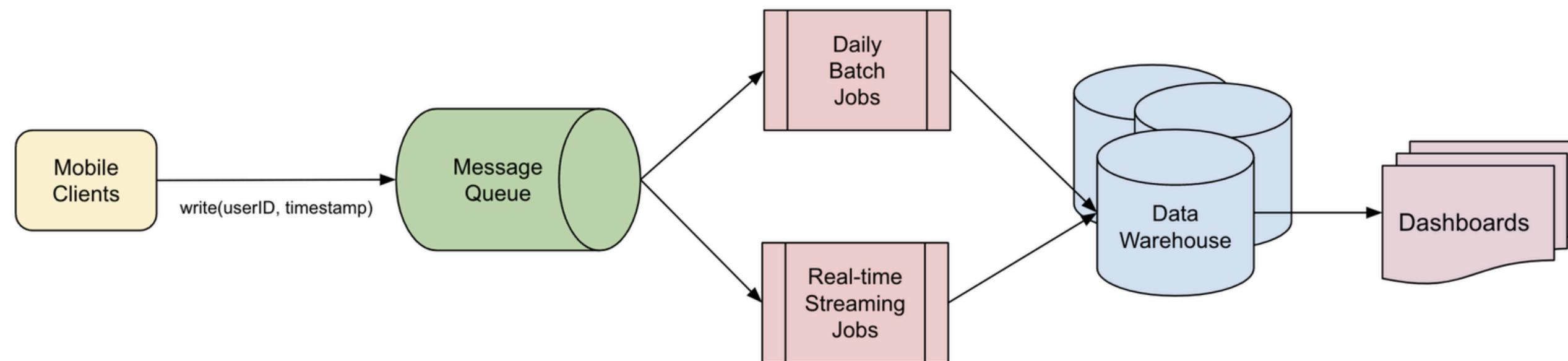
Nuestro juego continúa aumentando en popularidad, y tanto nuestros sistemas batch como en tiempo real siguen escalando exitosamente.

Con el tiempo, se añaden más casos de uso de negocio, como: ¿Cuántas partidas se juegan por día? ¿Cuántos jugadores por país? ¿Cuántos anuncios se muestran por hora? etc.

Bob también hace crecer la empresa contratando analistas de datos y formando un equipo de negocio.

Como resultado, necesitamos una forma más eficiente de ofrecer los datos a ellos, en lugar de ejecutar consultas manuales y responder por Slack,
decidimos construir una capa de servicio (serving layer).

Unimos las diferentes bases de datos que tenemos detrás de una única interfaz, y construimos herramientas y dashboards para que el equipo de negocio pueda visualizar los datos y ejecutar reportes de manera ad-hoc.



Kappa

Es un patrón de arquitectura orientado al streaming.

Ingiere datos en tiempo real, batch o casi tiempo real (como CDC) hacia un sistema de mensajería (ej: Apache Kafka).

Un motor de procesamiento de streams (ej: Spark Streaming, Flink) transforma los datos y publica los resultados nuevamente en el sistema de mensajería.

Los datos son enviados a: Data Lakes, Data Warehouses, sistemas de alerta, reportes, dashboards y sistemas de machine learning.

¿Cómo se diferencia de Lambda Architecture?

Lambda Architecture tiene dos capas separadas: batch + speed (streaming).

Kappa Architecture solo tiene una capa de streaming: todo se procesa como flujo de datos.

Beneficios de Kappa Architecture

Acoplamiento débil entre las fuentes de datos y la capa de servicio (gracias al sistema de mensajería como Kafka).

Almacenamiento de datos históricos en los sistemas de mensajería para re-procesamientos futuros.

Mayor agilidad en sistemas de análisis en tiempo real.

Casos de uso típicos de Kappa Architecture

Reportes y dashboards en tiempo real.

Procesamiento de reglas y alertas en streaming .

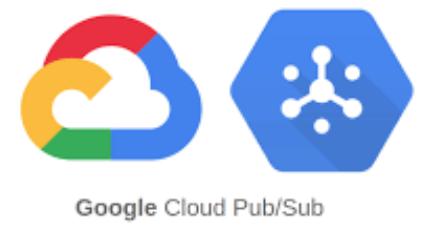
Ejecución de modelos de machine learning en tiempo real .

Inteligencia operacional en tiempo real (industria pesada, IT/OT).

Un sistema de mensajería es una capa intermedia donde los datos se reciben, almacenan temporalmente y se distribuyen como flujos de eventos. No es exactamente una "stage" de datos en el sentido tradicional de un ETL batch clásico, pero cumple un rol similar de buffer o staging para flujos continuos.

En Kappa Architecture, el sistema de mensajería actúa como una "capa staging viva" de eventos:

- Datos nunca se detienen.
- Datos son procesados inmediatamente o poco después de llegar.
- **Apache Kafka**
 - El estándar más popular para arquitecturas de streaming de datos distribuidos.
- **Amazon Kinesis**
 - Servicio de streaming de datos completamente gestionado en AWS.
- **Azure Event Hubs**
 - Solución de ingestión de eventos en tiempo real en la nube de Microsoft Azure.
- **Google Pub/Sub**
 - Servicio de mensajería distribuida en Google Cloud Platform.
- **Apache Pulsar**
 - Alternativa moderna a Kafka.
 - Nativo en soporte multi-tenant, arquitectura basada en segmentos y separación clara de cómputo y almacenamiento.



Medallón



La Arquitectura de Medallón es un patrón de diseño de datos que organiza la información en capas progresivas de calidad

- **Bronze Layer (Raw Data)**
 - Captura datos crudos directamente desde las fuentes (bases de datos, archivos, servicios de streaming).
 - Conserva el estado original para auditoría y trazabilidad.
- **Silver Layer (Refined Data)**
 - Limpia, transforma, valida y enriquece los datos.
 - Aplica un esquema consistente para generar datasets estructurados y confiables.
- **Gold Layer (Aggregated Data)**
 - Contiene datos altamente curados y optimizados.
 - Preparado para consumo en dashboards, KPIs, reportes analíticos y de negocio.

Beneficios principales de la Arquitectura de Medallón

- Escalabilidad independiente
- Cada capa puede crecer o optimizarse según las necesidades específicas de volumen o procesamiento.
- Mejora progresiva de la calidad de datos
- Gobernanza de datos simplificada
- La separación clara de etapas facilita cumplir normas, hacer auditorías y manejar la seguridad de datos.
- Facilidad de mantenimiento
- El diseño modular permite aislar problemas, corregir errores y evolucionar cada capa sin afectar todo el sistema.

```
analytics_db
├── bronze
│   ├── sales_raw
│   ├── customers_raw
│   ├── payments_raw
└── silver
    ├── sales_clean
    ├── customers_clean
    └── payments_clean
└── gold
    ├── sales_summary_by_region
    ├── customer_lifetime_value
    └── revenue_forecasting
```

MPP



MPP

¿Qué es una base de datos MPP?

MPP significa Massively Parallel Processing.

Una base de datos MPP es un tipo de sistema que divide un gran volumen de datos y consultas en muchas pequeñas partes para ser procesadas en **paralelo** por varios **nodos** o servidores al **mismo tiempo**.



teradata.

VERTICA



**Google
Big Query**



**GREENPLUM
DATABASE**

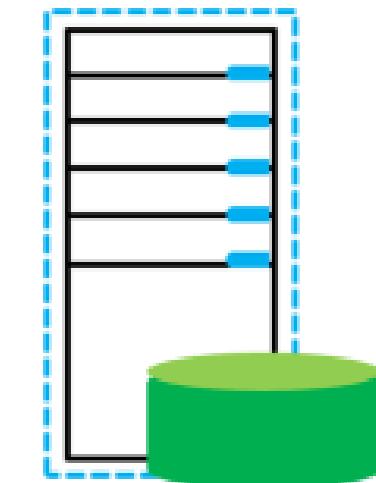


Exasol

¿Cómo funciona internamente un sistema MPP?

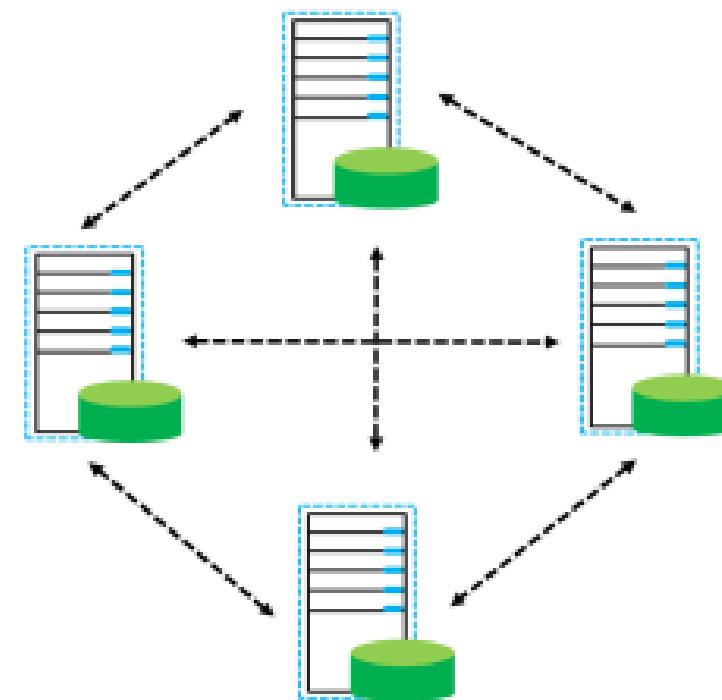
- Los datos se distribuyen entre muchos nodos (máquinas).
- Cada nodo procesa su parte localmente (sin pedir permiso a otros).
- Cuando haces una consulta o cargas datos:
- El trabajo se divide automáticamente entre los nodos.
- Cada nodo trabaja en paralelo (simultáneamente, no uno después del otro).
- Al final, se juntan los resultados y te dan la respuesta como si fuera una sola base.

Symmetric Multiprocessing



SMP vs. MPP

Massively Parallel Processing



**Usa un Data Lake para almacenar datos de cualquier tipo
de forma flexible y barata.**

**Usa una base MPP para consultas analíticas de alta
velocidad sobre datos estructurados.**

¿Cuándo es mejor usar una base MPP?

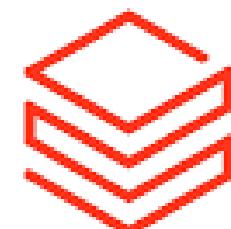
- Cuando necesitas consultas SQL analíticas rápidas sobre datos ya estructurados.
- Cuando quieres hacer reportes de negocio, dashboards y KPIs en tiempo real o casi real.
- Cuando tu prioridad es consultar datos rápido, no solo almacenarlos.
- Cuando tu carga de trabajo requiere procesamiento OLAP a gran escala.

Databricks

Es una plataforma unificada de análisis de datos basada en Apache Spark, que permite procesar grandes volúmenes de datos de forma distribuida, integrar datos estructurados y no estructurados, y desarrollar proyectos de análisis avanzado, machine learning y big data sobre Data Lakes.

Características principales de Databricks

- **Basado en clústeres Apache Spark**
 - Usa clústeres distribuidos que permiten escalar horizontalmente el procesamiento de datos.
- **Integración nativa con Data Lakes**
 - Trabaja directamente sobre almacenamientos como S3, Azure Data Lake, Google Cloud Storage sin necesidad de mover los datos.
- **Soporta múltiples tipos de carga de trabajo**
 - Procesamiento batch (por lotes).
 - Procesamiento streaming (en tiempo real).
 - SQL Analytics (consultas tipo base de datos).
 - Machine Learning (entrenamiento y despliegue de modelos).
 - Data Engineering (pipelines ETL/ELT).
- **MPP?**



databricks

Bibliografía

Oketunji, Finbarrs.

Medallion Architecture with Databricks.

2024.

Serra, James.

Deciphering Data Architectures.

O'Reilly Media, Inc., 2024.

Stewart Buse.

The Data Strategy Handbook.

2022.

Maja Ferle.

Snowflake Data Engineering.

Manning Publications, 2025.

Thompson Carter.

The Big Data Architect: Design Scalable Solutions for Complex Problems.

Lincoln Publishers, 2025.

Otros recursos

<https://databricks.com/glossary/lambda-architecture>

<https://www.confluent.io/learn/batch-vs-real-time-data-processing/>



DATASPHERE

GRACIAS!



Nelson Zepeda

nelson.zepeda@datasphere.tech
Mayo 2025

WWW.DATASPHERE.TECH