# UD2: ARQUITECTURAS Y PATRONES PARA BIG DATA

## Introducción: La Columna Vertebral de los Sistemas de Datos a Gran Escala

En el vasto universo del Big Data, la arquitectura no es simplemente un plano técnico; es la columna vertebral que soporta todo el ecosistema de datos de una organización. La elección de una arquitectura adecuada es, sin lugar a dudas, el factor más crítico que determina el éxito o el fracaso de cualquier iniciativa de datos a gran escala. Una arquitectura bien diseñada puede transformar volúmenes masivos de datos caóticos en una fuente de valor estratégico, mientras que una mal concebida puede convertirse en un sumidero de recursos, complejidad y oportunidades perdidas. Toda arquitectura de datos moderna nace de una tensión fundamental: la necesidad de equilibrar los exigentes requisitos del negocio —baja latencia para decisiones en tiempo real, alta precisión para análisis estratégicos, y una escalabilidad casi infinita— con las ineludibles restricciones del mundo físico y computacional, como el teorema CAP (Consistencia, Disponibilidad, Tolerancia a Particiones), los costos de infraestructura y la complejidad operativa.[1]

Este capítulo servirá como una hoja de ruta a través del paisaje evolutivo de las arquitecturas de datos. Iniciaremos nuestro viaje explorando los patrones arquitecturales fundamentales, **Lambda** y **Kappa**, que representan dos filosofías distintas para resolver el dilema entre la inmediatez y la completitud de los datos. A continuación, realizaremos un recorrido cronológico por la historia de las arquitecturas centralizadas, desde los sistemas transaccionales de los años 70 hasta el moderno paradigma del **Data Lakehouse**, observando cómo cada generación ha construido sobre las lecciones de la anterior. Finalmente, culminaremos con el paradigma más reciente y disruptivo: **Data Mesh**, un enfoque descentralizado que no solo replantea la tecnología, sino también la organización y la cultura en torno a los datos. Este recorrido nos proporcionará una narrativa coherente sobre cómo ha evolucionado el pensamiento en la ingeniería de datos, equipándonos con el conocimiento necesario para diseñar, evaluar y evolucionar los sistemas que impulsarán la próxima generación de aplicaciones basadas en datos.

## 2.1 Patrones Arquitecturales: El Dilema entre Latencia y

# Completitud

En el corazón del diseño de sistemas de Big Data yace un dilema fundamental: ¿cómo podemos satisfacer simultáneamente la necesidad de obtener vistas de datos precisas y completas, un proceso que inherentemente requiere tiempo para procesar grandes volúmenes de historia, y la demanda de vistas inmediatas y en tiempo real, que por naturaleza pueden ser aproximadas o incompletas?.[2] Los patrones arquitecturales como Lambda y Kappa no son meras recetas técnicas; son respuestas filosóficas y estratégicas a esta pregunta central. Representan dos enfoques distintos para gestionar el compromiso ineludible entre la latencia y la completitud, un desafío que toda organización basada en datos debe enfrentar.

## 2.1.1 Arquitectura Lambda: El Enfoque Híbrido para la Robustez Total

La Arquitectura Lambda, concebida por Nathan Marz, surgió como una solución ingeniosa para manejar datos masivos aprovechando lo mejor de dos mundos: el procesamiento por lotes (batch) y el procesamiento en tiempo real (streaming).[2] Su filosofía central es la de "vencer" las limitaciones impuestas por el teorema CAP, no resolviéndolo, sino eludiéndolo mediante la implementación de dos rutas de procesamiento paralelas. Una ruta está optimizada para la completitud, la precisión y la tolerancia a fallos, mientras que la otra está diseñada para una latencia mínima.[3]
El flujo de datos en una arquitectura Lambda es canónico: los datos entrantes se ingieren y se bifurcan para ser enviados simultáneamente a dos capas distintas: la capa batch y la capa de velocidad. Cada capa procesa los datos de forma independiente. Finalmente, los resultados precalculados de ambas capas, conocidos como "vistas", se fusionan en una capa de servicio. Esta capa final es la que responde a las consultas de los usuarios, combinando la vista histórica completa y precisa de la capa batch con la vista más reciente y en tiempo real de la capa de velocidad, ofreciendo así una visión completa y actualizada del estado de los datos.[2]

### Capa Batch (Cold Path): El Repositorio de la Verdad Absoluta

- **Propósito:** El objetivo principal de la capa batch es proporcionar una visión de los datos con una precisión perfecta. Para lograrlo, procesa la totalidad del conjunto de datos maestro, que se mantiene como un registro inmutable y de solo apéndice (*append-only*). Al poder procesar todo el conjunto de datos disponible cada vez que se ejecuta, esta capa puede corregir cualquier error simplemente recalculando las vistas

desde la fuente de verdad completa. El resultado de este proceso son las "vistas batch", que son representaciones precalculadas y exhaustivas de los datos históricos. Su principal compromiso es sacrificar la velocidad (alta latencia) en favor de la máxima precisión.[2]

- **Patrón de Procesamiento:** Esta capa opera exclusivamente bajo un **patrón de procesamiento *batch***. Los trabajos se ejecutan en grandes volúmenes de datos a intervalos programados, que pueden ser cada pocas horas o una vez al día. Este enfoque es ideal para cálculos complejos y análisis profundos que no requieren resultados inmediatos.[8]
- **Tecnologías Clave:** El motor de procesamiento distribuido por excelencia para la capa batch es **Apache Spark**. Su arquitectura *master-worker* y sus abstracciones fundamentales, como los *Resilient Distributed Datasets* (RDDs) y los *Directed Acyclic Graphs* (DAGs), le permiten procesar petabytes de datos de manera eficiente y tolerante a fallos.[11] Estos trabajos de Spark suelen operar sobre datos almacenados en sistemas de archivos distribuidos como HDFS (en el ecosistema Hadoop) o, más comúnmente en la actualidad, en almacenes de objetos en la nube como Amazon S3, Azure Data Lake Storage o Google Cloud Storage.[13]

### Capa de Velocidad (Hot Path): La Vía Rápida hacia el Dato Reciente

- **Propósito:** La capa de velocidad (también llamada capa de streaming) existe para compensar la alta latencia de la capa batch. Su función es proporcionar vistas en tiempo real o casi en tiempo real de los datos más recientes, llenando el "vacío" temporal que se crea entre las ejecuciones de los lotes. Esta capa no busca la completitud ni la precisión absoluta; su objetivo es minimizar la latencia, ofreciendo una visión inmediata de los datos a medida que llegan, aunque esta visión sea incremental y potencialmente aproximada.[2]
- **Patrón de Procesamiento:** Aquí domina el **patrón de procesamiento *streaming***. Los datos se procesan evento a evento o en pequeños micro-lotes tan pronto como se reciben. Esto permite obtener resultados con latencias que van desde milisegundos hasta unos pocos segundos, lo cual es crucial para aplicaciones que requieren una respuesta inmediata.[8]
- **Tecnologías Clave:** El sistema nervioso central de la capa de velocidad es **Apache Kafka**. Actúa como un bus de eventos distribuido, duradero y de alto rendimiento que desacopla a los productores de datos de los consumidores. Su arquitectura, basada en *brokers*, *topics* y *particiones*, garantiza la escalabilidad y la tolerancia a fallos, convirtiéndolo en el estándar de facto para la ingesta de datos en tiempo real.[2] Los datos que fluyen a través de Kafka son consumidos y procesados por motores de streaming como

**Apache Flink**, **Apache Spark Streaming** o Apache Storm.[2]

## Capa de Servicio (Serving Layer): Unificando Vistas para la Consulta

- **Propósito:** Esta capa actúa como el punto de encuentro de los dos mundos. Su función es indexar y almacenar las vistas generadas tanto por la capa batch como por la capa de velocidad, y exponerlas de manera que puedan ser consultadas con muy baja latencia por las aplicaciones finales. Un desafío clave para la capa de servicio es la capacidad de fusionar los resultados de la vista batch (histórica y completa) con los de la vista en tiempo real (reciente e incremental) para presentar una respuesta única y coherente al usuario.[2]
- **Tecnologías:** Para lograr un acceso rápido, esta capa suele emplear bases de datos NoSQL optimizadas para lecturas rápidas, como **Apache Cassandra** o **Apache HBase**, especialmente para almacenar los resultados de la capa de velocidad. Sistemas especializados como **Apache Druid**, **Apache Pinot** o ClickHouse están diseñados específicamente para servir como una capa de servicio unificada, capaces de ingerir datos de ambas rutas y responder a consultas analíticas ad-hoc con latencias de sub-segundo.[1]

## Ventajas y Desventajas

- **Ventajas:** La principal fortaleza de la arquitectura Lambda es su robustez. Es extremadamente tolerante a fallos, ya que el conjunto de datos maestro es inmutable y cualquier vista puede ser recalculada desde cero si es necesario. Proporciona una precisión de datos históricos inigualable y es lo suficientemente versátil como para manejar una amplia gama de casos de uso, desde el reporting tradicional hasta el análisis en tiempo real.[3]
- **Desventajas:** Su mayor debilidad es la complejidad. Mantener dos pipelines de procesamiento distintos es un desafío operativo significativo. Esto conduce a la duplicación de la lógica de negocio, que debe ser implementada y mantenida en dos bases de código diferentes (por ejemplo, una en Spark para el batch y otra en Flink para el streaming), lo que aumenta el riesgo de inconsistencias. Consecuentemente, los costos operativos, de desarrollo y de mantenimiento son considerablemente altos.[2]

## Casos de Uso del Mundo Real

- **Servicios Financieros:** Un caso de uso clásico es la detección de fraude. La capa de

velocidad analiza las transacciones en tiempo real para identificar y bloquear actividades sospechosas de inmediato. Mientras tanto, la capa batch procesa meses o años de datos históricos para entrenar modelos de Machine Learning más complejos que detectan patrones de fraude sutiles y a largo plazo.[4]

- **IoT y Telemática:** En la gestión de flotas de vehículos, la capa de velocidad puede monitorizar datos de sensores en tiempo real para generar alertas sobre fallos inminentes o comportamientos de conducción peligrosos. La capa batch analiza los datos históricos de toda la flota para optimizar rutas, programar el mantenimiento predictivo y analizar la eficiencia del combustible a lo largo del tiempo.[4]
- **Análisis de Redes Sociales:** Las plataformas de redes sociales utilizan la capa de velocidad para identificar y mostrar temas de tendencia en tiempo real (*trending topics*). Simultáneamente, la capa batch realiza análisis profundos sobre el comportamiento de los usuarios, el análisis de sentimiento y la efectividad de las campañas publicitarias a lo largo de semanas o meses.[7]

## 2.1.2 Arquitectura Kappa: La Simplificación Radical hacia el Streaming Puro

A medida que los motores de procesamiento de streaming se volvieron más potentes y maduros, la complejidad inherente de la arquitectura Lambda comenzó a ser vista como una carga innecesaria. En este contexto, Jay Kreps, uno de los co-creadores de Apache Kafka, propuso la Arquitectura Kappa en 2014. Su filosofía central es una simplificación radical: si tu motor de streaming es lo suficientemente rápido y robusto, puedes manejar tanto el procesamiento en tiempo real como el reprocesamiento histórico con un único pipeline de streaming. La capa batch, con su base de código y su infraestructura separadas, se convierte en una redundancia.[5]
El flujo de datos en Kappa es elegantemente lineal. Todos los datos se ingieren como un flujo de eventos en un log inmutable y duradero, que sirve como la única fuente de verdad. Un único motor de procesamiento de streaming consume de este log para generar y actualizar continuamente las vistas analíticas. Estas vistas se almacenan en una capa de servicio para ser consultadas. La magia ocurre cuando es necesario reprocesar datos, por ejemplo, debido a un cambio en la lógica de negocio. En lugar de ejecutar un trabajo batch, simplemente se "rebobina" el consumidor de streaming al inicio del log y se vuelve a procesar todo el historial de eventos a alta velocidad, generando una nueva versión de las vistas.[1]

### Capas (Lógica)

- **Capa de Ingesta y Log:** El corazón indiscutible de la arquitectura Kappa es el log de

eventos. **Apache Kafka** es la tecnología canónica para esta capa. No es solo un sistema de mensajería, sino un sistema de almacenamiento distribuido, un log de *commits* que almacena de forma duradera y ordenada todo el historial de eventos que han ocurrido en el sistema. Esta capacidad de retener datos a largo plazo y permitir su relectura es lo que hace posible la arquitectura Kappa.[17]

- **Capa de Procesamiento en Streaming:** Un único motor de procesamiento es responsable de toda la lógica de negocio. Tecnologías como **Apache Flink**, **Apache Spark Structured Streaming** o **ksqlDB** son las herramientas de elección. Este motor opera en dos modos: en el modo "en tiempo real", consume los eventos más recientes del final del log para mantener las vistas actualizadas con baja latencia. En el modo de "reprocesamiento", consume el log desde un punto anterior en el tiempo (a menudo, desde el principio) para reconstruir las vistas con una nueva lógica, aprovechando el paralelismo para hacerlo lo más rápido posible.[14]
- **Capa de Servicio:** Funcionalmente idéntica a la de la arquitectura Lambda, esta capa almacena las vistas materializadas (los resultados del procesamiento) en bases de datos optimizadas para lecturas rápidas, permitiendo que las aplicaciones de usuario final realicen consultas con baja latencia.[20]

## Ventajas y Desventajas

- **Ventajas:** La simplicidad es el principal beneficio. Al eliminar la capa batch, se reduce a la mitad la complejidad arquitectónica. Hay una sola base de código que mantener, lo que acelera el desarrollo y reduce la probabilidad de errores. Los costos de infraestructura y operativos también disminuyen, ya que solo se necesita gestionar y escalar un único sistema de procesamiento. Esto dota a los equipos de una mayor agilidad para evolucionar su lógica de negocio.[22]
- **Desventajas:** El principal desafío es que el reprocesamiento de grandes volúmenes de datos históricos (terabytes o petabytes) a través de un motor de streaming puede ser computacionalmente intensivo y llevar un tiempo considerable, aunque los motores modernos son cada vez más eficientes en esta tarea. La gestión del estado en aplicaciones de streaming complejas y de larga duración puede ser un reto técnico. Además, no todos los algoritmos de batch tienen un equivalente directo y eficiente en el mundo del streaming, lo que puede hacer que Kappa no sea adecuada para ciertos tipos de análisis muy complejos.[23]

## Casos de Uso y Empresas Reales

- **Casos Típicos:** La arquitectura Kappa brilla en escenarios donde la inmediatez y la

agilidad son primordiales. Esto incluye la detección de anomalías en tiempo real, la personalización de experiencias de usuario, la monitorización de sistemas y redes, y el análisis de *clickstream* para optimizar sitios web y aplicaciones sobre la marcha.[25]

- **Estudio de Caso - Uber:** Uber es un ejemplo paradigmático del uso de una arquitectura inspirada en Kappa a una escala masiva. Gestiona billones de mensajes diarios para alimentar casos de uso críticos como la tarificación dinámica (*surge pricing*), la asignación eficiente de conductores a pasajeros y la detección de fraude en tiempo real. Sin embargo, Uber se encontró con una limitación clave de Kappa pura: almacenar años de datos en Kafka es prohibitivamente caro. Su solución fue evolucionar hacia una arquitectura que denominaron **Kappa+**. En este modelo, los datos históricos se archivan en su data lake (basado en HDFS/Hive), que es mucho más económico. Cuando necesitan reprocesar datos históricos, su motor de streaming (basado en Apache Flink) es capaz de leer directamente desde el data lake, tratándolo como una fuente de streaming acotada, y luego cambiar sin problemas a la fuente de Kafka para el procesamiento en tiempo real. Esto les da la simplicidad de una única base de código (Kappa) con la eficiencia de costos del almacenamiento a largo plazo (data lake).[1]
- **Estudio de Caso - Netflix:** Netflix utiliza Apache Kafka como la columna vertebral de su vasta arquitectura de microservicios. Prácticamente toda la comunicación entre servicios se realiza a través de flujos de eventos. Esto se alinea perfectamente con el paradigma Kappa. Casos de uso como la personalización de recomendaciones, el procesamiento de datos de visualización para análisis de audiencia y los sistemas financieros internos se construyen sobre flujos de eventos procesados en tiempo real, demostrando la viabilidad de un enfoque *stream-first* a escala empresarial.[25]
- **Estudio de Caso - Spotify:** El gigante del streaming de audio utiliza una arquitectura similar para el análisis en tiempo real del comportamiento del usuario. Cada reproducción, salto, o adición a una lista de reproducción es un evento que fluye a través de su plataforma de datos. Estos flujos alimentan los sistemas de recomendación (como "Descubrimiento Semanal"), los paneles de control para artistas y las plataformas de publicidad en tiempo real, todos ellos ejemplos de aplicaciones que requieren la frescura de datos que proporciona un enfoque Kappa.[25]
- **Estudio de Caso - Pinterest:** Pinterest ofrece un caso de estudio revelador sobre la transición de Lambda a Kappa. Su infraestructura de "señales visuales" (datos derivados del análisis de imágenes mediante Machine Learning) se basaba inicialmente en una arquitectura Lambda. El pipeline batch, basado en Hadoop, tardaba 24 horas en procesar las nuevas imágenes, una latencia inaceptable para muchos casos de uso. La complejidad de mantener los pipelines batch y de streaming sincronizados ralentizaba la innovación. Al migrar a una nueva infraestructura inspirada en Kappa y basada en Apache Flink, lograron reducir drásticamente la latencia de disponibilidad de las señales, simplificar la arquitectura y aumentar la velocidad de desarrollo de los equipos, demostrando los beneficios prácticos de esta evolución.[37]

La transición de la arquitectura Lambda a la Kappa no es simplemente una mejora técnica, sino una respuesta directa a la "deuda de complejidad" que Lambda imponía a las organizaciones. Lambda fue una solución brillante y necesaria en una época en la que los motores de streaming no tenían la madurez suficiente para garantizar la misma corrección y rendimiento que los sistemas batch en reprocesamientos a gran escala.[2] Esta complejidad se manifestaba en la necesidad de duplicar la lógica de negocio, mantener dos sistemas distribuidos distintos y sincronizados, y el constante desafío de reconciliar los resultados de ambos pipelines.[3]

La propuesta de Jay Kreps con Kappa partía de la hipótesis de que la tecnología de streaming, especialmente con Kafka como un log inmutable y motores como Flink, había madurado lo suficiente como para que el reprocesamiento desde el stream fuera una alternativa viable y más simple que mantener una capa batch separada.[20] Por lo tanto, esta evolución es un ejemplo clásico de cómo la madurez tecnológica permite simplificar las arquitecturas. La industria ha aceptado, en muchos casos, una compensación: se reduce la complejidad en el desarrollo y mantenimiento diario a cambio de asumir una mayor demanda computacional durante los (relativamente infrecuentes) eventos de reprocesamiento. La elección entre Lambda y Kappa, por tanto, no es solo una cuestión técnica de "batch vs. streaming", sino una decisión estratégica sobre dónde una organización prefiere gestionar la complejidad: en el ciclo de vida del desarrollo (Lambda) o en la infraestructura y el cómputo bajo demanda (Kappa).

| Criterio | Arquitectura Lambda | Arquitectura Kappa |
|---|---|---|
| **Filosofía Central** | Híbrida: combina batch y streaming para obtener lo mejor de ambos mundos (precisión y velocidad). | Unificada: un único pipeline de streaming maneja tanto el procesamiento en tiempo real como el histórico. |
| **Complejidad** | Alta. Requiere gestionar, mantener y sincronizar dos sistemas de procesamiento distintos. | Baja. Un único sistema y una única base de código simplifican el desarrollo y la operación. |
| **Base de Código** | Duplicada. La misma lógica de negocio debe implementarse para el motor batch y el motor de streaming. | Única. Toda la lógica de negocio reside en una sola base de código de procesamiento de streaming. |
| **Latencia (Vista Real)** | Muy baja. Proporcionada por la capa de velocidad (milisegundos a segundos). | Muy baja. El procesamiento en tiempo real es la operación por defecto (milisegundos a segundos). |
| **Latencia (Vista Histórica)** | Alta. Depende de la frecuencia de ejecución de los trabajos batch (horas a días). | Variable. La vista histórica se actualiza en tiempo real, pero el reprocesamiento completo |

| | | |
|---|---|---|
| | | puede llevar horas. |
| **Consistencia de Datos** | Desafiante. Requiere un esfuerzo considerable para asegurar que las vistas batch y de velocidad produzcan resultados consistentes. | Alta. Al haber una única fuente de verdad (el log) y una única lógica, la consistencia es inherente. |
| **Tolerancia a Fallos** | Muy alta. El conjunto de datos maestro inmutable permite la reconstrucción completa de las vistas batch. | Alta. Se basa en la durabilidad del log de eventos y las capacidades de checkpointing del motor de streaming. |
| **Costo Operativo** | Alto. Costos duplicados de infraestructura, mantenimiento y personal especializado para ambos pipelines. | Moderado. Menores costos al operar y escalar un único sistema de procesamiento. |
| **Caso de Uso Ideal** | Sistemas que requieren análisis históricos muy complejos que son difíciles o ineficientes de implementar en streaming, junto con la necesidad de vistas en tiempo real. | Sistemas donde la agilidad y la baja latencia son críticas, y la lógica de negocio puede ser expresada de manera uniforme para datos históricos y en tiempo real. |
| **Tecnologías Típicas** | **Batch:** Spark, Hadoop. **Speed:** Kafka, Flink. **Serving:** Cassandra, Druid. | **Log:** Kafka, Pulsar. **Stream:** Flink, Spark Streaming. **Serving:** NoSQL, bases de datos analíticas. |

## 2.2 Arquitecturas de Datos Centralizadas: Una Evolución Histórica

La historia de las arquitecturas de datos es un fascinante viaje cronológico que refleja la evolución de las necesidades empresariales y las capacidades tecnológicas. Este recorrido nos muestra cómo hemos pasado de sistemas diseñados para registrar transacciones simples a plataformas complejas capaces de impulsar la inteligencia artificial. Para comprender esta evolución, es crucial introducir dos conceptos transversales: los **patrones de ingesta** y los **patrones de procesamiento**.

Los **patrones de ingesta** definen cómo los datos entran en un sistema. En el modelo **Push**, el sistema de origen envía activamente los datos al sistema de destino tan pronto como están

disponibles, ideal para escenarios en tiempo real como la telemetría de IoT.[38] En el modelo **Pull**, el sistema de destino solicita y extrae los datos del sistema de origen a intervalos regulares, un enfoque común en la carga de datos tradicional.[38]

Los **patrones de procesamiento** describen cómo se transforman los datos una vez ingeridos. El procesamiento **Batch** opera sobre grandes volúmenes de datos acumulados durante un período, optimizado para el rendimiento (throughput) pero con alta latencia.[8] El procesamiento **Streaming**, por otro lado, actúa sobre los datos evento a evento o en pequeños micro-lotes a medida que llegan, priorizando la baja latencia sobre el análisis profundo.[8]

La siguiente tabla ofrece un mapa conceptual de esta evolución, que detallaremos en las siguientes secciones.

| Generación | Década | Paradigma Arquitectónico | Tipo de Dato Principal | Objetivo Principal | Patrón de Procesamiento Dominante | Tecnologías Representativas |
|---|---|---|---|---|---|---|
| **Gen 0** | 1970s | Sistemas Transaccionales (OLTP) | Estructurado | Automatización de procesos | Transaccional en tiempo real | Mainframes, IBM IMS/CICS |
| **Gen 1** | 1980s | Data Warehouse (DWH) | Estructurado | Soporte a la decisión (BI) | Batch (ETL) | Teradata, Oracle, DB2 |
| **Gen 2** | 1990s | Almacenes Operacionales (ODS) | Estructurado | Reporting operacional actual | Near Real-Time (Micro-batch) | Bases de datos relacionales |
| **Gen 3** | 2000s | Gestión de Datos Maestros (MDM) | Estructurado | Única fuente de la verdad | Consolidación y Sincronización | Informatica MDM, SAP MDG |
| **Gen 4** | 2010s | Data Lake | Todos (Estructurado, Semi, No) | Análisis avanzado, ML | Batch, Streaming | Hadoop (HDFS, MapReduce), Spark |
| **Gen 5** | 2020s | Data Lakehouse | Todos (Estructurado, Semi, No) | Unificar BI y ML | Batch, Streaming | Delta Lake, Iceberg, Hudi |

## a) Generación 0 (1970s): Sistemas Transaccionales (OLTP)

En la era de los mainframes, el principal objetivo de la informática empresarial era la automatización de los procesos de negocio manuales. Esto dio origen a los Sistemas de Procesamiento de Transacciones en Línea (OLTP, por sus siglas en inglés).[40] Estos sistemas estaban diseñados para gestionar un gran número de transacciones cortas y concurrentes, como reservas de vuelos, transacciones bancarias o registros de ventas.
Las características definitorias de los sistemas OLTP eran su enfoque en datos altamente estructurados y normalizados (para evitar redundancia) y su optimización para operaciones de escritura rápidas y consistentes (INSERT, UPDATE, DELETE), garantizadas por las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad).[41] El análisis de datos era secundario y muy limitado. Sistemas pioneros como
**SABRE**, desarrollado por IBM para American Airlines, y los sistemas **IMS** y **CICS** de IBM, se convirtieron en la columna vertebral de industrias como la aviación y la banca, procesando miles de transacciones por día.[41]

## b) Generación 1 (1980s): Data Warehouse (DWH)

Pronto se hizo evidente que los sistemas OLTP, optimizados para transacciones, eran extremadamente ineficientes para la generación de informes y el análisis de negocio. Las consultas analíticas complejas podían ralentizar o incluso bloquear las operaciones diarias críticas. Esta necesidad de separar las cargas de trabajo operacionales de las analíticas dio lugar al nacimiento del **Data Warehouse (DWH)**, un concepto desarrollado para apoyar los Sistemas de Soporte a la Decisión (DSS).[43]
Un DWH se define como un repositorio central de datos que es **orientado a temas** (p. ej., ventas, clientes), **integrado** (los datos de múltiples fuentes se consolidan en un formato consistente), **histórico** (almacena datos a lo largo del tiempo para análisis de tendencias) y **no volátil** (los datos, una vez escritos, no se modifican).[46] En esta era, surgieron dos filosofías de diseño contrapuestas que aún hoy son relevantes:

- **Bill Inmon (Enfoque Top-Down):** Considerado el "padre del Data Warehouse", Inmon abogaba por un enfoque de arriba hacia abajo. Primero, se debía construir un **Enterprise Data Warehouse (EDW)** centralizado, normalizado (típicamente en Tercera Forma Normal o 3NF) e integrado, que sirviera como la "única fuente de la verdad" para toda la organización. A partir de este repositorio central, se creaban **Data Marts** departamentales, que contenían subconjuntos de datos específicos para las necesidades de cada área de negocio.[47]
- **Ralph Kimball (Enfoque Bottom-Up):** Kimball propuso un enfoque más pragmático y de abajo hacia arriba. Su metodología se centra en construir primero **Data Marts** individuales, orientados a procesos de negocio específicos (p. ej., gestión de pedidos, envíos). Estos Data Marts se diseñan utilizando un **modelo dimensional** (esquema de

estrella o copo de nieve), que optimiza los datos para la consulta y el análisis. Luego, estos Data Marts se integran gradualmente a través de **dimensiones conformadas** (dimensiones compartidas como 'Cliente' o 'Producto') para formar un DWH empresarial coherente.[45]

Independientemente del enfoque, el patrón de procesamiento dominante era puramente **batch**. Los datos se extraían de los sistemas OLTP, se transformaban para limpiar, integrar y modelar, y se cargaban en el DWH mediante procesos **ETL (Extract, Transform, Load)** que se ejecutaban típicamente durante la noche.[44]

## c) Generación 2 (1990s): Almacenes Operacionales (ODS)

El DWH resolvió el problema del análisis estratégico, pero su alta latencia (los datos solían tener un día o más de antigüedad) lo hacía inútil para el reporting operacional del día a día. Los gerentes de operaciones necesitaban una visión integrada y actualizada de lo que estaba sucediendo "ahora" en el negocio.[51]

Para llenar este vacío, surgió el concepto del **Almacén de Datos Operacional (ODS, por sus siglas en inglés)**. Un ODS se puede considerar como un área intermedia entre los sistemas transaccionales de origen y el DWH. Contiene datos integrados, detallados y muy actuales (casi en tiempo real), pero con un historial muy limitado (p. ej., los últimos 30-90 días). Su propósito no es el análisis estratégico a largo plazo, sino el reporting operacional táctico, como ver el estado de los pedidos de la última hora o el inventario actual en todas las tiendas.[52] Arquitectónicamente, el ODS a menudo actúa como una fuente de datos ya integrados para el proceso ETL del DWH, simplificando la carga de datos históricos.[52]

## d) Generación 3 (2000s): Gestión de Datos Maestros (MDM)

La década de 2000 vio una explosión en la adopción de aplicaciones empresariales como los sistemas de Planificación de Recursos Empresariales (ERP) y de Gestión de Relaciones con el Cliente (CRM). Esto creó un nuevo problema: la fragmentación de los datos maestros. Una misma entidad de negocio clave, como un "cliente" o un "producto", podía existir en múltiples sistemas con atributos diferentes e inconsistentes. Esto llevaba a preguntas sin respuesta clara: ¿Cuál es la dirección correcta del cliente? ¿Cuántos productos tenemos realmente?.[55]

La **Gestión de Datos Maestros (MDM, por sus siglas en inglés)** surgió como la disciplina y el conjunto de tecnologías para resolver este caos. El objetivo de MDM es crear y mantener un **"Golden Record"** o una única versión autoritativa y confiable de los datos maestros de la organización.[55] Una arquitectura MDM típica implica consolidar los datos maestros de diversas fuentes, aplicar reglas para limpiarlos, estandarizarlos y eliminar duplicados, y finalmente, sincronizar este registro maestro de vuelta a los sistemas operacionales o servirlo

como una fuente confiable para el análisis.[55]

## e) Generación 4 (2010s): Data Lake

La llegada del Big Data en la década de 2010 rompió los moldes de las arquitecturas existentes. El volumen, la velocidad y, sobre todo, la **variedad** de los nuevos datos (logs de servidores, datos de clics, redes sociales, sensores de IoT, texto no estructurado, imágenes) hacían que el DWH, con su estructura rígida y su modelo de *schema-on-write* (el esquema debe definirse antes de cargar los datos), fuera inadecuado y prohibitivamente caro.[59]
En este contexto, James Dixon acuñó el término **Data Lake**. La idea era crear un repositorio centralizado capaz de almacenar cantidades masivas de datos en su formato nativo, sin procesar. A diferencia del DWH, el Data Lake adopta un enfoque de *schema-on-read*: los datos se cargan tal cual y la estructura se aplica en el momento de la consulta. Esto proporciona una flexibilidad sin precedentes para "almacenarlo todo ahora" y descubrir su valor más tarde, especialmente para casos de uso exploratorios y de Machine Learning.[61]
La tecnología que hizo posible los primeros Data Lakes fue el ecosistema **Apache Hadoop**. **HDFS (Hadoop Distributed File System)** proporcionó un almacenamiento distribuido, escalable y de bajo costo sobre hardware genérico, mientras que **MapReduce** ofreció un modelo de programación para el procesamiento batch distribuido a gran escala.[59] Poco después,
**Apache Spark** emergió y rápidamente suplantó a MapReduce como el motor de procesamiento preferido, gracias a su rendimiento superior (al procesar en memoria) y su API más versátil, que soportaba no solo batch, sino también streaming, SQL y Machine Learning en un único framework.[59]

## f) Generación 5 (2020s): Data Lakehouse

A pesar de su flexibilidad, los Data Lakes a menudo sufrían de un problema grave: sin una gobernanza y gestión de calidad estrictas, podían degenerar en **"data swamps"** (pantanos de datos), vertederos de datos de baja calidad, indocumentados e inutilizables. Además, carecían de características cruciales de los DWH, como las transacciones ACID, la consistencia de los datos y un rendimiento de consulta optimizado para BI. Esto obligó a muchas organizaciones a mantener una arquitectura de dos niveles: un Data Lake para la ciencia de datos y el ML, y un DWH separado para el BI y el reporting, con complejos pipelines ETL moviendo datos entre ambos. Esta duplicación era costosa, compleja y generaba problemas de obsolescencia de datos.[64]
La arquitectura **Data Lakehouse** surge en la década de 2020 como una solución a este dilema. Su objetivo es combinar lo mejor de ambos mundos: la flexibilidad, escalabilidad y

bajo costo de un Data Lake con la fiabilidad, el rendimiento y las características de gestión de datos de un Data Warehouse.[64] Esto se logra mediante la implementación de una capa de metadatos transaccional directamente sobre el almacenamiento de objetos de bajo costo (como Amazon S3). Tecnologías como
**Delta Lake**, **Apache Iceberg** y **Apache Hudi** son los habilitadores clave de este paradigma. Añaden funcionalidades de DWH, como transacciones ACID, versionado de datos (*time travel*), y cumplimiento de esquemas, directamente a los archivos de datos abiertos (como Parquet) en el lago. Esto permite que un único sistema sirva tanto para las cargas de trabajo de BI como para las de IA/ML, eliminando la necesidad de la arquitectura de dos niveles.[64]
La historia de las arquitecturas de datos centralizadas puede entenderse no como una progresión lineal, sino como el movimiento de un péndulo que oscila entre dos necesidades fundamentales y a menudo contrapuestas: la **estructura** y la **flexibilidad**. Los sistemas OLTP y los Data Warehouses de las primeras generaciones representan el apogeo de la estructura. Sus esquemas predefinidos (*schema-on-write*) garantizaban la integridad, la calidad y el rendimiento para casos de uso bien conocidos, pero a costa de una gran rigidez e incapacidad para manejar nuevos tipos de datos.[41]
El Data Lake, en cambio, fue la oscilación radical del péndulo hacia la flexibilidad total. Su modelo de *schema-on-read* permitió la ingesta de cualquier tipo de dato sin planificación previa, abriendo la puerta a nuevos horizontes analíticos como el Machine Learning y la exploración de datos. Sin embargo, esta libertad sin restricciones a menudo condujo al caos de los "data swamps".[61]
El Data Lakehouse es el intento más reciente y sofisticado de detener el péndulo en un punto de equilibrio. Busca una síntesis: mantener la flexibilidad, la apertura y el bajo costo del almacenamiento del Data Lake, pero superponiendo una capa de estructura, gobernanza y fiabilidad inspirada en el Data Warehouse. Es la búsqueda de una "flexibilidad gobernada", el santo grial de la arquitectura de datos moderna.[64]

# 2.3 Arquitecturas de Datos Orientada por Dominios: El Paradigma Data Mesh

## 2.3.1 El Concepto: De la Centralización a la Descentralización Socio-Técnica

Incluso con la llegada del Data Lakehouse, que resuelve muchas de las limitaciones técnicas de las arquitecturas anteriores, persiste un problema fundamental que es de naturaleza organizacional. Un equipo de datos centralizado, por muy talentoso que sea, se convierte

inevitablemente en un cuello de botella a medida que la organización crece y la demanda de datos se diversifica. Este equipo central carece del contexto de negocio detallado y no puede escalar para satisfacer las necesidades ágiles y específicas de todos los departamentos de la empresa.[68]

Para abordar esta falla de escala organizacional, Zhamak Dehghani propuso un nuevo paradigma llamado **Data Mesh**. Es crucial entender que Data Mesh no es simplemente un nuevo patrón tecnológico, sino un **paradigma socio-técnico**. Aborda las limitaciones de los sistemas monolíticos y centralizados a través de la descentralización, no solo de la tecnología, sino también de la propiedad y la responsabilidad de los datos.[69]

## 2.3.2 Principio 1: Propiedad y Arquitectura Orientada al Dominio

Este es el principio fundamental de Data Mesh. La responsabilidad sobre los datos analíticos se transfiere desde el equipo central de datos a los **equipos de dominio de negocio**. Un dominio es una unidad lógica de la empresa alineada con una función de negocio específica (p. ej., "Marketing", "Logística", "Pagos"). La premisa es que el equipo que está más cerca de los datos, que los genera y entiende su contexto y significado, es el más adecuado para poseerlos y gestionarlos.[68]

Arquitectónicamente, esto implica descomponer el monolito de datos (sea un DWH o un Data Lake) en una red de nodos de datos distribuidos y autónomos, cada uno alineado y gestionado por un dominio de negocio. El equipo de "Logística", por ejemplo, se hace responsable de extremo a extremo de los datos analíticos relacionados con los envíos y el inventario.[71]

## 2.3.3 Principio 2: El Dato como Producto (Data as a Product)

La descentralización por sí sola podría conducir al caos y a la creación de nuevos silos. Para evitarlo, el segundo principio establece que los dominios no solo "poseen" sus datos, sino que tienen la responsabilidad de servirlos como un **producto** de alta calidad al resto de la organización (sus "clientes" o consumidores de datos). Este es un cambio de mentalidad profundo: los datos dejan de ser un subproducto residual de los procesos operativos y se convierten en un activo de primera clase, diseñado, construido y mantenido con el consumidor en mente.[71]

Para que un conjunto de datos sea considerado un "producto de datos", debe cumplir con una serie de características, a menudo resumidas con el acrónimo **DATSI**:

- **Descubrible (Discoverable):** Los consumidores deben poder encontrar fácilmente los productos de datos disponibles, típicamente a través de un catálogo de datos centralizado.

- **Direccionable (Addressable):** Cada producto de datos debe tener una ubicación única y permanente donde se pueda acceder a él.
- **Confiable (Trustworthy):** El dominio propietario debe garantizar la calidad de los datos y definir Acuerdos de Nivel de Servicio (SLAs) claros sobre su frescura, completitud y precisión.
- **Autodescriptivo (Self-describing):** La semántica y la estructura de los datos deben estar bien documentadas y ser fáciles de entender para que los consumidores puedan usarlos sin necesidad de consultar constantemente al equipo propietario.
- **Interoperable (Interoperable):** Los productos de datos deben adherirse a estándares globales (definidos por la gobernanza federada) para que puedan ser fácilmente combinados y utilizados con otros productos de datos en la malla.
- **Seguro (Secure):** El acceso a los datos debe estar gobernado por políticas claras y aplicadas de forma programática.

Un ejemplo concreto podría ser un "Producto de Datos de Sesiones de Usuario" ofrecido por el dominio de "Experiencia de Producto". Este producto podría consistir en un conjunto de tablas en un Data Lakehouse, accesibles a través de SQL, que proporcionan datos limpios, documentados, con SLAs definidos y con políticas de acceso claras sobre la actividad de los usuarios en la aplicación.[76]

### 2.3.4 Principio 3: Plataforma de Datos Self-Service

Exigir que cada equipo de dominio (compuesto por expertos en marketing o logística, no necesariamente en ingeniería de datos) construya y mantenga su propia infraestructura de datos desde cero sería inviable y conduciría a una duplicación masiva de esfuerzos. El tercer principio aborda este problema proponiendo la creación de una **plataforma de datos de autoservicio (self-service)**.[71]

Un equipo de plataforma central es responsable de construir y mantener esta plataforma, que ofrece la infraestructura de datos como un servicio. Su objetivo es reducir la carga cognitiva de los equipos de dominio, proporcionándoles herramientas estandarizadas y automatizadas para construir, desplegar, monitorizar y gestionar sus productos de datos de forma autónoma. Esta plataforma debe ofrecer capacidades como el almacenamiento políglota, motores de procesamiento de pipelines, herramientas de gobernanza, control de acceso y un catálogo de datos.[72]

Una herramienta clave en esta plataforma es un orquestador de flujos de trabajo como **Apache Airflow**. Airflow permite a los equipos de dominio definir sus pipelines de datos como código (a través de DAGs - Directed Acyclic Graphs), programarlos, ejecutarlos y monitorizarlos de forma independiente. Esto promueve una cultura de "infraestructura como código" y autonomía, que es fundamental para el éxito de Data Mesh.[79]

### 2.3.5 Principio 4: Gobierno Computacional Federado

El último principio resuelve la tensión entre la autonomía de los dominios y la necesidad de interoperabilidad y estándares globales. Un modelo de gobierno puramente centralizado recrearía el cuello de botella que Data Mesh intenta eliminar, mientras que una ausencia total de gobierno llevaría al caos. La solución es un **gobierno computacional federado**.[71]

- **Federado:** Se crea un equipo de gobierno compuesto por representantes de cada dominio de datos y del equipo de la plataforma. Este grupo colabora para definir las "reglas del juego" globales que todos los productos de datos deben seguir. Estas reglas cubren áreas como la seguridad, la privacidad, la interoperabilidad (p. ej., formatos de datos estándar, convenciones de nomenclatura) y los estándares de calidad.[71]
- **Computacional:** La clave de este principio es que estas reglas no son simplemente documentos estáticos en un wiki. Se implementan y automatizan como parte de la plataforma de autoservicio. Por ejemplo, una política de enmascaramiento de datos personales se implementa como un servicio en la plataforma que los equipos de dominio pueden aplicar a sus pipelines. El cumplimiento de las políticas se verifica y se hace cumplir automáticamente, reduciendo la carga manual y garantizando la coherencia en toda la malla.[71]

Los principios de Data Mesh pueden parecer revolucionarios, pero en realidad representan la aplicación de paradigmas que han demostrado ser exitosos en el mundo del desarrollo de software durante la última década, particularmente los microservicios y DevOps. El desarrollo de software evolucionó de grandes aplicaciones monolíticas a arquitecturas de microservicios para mejorar la escalabilidad, la agilidad y la autonomía de los equipos. El **Principio 1 (Propiedad de Dominio)** es una analogía directa, descomponiendo el monolito de datos en "microservicios de datos".[81]

Los microservicios se comunican a través de APIs bien definidas y versionadas, que son tratadas como "productos" para otros servicios. El **Principio 2 (Dato como Producto)** aplica esta misma filosofía a los datos, exponiéndolos a través de "puertos de datos" con contratos claros y SLAs.[80] La cultura DevOps se basa en la idea de que los equipos de desarrollo deben ser capaces de desplegar y operar su propio software de forma autónoma, lo cual es posible gracias a plataformas de infraestructura como servicio (IaaS) y tecnologías como Kubernetes. El **Principio 3 (Plataforma Self-Service)** es la encarnación de este concepto para el mundo de los datos.

Finalmente, en un ecosistema de microservicios, no existe un control centralizado absoluto, sino un conjunto de estándares y prácticas compartidas (gobernanza federada) para garantizar la interoperabilidad. El **Principio 4 (Gobierno Federado)** aplica este mismo modelo de gobernanza al ecosistema de datos. Por lo tanto, Data Mesh no es solo una "arquitectura

de datos"; es un cambio cultural profundo que busca alinear la organización de los equipos de datos con la forma en que las organizaciones de software más exitosas han logrado escalar. Su éxito depende tanto de la adopción de la tecnología adecuada como de una reestructuración organizativa deliberada.

# Conclusión: Hacia un Futuro de Arquitecturas Híbridas y Descentralizadas

A lo largo de este capítulo, hemos viajado desde los patrones fundamentales que equilibran velocidad y precisión, a través de la evolución histórica de los sistemas centralizados, hasta llegar al paradigma descentralizado que redefine la organización del trabajo con datos. Hemos diseccionado tres grandes enfoques: los patrones **Lambda/Kappa**, la arquitectura **Data Lakehouse** y el paradigma **Data Mesh**. Es fundamental entender que estos no son competidores que se excluyen mutuamente, sino herramientas complementarias en el arsenal de un arquitecto de datos moderno.

El futuro de la arquitectura de datos no reside en la elección de un único modelo "ganador", sino en la coexistencia y convergencia inteligente de estos paradigmas. Es perfectamente plausible, y de hecho cada vez más común, ver organizaciones que implementan un **Data Mesh** a nivel estratégico para descentralizar la propiedad de los datos. Dentro de esta malla, cada "producto de datos" ofrecido por un dominio podría ser, a nivel técnico, un **Data Lakehouse** que proporciona lo mejor de ambos mundos: flexibilidad y gobernanza. Y dentro de ese Lakehouse, los pipelines para generar vistas en tiempo real podrían estar implementados siguiendo un patrón **Kappa**, utilizando Kafka y Flink para procesar flujos de eventos con baja latencia.

En última instancia, la lección más importante es que no existe una "bala de plata". La mejor arquitectura es aquella que resuelve de manera efectiva los problemas de negocio específicos de una organización, en un momento dado de su madurez. La elección correcta siempre será un acto de equilibrio, sopesando cuidadosamente las capacidades tecnológicas disponibles con la estructura, la cultura y la estrategia de la organización. El rol del arquitecto de datos es, por tanto, no solo ser un experto en tecnología, sino también un estratega que comprende profundamente el negocio para construir sistemas que no solo funcionen, sino que generen un valor transformador.

### Obras citadas


1. Big Data Architectures — Lamdba & Kappa | by Atin Singh - Medium, fecha de acceso: septiembre 25, 2025, https://medium.com/@atinaks25/big-data-architectures-lamdba-kappa-f5e72f8efeec
2. Lambda architecture - Wikipedia, fecha de acceso: septiembre 25, 2025,

https://en.wikipedia.org/wiki/Lambda_architecture
3. What is Lambda architecture | System Design - GeeksforGeeks, fecha de acceso: septiembre 25, 2025, https://www.geeksforgeeks.org/system-design/what-is-lambda-architecture-system-design/
4. Lambda Architecture 101—Batch, Speed & Serving Layers - Chaos Genius, fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/lambda-architecture/
5. Data Processing Architectures: Lambda vs Kappa | Firas Esbai, fecha de acceso: septiembre 25, 2025, https://www.firasesbai.com/articles/2023/09/24/data-processing-architectures-lambda-vs-kappa.html
6. The Basics of Lambda Architecture for Big Data, fecha de acceso: septiembre 25, 2025, https://www.snowflake.com/en/fundamentals/lambda-architecture/
7. What Is Lambda Architecture? | Coursera, fecha de acceso: septiembre 25, 2025, https://www.coursera.org/articles/lambda-architecture
8. Batch Processing vs Stream Processing: Key Differences for 2025, fecha de acceso: septiembre 25, 2025, https://atlan.com/batch-processing-vs-stream-processing/
9. Batch vs Stream Processing: How to Choose - Prophecy, fecha de acceso: septiembre 25, 2025, https://www.prophecy.io/blog/batch-vs-stream-processing-differences
10. Difference between Batch Processing and Stream Processing - GeeksforGeeks, fecha de acceso: septiembre 25, 2025, https://www.geeksforgeeks.org/operating-systems/difference-between-batch-processing-and-stream-processing/
11. www.chaosgenius.io, fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/apache-spark-architecture/#:~:text=Apache%20Spark%20operates%20on%20a,Directed%20Acyclic%20Graphs%20(DAGs)
12. Apache Spark Architecture 101: How Spark Works (2025), fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/apache-spark-architecture/
13. Build a big data Lambda architecture for batch and real-time analytics using Amazon Redshift - AWS, fecha de acceso: septiembre 25, 2025, https://aws.amazon.com/blogs/big-data/build-a-big-data-lambda-architecture-for-batch-and-real-time-analytics-using-amazon-redshift/
14. Exploring Kappa Architecture with Pathway, fecha de acceso: septiembre 25, 2025, https://pathway.com/blog/exploring-kappa-architecture-with-pathway/
15. Batch Processing vs. Stream Processing: A Comprehensive Guide - Rivery, fecha de acceso: septiembre 25, 2025, https://rivery.io/blog/batch-vs-stream-processing-pros-and-cons-2/
16. Apache Kafka Architecture Deep Dive - Confluent Developer, fecha de acceso: septiembre 25, 2025, https://developer.confluent.io/courses/architecture/get-started/

17. Kafka Architecture - GeeksforGeeks, fecha de acceso: septiembre 25, 2025, https://www.geeksforgeeks.org/apache-kafka/kafka-architecture/
18. What is Lambda Architecture - GigaSpaces, fecha de acceso: septiembre 25, 2025, https://www.gigaspaces.com/data-terms/lambda-architecture
19. Revisiting the Lambda Architecture: Challenges and Alternatives | by CortexFlow | The Software Frontier | Medium, fecha de acceso: septiembre 25, 2025, https://medium.com/the-software-frontier/revisiting-the-lambda-architecture-challenges-and-alternatives-1da4fd5ce08f
20. Kappa Architecture - Where Every Thing Is A Stream, fecha de acceso: septiembre 25, 2025, https://milinda.pathirage.org/kappa-architecture.com/
21. Kappa Architecture - Hazelcast, fecha de acceso: septiembre 25, 2025, https://hazelcast.com/foundations/software-architecture/kappa-architecture/
22. Kappa Architecture - A big data engineering approach | Pradeep Loganathan's Blog, fecha de acceso: septiembre 25, 2025, https://pradeepl.com/blog/kappa-architecture/
23. Kappa Architecture: Stream Processing in Big Data Analytics | by Pratik Barjatiya - Medium, fecha de acceso: septiembre 25, 2025, https://medium.com/data-and-beyond/kappa-architecture-stream-processing-in-big-data-analytics-e539f4bf63cd
24. Kappa Architecture Design: Real-Time Analytics in Open Source Environments | by Giulio Talarico | Data Reply IT | DataTech | Medium, fecha de acceso: septiembre 25, 2025, https://medium.com/data-reply-it-datatech/kappa-architecture-design-6e5edc1ca5cd
25. Kappa Architecture - System Design - GeeksforGeeks, fecha de acceso: septiembre 25, 2025, https://www.geeksforgeeks.org/system-design/kappa-architecture-system-design/
26. Advantages of Kappa architecture in the Modern Data Stack – SQLServerCentral, fecha de acceso: septiembre 25, 2025, https://www.sqlservercentral.com/articles/advantages-of-kappa-architecture-in-the-modern-data-stack
27. Kappa Architecture 101—Deep Dive into Stream-First Design - Chaos Genius, fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/kappa-architecture/
28. Kappa vs Lambda Architecture: A Complete Comparison (2025) - Chaos Genius, fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/kappa-vs-lambda-architecture/
29. Lambda vs. Kappa Architecture. A Guide to Choosing the Right Data Processing Architecture for Your Needs - nexocode, fecha de acceso: septiembre 25, 2025, https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/
30. www.chaosgenius.io, fecha de acceso: septiembre 25, 2025, https://www.chaosgenius.io/blog/kappa-architecture/#:~:text=Common%20use%20cases%20of%20Kappa,%2Fevent%20streaming%E2%80%94and%20more.

31. Designing a Production-Ready Kappa Architecture for Timely Data ..., fecha de acceso: septiembre 25, 2025, https://www.uber.com/blog/kappa-architecture-data-stream-processing/
32. Moving from Lambda and Kappa Architectures to Kappa+ at Uber - Roshan Naik - YouTube, fecha de acceso: septiembre 25, 2025, https://www.youtube.com/watch?v=ExU7fJFw4Bg
33. Flink Forward San Francisco 2019: Moving from Lambda and Kappa ..., fecha de acceso: septiembre 25, 2025, https://www.slideshare.net/slideshow/flink-forward-san-francisco-2019-moving-from-lambda-and-kappa-architectures-to-kappa-at-uber-roshan-naik/140588812
34. How Netflix Uses Kafka for Distributed Streaming - Confluent, fecha de acceso: septiembre 25, 2025, https://www.confluent.io/blog/how-kafka-is-used-by-netflix/
35. Discuss Spotify system architecture. - Design Gurus, fecha de acceso: septiembre 25, 2025, https://www.designgurus.io/answers/detail/discuss-spotify-system-architecture
36. How Spotify built the tech behind streaming millions of songs - Okoone, fecha de acceso: septiembre 25, 2025, https://www.okoone.com/spark/technology-innovation/how-spotify-built-the-tech-behind-streaming-millions-of-songs/
37. Pinterest Visual Signals Infrastructure: Evolution from Lambda to ..., fecha de acceso: septiembre 25, 2025, https://medium.com/pinterest-engineering/pinterest-visual-signals-infrastructure-evolution-from-lambda-to-kappa-architecture-f8f58b127d98
38. Push vs. Pull - Data Engineering Works - GitHub Pages, fecha de acceso: septiembre 25, 2025, https://karlchris.github.io/data-engineering/data-ingestion/push-pull/
39. Sistema Push vs. Sistema Pull: La adopción de un enfoque híbrido para MRP | Tulip, fecha de acceso: septiembre 25, 2025, https://tulip.co/es/blog/what-is-a-push-system-vs-a-pull-system/
40. A Brief History of Decision Support Systems - DSSResources.COM, fecha de acceso: septiembre 25, 2025, https://dssresources.com/history/dsshistory.html
41. Transaction processing system - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Transaction_processing_system
42. 1970 | Timeline of Computer History, fecha de acceso: septiembre 25, 2025, https://www.computerhistory.org/timeline/1970/
43. What Is a Data Warehouse? - IBM, fecha de acceso: septiembre 25, 2025, https://www.ibm.com/think/topics/data-warehouse
44. The Evolution of the Data Warehouse - Panoply, fecha de acceso: septiembre 25, 2025, https://learn.panoply.io/hubfs/eBook-The-Evolution-of-the-Data-Warehouse.pdf
45. A Short History of Data Warehousing - DATAVERSITY, fecha de acceso: septiembre 25, 2025, https://www.dataversity.net/a-short-history-of-data-warehousing/

46. Bill Inmon, the recognized father of the data warehousing concept, defines a data warehouse as a subject-orientated, integrated, time variant, non-volatile collection of data in support of management's decision-making process. Another data warehousing pio | Society of American Archivists, fecha de acceso: septiembre 25, 2025, https://www2.archivists.org/glossary/citation/bill-inmon-the-recognized-father-of-the-data-warehousing-concept-defines-a-data-wa
47. Bill Inmon - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Bill_Inmon
48. Inmon Data Model - Rajanand, fecha de acceso: septiembre 25, 2025, https://rajanand.org/data-modeling/inmon-data-model
49. Data warehouse - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Data_warehouse
50. Kimball Techniques - Kimball Group, fecha de acceso: septiembre 25, 2025, https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/
51. What is Operational Data Store (ODS): Guide You Can't Miss - Airbyte, fecha de acceso: septiembre 25, 2025, https://airbyte.com/data-engineering-resources/operational-data-stores
52. What Is an Operational Data Store (ODS)? Complete Guide, fecha de acceso: septiembre 25, 2025, https://www.snowflake.com/en/fundamentals/a-modern-approach-to-the-operational-data-store/
53. What is an operational data store? | Stitch, fecha de acceso: septiembre 25, 2025, https://www.stitchdata.com/resources/operational-data-store/
54. Operational data store - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Operational_data_store
55. What is Master Data Management (MDM)? | Informatica, fecha de acceso: septiembre 25, 2025, https://www.informatica.com/resources/articles/what-is-master-data-management.html
56. The past, present and future state of Master Data Management | CluedIn, fecha de acceso: septiembre 25, 2025, https://www.cluedin.com/resources/articles/the-past-present-and-future-state-of-master-data-management
57. (PDF) DESIGN OF MASTER DATA ARCHITECTURE - ResearchGate, fecha de acceso: septiembre 25, 2025, https://www.researchgate.net/publication/374339198_DESIGN_OF_MASTER_DATA_ARCHITECTURE
58. Master Data Management Frameworks Explained - Semarchy, fecha de acceso: septiembre 25, 2025, https://semarchy.com/blog/master-data-management-frameworks/
59. History and evolution of data lakes | Databricks, fecha de acceso: septiembre 25, 2025, https://www.databricks.com/discover/data-lakes/history
60. The Evolution of Data Lake Architectures - TDWI, fecha de acceso: septiembre 25, 2025, https://tdwi.org/articles/2020/05/29/arch-all-evolution-of-data-lake-

architectures.aspx
61. Data Lake Explained: Architecture and Examples - AltexSoft, fecha de acceso: septiembre 25, 2025, https://www.altexsoft.com/blog/data-lake-architecture/
62. Data lake - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Data_lake
63. Apache Hadoop - Wikipedia, fecha de acceso: septiembre 25, 2025, https://en.wikipedia.org/wiki/Apache_Hadoop
64. What is a Data Lakehouse? | Databricks, fecha de acceso: septiembre 25, 2025, https://www.databricks.com/glossary/data-lakehouse
65. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics - CIDR, fecha de acceso: septiembre 25, 2025, https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf
66. What Is a Data Lakehouse? | IBM, fecha de acceso: septiembre 25, 2025, https://www.ibm.com/think/topics/data-lakehouse
67. Data Lakehouse Architecture - Databricks, fecha de acceso: septiembre 25, 2025, https://www.databricks.com/product/data-lakehouse
68. The 4 principles of data mesh | dbt Labs, fecha de acceso: septiembre 25, 2025, https://www.getdbt.com/blog/the-four-principles-of-data-mesh
69. Data Mesh Paradigm shift: Reshaping the data infrastructure - devopsdays Seattle 2020, fecha de acceso: septiembre 25, 2025, https://devopsdays.org/events/2020-seattle/program/zhamak-dehghani/
70. The four principles of Data Mesh - Thoughtworks, fecha de acceso: septiembre 25, 2025, https://www.thoughtworks.com/en-us/about-us/events/webinars/core-principles-of-data-mesh
71. Data Mesh Principles and Logical Architecture - Martin Fowler, fecha de acceso: septiembre 25, 2025, https://martinfowler.com/articles/data-mesh-principles.html
72. Data Mesh Principles (Four Pillars) Guide for 2025 - Atlan, fecha de acceso: septiembre 25, 2025, https://atlan.com/data-mesh-principles/
73. The Four Data Mesh Principles: Benefits, Challenges, and Use Cases, fecha de acceso: septiembre 25, 2025, https://hevoacademy.com/data-analytics-resources/data-mesh-principles/
74. www.ibm.com, fecha de acceso: septiembre 25, 2025, https://www.ibm.com/think/topics/data-as-a-product#:~:text=Data%20as%20a%20product%20(DaaP)%20is%20an%20approach%20in%20data,quality%2C%20usability%20and%20user%20satisfaction.
75. What Is Data as a Product (DaaP)? - IBM, fecha de acceso: septiembre 25, 2025, https://www.ibm.com/think/topics/data-as-a-product
76. What are data products? | SAP, fecha de acceso: septiembre 25, 2025, https://www.sap.com/resources/what-are-data-products
77. Practical Guide to Data Products - K2view, fecha de acceso: septiembre 25, 2025, https://www.k2view.com/what-is-a-data-product/
78. What is a Data Mesh? - Data Mesh Architecture Explained - AWS - Updated 2025,

fecha de acceso: septiembre 25, 2025, https://aws.amazon.com/what-is/data-mesh/
79. Architecture Overview — Airflow 3.0.6 Documentation - Apache Airflow, fecha de acceso: septiembre 25, 2025, https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html
80. Understanding Data Mesh, principles and implementation - Knowi, fecha de acceso: septiembre 25, 2025, https://www.knowi.com/blog/data-mesh/
81. What Is a Data Mesh? - IBM, fecha de acceso: septiembre 25, 2025, https://www.ibm.com/think/topics/data-mesh