Reis - Fundamentals of Data Engineering. Chapter 2

Joe Reis Matt Housley

November 24, 2025

Capítulo 2: El Ciclo de Vida de la Ingeniería de Datos

Este capítulo presenta el tema central del libro: el **Ciclo de Vida de la Ingeniería de Datos**. El objetivo es ir más allá de las tecnologías específicas (que cambian rápido) y enfocarse en los principios de gestión del ciclo de vida de los datos ("de la cuna a la tumba").

¿Qué es el Ciclo de Vida de la Ingeniería de Datos?

Es el proceso que convierte ingredientes de datos crudos en un producto final útil para analistas, científicos de datos e ingenieros de ML.

Se divide en **cinco etapas principales**:

1. **Generación**
2. **Almacenamiento (Storage)**
3. **Ingesta (Ingestion)**
4. **Transformación**
5. **Servicio (Serving)**

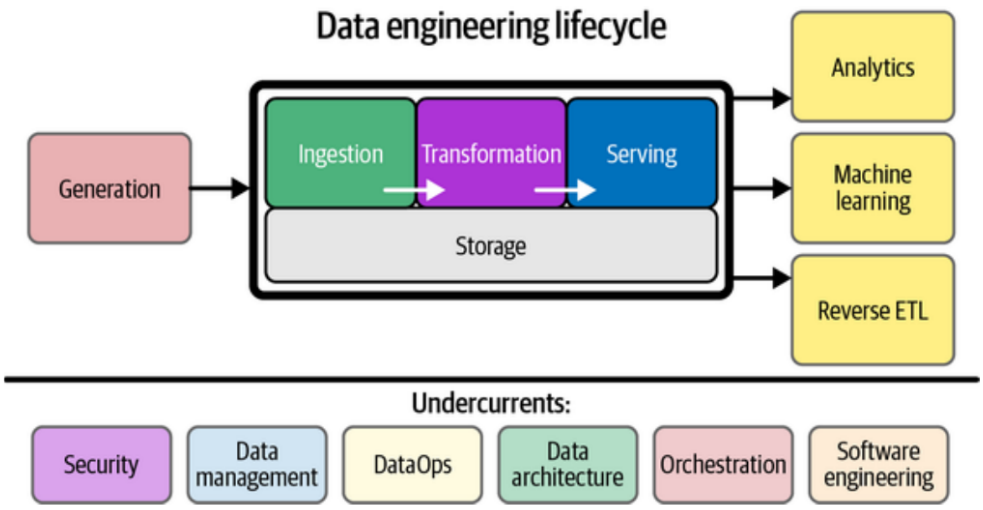

Figura 2-1: Componentes y corrientes subyacentes del ciclo de vida de la ingeniería de datos

Aunque se presentan como etapas distintas, en la práctica pueden superponerse u ocurrir fuera de orden. El diagrama muestra el **Almacenamiento** como una base que sustenta a las demás etapas, ya que los datos se almacenan a lo largo de todo el ciclo.

Además, existen las **corrientes subyacentes (undercurrents)**, que son ideas críticas que atraviesan todas las etapas:

- Seguridad
- Gestión de datos
- DataOps
- Arquitectura de datos
- Orquestación
- Ingeniería de software

El Ciclo de Vida de los Datos vs. El Ciclo de Vida de la Ingeniería de Datos

Hay una distinción sutil. El ciclo de vida de los datos abarca toda la vida útil del dato. El ciclo de vida de la ingeniería de datos es un **subconjunto** que se enfoca en las etapas

que el ingeniero de datos controla directamente.

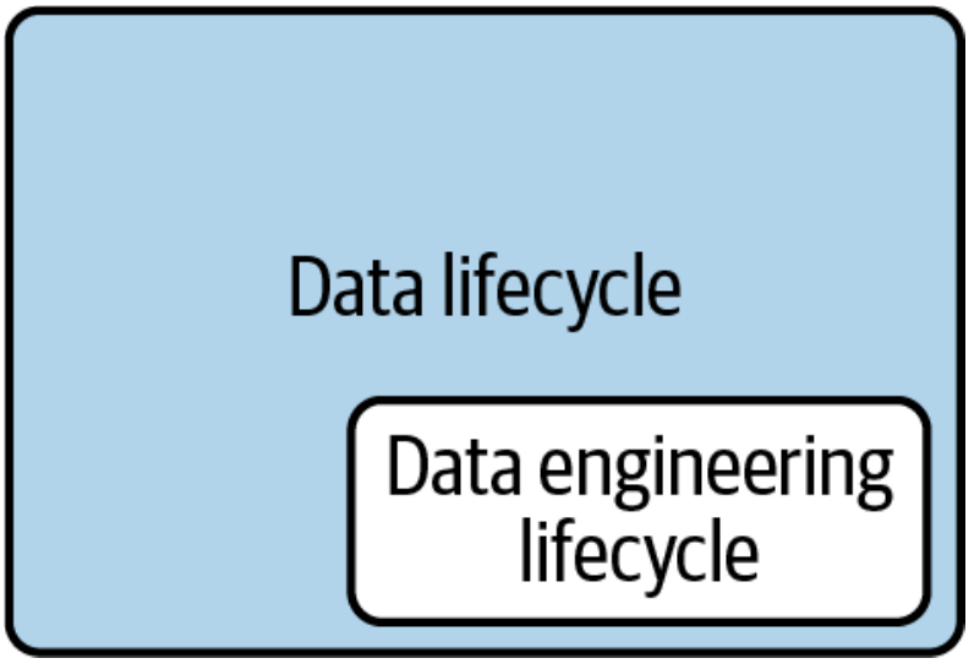

Figura 2-2: El ciclo de vida de la ingeniería de datos es un subconjunto del ciclo de vida completo de los datos

---

1. Generación: Sistemas Fuente

Un **sistema fuente** es el origen de los datos (ej. dispositivo IoT, cola de mensajes, base de datos transaccional). El ingeniero de datos consume datos de aquí pero típicamente no es dueño del sistema.

Es crucial entender cómo funciona la fuente: frecuencia, velocidad y variedad de los datos. Los ingenieros deben comunicarse con los dueños de las fuentes para evitar que cambios en el código "rompan" los pipelines.

Ejemplos de sistemas fuente:

1. **Base de datos de aplicación:** Servidores conectados a una base de datos relacional (RDBMS).

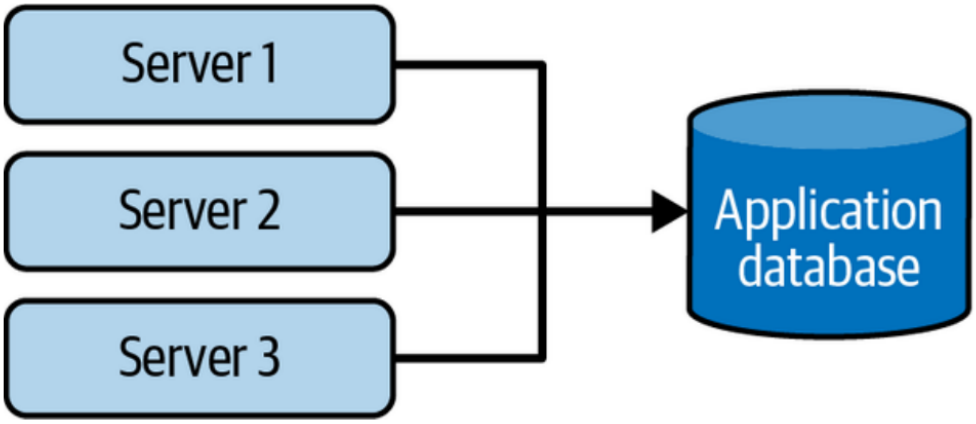

Figura 2-3: Ejemplo de sistema fuente: una base de datos de aplicación

2. **Enjambre IoT (IoT Swarm):** Flota de dispositivos enviando mensajes a una cola central.

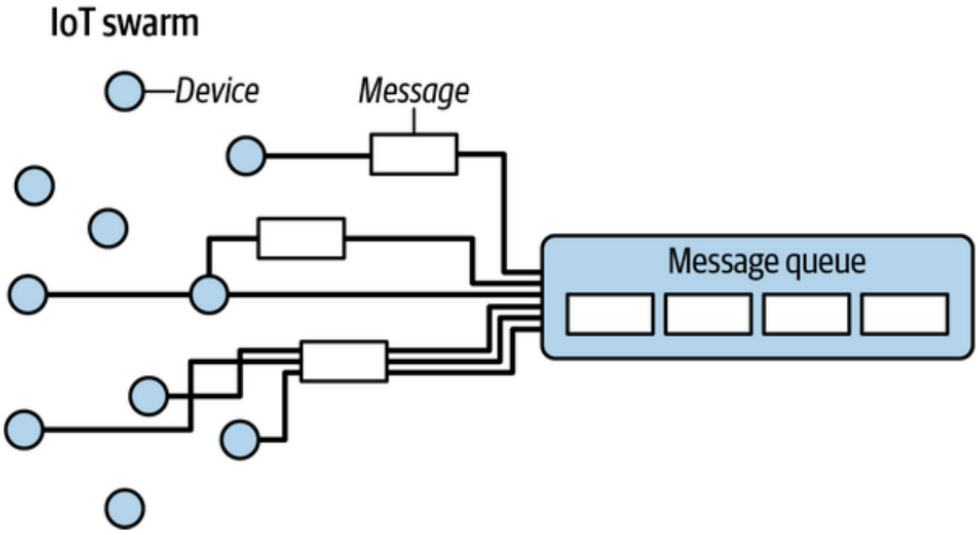

Figura 2-4: Ejemplo de sistema fuente: un enjambre IoT y cola de mensajes

Consideraciones clave al evaluar fuentes:

- **Características:** ¿Es una aplicación, IoT, web?
- **Persistencia:** ¿Se guardan los datos a largo plazo o se borran rápido?

- **Tasa de generación:** Eventos por segundo, GB por hora.
- **Consistencia:** ¿Calidad de datos, nulos, formato?
- **Errores y Duplicados:** Frecuencia.
- **Latencia:** ¿Llegan datos tarde (*late arriving data*)?
- **Esquema:** ¿Estático o cambia? ¿Cómo se comunican los cambios?

Esquema (Schema)

El esquema define la organización jerárquica de los datos.

- **Sin esquema (Schemaless):** La aplicación define el esquema al escribir (ej. MongoDB, logs).
- **Esquema fijo:** La base de datos fuerza la estructura (ej. RDBMS SQL).

El trabajo del ingeniero a menudo implica transformar datos crudos del esquema de la fuente a un formato valioso para analítica.

---

2. Almacenamiento (Storage)

Elegir la solución de almacenamiento es clave y complejo. A menudo se usan múltiples soluciones (Data Warehouse, Data Lake, Object Storage). El almacenamiento toca casi todas las etapas (ingesta, transformación, servicio).

Consideraciones clave de ingeniería:

- **Velocidad:** ¿Compatible con las necesidades de lectura/escritura?
- **Cuello de botella:** ¿El almacenamiento frenará procesos downstream?
- **Uso óptimo:** Evitar "actos antinaturales" (ej. altas actualizaciones aleatorias en Object Storage).
- **Escalabilidad:** Límites de capacidad y volumen.
- **Metadatos:** Captura de esquema, linaje, evolución.

- **Tipo:** ¿Puro almacenamiento (S3) o con motor de consulta (Warehouse)?

Frecuencia de Acceso a Datos (Temperatura)

- **Datos Calientes (Hot Data):** Accedidos muchas veces por segundo. Requieren almacenamiento rápido (y caro).
- **Datos Tibios (Lukewarm Data):** Accedidos cada semana o mes.
- **Datos Fríos (Cold Data):** Rara vez consultados. Para auditoría o recuperación ante desastres. Almacenamiento barato y lento (ej. S3 Glacier).

---

3. Ingesta (Ingestion)

Es el proceso de mover datos desde la fuente al almacenamiento o sistema de procesamiento. Suele ser el mayor cuello de botella por la falta de control sobre la fuente.

Consideraciones clave:

- **Casos de uso:** ¿Para qué se usarán los datos? ¿Reutilizables?
- **Fiabilidad:** ¿Está disponible la fuente?
- **Destino y Frecuencia:** ¿A dónde van y qué tan seguido?
- **Volumen y Formato.**
- **Calidad:** ¿Necesita limpieza inmediata?

Batch vs. Streaming

- **Streaming (Tiempo real):** Los datos se procesan continuamente, poco después de generarse (baja latencia). Es complejo y costoso.
- **Batch (Lotes):** Los datos se procesan en intervalos o por tamaño acumulado. Es más simple y estándar para reportes y entrenamiento de modelos.
  - *Consejo:* No adoptar streaming "porque sí". Evaluar si el negocio realmente necesita latencia de milisegundos.

Push vs. Pull

- **Push:** La fuente envía los datos al destino (ej. IoT enviando a una cola, CDC enviando cambios).

- **Pull:** El sistema de ingesta consulta y extrae los datos de la fuente (ej. ETL tradicional consultando una DB cada noche).

---

4. Transformación

Aquí es donde los datos comienzan a generar valor. Se convierten de su forma original a algo útil para análisis o ML. Sin transformación, los datos son inertes.

Tipos de transformación:

- **Básica:** Mapeo de tipos de datos, estandarización de formatos.

- **Normalización/Modelado:** Estructuración según lógica de negocio (Data Modeling).

- **Agregación/Featurization:** Resumen para reportes o creación de *features* para ML.

Consideraciones clave:

- **Costo/ROI:** ¿Vale la pena el proceso?

- **Lógica de negocio:** La transformación traduce reglas de negocio (ej. qué define una "venta") en código reutilizable. Es vital para tener una "única fuente de verdad".

---

5. Servicio de Datos (Serving Data)

La etapa final: entregar valor a los usuarios. Los datos no consumidos son "proyectos de vanidad".

Usos populares:

1. **Analítica (Analytics):**

- **Business Intelligence (BI):** Reportes sobre el pasado/presente. Lógica de negocio centralizada.

- **Analítica Operacional:** Detalles de grano fino en tiempo real para acción inmediata (ej. monitoreo de servidores).

- **Analítica Embebida:** Datos para clientes dentro de un producto (SaaS). Requiere seguridad robusta y multitenencia.

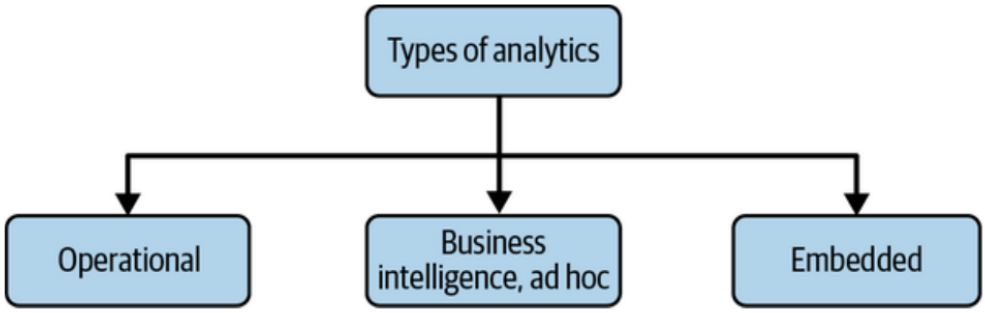

Figura 2-5: Tipos de analítica

2. **Machine Learning (ML):**

   - Los ingenieros de datos proveen la infraestructura para entrenar y servir modelos.

   - **Feature Store:** Herramienta que une ingeniería de datos y ML para gestionar y compartir *features*.

3. **Reverse ETL:**

   - Mover datos procesados *desde* el almacén de datos *hacia* sistemas operativos (ej. Salesforce, Google Ads).

   - Permite "operacionalizar" los insights analíticos.

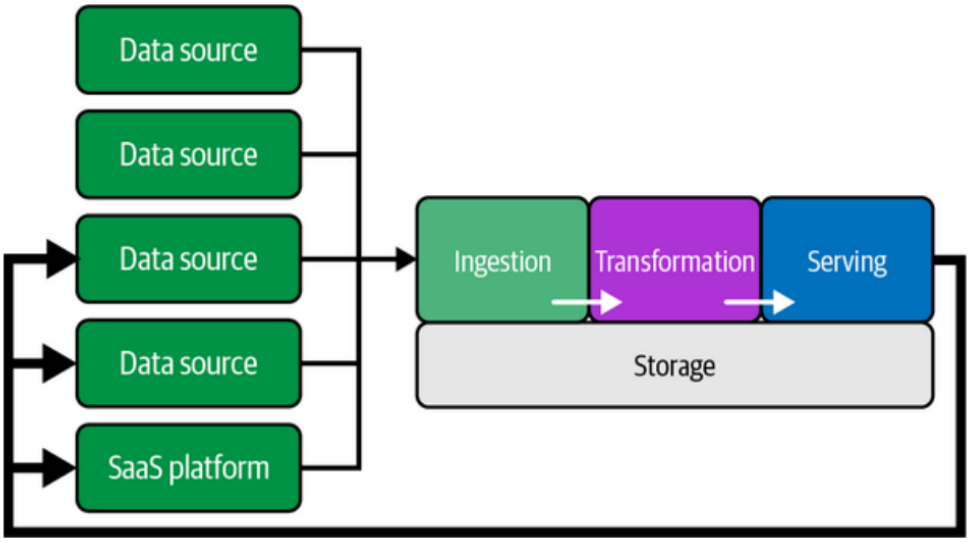

Figura 2-6: Reverse ETL

Corrientes Subyacentes (Undercurrents)

Son las **disciplinas transversales** necesarias para garantizar que el ciclo de vida de la ingeniería de datos sea **confiable, escalable, seguro y mantenible**. No son pasos secuenciales, sino prácticas continuas que deben aplicarse en todo momento, desde que el dato se genera hasta que se sirve.

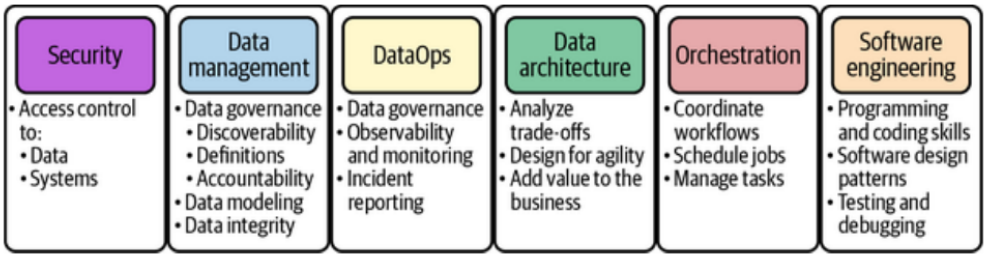

Figura 2-7: Las principales corrientes subyacentes de la ingeniería de datos

1. Seguridad (Security)

Es la corriente más importante ("Seguridad Primero"). Antes era responsabilidad del departamento de TI; hoy, **todo ingeniero de datos es un ingeniero de seguridad**.

- **Concepto Clave:** El **Principio de Menor Privilegio** (*Principio del menor privilegio*).
  - Significa dar acceso a un usuario o sistema **solo** a lo que necesita estrictamente para hacer su trabajo, y solo por el tiempo necesario.
- **Aplicación en el Ciclo:**
  - *Ingesta:* ¿Quién tiene permiso para leer la base de datos fuente?
  - *Almacenamiento:* ¿Están los datos encriptados en reposo (en el disco)?
  - *Servicio:* ¿Están enmascarados los datos sensibles (PII) antes de mostrarlos en un dashboard?
- **Responsabilidad Compartida (Nube):** El proveedor (AWS/Azure) asegura la nube (el hardware), tú aseguras lo que pones dentro (tus datos y accesos).

2. Gestión de Datos (Data Management)

Es el conjunto de prácticas para tratar los datos como un activo corporativo valioso. Convierte el "caos de datos" en orden. Incluye varios sub-componentes:

- **Gobernanza de Datos:** Las políticas y reglas. ¿Quién es el dueño de este dataset? ¿Cuánto tiempo debemos guardarlo por ley?
- **Descubribilidad:** Catálogos de Datos. Permite que los analistas encuentren los datos sin tener que preguntar por Slack "¿dónde está la tabla de ventas?".
- **Calidad de Datos:** Asegurar que el dato sea preciso, completo y oportuno. (Validación de esquemas, checks de nulos).
- **Linaje (Lineage):** El rastro de auditoría. ¿De dónde salió este dato? ¿Qué transformaciones sufrió? Si el reporte final está mal, el linaje te permite rastrear el error hacia atrás hasta la fuente.
- **Ética y Privacidad:** Cumplir con GDPR/CCPA. Manejo de datos personales.

3. DataOps

Es la aplicación de la filosofía **DevOps** y las prácticas *Lean* (manufactura esbelta) al mundo de los datos. Busca **agilidad y automatización**.

Tiene tres pilares técnicos fundamentales:

1. **Automatización:** Eliminar el trabajo manual. Usar **CI/CD** (Integración y Despliegue Continuo) para desplegar pipelines automáticamente. Usar **Infraestructura como Código** (Terraform) para no configurar servidores a mano.

2. **Observabilidad y Monitoreo:** No basta con que el pipeline corra. Tienes que saber *cómo* corrió. ¿Fue lento? ¿Hubo latencia? ¿Llegaron menos datos de lo normal? La observabilidad te avisa antes de que el usuario se queje.

3. **Respuesta a Incidentes:** Cuando algo falla (y fallará), tener un proceso claro y sin culpa (*blameless*) para arreglarlo rápido.

4. Arquitectura de Datos

Es el plano de diseño. Implica tomar las decisiones estructurales de alto nivel basándose en los requisitos del negocio.

- **Trade-offs (Compensaciones):** El arquitecto (o ingeniero) debe decidir qué sacrificar. ¿Queremos velocidad extrema (Streaming) o bajo costo (Batch)? ¿Queremos flexibilidad (Data Lake) o estructura (Warehouse)?

- **Diseño:** Aquí es donde decides si usar una arquitectura Lambda, Kappa, o Lakehouse.

- **Evolución:** Una buena arquitectura no es estática; es flexible y permite cambiar componentes (Principio de decisiones reversibles).

5. Orquestación

Es el "Director de la Orquesta". Es el sistema que coordina el flujo de trabajo para que las cosas pasen en el orden y tiempo correctos.

- **El Problema:** Tienes una tarea que baja datos, otra que limpia, otra que entrena un modelo. No pueden correr todas a la vez.

- **La Solución:** Herramientas como **Airflow**, **Dagster** o **Prefect**.

- Usan **DAGs** (Grafos Acíclicos Dirigidos) para definir dependencias: "La Tarea B solo arranca si la Tarea A terminó con éxito".

- Gestionan reintentos (retries) si algo falla y alertas.

6. Ingeniería de Software

Esta es la base técnica del rol moderno. Según Reis, **"Los ingenieros de datos son ingenieros de software especializados en datos"**.

- **El cambio de mentalidad:** Ya no se trata de escribir scripts SQL sueltos o usar herramientas "arrastrar y soltar" (ETL visuales) sin control.

- **Prácticas:**

  - Escribir código limpio y modular (Python, Scala, SQL).

  - Usar Control de Versiones (**Git**).

  - Escribir Pruebas (**Testing**): Unitarias, de integración y de datos.

  - Hacer *Code Reviews* (revisiones de código).

Conclusión

El capítulo establece que la ingeniería de datos no es solo usar herramientas, sino gestionar un ciclo de vida complejo para entregar valor.

El modelo mental es:

Etapas (Generación ⇒ Almacenamiento ⇒ Ingesta ⇒ Transformación ⇒ Servicio) + Corrientes Subyacentes (Seguridad, DataOps, etc.).