Microservicios, capas y pipelines

Explicaciones de conceptos importantes para la arquitectura de datos.

November 24, 2025

Pregunta 1.

Mi pregunta:
En el marco del capítulo 3 del libro de Reis y Housley, ¿me explicás lo que son los microservicios?

En el contexto del **Capítulo 3** de *Fundamentals of Data Engineering*, Joe Reis y Matt Housley presentan los **Microservicios** no como una "moda" de programación, sino como un **patrón de arquitectura** fundamental para entender cómo diseñar sistemas modernos, escalables y resilientes.

Para tu examen, la clave es entenderlos en **oposición al Monolito** y bajo el principio de **Acoplamiento Débil** (*Loose Coupling*).

Aquí tienes la explicación desglosada:

1. ¿Qué son los Microservicios?

Son un estilo de arquitectura donde una aplicación (o sistema de datos) se construye como un conjunto de **servicios pequeños, autónomos y separados**.

- **Polar opuesto al Monolito:** Mientras que el monolito es "todo bajo un mismo techo" (una sola base de código, una sola base de datos), los microservicios dividen esa funcionalidad en piezas pequeñas.
- **Descentralizados:** Cada microservicio tiene su propia lógica y, a menudo, su propia base de datos.
- **Débilmente Acoplados (Loosely Coupled):** Esta es la característica más importante según el libro. Significa que los servicios funcionan independientemente.

2. Características Principales (Según el libro)

1. **Responsabilidad Única:** Cada servicio hace una cosa y la hace bien (ej. un servicio solo maneja "Pedidos", otro solo maneja "Usuarios").

2. **Aislamiento de Fallos:** Si el servicio de "Recomendaciones" se cae, el servicio de "Pagos" sigue funcionando. La aplicación no muere por completo (a diferencia del monolito).

3. **Independencia Tecnológica:** Puedes escribir un servicio en Python y otro en Java, y que se comuniquen entre sí (generalmente vía APIs REST).

4. **Despliegue Independiente:** Un equipo puede actualizar el servicio de Ventas sin tener que redeployar toda la plataforma de la empresa.

3. ¿Cómo se aplica esto a la Ingeniería de Datos?

El libro hace un paralelo interesante. Aunque los microservicios nacieron en la ingeniería de software, el concepto se traslada a los datos:

- **El Monolito de Datos:** Es el **Data Warehouse Centralizado tradicional**. Todos los pipelines apuntan a un solo lugar gigante. Si ese Warehouse se cae o se llena, toda la empresa se queda ciega.

- **El Enfoque Microservicios en Datos:** Es dividir la arquitectura por dominios (similar al **Data Mesh**).

  - Tener un pipeline específico y un almacén específico para el dominio de "Ventas".

  - Otro separado para "Logística".

  - Se conectan entre sí mediante interfaces estándar, no mediante una base de datos compartida gigante y frágil.

4. La Advertencia de Reis (Pragmatismo)

Ojo con esto para el examen. Los autores **NO** dicen que los microservicios sean siempre mejores. De hecho, advierten sobre la complejidad.

- **El Monolito no es el enemigo:** A veces, para empezar rápido, un monolito es mejor porque es simple.

- **El costo de los Microservicios:** Al separar todo, introduces **complejidad de red, orquestación y consistencia**.

- **Consejo:** No rompas un monolito solo por moda. Hazlo solo si necesitas escalar organizativamente o técnicamente y el monolito se ha convertido en una "gran bola de lodo" (*big ball of mud*) inmanejable.

Resumen para el Examen 1.

Los **Microservicios** son un patrón de arquitectura **descentralizada y débilmente acoplada**, donde el sistema se divide en componentes pequeños y autónomos.
En ingeniería de datos, este patrón inspira arquitecturas modernas (como Data Mesh) para evitar los cuellos de botella de los sistemas centralizados, permitiendo que los equipos escalen y fallen de manera independiente sin tumbar todo el sistema.

Pregunta 2.

Mi pregunta:
¿Y cómo se relacionan los modelos de monolito y microservicios con las capas (tiers)? ¿Los microservicios son siempre multicapa?

Esta es una pregunta brillante porque toca el punto donde la mayoría se confunde. Para entenderlo, hay que visualizar la arquitectura en **dos dimensiones diferentes**.

Según el libro de Reis y Housley (Capítulo 3), **Monolitos/Microservicios** y **Capas (Tiers)** son conceptos ortogonales (distintos pero relacionados).

Aquí tienes la explicación definitiva para tu examen:

---

1. La Diferencia Dimensional

- **Capas (Tiers) → Corte HORIZONTAL (Técnico)**
  - Se refiere a la **infraestructura física/lógica**.
  - Separa funciones técnicas: "Aquí se guardan los datos (DB)", "Aquí se procesan (App)", "Aquí se muestran (UI)".
  - *Pregunta clave:* ¿La base de datos y el código corren en la misma máquina?

- **Monolito vs. Microservicios → Corte VERTICAL (Funcional)**

- Se refiere a la **lógica de negocio y el código**.
- Separa dominios: “Ventas”, “Inventario”, “Usuarios”.
- *Pregunta clave:* ¿El código de Ventas y el de Inventario están en el mismo proyecto/repositorio?

---

2. ¿Cómo se relacionan? (Las Combinaciones)

Puedes mezclar estos conceptos. Veamos los casos según el libro:

A. Monolito de Capa Única (Single-Tier Monolith)

- **Qué es:** Todo tu código (Ventas + Inventario) y tu base de datos están instalados en **un solo servidor** (o en tu laptop).
- **Veredicto de Reis:** Está bien para prototipos, **terrible para producción**. Si falla el servidor, pierdes todo.

B. Monolito Multicapa (Multitier Monolith)

- **Qué es:** Tienes **un servidor gigante** para la Aplicación (que tiene todo el código junto) y **otro servidor gigante** para la Base de Datos.
- **Relación:** Es un Monolito (código unido) pero es Multicapa (infraestructura separada).
- **Veredicto:** Es el estándar clásico de las empresas antiguas. Es robusto pero difícil de cambiar (si quieres cambiar algo de Ventas, tienes que redeployar todo el servidor de aplicaciones).

C. Microservicios (Por definición, tienden a ser Multicapa)

- **Qué es:** Tienes un servidor pequeño para “Ventas” y otro servidor pequeño para “Inventario”.
- **Relación:**
  - Cada microservicio suele tener su propia **Lógica** (Compute) y su propia **Data** (Storage) separadas.

– Por lo tanto, la arquitectura de microservicios es inherentemente **Multicapa y Distribuida**.

---

3. ¿Los microservicios son SIEMPRE multicapa?

En la práctica profesional y según el libro: SÍ.

¿Por qué? Porque el libro enfatiza el principio de **"Build Loosely Coupled Systems" (Construir sistemas débilmente acoplados)**.

1. **Desacoplamiento de Recursos:** Un microservicio necesita escalar su cómputo (CPU) independientemente de su almacenamiento (Disco).

   - *Ejemplo:* El servicio de "Procesamiento de Imágenes" necesita mucha CPU pero poca base de datos. Si metes la DB dentro del mismo contenedor (Single-tier), no puedes escalar eficientemente.

2. **Resiliencia:** Si pones la base de datos dentro del mismo contenedor del microservicio (Single-tier) y el contenedor crashea, pierdes los datos o la conexión. Por eso, siempre se usa una base de datos externa (RDS, DynamoDB, etc.), lo que crea automáticamente una arquitectura **Multicapa**.

Resumen para el Examen 2.

- **Tiers (Capas):** Tratan sobre separar **Recursos** (Cómputo vs. Almacenamiento).
- **Microservicios:** Tratan sobre separar **Dominios** (Ventas vs. Inventario).

Conclusión: Un sistema de **Microservicios** es una forma de organizar la lógica que, para ser robusta y escalable (Principios de Reis), **debe implementarse sobre una arquitectura Multicapa**, donde el cómputo y los datos están desacoplados físicamente.

Pregunta 3.

Mi pregunta:
¿Qué son y qué hacen los *pipelines* en este contexto de ingeniería de datos, en el marco del libro de Reis y Housley?

En el contexto del libro *Fundamentals of Data Engineering* de Joe Reis y Matt Housley, los **Data Pipelines** (Tuberías de Datos) no son simplemente "scripts que mueven datos". Son la **infraestructura circulatoria** que conecta todas las etapas del ciclo de vida.

Para tu examen, aquí tienes la definición y función desglosada según la filosofía de los autores:

1. ¿Qué son los Pipelines de Datos?

Son conjuntos de procesos automatizados que mueven datos desde un sistema fuente hasta un sistema de destino, transformándolos en el camino para que sean útiles.

La definición conceptual de Reis: Si el Ciclo de Vida (Generación → Almacenamiento → Ingesta → Transformación → Servicio) son los "órganos" del cuerpo de datos, los **Pipelines** son las **venas y arterias** que transportan la sangre (el dato) entre ellos.

Técnicamente (La estructura): En la ingeniería moderna, un pipeline se representa casi siempre como un **DAG (Directed Acyclic Graph)** o Grafo Acíclico Dirigido.

- **Directed:** Tiene una dirección (del paso A al paso B).
- **Acyclic:** No tiene bucles infinitos (no vuelve de B a A).
- **Graph:** Es un mapa de tareas dependientes entre sí.

2. ¿Qué hacen los Pipelines? (Sus funciones)

Su función principal es **conectar las etapas aisladas** del ciclo de vida. Sin pipelines, el dato se queda estancado en la fuente.

Sus tareas específicas son:

1. **Ingesta (Extracción):** Sacar el dato del sistema fuente (API, Base de Datos, Logs) sin romperlo.
2. **Transporte:** Mover el dato de manera segura y fiable a través de la red hacia el almacenamiento (Data Lake/Warehouse).
3. **Transformación:** Ejecutar la lógica de negocio (limpiar, agregar, filtrar, unir) para convertir datos crudos en información.
4. **Carga/Entrega:** Depositar el dato final en el sistema de consumo (BI, ML).

3. Los Dos Modos de Operación

El libro enfatiza la distinción entre dos tipos de pipelines:

- **Pipelines Batch (Por lotes):** Procesan un bloque finito de datos en un intervalo programado (ej: "Correr todos los días a las 00:00hs").
  - *Ventaja:* Más fáciles de construir y recuperar ante fallos.
- **Pipelines Streaming (Flujo continuo):** Procesan cada evento de dato en tiempo real o casi real, apenas se genera.
  - *Ventaja:* Baja latencia.
  - *Nota de Reis:* "Casi todos los datos nacen como stream, pero a menudo es más práctico procesarlos como batch".

4. Las "Reglas de Oro" de los Pipelines según Reis

Para que un pipeline sea "buena ingeniería" y no un script frágil, debe cumplir con ciertos principios que los autores repiten mucho (y son **preguntas de examen**):

A. Idempotencia (Idempotency)

Esta es la propiedad más importante.

- **Definición:** Si ejecutas el mismo pipeline con los mismos datos de entrada **múltiples veces**, el resultado debe ser **siempre el mismo**. No debe duplicar datos.
- *Ejemplo:* Si el pipeline falló a la mitad y lo corres de nuevo, no debería cargar las ventas dos veces. Debe ser capaz de limpiar y reescribir o ignorar lo ya procesado.

B. Pipelines como Código (Pipelines as Code)

Los pipelines no deben ser configuraciones manuales o clics en una herramienta visual ("drag and drop"). Deben ser **código** (Python, SQL, YAML).

- Esto permite usar control de versiones (Git), pruebas (Testing) y CI/CD (DataOps).

C. Reproducibilidad

Debes poder regenerar un dataset antiguo volviendo a correr el pipeline con los datos históricos y obtener el mismo resultado.

Resumen para el Examen 3.

En el marco de Reis y Housley, los **Data Pipelines** son el mecanismo de software (generalmente orquestado como DAGs) que **automatiza el flujo de datos** a través del ciclo de vida.
Su función es mover y transformar datos garantizando **fiabilidad** y **consistencia**, preferiblemente siguiendo el principio de **Idempotencia**.