Microservicios

MitoCode Network

Por: Andres Gonzales

MitoCode

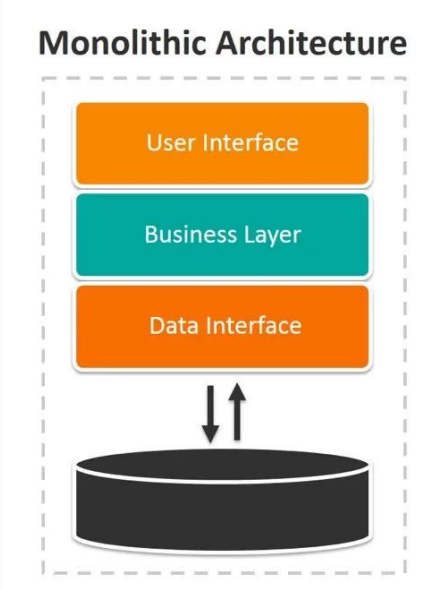
Agenda

- ¿Arquitectura Monolítica y de Microservicios?
- Caso Netflix
- Microservicios

MitoCode

¿Qué es una arquitectura monolítica?

Es una aplicación construida como un **bloque único**. Todo el código (controladores, servicios, lógica de negocio, acceso a datos) vive y se despliega como una sola unidad.



MitoCode

Lo bueno y lo malo

Ventajas

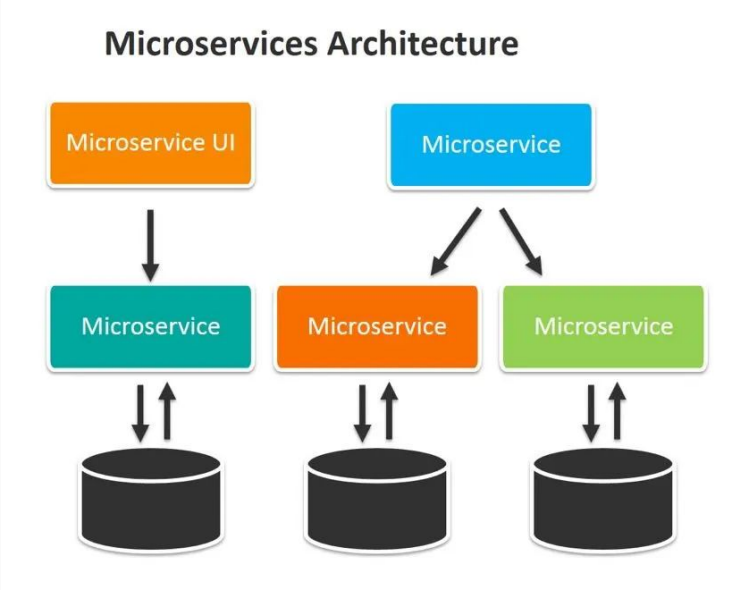
- Simplicidad inicial
- Menor complejidad operativa
- Desarrollo más rápido al inicio
- Depuración más directa
- Despliegue único

Desventajas

- Escalado limitado
- Alta dependencia entre módulos
- Despliegues arriesgados
- Dificultad para adoptar nuevas tecnologías
- Curva de mantenimiento creciente
- Límites difusos de responsabilidad
- Tiempo de inicio prolongado

¿Qué es una arquitectura de microservicios?

Una **arquitectura de microservicios** es un estilo de diseño de software donde una aplicación se divide en **pequeños servicios independientes**, cada uno ejecutándose por separado y comunicándose entre sí (generalmente por HTTP o mensajería).



MitoCode

Lo bueno y lo malo

Ventajas

- Escalabilidad específica por componente.
- Tiempos de despliegue más cortos.
- Mayor resiliencia: una falla no tumba todo el sistema.
- Equipos pequeños y autónomos.

Desventajas

- Complejidad operativa mayor.
- Costos en infraestructura (más servicios, más máquinas).
- Depuración y trazabilidad más difíciles.
- Gestión de consistencia distribuida.

Caso real: Netflix

En 2008, Netflix comenzó a migrar de una arquitectura monolítica (Java + Oracle) hacia una basada en microservicios.

Su sistema estaba fallando al manejar picos de carga y la recuperación ante fallos era muy lenta.

Necesitaban:

- Escalabilidad por componente
- Despliegue rápido
- Resiliencia

Resultado: dividieron su sistema en **cientos de microservicios autónomos**, como “servicio de recomendación”, “servicio de usuario”, “servicio de streaming”, etc.

¿Por qué Netflix fue clave?

Adoptó microservicios para poder escalar, desplegar y mantener más de 1000 servicios de forma autónoma.

Creó herramientas como:

- **Eureka** (descubrimiento de servicios)
- **Ribbon** (balanceo de carga)
- **Hystrix** (Circuit Breaker resiliencia, ya deprecado)
- **Zuul** (API Gateway, luego reemplazado por Spring Cloud Gateway)

¿Por qué Netflix fue clave?

Netflix no solo resolvió sus propios problemas, sino que **cambió el panorama del desarrollo backend** al compartir sus herramientas con la comunidad. Spring Cloud las adoptó, las estandarizó y ahora son parte de muchas arquitecturas modernas.

Pero… ¿Qué son los Microservicios?

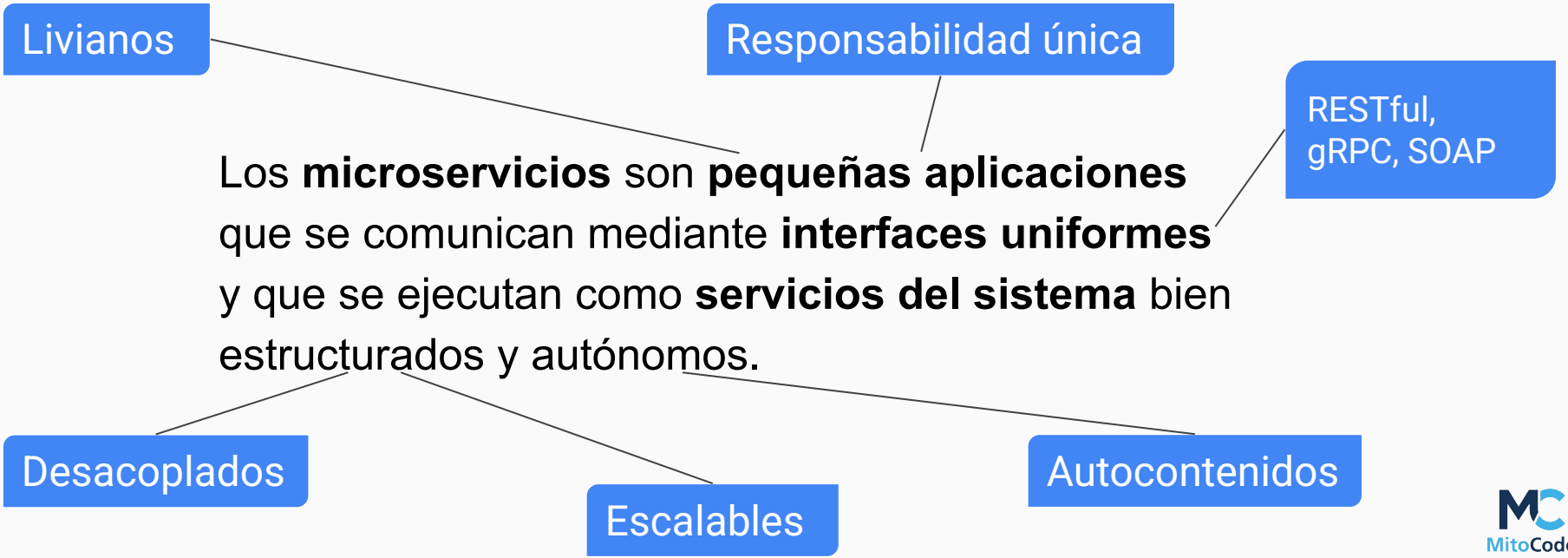
Década del 2000: el término "microservicios" aún no existía, pero algunas empresas empezaban a dividir aplicaciones monolíticas en módulos más pequeños.

2011–2012: *Netflix* enfrentaba problemas de escalabilidad con su arquitectura monolítica Java, y comenzó a dividir su sistema en pequeños servicios independientes.

2014: el término *microservices* se popularizaba tras un artículo de Martin Fowler y James Lewis.

Desde entonces, empresas como **Amazon, Uber, Spotify, Twitter** adoptaron esta arquitectura.

Microservicios



MitoCode

¿Por qué surgió esta necesidad?

Monolitos muy grandes eran difíciles de mantener, probar y escalar.

Se necesitaba:

- Despliegues rápidos
- Alta disponibilidad
- Escalabilidad horizontal
- Independencia por equipo de desarrollo

Pero… ¿Qué tan **micro** debe ser un microservicio?

¿Debe tener solo 300 líneas de código?

¿Debe hacer solo una cosa?

¿Hay una medida exacta?

¿Qué tan **micro** debe ser un microservicio?

Regla del equipo de 2 pizzas *(Jeff Bezos – Amazon)*

Un equipo debe ser lo suficientemente pequeño como para alimentarlo con 2 pizzas.
Si el equipo que mantiene un servicio necesita más personas, el servicio puede ser demasiado grande.

Enfócate en lo que importa

Un microservicio debe resolver una única responsabilidad clara y aportar valor de negocio.

Apóyate en DDD y Bounded Contexts

Un microservicio debería representar un *contexto delimitado* del dominio, no una funcionalidad técnica genérica.

~~**No más de 300 líneas de código** *(mito)*~~

No hay una regla estricta sobre líneas de código. Lo importante es la **cohesión** y **autonomía**, no el tamaño físico.

MitoCode

Más que una tecnología…
es una filosofía de diseño

No hay una regla estricta sobre tamaño o líneas de código.

Casi cualquier aplicación **podría convertirse en un microservicio** si cumple con ciertos principios:

- Es **autónoma**
- Tiene **alta cohesión** y **bajo acoplamiento**
- Se puede **desplegar de forma independiente**
- Representa una **unidad funcional clara** dentro del negocio (Bounded Context)

Se apoya en conceptos como **arquitectura distribuida**, **desacoplamiento**, **resiliencia**, y **automatización del despliegue**.

Entonces…

*El microservicio no es un archivo `.jar` pequeño. Es una **forma de pensar y diseñar software** que favorece la autonomía, la especialización y la evolución continua.*

¿Cuándo Si/No usar Microservicios?

Cuando Si:

- Equipos grandes y distribuidos
- Alta escalabilidad necesaria
- Despliegue frecuente por módulos

Cuando No:

- Proyectos pequeños o MVPs
- Equipos chicos
- Cuando se prioriza simplicidad

Todo es relativo…

MitoCode

Stack de Desarrollo

- Java 25
- Spring Boot 4.0.0
- Maven
- Docker
- Kafka
- IDE: IntelliJ IDEA (o IDE de preferencia)
- Git
- PostgreSQL + MongoDB

MitoCode

MitoCode Network

www.mitocode.com