



Siguen sin ser la solución definitiva, es de hecho, otro tipo de arquitectura.

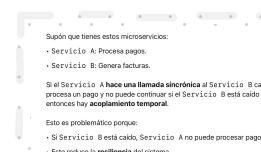
T6: Microservicios

Introducción

- Se basan en la idea de **designer separation**
- Independencia de otros módulos
- Máximo de responsabilidad única - un módulo debe ser responsable SOLO y únicamente de una funcionalidad
- Menos redondeamiento (más inter entre módulos)
- Lo contrario → Arq. Monolíticos → el código del sistema se desarrolla en un mismo proceso
 - Escalabilidad
 - Acaparamiento
 - Conflicto entre desarrolladores vs ownership y toma de decisiones
 - Acaparamiento en el desarrollo
 - Complicaciones cross
- Problemas de desarrollo
 - Monología, de desarrollo sencilla (todo se desarrolla en uno solo SIG)
 - Monitoreo con sentinela
 - end-to-end testing, ya q el flujo de desarrollo es claro
 - Rendimiento
 - Resiliencia de diseño
- Ventajas
 - monología, de desarrollo sencilla (todo se desarrolla en uno solo SIG)
 - Monitoreo con sentinela
 - end-to-end testing, ya q el flujo de desarrollo es claro
 - Rendimiento
 - Resiliencia de diseño
- Tipos de ARQ:
 - Single-process system: menor más robusto, tiene siendo un único proceso (no escalables)
 - Monolito distribuido: n servicios distinguidos todos juntos (entre muchos)
 - Third party black box system: sus desarrollado por otro, q no tenemos acceso a él

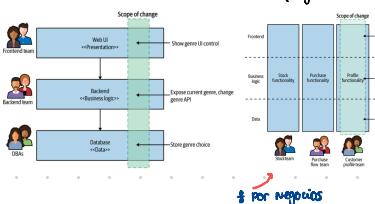
Microservicios

- Un tipo de Service-Oriented architecture (SOA)
- Las funcionalidades se modelan en un dominio de negocio → donde (fuera SOA) agrega
- Servicios liberables de forma index (despliegue index)
- Encapsulan funcionalidad accesible a otros technology agnostic
- Info hiding → ocultar tanto info como sea posible
- Explorar lo nuevo posible a través de las interfaces
- ↑ cohesion y ↓ acoplamiento

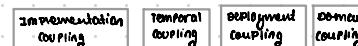


Arquitectura tradicional vs Arquitectura microservicios

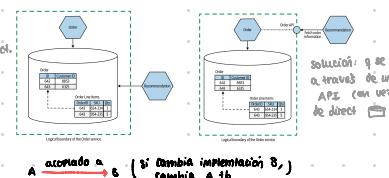
Caso práctico: tenemos una tienda de música, y hoy a cuadra una nueva música (logística), el cliente puede escuchar, generar cd's



TIPOS DE ACOPLAMIENTO



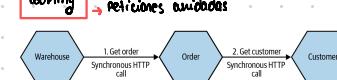
Implementation coupling



A → acoplado a B → si cambia implementación A → cambia a B, B → cambia implementación B → cambia a A

- El riesgo es de acoplamiento
- Risk of Solution
- cambiar una BD

Temporal coupling → peticiones asincronas



- ✓ todos los servicios tienen q estar disponibles
- ✓ usar cacheando
- ✓ mensajes asincrónicos

Deployment coupling

- Un request cambio → desplegar todo → Aun el resto corre igual
- Soluciones: Descomponer los procesos grandes en microservicios independientes
- Ventajas: En vez de despliegues fijos (release train), continuos delivery

Domain coupling

- Interacción entre servicios en nuestro dominio
- En cierto modo → INEVITABLE
se puede reducir la info compartida

Exemplo negativo

- Supón que tienes:
- Servicio A: Servicio de usuarios.
- Servicio B: Servicio de pedidos.

Si ambos servicios usan una misma clase o entidad de "Cliente" con los mismos campos y lógica, y cualquier cambio en la definición de esa entidad afecta a ambos, entonces hay acoplamiento de dominio.

También ocurre si un servicio entiende demasiado del negocio del otro o si comparte clases o librerías de dominio comunes, lo que es un antíptico en microservicios.

Beneficios Microservicios

- Experiencia del desarrollador
- Scalability tecnológica
- Coste → más procesos, máquinas, red, almacenamiento...
- Informes → la info está estandizada en AI sitios
- Monitorización y ↓ de errores
- Seguridad
- Testing
- Lateralidad
- Consistencia de datos → BBDD distribuidos

Comunicaciones

son mucho más inefficientes dado que la comunicación es ENTRE PROCESOS

"se procesan en el mismo proceso o entre procesos"

- Alto Rendimiento
 - no hay optimizaciones del compilador
 - Retardo
 - ↓ en el # de llamadas
- Cambio de interfaces
- Buena de errores

TIPOS DE COMUNICACIÓN

Biogeo sincrónico: biogeoizado hasta q se obtiene respuesta

Asincrónico: no biogeoizado

→ Asincrónico temporal



Asincrónico no biogeoizado: no biogeoiza el microservicio q hace la llamada

→ complejidad

→ # de operaciones

T6: Microservicios

Microservicios

Tipos de Comunicación

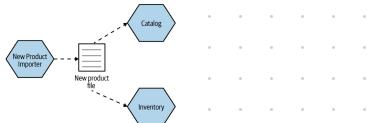
Bloqueo Racionado

Asincrónas no bloqueantes

Common Data

- un microservicio pone la info en una localización →
- otro microservicio tiene q encontrar la localización →

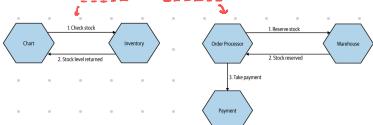
Asincrónico Lugar de datos
Fichero
Base de datos
Data warehouse



Request - Response : un microservicio hace una petición y espera una respuesta

- puede ser solicitud de info
- puede ser para realizar una acción

Puede ser sincrónico o asincrónico



Event-driven → asincrónico
Cuando: almacén de galletas ha sucedido

Tecnologías de microservicios

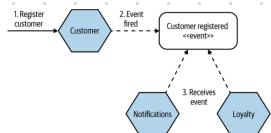
- Remote Procedure Calls (RPC)
- SOAP
- REST
- Graph API
- Message Brokers

Despliegue en microservicios

- Contenedores (Containers)
- Function as a Service
 - Any language
 - Cuando nade x ejec. este código

Un microservicio emite eventos (productor)

- Ayden o no ser consumidos por otros (clientes)
- El productor no sabe lo q van a hacer los clientes con lo q da



Asincrónico no bloqueante - Event Driven
Implementación:

- uso de message brokers (middleware)
- q. subscriben a un broker para recibir eventos
- Productores usan una API para publicar un evento al broker

RabbitMQ
Kafka

...