ARQUITECTURAS DE SOFTWARE

TECNOLOGÍAS WEB I

Carlos Rojas Sánchez

Licenciatura en Informática

Universidad del Mar

Contenido

Arquitecturas de Software

Definiciones

"La arquitectura de software es la organización fundamental de un sistema, representada por sus componentes, sus relaciones entre ellos y con el entorno, y los principios que guían su diseño y evolución."
(IEEE 1471-2000, ahora ISO/IEC/IEEE 42010)

Definiciones

“La arquitectura de software es la estructura o estructuras del sistema, compuestas por elementos de software, sus propiedades visibles externamente y las relaciones entre ellos.”
(Len Bass, Paul Clements y Rick Kazman (Autores del libro ”Software Architecture in Practice”))

Definiciones

“La arquitectura de un sistema de software abarca las decisiones sobre el diseño más significativas: aquellas que afectan el sistema de forma general.”
Roy Fielding (Creador de REST)

Definiciones

“La arquitectura de software es el conjunto de estructuras necesarias para razonar sobre el sistema, que comprenden elementos de software, relaciones entre ellos y propiedades de ambos.”
Software Engineering Institute (SEI)

Definiciones

“La arquitectura de software es el plano general del sistema: cómo se organiza, cómo se divide en partes, cómo se comunican entre ellas y qué reglas rigen su evolución.”
Definición práctica y sencilla (Según Chatgpt)

Arquitecturas de Software Monolítica y de Microservicios

⚙ Arquitectura Monolítica

Características:

- Toda la aplicación está en una sola unidad de despliegue.
- Todos los módulos (auth, base de datos, lógica de negocio, frontend, etc.) están juntos.
- Se ejecuta todo en un solo proceso.

⚙ Arquitectura Monolítica

Ventajas:

- Fácil de desarrollar y desplegar al principio.
- Menor complejidad inicial.
- Ideal para proyectos pequeños o MVPs.

⚙ Arquitectura Monolítica

Desventajas:

- Difícil de escalar por partes (escala todo o nada).
- El código se puede volver muy complejo con el tiempo.
- Un error puede afectar toda la app.
- Despliegues más lentos y riesgosos.

Arquitectura de Microservicios

Características:

- La app se divide en servicios independientes.
- Cada microservicio tiene su propia lógica, base de datos, y puede estar escrito en otro lenguaje.
- Se comunican mediante APIs (REST, gRPC, etc.)

Arquitectura de Microservicios

Ventajas:

- Escalabilidad independiente: puedes escalar solo el servicio que lo necesita.
- Desarrollo por equipos separados.
- Mejor resiliencia: si un microservicio falla, los demás siguen funcionando.
- Despliegues más ágiles y seguros.

Desventajas:

- Mayor complejidad en infraestructura y despliegue.
- Requiere manejo de comunicación entre servicios (latencia, fallos).
- Monitoreo y debugging más complicado.
- Puede haber redundancia de datos.

Ejemplo: Una Tienda Online

Ejemplo: Una Tienda Online

Imaginemos una tienda online tipo Amazon, con funcionalidades como:

- Registro e inicio de sesión
- Catálogo de productos
- Carrito de compras
- Procesamiento de pagos
- Gestión de pedidos

⚙ En una Arquitectura Monolítica

Todo esto está dentro de una sola aplicación. Podría ser, por ejemplo, una app en Java Spring o en Node.js, así:

TiendaOnline /

- > AuthController
- > ProductController
- > CartController
- > OrderController
- > PaymentService
- > Database / Users, Products, Orders, Payments...

⚙ En una Arquitectura Monolítica

Toda la lógica y datos están juntos en un solo código, una sola base de datos y se despliega como una unidad.

Ventajas

- Fácil de arrancar.
- Menos configuración.

Problemas

- Si algo se rompe en pagos, puede afectar toda la tienda.
- Si hay mucho tráfico en productos, hay que escalar toda la app.
- Los cambios requieren despliegue de toda la aplicación.

En una Arquitectura de Microservicios

Separás la tienda en distintos servicios independientes:

Auth Service → Maneja usuarios y autenticación

Product Service → Catálogo de productos

Cart Service → Carritos de compra

Order Service → Procesa pedidos

Payment Service → Maneja pagos

Frontend → Llama a todos los servicios vía API

En una Arquitectura de Microservicios

Cada uno tiene:

- Su propio repositorio.
- Su propia base de datos.
- Su propio despliegue (ej: contenedores Docker, Kubernetes).

En una Arquitectura de Microservicios

Ventajas

- Si solo querés cambiar la lógica de pagos, actualizás solo ese servicio.
- Podés escalar solo el Product Service si hay muchas visitas al catálogo.
- Si el Cart Service falla, no se cae toda la app.

Problemas

- Más complicado de montar: necesitas herramientas como API Gateway, service discovery, monitoring, etc.
- Debugging y trazabilidad entre servicios puede ser más difícil.