# **Microservices: Theory and Application**

Los patrones de diseño se basan en la comunicación. Para aplicar un patrón de diseño es necesario comunicar y decidir en conjunto con el equipo de desarrollo qué patrón utilizar. Esto también es parte de una comunicación a alto nivel dentro del equipo para acordar qué hacer y qué diseño hacer para cumplir con los objetivos del proyecto. Al estar todos informados, la tasa de éxito se incrementa. Asimismo, se puede detectar errores de forma más ágil.

#### **Sistemas monolíticos**

Sistemas o aplicaciones grandes que realizan muchas funciones con difícil mantenimiento. Con el transcurrir del tiempo, el costo de mantener un sistema monolítico excede sus beneficios y funcionalidades.

#### **Micro servicios**

Separa las funciones de negocio en contextos delimitados por el servicio.

Es amigable para migraciones graduales. Sin embargo, la migración de sistemas monolíticos a micro servicios toma tiempo.

Agnóstico a la tecnología. Elimina la restricción de exclusividad de un lenguaje presente en un sistema monolítico. Pues te permite agregar funcionalidades desarrolladas en lenguajes diferentes según los objetivos del negocio.

Es lo que debería tener una arquitectura de servicios.

Los principios más importantes son los primeros cuatro:

Encapsulamiento: Es necesario aislar los servicios en contextos delimitados.

Dominio céntrico: El encapsulamiento de cada servicio tiene que estar enfocado en un problema específico a solucionar, es decir, en un requerimiento.

Independiente: no debe depender de otro micro servicio.

A prueba de fallos y escalables.

Los últimos tres son más importantes para devOps: Automatización, Descentralizado, Observable.

El principal impulso de los microservicios es ofrecer escalabilidad. Se puede ilustrar como un cubo de 3 ejes:

Eje X: Implica la descomposición funcional del sistema o aplicación.

Eje Y: Consiste en la redundancia de servidores, clusters, redes, etc..

Eje Z: Muestra la fragmentación de los datos en varios en nodos.

Los principios de desarrollo SOLID y DRY se alinean con las bases para microservicios. Principalmente, SOLID; en cambio, DRY no se cumple al 100% porque algunas partes de los microservicios tienen duplicidad.

Por otra parte, una arquitectura de microservicios facilita la mantenibilidad del código. Una característica no funcional estimada por encima de la optimización en un ambiente empresarial.

Dado que, existen distintas peticiones de usuario, es necesario un API Gateway que las distribuya según sus necesidades a un microservicio compuesto. Es decir, los microservicios pueden estar compuestos por otros microservicios.

Las definiciones de éxito de un sistema o aplicación son las siguientes:

* El software hace lo que debería hacer
* Se cumplen las necesidad de los usuarios
* A tiempo
* Escalable. Puede soportar gran cantidad de usuarios
* Es tolerante a fallos
* Directo para realizar cambios
* Fácil de gestionar y lanzar
* Dentro del presupuesto
* Es seguro

Satisfacer estas definiciones implica comprender el negocio. Asimismo, tener una estructura organizativa dentro del equipo, También, definir el tamaño del código, del equipo, de recursos, etc. Además, es necesario definir la tecnología, en alineamiento con los objetivos.

Comprender el negocio: Consiste en dividir los problemas en partes más fáciles de gestionar. Se aplica el ciclo de vida software con la ayuda de diferentes metodologías ágiles. Se evalúa el sistema de acuerdo a usabilidad e interacción por parte del usuario.

Tener un estructura organizativa: Implica que los servicios pertenecen a un equipo y no por varios equipos. La simplicidad es la clave para tener éxito, por consiguiente, el uso de patrones ayudan a lograr el éxito.

#### **Beneficios**

Rendimiento: Reducción del time to market. El tiempo para elaborar y lanzar un producto al mercado se reduce.

Satisfacer las expectativas más altas de los usuarios: A través de la arquitectura como la escalabilidad, robustez, etc.

Más rápido de desplegar, Más fácil de probar, Más barato de escalar, Mejoramiento del aislamiento de fallos.

#### **Retos**

* Mayor complejidad en sistemas distribuidos
* Sistema de pruebas por la interdependencia los servicios.
* Transacciones distribuidas
* Gestión del sistema
* Se requiere mayor memoria
* Organización y cultura
* Madurez general

### **Tecnologías para aplicar a microservicios.**

* CQRS (Command Query Responsibility Segregation): El sistema se subdivide en 2 subsistemas: el de command, operaciones de escritura; y el de query, consultas.
* Domain driven design: Diseño guiado por el dominio. Se delimita el contexto del microservicio a su dominio.
* Aprovisionamiento de eventos: Se almacena las acciones realizadas sobre el sistema de forma que garantice la trazabilidad y la contabilidad. Además, volver a recuperar un estado del sistema anterior.