Micro services: Theory and Application

Los micros servicios son importantes simplemente porque agregan un valor único en una forma de simplificación de la complejidad en los sistemas. Al dividir el sistema o aplicación en partes más pequeñas, aumentar la cohesión y disminuir el acoplamiento entre las partes, hace que las partes generales del sistema sean más fáciles de comprender, más escalables y más fáciles de cambiar. La desventaja de un sistema distribuido es que siempre es más complejo desde el punto de vista de un sistema.

La arquitectura en el contexto de software toma el concepto de patrones de diseño para dejar establecida una manera de resolver problemas (comunicar soluciones). Sirve para estandarizar y establecer metas más rápido.

La estandarización y los objetivos son parte de la arquitectura de software. Uno debe estar seguro en lo que está haciendo y el diseño que está haciendo para cumplir el objetivo que puede ser negativo o positivo pero hará que el su probabilidad de éxito sea mayor

Monolítica hace referencia al libro de programación UNIX, Los sistemas monolíticos son generalmente antiguos y difíciles de mantener.

Los micros servicios te ayudan a dividir tu negocio en pequeños contextos con límites claros y resolver los problemas de manera localizada, te permite migrar a ellos gradualmente y permite ser independiente en cuanto a la tecnología que se usa (no necesitas tener, por ejemplo, todo en java. Puede haber una parte en java, otra en Python, etc.)

Principios

Encapsulación

Automatización

Domain Centric → centrado en el problema principal

Descentralizado → no tiene que estar “en un solo sitio” la nube, on premise, etc.

Independiente

Fail – Safe (a prueba de fallos)

Observable (monitoreable)

Patrones y enfoques principales: CQRS, Event Sourcing, API gateway/proxy y Orchestrated API.

Approach to Success

1.2 Se debe entender el negocio y tener en claro el tamaño de la solución.

No se debería tener defectos (hacer lo que debería), cumplir las necesidades del usuario, ser segura, ser escalable, robusta, fácil de administrar y desplegar, amigable al cambio, dentro del presupuesto y entregada a tiempo. Se debe mantener una estructura organizacional, cada servicio le pertenece a un equipo. Se deben usar los patrones DRY: Don't repeat yourself y SOLID El equipo debe ser de preferencia pequeño y confiable.

Tecnología: Se debe escoger de acuerdo al problema Escoger de preferencia lenguajes basados en eventos

Estrategia de particionamiento: Single responsibility Principle (solo hacer una cosa)

Beneficios: Desempeño alto, suele cumplir las expectativas altas, el lenguaje de patrones "está en todos lados". Despliegues rápidos, fácil de probar, escalabilidad barata y aislamiento de fallas.

Desafíos: Muy complejo en sistemas distribuidos, difícil probarlo como parte de un "sistema" (al estar muy aislado probarlo en conjunto con los demás elementos puede ser difícil). Independizar sistemas si estabas antes en uno monolítico puede ser difícil. También es complicado encajar esta filosofía con la cultura organizacional tradicional.

2 CQRS: Separa los Reads de los Writes, es un buen punto de partida para entender como dividir los problemas

Event sourcing: confiabilidad y transparencia, se guardan los eventos que se a dado a lo largo del tiempo desde la salida del sistema, uno de los mejores beneficios es la capacidad de poder recrear los eventos pasados (replayability). Sirve mucho para hacer auditorias. El problema es que ocupa mucho espacio, pues se salva el evento por el cual paso el objeto en sí, no el estado del objeto como tal. Se usa el patrón de publicador y subscriptor.

API GAteway: este patrón extraes el consumidor del micro servicio y así es más fácil llamar al micro servicio