**Microservices: Theory and Application**

**Teoría**

Introducción a la arquitectura

Los patrones de diseño se basan en la comunicación. Para aplicar un patrón de diseño es necesario que haya una comunicación de alto nivel dentro del equipo para tener en claro qué hacer y cómo diseñarlo para así cumplir con los objetivos del proyecto. Mientras haya una comunicación de alto nivel, las probabilidades de éxito se incrementan.

Sistemas monolíticos

Sistemas o aplicaciones grandes que realizan varias funciones y son difíciles de mantener.

Mientras más tiempo transcurra con el uso de sistemas monolíticos, mayor es el costo de mantenimiento, y esto ocasiona que estos costos excedan a los beneficios que brinda.

Microservicios

Los microservicios tienen las siguientes características:

-Separa las funciones del negocio en contextos limitados, un contexto delimitado por servicio.

-Amigable a las migraciones graduales

Pero las migraciones de sistemas monolíticos a microservicios toma tiempo.

-Agnóstico a la tecnología

A diferencia del sistema monolítico el cual solo permite el uso de un solo lenguaje, los microservicios eliminan esa restricción y permiten el uso de funcionalidades en varios lenguajes diferentes según los objetivos que tenga el negocio.

-Es lo que SOA (arquitectura de servicios) debió tener.

Principios de microservicios

1. Encapsulamiento: Separar los servicios en contextos delimitados

2. Dominio céntrico: Cada servicio debe estar enfocado en un requerimiento

3. Independiente: Cada servicio debe ser independiente

4. A prueba de fallos: Los servicios deben ser tolerantes a fallos.

Principios de microservicios más orientados a DevOps

5. Automatización

6. Descentralizado

7. Observable

El objetivo principal del uso de los microservicios es para obtener escalabilidad y se puede ilustrar como un cubo que tiene tres ejes:

Eje X: Duplicación horizontal

Eje Y: Descomposición funcional, se obtiene la escalabilidad mediante la separación de componentes diferentes.

Eje Z: Partición de data, se obtiene la escalabilidad mediante la separación de componentes similares.

Cambio en el mantenimiento

Mientras la industria evoluciona, la mantenibilidad es un factor clave en los sistemas de software, aun así, no se puede dejar de lado el rendimiento, el cual también es un factor importante.

Los principios de desarrollo SOLID y DRY se alinean con las bases para los microservicios. Principalmente SOLID, debido a que la duplicación es un efecto secundario en los microservicios y esto hace que el principio DRY no se cumpla del todo.

Algunos enfoques de los microservicios que nos permiten entender la importancia de la mantenibilidad son el CQRS, el cual es en esencia un patrón de enseñanza que muestra que no tienes que hacer las cosas de una sola forma, el abastecimiento de eventos, el cual es importante porque varios sistemas funcionan mediante este enfoque y encaja en una gran variedad de sistemas, el API Gateway / Proxy, el cual permite abstraer algunas complicaciones en seguridad o registro y el API orquestado.

Aproximación al éxito

Definiciones del éxito para un sistema o aplicación:

-No hay errores, es decir, el software hace lo que debería hacer.

-Se cumplen los requerimientos de los usuarios

-Es seguro

-Escalable, el software está diseñado para soportar una gran cantidad de usuarios.

-Robustez, el software puede soportar fallos o cambios inesperados

-Fácil de gestionar y lanzar.

-Abierto a realizar cambios.

-Dentro del presupuesto.

-Se cumple con los tiempos.

Cumplir con estas definiciones conlleva a un correcto entendimiento del negocio, tener organizado estructuralmente al equipo, conocer el tamaño del código, del equipo, saber lo que realmente importa en el negocio y definir si las tecnologías empleadas realmente van acorde a los objetivos trazados.

Entendimiento del negocio

Consiste en dividir problemas grandes en partes manejables. Aplicación del ciclo de vida del software con el proceso, historias, casos de uso, escenarios y la evaluación del sistema respecto a la usabilidad e interacción que se tenga con los usuarios.

Estructura organizacional

Los servicios le pertenecen a un equipo y no a varios servicios. La simplicidad es la clave del éxito, el uso de patrones DRY y SOLID también ayuda.

Beneficios / Retos

Beneficios

-Rendimiento en ejecución, desarrollo y tiempo que toma llevarlo al mercado.

-Satisfacer las necesidades de los usuarios mediante la arquitectura, debido a que es escalable y robusta.

-Desarrollo rápido

-Fácil de probar

-Costo bajo para la escalabilidad

-Mejora el aislamiento de los fallos

Retos

-Más complejidad en los sistemas distribuidos

-Sistema de pruebas para cada servicio

-Transacciones distribuidas

-Gestión del sistema

-Requiere mayor memoria (menos los contenedores)

-Organización y cultura, debe haber una relación entre ambas para que así funcionen los microservicios en el negocio.

-Madurez

**Aplicaciones**

Tecnologías

CQRS(Command Query Responsability Segregation)

Separa lectura de escritura, el sistema tiene dos subsistemas, query(lectura) y command(escritura, updates).

Aprovisionamiento de eventos

Base para diseños de negocio que necesiten confiabilidad y transparencia (contabilidad, banca, blockchain). Además, permite volver a recuperar una versión anterior del sistema.

Tiene las siguientes características: Es nuevo y no convencional, confiable, consistente, agnóstico a la base de datos a usar y da “problemas” de implementación debido a que se debe decidir cómo se quiere implementar esto en los servicios los cuales son independientes.

Domain Driven Design

El diseño es complicado, requiere el conocimiento del negocio, los límites del negocio y el aislamiento de esos límites.

Cada contexto tiene un lenguaje ubicuo bien específico.

API Gateway

Resuelve los siguientes retos: Granularidad muy fina, distintos clientes requieren distinta data, el número de ubicaciones de las instancias de los servicios cambian dinámicamente y la partición de los servicios está oculta a los clientes.