Pregunta 1: Microservices: Theory and Application.

Teórico:

Introducción a la Arquitectura de Microservicios:

La base principal para implementar patrones de diseño es la fuerte relación que se tiene con la comunicación, cuando se decide aplicar un determinado patrón es necesario establecer una coordinación con el equipo de trabajo teniendo una conversación de alto nivel para asegurar las probabilidades de éxito del negocio y ofrecerle al cliente un software mantenible, seguro y ágil para resolver los problemas.

Arquitecturas Monolíticas:

Representan un sistema o aplicación extensa que realiza muchas funciones pero que a su vez su mantenimiento es complicado puesto que cuando empieza a crecer exponencialmente su desarrollo se vuelve trabajoso no siendo recomendado para los negocios. A consecuencia de esto la migración desde la arquitectura Monolítica a una de Microservicios se ha vuelto más fuerte en los últimos años.

Arquitectura de Microservicios:

Consiste en separar las funciones del negocio en servicios pequeños que a su vez están delimitados por contextos y se ejecutan dentro de su mismo proceso.

Tiene la característica de ser amigable cuando se desea realizar migraciones graduales a pesar de que su proceso es lento, agnóstico a la tecnología puesto que permite agregar características de otros lenguajes y no emplear solo un lenguaje exclusivo, y además representa lo que un SOA debería tener. Como toda arquitectura los Microservicios están basados en principios, se encuentran divididos en 2 grupos de importancia.

Más importantes:

- Encapsulamiento: es elemental para que los servicios se puedan contener y delimitar.
- Dominio céntrico: comparte relación con el encapsulamiento puesto que debe estar lo más enfocado posible en el problema.
- Sistema a prueba de fallos e independencia.

Importantes para DevOps (Unificación de Desarrollo de Software y su Operación):

- Sea descentralizado.
- Automatizable y observable.

Escalabilidad: al delimitar los procesos, estos se pueden escalar de forma independiente ahorrando costos significativos, esto es representado gráficamente en un cubo con sus 3 ejes correspondientes (el eje ‘x’ representa la duplicación funcional del sistema, el ‘y’ el factor de redundancia de clústers o servidores y el eje ‘z’ la partición de la data del sistema o aplicación).

Mantenibilidad: se considera clave hoy en día en los microservicios a medida que la industria evoluciona, puesto que el código al ser mantenible representa más ingresos para el negocio, los principios SOLID también tienen un papel destacado en este tipo de arquitectura ya que se alinean junto con los cimientos de los microservicios desde sus origines mientras que DRY no es aplicable en muchos de los casos por presentar duplicidad en los servicios.

Acercamiento al éxito:

Para el logro del éxito es necesario entender cómo funciona el negocio, la forma de la estructura de la organización, determinar que las tecnologías empleadas cubran las metas propuestas y cumplir con las necesidades de los clientes.

A continuación se muestran diversas definiciones del éxito:

- Seguro
- El software "hace lo que deba hacer"
- Escalable
- Estar dentro del presupuesto
- Robusto: soportar fallas o cambios inesperados
- Facilidad de manejo
- A tiempo

El proceso para entender el negocio es dividir los problemas en partes que se puedan manejar, entender el ciclo de vida, manejo de casos de uso, historias de usuario, entre otras.

Estructura organizacional:

En la parte estructural de una organización los servicios deben pertenecer solo a un equipo reducido de personas, en vez de muchos equipos por separado, esto beneficia al rendimiento del software que se quiere diseñar, para lograr medir el tamaño del equipo responsable es necesario considerar algunos factores:

- Alta confianza entre los integrantes.
- "Two Pizza Sized Teams": se refiere a que 2 pizzas deberían alimentar como máximo a 10 personas del equipo.
- Dependencia del mismo equipo para lograr efectividad (evitar cuellos de botella cuando se tiene a muchas personas dentro de una misma tarea, lo mejor es simplificarlo para que el proceso sea fluido).

Beneficios / Retos:

Beneficios:

- Rendimiento: al ejecutarse y gestionarse
- Satisfacer expectativas del usuario: robustez y escalabilidad
- Lenguaje ubicuo: presente en todas las partes del sistema o aplicación
- Fácil de probar
- Aislamiento de fallos mejorado
- Barato de escalar
- Rápido despliegue

Retos:

- Mayor complejidad en sistemas distribuidos
- Sistema de pruebas: independencia
- Transacciones distribuidas
- Necesidad de mayor memoria
- Madurez
- Organización y cultura
- Gestión del sistema

Aplicaciones:

Tecnologías:

- CQRS (Command Query Responsability Segregation): se separa en 2 sub-sistemas, el primero *command* que son las peticiones (operaciones de escritura) y el *query* que representa las consultas.
- Abastecimiento de eventos: en lugar de eliminar algún error de una transacción del sistema, se realiza una recuperación del estado anterior garantizando la contabilidad.
- DDD (Domain Driven Design – Diseño guiado por el Dominio): conocido el negocio debe aislarse para facilitar el diseño del mismo.
- API Gateway: soluciona el nivel de detalle o granularidad muy fina, oculta la partición de servicios de los clientes.
- Lenguajes para problemas de espacio: NoSQL-Inmutable-Estadístico.