**Pregunta 1: *Microservices: Theory and Application***

1. **Teoría**
   1. **Definiendo la arquitectura**

En el mundo real, los clientes son reacios a usar otros patrones que no sean los que usan en su negocio. Siempre buscarán estandarizar la forma en la que trabajan y el diseño que siguen. Y puede que estén en lo correcto o en un error, pero lo que se busca es un alto nivel de conversación. De esta manera las chances de llegar al éxito en sus objetivos son mayores o al menos se reduce el tiempo en el que se dan cuenta de que hay un error y que no se alcanzarán estos objetivos.

**Monolíticos: El problema**

Hace referencia a lo que puede llegar a convertirse un programa que se viene desarrollando por mucho tiempo: un gran sistema o aplicación que hace demasiadas cosas y deja de ser mantenible, a veces al punto de que su costo de mantenimiento supera su beneficio. También se caracteriza por ser “doloroso”, pues, por ejemplo, causa mucha demora en la realización de requerimientos.

**Microservicios: La solución**

Siguiendo el enfoque de los microservicios, lo que se haría es separar las distintas funciones o partes del sistema en varios contextos encapsulados. Cada uno de esos contextos representará un servicio. Una característica de este enfoque es que es amigable cuando se trata de migraciones graduales (de funciones del sistema hacia otros contextos, como se mencionó antes)

Los microservicios siguen una serie de principios como: encapsulación, centralizado en dominio, independiente y aprueba de fallas.

La escalabilidad es la característica principal de la arquitectura de microservicios. Esta puede ser representada mediante un cubo; en el cual el eje Y es la dimensión en la cual se descomponen las diferentes funciones del sistema; el eje Z es la dimensión en la cual se particiona la data similar del sistema; y finalmente el eje X sirve para todo lo que llegue a duplicarse.

Asimismo se sigue una filosofía en esta arquitectura, la cual plantea que cada microservicio debe hacer una sola cosa y la debe hacer bien. Esta simple cláusula también es el núcleo de otros sistemas y herramientas que actualmente existen. Además se estipula que en los microservicios las cosas se deben dividir finamente en servicios enfocados.

En las últimas décadas las empresas han cambiado su enfoque hacia mantenibilidad (del software). Esto se debe a que el software no mantenible debe ser desarrollado de nuevo, lo cual significa mucho más costo para la empresa (en especial porque actualmente el costo de mano de obra se ha disparado por los cielos, a diferencia del de hardware que ha ido decayendo).

* 1. **Camino al éxito**

Tener éxito significa entender el negocio (ser capaz de dividir los problemas en partes más gestionables, tener en claro cuál es el ciclo de vida de software que se maneja, saber cuál es la usabilidad del software y el nivel de interacción con el usuario que se maneja y estar familiarizado con el modelo de negocio que representan las APIs); tener y entender la estructura organizacional del equipo (el cual debe encargarse de todos los servicios sin necesidad de otros equipos); saber que significa el tamaño en diferentes partes de la empresa como el código o el equipo; y asegurarse que la tecnología esté acorde con los objetivos.

* 1. **Beneficios / Retos**

Los beneficios que trae consigo el enfoque de microservicios son en cuanto al rendimiento (en ejecución y desarrollo), satisfacer grandes expectativas de los usuarios a través de la arquitectura (robusta y escalable) y lenguaje ubicuo en todos los microservicios. Otros beneficios son un despliegue más rápido, la facilidad de hacer pruebas, más ahorro en cuanto al escalamiento y un mejor aislamiento de fallos.

Finalmente esta arquitectura trae ciertos retos como una mayor complejidad en sistemas distribuidos, la realización de pruebas en sistemas, transacciones distribuidas (usualmente solucionado con consistencia eventual), gestión del sistema (generalmente abordado con el manejo de herramientas), y mayor memoria. Otros retos son en cuanto a la organización y la cultura, y la madurez.

1. **Aplicación**
   1. **Tecnologías**

Las tecnologías que se aplican en la arquitectura de los microservicios son:

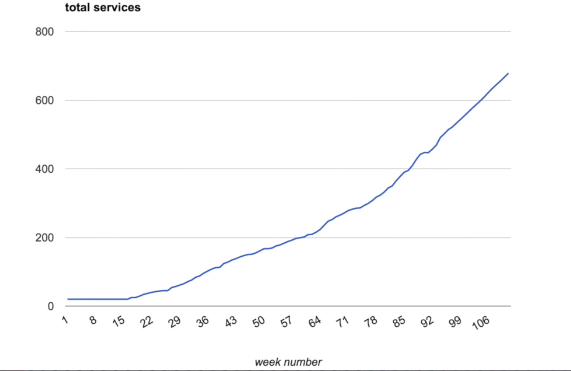
* ***CQRS***: Nos habla de separar las operaciones de lectura y escritura, es un buen punto de partida.
* ***Event Sourcing***: Base para diseños de negocio que necesitan confiabilidad y transparencia. Es nuevo y poco convencional, confiable, tiene consistencia eventual, agnóstico con bases de datos, tiene algunos problemas en la implementación.
* ***DDD* / Contextos encapsulados**: El diseño es difícil, se debe conocer el negocio, dividirlo en varios contextos completamente encapsulados o aislados. Cada contexto tiene su propio lenguaje ubicuo que es muy específico.
* ***API Gateway***: Soluciona ciertos retos como: divisiones demasiado finas, clientes distintos necesitan data distinta, los números y ubicaciones de las instancias de servicio cambian dinámicamente, y el particionamiento de servicios debería estar oculto para los clientes.

**Pregunta 2: *What I Wish I had known Before Scaling Uber to 1000 Services***

En esta charla tratan acerca de los problemas que surgen debido al crecimiento de microservicios en Uber, así como la manera en que afrontaron algunos de estos. Más que una presentación de teoría, se exponen experiencias y consejos que todos deberían tomar en cuenta. A continuación se describen algunos de los tópicos mencionados por Matt Raney.

1. **Microservicios**

Como se mencionó anteriormente, Uber ha sufrido un gran crecimiento de microservicios a lo largo de los años. Una pequeña muestra presentada fue que al inicio del año 2016 tenían un aproximado de 100 microservicios y para el medio año siguiente se habían incrementado en 200. Asimismo, se mencionó que la aplicación es mucho más confiable los fines de semana cuando no se realizan cambios en los microservicios. Esto, refleja la manera de trabajar de esta compañía. Usar microservicios permite a diversos grupos de trabajadores laborar de forma independiente a través de un sistema distribuido.



Fuente: https://www.youtube.com/watch?v=kb-m2fasdDY

1. **Costos y beneficios**

Sin embargo, no todo es maravilloso al utilizar microservicios. Matt Raney señala varios problemas relacionados a ellos. Para empezar, menciona que debido a la cantidad de grupos trabajando en la empresa de una forma independiente muchas veces no existe una actitud de colaboración. Es decir, estas personas prefieren desarrollar software de forma autónoma en vez de conversar y llegar a un acuerdo que pueda beneficiar a los implicados. Una de las posibles causas podría ser la diversidad de lenguajes que maneja la aplicación y que genera debates acerca de cuál es considerada mejor. Pero este no es el único problema relacionado a lenguajes de programación, también menciona lo difícil que es para una persona adaptarse a otro grupo con distintos factores.

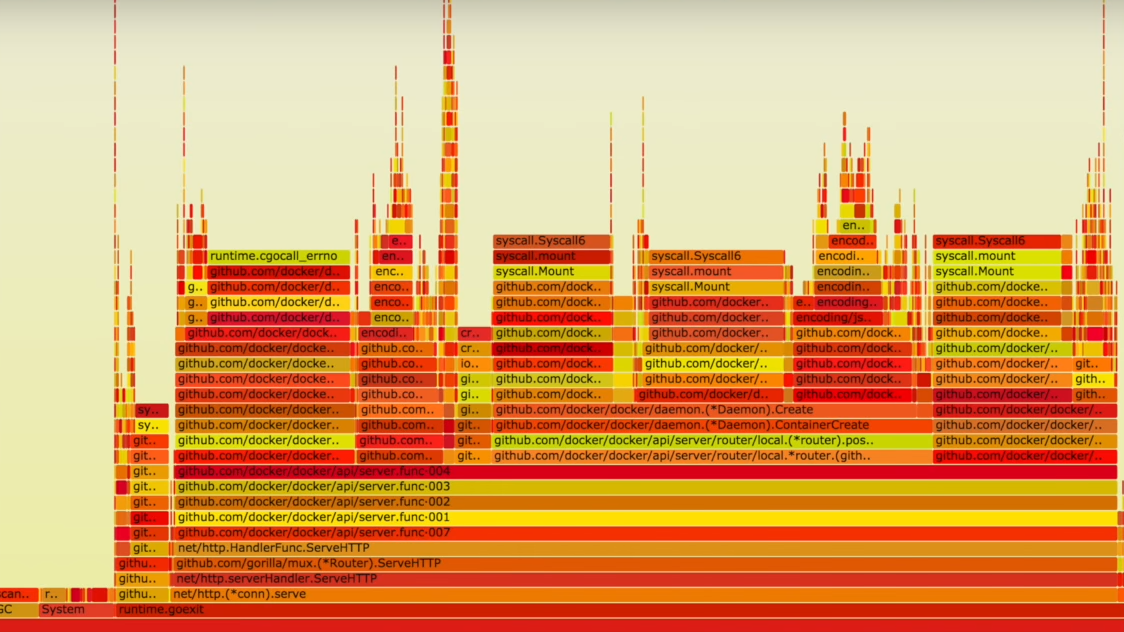
Otra problemática está relacionada con la complejidad de HTTP/Rest. Se tomó por ejemplo la semántica de HTTP, ¿para qué sirven las cabeceras? , ¿cuál es el estatus del código? Además, se resaltó que en equipos grandes usar estos protocolos ocasionaban gastos a la empresa. Lo anterior ocurre, debido a que hay mayor desorganización en este tipo de equipos. Matt Raney menciona también que JSON, a pesar de ser útil, si no tiene un esquema también genera un costo.

En relación al uso de repositorios mencionó a dos tipos de categorías. Las empresas que manejan muchos de ellos y las que solo manejan uno para sus proyectos. Ambas tienen tanto sus ventajas como sus desventajas. Para el primero de ellos, permite una mayor independencia en los módulos, pero resulta muy difícil navegar por el código. Por otro lado, en las empresas que se manejan un solo repositorio es mucho más fácil navegar por el código, así como realizar cambios transversales. Sin embargo, este solo puede funcionar con herramientas especializadas.

Otro tema presentado fue la coordinación que debe haber entre los grupos cuando ocurre un fallo en uno de los microservicios. Si bien se puede trabajar con fallos en el software, muchas veces estos perjudican a otros grupos por lo que se genera como un estado de bloqueo hasta que otros equipos realicen cambios. Esto refleja que la independencia no se aplica a un 100%.

En relación al rendimiento, Matt expone la forma que ellos utilizan para ordenarse. Desde el uso de gráficos de llamas, hasta el uso de dashboards estandarizados, se enfoca en que el problema no radica en las herramientas que se utilizan, sino en las personas. No existe un interés en el rendimiento ni una cultura de optimización. Para ello, una forma de afrontar estos problemas fue la aplicación de una lista de requerimientos básicos, como el tiempo de respuesta, para todos aquellos desarrolladores. De esa forma sabrían que al menos los elementos de esa lista eran respetados. Matt menciona que otra manera que ellos utilizan para controlar el rendimiento. Esta es el rastreo a través de un software llamado Zipkin.

Gráfico de llama



Fuente: https://www.youtube.com/watch?v=kb-m2fasdDY

Con respecto a las pruebas Matt llegó a la conclusión que era importante realizarlas preferentemente todo el tiempo aún si van en contra del código de producción. Sin embargo, habían personas que se resistían a algunos tipos de prueba como la de fallos. Esto, se debía a que muchas veces era necesario eliminar partes de un código con el cual una persona se encontraba muy apegada.

Finalmente, un último tópico que menciona son los cambios. Es muy difícil realizar modificaciones a códigos antiguos, puesto que las personas suelen ser temerosas a dañarlas. Esto, ocasionaba que se les tuviera que forzar a actualizar todas aquellas partes desfasadas. Matt reflexiona en lo importante que es hacer ver a las personas lo valioso de innovar en, por ejemplo, la arquitectura y/o infraestructura, más que hacerles sentir como si fuese una obligación. Asimismo, relacionado al tema de cambios con open source, había un sentimiento de tristeza. Esto pues muchas personas a veces trabajaban en una plataforma de bajo nivel y, debido a la competencia, sentían que lo que ofrecían no era lo mejor.

1. **Conclusión general**

Al manejar gran cantidad de microservicios es necesario siempre llevar una organización en toda la empresa. Es muy importante el diálogo en equipos para que en vez de trabajar por separado, puedan generar un mayor beneficio colaborando entre ellos. Además, se necesita llevar un registro de rendimiento para poder ofrecer una mejor experiencia al cliente, así como también para poder realizar cambios que mejoren al código.

**Pregunta 3**

Ver archivos adjuntos en el repositorio.