**Pregunta 2: Presentación de Matt Ranney de cómo Uber maneja los micro servicios**

Matt Ranney, cofundador y Jefe del diseño de arquitectura de Uber, empresa encargada del servicio de taxis, expuso sobre cómo Uber maneja los micro servicios y sobre qué le hubiese gustado tener como conocimiento previo de cómo funciona todo esto.

El escalamiento de Uber es sorprendente, en algunos años ha tenido hasta crecimiento de x10 por cada uno. Actualmente utilizan crecimientos, donde se rompe el esquema de una sola gran aplicación (Monolito) hacia pequeñas APIs.

El expositor empieza a mencionar características y ventajas del desarrollo de aplicaciones en micro servicios.

* Una ventaja de esta arquitectura es que si rompes algo no rompes todo, eliminando el riesgo de malograr todo el programa al intentar de cambiar algo.
* Puedes crearlos, moverlos, modificarlos y desplegarlos de manera independiente.
* Uno es dueño del uptime.
* Decides que cosa es mejor en cada contexto.
* Puedes construir según los problemas que se te presentan y cambiar complejidad por políticas o consensos.

Originalmente Uber trabajó en su primera app con PHP, y actualmente trabaja con micro servicios, a diferencia del pasado donde trabajó con una aplicación monolítica.

* Dispatch: Node.JS, migrando a Go
* Core Services: Python, migrando a Go
* Maps: Python & Java
* Data: Python & Java
* Métricas: Go

Problemas que se tiene al trabajar con micro servicios: Cuando las cosas se convierten en RPCs (Remote Procedure Call), el cual es un protocolo que hace que un programa pida el servicio de otro sin entender los detalles de la red:

* Equipos se comunican entre sí
* Problemas de interpretación de métodos
* Muchos repositorios:
  + - Mucho o uno es bueno
    - Mucho o uno es malo
  + Todo esto depende del caso que se de.
* Usas las cosas sin saber como son, como sucede con los repositorios de google.

La performance de la aplicación depende del código, hay gente que no le importa y prefieren comprar nuevos servidores a tener que contratar ingenieros. Esto es una mala práctica, debido a que no importa hasta que afecte, y cuando afecte el costo de la optimización será muy elevado.

La latencia debe ser mínima, y como ha sucedido en Uber el 1% de usuarios tendrían problemas de esta, afectando con la velocidad del servicio y esto es considerado mucho.

Logging o registro: Al tener distintos equipos y lenguajes varios intentan mantener registro de diversas e incompatibles formas. Cuando generas un registro de manera muy seguida mucho generas problemas por causar presión en el log. Menciona que en el caso de su empresa hubiese sido bueno mantener una contabilidad que organice todo desde un inicio.. El tamaño de la data que la gente mantiene registro es gigantesco pero gratuito. Ellos actualmente usan ZAP que es open-source.

Load testing: Se buscan hacen antes de producción. Ellos hicieron el load testing en contra de la producción durante tiempos que no son cruciales, pero generó problemas, hizo que la gente piense que tienen más tráfico de lo normal. /A veces no hay forma de crear un entorno simulado con tráfico simulado/ . Hubiera sido ideal haber contado con un sistema que controle este tráfico simulado desde antes.

Failure testing: Es sorprendente que no a todos le gusta y la gente suele odiarlo al tener que implementarlo después. Escribiste el servicio, lo pones en producción pero pueden sentir que lo estas apagando.

Migraciones: De legacy hacia nuevas tecnologías. Se busca hacer la migración de forma constante de cosas legacy a cosas no tan legacy. Que paso con estos micro servicios inmutables, abandonas las cosas mientras que sirven trabajando. Tienes que aprender que “debes adoptar nuevos sistemas”, las cosas cambian.

Open Source:

* Build/sacrificio es difícil
* Todo que es de infraestructura y parte de la plataforma tiene sentido que lo desarrolles por tu propia cuenta.

Políticas: Al dividir los servicios en pequeñas partes se tienen que aplicar. Suceden cuando violas una propiedad, para evitar poner tus decisiones sobre la de la empresa. Cosas que no me gustan. Es útil y se debe de considerar porque para implementar desarrollo modular rápido existe tentación de no cumplir estándares, rompiendo con lo que t comprometes con la empresa.