# **GOTO 2016 • What I Wish I Had Known Before Scaling Uber to 1000 Services • Matt Ranney**

La entrevista trata sobre recomendaciones que le hubiese gustado obtener durante el proceso (antes y durante) de cómo escaló y se posicionó Uber.

Por el crecimiento que tuvieron en número de servicios, utilizan microservicios que evitan tener sistemas monolíticos. debido a algunas eventualidades por el uso e incremento de los microservicios, se muestra la agilidad e independencia.

Por el contrario, los costos aumentan, sin intención alguna, debido a la resolución rápida de problemas, utilizar y conservar métodos y herramientas para el trabajo de resolver problemas y para futuro. Los problemas en si son :

Lenguaje: debido a que cada codigo esta en diferente lenguaje, en diferente interpretación, no existe la flexibilidad al utilizarse en diferentes dispositivos. esto hace también que los desarrolladores se agrupen de acuerdo al lenguaje compartido que manejan.

RPC: indica que todo se convierte en RPC, provocando que las http rest se vuelven complicados, teniendo problemas de interpretación. modificando y malentendiendo los archivos y la data en formato JSON. la programación web cambia sus funcionalidades de servicios de browser, que será mejor tratarlas como llamadas funcionales a usar los request.

Repositorios: Existe una gran cantidad de repositorios, en vez de esto sería utilizar un repo múltiple o un único repo. Ambos cuentan con gran beneficio, al ser múltiple se puede tener diferente pequeños módulos que actualizar y gestionar rápidamente. y con un gran repo, lo difícil será revisarlo y encontrar posible errores y mejoras de ciertos puntos, debido a que seria un único gran sistema. por ejemplo se dice que uber cuenta miles de repositorios.

Operaciones:

cada equipo tiene su ritmo y uso de recursos al desplegar los microservicios. algunos equipo demoran más o menos y pueden detener o esperar a otro. pero cad auno hace parte al final todos funcionan en conjunto para correr todo el servicio completo.

Performance: debido a los diferentes formatos de perfil que tienen los microservicios hace que sea difícil poder descubrir el verdadero performance de cada servicio. además de las métricas que tienden a ser hechas con diferentes variables. seria bueno un estándar. permitiendo que se genere automáticamente, que tenga menos carga y mejor entendimiento a la hora de revisarlo. Siempre, hay que tener en cuenta el desempeño del performance, el performance no es importante, hasta lo que es. no tiene que ser bueno, solo saber.

Fanout: la espera de los servicios por la latencia, hace un problema que debe contenerse y mantenerse, en este caso con un porcentaje de latencia mínimo. la intención es reducir la espera para conexión. de aquí salen algunas métricas más.

Logging: dado el problema de los diferentes lenguajes, se necesita emplear el logging estructurado, no solo para mejorar el entendimiento, hay que tener alguna manera de contabilizar estos logs, pues algunas personas loguean demasiada data, y el indexado se vuelve un problema mayor.

Load Testing: para poder usarlo, se necesita no salirse de las métricas existentes. paar producción se debe guiar del tráfico de data, por lo que manejar el tráfico es ideal par aun sistema.

Migraciones: ojo, mucho cuidado con los legacy, la migraciones son importante y comunes, es cambiarse porque se necesita cambiar, no por qué aparece una nueva tecnología.

Políticas: al usar microservicios es importante definir las reglas y las políticas, par las tomas de decisiones, que violen propiedad de otros, cuando se dañe algún individuo.

Para finalizar, todo es intercambio, lo ideal es que sea planificado, intencional y no porque no hay de otra. la mejor manera de utilizar y trabajar con microservicios en dando funciones con verdadero valor.