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

La charla trata sobre lecciones aprendidas durante toda la trayectoria de desarrollo de Uber, en especial el continuo escalamiento de sus servicios.

**Things. Change. Quickly.**

Empieza mostrando cómo el cambio sucede de manera rápida en cualquier sistema, tomando como ejemplo algunas ciudades chinas como Beijing. Así como estas ciudades, el equipo de Uber creció 10 veces más a comparación de cuando empezaron.

**Lecciones a lo largo de su trayectoria**

Los microservicios con parte importante de toda tecnología, ya que sirve para adaptarse al rápido cambio en el mundo. Nació como una solución para terminar con los sistemas monolíticos, pero, así como estos, también existen ciertas desventajas. Como, por ejemplo, si estás construyendo muchos servicios, puede haber cierta fragmentación y falta de comunicación. ¿Por qué utilizar microservicios? Matt Ranney opina lo siguiente:

* Ayuda a estructurar y organizar mejor el sistema con el fin de trabajar de acuerdo con cada área/proyecto de este, mejorando de manera independiente.
* Cada integrante/equipo de desarrollo tiene su propio tiempo definido para su desarrollo e implementación a producción.
* Usa la herramienta más adecuada para el trabajo de cada proyecto.

¿Cuáles son sus costos de hacer un despliegue de un sistema grande de microservicios? Los costos más obvios y significativos vendrían a ser los siguientes:

* Ahora se basa en un sistema distribuido, los cuales son más difíciles de trabajar que un sistema monolítico, ya que debes trabajar con diferentes hardware y software.
* Todos los nodos hacen uso de una llamada a procedimiento remoto (*Remote, Procedure Call*, RPC) con el fin de ejecutar código de forma remota sin preocuparse por las comunicaciones entre ambas, ocasionando errores por distintas razones.
* El *troubleshooting* de errores se vuelve más extenso debido a que el análisis es más largo por las distintas partes utilizadas en el servicio.

Por otro lado, los costos menos impactantes y obvios son:

* Obtienes algo a cambio de otra cosa al implementar un sistema así. Por ejemplo, al reemplazar una función que no tiene reparo por un nuevo microservicio o *feature*.
* Puedes terminar construyendo nuevos *features* alrededor de un problema sin resolver.
* Puedes terminar cambiando complejidad por política. Se pierde la comunicación entre integrantes del grupo a cambio de que el software hable por cada uno.
* Los estereotipos existentes pueden perjudicar al sistema.

**Lenguajes**

Al igual que los microservicios le permiten utilizar innumerables lenguajes de programación, también hace difícil compartir código o cambiar de equipos de desarrollo, además de fragmentar la cultura de trabajo.

**Remote Procedure Call (RPC)**

Al usar este camino escalable, los protocolos HTTP/REST se vuelven más complicados. Además de que JSON necesite un nuevo esquema.

Por otro lado, y en comparación a los sistemas monolíticos, los RPCs son más lentos que los PCs debido a contar con nodos en diferentes locaciones. Es muy importante saber que los servidores no son exploradores.

**¿Cuántos repositorios son necesarios?**

Tener un solo repositorio es tan bueno como tener varios. Al mismo tiempo tener solo un repositorio es tan malo como tener varios. Matt Ranney opina que todo depende del sistema y los diferentes trabajos para que estos sean más fáciles de manejar y trabajar.

**Perspectiva Operacional**

Las consecuencias de qué sucede cuando un proyecto falla son un factor a tener en cuenta, ya que puede afectar a otros proyectos además de toda la empresa y sus procesos de negocio. La recomendación es pensar en el servicio a larga escala.

**Rendimiento**

El rendimiento es un punto importante que tomar en cuenta siempre. Dependiendo del tipo de hardware que se utilice en el sistema, el rendimiento siempre va a ser dependiente del lenguaje de programación que se adecue a la solución del proceso de negocio.

“El rendimiento nunca importa hasta que importa”. Probablemente se quiera tener un requisito simple para siempre pasar más allá del mínimo. La recomendación de Matt vendría ser no ser perfeccionador sino precavido, pensando en el futuro.

**Registro**

Para temas de consultoría o los mismos procesos de negocio, es importante un registro consistente y estructurado para que cuando haya problemas, los múltiples lenguajes no hagan de esta tarea algo difícil.

Una gran ola o cantidad de registro pueden provocar problemas, los cuales pueden ser resueltos con cierta contabilidad.

**Prueba de carga**

Se necesita probar o simular un servicio como si estuviese en producción, sin romper con las métricas. Lo recomendable es realizar estas pruebas todo el tiempo, ya que todos los sistemas necesitan ser capaces de manejar el tráfico de prueba como mínimo.

Por otro lado, las **pruebas de fallas** son necesarias para saber qué se debe mejorar, les guste o no.

**Migraciones**

Un punto que tomar en cuenta es que hasta los viejos sistemas pueden funcionar aún, en especial aquello denominado inmutable. Las migraciones obligatorias pueden llegar a ser perjudiciales ya que obligas al equipo a cambiar, pudiendo ocasionar problemas en el ambiente de la empresa como en el mismísimo desarrollo.