## 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 es que 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.

**Life Lessons**

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 a cada área/proyecto de este, de manera que se mejora 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.

¿Cuales 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 distribuído, 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. Esto puede provocar diferentes 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, lo cual puede terminar siendo un problema.
* Los estereotipos existentes pueden perjudicar al sistema.