**What I Wish I Had Known Before Scaling Uber to 1000 Services**

El expositor es Matt Ranney, cofundador de Uber. Se comienzas con datos que presenta Uber hasta Abril del 2016, en donde se puede ver que Uber se encontraba en más de 400 ciudades alrededor del mundo, 70 países; además, cuenta con más de 6000 empleados.

Uber ha crecido 10 veces más en un año y medio, dando como ejemplo el incremento de ingenieros, teniendo 200 de ellos y en un año y medio llegó hasta 2000 ingenieros trabajando.

Ellos utilizan microservicios para poder evitar tener sistemas monolíticos en varios servicios que ellos presentan porque pueden moverse y lanzarse independientemente, poseer su tiempo de actividad, y usar la mejor herramienta para el trabajo.

Entonces por qué hay más costos, ya sea porque ahora se tiene sistemas distribuidos, cada cosa está en un RPC e incluso si algo se rompe. Sin embargo, se pueden obtener costos bajos ya que cada cosa es una solución de compromiso, se puede construir alrededor de problemas, poder comercializar complejidad por políticas.

**Lenguajes**

Difícil para compartir código ya que hay diferentes lenguajes; además no se puede usar en varios dispositivos ya que no hay flexibilidad.

**RPC**

Indica que todo se convierte en RPC, HTPP/REST son complicados, también JSON necesita un esquema. Además, RPCs son más lentos que las PCs y los servidores no son navegadores.

**Cuántos repositorios**

Muchos repositorios está bien, uno está bien; ya que cada una de ellas puedes tener funciones en cada una ellas que te ayudan a gestionar mejor e incluso tener todo de manera organizada en un solo lugar

Operaciones: qué pasa cuando las cosas se rompen, esto ayuda a que los equipos sepan qué hacer cuando ellos lancen su servicio dependiendo de cómo se maneje el equipo. Además, se tiene que entender un servicio en un contexto largo, ya que así se puede llegar más a fondo en las operaciones de este

**Performance**

Depende de las herramientas de los lenguajes, ya que estas herramientas son diferentes según el lenguaje que se tiene. Esto más depende la selección del lenguaje que elija la persona. No importa la performance hasta que esté hecha, probablemente se quiere requerimientos cada vez más simples.

**Fanout**

La latencia en general es mayor o igual a la latencia más lenta. Este tema es explicado con un ejercicio. Se recomienda tener un porcentaje de latencia bajo. Para poder entender a Fanoout se puede hacer a través de Tracing.

**Logging**

Necesita consistencia, logging estructurado. Múltiples lenguajes hacen que esto sea difícil. Varios registros pueden amplificar problemas.

**Load Testing**

Se necesita probar contra la producción. Sin romper métricas. Preferible todo el tiempo. Todos los sistemas necesitan manejar el tráfico de pruebas.

**Migraciones**

Las cosas viejas todavía tienen que trabajar. Entonces surge la cuestión qué pasa si es mutable. Además, los mandatos son malos.

**Políticas**

Los servicios permiten a las personas manejar las políticas.

Por último, todas las cosas son soluciones de compromisos y hay que tratar de hacerlo intencionalmente.