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

La gran escalabilidad de Uber ha sido apoyada en el enfoque para desarrollar software como una serie de microservicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí. Cada microservicio es pequeño y corresponde a un área de negocio de la aplicación.

Matt Ranney comienza la conferencia compartiéndonos que el crecimiento en general, consecuencia de la globalización está generan un proceso de scaling a una velocidad sin precedentes, donde se pueden observar grandes cambios en el transcurso de tan solo 1 año como le pasó a Uber.

Para el caso de Uber, se necesitan crear y dejar un buen número de servicios por la usabilidad que maneja. Sin embargo, surgen algunas eventualidades resultado del uso y el incremento de los microservicios, reconoce sus ventajas como agilidad e independencia de cada uno de los servicios.

Surgen múltiples problemas como:

Lenguaje: Los microservicios permiten a los equipos escribir en el lenguaje que deseen y comunicarse entre sí sin problemas. Sin embargo, operar de esta manera trae consigo algunas desventajas: es complicado compartir código entre equipos, es complicado reubicar miembros de un equipo específico en uno distinto, de alguna forma fragmenta la cultura empresarial ya que se aleja de manejar un estándar como empresa.

RPC: Todo se vuelve un RPS lo que hace que los lenguajes s e vuelvan complicadas hacen es que json/rest se complicado, json necesita un esquema. Todas estas tecnologías que se ven genial en pequeña escala. Cuando se ven en gran escala empieza a añadirse un costo.

Repos: Al uso de repositorios está el debate en que si uno es bueno o malo. Al tener solo uno es fácil de buscar, pero no se puede ver el track ya que se perderían. Tener muchos repositorios lo hace complicado en el ordenamiento,

pero fácil de ver los historiales. Uber, por ejemplo, tiene alrededor de 8000 repositorios.

Operacional: En el deployment de los microservers, queda el problema de que, al cada uno de los equipos tener su propio ritmo, algunos pueden retener a otros equipos porque necesitan que el primer equipo despliegue su parte, lo que actúa en contraparte de la versatilidad de tener muchos equipos, hay que tener coordinación entre todos, recordar que todos son un gran equipo único, al final, todos trabajan en un solo proyecto que debe correr.

Performance: Dada la cantidad de herramientas e idiomas, para entender que tal es el desempeño de cada uno de los microservicios, se necesitan diferentes formatos de perfilamiento, lo que no ayuda a uniformizar gráficos y representaciones entre los diferentes, lo que hace muy difícil el entender el real performance de cada servicio. Ha esto se incluyen los dashboards, que tienden a ser hechos con diferentes variables y métricas por equipo, por lo que es necesario crear un dashbord standard, y hecho automáticamente, lo que libera al equipo de tener que hacerlo y permite mejor entendimiento a la hora de revisar.

FanOut: Surge el problema de fanout cuando hay mucha gente accediendo al servicio y siempre habrá una persona que le demorara más al aumentar la cantidad de persona habrá más gente que le tarde el servicio para esto sale tracing de los servicios para saber los problemas e intentar solucionarlos.

Failure testing: Se deben hacer testeos de errores, a los usuarios no les gustara ya que pueden perturbar sus operaciones, pero se deben hacer de todas maneras.