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

Uber ha crecido notablemente en 1 año y medio, pasando de los 200 desarrolladores a 2000 desarrolladores.

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

Hay un debate sobre cuantos repositorios usar, si usar varios o solo uno. Tener varios repositorios ayudan a la gestión de microservicios más pequeños. Si deseas hacer cambios, al tener varios repositorios tendrías que cambiar modulo por modulo; sin embargo, con un único repositorio solo harías un gran cambio. Uber 8000 repositorios.

Esto ha permitido que todos sus desarrolladores tengan autonomía ya que su código puede ser desplegado sin afectar a los demás. Además cada uno puede escribir en un lenguaje de programación diferente, ya que solo exponen la API (una interfaz común, a la que le da igual el lenguaje de programación en la que el microservicio esté programado por debajo) al resto de microservicios. Pero aquí hay un problema ya que, como dice Matt Ranney, ¿Qué pasaría si se quiere hacer un intercambio de equipos? Un miembro de un equipo programando en Python tiene que ir a otro programando en Java, no se podrá o que el miembro tenga que adaptarse le será complicado. Esa pequeña desventaja

Al tener pequeños equipos trabajando rápido en diferente microservicios aún hay un pequeño problema que es el de entender que todo el sistema trabaja como uno solo, como un solo equipo.

Entender la performance cuando hay diferentes lenguajes se convierte en un verdadero reto. Lo que han intentado hacer es tener un mismo formato para poder medir el performance a través de flame graphs (gráficos de flama).

Los dashboards se tienen que generar automaticamente, sino los equipos se la pasaran haciendo dashboards de cosas que ellos piensan que son importantes y luego cuando quieres rastrear un problema, los dashboards entre los equipos se verán totalmente diferentes. Lo que debieron hacer desde un comienzo es estandarizar los dashboards para cada microservicio con las cosas que todos estan de acuerdo que son necesarias.

Los equipos ven mal que su trabajo sea atacado por la prueba de fallos. Pero es algo que si o si se debe hacer y se tiene que acostumbrar a los equipos a la prueba de fallos.

No se puede desligar la política de los microservicios, ya que se pueden tomar decisiones que afecten a otros, poniendo a una persona sobre el equipo.

Todo lo relacionado con los microservicios es un constante intercambio, tienes que dejar unas cosas por otras, sacrificar. Se tiene que planificar y realizar los intercambios y sacrificios de manera intencional. Pero no como si fuera tu última opción. Para conllevar estos intercambios lo mejor es mentalizándote de que tienes que perder lo que te puedes dar el lujo de perder a cambio de nuevas funciones con valor añadido.