**“What I wish I knew before scaling Uber to 1000 services” – Matt Ranney**

El expositor comienza explicando el fenómeno de crecimiento que ocurre en algunas ciudades de China. Destaca la precisión de UBER en muchos lugares del mundo y la creación veloz de microservicios (1000) junto a ingenieros (2000), almacenados en 8000 repositorios de git. Señala que esto brinda lecciones.

Ranney concibe el crecimiento acelerado como algo natural, necesario para el mercado. Para esto, la arquitectura basada en microservicios es ideal. En una vista general, en esta exposición se señala la ventaja de tener muchos o pocos repositorios, la relación de equipos de trabajo en una organización y los métodos ideales para una labor uniforme entre equipos.

Los microservicios consisten básicamente en la arquitectura que divide la anterior arquitectura monolítica en partes más pequeñas. En esta se destaca la pregunta constante de “¿Qué pasa si se rompe? como algo negativo. Las cosas son más susceptibles a romperse (fallar) cuando se cambian, no obstante, el cambio es necesario. Se corre un gran riesgo al intentar cambiar algo del código. Sin embargo, también destaca los beneficios de los microservicios, como permitir formación rápida de equipos de trabajo y ejecutar en lo que trabajan de forma independiente, usar la mejor herramienta para el trabajo que realizan, etc.

**Características generales de microservicios**

Por otro lado, se mencionan los costos del uso de microservicios, algunos más notorios que otros:

* Se está ejecutando un sistema distribuido: Más difícil de trabajar que la arquitectura monolítica.
* Ahora todo es una llamada a procedimiento remoto, para utilizar código de otras máquinas.
* Incertidumbre ante el surgir de un problema al preguntarse si las personas localizadas en el área son las adecuadas.
* Construir entorno a los problemas es complicado
* Mantener tus prejuicios: Se toman decisiones en base a los equipos de trabajo. Por ejemplo: el lenguaje de programación a usar.
* Se escribe más software y se habla menos.

**Distintos lenguajes de programación**

Luego, se hace mención a la inclusión de distintos lenguajes de programación (Java, Python, Go). Es posible la comunicación entre distintos lenguajes, pero es difícil compartir código y moverse entre equipos (una persona al cambiar de equipo también cambiaria de lenguaje de programación, el cual tendría que aprender). Cada parte puede usar tecnologías diferentes, se puede usar la mejor tecnología para un problema específico, Puede ocurrir que todas las partes tienen tecnologías diferentes y existe el riesgo de que si una parte falla ninguno de los otros equipos pueda ayudar separando los grupos completamente. Igualmente, tiene un costo.

**RPC**

* Es en lo que se basa la comunicación de equipos de trabajo
* HTTP/REST complicados cuando se realiza la programación en servidor
* JSON genera problemas al realizarle cambios en alguna parte.
* Los RPC son más lentos que una llamada de procedimiento
* Los servidores no son navegadores: Tiene más sentido tratar todo como una llamada a función que un servicio web

**Repositorios**

Esta es una temática principal de la conferencia. Se dan distintos motivos para distintas posturas respecto a que si es mejor uno o muchos:

|  |  |  |
| --- | --- | --- |
|  | Es bueno | Es malo |
| Uno | Puedes hacer cambios transversales. Es fácil acceder a todo el código. | Será tan grande que será imposible de manejar. |
| Muchos | Sigue la tendencia de la industria a modularizar. Es mucho más fácil de manejar. | Implica forzar el sistema de compilación. Es difícil navegar por el código y realizar cambios transversales. |

**Problemas Operacionales**

También existen problemas con la implementación de microservicios.

* Generalmente un equipo de absoluto dueño de su servicio y otros equipos no pueden intervenir para realizar el “reléase”.
* A pesar de que los equipos se mueven rápidamente (y están divididos en microservicios), se debe seguir concibiendo el sistema como un todo.

**Problemas de desempeño**

* Los RPC tienen un costo más alto al tratar con distintos lenguajes de programación. Esto genera un desafío.
* Se debe considerar que tan diferentes son las herramientas usadas por los equipos de trabajo.
* Cuando un servicio es creado, debe tener un dashboard con un conjunto de datos de forma estandarizada y no variar dependiendo del grupo de trabajo.
* Es un gran debate si es necesario preocuparse por el rendimiento, peo siempre debemos optimizar.
* “Comprar computadoras es más barato que contratar ingenieros”
* Es mejor tener un acuerdo de nivel de servicio.

**Fanout**

* Fanout causa muchos problemas de rendimiento.
* Se rastrean problemas de despliegue distribuyendo el seguimiento. Hay que entender el recorrido de las solicitudes a través de la arquitectura.
* También se pueden usar registros, los cuales enlazan todos los servicios con un mismo ID.
* El rastreo representa mucho trabajo, por lo que una opción es rastrear una porción estadísticamente representativa de solicitudes (UBER solo lo hace en el 1%).
* El rastreo necesita de la propagación del contexto a través del lenguaje (varios lenguajes).

Logging

* Habiendo distintos lenguajes, muchos equipos y gente nueva constantemente, hay muchas maneras de acceder al sistema (logueo).
* Lo que se quiere hacer es compartir un enfoque común para el registro a través de herramientas que sean fáciles de usar para que así las personas no busquen otra forma de hacerlo. Esto a pesar de los múltiples lenguajes.
* Puede generarse sobrecarga de registros, por lo que se plantea una noción de contabilidad para rastrear quien está produciendo demasiados datos de registro.

**Pruebas de carga**

No hay forma de construir un entorno tan grande como el de producción y no se pueden generar datos de prueba realistas para probar todas las partes del sistema, por lo que es mejor ejecutar pruebas cuando no sean momentos pico.

**Migraciones**

* Alguien siempre está migrando algo a alguna parte.
* Las cosas deben seguir funcionando, aunque hay un tiempo de inactividad aceptable
* Al ser este un negocio global, siempre hay momentos pico de actividad.
* No es bueno obligar a migrar. Hay que ofrecer un sistema mejor.

**Open Source**

* Se plantea cuando se debe comprar y cuando construir.
* Alguna cosa en la que estas invirtiendo tiempo, probablemente alguien ya lo hizo más barato y mejor.
* Hay que evaluar porque uno debería usar algo privado (construido) como servicio.

**Política**

Esta se rompe cuando alguien pone sus propios valores por encima del grupo de trabajo o cuando un grupo de trabajo se pone por encima de la empresa.