Escuela Politécnica Nacional

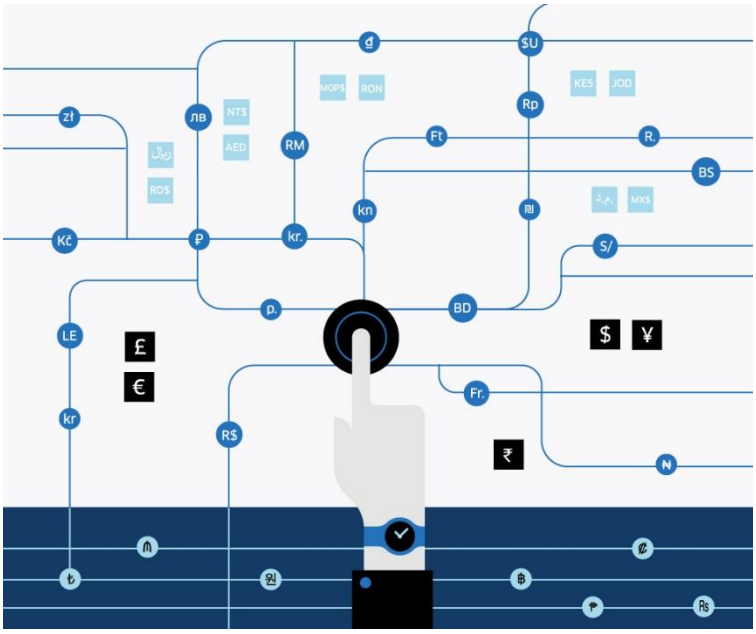
Aplicaciones Web Avanzadas

Nombre: Bryan Ortuño Barrera

Tema: Uber migración a microservicios.

Recientemente ha habido una discusión sustancial sobre las desventajas de las arquitecturas orientadas a servicios y las arquitecturas de microservicios en particular. Mientras que hace solo unos años, muchas personas adoptaron fácilmente arquitecturas de microservicios debido a los numerosos beneficios que proporcionan, como la flexibilidad en forma de implementaciones independientes, la propiedad clara, las mejoras en la estabilidad del sistema y una mejor separación de las preocupaciones, en los últimos años las personas han comenzado a denunciar los microservicios por su tendencia a aumentar en gran medida la complejidad, a veces haciendo que incluso las características triviales sean difíciles de construir.

A medida que Uber ha crecido a alrededor de 2,200 microservicios críticos, experimentamos estas compensaciones de primera mano. En los últimos dos años, Uber ha intentado reducir la complejidad de los microservicios sin dejar de mantener los beneficios de una arquitectura de microservicios. Con este ensayo se presenta el enfoque generalizado de las arquitecturas de microservicios, a las que Uber se refiere como "Arquitectura de microservicios orientada al dominio" (DOMA).



En Uber, se adoptó una arquitectura de microservicios porque se tenia (alrededor de 2012-2013) principalmente dos servicios monolíticos y se encontraron con muchos de los problemas operativos que resuelven los microservicios.

Riesgos de disponibilidad. Una sola regresión dentro de una base de código monolítica puede derribar todo el sistema (en este caso, todo Uber).

Implementaciones arriesgadas y costosas. Estos fueron dolorosos y lentos de realizar con la necesidad frecuente de retrocesos.

Mala separación de preocupaciones. Era difícil mantener buenas separaciones de preocupaciones con una enorme base de código. En un entorno de crecimiento exponencial, la conveniencia a veces conducía a límites pobres entre la lógica y los componentes.

Ejecución ineficiente. Estos problemas combinados dificultaron que los equipos se ejecutaran de forma autónoma o independiente.

En otras palabras, a medida que Uber creció de 10 a 100 ingenieros con múltiples equipos que poseían piezas de la pila tecnológica, la arquitectura monolítica unió el destino de los equipos y dificultó la operación independiente.

Como resultado, se adoptó una arquitectura de microservicios. En última instancia, los sistemas se volvieron más flexibles, lo que permitió a los equipos ser más autónomos.

Fiabilidad del sistema. La confiabilidad general del sistema aumenta en una arquitectura de microservicios. Un solo servicio puede caer (y ser revertido) sin derribar todo el sistema.

Separación de preocupaciones. Arquitectónicamente, las arquitecturas de microservicios le obligan a hacer la pregunta "¿por qué existe este servicio?" definiendo más claramente los roles de los diferentes componentes.

Propiedad clara. Se vuelve mucho más claro quién poseía qué código. Los servicios suelen ser propiedad a nivel individual, de equipo u organización, lo que permite un crecimiento más rápido.

Ejecución autónoma. Las implementaciones independientes + líneas de propiedad más claras desbloquean la ejecución autónoma por parte de varios equipos de productos y plataformas.

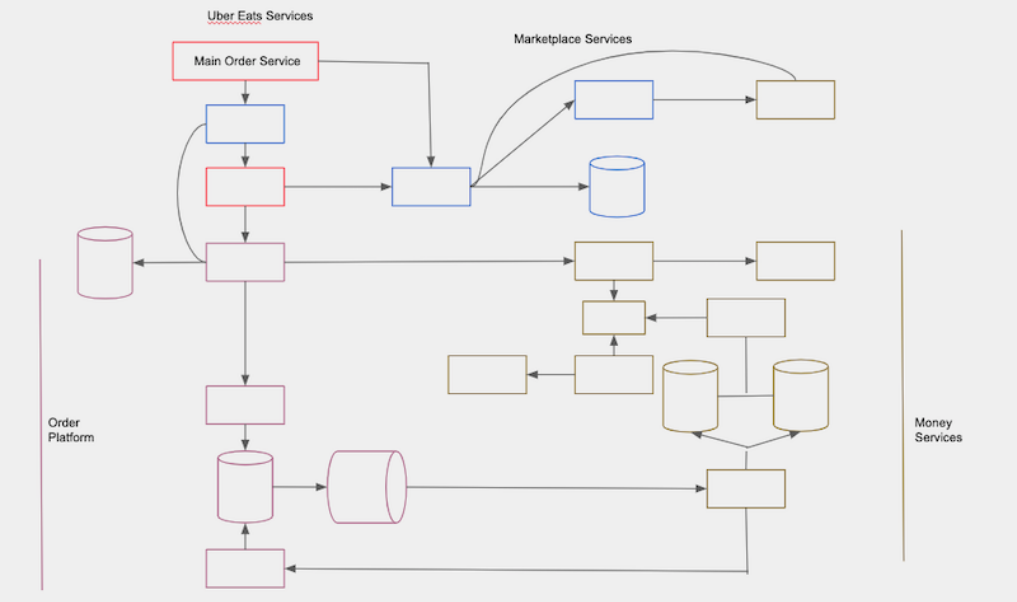
Velocidad del desarrollador. Los equipos pueden implementar su código de forma independiente, lo que les permite ejecutar a su propio ritmo.

No es una exageración decir que Uber no habría podido lograr la escala y la calidad de ejecución que se mantiene hoy en día sin una arquitectura de microservicios.

Sin embargo, a medida que la compañía crecía aún más, de 100 a 1000 ingenieros, se comenzó a notar un conjunto de problemas asociados con una complejidad del sistema mucho mayor. Con una arquitectura de microservicios, se intercambia una única base de código monolítica por cajas negras cuya funcionalidad puede cambiar en cualquier momento y causar fácilmente un comportamiento inesperado.

Por ejemplo, los ingenieros tuvieron que trabajar a través de alrededor de 50 servicios en 12 equipos diferentes para investigar la causa raíz del problema.

Comprender las dependencias entre servicios puede ser bastante difícil, ya que las llamadas entre servicios pueden profundizar en muchas capas. Un pico de latencia en la enésima dependencia puede causar una cascada de problemas aguas arriba. La visibilidad de lo que realmente está sucediendo es imposible sin las herramientas adecuadas, lo que dificulta la depuración.



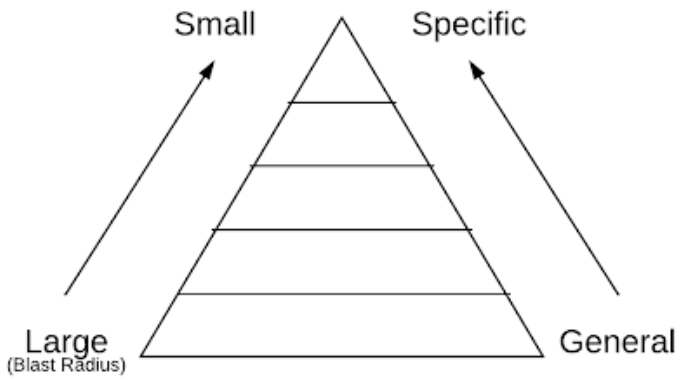
Dominios

Los dominios uber representan una colección de uno o más microservicios vinculados a una agrupación lógica de funcionalidad. Una pregunta común en el diseño de un dominio es "¿qué tan grande debe ser un dominio?" No damos ninguna orientación aquí. Algunos dominios pueden incluir decenas de servicios, algunos dominios solo un servicio único. La tarea importante es pensar cuidadosamente sobre el papel lógico de cada colección. Por ejemplo, nuestros servicios de búsqueda de mapas constituyen un dominio, los servicios de tarifas son un dominio, la plataforma coincidente (coincidencia de pasajeros y conductores) es un dominio. Estos tampoco siempre siguen la estructura organizativa de la empresa. La organización de Uber Maps en sí se divide en tres dominios, con 80 microservicios detrás de 3 puertas de enlace diferentes.

Diseño de capas

El diseño de capas responde a la pregunta de "¿qué servicio puede llamar a qué otro servicio?" dentro de la arquitectura de microservicios de Uber. Como resultado, podemos pensar en el diseño de capas como "separación de preocupaciones a escala". Alternativamente, podemos pensar en el diseño de capas como "gestión de dependencias a escala".

El diseño de capas describe un mecanismo para pensar en el radio de explosión de fallas y la especificidad del producto en todas las dependencias del servicio en Uber. A medida que los dominios se mueven de la capa inferior a la capa superior, afectan a menos servicios en el caso de una interrupción y representan casos de uso de productos más específicos. Por el contrario, la funcionalidad en las capas inferiores tiene más dependientes y, como resultado, tiende a tener un radio de explosión más grande y representa un conjunto más general de funcionalidad empresarial. La siguiente figura ilustra este concepto.



Uno puede pensar en las capas superiores como experiencias de usuario específicas (como las características móviles), y las capas inferiores como funcionalidad comercial generalizada (como la administración de cuentas o los viajes al mercado). Las capas solo dependen de las capas debajo de ellas, lo que nos da una heurística útil para pensar en preguntas como el radio de explosión y la integración de dominios.

Vale la pena señalar que la funcionalidad a menudo mueve "hacia abajo" este gráfico de específico a más general. Uno puede imaginar una característica simple que eventualmente se convierte cada vez más en una plataforma a medida que evolucionan los requisitos. De hecho, se espera este tipo de migración a la baja, y muchas de las plataformas comerciales principales de Uber comenzaron como funcionalidades específicas para pasajeros o conductores que se generalizaron a medida que desarrollamos más líneas de negocio y asumieron más dependencias (como Uber Eats o Uber Freight).

Dentro de Uber, se establecen las siguientes cinco capas.

Capa de infraestructura. Proporciona funcionalidad que cualquier organización de ingeniería podría utilizar. Es la respuesta de Uber a las grandes preguntas de ingeniería, como el almacenamiento o las redes.

Capa de negocio. Proporciona una funcionalidad que Uber como organización podría usar, pero que no es específica de una categoría de producto o línea de negocio (LOB) en particular, como Viajes, Comidas o Fletes.

Capa de producto. Proporciona funcionalidad que se relaciona con una categoría de producto o LOB en particular, pero es independiente de la aplicación móvil, como la lógica de "solicitar un viaje" que se aprovecha en múltiples aplicaciones orientadas a Rides (Rider, Rider "Lite", m.uber.com, etc.).

Presentación. Proporcionar funcionalidad que se relacione directamente con las características que existen dentro de una aplicación orientada al consumidor (móvil/web).

Capa de borde. Expone de forma segura los servicios de Uber al mundo exterior. Esta capa también es consciente de la aplicación móvil.

Como puede ver, cada capa posterior representa una agrupación de funcionalidad cada vez más específica, y tiene un radio de explosión cada vez más pequeño (o, en otras palabras, menos componentes dependen de la funcionalidad dentro de esa capa).

Referencias Bibliográficas

[1]A. Gluck, "Introducing Domain-Oriented Microservice Architecture", Uber Engineering Blog, 2022. [Online]. Available: https://eng.uber.com/microservice-architecture.

[2]E. Reinhold, "Rewriting Uber Engineering: The Opportunities Microservices Provide", Uber Engineering Blog, 2022. [Online]. Available: https://eng.uber.com/building-tincup-microservice-implementation/.