Capítulo 5: Dividiendo aplicaciones monolíticas

Víctor Iranzo

25 de mayo de 2018

En este capítulo se explica como abordar la transformación de una aplicación monolítica de forma evolutiva hacia un sistema basado en microservicios. Las aplicaciones monolíticas tienden a crecer con el tiempo, por lo que se hacen frágiles y poco mantenibles al juntar muchas veces código no relacionado. Es por este motivo por el que un equipo pueda preferir no modificar una aplicación así, al menos no de forma descontrolada.

1. Costuras: pasos para dividir lo monolítico

En el libro "Working Effectively with Legacy Code"se define una costura como una porción de código que se puede tratar de manera asilada sin alterar al resto del sistema. Las costuras son firmes candidatos a convertirse en futuros servicios.

El primer paso de nuestra refactorización será identificar las costuras. Muchos lenguajes de programación permiten la creación de espacios de nombres o paquetes. Si podemos, moveremos todo el código del contexto que hemos encontrado a un nuevo paquete mediante refactorizaciones del IDE.

Siguiendo este procedimiento, terminaremos viendo qué código se ha agrupado correctamente y qué código parece que no encaje en ningún paquete. El código sobrante puede estudiarse para ver si se puede agrupar como uno o varios paquetes o se puede añadir a la solución de otra forma.

Durante todo el proceso, el código debe representar una situación real, por lo que las interacciones y dependencias entre los paquetes será similar a la existente en la realidad. También cabe mencionar que la transformación a microservicios no necesita realizarse de golpe, sino que se pueden ir transformando paquetes progresivamente para limitar el impacto de hacer algo mal. A la hora de elegir por qué paquete comenzar, podemos elegir el

que supongamos que nos aportará mayor beneficio al separarlo del resto o el paquete que menos dependencias tenga con el resto.

2. Razones para refactorizar aplicaciones monolíticas

- Tranquilidad de cambio: un cambio en un servicio se podrá hacer más rápido de lo que se hacia antes, además de poderse implementar y desplegar de forma autónoma.
- Organización de los equipos: diferentes equipos pueden encargarse de distintos microservicios sin que interfieran los cambios de unos con los de otros.
- Seguridad: la seguridad ahora puede aplicarse a un servicio concreto en lugar de a todo el sistema.
- Tecnológica: los servicios son más sencillos de refactorizar y cambiar a una nueva tecnología se puede hacer de forma progresiva y aislada en un único contexto.

3. Refactorizaciones en bases de datos

La base de datos es la infraestructura donde más enredo de dependencias hay. El proceso a seguir para separar la persistencia de la aplicación en diferentes bases de datos consiste también en buscar costuras. Vamos a explicar algunos problemas que se pueden encontrar en este proceso.

3.1. Claves ajenas

Un problema típico es que una tabla en BD que pertenece a uno de los microservicios extraídos referencie a través de una clave ajena a un elemento de una tabla de otro contexto. Estas dos tablas van a estar en servicios diferentes, así que para obtener los datos que una tupla referencia de la otra tabla, en lugar de recorrer la clave ajena, haremos una llamada a la API del otro servicio.

Como consecuencia, cuando queramos obtener un elemento de la primera tabla haremos primero una llamada a la base de datos de nuestro servicio

y después una llamada a la API del otro microservicio para obtener el elemento referenciado. Este microservicio hará una llamada a su base de datos y devolverá el elemento referenciado al servicio que lo ha invocado.

Al haber eliminado la clave ajena explícitamente en base de datos, las restricciones asociadas a esta (como por ejemplo la existencia del elemento referenciado) han de mantenerse a nivel de código.

3.2. Datos estáticos compartidos

Algunas tablas son referenciadas por muchas otras y cambian inusualmente debido al carácter de los datos que almacenan, por ejemplo, tablas con códigos de países. Si diferentes contextos acceden a esta tabla, una posible solución es duplicar la tabla en cada nueva base de datos de los servicios e implementar algún mecanismo que mantenga la consistencia en todas ellas.

Otra manera aún más sencilla es mover estos datos al código como un archivo o recurso en cada servicio. El problema de la consistencia sigue estando, pero es más fácil de resolver en el código que en las bases de datos.

Una tercera solución es crear un microservicio que tenga estos datos estáticos. Los servicios que quieran acceder a ellos lo harán llamando a su interfaz. Esta alternativa puede ser demasiado compleja si los datos estáticos tienen poco volumen o complejidad.

3.3. Datos mutables y tablas compartidas

Surge un problema cuando dos de las costuras que hemos encontrado acceden a la misma tabla en modos de lectura y escritura. Esto puede ocurrir porque estamos almacenando en una tabla conceptos que no son de un mismo dominio o porque existe una entidad del dominio en la base de datos que no existe como tal en el código.

En el primer caso, la respuesta pasa por dividir la tabla en dos porque se están mezclando conceptos de diferentes dominios. Cada uno de los microservicios acogerá una de las dos tablas.

En el segundo caso, la solución pasa por crear como tal la entidad en el código y que ya existía en la base de datos. Esta nueva entidad pertenecerá a un nuevo microservicio. Para abordar la refactorización progresivamente, primero lo crearemos como un paquete independiente y después haremos que todos los servicios que accedían a la antigua tabla accedan a la entidad a través de llamadas a la interfaz que expone el servicio.

4. Transacciones

5. Interacción con grandes volúmenes de datos