Capítulo 4: Integración de microservicios

Víctor Iranzo

22 de mayo de 2018

La integración de servicios es la parte más relevante en los sistemas basados en este concepto. Hacerlo correctamente nos asegurará su autonomía y su despliegue de manera independiente. Existen muchas tecnologías para la integración: SOAP (Simple Object Access Protocol), RPC (Remote Procedure Call) o REST (Representational State Transfer). De cualquiera de estas tecnologías esperamos las siguientes características:

- Evitar cambios en los consumidores: la tecnología escogida debe hacer que el número de cambios en un servicio que impliquen cambios en sus consumidores sean los menos posibles.
- No imponer una tecnología específica: la tecnología empleada para la comunicación entre servicios no debe restringir la tecnología empleada en estos. Se debe mantener la heterogeneidad tecnológica de los servicios y el protocolo empleado para integrarlos debe poderse emplear en cuantas más tecnologías mejor.
- Hacer simple el consumo de un servicio: los consumidores deberían de tener total libertad en la tecnología que emplean y consumir un servicio para ellos no debe ser complejo de implementar.
- Ocultar detalles de la implementación: el consumidor de un servicio no debe conocer los detalles de como este está implementado internamente. Así, los interlocutores están desacoplados y se evitan cambios en el consumidor asociados al servicio.
- Soportar operaciones más allá de las CRUD: las operaciones CRUD para crear, leer, actualizar y eliminar elementos están soportadas en la mayoría de tecnologías de integración. Sin embargo, un sistema requiere dar soporte a más procesos que se deben poder exponer en una interfaz de un servicio.

1. Integración por base de datos

La integración a través de bases de datos es una de las más empleadas en la industria. En nuestro contexto, esto se traduce a que todos los servicios de un sistema comparten la misma base de datos y cuando uno de ellos quiere hacer una operación u obtener información de otro, lo hace a través de la base de datos sobre las entidades del servicio consumido.

Este tipo de integración tiene un gran número de desventajas. En primer lugar, se está permitiendo a terceros ver y modificar detalles de la implementación interna de un servicio. Como consecuencia, un cambio en el esquema de la base de datos puede suponer romper los consumidores, que se puede comprobar invirtiendo en pruebas de regresión.

En segundo lugar, los consumidores están obligados a emplear una tecnología específica. De esta forma los servicios pierden autonomía a la hora de elegir su propia tecnología y quedan acoplados unos con otros. Por ejemplo, un cambio en el tipo de base de datos empleada (de un esquema relacional a uno clave-valor) implica cambios en todo el sistema.

Por último, con esta aproximación la lógica del servicio queda repartida entre sus consumidores. Acciones como modificar un elemento del servicio pueden ser invocadas por múltiples consumidores. Pongamos por caso que esto lo realizan 3 consumidores. Cada uno es responsable de hacer una consulta a la base de datos para este propósito, lo que se traduce en que la misma lógica quede replicada en tres sitios. Esto supone la pérdida de cohesión en el sistema, porque servicios independientes cambiarían juntos por consumir a otro.

En definitiva, la integración por base de datos es una buena forma de compartir datos pero no lógica y emplearla en una solución basada en microservicios implica la pérdida de la cohesión y el acoplamiento entre servicios.

2. Llamadas síncronas y asíncronas

En la comunicación síncrona, cuando se realiza una llamada a un servidor esta se bloquea hasta que la operación solicitada se complete. En la comunicación asíncrona, el invocador no espera a que la llamada se complete para continuar trabajando.

Es más sencillo razonar sobre las primeras, pero son las llamadas asíncronas las más efectivas cuando se solicitan operaciones largas o cuando se desea hacer una aplicación responsive.

El primer tipo de comunicación está ligado al patrón petición-respuesta (en inglés, request-response), aunque también puede ser empleado de forma asíncrona a través de callbacks que recojan la respuesta del servidor. Otro tipo de patrón es el basado en eventos: el cliente, en lugar de hacer una petición para realizar una operación, comunica al servidor un evento que ha observado y es el servidor quien debe saber qué hacer ante lo ocurrido. Este patrón es de naturaleza asíncrona y disminuye el acoplamiento entre cliente y servidor.

3. Patrones orquestador y coreógrafo

A la hora de implementar un proceso de negocio múltiples servicios deben participar. Para su coordinación, existen pos estilos arquitectónicos que se pueden seguir: la orquestación, basado en un componente que guía y dirige al resto de servicios como un director de orquesta, y la coreografía, donde cada parte sabe el trabajo que debe realizar y se coordina con el resto como bailarines realizando una coreografía.

Por una parte, el patrón orquestador es más sencillo de implementar, pudiéndose hacer con herramientas para el modelado de procesos de negocio. Sin embargo, al usarlo se otorga demasiada autoridad a uno de los servicios del sistema y el resto pueden llegar a transformarse en simples servicios CRUD.

Por otra parte, el patrón coreógrafo se basa en la comunicación asíncrona. El proceso se inicia mediante un evento al que los servicios que deban reaccionar se suscriben. Así se puede formar una cadena donde cuando uno de los servicios acaba puede comenzar la actividad siguiente mediante el envío de un evento. Su principal ventaja es que reduce el acoplamiento entre los servicios, pero puede llegar a dificultar la visión del proceso de negocio y requiere más trabajo para ser monitorizado.

En general, las soluciones que siguen el patrón orquestador son más difíciles de cambiar y se suelen programar de forma síncrona, por lo que puede ser perjudicial para procesos largos. Es por esto que se recomienda más el segundo patrón, mucho más flexible, aunque se puede optar por una solución híbrida.

4. Integración RPC

La llamada a procedimiento remoto (en inglés, Remote Procedure Call o RPC)es una técnica que permite ejecutar una llamada a un servicio remoto como si de una llamada local se tratara. Existen diferentes tecnologías que la implementan a través del uso de interfaces, que facilitan la comunicación entre cliente y servidor. No es necesario que cliente y servidor empleen la misma pila tecnológica, aunque algunas tecnologías como Java RMI sí lo requieren y limitan al resto de servicios.

El formato de los mensajes o los protocolos de red empleados también varían de una tecnología a otra. En cuanto a los formatos, SOAP (proviene del inglés Simple Object Access Protocol) emplea XML (también del inglés eXtensible Markup Language) mientras que por ejemplo Java RMI transmite mensajes binarios. Los protocoles de red empleados van desde HTTP sobre TCP, que emplea SOAP, hasta UDP, un protocolo más rápido y ligero.

El coste asociado a hacer una llamada remota es mucho mayor al de una local. Los objetos han de serializarse y mandarse a través de la red, teniendo en cuenta que esta no es segura. El tratamiento de errores no puede ser el mismo y ha de estar prevenido de la latencia de la red.

Algunas tecnologías RPC pueden ser frágiles, como por ejemplo Java RMI. Las interfaces que ofrecen los servidores están muy ligadas a la representación del objeto sobre el que se desea hacer una operación. Además, un cambio en la interfaz puede suponer romper alguno de los consumidores, con lo que puede ocurrir que al desplegar el servidor se tengan que desplegar todos los clientes que lo invocan.

5. Integración REST

La transferencia de estado representacional (en inglés, REpresentational State Transfer) es un estilo arquitectónico inspirado en la Web. Se basa en el concepto de recurso, un objeto que el servicio conoce y del que puede crear diferentes representaciones bajo demando. La representación del recurso está completamente desacoplada de como se almacena.

La arquitectura REST es ampliamente usada junto con HTTP (término que proviene del inglés Hypertext Transfer Protocol). Los verbos que se definen en HTTP actúan sobre recursos: por ejemplo, con el verbo GET se puede obtener una representación del recurso y con el POST crear uno nuevo.

5.1. HATEOAS: Hypermedia As the Engine of Application State

Un principio introducido en HTTP y que reduce el acoplamiento es el de hipermedia como motor del estado de la aplicación (HATEOAS). Este consiste en que un cliente, a través de enlaces que contiene un recurso hacia otros, interactúa con el servidor navegando con los enlaces hacia los recursos que le interesan.

En una conversación normal entre cliente y servidor, el cliente comenzará solicitando un recurso que estará enlace con otros a través de referencias. El cliente debe conocer la semántica del resultado devuelto por el servidor para saber que significan cada uno de los enlaces del recurso. Aunque cambie la lógica del servidor, el significado del enlace seguirá siendo el mismo y se podrá consumir el recurso de igual manera. Esto reduce el acoplamiento entre servidor y cliente, a menos que se cambie la semántica del servidor y cada enlace pase a tener otro significado. Como consecuencia, esto puede suponer que se rompan todos los consumidores.

Entre las desventajas de este principio se encuentra que la comunicación entre cliente y servidor se ve alargada porque el primero tiene que hacer numerosas navegaciones para encontrar el recurso que busca.

5.2. Frameworks para crear servicios RESTFul

Existen algunos frameworks como Sprint Boot que promueven malas prácticas para fabricar servicios más rápidamente. Estos frameworks toman la representación en base de datos de los objetos, la deserializan y exponen al exterior del servicio para su consumo. Se debe hacer una diferenciación entre cómo se almacenan los datos y cómo se exponen a otros servicios para reducir el acoplamiento entre ambas representaciones.

Una técnica que se puede emplear es retrasar la implementación de cómo se almacenan los datos hasta que se estabilice la interfaz del servicio. De esta manera nos aseguraremos que la representación que obtienen los consumidores es la que realmente quieren y no depende de cómo se almacena.