Capítulo 1: Microservicios

Víctor Iranzo

16 de mayo de 2018

1. ¿Qué son los microservicios?

Los microservicios son servicios pequeños y autónomos que trabajan conjuntamente. Cuanto mayor énfasis se haga en estas dos características de los microservicios a la hora de desarrollarlos, mayores serán los beneficios de usar esta aproximación.

1.1. Pequeños

¿Cómo de pequeño ha de ser un microservicio? En piezas de código muy largas resulta difícil encontrar dónde se debe realizar un cambio o dónde se encuentra un defecto. En todo momento han de cumplirse los principios de cohesión, coherencia y responsabilidad única: el código relacionado debe agruparse conjuntamente porque será modificado por el mismo motivo. Otro factor a tener en cuenta a la hora de establecer los límites de un microservicio es el del tamaño del equipo que lo mantiene: si el tamaño del servicio es demasiado grande para ser gestionado por el equipo, debería ser dividido en partes más pequeñas.

Una regla que puede ser aplicada según el autor Jon Eaves es establecer un tamaño para un microservicio tal que pueda ser completamente reescrito en 2 semanas.

1.2. Autónomos

Un microservicio es una entidad separada: cada uno cambia independientemente del resto y al hacerlo los consumidores no necesiten ser modificados. Para lograrlo, lo más habitual es que cada servicio exponga una interfaz de

programación de aplicaciones (API) y todas las comunicaciones se realicen mediante llamadas a la red.

2. Beneficios de los microservicios

2.1. Heterogeneidad tecnológica

El desarrollo basado en microservicios nos permiten poder escoger una tecnología diferente para cada uno de ellos. De esta manera, escogeremos aquella que mejor se adecue a las necesidades concretas y no una tecnología estándar válida para todos ellos.

Sin embargo, se puede poner límite a esta heterogeneidad. Empresas como Netflix y Twitter emplean mayoritariamente Java Virtual Machine (JVM) como base de sus microservicios combinado con diferentes *stacks tecnológicos* que utilizan según sus necesidades.

2.2. Resilencia

La resilencia es la capacidad de un sistema para seguir funcionando cuando un error ocurre. Si una funcionalidad del sistema falla y este no se propaga en cascada, el error puedo aislarse y el resto del sistema puede continuar funcionando.

En los sistemas distribuidos, hace falta comprender y lidiar nuevas fuentes de errores. Una buena práctica es asumir, por ejemplo, que las redes pueden y van a fallar.

2.3. Escalabilidad

En las arquitecturas monolíticas, para escalar una funcionalidad concreta hace falta escalar el sistema completo. Empleando microservicios, basta con estudiar y escalar solo aquellos que realmente necesitan ser escalados. Esto produce un mejor aprovechamiento de los recursos y un ahorro en los costes.

2.4. Facilidad de despliegue

Un cambio en una línea de código puede suponer el despliegue de la aplicación completa para publicar dicho cambio. Esto se traduce en un aumento

del riesgo a la hora del despliegue. Podría parecer que una solución a este problema fuera hacer un menor número de despliegues. Sin embargo, como la integración continua sugiere, incrementar el número de cambios entre dos publicaciones también supone un gran impacto.

Los microservicios permiten que un cambio en un único servicio se despliegue de manera independiente, haciendo que los cambios lleguen al cliente final de forma más rápida.

2.5. Alineación con la organización del trabajo

Problemas asociados a bases de código inmensas donde colabora un gran equipo pueden ser evitados empleando microservicios. Si el código está repartido entre diferentes componentes, equipos pequeños pueden encargarse de mantener cada una de ellas. De esta manera, la organización del trabajo adquiere un enfoque más ágil al estar formado por equipos independientes y auto-organizados.

2.6. Componibilidad

Si una aplicación se divide en componentes, es más sencillo reutilizar funcionalidad, que puede ser consumida de diferentes formas.

2.7. Reemplazamiento

En la mayoría de empresas abundan los sistemas legados que nadie desea mantener. La refactorización de estos sistemas es imposible por su tamaño y riesgo.

Si en lugar de haber empleado una arquitectura monolítica para su diseño se emplearan microservicios observaríamos como los barreras para la refactorización no existen al poderse reescribir el microservicio al completo en pocos días.

3. Arquitectura orientada a servicios

Las arquitecturas orientadas a servicios (en inglés service-oriented archicture, SOA) son una aproximación de diseño donde múltiples servicios

colaboran para ofrecer una serie de funcionalidades finales. Los microservicios pueden entenderse como una aproximación específica de las arquitecturas SOA. Son una solución para un gran número de problemas de las aplicaciones monolíticas:

- Promueven la reusabilidad del código.
- Son más fáciles de mantener.
- Fomentan buenas prácticas como el principio de responsabilidad única.
- Permiten solventar un mismo problema de diferentes formas.

No obstante, existe una falta de consenso sobre cómo debe ponerse en práctica este tipo de arquitecturas en aspectos como los protocolos de comunicación a emplear o la granularidad de los servicios.

4. Otras técnicas de descomposición

4.1. Librerías

Las librerías son una forma de compartir funcionalidades entre equipos y servicios. Pueden haber sido desarrolladas dentro de la propia organización o por una tercera entidad. Entre las desventajas que suponen su uso frente al de los microservicios se encuentran:

- Pérdida de la heterogeneidad tecnológica: una aplicación que pretenda usar una librería normalmente debe implementarse en la misma plataforma para la que la librería se desarrolló.
- Decremento de la escalabilidad: al depender unas partes del sistema de otras, no se puede escalar una de ellas sin escalar la otra.
- Incapacidad para desplegar de manera independiente: a menos que se empleen librerías dinámicas, no se puede desplegar una librería sin desplegar el sistema completo donde se emplea.
- Aumento del acoplamiento: el código compartido puede suponer que una parte del sistema dependa en gran medida de una librería cuando la funcionalidad que ofrece debería pertenecer íntegramente a esa parte y no a la librería.

4.2. Módulos

Los módulos permite gestionar su propio ciclo de vida sin necesidad de detener el resto de procesos del sistema. Un lenguaje de programación que permite la descomposición en estos elementos es Erlang. Los problemas asociados a este tipo de arquitecturas son similares a los de las librería de código.