Aplicaciones Distribuídas en Internet

Curso 2022-2023 - Trabajo teórico

Microservicios

Florián Rodríguez Mirón

Pablo Ortega Fuster

Microservicios

1. ¿Qué son los microservicios?
2. Características
3. Arquitectura monolítica vs. microservicios
4. Desafíos
5. Tecnologías y herramientas clave
6. Microservicios y cloud
7. Patrones
8. Antipatrones
9. Referencias

1. ¿Qué son los microservicios?

Los microservicios son un enfoque para desarrollar aplicaciones de software en el que se divide una aplicación en pequeños servicios independientes que se comunican entre sí a través de una interfaz común. Cada servicio se ejecuta y se puede escalar de forma independiente, lo que permite una mayor flexibilidad y escalabilidad en comparación con una arquitectura monolítica tradicional. Los microservicios también son más fáciles de mantener y depurar debido a su baja granularidad y aislamiento entre ellos.

2. Características

Algunas de las características comunes de los microservicios incluyen:

- **Independencia**: cada servicio se ejecuta de forma independiente y se puede escalar de forma independiente.
- **Granularidad**: los servicios son pequeños y se enfocan en realizar una tarea específica.
- **Interfaz común**: los servicios se comunican entre sí a través de una interfaz común, como una API.
- **Tecnología**: cada servicio puede estar construido con diferentes tecnologías, según la necesidad de ese servicio.
- **Desacoplamiento**: los servicios están desacoplados entre sí, lo que permite cambios en un servicio sin afectar al funcionamiento de los demás.
- **Resiliencia**: los servicios no dependen unos de otros, por lo que la caída o fallo de uno no conlleva la caída o fallo de los demás.
- **Monitoreo**: los servicios se monitorean de forma independiente lo que permite detectar y solucionar problemas más rápidamente.

- **Despliegue continuo**: se despliegan de forma independiente y automatizada lo que permite una mayor velocidad en el lanzamiento de características.

3. Arquitectura monolítica vs. microservicios

Las arquitecturas monolíticas y los microservicios son dos enfoques opuestos para desarrollar aplicaciones de software.

Una arquitectura monolítica es el enfoque tradicional en el que todas las funcionalidades de una aplicación se encuentran en un solo paquete o módulo. Esto significa que todas las capas de una aplicación, como la capa de presentación, la capa de negocio y la capa de datos, están todas juntas en una sola aplicación. Esto hace que sea más fácil desarrollar y probar una aplicación, ya que todo está en un solo lugar. Sin embargo, esto también significa que es más difícil escalar y mantener una aplicación a medida que crece y se vuelve más compleja.

En cambio, los microservicios, cada servicio se enfoca en realizar una tarea específica y se puede escalar de forma independiente. Esto significa que es más fácil escalar y mantener una aplicación a medida que crece y se vuelve más compleja. Sin embargo, es más difícil desarrollar y probar una aplicación debido a que hay más piezas móviles y se requiere una mayor coordinación entre los servicios.

En general, las arquitecturas monolíticas son adecuadas para aplicaciones pequeñas y simples, mientras que los microservicios son adecuados para aplicaciones grandes y complejas que requieren escalabilidad y flexibilidad.

4. Desafíos

Algunos de los desafíos comunes que presentan los microservicios incluyen:

- **Complejidad en la coordinación**: El hecho de que hay más piezas móviles y se requiere una mayor coordinación entre los servicios puede hacer que sea más difícil desarrollar y probar una aplicación.
- **Mayor complejidad en el despliegue**: el despliegue y configuración de microservicios es más compleja ya que se necesita un sistema de orquestación y un sistema de balanceo de carga.
- **Mayor complejidad en el monitoreo**: El monitoreo de una gran cantidad de servicios independientes puede ser un desafío.
- **Mayor complejidad en la seguridad**: se requiere una mayor seguridad en la comunicación entre los servicios.
- **Mayor complejidad en la gestión de datos**: los datos deben ser compartidos y sincronizados entre los servicios, lo cual puede ser un desafío.
- **Mayor complejidad en la gestión de errores**: los errores en un servicio pueden afectar a otros servicios, por lo que es necesario tener una buena gestión de errores.

5. Tecnologías y herramientas clave

Algunas tecnologías y herramientas clave utilizadas en la implementación de microservicios incluyen:

- **Contenedores**: Herramientas como Docker permiten empaquetar y desplegar aplicaciones de manera consistente en diferentes entornos.
- **Orchestration**: Herramientas como Kubernetes, Mesos y Docker Swarm permiten automatizar el despliegue, escalado y gestión de contenedores en un cluster.
- **Service discovery**: Herramientas como Consul, Etcd y Apache ZooKeeper permiten a los servicios encontrarse y comunicarse entre sí de manera dinámica.
- **APIs**: Una buena práctica en microservicios es utilizar APIs RESTful para comunicarse entre servicios.
- **Registros**: Herramientas como Elasticsearch, Logstash, Kibana o Splunk permiten recopilar y analizar los registros de los servicios para facilitar la solución de problemas.
- **Gateway**: Herramientas como Kong, Tyk o Netflix Zuul permiten establecer un punto de entrada único para los servicios, lo que permite una mejor gestión de la seguridad y el enrutamiento de las peticiones.
- **Sistemas de monitoreo**: Herramientas como Prometheus, InfluxDB, Grafana permiten monitorear el rendimiento y la disponibilidad de los servicios.
- **Mensajería y transmisión de sucesos**: Es el caso de plataformas de transmisión de sucesos como por ejemplo, Apache Kafka o intermediarios de paso de mensajes de uso general. Una combinación de microservicios con una arquitectura basada en sucesos es garantía para crear sistemas altamente resilientes y escalables, que en todo momento utilicen los recursos necesarios para su correcto funcionamiento.

6. Microservicios y cloud

Los microservicios y la nube (cloud) son dos tecnologías relacionadas que se utilizan juntas para desarrollar y desplegar aplicaciones.

Los microservicios y la nube se complementan mutuamente, ya que los microservicios requieren una escalabilidad y flexibilidad que se puede lograr fácilmente en la nube. Además, el uso de microservicios en la nube permite esta mayor escalabilidad y flexibilidad, ya que cada servicio se puede escalar de forma independiente y se pueden utilizar diferentes tecnologías para cada servicio.

Algunas de las principales plataformas de cloud computing que son populares para desplegar microservicios son Amazon Web Services (AWS), Google Cloud Platform (GCP) y Microsoft Azure. Estas plataformas ofrecen una variedad de servicios y herramientas para facilitar el despliegue y la gestión de microservicios en la nube, como contenedores, orquestación, balanceo de carga, monitoreo y más.

7. Patrones

- **Patrón de Backend-For-Frontend (BFF)**: este patrón inserta una capa entre la experiencia del usuario y los recursos que invoca dicha experiencia, que permite a los desarrolladores crear y dar soporte a un tipo de *back end* por interfaz de usuario utilizando las mejores opciones para esa interfaz. Por ejemplo, una aplicación utilizada en un escritorio tendrá diferentes límites de tamaño de pantalla, visualización y rendimiento que un dispositivo móvil.

- **Patrones de descubrimiento de servicios**: estas aplicaciones y servicios de ayuda se encuentran entre sí. En una arquitectura de microservicios, las instancias de servicio cambian dinámicamente debido al escalado, las actualizaciones, la anomalía de servicio e incluso la terminación del servicio. Estos patrones proporcionan mecanismos de descubrimiento para gestionar esta transitoriedad.

- **Patrones de microservicios de adaptador**: pensemos en estos patrones como en adaptadores de enchufes o USB que utilizamos a diario, su finalidad es ayudar a convertir relaciones entre clases u objetos que de otro modo son incompatibles. Una aplicación que se basa en un API externo podría necesitar aplicar este patrón.

- **Patrón de aplicación Strangler**: estos patrones permiten gestionar la refactorización de una aplicación monolítica en aplicaciones de microservicios. El nombre Strangler (estrangulador) hace referencia a cómo una vid (microservicios) estrangula lentamente y con el tiempo un árbol (una aplicación monolítica).

Una vez vistos los principales patrones para desarrollar microservicios correctamente, pasemos a ver aquellos que rápidamente nos crearán problemas.

8. Antipatrones

- **La primera regla de los microservicios es "no crear microservicios"**: o más concretamente, no empezar con microservicios. Los microservicios son una forma de gestionar la complejidad una vez que las aplicaciones se han vuelto demasiado grandes y difíciles de mantener. Hasta que esta dificultad no se presente, no estaremos tratando con un monolito que necesita refactorización.

- **Crear microservicios de un tamaño adecuado**: si nos pasamos con el "micro", podría sobrecargar fácilmente el sistema de comunicación y aumentar la complejidad, cosa que claramente no nos beneficia. Es mejor inclinarse por servicios más grandes y solo dividirlos cuando comiencen a desarrollar problemas que los microservicios puedan resolver, es decir, que cada vez sea más difícil y lento desplegar cambios, que un modelo de datos común se esté volviendo demasiado complejo, o que diferentes partes del servicio tengan diferentes requisitos de carga/escala.

- **No confundir los microservicios con SOA**: los microservicios y la arquitectura orientada a servicios (SOA) a menudo se confunden entre sí, dado que en su nivel más básico ambos están interesados en crear componentes individuales reutilizables que puedan consumirse por otras aplicaciones. La diferencia entre los microservicios y SOA es que los proyectos de microservicios suelen implicar la refactorización de una aplicación para que sea más fácil de gestionar, mientras que SOA cambia la forma en la que los servicios informáticos trabajan en toda la empresa.

- **No intentar ser Netflix**: Netflix fue uno de los pioneros de la arquitectura de microservicios al crear y gestionar una aplicación que englobaba en el momento, un tercio de todo el tráfico de Internet, una especie de tormenta perfecta que les obligaba a crear una gran cantidad de código personalizado y servicios que son innecesarios para la aplicación media actual. Se recomienda, para empezar, evitar la complejidad y utilizar tantas herramientas predefinidas como sea posible.

9. Referencias

https://www.ibm.com/es-es/cloud/learn/microservices#toc-tecnologas-kxuIbw7P

https://en.wikipedia.org/wiki/Microservices#Introduction

https://es.wikipedia.org/wiki/Arquitectura_de_microservicios

https://www.redhat.com/es/topics/microservices

https://www.redhat.com/es/topics/microservices/what-are-microservices#m%C3%A9todo-para-dise%C3%B1ar-aplicaciones

https://www.vmware.com/es/topics/glossary/content/microservices.html

Microservices explained in 5 minutes

¿Qué es la Arquitectura de Microservicios?