Muñoz Alonso Gustavo

MICROSERVICIOS

Son una arquitectura de diseño de software con la que una aplicación se construye como un conjunto de pequeños servicios independientes, cada uno de los cuales se enfoca en una tarea específica y tiene su propia lógica de negocio y base de datos. Los microservicios se comunican entre sí a través de interfaces bien definidas, como API REST o mensajería.

Tambien sirven para descomponer una aplicación monolítica en componentes más pequeños y manejables que pueden desarrollarse, desplegarse y escalar de forma independiente. Algunas de las ventajas de utilizar microservicios son:

1. *Escalabilidad:* Puedes escalar cada servicio de forma independiente según sus necesidades, y da un uso más eficiente de los recursos.
2. *Mantenimiento y desarrollo más sencillos*: Al dividir una aplicación en servicios más pequeños, el mantenimiento y el desarrollo se vuelven más manejables. Los equipos pueden trabajar en diferentes servicios de forma paralela y realizar actualizaciones o mejoras sin afectar a toda la aplicación.
3. *Resistencia y robustez:* Si un microservicio falla, no afecta necesariamente al funcionamiento de otros servicios. Esto hace que el sistema en su conjunto sea más resistente y robusto.
4. *Tecnología adecuada para cada tarea*: permiten utilizar la tecnología más adecuada para cada servicio, ya que no estás limitado por una única tecnología para toda la aplicación.

Para implementar microservicios, puedes seguir varios enfoques, pero algunos pasos comunes incluyen:

- *Definir los límites del dominio:* Identificar y dividir las funcionalidades de la aplicación en servicios individuales. Cada servicio debe tener una única responsabilidad bien definida.
- *Definir interfaces claras:* Establecer interfaces claras y bien definidas para que los servicios se comuniquen entre sí. Esto puede incluir el uso de API REST, mensajería basada en eventos, o incluso comunicación síncrona a través de llamadas de procedimiento remoto (RPC).
- *Implementar y desplegar los servicios:* Desarrollar cada servicio de forma independiente y desplegarlo en un entorno adecuado. Puedes utilizar contenedores (como Docker) para encapsular y desplegar los servicios de manera más fácil y consistente.
- *Gestionar la comunicación entre servicios:* Utilizar herramientas como balanceadores de carga, gateways API, sistemas de descubrimiento de servicios y circuit breakers para gestionar la comunicación entre los diferentes microservicios de manera eficiente y confiable.

- *Monitorizar y mantener:* Implementar herramientas de monitorización y registro para asegurar el rendimiento y la disponibilidad de los servicios. Además, es importante mantener actualizados y seguros los servicios a lo largo del tiempo.

Pueden clasificarse de diversas formas según diferentes criterios, como su funcionalidad, su implementación tecnológica o su grado de acoplamiento. Los más comúnmente utilizados son:

MICROSERVICIOS DE DOMINIO ESPECÍFICO

Se centran en una tarea o funcionalidad específica dentro del dominio de la aplicación. Por ejemplo, un servicio de autenticación, un servicio de gestión de usuarios o un servicio de procesamiento de pagos.

MICROSERVICIOS DE DATOS

Se utilizan para gestionar y proporcionar acceso a los datos de la aplicación. Pueden incluir servicios de almacenamiento y recuperación de datos, servicios de bases de datos específicas (como un servicio de bases de datos NoSQL para datos no estructurados) o servicios de búsqueda y análisis de datos.

MICROSERVICIOS DE BACKEND

Se encargan de la lógica de negocio y la funcionalidad principal de la aplicación. Pueden incluir servicios de procesamiento de transacciones, servicios de generación de informes o servicios de gestión de recursos.

MICROSERVICIOS DE INFRAESTRUCTURA

Proporcionan funcionalidades relacionadas con la infraestructura subyacente de la aplicación, como servicios de monitorización, servicios de registro, servicios de gestión de configuración o servicios de orquestación de contenedores.

MICROSERVICIOS DE INTEGRACIÓN

Se utilizan para integrar diferentes sistemas y servicios dentro de la arquitectura de la aplicación. Pueden incluir servicios de traducción de protocolos, servicios de mensajería o servicios de integración de sistemas externos.

Muñoz Alonso Gustavo

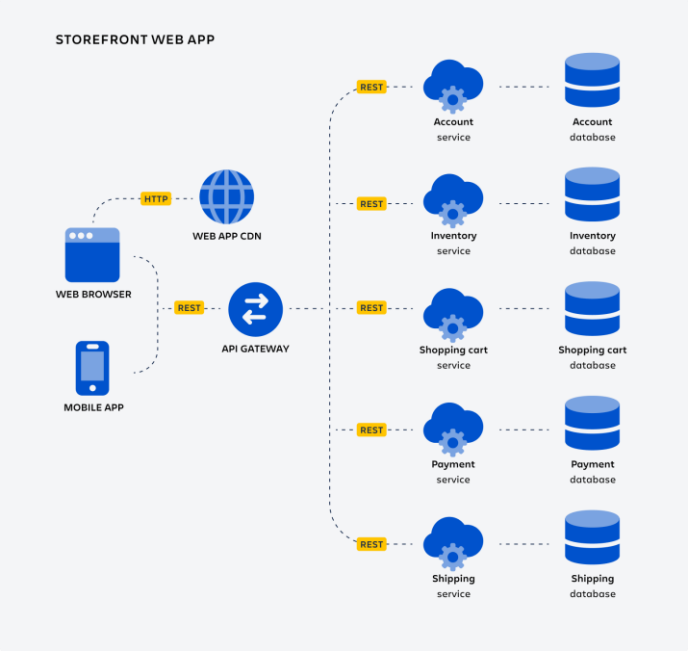
MICROSERVICIOS DE GESTIÓN DE IDENTIDAD Y SEGURIDAD

Se utilizan para gestionar la autenticación, la autorización y otras funcionalidades relacionadas con la seguridad dentro de la aplicación. Pueden incluir servicios de autenticación única (SSO), servicios de gestión de tokens de acceso o servicios de cifrado y firma digital.

EJEMPLO DE ARQUITECTURA DE MICROSERVICIOS

Para un proyecto de software de comercio electrónico, con una aplicación web y una aplicación móvil que interactúan con varios microservicios, cada uno de los cuales proporciona funciones específicas para un dominio.

Las aplicaciones web modernas se ejecutan en navegadores y, a menudo, se sirven desde *una red de distribución de contenido (CDN)*. Las CDN proporcionan la ventaja de poder distribuir aplicaciones web a servidores de todo el mundo, para que los navegadores web las puedan descargar rápidamente. También se utilizan para ofrecer recursos multimedia, como imágenes, audio y vídeo. Por ejemplo, en este sistema las imágenes y los vídeos de los productos a la venta se sirven desde la CDN.



Los microservicios de este gráfico son los siguientes:

- Servicio de cuentas: el cual proporciona información sobre la cuenta del cliente, como la dirección y la información de pago.
- Servicio de inventario: que ofrece información de inventario actualizada sobre los bienes que el cliente puede comprar.
- Servicio de carrito de la compra: es usado por los clientes para seleccionar los productos del inventario que quieren comprar.

- Servicio de pago: se usa para pagar por los productos que han añadido al carrito de la compra.
- Servicio de envío: se programa el embalaje y la entrega de los bienes adquiridos.

Las aplicaciones interactúan con los microservicios a través de las API de REST que publica cada uno de los microservicios. Una puerta de enlace de API permite que las aplicaciones se basen en las API proporcionadas por los microservicios y permite que se intercambien unos microservicios por otros con la misma API.

Cada microservicio se compone de un servicio y una base de datos. Los servicios gestionan la API de REST, implementan la lógica empresarial y almacenan datos en una base de datos. Los recursos de los distintos microservicios, como bases de datos y colas, se aíslan siguiendo el contrato de *12 Factor App*(Metodologia para crear aplicaciones como servicio)o conocido como aplicación de 12 factores.

PARA CREAR UN MICROSERVICIO

Muchas organizaciones empiezan con una arquitectura monolítica. Luego, hay que dividir una base de código en varios servicios, para implementar los patrones correctos para fallar de forma limpia y recuperarte de las incidencias en la red, lidiar con la coherencia de los datos, supervisar la carga de servicio, etc. Y esto teniendo en cuenta solo la parte técnica. También hay que reorganizar los equipos y, muy probablemente, adoptar una cultura de DevOps. Luego viene la parte difícil: descomponer el monolito en microservicios que es la refactorización de un esquema de base de datos monolítico puede ser una operación delicada. Es importante determinar con claridad qué conjuntos de datos necesita cada servicio y las superposiciones. La entrega continua ayuda a reducir los riesgos de fallos de publicación, así como a conseguir que el equipo se centre en crear y ejecutar la aplicación, en lugar de quedarse atascado en implementarla.

COMPARACIÓN ENTRE LA ARQUITECTURA MONOLÍTICA Y LA ARQUITECTURA DE MICROSERVICIOS

Una arquitectura monolítica es un modelo tradicional de un programa de software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. Una arquitectura de microservicios es un método que se basa en una serie de servicios que se pueden implementar de forma independiente. La arquitectura monolítica puede resultar práctica al principio de un proyecto para aliviar la sobrecarga cognitiva de la gestión de código, así como la implementación. Pero una vez que una aplicación monolítica se vuelve grande y compleja, resulta difícil escalarla, la implementación continua pasa a ser un desafío y las actualizaciones pueden resultar complicadas.

Si bien una aplicación monolítica se crea como una sola unidad indivisible, los microservicios dividen esa unidad en una colección de unidades independientes que contribuyen a un todo más amplio. Una aplicación se construye como una serie de servicios que se pueden implementar de forma independiente, están descentralizados y se desarrollan de forma autónoma.

Muñoz Alonso Gustavo

SPRING - MVS

Es un marco de trabajo de desarrollo de aplicaciones web para Java. Spring MVC es parte del ecosistema de Spring Framework y proporciona una arquitectura de patrón Modelo-Vista-Controlador (MVC) para construir aplicaciones web robustas y escalables.

Aquí te explico brevemente cada parte del acrónimo MVC:

1. Modelo (Model): Representa los datos y la lógica de negocio de la aplicación. El modelo generalmente consta de clases Java que encapsulan la lógica de la aplicación y los datos.

2. Vista (View): Es la interfaz de usuario de la aplicación. Puede ser una página HTML, un fragmento de una página, o cualquier otro tipo de interfaz que los usuarios vean e interactúen.

3. Controlador (Controller): Es el componente que maneja las solicitudes del usuario y coordina la interacción entre el modelo y la vista. En Spring MVC, los controladores son clases Java que reciben solicitudes HTTP, procesan los datos proporcionados por el cliente, interactúan con el modelo correspondiente y devuelven la vista apropiada.

Para implementar Spring MVC, se debe de hacer:

1. Configurar un proyecto Spring MVC: Puedes usar herramientas como Spring Initializar o configurar manualmente un proyecto Maven o Gradle con las dependencias adecuadas de Spring MVC.

2. Definir el modelo: Crea las clases Java que representan los datos y la lógica de negocio de tu aplicación.

3. Crear los controladores: Define las clases de controlador que manejarán las solicitudes HTTP. Estas clases anotadas con `@Controller` o `@RestController` contienen métodos anotados con `@RequestMapping` o `@GetMapping`, `@PostMapping`, etc., que mapean las solicitudes a métodos específicos.

4. Configurar las vistas: Define las vistas utilizando tecnologías como JSP, Thymeleaf, FreeMarker, etc. Las vistas se resolverán y mostrarán al usuario según lo especificado por los controladores.

5. Configurar el archivo de configuración de Spring: Configura el archivo de configuración de Spring (`applicationContext.xml` o usando anotaciones `@Configuration`) para definir los beans de Spring necesarios, como los beans del controlador y la configuración del enrutamiento.

6. Desplegar y ejecutar la aplicación: Despliega la aplicación en un servidor web compatible con Java, como Tomcat, Jetty, etc., y accede a ella a través de un navegador web.