UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACION III
BACK END

Unidad Temática N°4:
Microservicios con SpringBoot

Material de Estudio
2° Año – 3° Cuatrimestre

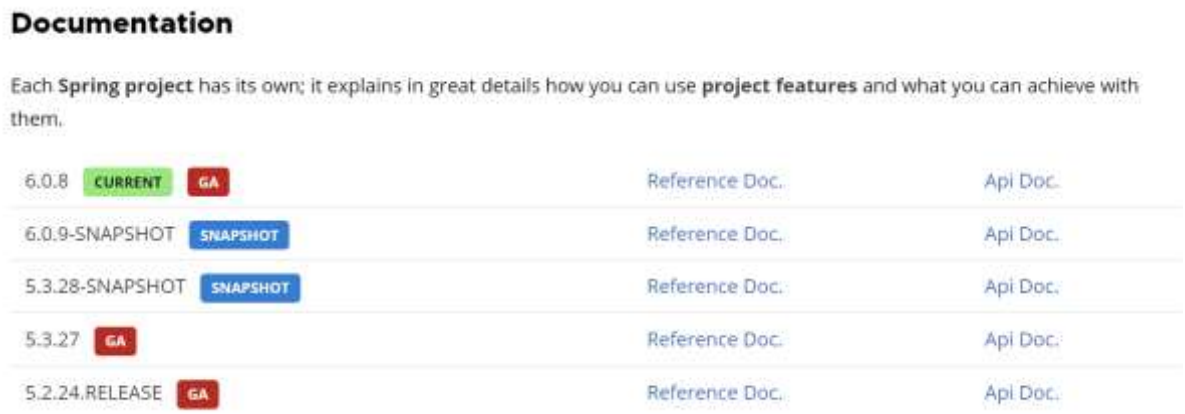
Índice

Introducción a Spring Framework

Spring Framework proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación.

Un elemento clave de Spring es el soporte de infraestructura a nivel de aplicación: Spring se enfoca en la "plumbing" de las aplicaciones empresariales para que los equipos puedan enfocarse en la lógica de negocios a nivel de aplicación, sin ataduras innecesarias a entornos de implementación específicos.

La documentación oficial del sitio se encuentra en la siguiente URL (https://spring.io/projects/spring-framework#learn). Estas son las versiones disponibles al momento de la confección de este documento:



Overview

Spring facilita la creación de aplicaciones empresariales Java. Proporciona todo lo que necesita para adoptar el lenguaje Java en un entorno empresarial, con soporte para Groovy y Kotlin como lenguajes alternativos en JVM y con la flexibilidad para crear muchos tipos de arquitecturas según las necesidades de una aplicación. **A partir de Spring Framework 6.0, Spring requiere Java 17+**.

Spring admite una amplia gama de escenarios de aplicación. En una gran empresa, las aplicaciones suelen existir durante mucho tiempo y tienen que ejecutarse en un JDK y un servidor de aplicaciones cuyo ciclo de actualización está fuera del control del desarrollador. Otros pueden ejecutarse como un solo contenedor con el servidor integrado, posiblemente en un entorno de nube. Sin embargo, otras pueden ser aplicaciones independientes (como cargas de trabajo por lotes o de integración) que no necesitan un servidor.

Spring es de código abierto. Tiene una comunidad grande y activa que proporciona comentarios continuos basados en una amplia gama de casos de uso

del mundo real. Esto ha ayudado a Spring a evolucionar con éxito durante mucho tiempo.

¿Qué quiere decir Spring?

El término "Spring" significa diferentes cosas en diferentes contextos. Se puede usar para referirse al proyecto Spring Framework en sí, que es donde comenzó todo. Con el tiempo, otros proyectos de Spring se han construido sobre Spring Framework. La mayoría de las veces, cuando las personas dicen "Spring", se refieren a toda la familia de proyectos. Esta documentación de referencia se centra en la base: Spring Framework en sí.

Filosofía de diseño

Cuando aprende sobre un marco, es importante saber no solo lo que hace, sino también qué principios sigue. Estos son los principios que rigen en Spring Framework:

- **Ofrece opciones en todos los niveles**. Spring le permite aplazar las decisiones de diseño lo más tarde posible. Por ejemplo, puede cambiar de proveedor de persistencia a través de la configuración sin cambiar su código. Lo mismo es cierto para muchas otras preocupaciones de infraestructura e integración con API de terceros.
- **Acomodar diversas perspectivas**. Spring adopta la flexibilidad y no tiene opiniones sobre cómo se deben hacer las cosas. Es compatible con una amplia gama de necesidades de aplicaciones con diferentes perspectivas.
- **Mantenga una fuerte compatibilidad con versiones anteriores**. La evolución de Spring se ha manejado cuidadosamente para forzar pocos cambios importantes entre versiones. Spring admite una gama cuidadosamente seleccionada de versiones de JDK y bibliotecas de terceros para facilitar el mantenimiento de aplicaciones y bibliotecas que dependen de Spring.
- **Preocúpate por el diseño de la API**. El equipo de Spring dedica mucho tiempo y pensamiento a crear API que sean intuitivas y que se mantengan en muchas versiones y muchos años.
- **Establezca altos estándares para la calidad del código**. Spring Framework pone un fuerte énfasis en javadoc significativo, actual y preciso. Es uno de los pocos proyectos que puede reclamar una estructura de código limpia sin dependencias circulares entre paquetes.

Getting Started

Si recién está comenzando con Spring, es posible que desee comenzar a usar Spring Framework creando una aplicación basada en Spring Boot . Spring Boot proporciona una forma rápida (y obstinada) de crear una aplicación basada en Spring lista para producción. Se basa en Spring Framework, favorece la convención sobre la configuración y está diseñado para ponerlo en funcionamiento lo más rápido posible.

Puede usar start.spring.io para generar un proyecto básico o seguir una de las guías de "Introducción" , como Introducción a la creación de un servicio web RESTful . Además de ser más fáciles de digerir, estas guías están muy enfocadas en tareas y la mayoría de ellas están basadas en Spring Boot. También cubren otros proyectos de la cartera de Spring que quizás desee considerar al resolver un problema en particular.

Introducción a Spring Boot

Spring Boot facilita la creación de aplicaciones basadas en Spring independientes y de grado de producción que puede "**simplemente ejecutar**".

Tomamos el dogma de la plataforma Spring y las bibliotecas de terceros para que pueda comenzar con el mínimo esfuerzo. La mayoría de las aplicaciones de Spring Boot necesitan una configuración mínima de Spring.

¿Qué es Spring Boot y por qué usarlo?

Java Spring Framework (Spring Framework) es una popular estructura empresarial de código abierto para crear aplicaciones independientes de nivel de producción que se ejecutan en la máquina virtual Java (JVM).

Java Spring Boot (Spring Boot) es una herramienta que hace que el desarrollo de aplicaciones web y microservicios con Spring Framework sea más rápido y fácil a través de tres funcionalidades principales:

- Configuración automática
- Un enfoque obstinado de la configuración
- La capacidad de crear aplicaciones independientes

Estas características funcionan juntas para brindarle una herramienta que le permite configurar una aplicación basada en Spring con una configuración y preparación mínimas.

A pesar de lo completo que es Spring Framework y de las funcionalidades que presenta, hacen falta un tiempo considerable y conocimientos suficientes para configurar, instalar e implementar las aplicaciones Spring. Spring Boot alivia este esfuerzo con tres funcionalidades importantes.

Configuración automática

La configuración automática significa que las aplicaciones se inicializan con dependencias preestablecidas que no tienen que configurarse manualmente. Como Java Spring Boot viene con funciones de configuración automática integradas, configura automáticamente tanto el Spring Framework subyacente como los paquetes de terceros según su configuración (y según las mejores prácticas, lo que ayuda a evitar errores).

Aunque puede anular estos valores predeterminados una vez que se completa la inicialización, la función de configuración automática de Java Spring Boot le permite comenzar a desarrollar sus aplicaciones basadas en Spring rápidamente y reduce la posibilidad de errores humanos.

Enfoque obstinado

Spring Boot utiliza un enfoque obstinado para agregar y configurar dependencias de inicialización, según las necesidades de su proyecto. Siguiendo su propio criterio, Spring Boot elige qué paquetes instalar y qué valores predeterminados usar, en lugar de pedirle que tome todas esas decisiones usted mismo y configure todo manualmente.

Puede definir las necesidades de su proyecto durante el proceso de inicialización, durante el cual elige entre varias dependencias de inicio, denominadas Spring Starters, que cubren casos de uso típicos. La ejecución de Spring Boot Initializr se realiza mediante un simple formulario web, sin programación.

Por ejemplo, la dependencia de iniciador "Spring web" permite crear aplicaciones web basadas en Spring con una configuración mínima al agregar todas las dependencias necesarias, como el servidor web Apache Tomcat, a su proyecto. "Spring Security" es otra dependencia de inicialización popular que agrega automáticamente funciones de autenticación y control de acceso a su aplicación.

Spring Boot incluye más de 50 Spring Starters, y hay muchos más starters de terceros disponibles.

Aplicaciones independientes

Spring Boot ayuda a los desarrolladores a crear aplicaciones que simplemente se ejecutan. Específicamente, le permite crear aplicaciones independientes que se ejecutan por sí mismas, sin depender de un servidor web externo, al integrar un servidor web como Tomcat o Netty en su aplicación durante el proceso de inicialización. Como resultado, puede iniciar su aplicación en cualquier plataforma con el comando Ejecutar (puede renunciar esta función para crear aplicaciones sin un servidor web integrado).

Creación de proyectos con Spring Boot

La creación de proyectos con Spring Boot es un proceso sencillo y rápido que nos permite desarrollar aplicaciones Java de manera eficiente y con una configuración mínima. Spring Boot, como extensión del popular framework Spring, nos brinda una serie de características y herramientas que facilitan el desarrollo y despliegue de aplicaciones robustas.

Para comenzar, es necesario tener instalado Java Development Kit (JDK) en nuestra máquina. A continuación, podemos utilizar diferentes herramientas para crear un proyecto con Spring Boot, como Spring Initializr o el complemento Spring Boot CLI.

Spring Initializr es una herramienta en línea que nos permite generar la estructura básica de nuestro proyecto de Spring Boot. Podemos acceder a ella a través de un navegador web y seleccionar las dependencias y configuraciones específicas que necesitamos para nuestro proyecto. Una vez configurado, descargamos el proyecto generado en formato ZIP y lo importamos en nuestro entorno de desarrollo integrado (IDE) preferido, como IntelliJ IDEA o Eclipse.

Otra opción es utilizar el complemento Spring Boot CLI, que nos permite crear y ejecutar proyectos de Spring Boot desde la línea de comandos. Con esta herramienta, podemos generar un nuevo proyecto utilizando el comando spring init, especificando las dependencias y configuraciones necesarias. Una vez creado, podemos importar el proyecto en nuestro IDE o trabajar con él directamente desde la línea de comandos utilizando el comando spring run.

Independientemente de la opción elegida, al crear un proyecto con Spring Boot, obtendremos una estructura base que incluye un archivo de configuración principal, por lo general llamado Application.java o similar, donde se encuentra la clase principal que inicia la aplicación. Además, se incluyen archivos de configuración predeterminados, como application.properties o application.yml, donde se pueden

definir propiedades específicas de la aplicación, como la configuración de la base de datos o la configuración del servidor web.

Una vez que tenemos nuestro proyecto creado, podemos comenzar a desarrollar nuestras funcionalidades utilizando los componentes y características de Spring Boot. Podemos aprovechar la autoconfiguración de Spring Boot, que analiza las dependencias y configura automáticamente gran parte del entorno de ejecución de nuestra aplicación. Además, Spring Boot proporciona un conjunto de iniciadores (starters) que simplifican la incorporación de tecnologías y frameworks populares, como Spring Data JPA para el acceso a la base de datos, Spring Security para la seguridad de la aplicación, o Thymeleaf para el desarrollo de vistas web.

Configuración de propiedades

En Spring Boot, la configuración de propiedades juega un papel fundamental para personalizar el comportamiento de una aplicación. Spring Boot ofrece un enfoque flexible y potente para administrar la configuración mediante el uso de propiedades externas. Estas propiedades permiten ajustar el comportamiento de la aplicación sin necesidad de modificar el código fuente.

Spring Boot admite diferentes fuentes de configuración de propiedades, como archivos de propiedades, archivos YAML, variables de entorno y argumentos de línea de comandos. Esto permite que la configuración de una aplicación sea externa y fácilmente modificable sin la necesidad de recompilar el código.

La configuración de propiedades en Spring Boot sigue una jerarquía de precedencia que determina qué valores de configuración se aplican en caso de haber configuraciones en múltiples fuentes. Esta jerarquía de precedencia permite que las configuraciones sean personalizadas en diferentes entornos sin afectar al código base.

Una forma común de configurar propiedades en Spring Boot es a través de archivos de propiedades o archivos YAML. Estos archivos contienen pares clave-valor que definen diferentes aspectos de la aplicación, como la configuración de la base de datos, la configuración del servidor, las URLs de servicios externos, entre otros. Spring Boot busca automáticamente estos archivos en ubicaciones predefinidas, como el directorio de recursos del proyecto, y los carga durante el inicio de la aplicación.

Además de los archivos de propiedades y YAML, también es posible configurar propiedades utilizando variables de entorno. Spring Boot proporciona una integración transparente con variables de entorno, lo que permite que las

propiedades se obtengan automáticamente de variables de entorno existentes en el sistema.

Otra opción es utilizar argumentos de línea de comandos para configurar propiedades. Spring Boot permite pasar valores de configuración como argumentos al ejecutar la aplicación, lo que brinda flexibilidad y la posibilidad de ajustar la configuración según las necesidades específicas de cada ejecución.

Ejecución y despliegue de aplicaciones Spring Boot

Spring Boot proporciona diferentes opciones para ejecutar y desplegar aplicaciones, permitiendo adaptarse a diferentes entornos y requisitos de implementación.

Una de las formas más comunes de ejecutar una aplicación Spring Boot es mediante el uso de un archivo JAR ejecutable. Spring Boot permite empaquetar una aplicación en un archivo JAR que incluye todas las dependencias necesarias y el servidor de aplicaciones integrado. Esto significa que la aplicación se puede ejecutar simplemente con el comando **java -jar** en la línea de comandos, sin necesidad de instalar y configurar un servidor de aplicaciones externo.

Otra opción para ejecutar una aplicación Spring Boot es mediante el uso de un contenedor servlet, como Apache Tomcat o Jetty. Spring Boot permite generar un archivo WAR que puede ser desplegado en un contenedor servlet estándar. Esto proporciona mayor flexibilidad si se desea implementar la aplicación en un entorno que ya tiene un servidor de aplicaciones configurado.

Además de estas opciones tradicionales, Spring Boot también admite la ejecución y despliegue de aplicaciones en la nube. Spring Boot es compatible con plataformas de nube populares, como Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP). Esto permite aprovechar las características y servicios de estas plataformas para implementar aplicaciones de manera eficiente y escalable.

Independientemente de la opción de ejecución y despliegue elegida, Spring Boot proporciona características adicionales para facilitar la gestión y supervisión de la aplicación en tiempo de ejecución. Por ejemplo, Spring Boot Actuator es un módulo que proporciona endpoints y métricas para supervisar el estado y el rendimiento de la aplicación. Esto permite monitorear la salud de la aplicación, recopilar métricas y realizar tareas de administración de forma fácil y conveniente.

Desarrollo de aplicaciones con Spring Boot

El desarrollo de aplicaciones con Spring Boot ofrece una forma rápida y eficiente de crear aplicaciones Java robustas y de alta calidad. Spring Boot es una extensión del popular framework Spring, diseñado para simplificar el desarrollo y la configuración de aplicaciones empresariales. Los desarrolladores pueden enfocarse en la implementación de funcionalidades de valor sin tener que preocuparse por la configuración tediosa y repetitiva.

Además, Spring Boot proporciona un conjunto de iniciadores (starters) que facilitan la incorporación de tecnologías y frameworks populares, como Spring Data JPA y Spring Security. Estos iniciadores permiten integrar rápidamente funcionalidades comunes en la aplicación y acelerar el proceso de desarrollo.

Desarrollo de API RESTful con Spring Boot

El desarrollo de API RESTful con Spring Boot es una forma eficiente y poderosa de construir servicios web escalables y de alto rendimiento. Spring Boot proporciona un conjunto de herramientas y características que simplifican el proceso de creación de API RESTful, permitiendo a los desarrolladores implementar rápidamente servicios web modernos y flexibles.

Una API RESTful es una interfaz de programación de aplicaciones que sigue los principios del estilo arquitectónico REST (Representational State Transfer). Este estilo de arquitectura se basa en la utilización de los verbos HTTP (GET, POST, PUT, DELETE, etc.) para realizar operaciones en recursos, que pueden ser representados y manipulados a través de URLs.

Spring Boot ofrece una integración perfecta con Spring MVC, el módulo de Spring para el desarrollo de aplicaciones web. Spring MVC proporciona un modelo de programación basado en anotaciones, lo que facilita la creación de controladores RESTful. Los controladores RESTful en Spring Boot son responsables de recibir las peticiones HTTP, procesarlas y devolver las respuestas adecuadas. Además de las características básicas de Spring MVC, Spring Boot proporciona iniciadores (starters) específicos para la creación de API RESTful. Estos iniciadores incluyen dependencias y configuraciones predefinidas para facilitar la implementación de funcionalidades comunes en una API RESTful, como la serialización y deserialización de objetos JSON, la validación de datos, el manejo de excepciones, la autenticación y la autorización.

Spring Boot también ofrece integración con otras bibliotecas y tecnologías populares para el desarrollo de API RESTful. Por ejemplo, puede utilizar Spring Data

JPA para acceder a una base de datos de manera sencilla y realizar operaciones CRUD en los recursos de la API. También puede integrar la documentación de la API utilizando herramientas como Swagger o Springfox, lo que facilita la comprensión y el consumo de la API por parte de otros desarrolladores.

Documentación de API con Swagger

La documentación de API con Swagger es una forma popular y efectiva de documentar y visualizar una API de manera clara y comprensible. Swagger es un conjunto de herramientas que permite describir, construir y documentar API de manera sencilla y automatizada.

La documentación de una API es esencial para facilitar su uso por parte de otros desarrolladores. Proporciona información detallada sobre los endpoints, los parámetros requeridos, las respuestas esperadas y cualquier otra información relevante para interactuar correctamente con la API.

Swagger utiliza una especificación llamada OpenAPI, anteriormente conocida como Swagger Specification, para describir la estructura y el funcionamiento de una API. Esta especificación utiliza un formato legible por humanos y máquinas, generalmente en formato JSON o YAML, para definir los endpoints, los métodos HTTP, los parámetros y las respuestas de la API.

La herramienta Swagger UI es una interfaz de usuario interactiva que se genera automáticamente a partir de la especificación OpenAPI. Esta interfaz permite visualizar y explorar la documentación de la API de manera intuitiva. Los desarrolladores pueden ver los endpoints disponibles, probarlos directamente desde la interfaz y obtener información detallada sobre cada uno.

Además de la visualización interactiva, Swagger también permite generar automáticamente código cliente en diferentes lenguajes de programación a partir de la especificación OpenAPI. Esto facilita la implementación de clientes para consumir la API, ya que gran parte del código se genera automáticamente según la descripción de la API.

Una de las ventajas de Swagger es su integración con Spring Boot. Springfox es una biblioteca de integración que permite generar automáticamente la especificación OpenAPI y la interfaz de usuario Swagger UI a partir de las anotaciones en el código fuente de una aplicación Spring Boot. Esto significa que simplemente al agregar las anotaciones adecuadas a los controladores de la API, se generará automáticamente la documentación correspondiente.

Manipulación de datos utilizando Spring Data JPA y Spring Boot

La manipulación de datos utilizando Spring Data JPA y Spring Boot es una combinación poderosa que simplifica y agiliza el acceso a la base de datos en aplicaciones Java. Spring Data JPA es un módulo de Spring que proporciona una capa de abstracción sobre el framework JPA (Java Persistence API), permitiendo realizar operaciones CRUD (Create, Read, Update, Delete) de manera sencilla y eficiente.

Spring Data JPA utiliza anotaciones y convenciones para mapear las entidades de Java a las tablas de la base de datos. Esto significa que los desarrolladores pueden definir clases Java como entidades y Spring Data JPA se encargará de crear y mantener las tablas correspondientes en la base de datos.

Al utilizar Spring Boot junto con Spring Data JPA, se obtiene una configuración automática y una integración fluida. Spring Boot proporciona iniciadores (starters) específicos para Spring Data JPA, que incluyen las dependencias necesarias y la configuración predeterminada para comenzar a trabajar rápidamente con la persistencia de datos.

Para realizar operaciones CRUD en la base de datos, Spring Data JPA ofrece una interfaz de repositorio. Los repositorios en Spring Data JPA son interfaces que extienden la interfaz JpaRepository, y mediante la definición de métodos, permiten realizar operaciones como guardar, actualizar, eliminar y buscar entidades en la base de datos. Estas operaciones son implementadas automáticamente por Spring Data JPA, lo que evita la necesidad de escribir consultas SQL manualmente.

Además de las operaciones básicas de CRUD, Spring Data JPA también permite definir consultas personalizadas utilizando consultas con nombre o consultas nativas en SQL. Esto brinda flexibilidad para realizar consultas más complejas y específicas según los requisitos del proyecto. También ofrece funcionalidades adicionales, como la paginación de resultados, el ordenamiento, la especificación de criterios de búsqueda y la gestión automática de transacciones. Estas características facilitan el desarrollo de aplicaciones que manejan grandes volúmenes de datos y requieren un acceso eficiente y escalable a la base de datos.

Manipulación de datos utilizando Spring JDBC y Spring Boot

La manipulación de datos utilizando Spring JDBC y Spring Boot es una forma eficiente y flexible de interactuar con una base de datos relacional en aplicaciones Java. Spring JDBC es un módulo de Spring que proporciona un conjunto de clases y

utilidades para simplificar el acceso y la manipulación de datos a través de JDBC (Java Database Connectivity).

JDBC es una API estándar de Java que permite a las aplicaciones interactuar con bases de datos relacionales mediante consultas SQL. Spring JDBC se basa en JDBC y agrega características adicionales y simplificaciones para facilitar el desarrollo de aplicaciones que acceden a la base de datos.

Con Spring Boot, la configuración y el uso de Spring JDBC se simplifican aún más. Spring Boot proporciona iniciadores (starters) específicos para Spring JDBC, que incluyen las dependencias necesarias y la configuración predeterminada para comenzar a trabajar rápidamente con la manipulación de datos.

Para realizar operaciones de manipulación de datos utilizando Spring JDBC, se utilizan clases como JdbcTemplate y NamedParameterJdbcTemplate. Estas clases proporcionan métodos convenientes para ejecutar consultas SQL, realizar inserciones, actualizaciones y eliminaciones de registros en la base de datos.

Además, Spring JDBC permite la integración con el mapeo objeto-relacional (ORM) a través de bibliotecas como Hibernate. Esto significa que se puede utilizar el mapeo objeto-relacional para representar entidades de Java como objetos y realizar operaciones de persistencia en la base de datos de manera más sencilla. También ofrece características como el manejo de transacciones, el control de concurrencia y la gestión de excepciones relacionadas con la base de datos. Estas características son fundamentales para garantizar la integridad de los datos y la consistencia en aplicaciones que manejan transacciones y concurrencia.

Integración de servicios externos

La integración de servicios externos es un aspecto fundamental en el desarrollo de aplicaciones modernas, ya que permite que una aplicación se comunique y colabore con otros sistemas o servicios externos. Esta integración puede involucrar la comunicación con API web, servicios web SOAP, servicios RESTful, bases de datos externas, servicios de autenticación, servicios de pago, entre otros.

En el contexto de Spring Boot, existen varias herramientas y enfoques que facilitan la integración de servicios externos:

- **Consumo de API RESTful**: Spring Boot proporciona una integración nativa con la biblioteca RestTemplate o, en versiones más recientes, con WebClient. Estas bibliotecas permiten realizar peticiones HTTP a servicios externos, enviar y recibir datos en formatos como JSON o XML, y manejar las respuestas de manera eficiente.
- **Implementación de servicios RESTful**: Spring Boot también permite exponer servicios RESTful propios, los cuales pueden ser consumidos por otras aplicaciones o sistemas externos. Al utilizar anotaciones como @RestController y @RequestMapping, es posible definir endpoints y establecer la lógica de negocio para procesar las solicitudes entrantes.
- **Integración con servicios SOAP**: Spring Boot puede integrarse con servicios web SOAP mediante bibliotecas como Apache CXF o Spring Web Services. Estas bibliotecas permiten consumir y exponer servicios web SOAP, generando automáticamente el código necesario a partir de la descripción del servicio (WSDL) o utilizando anotaciones específicas.
- **Integración con bases de datos externas**: Spring Boot proporciona soporte para la integración con una amplia gama de bases de datos a través de Spring Data JPA o Spring Data JDBC. Esto permite acceder y manipular datos almacenados en bases de datos externas de manera eficiente y segura.
- **Gestión de autenticación y autorización**: Spring Security es un módulo ampliamente utilizado en Spring Boot para gestionar la seguridad en las aplicaciones. Proporciona características como la autenticación de usuarios, la autorización basada en roles y la integración con proveedores de autenticación externos, como OAuth o SAML.
- **Integración de servicios de terceros**: Spring Boot se puede integrar fácilmente con servicios de terceros a través de bibliotecas y herramientas específicas. Por ejemplo, existen bibliotecas para servicios de almacenamiento en la nube, servicios de notificaciones push, servicios de correo electrónico, etc.

Pruebas unitarias y de integración en Spring Boot

Spring Boot proporciona una integración nativa con JUnit, que es uno de los frameworks de pruebas más populares en el ecosistema Java. JUnit permite escribir y ejecutar pruebas unitarias de manera eficiente y estructurada. Spring Boot se integra con JUnit al proporcionar anotaciones específicas para simplificar la configuración y la ejecución de pruebas en el contexto de una aplicación Spring Boot.

Algunas de las anotaciones más comunes que Spring Boot ofrece para las pruebas unitarias son:

- **@RunWith(SpringRunner.class)**: Esta anotación se utiliza para configurar JUnit para que se ejecute con el soporte de Spring. Proporciona una infraestructura especial para la ejecución de pruebas de Spring Boot.
- **@SpringBootTest**: Esta anotación se utiliza para cargar y configurar el contexto de Spring Boot durante las pruebas de integración. Carga todas las configuraciones de la aplicación y permite acceder a los beans administrados por Spring en las pruebas.
- **@MockBean**: Esta anotación se utiliza para crear un objeto simulado (mock) de una dependencia de Spring. Permite simular el comportamiento de los componentes externos y controlar sus respuestas durante las pruebas unitarias.
- **@DataJpaTest**: Esta anotación se utiliza para las pruebas de integración que involucran repositorios de Spring Data JPA. Configura un entorno de prueba específico para las operaciones de persistencia de datos.
- **@WebMvcTest**: Esta anotación se utiliza para las pruebas de integración de controladores REST en aplicaciones web. Configura un entorno de prueba específico para probar los controladores y las solicitudes HTTP.
- Además de las anotaciones, Spring Boot proporciona utilidades y clases de apoyo para simplificar las pruebas unitarias y de integración:
- **TestEntityManager**: Es una interfaz que proporciona métodos convenientes para interactuar con la base de datos durante las pruebas de integración. Permite realizar operaciones CRUD y consultas a la base de datos utilizando JPA.
- **TestRestTemplate**: Es una clase que proporciona un cliente HTTP para realizar solicitudes a la aplicación durante las pruebas de integración. Permite realizar llamadas a los endpoints REST y verificar las respuestas recibidas.

- **@AutoConfigureMockMvc**: Esta anotación se utiliza en las pruebas de integración para configurar automáticamente el objeto MockMvc, que proporciona un entorno de prueba completo para los controladores web.

Estas son solo algunas de las herramientas y funcionalidades que Spring Boot ofrece para facilitar las pruebas unitarias y de integración. Spring Boot integra JUnit de manera nativa y proporciona anotaciones y utilidades específicas para simplificar la configuración y la ejecución de pruebas en el contexto de una aplicación Spring Boot. Esto permite a los desarrolladores realizar pruebas eficientes y estructuradas, garantizando la calidad y el correcto funcionamiento del código desarrollado.

Anotaciones

Spring Boot ofrece una amplia gama de anotaciones que facilitan la configuración y el desarrollo de aplicaciones. Estas anotaciones se utilizan para definir componentes, configurar propiedades, habilitar características específicas y controlar el comportamiento de la aplicación. A continuación, se presentan algunas de las anotaciones más utilizadas en Spring Boot:

@SpringBootApplication

Esta es una anotación compuesta que combina varias anotaciones, incluyendo @Configuration, @EnableAutoConfiguration y @ComponentScan. Se utiliza para marcar la clase principal de la aplicación Spring Boot y habilita la autoconfiguración y el escaneo de componentes.

@Configuration

Esta anotación se utiliza para marcar una clase de configuración en Spring. Indica que la clase contiene métodos que definen la configuración de la aplicación, como la creación de beans y la configuración de componentes.

@ConfigurationProperties

Se utiliza para enlazar automáticamente propiedades de configuración con campos de una clase. Permite una fácil configuración y acceso a las propiedades en un archivo de configuración, como application.properties.

@Controller

Se utiliza para marcar una clase como un controlador web en una aplicación Spring MVC. Indica que la clase es responsable de manejar las solicitudes HTTP y de devolver las respuestas adecuadas.

@RestController

Esta anotación es similar a @Controller, pero se utiliza específicamente para controladores RESTful. Combina la anotación @Controller y @ResponseBody, lo que significa que los métodos en un controlador anotado con @RestController devuelven directamente los datos en lugar de ser resueltos a través de una vista.

@GetMapping

Esta anotación combina las anotaciones @RequestMapping y @RequestMethod.GET. Se utiliza para mapear una solicitud HTTP GET a un método específico en un controlador.

@PostMapping

Esta anotación combina las anotaciones @RequestMapping y @RequestMethod.POST. Se utiliza para mapear una solicitud HTTP POST a un método específico en un controlador.

@PutMapping

Esta anotación combina las anotaciones @RequestMapping y @RequestMethod.PUT. Se utiliza para mapear una solicitud HTTP PUT a un método específico en un controlador.

@DeleteMapping

Esta anotación combina las anotaciones @RequestMapping y @RequestMethod.DELETE. Se utiliza para mapear una solicitud HTTP DELETE a un método específico en un controlador.

@ExceptionHandler

Se utiliza para manejar excepciones específicas en un controlador. Permite definir un método que se ejecutará cuando ocurra una excepción particular, brindando una respuesta adecuada al cliente.

@RequestMapping

Se utiliza para mapear las solicitudes HTTP a métodos específicos en los controladores. Permite definir las URL y los métodos HTTP (GET, POST, PUT, DELETE, etc.) que un controlador debe manejar.

@PathVariable

Utilizada para vincular variables de una URL con los parámetros de un método en un controlador. Permite acceder a valores dinámicos en la URL de una solicitud.

@RequestAttribute

Utilizada para vincular atributos de la solicitud HTTP a los parámetros de un método en un controlador. Permite acceder a los atributos establecidos en la solicitud.

@RequestBody

Esta anotación se utiliza para vincular el cuerpo de una solicitud HTTP con un objeto en un método de un controlador. Permite recibir y procesar datos enviados en el cuerpo de la solicitud, como en una solicitud POST o PUT.

@RequestHeader

Se utiliza para vincular los encabezados de una solicitud HTTP a los parámetros de un método en un controlador. Permite acceder a los valores de los encabezados de la solicitud.

@ResponseStatus

Esta anotación se utiliza para establecer el código de estado de la respuesta HTTP devuelta por un método en un controlador.

@Valid

Se utiliza en parámetros de métodos o argumentos de constructores para activar la validación de datos según las anotaciones de validación de Spring, como @NotNull, @Size, etc.

@Autowired

Utilizada para realizar la inyección de dependencias en Spring. Se aplica a campos, constructores o métodos "setter" y permite que Spring resuelva e inyecte las dependencias necesarias automáticamente.

@Value

Esta anotación se utiliza para inyectar valores de propiedades en los campos de una clase. Permite que las propiedades definidas en un archivo de configuración (como application.properties) sean inyectadas en las variables correspondientes.

@Bean

Esta anotación se utiliza para indicar que un método en una clase de configuración de Spring debe ser tratado como un bean y gestionado por el contenedor de Spring.

@Component

Se utiliza para marcar una clase como un componente de Spring. Indica que la clase es un candidato para ser detectado y configurado automáticamente por Spring.

@Repository

Esta anotación se utiliza para marcar una clase como un componente de acceso a datos. Indica que la clase se encarga de interactuar con una base de datos o un origen de datos.

@Service

Esta anotación se utiliza para marcar una clase como un servicio de negocio. Indica que la clase contiene la lógica de negocio de la aplicación y se utiliza para separar las responsabilidades de la capa de presentación y la capa de acceso a datos.

@Transactional

Esta anotación se utiliza para marcar un método o clase con transacciones. Indica que el método o clase debe ser ejecutado dentro de una transacción. Es comúnmente utilizada en métodos de servicios o de acceso a datos para garantizar la integridad y consistencia de las operaciones.

@Async

Se utiliza para marcar un método como asíncrono. Indica que el método se ejecutará en un hilo separado y no bloqueará la ejecución del hilo principal.

@Scheduled

Esta anotación se utiliza para programar la ejecución de un método a intervalos regulares. Permite configurar tareas programadas dentro de la aplicación.

@Retryable

Se utiliza para marcar un método que debe ser reintentado automáticamente en caso de excepciones específicas. Permite gestionar automáticamente la recuperación de errores transitorios.

@Recover

La anotación define un método de recuperación separado cuando un método @Retryable falla con una excepción especificada

Estas son solo algunas de las anotaciones más comunes utilizadas en Spring Boot. La plataforma Spring ofrece muchas más anotaciones para cubrir una amplia gama de casos de uso, como el manejo de transacciones, la seguridad, la programación asincrónica, entre otros. Estas anotaciones ayudan a simplificar el desarrollo de aplicaciones y promueven una arquitectura modular y flexible.

Anotaciones para integración con Base de Datos

Para integrar una base de datos en una aplicación de Spring Boot, hay varias anotaciones que puedes utilizar para modelar los objetos de la base de datos y establecer la relación entre ellos. Las anotaciones más comunes son:

@Entity

Esta anotación se utiliza para marcar una clase como una entidad de la base de datos. Cada entidad representa una tabla en la base de datos y cada instancia de la entidad representa una fila en esa tabla.

@Table

Esta anotación se utiliza para especificar el nombre de la tabla asociada a una entidad. Puede utilizarse para personalizar el nombre de la tabla y sus columnas, así como para establecer restricciones adicionales.

@Id

Esta anotación se utiliza para marcar una propiedad de una entidad como la clave primaria de la tabla. Generalmente, se utiliza en combinación con la anotación @GeneratedValue para indicar que el valor de la clave primaria será generado automáticamente por la base de datos.

@Column

Esta anotación se utiliza para mapear una propiedad de una entidad a una columna en la tabla de la base de datos. Puede utilizarse para personalizar el nombre de la columna, su tipo de datos y otras características, como la longitud o si es nullable.

@OneToMany

Esta anotación se utiliza para establecer una relación uno a muchos entre dos entidades. Indica que una entidad tiene una colección de otras entidades relacionadas.

@ManyToOne

Esta anotación se utiliza para establecer una relación muchos a uno entre dos entidades. Indica que varias entidades están relacionadas con una sola entidad.

@OneToOne

Esta anotación se utiliza para establecer una relación uno a uno entre dos entidades. Indica que una entidad está relacionada con otra entidad de forma exclusiva.

@Formula

Esta anotación se utiliza para especificar una fórmula SQL personalizada que se calculará y se asignará a una propiedad de una entidad. Puede ser útil cuando necesitas un valor calculado en base a otras propiedades de la entidad.

@ManyToMany

Esta anotación se utiliza para establecer una relación muchos a muchos entre dos entidades. Indica que una entidad puede tener múltiples instancias relacionadas con múltiples instancias de otra entidad.

@JoinColumn

Esta anotación se utiliza para especificar la columna que se utilizará como clave extranjera en una relación entre entidades. Puede utilizarse para personalizar el nombre de la columna o establecer otras características.

@JoinTable

Esta anotación se utiliza para especificar una tabla de unión en una relación muchos a muchos. Permite personalizar el nombre de la tabla de unión y las columnas utilizadas para la relación.

@Transient

Esta anotación se utiliza para marcar una propiedad de una entidad como transitoria, lo que significa que no se persistirá en la base de datos. Puede ser útil cuando necesitas una propiedad calculada o temporal que no requiere almacenamiento persistente.

@Enumerated

Esta anotación se utiliza para mapear una propiedad enumerada a una columna de base de datos. Permite especificar cómo se debe almacenar y recuperar el valor de la propiedad enumerada.

@Temporal

Esta anotación se utiliza para mapear una propiedad de fecha o tiempo a una columna de base de datos. Permite especificar el tipo de temporalidad de la propiedad, como fecha, hora o fecha y hora.

@Version

Esta anotación se utiliza para marcar una propiedad que representa la versión de una entidad. Se utiliza en combinación con una estrategia de control de concurrencia para garantizar la consistencia de los datos.

Microservicios

¿Qué son los microservicios?

Los microservicios (o arquitectura de microservicios) son un enfoque arquitectónico nativo de la nube en el que una sola aplicación se compone de muchos componentes o servicios más pequeños acoplados de forma flexible e independiente. Normalmente, estos servicios:

- tienen su propio lote de tecnología, que incluye la base de datos y el modelo de gestión de datos;
- se comunican entre sí mediante una combinación de API REST, transmisión de eventos y Message Brokers, y
- se organizan por capacidad de negocio, donde la línea que separa los servicios a menudo se denomina un contexto limitado.

Si bien gran parte de la discusión sobre los microservicios ha girado en torno a las definiciones y características arquitectónicas, su valor se entiende mejor a través de unos beneficios empresariales y organizacionales bastante simples:

- El código se puede actualizar más fácilmente: se pueden agregar nuevas características o funcionalidades sin intervenir en toda la aplicación.
- Los equipos pueden utilizar diferentes lotes y diferentes lenguajes de programación para diferentes componentes.
- Los componentes se pueden escalar independientemente uno del otro, lo que reduce los residuos y los costos asociados con tener que escalar aplicaciones enteras porque una única característica podría estar enfrentando demasiada carga.

Los microservicios también pueden entenderse en contraposición a dos arquitecturas de aplicaciones anteriores: la arquitectura monolítica y la arquitectura orientada a servicios (SOA).

La diferencia entre los microservicios y la arquitectura monolítica es que los microservicios componen una única aplicación de muchos servicios más pequeños y poco acoplados, a diferencia del enfoque monolítico de una aplicación grande y de acoplamiento completo.

Las diferencias entre los microservicios y SOA pueden ser un poco menos claras. Si bien se pueden establecer diferencias técnicas entre microservicios y SOA, especialmente en torno al rol del bus de servicio empresarial (ESB), es más fácil considerar la diferencia de ámbito. SOA fue un esfuerzo de toda la empresa para

estandarizar la forma en que todos los servicios web en una organización se comunican e integran entre sí, mientras que la arquitectura de microservicios es específica de la aplicación.

Cómo los microservicios benefician a la organización

Es probable que los microservicios sean al menos tan populares entre los ejecutivos y los líderes del proyecto como con los desarrolladores. Esta es una de las características más inusuales de los microservicios porque el entusiasmo arquitectónico es típicamente reservado para los equipos de desarrollo de software. La razón de esto es que los microservicios reflejan mejor la forma en que muchos líderes empresariales quieren estructurar y ejecutar sus equipos y procesos de desarrollo.

Dicho de otro modo, los microservicios son un modelo arquitectónico que facilita un modelo operativo deseado. En una encuesta reciente de IBM a más de 1,200 desarrolladores y ejecutivos de TI, el 87 % de los usuarios de microservicios afirmó que el gasto y el esfuerzo necesarios para adoptar los microservicios valen la pena.

Estos son sólo algunos de los beneficios empresariales de los microservicios.

Implementables de forma independiente

Tal vez la característica más importante de los microservicios es que, debido a que los servicios son más pequeños e independientemente implementables, ya no requieren una legislación para cambiar una línea de código o añadir una nueva característica en una aplicación.

Los microservicios prometen a las organizaciones un antídoto contra la frustración que provoca que la aplicación de pequeños cambios requiera grandes cantidades de tiempo. No necesita un doctorado en ciencias de la computación para ver o entender el valor de un enfoque que facilita más la velocidad y la agilidad.

Pero la velocidad no es el único valor de diseñar servicios de esta manera. Un modelo organizativo emergente común es reunir equipos multifuncionales en torno a un problema empresarial, servicio o producto. El modelo de microservicios encaja perfectamente con esta tendencia, ya que permite a una organización crear equipos pequeños y multifuncionales en torno a un servicio o una colección de servicios y hacer que funcionen de forma ágil.

El acoplamiento suelto de microservicios también crea un grado de aislamiento de fallas y una mejor resiliencia en las aplicaciones. Y el pequeño tamaño de los servicios, combinado con sus claros límites y patrones de comunicación, hace que sea más fácil para los nuevos miembros del equipo entender la base de código

y contribuir con ella rápidamente, un claro beneficio en términos de la velocidad y moral de los empleados.

La herramienta adecuada para el trabajo

En los patrones de arquitectura tradicionales de n niveles, una aplicación normalmente comparte un lote común, con una gran base de datos relacional que da soporte a toda la aplicación. Este enfoque tiene varios inconvenientes obvios. El más significativo es que cada componente de una aplicación debe compartir un recurso común, un modelo de datos y una base de datos, incluso si hay una herramienta clara y mejor para el trabajo para determinados elementos. Esto perjudica a la arquitectura y es frustrante para los desarrolladores, que son conscientes en todo momento de que hay una forma mejor y más eficaz crear estos componentes.

Por el contrario, en un modelo de microservicios, los componentes se implementan de forma independiente y se comunican a través de una combinación de REST, streaming de eventos y Message Brokers, por lo que es posible que el lote de cada servicio individual esté optimizado para ese servicio. La tecnología cambia todo el tiempo, y es mucho más fácil y menos costoso que una aplicación formada por múltiples servicios más pequeños evolucione con la tecnología a medida que esta avance.

Escalado preciso

Con los microservicios, los servicios individuales se pueden implementar individualmente, pero también se pueden escalar individualmente. Hecho correctamente, los microservicios requieren menos infraestructura que las aplicaciones monolíticas porque permiten el escalado preciso de sólo los componentes que lo requieren, en lugar de toda la aplicación, como es el caso de aplicaciones monolíticas.

También existen algunos desafíos

Los beneficios significativos de los microservicios vienen con desafíos significativos. Cambiar de una aplicación monolítica a los microservicios implica una mayor complejidad de gestión: muchos más servicios, creados por muchos más equipos, implementados en muchos más lugares. Los problemas en un servicio pueden causar, o ser causados por, problemas en otros servicios. Los datos de registro (utilizados para supervisión y resolución de problemas) son más voluminosos y pueden ser inconsistentes entre los servicios. Las nuevas versiones pueden causar problemas de compatibilidad con versiones anteriores. Las aplicaciones requieren más conexiones de red, lo que significa más oportunidades para que ocurran problemas de latencia y conectividad. Un enfoque de DevOps (como se lee más

abajo) puede abordar muchos de estos problemas, pero la adopción de DevOps tiene retos propios.

Sin embargo, estos desafíos no impiden que los no usuarios adopten el uso de microservicios, o que los usuarios aumenten sus compromisos de microservicios. Nuevos datos de una encuesta de IBM revelan que el 56 % de los no usuarios actuales probablemente, o muy probablemente, adoptarán microservicios en los próximos dos años, y el 78 % de los usuarios actuales de microservicios probablemente aumenten el tiempo, dinero y esfuerzo que han invertido en microservicios.

Los microservicios permiten, y requieren, DevOps

La arquitectura de microservicios se describe a menudo como optimizada para DevOps y la integración continua/entrega continua (CI/CD), y en el contexto de pequeños servicios que se pueden implementar con frecuencia, es fácil entender por qué.

Pero otra forma de ver la relación entre los microservicios y DevOps es que las arquitecturas de microservicios realmente requieren DevOps para tener éxito. Mientras que las aplicaciones monolíticas tienen una serie de inconvenientes que se han discutido anteriormente en este artículo, tienen el beneficio de no ser un complejo sistema distribuido con múltiples partes móviles y recursos de tecnología independientes. Por el contrario, dado el aumento masivo de la complejidad, de las partes móviles y de las dependencias que vienen con los microservicios, sería poco prudente adoptar el uso de microservicios sin inversiones significativas en la implementación, la supervisión y la automatización del ciclo de vida.

Tecnologías y herramientas clave

Mientras que casi cualquier herramienta o lenguaje moderno puede ser utilizado en una arquitectura de microservicios, hay un puñado de herramientas básicas que se han vuelto esenciales y prácticamente definitivas para los microservicios:

Contenedores, Docker y Kubernetes

Uno de los elementos clave de un microservicio es que, en general, es bastante pequeño. (No hay una cantidad arbitraria de código que determine si algo es o no un microservicio, pero "micro" está incluido por algo en el nombre).

Cuando Docker inició la era moderna de contenedores en 2013, también introdujo el modelo de computación que se asociaría más estrechamente con los microservicios. Como los contenedores individuales no tienen la sobrecarga de su propio sistema operativo, son más pequeños y ligeros que las tradicionales máquinas virtuales y pueden girar hacia arriba y hacia abajo más rápidamente, lo que los convierte en una combinación perfecta para los servicios más pequeños y ligeros que se encuentran dentro de las arquitecturas de microservicios.

Con la proliferación de servicios y contenedores, la orquestación y la gestión de grandes grupos de contenedores rápidamente se convirtieron en uno de los principales retos. Kubernetes, una plataforma de orquestación de contenedores de código abierto, se ha convertido en una de las soluciones de orquestación más populares porque realiza ese trabajo muy bien.

Gateways de API

Los microservicios a menudo se comunican a través de API, especialmente cuando se establece el estado por primera vez. Si bien es cierto que los clientes y los servicios pueden comunicarse entre sí directamente, los gateways de API suelen ser una capa intermedia útil, especialmente a medida que el número de servicios en una aplicación crece con el tiempo. Una gateway de API actúa como un proxy inverso para los clientes mediante solicitudes de direccionamiento, distribuyendo las solicitudes por varios servicios y proporcionando seguridad y autenticación adicionales.

Existen varias tecnologías que se pueden utilizar para implementar gateways de API, incluidas las plataformas de gestión de API, pero si la arquitectura de microservicios se implementa mediante contenedores y Kubernetes, el gateway se implementa normalmente mediante Ingress o, más recientemente, Istio.

Mensajería y streaming de eventos

Aunque la práctica recomendada sería el diseño de servicios sin estado, el estado existe y los servicios deben tenerlo en cuenta. Y aunque una llamada de API es a menudo una forma efectiva de establecer inicialmente el estado de un determinado servicio, no es una manera particularmente efectiva de mantenerse al día. El enfoque "¿ya llegamos?" de sondeo constante para mantener los servicios actualizados no resulta práctico.

En su lugar, es necesario acoplar las llamadas de API que establecen el estado con la mensajería o el streaming de eventos para que los servicios puedan emitir cambios en el estado y otros stakeholders puedan escuchar esos cambios y ajustarse a ellos. Es probable que este trabajo sea el más adecuado para un

message broker de uso general, pero hay casos en los que una plataforma de streaming de eventos como, por ejemplo, Apache Kafka, puede ser una buena solución. Y con la combinación de microservicios con la arquitectura impulsada por eventos, los desarrolladores pueden crear sistemas distribuidos, altamente escalables, tolerantes a errores y ampliables que pueden consumir y procesar grandes cantidades de eventos o información en tiempo real.

Sin servidor

Las arquitecturas sin servidor llevan algunos de los patrones de nube y microservicios principales a su conclusión lógica. En el caso de sin servidor, la unidad de ejecución no es solo un pequeño servicio, sino una función, que a menudo puede ser solo unas pocas líneas de código. La línea que separa una función sin servidor de un microservicio es borrosa, pero comúnmente se entiende que las funciones son incluso más pequeñas que un microservicio.

El aspecto en el cual las arquitecturas sin servidor y las plataformas de funciones como servicio (FaaS) comparten afinidad con los microservicios es que ambas están interesadas en crear unidades de implementación más pequeñas y escalar con precisión en relación a la demanda.

Servicios de microservicios y nube

Los microservicios no son necesariamente exclusivamente relevantes para la computación en la nube, pero hay algunas razones importantes por las que suelen ir juntos, razones que van más allá del hecho de que los microservicios son un estilo arquitectónico popular para nuevas aplicaciones y la nube es un popular destino de hospedaje para nuevas aplicaciones.

Entre las ventajas principales de la arquitectura de microservicios, se encuentran la utilización y los beneficios de costos asociados a la implementación y la ampliación de componentes de forma individual. Aunque estos beneficios también están presentes en cierta medida en la infraestructura local, la combinación de componentes pequeños y escalables de manera independiente con una infraestructura de pago por uso según demanda es donde se encuentran la verdadera optimización de costos.

En segundo lugar, y quizás más importante, otra ventaja principal de los microservicios es que cada componente individual puede adoptar el recurso más adecuado para su trabajo específico. La proliferación de recursos puede llevar a una grave complejidad y sobrecarga cuando la gestiona usted mismo, pero consumir el recurso de soporte como servicios en la nube puede minimizar drásticamente los retos de la gestión. Dicho de otra manera, aunque no es imposible implementar su

propia infraestructura de microservicios, no se recomienda, especialmente cuando se acaba de empezar.

Patrones comunes

Dentro de las arquitecturas de microservicios, hay muchos patrones comunes y útiles de diseño, comunicación e integración que ayudan a abordar algunos de los retos y oportunidades más comunes, incluyendo los siguientes:

Patrón back-end-for-frontend (BFF).

Este patrón inserta una capa entre la experiencia del usuario y los recursos a los que recurre la experiencia. Por ejemplo, una aplicación utilizada en un desktop tendrá diferentes tamaños de pantalla, visualización y límites de rendimiento que un dispositivo móvil. El patrón BFF permite a los desarrolladores crear y dar soporte a un tipo de programa de backend por interfaz de usuario con el uso de las mejores opciones para esa interfaz, en lugar de intentar dar soporte a un programa de backend genérico que funcione con cualquier interfaz, pero que pueda afectar negativamente al rendimiento de frontend.

Entidad y patrones agregados.

Una entidad es un objeto distinguido por su identidad. Por ejemplo, en un sitio de comercio electrónico, un objeto de producto puede distinguirse por nombre de producto, tipo y precio. Un agregado es un grupo de entidades relacionadas que deben tratarse como una unidad. Por lo tanto, para el sitio de comercio electrónico, un pedido sería un grupo (agregado) de productos (entidades) solicitados por un comprador. Estos patrones se utilizan para clasificar datos de forma significativa.

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 el término del servicio. Estos patrones proporcionan mecanismos de descubrimiento para hacer frente a esta transición. El equilibrio de carga puede utilizar patrones de descubrimiento de servicios mediante comprobaciones de estado y fallos del servicio como desencadenantes para reequilibrar el tráfico.

Patrones de microservicios del adaptador.

Piense en los patrones de adaptador como si fueran los adaptadores de enchufes que utiliza al viajar a otro país. La finalidad de los patrones de adaptador es ayudar a convertir las relaciones entre clases u objetos que de otro modo son incompatibles. Una aplicación que depende de API de terceros puede necesitar

utilizar un patrón de adaptador para garantizar que la aplicación y las API puedan comunicarse.

Patrón de aplicación Strangler.

Estos patrones ayudan a gestionar la refactorización de una aplicación monolítica en aplicaciones de microservicios. El nombre Strangler (estrangulador) se refiere a cómo una vid (microservicios) lentamente y con el tiempo supera y estrangula un árbol (una aplicación monolítica).

Antipatrones

Aunque hay muchos patrones para aplicar bien los microservicios, hay un número igualmente grande de patrones que pueden crear problemas rápidamente a cualquier equipo de desarrollo. Algunos de ellos, reformulados como "que no hacer con los microservicios", son los siguientes:

La primera regla de los microservicios es no crear microservicios.

Dicho con más precisión, no empiece con microservicios. Los microservicios son una forma de gestionar la complejidad una vez que las aplicaciones se han vuelto demasiado grandes y difíciles de actualizar y mantener. Solo cuando sienta que la dificultad y la complejidad de las aplicaciones monolíticas son relevantes será recomendable considerar cómo puede refactorizar esa aplicación en servicios más pequeños. Hasta que no sienta esa dificultad, ni siquiera tiene realmente un monolito que necesite refactorización.

No haga microservicios sin DevOps o servicios en la nube.

La creación de microservicios significa crear sistemas distribuidos, y los sistemas distribuidos son difíciles (y son especialmente difíciles si se toman decisiones que los dejan aún más complicados). Intentar realizar microservicios sin a) una implementación adecuada y una automatización de supervisión o b) servicios de nube gestionados para dar soporte a su infraestructura heterogénea ahora en expansión, es buscar muchos problemas innecesarios. Ahórrese esos problemas y dedique su tiempo a preocuparse por el estado.

No cree demasiados microservicios haciéndolos demasiado pequeños.

Si va demasiado lejos con el "micro" de los microservicios, podría encontrarse fácilmente sobrecargado y con una complejidad que supera los beneficios generales de una arquitectura de microservicios. Es mejor inclinarse hacia servicios más grandes y luego sólo dividirlos cuando comienzan a desarrollar características que los microservicios solucionan, como el hecho de que se está volviendo difícil y lento implementar cambios, un modelo de datos común se está volviendo demasiado

complejo, o que diferentes partes del servicio tienen diferentes requisitos de carga/escala.

No convierta los microservicios en SOA.

Los microservicios y la arquitectura orientada a servicios (SOA) a menudo se confunden entre sí, dado que, en su nivel más básico, ambos buscan crear componentes individuales reutilizables que pueden ser consumidos por otras aplicaciones. La diferencia entre microservicios y SOA es que los proyectos de microservicios suelen implicar refactorizar una aplicación para que sea más fácil de gestionar, mientras que SOA se ocupa de cambiar la forma en que los servicios de TI trabajan en toda la empresa. Un proyecto de microservicios que se transforma en un proyecto SOA probablemente se hundirá bajo su propio peso.

No intentes ser Netflix.

Netflix fue uno de los pioneros de la arquitectura de microservicios al crear y gestionar una aplicación que representaba un tercio de todo el tráfico de Internet, una especie de tormenta perfecta que les obligaba a crear un montón de código personalizado y servicios que son innecesarios para la aplicación media. Se recomienda empezar a un ritmo que pueda manejar, evitar la complejidad y utilizar tantas herramientas predefinidas como sea posible.

BIBLIOGRAFÍA

- Página oficial Spring(https://spring.io/)
- Página oficial Spring Boot(https://spring.io/projects/spring-boot)
- Página oficial Spring Data(https://spring.io/projects/spring-data)
- Página oficial Spring Framework(https://spring.io/projects/spring-framework)