Bootstrapping Microservices: preparando su arquitectura de microservicio

por Alexsandro Souza · 17 y 18 de enero · Zona de microservicios

Descargue Microservices for Java Developers : una introducción práctica a frameworks y contenedores. Presentado en asociación con Red Hat .

El mundo de la informática ha visto una creciente atención en la arquitectura de software de microservicios con el fin de mejorar la escalabilidad y la eficiencia del software. Microservicios ofrece muchos beneficios para las organizaciones tecnológicas. Sin embargo, también está claro que, a pesar de los beneficios de la modularización y la contenedorización, muchas organizaciones siguen teniendo problemas con los microservicios.

Una aplicación basada en microservicios comprende numerosos servicios independientes pequeños que están integrados entre sí para ofrecer las funcionalidades deseadas del software. Si bien estos pequeños servicios individuales pueden ser simples, existe una complejidad significativa que surge de su interacción en un esfuerzo por orquestar las operaciones comerciales deseadas. Mientras que en la práctica, la idea de microservicios funciona de

para minimizar la mayoría de los desafíos encontrados en esta arquitectura, hoy tenemos disponibles muchos marcos, herramientas y patrones listos para integrarse fácilmente en su proyecto de microservicio.

Ya conocemos la complejidad añadida a la arquitectura de microservicios y también sabemos que la comunidad ya ha creado muchas cosas para minimizarla. Ahora necesita saber acerca de mi proyecto de código abierto, que lo ayuda a comenzar con microservicios y abordar muchos desafíos comunes utilizando las soluciones creadas por la comunidad de código abierto.

Bootstrapping a un proyecto de microservicio

La idea de este proyecto es proporcionarle un arranque para su próxima arquitectura de microservicio utilizando Java. estamos abordando algunos de los principales desafíos que todos enfrentan al comenzar con microservicios. Este proyecto definitivamente le ayudará a comprender el mundo de los microservicios y le ahorrará mucho tiempo en configurar su arquitectura inicial de microservicio.

Básicamente, si está interesado en microservicios y estudia o quiere implementar un enfoque de microservicios en su trabajo, este proyecto es para usted.

Algunos desafíos a los que ya estamos brindando una solución en este proyecto:

- Gestión de registro
- Descubrimiento de servicio
- Tolerancia a fallos

- Configuración centralizada
- Autenticación distribuida
- Transacciones distribuidas
- Evento Distribuido
- Gestión de aplicaciones
- Gestión de la infraestructura
- Despliegue continuo

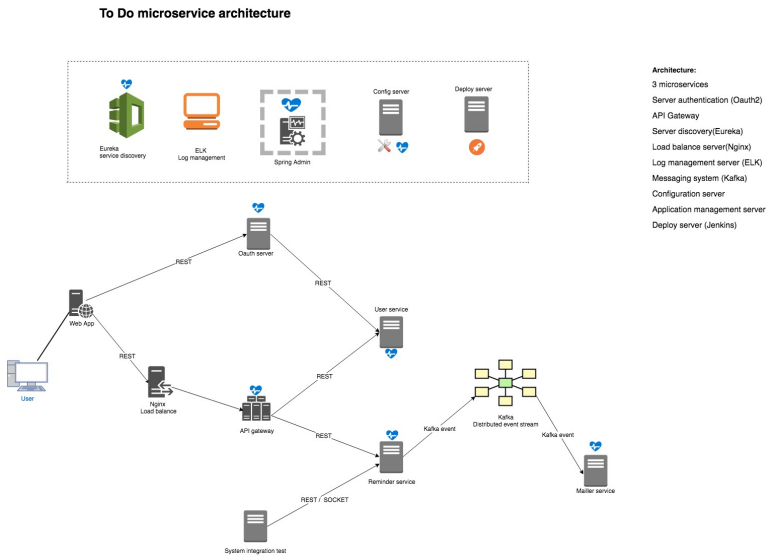
En esta publicación de blog, analizaré algunos conceptos y tecnologías que hemos incluido en nuestro proyecto de microprocesamiento bootstrap. No estoy planeando profundizar en los conceptos y herramientas, tenemos muchos comentarios sobre los que están ahí. La intención aquí es presentar un ejemplo de aplicación que contenga los patrones, las herramientas y las tecnologías utilizadas para desarrollar microservicios.

Vamos a trabajar con una aplicación pendiente, que estará compuesta por seis servicios (Recordatorio, Usuario, Aplicación de correo, Servidor OAuth, Pasarela API y Cliente de aplicaciones web) y siete herramientas (servidor de descubrimiento de servicios, servidor de equilibrio de carga, configuración servidor, administración de registros, administración de aplicaciones, intermediario de eventos y servidor de implementación continua).

Algunos patrones, herramientas y tecnologías que verá en este sistema:

Spring Boot, Spring Data, Spring Cloud Eureka, Equilibrio de carga con Ribbon, Clientes REST

monitorización de aplicaciones JMX, seguridad con Spring OAuth de seguridad, Oauth2 con JWT, programación orientada a aspectos, eventos distribuidos con Kafka, Spring Stream, proyecto Maven Multimodule, Event Sourcing, CQRS, REST, Web Sockets, implementación continua con Jenkins y todo lo desarrollado con Java 8. Este primer artículo proporcionará una visión general de todo el proyecto, y en los próximos posts, explicaré más detalladamente qué y cómo estamos usando los componentes en cada uno de ellos. microservicio.



En la imagen de arriba, puede ver cómo interactúa nuestro sistema, junto con todos los microservicios. El usuario accederá a una aplicación web escrita usando Angular 2, se conectará a un servidor de autorización OAuth, que será un punto central donde se pueden asignar usuarios y autoridades. Este servidor devolverá un token web JSON que contiene información sobre el cliente con sus autoridades y el alcance rallado. Después de que el usuario sea autenticado y con el token, la aplicación web podrá hablar con el gateway API, tomará el JWT, verificará si proviene del servidor de autorización, luego realizará llamadas a los microservicios y generará la respuesta.

El servicio de usuario está siendo utilizado por el

pendientes. El Servicio restante tiene un trabajo programado para verificar los recordatorios y notificar al usuario por correo electrónico. Los correos electrónicos son enviados por el Servicio de correo, que se desencadena por el servicio Recordatorio mediante un evento que utiliza Kafka.

La Prueba de integración del sistema es una aplicación Java responsable de alcanzar los puntos finales del servicio Recordatorio.

He grabado algunos videos que presentan el proyecto y cómo probar todo el proyecto en su computadora y desplegarlo en AWS. Compruébalo .

Conexión de microservicios

En la arquitectura de microservicios, tenemos que lidiar con muchos microservicios que se ejecutan en diferentes IP y puertos; por lo tanto, necesitamos encontrar una forma de administrar cada dirección sin una codificación rígida, y ahí es donde Netflix Eureka viene a rescatar. Es un descubrimiento del servicio del lado del cliente que permite que los servicios se encuentren y se comuniquen entre sí de forma automática. Estamos usando Spring Cloud Eureka en nuestro sistema y necesitará ver cómo funciona para comprender cómo se comunican nuestros servicios REST entre diferentes microservicios. Una vez que Eureka se preocupa por dónde se están ejecutando los servicios, podemos agregar instancias y aplicar el equilibrio de carga para distribuir el tráfico de aplicaciones entrantes entre nuestros microservicios.

En nuestro sistema, estamos utilizando Netflix Ribbon como un equilibrador de carga del lado del cliente que nos permite lograr tolerancia a fallas y aumentar la confiabilidad y disponibilidad a través de la redundancia. Estamos utilizando Netflix Foreign

estamos tratando de aislar nuestra aplicación de fallas de dependencia utilizando Netflix Hystrix Circuit Breaker, que ayuda a detener fallas en cascada y nos permite fallar rápidamente y hacer una recuperación rápida, o agregar retrocesos. Hystrix mantiene un grupo de subprocesos para cada dependencia y rechaza las solicitudes (en lugar de las solicitudes de cola) si el grupo de subprocesos se agota. Proporciona funcionalidad de interruptor automático que puede detener todas las solicitudes de una dependencia. También puede implementar lógica de respaldo cuando falla una solicitud, se rechaza o se agota el tiempo de espera.

Autenticación

La seguridad es muy importante cuando estamos desarrollando cualquier sistema y con la arquitectura de microservicios no es diferente. La pregunta "¿Cómo puedo mantener la seguridad en mis microservicios?" Aparece de inmediato, ¡y la primera respuesta es OAuth2! Y definitivamente, OAuth2 es una muy buena solución; es una tecnología de autorización muy conocida, se usa ampliamente para Google, Facebook y GitHub para sus API.

Es imposible hablar de seguridad y no mencionar Spring Security, y en nuestro proyecto, lo estamos usando junto con OAuth2.

Spring Security y OAuth2 son opciones obvias cuando hablamos de sistemas distribuidos seguros, sin embargo, estamos agregando un elemento más a nuestra preocupación de seguridad: JWT (JSON Web Token). Al usar solo OAuth, necesitaríamos tener un Servidor de Autorización OAuth para autenticar al usuario y generar el token, y también un punto final para que los servidores de Recursos pregunten si el token es válido y qué permisos otorga, requiriendo el doble de solicitudes al Authorization Server de lo que realmente necesitamos. JWT proporciona una forma

con una clave RSA privada.

Puede echar un vistazo a las implementaciones del servidor de autorización OAuth2 (OAuth-server) y del servidor de recursos (gateway API) para ver cómo se ve el código. La implementación se realizó principalmente después de esta publicación en el blog.

DESCANSO

En nuestro sistema, tenemos dos estilos de interacción: sincrónico y asincrónico. Para el estilo asíncrono, estamos utilizando eventos distribuidos con Kafka, siguiendo el modelo de publicación / suscripción, y para la sincronización, tenemos el estilo REST que admite JSON y XML.

Hay cuatro niveles de madurez de servicios RESTful, comenzando en el nivel 0, como lo describe Martin Fowler, y nuestros servicios están en el nivel 2 porque decidí no implementar los Controles de Hypermedia usando el patrón de diseño de HATEOAS, para simplificar.

Debido a que estamos utilizando Spring Cloud, tenemos algunos patrones de escalabilidad listos para usar, que se ubican en nuestras conexiones HTTP y vale la pena mencionar: Disyuntor, mamparos, equilibrio de carga, agrupamiento de conexiones, tiempos de espera y reintento.

Eventos Distribuidos

Como se mencionó anteriormente, nuestra comunicación entre el servicio Reminder y el servicio Mailer se realiza de forma asíncrona usando Kafka para distribuir nuestros eventos a través de los otros microservicios. En nuestro proyecto, estamos administrando Distributed Events a través de un flujo continuo de eventos, posibilitado por Spring Stream, donde el marco oculta las preocupaciones estándar y

procesamiento de flujo de alto rendimiento y un caso de uso impulsado por eventos.

Event Sourcing y CQRS

Las aplicaciones monolíticas suelen tener una sola base de datos relacional y podemos usar transacciones ACID. Como resultado, nuestra aplicación simplemente puede comenzar una transacción, cambiar varias filas, confirmar la transacción si todo va bien y deshacer si algo sale mal. Desafortunadamente, lidiar con el acceso a datos en la arquitectura de microservicio es mucho más complejo debido a que los datos se distribuyen en diferentes bases de datos, y la implementación de transacciones comerciales en múltiples servicios es un gran desafío.

En nuestro proyecto "Tareas pendientes", estamos utilizando eventos para hacer frente a una transacción comercial que abarca múltiples servicios, y puede ver la implementación de Event Sourcing con CQRS aplicado en el servicio Mailer. Verá cómo separar lecturas y escrituras que nos permiten escalar cada parte fácilmente. Estamos utilizando una base de datos relacional como tienda de eventos, y luego distribuimos los eventos usando Kafka. Tendremos que realizar estas dos acciones atómicas, evitar el almacenamiento del evento y no publicar en un eventual bloqueo de JVM. No estoy usando Kafka como tienda de eventos porque es más sencillo construir los agregados a partir de una base de datos relacional, y aquí estamos tratando de facilitar las cosas.

Gestión de registro

Tener estrategias bien planeadas de registro, monitoreo y análisis es la clave para este tipo de proyecto. Deben implementarse desde el comienzo del proyecto para aumentar el desarrollo y la velocidad de prueba, así como para proporcionar una

contenedor responsable de enviar los registros a un servidor de administración de registros, donde almacenará todos los registros de nuestros componentes de microservicio y proporcionará una increíble interfaz de usuario para buscar y crear informes en función de los datos enviados.

Gestión de aplicaciones

Para administrar un sistema que consta de múltiples microservicios, debe recopilar toda la información relevante en un lugar centralizado. Esto se aplica a los registros y a los detalles sobre el estado de todas las instancias de aplicaciones, que se están ejecutando ahora en nuestro clúster de microservicio.

Estamos usando Spring Boot Admin para ayudarnos a administrar nuestras aplicaciones Java. Es una solución simple creada para administrar y monitorear las aplicaciones de Spring Boot, y obtiene los datos de los puntos finales de Actuator y proporciona información sobre todas las aplicaciones registradas en un solo panel.

Gestión de Infraestructura

Los microservicios se adaptan bien a las tecnologías de contenedores y, a menudo, funcionan conjuntamente. Los contenedores suelen ser la opción preferida porque son autónomos y se aprovisionan o clonan rápidamente. En nuestro proyecto, todos los componentes se gestionan dinámicamente utilizando Docker, lo que significa que no necesita preocuparse por la configuración de su entorno local, lo único que necesita es tener instalado Docker.

Con servicios separados, tendrá que administrar la infraestructura para cada servicio. La infraestructura como código (IaC) nació como una solución a este desafío. Todo lo que necesita nuestra aplicación se describirá en un archivo (Dockerfile). Junto con el

Gestión de configuración

En una aplicación tradicional, los archivos de propiedades están vinculados a la base de código o empacados estáticamente, por lo que cualquier cambio en el archivo de propiedades significa reconstruir y volver a implementar la aplicación, lo cual es una violación del principio de microservicios. Con microservicios, creamos un servidor de configuración central donde todos los parámetros configurables de microservicios se escriben, controlan la versión y se administran en una administración de configuración centralizada. El beneficio de un servidor de configuración central es que si cambiamos una propiedad para un microservicio, puede reflejar eso sobre la marcha sin volver a implementar el microservicio.

En nuestra aplicación de arranque, ya estamos abordando la administración de configuración centralizada con Spring Config Server. Por defecto, nuestro servidor de configuración no está usando Git, estamos colocando nuestras propiedades en la estructura de archivos local, pero en entornos de producción, debe cambiar a un perfil de Git y configurar su dirección de repositorio de Git y poder cambiar sus propiedades de configuración fácilmente .

Despliegue automatizado

Cada microservicio debe tener la capacidad de implementar de forma automática y confiable cualquier versión en cualquier entorno. Mantener la implementación completamente automatizada y simple hace que sea fácil implementar cambios pequeños a menudo con confianza.

En nuestra arquitectura, tenemos un servidor de integración continua que utiliza Jenkins, que hace posible implementar todo nuestro sistema automáticamente en el sistema de contenedores de

origen.

Nuestra implementación automatizada de nuestros servicios incluye extraer el código fuente, crear el código Java, empaquetar el contenedor, crear una imagen Docker, desplegar la imagen Docker en un repositorio Docker e implementar el contenedor en el servicio contenedor AWS. Para ejecutar todos esos pasos, necesitamos tener un entorno con un montón de software y muchas herramientas instaladas. Para eso, decidimos crear un proyecto separado para mantener una imagen de Docker con todo lo que necesitamos para construir e implementar nuestra arquitectura en AWS.

Próximos pasos

Como puede ver, ya tenemos muchas cosas en este proyecto, pero todavía hay muchos desafíos que aún no se abordan aquí. Sin embargo, este es un proyecto en desarrollo y estamos planeando agregarle más cosas, así que estad atentos a nuestro repositorio de GitHub para ver más cosas divertidas.

Temas: ARQUITECTURA DE MICROSERVICIO, ARQUITECTURA DISTRIBUIDA, NUBE DE PRIMAVERA, MICROSERVICIOS

Creación de microservicios reactivos en Java: diseño de aplicaciones asíncronas y basadas en eventos
Programa de desarrollo de Red Hat