Aplicaciones Distribuídas en Internet

Arquitectura de microservicios

Autores:

- Raúl García Carrión
- Víctor Navarro Martínez-reina

Realizado el día 03 de enero de 2021

Introducción

En el presente documento, explicaremos el concepto de las arquitecturas basadas en microservicios, qué características tienen, las ventajas e inconvenientes que presentan y algún ejemplo práctico que explicaremos como lo hemos hecho y lo adjuntaremos junto con este documento.

La historia de los microservicios se puede rastrear hasta Mayo del 2011 en Venecia, donde un grupo de arquitectos de software se pusieron de acuerdo en utilizar la palabra para caracterizar los sistemas en cuyo desarrollo estaban involucrados.

Con la influencia de algunos de los pioneros en el campo (Adrian Cockcroft, Netflix,…) los microservicios han ido ganando presencia y, hoy en día, se aceptan en la industria como un estilo arquitectónico bien establecido.

Muchas organizaciones están migrando sus sistemas a esta arquitectura, descomponiendo sus monolitos en unidades más pequeñas y distribuidas.

La idea principal es la de procesos pequeños y autónomos que trabajan en colaboración para cumplir cierta lógica de negocio. Pero llegar a este concepto es una tarea difícil. Si no se tiene un conjunto de principios coherente que guíe, el camino hacia esta arquitectura puede ser muy largo. En un sistema distribuido hay más decisiones que tomar que en un sistema monolítico. ¿Un único lenguaje para todo? ¿Qué almacén de datos? ¿Cuál es el protocolo de comunicaciones que debo utilizar? ¿Qué servidores?

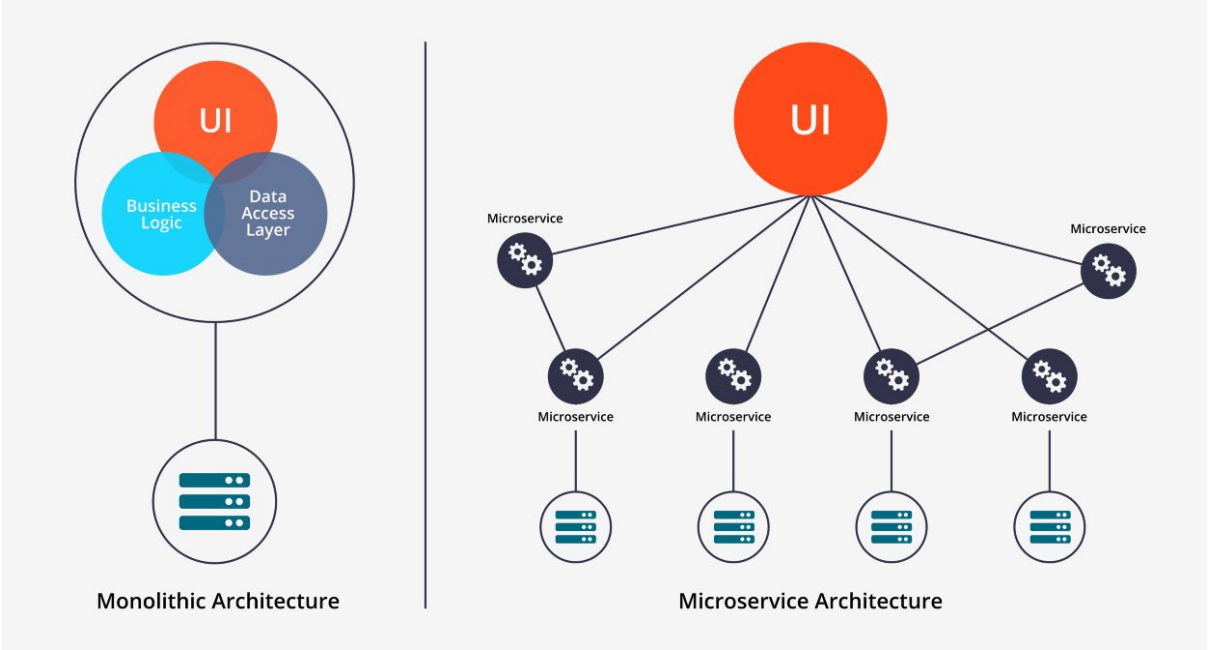
Todas estas preguntas y más vamos a intentar contestarlas a continuación.

Arquitecturas basadas en microservicios.

En un principio, el diseño de software se realizaba desde un enfoque monolítico en el que la aplicación contiene todas las funcionalidades, se aloja en un mismo servidor, y por lo tanto, existe un gran acoplamiento entre las diferentes partes del software. Esto puede funcionar bien para aplicaciones que no evolucionan en el tiempo y que estén bien definidas, pero actualmente la mayor parte de los sistemas software está en continuo cambio y el enfoque monolítico hace que las aplicaciones sean poco escalables. Una de las soluciones a este importante problema es la arquitectura basada en microservicios.

La arquitectura basada en microservicios consiste en dividir una aplicación en diferentes componentes de manera que puedan tratarse de manera independiente sin afectar a los demás. Esto nos da la posibilidad de alojar cada componente en un servidor diferente y realizar actualizaciones, nuevas implementaciones y escalar en función de la demanda. También, si es necesario, cada microservicio gestiona su propia base de datos. La comunicación con estos microservicios se suele hacer mediante APIs ligeras bien definidas.

A continuación, vemos una imagen que simplifica la comparación entre el enfoque monolítico y la arquitectura basada en microservicios:



Las arquitecturas basadas en microservicios están teniendo un gran impacto gracias a tres importantes avances en las últimas dos décadas:

- Las conexiones a internet.
- Los componentes hardware que utilizan los servidores para gestionar las peticiones y almacenar información (velocidad de procesamiento y almacenamiento).
- Los sistemas software, tanto de desarrollo (frameworks) como de producción (virtualización y contenedores).

Características de los microservicios

Las arquitecturas basadas en microservicios son muy variadas, aunque la mayor parte de ellas tienen unas características comunes bien diferenciadas. Son las siguientes:

1. **Componentes como servicios**: La manera más común de hacer una aplicación gestionada por microservicios es implementando botones que internamente gestionan las peticiones a los servicios en lugar de hacer llamadas a bibliotecas en memoria. La comunicación con los microservicios se hace mediante los protocolos de servicios web o RPC (llamadas a procedimientos remotos).

2. **Organización sobre las funcionalidades del negocio:** Cada servicio debe encargarse de una función específica del sistema al que pertenece y limitar su responsabilidad. El servicio se encargará de implementar desde la interfaz de usuario, la capa de negocio, la persistencia de los datos y sus relaciones con otros servicios. También, puede ser que algunos servicios no implementen alguna de las capas nombradas.

3. **Productos de proyectos.** Normalmente, con esta arquitectura, cada equipo se encarga de un servicio en todo su ciclo de vida (diseño, construccion, produccion y mantenimiento). Esto hace que se vea el software como una relación continua en donde los cambios que se aplican lo realizan las personas especializadas en ello.

4. **Extremos inteligentes.** Cada microservicio recibe una petición, aplica su lógica y produce una respuesta. Esta sucesión de pasos se realiza usando protocolos simples como HTTP con REST.

5. **Tecnologías que se adaptan mejor a cada funcionalidad.** Ya que cada servicio es independiente unos de otros, se pueden utilizar tecnologías que mejor se adapten a las necesidades, como por ejemplo lenguajes de

programación, bases de datos, frameworks, etc. Esto es posible siempre que el tipo de petición y la respuesta se adecuen a los demás servicios.

6. **Gestión descentralizada de datos.** Es posible que cada servicio administre su propia base de datos e incluso utilizar diferentes tecnologías. Existen ciertos problemas en la consistencia de datos ya que las transacciones distribuidas son muy costosas de implementar. Es por eso que se permite la coordinación no transaccional entre servicios, mientras que el coste de solucionar errores sea pequeño.

7. **Diseño tolerante a fallos.** Todo servicio tiene la posibilidad de fallar por algún motivo, por lo que el cliente tiene que estar preparado ante estas situaciones y evitar un fallo en cascado que imposibilite el uso del sistema al completo. La forma más común para esto es aplicar los siguientes patrones, entre otros:
   - Tiempos de espera máximos.
   - Disyuntores. Es una especie de interruptor que cuando un servicio falla se puede apagar para así no acceder a él, y cuando vuelve a estar operativo se activa de nuevo.

8. **Automatización de la infraestructura.** Los microservicios son desarrollados por equipos que usan una metodología de entrega continua e integración continua. Y para conseguir esto es necesario tres factores:
   - Herramienta de automatización de la construcción del proyecto que pase pruebas automáticas.
   - Control de versiones y configuraciones para tener identificados los posibles fallos a que versiones o configuraciones pertenecen.
   - Arquitectura adecuada para realizar cambios sin afectar a los demás componentes del sistema (la arquitectura basada en microservicios ya aporta esta característica).

9. **Diseño evolutivo.** Es necesario que cada servicio debe poder ser reemplazado o actualizado independientemente.También se tiene que evitar que la evolución del sistema afecte a los usuarios.

Ciclo de vida del desarrollo en microservicios

Para poder tener éxito y beneficios potenciales y esperados del propio desarrollo de aplicaciones con microservicios, tendremos que saber los **Factores Críticos de Éxito**.

Lo primero que debe asegurar cualquier organización es la correcta "**Definición e implantación de una estrategia de desarrollo en microservicios**, que asegure la viabilidad del cambio y la obtención de los beneficios según el Ciclo de Vida previsto", es decir una estrategia que nos asegure que al hacer el cambio de una aplicación monolítica a una arquitectura de microservicios, nos aporte beneficios claros.

Una estrategia suele ser la visión de una situación futura con unos beneficios asociados, que nos transporta a un estado "ideal" en el que nos vemos trabajando con soltura y en un ambiente de inmensa alegría compartida bajo el nuevo modelo de desarrollo en el que todo es sencillo, funciona a la primera y los equipos están eufóricos y en armonía.

Una buena estrategia para este éxito es la que está basada en el **Ciclo de Vida del desarrollo en microservicios**. Ahora mostramos un esquema:

En general, el Ciclo de Vida se divide en tres Fases: Lanzamiento, Expansión y Estabilización. Cada una de estas fases necesita de cierto tiempo y otros recursos.

La Fase de Expansión es la que mayor beneficio nos da en el tiempo, por lo que cuanto antes lleguemos a ella antes obtendremos el valor del desarrollo en microservicios. Sin embargo, si la Fase de Lanzamiento no se realiza correctamente en la Fase de Expansión puede que nos de valor, pero a un coste insostenible. Adicionalmente, si la Fase de Expansión no se realiza correctamente, no podremos llegar a la “eficiencia prometida”, la intuiremos, pero no la obtendremos.

Lo más recomendable es definir la estrategia a la vez que vamos definiendo la implantación, lo que supone hacernos las preguntas adecuadas y aportar respuestas viables a los ámbitos funcional, de arquitectura, de metodología y de equipo. De forma práctica, este ejercicio no es complicado y se debe realizar independientemente del tamaño de los equipos, de los volúmenes de desarrollo o del alcance funcional que se quiere tener con el desarrollo en microservicios. Además, nos permitirá estimar la inversión necesaria (no solo la económica) y el retorno esperado. Es importante seguir esta metodología antes de empezar con las iniciativas de desarrollo en microservicios.

Ventajas e inconvenientes

Al igual que cualquier otra arquitectura de desarrollo software, los microservicios tienen sus ventajas y desventajas ya que nada es blanco o negro, todo suele ser gris, simplemente hay que saber en qué ámbitos es mejor aplicar ciertas arquitecturas u otras. A continuación vamos a pasar a hacer un listado con los distintos aspectos de los microservicios.

Ventajas

Los microservicios potencian a los equipos y las rutinas. También puede desarrollar múltiples microservicios de forma paralela. Gracias a ello, más desarrolladores pueden trabajar en la misma aplicación al mismo tiempo y reducir el tiempo de desarrollo. Vamos a pasar con las distintas ventajas que podemos encontrar:

1. **Comercialización más rápida.** Las aplicaciones desarrolladas con la arquitectura de microservicios permiten una implementación y actualización más rápida por la reducción de ciclos de desarrollo.

2. **Escalabilidad.** Conforme los servicios de nuestra aplicación van creciendo en demanda se puede escalar fácilmente e implementar en distintos servidores o infraestructuras, tantas como sean necesarios.

3. **Capacidad de recuperación.** Si hemos sido capaces de desarrollar correctamente los servicios de nuestra aplicación, un fallo en una de estas partes sólo afectará a aquella que haya dado el fallo sin que el resto de la aplicación lo note.

4. **Facilidad de implementación.** Las aplicaciones basadas en microservicios son más modulares y más pequeñas, por tanto a la hora de su implementación se requiere más coordinación pero a cambio de múltiples ventajas como las que estamos viendo.

5. **Accesibilidad.** En este tipo de arquitecturas las aplicaciones grandes se “trocean” en pequeñas aplicaciones más fáciles de entender, desarrollar y actualizar, por ello los ciclos de desarrollo son más rápidos.

6. **Aplicaciones más abiertas.** Cada programador y desarrollador va a tener la posibilidad de elegir el lenguaje o tecnología que mejor se adapte a las funcionalidades que quiera implementar en el servicio.

Inconvenientes

El mayor de los inconvenientes de este tipo de arquitectura es la migración que existe cuando pasamos de una arquitectura diferente a los microservicios, ya que, a parte de cambiar las aplicaciones también se cambiará la forma de trabajar de las personas. Vamos a enumerar algunas categorías que suponen un cambio y un añadido en la dificultad de esta arquitectura:

1. **Diseño.** Se debe invertir más tiempo en identificar las dependencias entre los distintos servicios de la arquitectura. Además es probable que aparezca la necesidad de otros servicios por estas dependencias. Se deben considerar los efectos de los microservicios en sus datos.

2. **Pruebas.** Las pruebas de integración, así como las pruebas finales, pueden tomarse más complejas e importantes que nunca ya que si falla una parte de la arquitectura, puede desencadenarse un error muy grave en la aplicación, en función del diseño que hayamos hecho.

3. **Implementación.** Sobre todo durante la configuración inicial de la aplicación vamos a encontrarnos dificultades a la hora de implementarla. Para simplificar, lo primero y más importante es que se debe invertir mucho tiempo en la automatización ya que la complejidad de los microservicios resulta agobiante para la implementación humana.

4. **Registro.** Con estos sistemas distribuidos, vamos a necesitar registros centralizados para integrar todos los elementos, este tipo de registros suelen dar más problemas.

5. **Depuración.** La depuración remota no es opción y no funcionará en decenas ni cientos de servicios. Desgraciadamente, no hay una única respuesta sobre cómo realizar la depuración en este momento.

Herramienta de desarrollo de arquitectura de microservicios: JHipster

JHipster es una plataforma de desarrollo para generar, desarrollar e implementar rápidamente aplicaciones web y arquitecturas de microservicios modernas.

Esta herramienta utiliza varios tipos de tecnologías tanto en la parte del cliente como en la parte del servidor:

Tecnología en el lado del cliente

- Angular, React o Vue
- Diseño web responsive con Twitter Bootstrap
- HTML5 Boilerplate
- Compatible con buscadores modernos (Chrome, Firefox, Microsoft Edge…)
- Soporte SASS opcional para diseño con CSS
- Soporte de WebSocket opcional con Spring Websocket

Tecnología en el lado del servidor

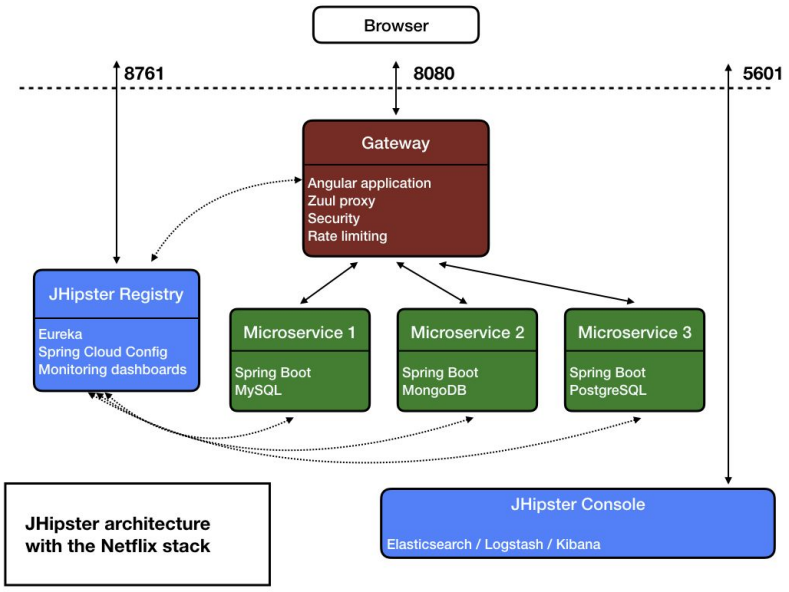
- Spring Boot para la configuración de la aplicación
- Configuración de Maven o Gradle para construir, testear y ejecutar la aplicación
- Spring Security
- Spring MVC REST + Jackson
- Spring Data JPA + Bean Validation
- Actualización de la base de datos con Liquibase
- Soporte con Kafka si se quiere usar un sistema de mensajes Publicación-Suscripción

Ahora vamos a ver de forma breve de qué forma se configura y qué tecnologías se usan para la arquitectura de microservicios.

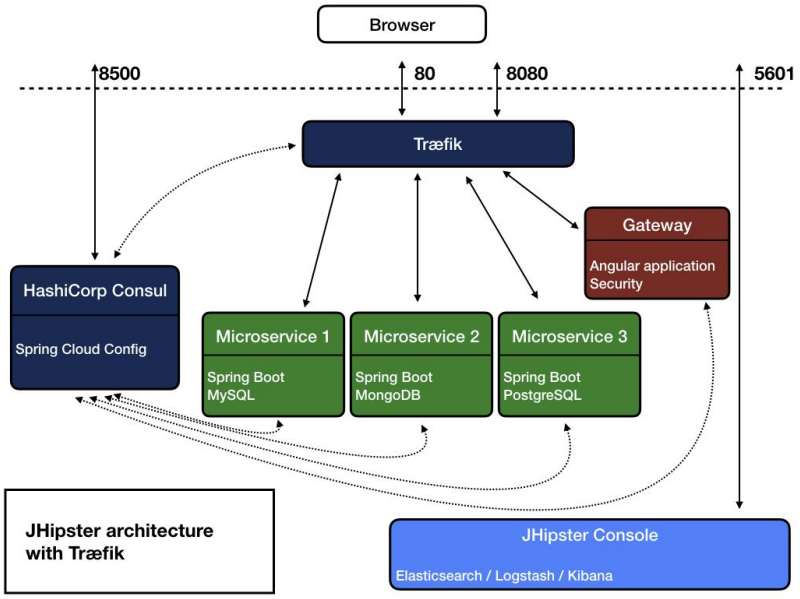
Microservicios

La arquitectura de microservicios de JHipster funciona de la siguiente forma:

- Un Gateway es una aplicación generada por JHipster que maneja el tráfico web y devuelve una aplicación de Angular o React. Más específicamente nos proporciona enrutamiento HTTP, balanceador de carga, calidad de servicio, seguridad y documentación API para todos los microservicios.

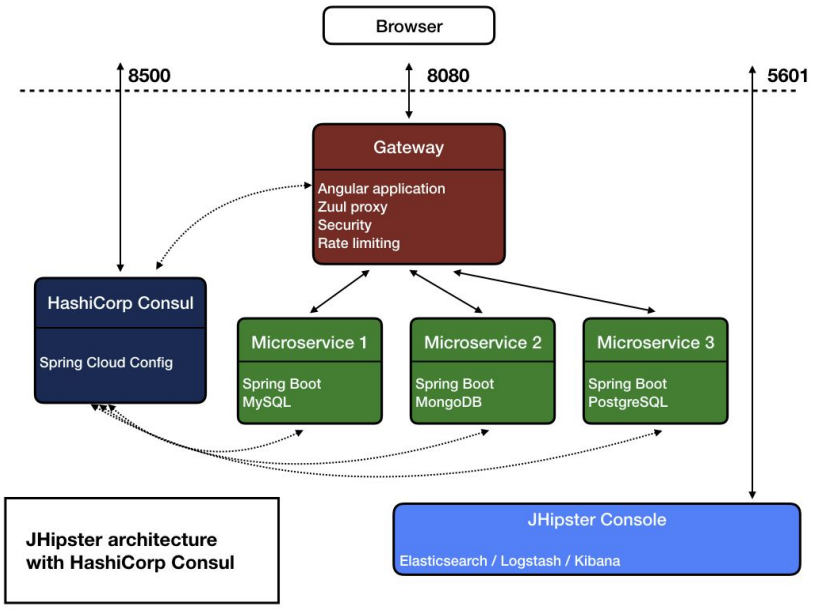


- **Traefik**. Es un moderno inverso proxy HTTP y balanceador de carga que puede trabajar con un Gateway y está hecho para desplegar los microservicios con facilidad. Uno de los beneficios de Traefik es que puede trabajar con muchos servicios diferentes.



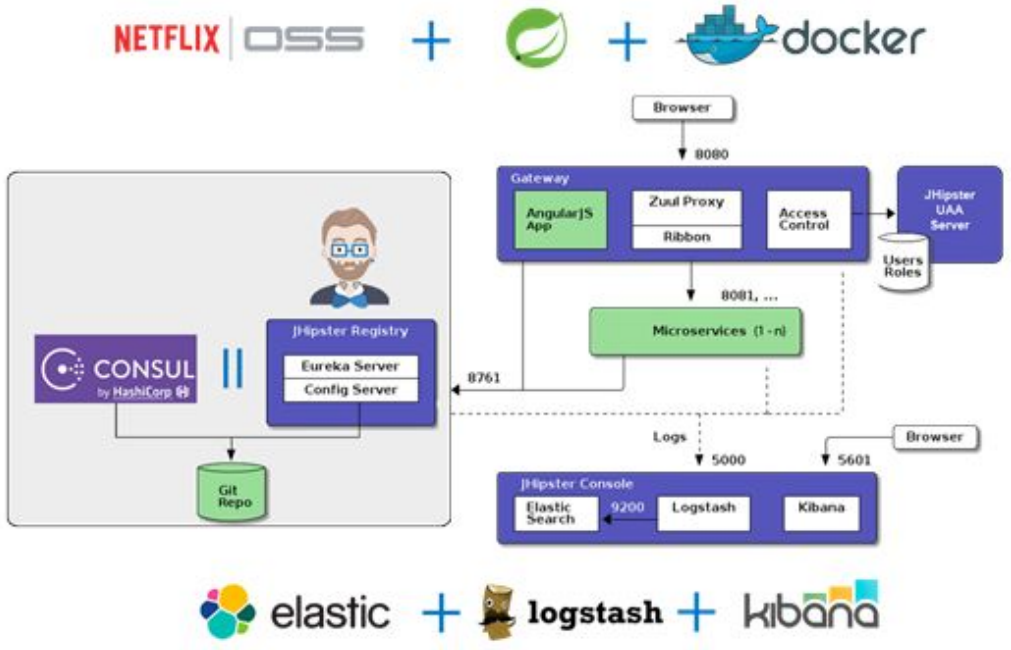
- **JHipster Registry**. Es una aplicación en tiempo de ejecución en la que todas las aplicaciones se registran y obtienen su configuración. Tiene 3 propósitos: Como un servidor Eureka que sirve como servidor de descubrimiento para aplicaciones, como un servidor Spring Cloud Config que proporciona configuración en tiempo de ejecución a las aplicaciones y además como un servidor de administración con un cuadro de mandos donde monitorizar y administrar las aplicaciones.

- **Consul**. Como una alternativa a JHipster Registry podemos elegir Consul, una solución de administración y centro de datos de Hashicorp.



- **Microservicios**. Son aplicaciones generadas por JHipster que manejan peticiones de tipo REST. Muchas instancias de estos pueden ser lanzados en paralelo para manejar grandes cargas. Los microservicios no tienen front-end (ya que son generador en el Gateway) y que suelen trabajar con JHipster Registry para ser configurados y manejados.

Diagrama de despliegue final



Bibliografía

https://comunytek.com/introduccion-a-los-microservicios/
https://aws.amazon.com/es/microservices/
https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4
https://es.wikipedia.org/wiki/Arquitectura_de_microservicios
https://www.redhat.com/es/topics/microservices/what-are-microservices
https://www.hiberus.com/crecemos-contigo/de-una-arquitectura-tradicional-a-microservicios/
https://www.jhipster.tech/