uc3m | Universidad Carlos III de Madrid

Grado Ingeniería de Sistemas Audiovisuales

2018-2019

Trabajo Fin de Grado

“Diseño e implementación de un microservicio con Spring”

Jesús Rienda Iáñez

Tutor/es

Carmen Pelaez Moreno

Leganés, 2019

RESUMEN

Palabras clave:

DEDICATORIA

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Planteamiento del problema

Contamos con datos reales sobre listas de reproducción de Spotify en formato json los cuales contienen información de cada lista y las canciones que contiene. Por otra parte tenemos datos de usuarios con las canciones que escuchan.

Necesitamos almacenar estos datos en una base de datos para posteriormente consultarla, actualizarla o crear nuevos registros. Podríamos actuar directamente sobre la base de datos con un cliente pero no seria admisible para un usuario final. Por tanto tendremos que crear un programa el cual haga las consultas a la bbdd y devuelva los datos en un formato optimo para mostrarlos por pantalla. Para ello Por lo que necesitamos crear un programa que al invocar nos devuelva los datos almacenados con un tratamiento especifico y un formato definido. Necesitamos que sea sencillo y simple para el cliente que va a consumir dicho servicio. Por ejemplo uno de los tratamiento necesarios sería filtrar las listas en función de las canciones que escuche cada usuario, si ha escuchado mas de 3 canciones de una, deberíamos devolverla.

Este programa tendrá que tener una alta disponibilidad y escalarse cuando sea necesario para siempre tener unos tiempos de respuesta bajos. Ya que existe una gran cantidad de personas que van a consumir el programa al mismo tiempo en determinadas franjas horarias.

1.2. Solución propuesta

Para la solución de nuestro problema vamos a necesitar productos de distintos proveedores, ya que cada pequeña parte del programa la gestiona un software diferente.

Uno de los puntos mas importantes a decidir seria que base de datos usar, en este caso usaremos CosmosDB. Como pieza de software que se comunique con la bbdd, trate lso datos y los devuelva crearemos un servicio web mediante la arquitectura de microservicios desarrollado en java con el framework Spring Boot. Como protocolo para la transferencia de datos usaremos REST apoyándonos en sus verbos GET, POST, PUT y DELETE.

1.3. Justificación de la solución

La arquitectura de microservicios pretende dividir una aplicación compleja en pequeños servicios que solo realicen una función especifica y se comuniquen entre ellos para formar la aplicación final.

Cada microservico es totalmente independiente de desarrollar frente al conjunto, lo cual nos viene genial ya que nuestra idea es realizar una pequeña aplicación que aceda a una base de datos y en un futuro ampliar a varias aplicaciones o incluso un frontal. Los microservicios nos permiten desarrollar cada una en un lenguaje de programación diferente.

Al no disponer de una maquina donde desplegar la aplicación, es ideal que los microservicios lleven un servidor de despliegue(tomcat) embebido y así desplegar en la nube dentro de un contenedor de aplicaciones(Dockers).

Una vez desplegado en la nube decidimos que la comunicación sería mediante REST ya que es un protocolo simple y muy eficaz para realizar las distintas operaciones(verbos) en base de datos: añadir, recuperar, actualizar y eliminar, esto en REST seria GET, POST, PUT y DELETE.

En cuanto a base de datos hemos elegido PostgreSQL ya que es open source y totalmente compatible con muchos lenguajes de programación, no solo con Java que es el caso de nuestra aplicación, sino que si en un futuro queremos creamos otro microservicio con python sería posible reutilizar la base de datos. Ademas también es capaz de responder a gran cantidad peticiones en un mismo instante y no bloquearse, esto es totalmente esencial ya que necesitamos alta disponibilidad.

1.4. Estado del arte

En la década de los 60 surgió lo que a día de hoy conocemos como arquitectura de software, esta fue tomando cada vez mas interés hasta que en la década de 1980 se integro totalmente el diseño en el desarrollo de software.

El Instituto de Ingeniera de Software la define como: "La arquitectura de software es una representación del sistema que ayuda a comprender cómo se comportará un programa.

La arquitectura del software sirve como modelo tanto para el sistema como para el proyecto que lo desarrolla. La arquitectura es la principal portadora de cualidades del sistema, como el rendimiento, la modificabilidad y la seguridad, ninguna de las cuales se puede lograr sin una visión arquitectónica unificadora. La arquitectura es un artefacto para el análisis temprano para asegurar que un enfoque de diseño proporcionará un sistema aceptable. Al construir una arquitectura efectiva, puede identificar los riesgos de diseño y mitigarlos al inicio del proceso de desarrollo."[1].

El sistema se divide en elementos de software también llamados módulos, con propiedades y relaciones existentes entre ellos. Las propiedades de estos elementos pueden ser de dos tipos, internas y externas.

Las **propiedades internas** son aquellas que definen el módulo, es decir, el lenguaje en el que está desarrollado y todos los detalles de la implementación de este como pueden ser:

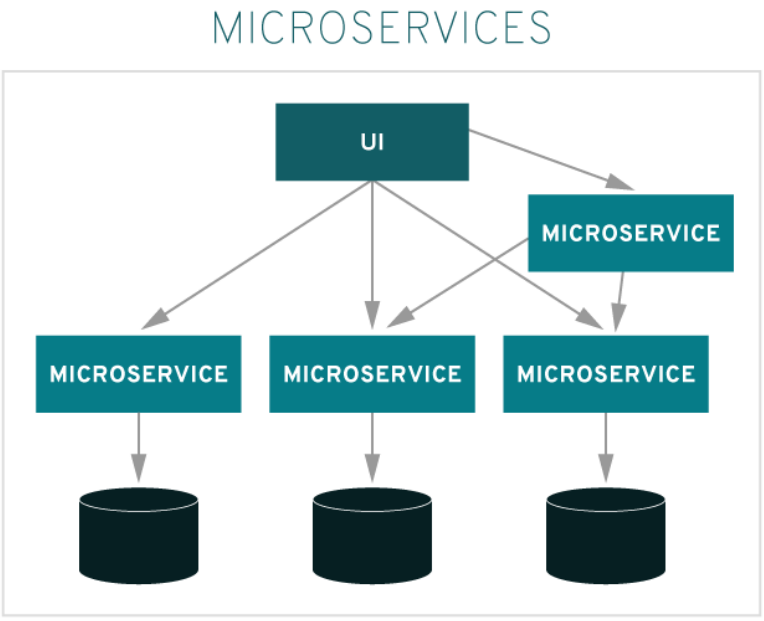

Fig. 1.1

- Entidades dinámicas en tiempo de ejecución como objetos e hilos.

- Entidades lógicas en tiempo de desarrollo como clases y módulos.

- Entidades físicas como nodos o carpetas.

Las **propiedades externas** son los contratos que existen entre módulos y que permiten a otros módulos establecer dependencias/conexiones entre ellos. Es de vital importancia que las interfaces que definen los contratos estén bien definidas para la perfecta integración de los elementos.

Es necesario dividir los requerimientos en módulos con los que va a contar el sistema, sus propiedades y como se relacionan entre si. Por ejemplo en la figura 1.1 podemos ver una estructura de un sistema basado en microservicios donde aparecen bases de datos que serian un modulo donde habría que definir sus propiedades, tablas, colecciones de datos, etc; cada microservicio es un modulo del sistema donde habría que definir un api en el que indicar la entrada y salida del modulo; por ultimo tendríamos el modulo de presentación UI(Interfaz de usuario).

En el momento de diseño de una arquitectura existen dos formas de actuar, diseñar una arquitectura nueva o buscar soluciones a nuestro problema. Si elegimos la segunda opción existen patrones de diseño, son una forma reutilizable para resolver los problemas comunes. Estos provienen de la arquitectura civil concretamente del libro .A pattern language"[2], en él se nombran los patrones para que con hacer referencia a su nombre cualquier arquitecto sea capaz de entender que solución se ha usado. Sin embargo en la arquitectura de software no aparecieron hasta 1994 en un libro que ha día de hoy sigue siendo referencia[3].

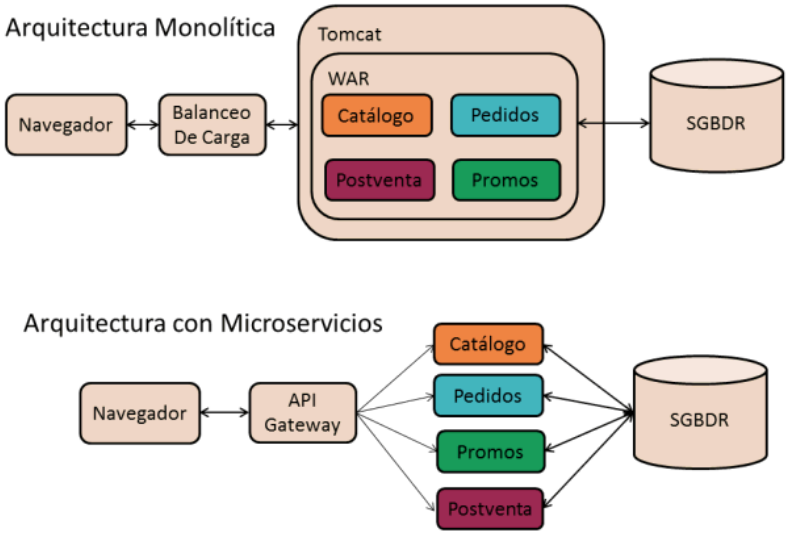

Fig. 1.2. Arquitectura Monolítica frente Microservicios

Los patrones en el libro [3] se dividen en función del problema a resolver:

- Patrones creacionales

- Patrones estructurales

- Patrones de comportamiento

Primero surgió la arquitectura orientada objetos mas adelante la orienta a componentes. Pero no fue hasta 1996 cuando se desarrollo por primera vez SOA, arquitectura orientada a servicios. En ella se desarrollaban todos los servicios que tu necesitas conjuntamente y se empaquetan en un war el cual se despliega en un servidor de aplicaciones(tomcat) dentro de una maquina, esto lo podemos ver en la figura 1.2.

Todos los servicios tenían que estar desarrollados con el mismo lenguaje y no podías asignar mas recursos a uno de ellos sino que se lo asignabas a todo el conjunto, escalando el war en varias maquinas o replicas en la misma. Para ello necesitabas un balanceador de carga antes que determine maquina va a atender tu petición.

Todo esto antes era mas que suficiente para las empresas, pero a día de hoy cuando una aplicación monolítica(SOA) crece mucho es difícil mantener y es complicado añadir nuevas funcionalidades, ya que cada linea modificada implica re-desplegar toda la aplicación, lo que en una empresa grande puede llevar bastante tiempo, ya que en los despliegues normalmente están involucrados varios departamentos de la empresa como seguridad, operaciones, arquitectura y desarrollo, que impide al equipo seguir desarrollando. También es complicado encontrar el origen de algún error en el código.

La necesidad de resolver todos estos problemas desencadeno en la arquitectura de microservicios. La primera vez que se menciono la palabra "microservicios"fue en 2011 en una conferencia sobre computación en la nube donde el Dr. Peter Rogers[4] se refirió a ello para describir la arquitectura que estaban usando grandes empresas como Netflix,

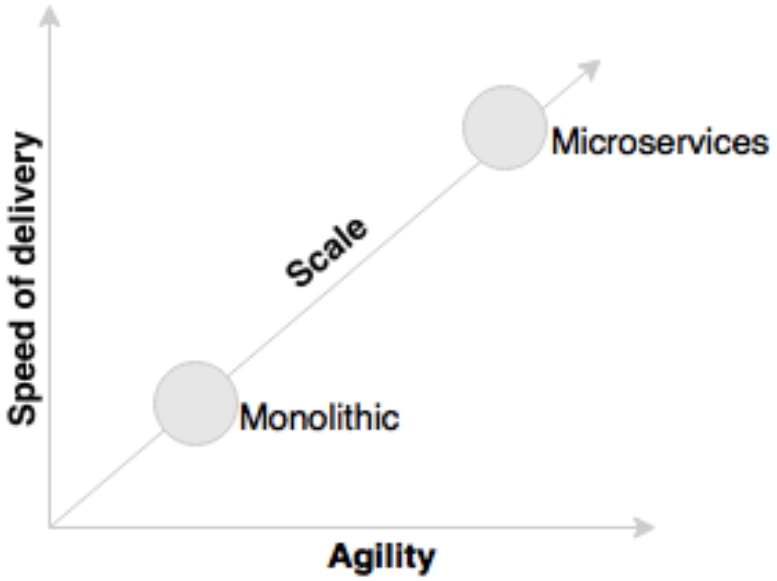

Fig. 1.3. Eficiencia microservicios frente SOA

Facebook, Amazon o PayPal.

Los microservicios gestionan la complejidad granulando funcionalmente en un conjunto de servicios pequeños e independientes. Con esto se consigue que el equipo de desarrollo sea capaz de desarrollar varias funcionalidades a la vez sin tocar código de otra funcionalidad y desplegar cada modulo por separado tal y como se ve en la figura 1.2 donde cada microservicio esta separado del resto, y puede o no tener una base de datos común.

El cambio mas notable respeto a SOA es que los equipos de desarrollo tienen una mayor responsabilidad, lo que se traduce en una gran facilidad, ya que ellos manejan todo el proceso de desarrollo, desligues en distintos entornos, gestión de contenedores como Kubernetes, etc. Todo esto antes tenían que realizarlo otros departamentos de la empresa con el aumento de tiempos que suponía.

Aunque la arquitectura de microservicios resuelve todos los problemas que presenta SOA y cada vez es mas popular, aun esta en su base de inicio como se menciona en el articulo[5] y aun le queda mucho por mejorar y evolucionar.

Rajest RV en su libro "Spring Microservices"[6] nos muestra un gráfico1.3 en el que se ve como es mucho mas rápido y ágil el desarrollo de aplicaciones con microservicios frente a tradicionales. "Los microservicios prometen más agilidad, velocidad de entrega y escala. En comparación con las aplicaciones monolíticas tradicionales."

En un futuro se deberían solucionar problemas debidos a estar poco restringidos, por ejemplo si cada microservicio lo desarrollas con un lenguaje diferente no esta del todo claro que los protocolos que uses en cada uno de comunicación sean totalmente compatibles.

En otro tema que tienen que mejorar es en la seguridad, ya que cuando tu descompones una aplicación en cientos de microservicios creas dificultad en la depuración, monitoreo, auditoría y análisis forense de toda la aplicación. Los atacantes podrían aprovechar esta complejidad para atacar.

Lo que si es seguro es que han surgido para quedarse y que cada vez mas gente se esta pasando a ellos.

BIBLIOGRAFÍA

[1] S. E. Institute, "Software Architecture", 2017. [En línea]. Disponible en: https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328.

[2] C. Alexander, S. Ishikawa y M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[3] E. Gamma, *Patrones de diseño: elementos de software orientado a objetos reutilizable*, ép. Addison-Wesley professional computing series. Pearson Educación, 2002. [En línea]. Disponible en: https://books.google.es/books?id=gap_AAAACAAJ.

[4] L. Mauersberger, "A brief history of microservices", *blog.leanix.net*, 2017. [En línea]. Disponible en: https://blog.leanix.net/en/a-brief-history-of-microservices.

[5] N. Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow", en *Present and Ulterior Software Engineering*, M. Mazzara y B. Meyer, eds. Cham: Springer International Publishing, 2017, pp. 195-216. DOI: 10.1007/978-3-319-67425-4_12. [En línea]. Disponible en: https://doi.org/10.1007/978-3-319-67425-4_12.

[6] R. RV, *Spring Microservices*. Packt Publishing, 2016. [En línea]. Disponible en: https://books.google.es/books?id=pwNwDQAAQBAJ.