Un acercamiento a los microservicios

Karen Tatiana Gómez Suárez, Raquel Anaya y Andrés Felipe Cano - Corporación Universitaria Adventista

Resumen

El desarrollo basado en microservicios es una tendencia emergente que surge de las necesidades de la industria de software, para mejorar la escalabilidad y flexibilidad de las aplicaciones web y que hoy en día se reconoce como un nuevo modelo de arquitectura conocido como microservicios. Generalmente, las aplicaciones web tradicionales siguen una arquitectura por capas, en las que se hace separación lógica de la solución en tres capas: la interfaz de usuario, la lógica de la aplicación y el sistema de gestión de datos; sin embargo, el despliegue de la solución se realiza como una unidad monolítica que se ejecuta en un solo espacio de direcciones, generando problemas ante la demanda de aplicaciones con servicios especializados que requieren manejo de grandes volúmenes de datos integrados con enfoques IoT y con requerimientos de procesamiento distribuido. Las arquitecturas basadas en microservicios proponen una arquitectura en la que cada funcionalidad de negocio se descompone en servicios web altamente cohesivos que pueden ser desplegados, evolucionados y escalados de manera independiente. Este enfoque trae beneficios como la flexibilidad, escalabilidad y productividad del equipo de trabajo, pero también conlleva nuevos retos que están siendo enfrentados, como la seguridad, el desempeño y la mantenibilidad de la solución. Uno de los aspectos más importantes para tener en cuenta en el desarrollo de este tipo de aplicaciones es el alineamiento organizacional a través de un enfoque de desarrollo orientado al dominio (DDD).

Palabras clave: microservicios, monolítica, arquitectura.

Keywords: microservices, monolithic, architecture.

1. Introducción

Una de las tendencias recientes en la práctica de construir aplicaciones web distribuidas es el enfoque de microservicios, el cual emerge como

propuesta de arquitectura en las necesidades reales de la industria de software, para atender las demandas de mantenimiento y escalabilidad de las aplicaciones modernas, arquitectura que se hizo popular cuando las prácticas de integración continua y DevOps cobraron impulso en la industria del software [1][2]. Con este enfoque se busca que un equipo se encargue de su mantenimiento y evolución, por lo cual subdividir un gran sistema en microservicios, facilita el mantenimiento y la evaluación del software, manejando cada microservicio como un sistema independiente. El enfoque de microservicios es un estilo de arquitectura definido como un conjunto de servicios pequeños y autónomos que trabajan juntos; el tamaño de cada servicio. Esto significa que cada microservicio tiene un conjunto reducido de líneas de código, para facilitar los cambios, utilizando el principio de servicio de responsabilidad simple, permitiendo que un equipo se encargue de su mantenimiento y evolución [3].

Hay tres razones por las cuales los microservicios están ganando popularidad: En primer lugar, se ven como una alternativa para evolucionar sistemas monolíticos, dado que dichos sistemas son cada vez más complejos, costosos para evolucionar, no cumplen las exigencias de escalabilidad y disponibilidad que exige el negocio, y por ello se opta por estrategias de modernización gradual de dichos sistemas utilizando microservicios [4][5].

En segundo lugar, el enfoque de microservicios resulta adecuado para el desarrollo de nuevas soluciones de software que buscan hacer uso intensivo de los recursos de red y comunicaciones disponibles; algunas de las experiencias publicadas en la literatura son: el desarrollo de sistemas que se despliegan en la nube y pueden tomar ventaja la elasticidad y aprovisionamiento de funcionalidad bajo demanda [6], diseño de plataformas IoT en soluciones orientadas a ciudades inteligentes [2] o para apoyar la optimización de los recursos naturales, como la energía a través de soluciones que permiten la interoperabilidad del hardware [7].

En tercer lugar, la adopción de microservicios ha sido motivada por el éxito de grandes compañías de Internet, como Amazon y Netflix, y se convierte en una gran motivación para que otras empresas consideren hacer el cambio [8].

Si bien esta nueva tendencia de desarrollo de software ofrece soluciones prometedoras, también trae consigo nuevos retos que deben ser enfrentados y considerados por la industria de software.

El propósito de este artículo es analizar desde una perspectiva práctica las características de este nuevo patrón de arquitectura y los beneficios y retos que conlleva su adopción.

El artículo está estructurado de la siguiente manera: en la sección 2 se describe la diferencia entre el enfoque

de SOA y el enfoque de microservicios, y entre estos y las aplicaciones web tradicionales. En la sección 3 se presenta una guía de implementación en Java. En la sección 4 se presentan las consideraciones de base para adoptar el enfoque de microservicios y, finalmente, en la sección 5 se presentan las conclusiones.

Motivación personal

Como primera autora de este artículo, he decidido realizar este trabajo porque me resulta un tema muy interesante y novedoso, además de tener una gran proyección en un futuro muy próximo.

Fue en mis prácticas profesionales donde tuve un primer acercamiento con este tema, y el actual artículo me permite consolidar los conocimientos que he adquirido día a día, además de permitirme apreciar los múltiples detalles que se deben tener en cuenta para que todo funcione de la manera deseada en una aplicación de este tipo.

1. Conceptos generales

2.1 Micro servicios vs SOA

El enfoque de construir soluciones a partir de servicios no es una idea nueva; una de las propuestas arquitectónicas ampliamente utilizadas y estudiadas es la Arquitectura Orientado a Servicios (SOA). La motivación de descomposición de servicios es implementar soluciones software a partir de elementos débilmente acoplados, que evolucionan y se desplieguen de manera independiente.

En una solución tipo SOA, la funcionalidad aplicativa se brinda a través de componentes denominados servicios, que presentan interfaces estándar bien definidas, que representan funciones de negocio; los servicios se pueden combinar a través de una coreografía de servicios, permitiendo implementar de ese modo procesos de negocio ágiles y flexibles [9].

La principal diferencia entre un enfoque SOA y los microservicios es el nivel de granularidad del cual se está hablando: mientras que en un enfoque SOA se tienen servicios de granularidad gruesa centrados en funcionalidad de negocio y generalmente con protocolos de comunicación rigurosos, un enfoque de microservicios se orienta en la estructura interna de una aplicación, con servicios de granularidad fina, cada uno de ellos especializado en una funcionalidad específica y haciendo uso de protocolos livianos de comunicación como HTTP mediante APIs RESTFull [1].

Según lo explicado por Martin Fowler, mientras que SOA es un enfoque de arquitectura orientado a la integración de aplicaciones, la arquitectura de microservicios describe un modo particular de construir aplicaciones con servicios muy pequeños, cada uno de los cuales se enfoca en hacer solo una cosa bien (https://martinfowler.com/microservices/).

2.2 Microservicios vs aplicaciones web clásicas

La mejor manera de explicar los microservicios es compararlos con la arquitectura de las aplicaciones web clásicas:

En el diseño tradicional de tres capas, la interfaz de usuario, de lado del cliente, denominada la capa de presentación, la lógica de negocio y la capa de datos que ofrece los servicios de persistencia. Si bien este tipo de soluciones presenta una descomposición lógica de la solución, su despliegue se realiza como una unidad monolítica que se ejecuta en un solo espacio de direcciones y por lo tanto, su escalamiento se realiza replicando la misma aplicación en múltiples servidores o instancias de ejecución (en la nube), lo que acarrea alto consumo de recursos.

En la arquitectura de microservicios, la aplicación se construye a partir de múltiples servicios pequeños, de granularidad fina, aislados, independientes y distribuibles (ver Figura 1).

El hecho de que estos servicios se trabajen de manera desacoplada es trascendental, pues permite priorizar los recursos escasos a los microservicios más relevantes, sobre los demás. Estos servicios pueden ser desarrollados en paralelo, por diversos equipos, usando las tecnologías más adecuadas para cumplir sus propósitos. El desacoplamiento de los microservicios permite que estos puedan ser escalados de manera independiente. Por ejemplo, un servicio que consume mucha CPU, pero no necesita tanta memoria puede ser instalado en servidores equipados con un procesador potente pero con menor memoria.

Surge entonces el interrogante: ¿Cómo compartir código entre servicios que son independientes y aislados? La reutilización de funcionalidades existentes es un aspecto que sigue siendo relevante en este enfoque, que sigue el principio DRY (no escribir código duplicado), lo negativo es que aumenta el acoplamiento entre los servicios. Una solución es compartir solamente las librerías de utilidades técnicas y las de funcionalidades comunes, estas se pueden configurar como servicios independientes a los cuales otros servicios pueden llamar.

Otro aspecto importante es la comunicación entre estos microservicios, la cual puede implementarse de dos maneras: mediante peticiones HTTP o a través de cola de mensajes (Azure Service Bus, RabbitMQ, Kafka Apache, etc.).

HTTP es comunicación directa y debe usarse cuando se desea una respuesta inmediata del otro servicio, mientras que el mecanismo publicación/suscripción de la cola de mensaje, es una comunicación asíncrona, que hace la solicitud del servicio y continúa la operación cuando recibe la respuesta de la contraparte.

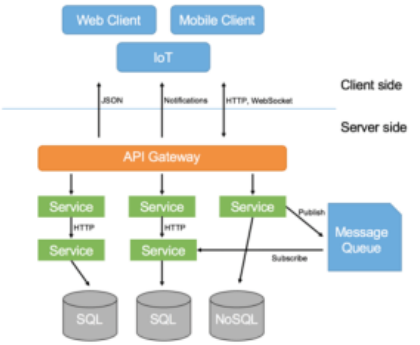

Figura 1. Vista General de una Arquitectura de Microservicios. Fuente: https://co.pinterest.com/khoadinh290/software-architecture/

Para permitir cambios en los servicios sin afectar a los clientes, se utiliza una API Gateway. La API Gateway es una capa abstracta que oculta a todos los microservicios, dejando un único endpoint para que los clientes se comuniquen. Las solicitudes que lleguen al Gateway son enrutadas hacia los servicios específicos; el Gateway también permite monitorear fácilmente el tráfico y uso de los servicios.

3. Guía de implementación – Java

Consecuente con el surgimiento del enfoque de microservicios desde una perspectiva práctica, en esta sección se introduce el modelo de referencia presentado por Daniel Sánchez en su sitio web, en donde se explican los componentes principales de esta arquitectura:

Servidor de configuración central.

Este componente se encarga de centralizar y proveer remotamente la configuración a cada microservicio. La configuración de cada microservicio es mantenida en un repositorio Git, lo que permite una gestión automática de su propio ciclo de vida y versionado.

Servicio de registro/descubrimiento.

Es un servicio centralizado, encargado de proveer los endpoints de los servicios para su consumo. Todo microservicio se registra automáticamente en él en tiempo de bootstrap.

Balanceo de carga (Load balancer).

Este patrón de implementación permite el balanceo entre distintas instancias de forma transparente a la hora de consumir un servicio.

Tolerancia a fallos (Circuit breaker).

Mediante este patrón se conseguirá que cuando se produzca un fallo, este no se propague en cascada, y se pueda gestionar el error de forma controlada a nivel local del servicio donde se produjo.

Servidor perimetral.

Es un Gateway en el que se expondrán los servicios por consumir.

Centralización de Logs.

Es un mecanismo centralizado para llevar la gestión de logs de todos los microservicios.

Servidor de Autorización.

Utilizado para implementar la capa de seguridad; es recomendable que se haga en la capa de servicios API.

Monitorización.

Es conveniente disponer de mecanismos y algún dashboard para monitorizar aspectos de los nodos como salud y carga de trabajo.

Para aquel lector que desee entender al detalle la manera como se implementa un microservicio, se recomienda revisar el ejemplo presentado por Daniel Sánchez en este mismo sitio web, el cual muestra paso a paso la implementación de un microservicio utilizando las siguientes tecnologías:

- Spring Boot: como framework de aplicación.
- Spring MVC: como framework web.
- Spring Cloud: para integrar el servicio con otros componentes.
- Swagger: para documentar y definir el API.

Existen otros sitios como el presentado por Chris Richardson, los cuales permiten acceder a ejemplos que ilustran una solución de microservicios [8]. En este sitio se recomienda también el uso de algunos patrones útiles para implementar algunas de las características de este tipo de solución.

4. Consideraciones para la implementación de microservicios

El buen uso de este nuevo estilo de arquitectura trae beneficios evidentes con respecto al desarrollo web tradicional, pero también trae consigo nuevos desafíos. Se trata entonces de buscar un equilibrio entre las múltiples propiedades que acarrea este enfoque, algunas de las cuales entran en conflicto [5].

Existen estudios sistemáticos que recopilan los trabajos recientes en el tema de microservicios [1][10], los cuales buscan identificar el estado actual de la práctica y los desafíos de investigación con respecto a este tema . El estudio realizado por la Universidad de Brighton, UK [1], plantea dos interrogantes importantes, que se describen en las siguientes secciones. ¿Cuáles son los atributos de calidad relacionados con los microservicios? ¿Cuáles son los retos arquitecturales que enfrenta un sistema de microservicios?

4.1 Beneficios de un enfoque de microservicios

Los siguientes son los principales beneficios que este estilo de arquitectura provee [11]:

Capacidad de recuperación ante fallas (resilience), y la facilidad de despliegue se logra por la descomposición a través de servicios que proporciona componentes con límites claros, lo que permite aislar fallas y degradar gradualmente la funcionalidad del sistema, así como actualizar e implementar servicios individuales de forma independiente.

Escalabilidad: La escalabilidad en microservicios se concibe desde tres dimensiones: por duplicación horizontal de recursos de máquinas (Xaxis), por particionamiento de datos (eje Z), y por descomposición funcional (eje Y).

Alineamiento organizacional: Es habilitado por la organización de los microservicios alrededor de las capacidades del negocio, y motiva a

los equipos más pequeños a componentes como unidades pequeñas, lo cual facilita la mantenibilidad.

Incremento en la agilidad: El hecho de tener servicios escalables, adaptables, modulares y rápidamente accesibles hace que los microservicios favorezcan la evolución del software y así dar una mejor respuesta a las demandas de los clientes en entornos cambiantes. El esfuerzo del desarrollo es realizado alrededor de múltiples equipos y cada equipo es responsable por uno o más servicios simples y puede desarrollar, desplegar y escalar sus servicios independientemente de todos los otros equipos [8].

Heterogenidad de tecnología: Este enfoque favorece la programación y persistencia políglota, lo cual permite la coexistencia de diferentes tecnologías utilizadas por diversos componentes en el sistema.

4.2 Retos de un enfoque de microservicios

Comunicación e integración: Definir una estrategia de comunicación adecuada que involucre aspectos como el protocolo correcto, las expectativas del tiempo de respuesta, los tiempos de espera (timeout) y el diseño de API. Optimizarlas haciendo uso de servicios de caché altamente disponibles en donde se pueda, es una estrategia ventajosa también en cuanto al rendimiento y las comunicaciones.

Descubrimiento de servicios: Es conveniente tener un estándar y un proceso consistente por el cual los servicios puedan registrarse y anunciarse. También involucra decidir la estrategia del consumidor correcta, y especificar la manera como el API Gateway es configurado para reportar disponibilidad e identificación de un servicio.

Desempeño: Este es uno de los atributos que más se ven penalizados en una solución de microservicios, con respecto a una solución monolítica en la que los llamados están en un mismo espacio de direcciones. Atender una funcionalidad de negocio particular disparada por la capa de interface, requiere un proceso de orquestación de múltiples servicios de granularidad fina, lo cual introduce tiempos de espera adicionales para la experiencia de usuario. Se requiere, por lo tanto, definir mecanismos de datos compartidos y primitivas de sincronización para evitar la sobrecarga de comunicación causada por la copia de datos que ocurre durante las invocaciones del servicio.

El balanceo de cargas es otro aspecto directamente relacionado con el desempeño. Existen dos formas de hacerlo: una es es usar un balanceador, que es un ente que se encarga de usar una estrategia (ya sea round robin o sticky session) para direccionar las peticiones hacia alguna instancia del microservicio que se encargue de procesarla. La segunda estrategia, que desde el punto de vista de los autores puede ser más conveniente, es usar comunicaciones asíncronas a través de colas de mensajes, esto facilita que las instancias del microservicio se comporten como consumidores de mensajes y autorregulen su carga, pues cada vez que tienen capacidad para procesar un mensaje irán por el

siguiente, de esta manera la carga se distribuye más eficientemente, haciendo un mejor uso de los recursos de infraestructura. Esta última estrategia favorece el desacople entre emisor y consumidor del mensaje y la escalabilidad de los consumidores de una manera más fácil.

Tolerancia a fallos: Los desarrolladores de los microservicios deben considerar mecanismos de recuperación o finalización del servicio ante fallas en el sistema.

Seguridad: La seguridad es otro de los retos que deben ser cuidadosamente abordados en una solución de microservicios. Es vital que exista un usuario identificado a lo largo de toda la cadena de comunicaciones que sucede entre los microservicios.

Rastreo y logueo: Es vital en este tipo de soluciones contar con servicios de rastreo y logueo que les permita a los desarrolladores entender el comportamiento del sistema como un todo.

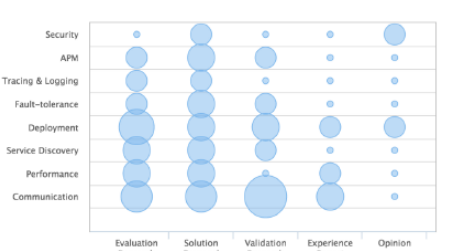

Figura 2. Distribución de trabajo según retos identificados: Tomado de [1].

La Figura 2 da una visión general de la manera como los esfuerzos de investigación están realizándose alrededor de los diferentes retos. Se puede observar, por ejemplo, que el tema de despliegue es el más trabajado en la investigación y que el mayor número de trabajos se enfoca en proponer soluciones que enfrenta buena parte de los retos mencionados.

4.3 Consideraciones metodológicas

De acuerdo con Fowler, una característica importante de los microservicios es que estos deben estar organizados alrededor de capacidades del negocio, y pueden verse como una estrategia para descomposición de funcionalidad en un contexto de diseño orientado al dominio.

Domain.Driven Design (DDD) es un conjunto de estrategias de modelado estratégico y táctico necesario para diseñar software que se articula con los objetivos de negocio [12]. Esta aproximación provee mecanismos para representar el mundo real en la arquitectura, de tal manera que los elementos del espacio del diseño y la implementación, preserven la semántica del espacio del problema. Según Newman [3], el uso de DDD es un factor clave para construir aplicaciones basadas en microservicios, tal como lo ilustra el trabajo realizado por Steinegger et al. [13].

5. Conclusiones

En resumen, la arquitectura de microservicios es un enfoque de desarrollo de una aplicación como un conjunto de servicios de granularidad

fina; cada uno se ejecuta en su propio proceso y se exponen, normalmente, a través de protocolos HTTP mediante APIs RESTFull. Estos servicios están construidos alrededor de capacidades o funcionalidades de negocio y con independencia de despliegue a través de un sistema de automatización. Además, pueden estar escritos en diferentes lenguajes de programación y utilizar distintas tecnologías de almacenamiento de base de datos, facilitando el mantenimiento y evolución en el tiempo de grandes sistemas en el tiempo, haciéndolos también más escalables y disponibles.

La velocidad es lo que predomina en el mercado, y la idea de esta nueva arquitectura es ganar en velocidad, también. La Arquitectura de Microservicios es un estilo muy potente e interesante, que debe utilizarse para aplicaciones empresariales, o proyectos que se sabe seguro que van a escalar.

La preparación del equipo es de vital importancia si se van a usar microservicios, puede que sea buena idea integrar primero DevOps y las herramientas de DevOps en la organización.

Este estilo de arquitectura surge de la necesidad práctica de la industria de software de contar con pequeños equipos de trabajo altamente productivos, que generalmente siguen prácticas ágiles y que usan entornos de trabajo con capacidades para integración y despliegue continuo, a la vez que se enfocan en la solución de problemas donde el desempeño y escalabilidad desempeñan un papel importante.

Subdividir un gran sistema en microservicios, facilita el mantenimiento y la evolución del software, puesto que cada microservicio se maneja como un sistema independiente. Por otra parte, esto requiere un alto nivel de interacción entre los equipos de trabajo, de manera que todos los servicios respeten los contratos establecidos en el momento de su diseño, de cara a la funcionalidad de negocio.

Bibliografía

1. Alshuqayran, N., Ali, N. & Evans, R. (2016, November). A Systematic Mapping Study in Microservice Architecture. In Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on (pp. 44-51). IEEE.

2. Krylovskiy, A., Jahn, M. & Patti, E. (2015, August). Designing a smart city internet of things platform with microservice architecture. In Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on (pp. 25-30). IEEE.

3. S. Newman, Building Microservices, 1st ed. O'Reilly Media, Inc., 2015.

4. Knoche, H. (2016, March). Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (pp. 121-124). ACM.

5. Levcovitz, Alessandra, Ricardo Terra, and Marco Tulio Valente. "Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems." arXiv preprint arXiv:1605.03175 (2016).

6. Parejo, J. A., Ramírez, A., Romero, J. R., Segura, S. & Ruiz-Cortés, A. Configuración guiada por búsqueda de aplicaciones basadas en microservicios en la nube.

7. Rodríguez Molina, Jesús. Distribution of microservices for hardware interoperability in the Smart Grid. Diss. ETSIS_Telecomunicacion, 2015.

8. Chris Richardson. Pattern: Microservices architecture. http://microservices.io/patterns/microservices.html, 2014. Ajustar espacios

9. Bolo M. Arquitectura de integración orientada a servicios. Interfases. 2006 Feb 25(001):19-46.

10. Claus Pahl and Pooyan Jamshidi. Microservices: A systematic mapping study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science, pages 137–146, 2016.

11. Krylovskiy, A., Jahn M. y Patti E. Designing a smart city internet of things platform with microservice architecture. In Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on 2015 Aug 24 (pp. 25-30). IEEE.

12. Vernon, Vaugh. Implementing domain-driven design. Addison-Wesley, 2013.

13. Steinegger, Roland H. et al. “Overview of a Domain-Driven Design Approach to Build Microservice-Based Applications.” SOFTENG 2017: The Third International Conference on Advances and Trends in Software Engineering.

Fecha de recepción: 24 de julio de 2018.
Fecha de aprobación: 25 de julio de 2018.

Karen Tatiana Gómez Suárez - Corporación Universitaria Adventista, Medellín, Colombia
Correo electrónico: ktgomez@unac.edu.co
Raquel Anaya - Corporación Universitaria Adventista, Medellín, Colombia
Correo electrónico: raquel.anaya.hdez@gmail.com
Andrés Felipe Cano - Corporación Universitaria Adventista, Medellín, Colombia
Correo electrónico: andres.cano@ceiba.com.co