Escuela Politécnica Nacional
Facultad de Ingeniería de Sistemas
Carrera de Ingeniería de Software

Materia de Aplicaciones Web Avanzadas

Nombre: Luis Alejandro Llanganate Valencia

Ensayo Microservicios vs. SOA - Caso GitHub

Actualmente, es frecuente que las empresas de desarrollo de software adopten aplicaciones ágiles y microservicios, sin intención de extinguir a la Arquitectura Orientada a Servicios conocida por sus siglas como SOA. A primera vista, los dos enfoques suenan muy similares ya que involucran entornos de nube o nube híbrida para el desarrollo y la implementación de aplicaciones ágiles, y ambos pueden escalar para satisfacer las demandas operativas y de velocidad de datos [1]. Ambas arquitecturas dividen aplicaciones grandes y complejas en componentes pequeños y flexibles con los que es más fácil trabajar. No obstante, estas se diferencian de una arquitectura monolítica tradicional en que cada servicio tiene su propia responsabilidad [2].

GitHub fue fundado en 2008 acompañado fuertemente por desarrolladores de Ruby on Rails, partiendo por una arquitectura monolítica que durante años permitió módulos que manejaban múltiples implementaciones por día con una interfaz de usuario bastante eficiente. En su entorno monolítico, desde los inicios fue más fácil comenzar a funcionar más rápido, sin tener que preocuparse por dependencias complejas. En aquel tiempo alguien podía utilizar GitHub en su máquina local en muy poco tiempo [3]. Considerando que tanto las arquitecturas monolíticas como las de microservicios tienen sus ventajas, como por ejemplo que en las arquitecturas monolíticas hay cierta simplicidad a nivel de código; es decir, no se requiere agregar lógica adicional para lidiar con los tiempos de espera o preocuparse por fallar correctamente debido a la latencia de la red y las interrupciones [4].

Además, debido a que en la empresa la mayoría de equipos trabajan en una misma pila de tecnología compartida y están familiarizados con la misma base de código, es más fácil mover personas y equipos para trabajar en diferentes funciones dentro del monolito e impulsar una priorización más global de las funciones. Debido a la forma en que GitHub creció en los últimos 18 meses del 2021, algunas de las ventajas de un entorno de microservicios fue realmente considerados

como una opción. Por ejemplo, configurar equipos de características con propiedad a nivel de sistema y tener límites funcionales a través de contratos de API claramente definidos. Los equipos tienen mucha libertad para elegir la pila de tecnología que tenga más sentido para ellos, siempre que se sigan los contratos de API [5].

Los servicios más pequeños en GitHub significaron tener un código más fácil de leer, un tiempo de aceleración más rápido y una resolución de problemas más fácil dentro de esa base de código. Permitió que sus desarrolladores ya no tengan que comprender todo el funcionamiento interno de una gran base de código monolítico para ser productivo [4]. Permitiéndoles que los servicios puedan escalar por separado en función de sus necesidades individuales. La buena arquitectura comienza con la modularidad. El primer paso para romper un monolito es pensar en la separación del código y los datos en función de las funcionalidades de las funciones. Esto se puede hacer dentro del monolito antes de separarlos físicamente en un entorno de microservicios.

Por lo general, es una buena práctica arquitectónica hacer que el código base sea más manejable, comenzando con los datos y prestando atención a cómo acceder a ellos. Fue fundamental asegurarse que cada servicio posea y controle el acceso a sus propios datos, y que el acceso a los datos solo se produzca a través de contratos de API claramente definidos. Por otro lado, el monitoreo, CI/CD y la contenedorización no son conceptos nuevos, pero realizar los cambios operativos necesarios para respaldar la transformación de monolito a microservicios puede generar ahorros de tiempo significativos y ayudar a acelerar la transición hacia los microservicios [3].

Para este proceso de adaptación se tuvo en cuenta las características principales de los microservicios cuando realice estos cambios en el flujo de trabajo. El soporte operativo de numerosos servicios pequeños y que se ejecutan de forma independiente con diversas pilas de tecnología fue muy diferente de ejecutar una canalización única y altamente personalizada para un gran monolito. Fue necesario actualizar el monitoreo de métricas de llamadas funcionales a métricas de red e interfaces de contrato, y dar una salida hacia una canalización de CI/CD más automatizada y confiable que se pueda compartir entre servicios. Para esto fue necesario la creación de contenedores para admitir una variedad de lenguajes y pilas tecnológicas; creando plantillas de flujo de trabajo para habilitar la reutilización [5].

Por ejemplo, en GitHub, se creó una plataforma de tiempo de ejecución de autoservicio para entregar microservicios en una caja. El objetivo fue reducir drásticamente los gastos generales operativos de cada equipo para crear microservicios [6]. Se disponían de plantillas preparadas para Kubernetes, configuración gratuita de Ingress para equilibrar la carga, canalización automática de registros a Splunk e integración en nuestro proceso de implementación interno. Por lo tanto, era más fácil para cualquier equipo que quiera experimentar o configurar un nuevo microservicio para comenzar .

En conclusión, la transición de monolito a microservicios se basó inicialmente en la importancia de mantener una modularidad y la separación de datos, que permita enfrentar dependencias complejas y permitir que el sistema sea escalable. GitHub comenzó con aplicar cambios operativos en torno a las comunicaciones servicio a servicio y la creación de sistemas resistentes; relacionando así los microservicios con el producto y el valor comercial.

Referencias

[1] "SOA vs. Microservices: What's the difference?", Ibm.com. [En línea]. Disponible en: https://www.ibm.com/cloud/blog/soa-vs-microservices. [Consultado: 11-mar-2022].

[2] B. Linders, "The journey from monolith to microservices at GitHub: QCon plus Q&A", InfoQ, 07-ene-2021. [En línea]. Disponible en: https://www.infoq.com/news/2021/01/monolith-microservices-github/. [Consultado: 11-mar-2022].

[3] T. Anderson, "GitHub's journey towards microservices and more: 'We actually have our own version of Ruby that we maintain'", The Register, 01-dic-2020. [En línea]. Disponible en: https://www.theregister.com/2020/12/01/githubs_journey_towards_microservices/. [Consultado: 11-mar-2022].

[4] O. Bonilla Bitkeeper, "The advantages and disadvantages of monolithic, multiple, and hybrid repositories", Bitkeeper.org. [En línea]. Disponible en: https://www.bitkeeper.org/BK_Nested_White_Paper.pdf. [Consultado: 11-mar-2022].

[5] "GitHub history and deleted files", Newrelic.com. [En línea]. Disponible en: https://docs.newrelic.com/docs/style-guide/writing-docs/writer-workflow/github-history/. [Consultado: 11-mar-2022].

[6] "Why does Google use a monolithic Perforce-style code repository for the entire company when the larger programming community has mostly m", Quora. [En línea]. Disponible en: https://www.quora.com/Why-does-Google-use-a-monolithic-Perforce-style-code-repository-for-the-entire-company-when-the-larger-programming-community-has-mostly-moved-to-dispersed-Git-repositories. [Consultado: 11-mar-2022].