Joe Martinez Seguir
Soy un ingeniero de software con experiencia empresarial. Me apasiona el desarrollo de la nube de back-end, especialmente API y microservicios.
25 de dic. · 6 minutos de lectura

¿Qué se considera un microservicio único?

Mi nombre es Joe Martinez. Soy un ingeniero de software con experiencia empresarial y recientemente he estado explorando la arquitectura de microservicio para soluciones basadas en la nube. Este artículo es el primero de varios en los que compartiré con ustedes algunas de mis ideas de mi aprendizaje de viaje y el dominio de microservicios desde una perspectiva amplia y profunda.

Mientras estaba aprendiendo acerca de la arquitectura de microservicios, comencé buscando algunas definiciones básicas en línea. Pude encontrar muchas definiciones de "microservicios", pero el término se utilizó como sinónimo de "arquitectura de microservicio". El artículo de Wikipedia tiene una definición básica bastante buena que comienza con:

Microservicios *es una variante del estilo arquitectónico de arquitectura orientada a servicios (SOA) que estructura una aplicación como una colección de servicios débilmente acoplados . En una arquitectura de microservicios, los servicios deben ser de grano fino y los protocolos deben ser livianos.*

Esto, junto con otros artículos web, publicaciones de blogs y videos que encontré, hicieron un buen trabajo al explicar qué son los "microservicios" como estilo arquitectónico. Aprendí los principios más comúnmente aceptados, los beneficios, los desafíos y las trampas. Sin embargo, ninguno de ellos parecía responder una pregunta básica: **¿qué constituye realmente un microservicio único?**

Es un microservicio:

A) ¿Una única función / método / extremo HTTP?

B) ¿Una colección de operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una sola entidad?

C) ¿Un conjunto reutilizable de pequeñas funciones que diferentes partes de una organización pueden usar?

D) ¿Un modelo para todo un departamento dentro de una organización empresarial?

E) ¿Un subconjunto de funciones relacionadas que dicho departamento puede realizar?

Puedes argumentar que solo estamos discutiendo la semántica. Si bien esto puede ser cierto, la forma en que pensamos acerca de un término como "microservicio" en forma singular puede terminar afectando la forma en que diseñamos nuestra solución completa. Tuve varios comienzos en falso mientras comencé a diseñar algunos proyectos simples de microservicios porque no tenía una buena idea de lo que era un microservicio individual en realidad.

Exploremos las definiciones anteriores (AE) en el contexto de algunos principios comúnmente aceptados de microservicios y arquitectura de software en general:

1) **Los microservicios múltiples no deben compartir una única base de datos.** Este es un principio muy importante sobre el que la mayoría de las autoridades están de acuerdo. Cada microservicio debe poseer sus propios datos. Las bases de datos compartidas introducen un acoplamiento estricto entre los microservicios, que está en conflicto directo con la autonomía, sin la cual no se están haciendo microservicios. Ahora, consideremos una tabla de clientes en una base de datos. Probablemente va a tener una función (un punto final HTTP / REST si está expuesto de esa manera) para crear un nuevo cliente, otra para agregar detalles de perfil, otra para cambiar su dirección de correo electrónico, etc. Lo más probable es que todas estas operaciones necesiten toque la tabla / base de datos del Cliente. Si tuviéramos que considerar cada una de estas funciones como microservicios individuales, estarían rompiendo la regla contra el uso compartido de la base de datos. Entonces, parece**La definición A no es correcta** . Supongo que técnicamente podría solucionar esto diseñando un "microservicio" con un único punto extremo uber que haga varias cosas

diferentes basadas en diferentes cargas útiles JSON / XML. Pero esto sería feo y no muy amigable para el consumidor.

2) Los **microservicios deben ser altamente cohesivos.** Volviendo al ejemplo de nuestro cliente, podemos razonar que nuestra tabla de clientes es nuestra entidad, y debemos crear un microservicio cuyo propósito es realizar las operaciones CRUD en esta tabla (**definición B**) En su libro "Building Microservices" (O'Reilly), Sam Newman advierte contra esto, refiriéndose a ellos como "servicios Anemic CRUD-based". Además de sus operaciones CRUD básicas, es probable que haya una lógica especializada y procesos adicionales que se deben producir al realizar estas operaciones. También es probable que haya varias otras operaciones que deberá realizar en una entidad del Cliente además del CRUD básico. Para lograr un alto nivel de cohesión, todos estos deben ser parte del mismo microservicio. Entonces, si se encuentra construyendo un microservicio que no es más que un wrapper CRUD, probablemente debería ser parte de un microservicio más funcional.

3) **No repita usted mismo (principio DRY) .**Este principio no es específico de microservicios, sino un principio comúnmente citado de desarrollo de software, formulado en el libro The Pragmatic Programmer por Andy Hunt y Dave Thomas. DRY busca eliminar la duplicación de la funcionalidad al irrumpir o extraer funciones o métodos que se usarán en varios lugares de tu código. Esto se puede aplicar a microservicios, pero debes tener cuidado. Tradicionalmente, esto se haría a través de bibliotecas compartidas, y generalmente se desaconseja compartir bibliotecas entre microservicios, ya que hacen que sea demasiado fácil introducir de forma accidental un estrecho acoplamiento entre servicios. Por otro lado, existirán preocupaciones transversales genuinas en su aplicación, y ¿realmente desea crear funcionalidades de correo electrónico, por ejemplo, en cada microservicio individual? Una forma de resolver este conflicto es utilizar bibliotecas estables de terceros para estas cuestiones transversales, pero puede haber una necesidad genuina de funciones de utilidad que sean específicas de su dominio o que no estén disponibles como bibliotecas de terceros. En este caso, puede crear estos como sus propios microservicios que pueden ser consumidos por sus otros microservicios, siempre que tenga cuidado de no introducir acoplamientos mediante la exposición de datos internos, por ejemplo. Asi que, por ejemplo. Asi que, por ejemplo. Asi que,**La definición C se** vuelve razonable, pero es solo una parte menor de la respuesta.

4) **D Diseño Omain-Driven (DDD)** is an important concept in software engineering, formulated in the book Domain Driven Design by Eric

Evans, where you define a “domain”, which is broken down into subdomains or “bounded contexts”. A complete discussion of DDD is outside the scope of this post, but in general terms, a “domain” is a “sphere of knowledge”. The book Building Microservices by Sam Newman has a good discussion of DDD in the microservices context. In Sam’s book, he gives an example where a domain could represent the concerns of an entire company, with the bounded contexts representing departments withing that company, such as finance and warehouse. So does one of these things correspond do a single microservice? Well, if you consider your domain as all the technological concerns of your business, then that would certainly be way too big to be considered a microservice. If you correspond a microservice to a department within a company (i.e. a warehouse microservice), that would be **Definition D**. Eric actually advocates for defining your microservices this way, at least when starting out, even though he admits that it is a bit monolithic. He then advocates eventually working toward more fine-grained microservices that represents smaller concerns, such as order fulfillment and inventory management by breaking them out later as your team gains more experience with the domain. This is more in line with **Definition E**.

También debe considerar la complejidad de su funcionalidad y observar el Principio de Responsabilidad Individual cuando defina los límites de su microservicio. Si un microservicio único se está haciendo demasiado grande e intenta hacer demasiado, probablemente sea un buen indicador de que se debe dividir en microservicios más pequeños. ¿Qué tan grande es "demasiado grande" es subjetivo, y hay muchas opiniones sobre el tema. Obtendrá una mejor comprensión de la granularidad óptima a medida que gane experiencia con microservicios en general y también con dominios comerciales específicos.

Resumen

Una buena forma de ver un microservicio individual es como una colección de funciones relacionadas o puntos finales que componen un contexto delimitado dentro del dominio de su negocio. Cada microservicio debe estar vinculado de tal manera que pueda poseer sus propios datos y operar de forma autónoma, comunicándose con otros microservicios a través de protocolos livianos solamente. El nivel de granularidad puede variar en función de la complejidad de los procesos que está modelando en su aplicación. También puede tener algunos microservicios generales transversales para proporcionar la funcionalidad requerida por otros microservicios múltiples dentro de su aplicación, siempre y cuando tenga cuidado de no utilizarlos para

compartir datos internos entre servicios o de otro modo introducir un acoplamiento estricto.

Espero que hayas encontrado este artículo útil en tu propia exploración en microservicios. Me encantaría escuchar tus opiniones en los comentarios. Por favor comparta este artículo si lo encontró útil.