Desarrollo de Sistemas Distribuidos - Arquitectura Basada en Microservicios

19 de junio de 2023

Índice

1. Lista de roles

- Jose Luis Molina Aguilar - Crítico
- Jose Luis Rico Ramos - Proponente
- David Martinez Diaz - Preguntador
- Miguel Tirado Guzman - Resumidor
- Manuel Zafra Mota - Proporcionador de Ejemplos

2. Resumen

En el siguiente trabajo abordamos la arquitectura basada en microservicios, estableciendo primero una serie de definiciones y rasgos comunes que se pueden discernir en los distintos tipos de arquitectura.
A continuación procedemos a desarrollar una serie de sub arquitecturas más importantes dentro de la arquitectura basada en microservicios aportando también ventajas e inconvenientes de cada una. A su vez, dentro de este apartado de análisis también hacemos referencias a las ventajas e inconvenientes de la arquitectura en su conjunto.
Por último damos una serie de conclusiones acerca de las ventajas e inconvenientes de esta arquitectura, cómo se usa en la actualidad y cómo será utilizada en el futuro.

3. Introducción

En las últimas décadas hemos asistido al incremento cuantitativo tanto en número de servicios como de usuarios, lo cual hace necesario un incremento en la escalabilidad de los sistemas que los sirven.
Del mismo modo, existe un mayor dinamismo en el entorno tecnológico el cual cambia constantemente, requiriendo sistemas cuyas implementaciones sean flexibles ante esta situación y sean capaces de adaptarse a mejores soluciones de forma continua.
Tradicionalmente, los sistemas se han construido siguiendo una arquitectura monolítica la cual no satisface ninguna de estas dos cualidades presentadas anteriormente, ya que es difícilmente escalable y aún más difícilmente mantenible y flexible al encontrarse todo el software unido en un mismo "cuerpo" y presumiblemente tener módulos muy acoplados unos con otros.
Todo ello, junto con el auge de la computación en la nube, hace que aparezcan otras alternativas de arquitecturas de los sistemas como es la arquitectura basada en microservicios, la cual consigue dividir la anterior arquitectura monolítica en distintos servicios independientes con sus propias tecnologías, facilitando mantenibilidad (se puede mantener cada servicio por separado), flexibilidad (se puede adaptar/mejorar cada servicio por separado sin que afecte a los demás) y escalabilidad (se pueden replicar servicios con una mayor granularidad que en la arquitectura monolítica).

4. Descripción

[2]
Nuestra inclinación natural es a definir los sistemas mediante un punto de vista monolítico. Por ejemplo, cuando pensamos en un sistema para una empresa solemos dividirlo en 3 componentes: las vistas con las que interactúan los clientes, una base de datos donde se guardará la información de cada cliente y la parte del servidor de la aplicación, la cual lleva las conexiones con los clientes (por ejemplo, en el caso de que nuestro sistema esté basado en web, despacharía las peticiones HTTP), saca la información y actualiza la base de datos, implementa la lógica del negocio... En la parte del servidor es donde está la clave para hacer la distinción entre un sistema de arquitectura monolítica o basada en microservicios.
Para el caso que hemos descrito el servidor sería un único ejecutable lógico y cualquier cambio que quisiéramos hacer a este apartado involucraría una nueva versión de esta parte de la aplicación así como volver a construirlo y desplegarlo. Si el sistema es suficientemente complejo, son operaciones muy costosas en tiempo.
A su vez, toda la lógica iría en un único proceso y podríamos usar las características básicas del lenguaje de programación como clases, funciones y espacios de nombre.
Estas características hacen que los sistemas monolíticos sean difíciles de escalar (ya que escalas todos los servicios independientemente de que haya servicios que no sean cuello de botella) y de mantener en módulos separados y solucionar pequeños errores.
Sin embargo, la arquitectura microservicios estructura la parte del servidor de la aplicación de una manera muy distinta, las aplicaciones son ahora conjuntos de servicios los cuales son independientes. Esta independencia se marca sobre todo a la hora de desplegar y escalar estos servicios.
Con esta arquitectura podemos desplegar de manera simultanea cada servicio y sus sucesivas réplicas así como actualizar cada servicio por separado, sin afectar a los demás.
Además, cada servicio puede estar escrito en un lenguaje de programación más indicado para el servicio en cuestión. Por ejemplo, si quisiéramos dar un servicio que haga cálculos matriciales complejos podríamos implementarlo en un lenguaje de programación más eficiente pero difícil de programar en él y definir el servicio para atender las peticiones HTTP de los clientes en otro lenguaje que tenga mucha más funcionalidad en este ámbito.
A continuación, se describen nueve características usualmente comunes a los microservicios:

- **Componentización mediante Servicios.** Con la arquitectura de microservicios se persigue construir sistemas mediante la unión de componentes, igual que se haría en un coche o un ordenador de sobremesa por ejemplo. La **definición de componente** que usaremos es: *"Unidad de software que es independientemente reemplazable y actualizable"*. Para realizar esta componentización usan **bibliotecas** que se definen como *"Componentes enlazados a un programa y usados usando llamadas a función"*. Pero su principal forma de componentizar es mediante **servicios, los cuales definimos** como *"Componentes fuera del proceso que se comunican mediante mecanismos como peticiones web o remote procedure call"*.

- **Organización en torno a las capacidades empresariales.** Con esto nos referimos a que en lugar de tener una organización por capas, donde por ejemplo tengamos la lógica y

ésta esté compartida por todo el sistema, en esta arquitectura se puede tener lógica en todos sitios. Los servicios tienen una implementación de software para cada área comercial incluyendo interfaz de usuario, almacenamiento...

- **Productos y no proyectos.** En lugar de que el software simplemente consista en terminar una serie de funcionalidades, cuando se implementa esta arquitectura también se propone (no siempre es obligatorio) que los mismos desarrolladores se encarguen de su mantenimiento, teniendo relación diaria con el comportamiento de su software y preguntándose cómo podría mejorar la capacidad empresarial en cuestión.
- **Extremos inteligentes y tuberías tontas.** A pesar de que últimamente se han hecho esfuerzos en agregar inteligencia al mecanismo de comunicación en sí mediante capacidades de enrutamiento de mensajes, transformación y aplicación de reglas de negocio, la comunidad de microservicios utiliza un punto de vista diferente para lograr el mínimo acoplamiento: hacen estos mecanismos de comunicación lo más simple posible.
- **Gobernanza descentralizada.** La comunidad de microservicios no persigue tanto la estandarización de las tecnologías ni el uso de sólo herramientas estandarizadas, si no que se concentran en usar herramientas útiles ya sea desarrolladas internamente o por software libre. Además, favorece la descentralización tanto de los sistemas como de los equipos de desarrollo ya que se disponen de varios servicios con sus respectivos almacenamientos lógicos de datos, recursos...
- **Gestión de datos descentralizada.** En microservicios se suele descentralizar las bases de datos en una por servicio o en una instancia de la misma tecnología de base de datos por servicios.
- **Automatización de la infraestructura.** Las técnicas de automatización de infraestructuras son ampliamente utilizadas en las arquitecturas microservicios mediante tests y despliegue automáticos, ya que al involucrar muchos procesos independientes sería tedioso llevarlo a cabo manualmente.
- **Tolerancia a fallos.** Como cada servicio por separado puede fallar, el sistema debe ser tolerante a fallos y asegurar que la caída de cualquier servicio no involucre una caída del sistema completo. También es importante detectar los fallos y restaurar los servicios lo más rápido posible.
- **Diseño evolutivo.** El diseño evolutivo de los microservicios se centra en la flexibilidad, adaptabilidad y capacidad de ser reemplazados independientemente de los demás componentes de la aplicación. Todo ello permite que los cambios, ya sea por aparición de nuevas y mejores tecnologías o nuevas necesidades, se puedan hacer de una forma mucho más ágil y controlada.

5. Análisis

Lo primero que vamos a realizar en la parte de análisis del trabajo es la especificación de los distintos patrones con la implementación de los microservicios, también después de hablar de

los distintos puntos de vista de la arquitectura, tanto ventajas como desventajas.
[3]
A día de hoy, todavía se encuentran dificultades para comprender cómo construir este tipo de arquitecturas, por lo que los desarrolladores suelen adoptar distintos patrones y para su identificación, este se encuentra respaldado por un mapeo sistemático, el cual tuvo como objetivo clasificar y comparar el contenido para poder distinguir los distintos patrones que hay como sus características.
Para ello, vamos a hablar del proceso de identificación de patrones, donde lo más común es utilizar una combinación de "snowballing" sistemático con la revisión de literatura para conseguir una visión más completa de los patrones de arquitectura basados en microservicios.

5.1. Composición de Servicios – Patrón de la API-Gateway

Este patrón se basa principalmente en que los microservicios pueden proporcionar sus funciones a otros servicios a través de una API, sin embargo, esta implementación puede requerir que para la creación de aplicaciones para usuarios se necesite un mecanismo de agregación en el lado del servidor.

La API-Gateway se trata de la entrada del sistema que se encarga de enrutar las distintas solicitudes que le lleguen de los microservicios correspondientes, también invocando múltiples microservicios y agregando resultados, por lo que dicha API será personalizada para cada cliente y se encargará principalmente de enrutar como para transformar los protocolos e implementar la lógica compartida como autenticación y los límites de velocidad, además de balancear la carga.
Su objetivo es aumentar el rendimiento del sistema y simplificar las interacciones, reduciendo así el número de solicitudes por cliente, además, actúa como punto de entrada para el enrutamiento de las solicitudes de los servicios conectados y agregando los contenidos requeridos y sirviendoles a los clientes.

Ventajas

- **Facilidad de extensión:** Ya que el patrón puede proporcionar API personalizadas, la implementación de nuevas funciones es más fácil que en otras arquitecturas.
- **Compatibilidad hacia atrás:** La puerta de enlace garantiza que los servicios en evolución no afecten a los clientes existentes con cambios en los puntos de conexión de la interfaz.

Desventajas

- **Cuello de botella:** la capa de la puerta de enlace es un único punto de entrada para todas las solicitudes, así que si no se diseña correctamente podría convertirse en un cuello de botella.
- **Escalabilidad:** cuando el número de microservicios aumenta, se necesitará un mecanismo de enrutamiento más eficiente y escalable para dirigir el tráfico a través de las APIs de los servicios así como una mejor gestión de estos.

5.2. Patrón de Descubrimiento en el Lado del Servidor.

En este patrón, los clientes realizan solicitudes a través de un balanceador de carga, que consulta el registro y reenvía la solicitud a una instancia disponible. A diferencia del patrón de API-Gateway, este patrón permite que los clientes y los microservicios se comuniquen directamente entre sí. Se basa en un Registro de Servicios, que actúa de manera similar a un servidor DNS.

El Registro de Servicios conoce la ubicación dinámica de cada instancia de microservicio. Cuando un cliente solicita acceso a un servicio específico, primero consulta el registro para obtener la ubicación del servicio; el registro se pone en contacto con el microservicio para asegurarse de su disponibilidad y reenvía la ubicación al cliente que realiza la llamada.
Finalmente, a diferencia del API-Gateway, los clientes se comunican directamente con los servicios requeridos y acceden a todas las API disponibles expuestas por el servicio.
Este patrón arquitectónico tiene varias ventajas y desventajas: Ventajas:

- **Mayor Mantenibilidad.** Todos los trabajos informaron de un aumento en la mantenibilidad.
- **Facilidad de Comunicación.** Los servicios pueden comunicarse entre sí directamente, sin interpretaciones.
- **Seguridad ante fallos.**En caso de fallo, los microservicios se pueden reiniciar fácilmente debido a sus propiedades sin estado.

Desventajas:

- **Diseño de la Interfaz Fijo.**Durante la fase de mantenimiento, los servicios individuales podrían transformarse internamente pero podría ser necesario actualizar también la interfaz, lo que requiere que todos los servicios conectados se adapten.
- **Complejidad del Registro de Servicios.**La capa del registro aumenta la complejidad, ya que requiere la implementación de varias interfaces por servicio.

5.3. Patrones de Almacenamiento de Datos – El Patrón de Clúster de Bases de Datos.

Este patrón propone almacenar datos en un clúster de bases de datos. Esto mejora la escalabilidad, permitiendo mover las bases de datos a hardware dedicado.
Para mantener la consistencia de los datos, los microservicios tienen un subconjunto de tablas de la base de datos que solo pueden ser accedidas por un solo microservicio; en otros casos, cada microservicio puede tener un esquema de base de datos privado. Su principal diferencia está en cómo se adopta la base de datos de manera interna.
El patrón fue implementado utilizando un esquema de base de datos separado para cada servicio y para replicar los datos en las bases de datos de cada servicio.

Entre las ventajas a destacar, podríamos comentar sobre todo la mejora de la escalabilidad y su recomendación para cuando se interese la implementación de un alto tráfico de datos.

En cuanto a las desventajas, el aumento de la complejidad debido a la arquitectura del clúster y mayor riesgo de fallos debido a la introducción de otro componente y al mecanismo distribuido.

5.4. Estrategias y Patrones de Implementación – El Patrón de Múltiples Servicios por Host.

Este patrón está basado en ejecutar varios servicios por el mismo host (nodo), en este enfoque los servicios coexisten en el mismo host y se benefician de la capacidad de procesamiento, memoria y almacenamiento que ofrece, al ejecutar múltiples servicios en un mismo host, se puede lograr una mayor eficiencia y utilización de recursos, ya que así se podría evitar la duplicación de infraestructuras y se aprovecha al máximo la capacidad del hardware que esté disponible.
La idea detrás de este patrón es aprovechar la capacidad de un host para ejecutar varios servicios de manera simultánea, lo que puede resultar en una mejor utilización de los recursos y una reducción de los costos operativos.

Entre sus ventajas, podemos destacar:

- **Escalabilidad.** Los sistemas pueden escalar fácilmente para implementar múltiples instancias de servicio en el mismo o en diferentes hosts.
- **Rendimiento.** Los múltiples contenedores permiten el despliegue rápido de nuevos servicios en comparación con las máquinas virtuales.

[1]
Vamos a analizar patrones relacionados con la arquitectura de microservicios.
Uno de los primeros conjunto de patrones de los que hablaremos en esta segunda parte es de los patrones de colaboración de servicios:

5.5. Colaboración de Servicios – Patrón Saga

Uno de los problemas principales que tiene el patrón de microservicios de base de datos por servicio es el propio hecho de que cada microservicio se implementa con una base de datos propia, pero imaginemos que necesitamos de transacciones donde se implementan varios microservicios.
Por ejemplo, una transacción comercial se necesita de varios microservicios, pero tiene que cumplir unas normas, como por ejemplo que no ejecute la transacción si no alcanza el dinero solicitado, pero para no se podría realizar ya que cada uno de los microservicios tiene su propia base de datos y no tiene acceso a las demás.
Para ello implementamos la solución con Patrón saga. Esta implementación nos permite juntar varios microservicios y poder acceder a la base de datos de cada uno de estos de tal manera que si en algún momento se echa para atrás por cualquier razón algo en alguna base de datos por parte de algún microservicio se puede deshacer cambios en transacción es de todos los microservicios.
Hay dos ejemplos de coordinación de sagas:

- **Coreografía:** Cada transacción local publica eventos de dominio que desencadenan transacciones locales en otros servicios.
- **Orquestación:** Un orquestador (objeto) les dice a los participantes qué transacciones locales ejecutar.

Ventajas del patrón:

- Permite que el patrón coordine y mantenga la coherencia de la base de datos en varios de los microservicios que se utilizan de forma simultánea.

Inconvenientes del patrón:

- El modelo de programación se complica cuanto más microprocesos tenga la saga.

5.6. Ventajas y Desventajas del uso de la Arquitectura de Microservicios

[4]
Como bien sabemos la arquitectura de los microservicios cada vez ha ido ganando más popularidad en los últimos años, esta ofrece una serie de ventajas sobre las arquitecturas monolíticas tradicionales. Sin embargo, también es importante tener en cuenta las posibles desventajas que conlleva. A continuación, explicaremos los aspectos positivos y negativos de los microservicios.

5.6.1. Ventajas de los Microservicios:

En primer lugar, podemos comentar que los microservicios suelen ser más simples y eficientes que las aplicaciones monolíticas, lo que puede resultar en menores costos en general, por otro lado, como los servicios son independientes, no requieren del mismo nivel de coordinación y comunicación que se necesita para las aplicaciones monolíticas.
Al permitir que las organizaciones utilicen la tecnología adecuada para cada tarea, los microservicios pueden mejorar la eficiencia y reducir la cantidad de errores. Además, estos pueden escalar horizontalmente con facilidad, ya que al tratarse de elementos pequeños y modulares, se pueden implementar mucho más rápido que las aplicaciones monolíticas tradicionales.
Por otro lado, es mucho más fácil actualizarlos que actualizar una aplicación monolítica, ya que los servicios son pequeños y autónomos, esto hace que el mantenimiento y la actualización sean menos arriesgados y menos consumidos en el tiempo.
Por último, mencionar la mayor modularidad de los microservicios, lo que puede ser una ventaja importante para las empresas y organizaciones que necesitan cambiar o transformar sus aplicaciones de manera rápida y eficaz y la mayor tolerancia a fallos, ya que si un microservicio falla, no colapsara toda la aplicación, ya que estos se gestionan y se implementan de manera independiente, lo cual es una ventaja bastante importante para aquellas organizaciones que no pueden permitirse que sus aplicaciones caigan.

5.6.2. Desventajas de los Microservicios:

Aunque los microservicios ofrecen muchas ventajas, también implica un mayor grado de complejidad, esto puede convertirse en un reto importante ya que al tratarse de una arquitectura novedosa, no hay mucha especialización en este sector, y como ya hemos mencionado anteriormente al ser independientes, puede resultar complicado el rastrear errores y resolverlos.
Los microservicios, al depender en gran medida de la red para comunicarse entre ellos, puede dar lugar a tiempos de respuesta más lentos [latencia de red] y un aumento del tráfico de la red, puede ser difícil rastrear errores que ocurren cuando múltiples microservicios se comunican entre sí.
Estos además requieren más tiempo para desarrollarse que las aplicaciones monolíticas, ya que estos son más complejos y requieren de una mayor coordinación, tienen una capacidad limitada para reutilizar código, lo que puede inducir a un mayor tiempo y costos de desarrollo debido a que estos suelen estar programados en múltiples lenguajes de programación y utilizan diferentes tecnologías.
Por último, podemos mencionar las dificultades en las pruebas y depuración globales debido a que la aplicación se encuentra distribuida en múltiples servidores y dispositivos, ya que es necesario tener acceso a todos los servidores y dispositivos que forman parte del sistema.

5.7. Casos de Uso de Aplicaciones reales

Tras haber explicado en profundidad esta arquitectura, pasaremos a hablar sobre algunos ejemplos de aplicaciones reales y sobre cómo hacen uso de los microservicios:

El primer ejemplo es Netflix, una plataforma de transmisión de contenido en línea. Cada función o servicio dentro de Netflix, como la búsqueda de películas o series, reproducción de los vídeos, la gestión de los usuarios o la recomendación de contenido, se descomponen en microservicios individuales.
Estos microservicios están diseñados con la idea de que sean independientes y autónomos, esto permite que se desarrollen, se desplieguen y escalen de forma independiente unos de otros.
Lo que resulta en poder actualizar y mejorar un servicio sin que afecte en el resto, así como la capacidad de trabajar paralelamente en distintos microservicios sin depender de un único código base.

Otra aplicación que hace uso de esta arquitectura es Uber, la cual utiliza estos microservicios para gestionar y coordinar todas las funciones clave de su servicio.
Algunas de las funciones que hacen uso de los microservicios son:
La gestión de viajes (asignar el conductor, gestión de la ruta y tiempo estimado de llegada), el servicio de pagos (procesar el pago de los usuarios), el servicio de geolocalización (proporciona los datos de ubicación en tiempo real de los conductores y usuarios, calcula las rutas óptimas, etc.) y el servicio de perfiles de usuarios (gestiona la información de los usuarios de la aplicación).
Al igual que hemos explicado anteriormente, cada uno de estos microservicios puede ser de-

sarrollado, desplegado y escalado de forma independiente, lo que permite a Uber realizar un mantenimiento y actualizar cualquiera de estos servicios sin afectar a otros. Así como gestionar la gran escala y demanda de su servicio pudiendo agregar instancias adicionales de los microservicios manejar picos altos de tráfico durante horas punta o eventos especiales, y reajustar la capacidad según sea necesario.

6. Conclusiones

Habiendo analizado tanto la actualidad en el uso de redes así como las ventajas y desventajas de la arquitectura basada en microservicios y las distintas sub arquitecturas existentes podemos llegar a las siguiente conclusiones:

- El cada vez mayor uso de dispositivos electrónicos para todo tipo de tareas, y además el creciente número de usuarios que requieren de servicios telemáticos hacen muy necesaria la aparición de este tipo de arquitecturas, centrándose sobre todo en su carácter escalable.
- El incremento en complejidad de las aplicaciones también hace necesario el uso de este tipo de arquitecturas, que ayudan en su mantenimiento al mantenerlas separadas y reducen en gran medida el acoplamiento. A su vez, el carácter divisor de esta arquitectura ayuda a que si un servicio comienza a crecer de manera desorbitada y sea suficientemente grande como para poder dividirlo en componentes pueda dividirse en distintos servicios.

  Aunque una de las desventajas es que no hay ningún criterio empírico por el cual el desarrollador software conozca exactamente cuando y como debe dividir una aplicación en servicios o un servicio en distintos subservicios. La falta de tales mecanismos es uno de los grandes retos futuros de esta arquitectura.
- Las arquitecturas basadas en microservicios propician también que un mismo servicio pueda pasar por distintos equipos de desarrolladores al ser un mecanismo descentralizado. Ésto puede llevar a la degradación arquitectónica del sistema.
- Sin embargo, también trae problemas debido a la naturaleza de la solución que propone: Al tener cada sistema una gran cantidad distinta de servicios los cuales se comunican entre sí por la red, pueden crearse problemas de sobrecarga de la red.
- La integridad en los datos también es difícil de mantener en las arquitecturas donde las bases de datos se instancian en distintos servicios al mismo tiempo o incluso en aquellas donde cada servicio tenía su propia base de datos.

La arquitectura basada en microservicios tiene grandes ventajas basadas en escalabilidad, flexibilidad y mantenibilidad las cuales la hacen suficientemente consistente como para competir frente a las arquitecturas monolíticas a pesar de sus desventajas (muchas de las cuales se deben al todavía poco desarrollo de herramientas para esta arquitectura).
Además, si tenemos en cuenta las últimas tendencias como son la computación basada en la nube descubrimos que es una arquitectura cuyo uso aumentará aún más en el futuro.

7. Autoevaluación final

Este proyecto ha sido realizado conforme a las indicaciones del trabajo del tema 1 disponible en prado pero con una mayor extensión debido al mayor nivel de detalle analizado en este trabajo. Nos hemos proporcionado apoyo entre los distintos componentes del grupo cuando ha sido necesario, consiguiendo un nivel de entrega similar entre todos los integrantes del grupo en la realización del proyecto.

Referencias

[1] *Microservice Architecture*. https://microservices.io/patterns/microservices.html.

[2] *Microservices a definition of this new architectural term*. https://martinfowler.com/articles/microservices.html.

[3] Davide Taibi, Valentina Lenarduzzi y Claus Pahl. "Architectural Patterns for Microservices: A Systematic Mapping Study". En: mar. de 2018. DOI: 10.5220/0006798302210232. URL: https://www.researchgate.net/publication/323960272_Architectural_Patterns_for_Microservices_A_Systematic_Mapping_Study.

[4] *What Are Advantages & Disadvantages of Microservices Architecture?* https://www.orientsoftware.com/blog/microservices-advantages-and-disadvantages/.