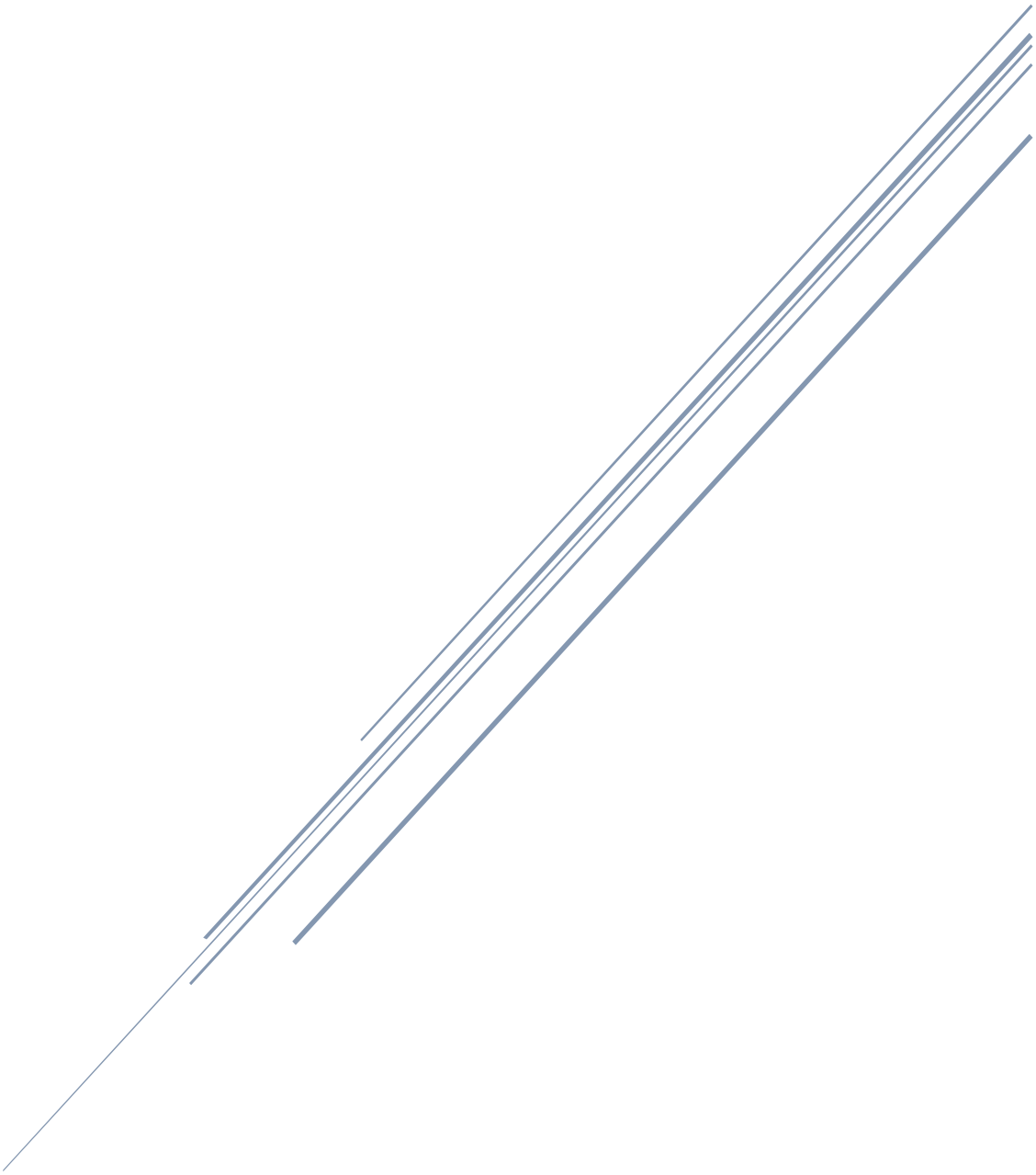
Microservices trade-offs



Covadonga Vega Fernández-UO257507
Andrés del Pozo Amo-UO271035

Índice

DEFINICIONES

Para entender mejor los temas a tocar en esta presentación es necesario introducir una serie de conceptos previos.

- ## Monolito de implementación

Es una aplicación que se desarrolla a partir de una arquitectura de capas o MVC, y todo el código está en un mismo repositorio con una sola BBDD.

- ## Microservicio

Es la misma aplicación separada por diferentes partes pequeñas que son independientes entre sí, cada uno con su respectiva BBDD, y se pueden desarrollar de manera independiente que luego se comunican entre sí.

INTRODUCCIÓN

Los orígenes del uso de microservicios no son muy claros pero un factor de influencia fueron algunos de los hiperescaladores populares.

Amazon, Netflix y algunos otros hiperescaladores se vieron obligados a solucionar algunos de los problemas encontrados en los entornos **TI** (Tecnología de la Información).

Algunos de ellos fueron los siguientes:

- Necesitaban escalar sus servicios de TI a un tamaño que nadie más había hecho. Las arquitecturas de TI conocidas y populares no funcionaron a esa escala.
- Al mismo tiempo, necesitaban moverse extremadamente rápido en sus mercados altamente competitivos.
- Finalmente, muchos de sus clientes confiaron en sus servicios, lo que significó que, comprometer la calidad de sus sistemas de TI por tiempos de ciclo más cortos, no era una opción.

El resultado de su esfuerzo fue el uso de los **microservicios**, un estilo de arquitectura basado en servicios que tenía algunas propiedades diferentes a las **implementaciones SOA** (Arquitecturas Orientadas a Servicios) tradicionales habituales.

Se descubrieron muchos beneficios como son:

- L**ímites de módulo sólidos**: Los microservicios refuerzan la estructura modular, que es particularmente importante para equipos más grandes.
- **Implementación independiente**: Donde los servicios simples son más fáciles de implementar y, dado que son autónomos, es menos probable que causen fallas en el sistema cuando fallan.
- **Diversidad tecnológica**: Con los microservicios se pueden combinar varios lenguajes, marcos de desarrollo y tecnologías de almacenamiento de datos.

Pero a lo largo de sus esfuerzos de implementación, también aprendieron que los microservicios no eran tarea fácil. Algunas de las complicaciones con las que se encontraron fueron las siguientes:

- **Distribución:** Los sistemas distribuidos son más difíciles de programar, ya que las llamadas remotas son lentas y siempre corren el riesgo de fallar.
- Coherencia eventual: Mantener una coherencia sólida es extremadamente difícil para un sistema distribuido, lo que significa que todos deben gestionar la coherencia final.

- **Complejidad operativa:** Necesita un equipo maduro para administrar muchos servicios, que se redistribuyen con regularidad.

Finalmente, los hiperescaladores que adoptaron microservicios comenzaron a hablar sobre lo que hacían, fueron a conferencias, escribieron artículos en revistas de TI escribieron en sus blogs de desarrolladores, etc.

Junto con esto, evolucionaron varios esquemas de justificación para la introducción de microservicios. Los más extendidos son:

- Los microservicios son necesarios para la escalabilidad.
- Los microservicios son más simples que los monolitos.
- Los microservicios mejoran la reutilización.
- Los microservicios mejoran la autonomía del equipo.
- Los microservicios conducen a un mejor diseño.
- Los microservicios facilitan el cambio de tecnología.

Desafortunadamente, todas son falacias, o están completamente equivocados, o al menos no necesita microservicios, ni el precio a paga por ellos, para lograr las propiedades prometidas.

Falacias de los microservicios

1.1. ESCALABILIDAD

Enlace: https://www.ufried.com/blog/microservices_fallacy_2_scalability/

"Se necesitan microservicios para abordar los problemas de escalado"

Una de las justificaciones más utilizadas para los microservicios es la escalabilidad: "Necesitamos microservicios para escalar nuestra aplicación dinámicamente".

Las discusiones acerca de escalabilidad están casi siempre relacionadas con escenarios en los que un gran número de usuarios poseen acceso a la aplicación de una empresa, habitualmente aplicaciones web o móviles.

Muchas personas defienden que los problemas de escalabilidad deben solucionarse cambiando la arquitectura a microservicios, buscando soluciones complejas y elaboradas, cuando cuidando un poco el diseño y retocando algunos detalles como el equilibrio de carga, se podrían solucionar casi por completo los problemas de la escalabilidad.

1.2. SENCILLEZ

Enlace: https://www.ufried.com/blog/microservices_fallacy_3_simplicity/

"Las soluciones se vuelven más simples con los microservicios"

Otra justificación generalizada de los microservicios es que serían más simples que los monolitos.

Pasar a los microservicios significa que sus aplicaciones se convierten en sistemas distribuidos con lo que eso supone: asumir todas las complejidades derivadas de las llamadas remotas.

Lo que se necesita para unidades de trabajo de desarrollo más simples es la **modularización** (dividir el proyecto en módulos). Por lo cual, no es imprescindible un estilo de arquitectura de tiempo de ejecución específico como microservicios.

También es importante recordar que la **complejidad esencial** (complejidad para encontrar una soluciona al problema) no desaparece porque divida la solución en partes más pequeñas.

Cuanto más pequeñas se vuelven las partes, más compleja es la estructura y, aún más si se realizan llamadas remotas. Con lo cual, los microservicios no conducen a soluciones más simples, sino que conducen a soluciones mucho más complejas.

1.3. REUTILIZACIÓN

Enlace: https://www.ufried.com/blog/microservices_fallacy_4_reusability_autonomy/

"Los microservicios mejoran la reutilizabilidad y la amortización"

La reutilización se utiliza a menudo como argumento para vender diferentes tipos de paradigmas de modularización a los responsables de la toma de decisiones: El alto grado de reutilización que se esperaría del nuevo paradigma garantizaría que las inversiones iniciales necesarias para introducir el nuevo paradigma se amortizarían muy pronto.

Al utilizar reutilización de codigo puede darse un **acoplamiento apretado**. Es la forma más estricta de acoplamiento que existe, y si se produce un error en la pieza reutilizada, la parte de reutilización también falla inevitablemente, es decir, no es capaz de completar su trabajo.

Para garantizar la disponibilidad de sistemas distribuidos como aplicaciones basadas en microservicios, se debe evitar estos **errores en cascad,** es decir bloquear el microservicio que falle y dejando a los demás servicios funcionar con normalidad.

1.4. AUTONOMÍA

Enlace: https://www.ufried.com/blog/microservices_fallacy_4_reusability_autonomy/

"Los microservicios mejoran la autonomía del equipo de desarrollo"

Toda la idea de que los microservicios mejoran la autonomía de los desarrolladores comienza en el extremo equivocado. La autonomía comienza a nivel organizativo.

En primer lugar, es necesario organizarse de una manera que los equipos puedan tomar decisiones sin tener que coordinarse con personas ajenas al equipo. Para llegar allí, debe implementar una gran cantidad de requisitos previos. Los dos más importantes son:

- Necesidad de establecer una autoridad de decisión descentralizada: Las organizaciones jerárquicas tradicionales con procesos de toma de decisiones centralizados impiden la autonomía del equipo por definición.

- Organización de la fuerza de trabajo de una manera que la toma de decisiones descentralizada sea posible y proporcione una clara ventaja sobre la toma de decisiones centralizada.

Como conclusión, se entiende que los microservicios por sí solos no permiten ni mejoran la autonomía. La autonomía debe establecerse a nivel organizativo, funcional y de gobernanza.

1.5. Diseño

Enlace: https://www.ufried.com/blog/microservices_fallacy_5_design/

Esta falacia se refiere a que los microservicios conducen a un mejor diseño de la solución.

Se dice que en el contexto de los microservicios es que los microservicios conducen a un mejor diseño de la aplicación.

Cuando en realidad es la estructura de tiempo de ejecución elegida no impide de ninguna manera la estructura de su código fuente. Las unidades estructurales en tiempo de ejecución normalmente son al menos tan grandes como las unidades estructurales del código fuente, generalmente más grandes. Nuestras herramientas de compilación suelen permitir asignaciones arbitrarias desde el código fuente a las unidades de tiempo de ejecución. Esto significa que puede organizar y estructurar su código fuente de forma completamente independiente de la estructura de tiempo de ejecución elegida.

Sin embargo, los monoliths rara vez están bien estructurados, debido a la falta de conocimiento, experiencia y disciplina d ellos ingenieros de software involucrados.

Podemos concluir con el razonamiento anterior de los microservicios no es verdad ya que no conducen a un mejor diseño, si no lo contrario ya que necesita mejores habilidades de diseño.

La solución es un diseño basado en dominios para resolver todos los problemas de diseño.

1.6. Tecnología

Enlace: https://www.ufried.com/blog/microservices_fallacy_6_technology/

Esta falacia se refiere a que los microservicios facilitan cambios tecnológicos.

1.6.1 Cambio de tecnología más fácil

Se discute que los microservicios facilitan el cambio y la exploración de nuevas tecnologías.

Esto es cierto ya que las unidades de tiempo en ejecución más pequeñas facilitan la reescritura de una sola unidad en una nueva tecnología.

En la publicación habla de que el principio de uniformidad de esfuerzo de migración de tecnología es cierto, ya que se necesita el mismo esfuerzo para migrar un sistema existente a una nueva tecnología que para desarrollar el sistema hasta tu estado actual.

Una de las ventajas que tiene esto es que puede estirar mejor la migración, ya que puede migrar la aplicación servicio por servicio, por lo tanto, de esta forma, la descomposición de la aplicación en servicios apoya la gestión de riesgos.

Otra de las ventajas es que puedes probar nuevas tecnologías de manera simple reescribiendo un solo servicio utilizando la nueva tecnología, sin embargo, no puede ejecutar muchas tecnologías diferentes a la vez, ya que aumenta la dificultad.

1.6.2 Controladores que impiden las actualizaciones

- **Procesos de gobernanza rotos:** un defecto de este controlador es que las aplicaciones basadas en software se confunden con bienes físicos, como consecuencia a esto se descuida el hecho de que el software deba cambiarse continuamente para no perder su valor debido a que los bienes físicos no se cambian.
- **Dependencias de la aplicación en el nivel del sistema operativo:** Anteriormente el sistema operativo era un programador de recursos, asegurándose que los recursos que obtiene son los recursos que necesita y con el tiempo se volvió imposible crear una aplicación que no tuviera dependencia del sistema operativo.
  Hasta que se creó la tecnología de contenedores que lo que hace es que se reajusta la línea de separación entre la aplicación y el sistema operativo, es decir, que la aplicación con todas sus dependencias de tiempo de ejecución, se incluyen en el contendor mientras que el sistema operativo coordina el acceso a computación, almacenamiento red y otros recursos para un conjunto de aplicaciones.

1.7. Uso de los microservicios

Enlace: https://www.ufried.com/blog/microservices_fallacy_7_actual_reasons/

Esta falacia justifica el uso de los microservicios.

1.7.1 Entornos en los que los microservicios despliegan sus puntos fuertes

- **Moverse rápido:** Para elegir los microservicios lo más importante es que lo ayudan a moverse más rápidos en mercado dinámico y altamente competitivo, es decir, que necesitan moverse rápido para adaptar de manera continua y rápida su oferta de servicios a las necesidades y demandas de sus usuarios.

  Para moverse rápido, se organizan en equipos de capacidad orientados al mercado, permitiendo que vayan a su propia velocidad y si se dividen correctamente, las capacidades de cara al mercado son independientes entre sí. Esto minimiza la coordinación entre equipos dando lugar a una relentización.

- **NFR muy dispares:** Los NFR son los requerimientos no funcionales.
  Este entorno se basa en la capacidad de dividir su aplicación en múltiples partes, cada una de las cuales implementa NFR muy diferentes. Esto hace que las diferentes partes sean más simples, ya que solo necesitan preocuparse por su conjunto relevante de NFR.

1.8. Cuando se usan los microservicios

Enlace: https://www.ufried.com/blog/microservices_fallacy_8_choices/

Esta falacia da sentido al cuando usar los microservicios, cuando no usaros y que hacer en su lugar.

1.8.1 Cuando usar microservicios

- Si se necesitan tiempos de ciclo muy cortos.
- Si se necesita ir rápido y tienes varios equipos o esperas crecer en un futuro.

1.8.2 Cuando no usar microservicios

- Si no se necesitan ciclos de retroalimentación muy cortos con el mercado en términos de lanzar nuevas funciones.
- Si eres solo un equipo, porque no te dan ninguna ventaja.
- Si no están dispuestos o no se pueden hacer todas las cosas, por lo tanto se puede optar por un estilo arquitectónicos mas simple y que se adapte a sus necesidades.

1.8.3 Alternativas a los microservicios

- **Moduliths (Falacia 9).**
- **Microliths (Falacia 10).**

1.9. Moduliths

Enlace: https://www.ufried.com/blog/microservices_fallacy_9_moduliths/

En esta falacia se habla de los moduliths como alternativa a los microservicios.

Un modulith de implementación es la arquitectura en tiempo de ejecución más simple.

El problema que tienen los moduliths de implementación es que su estructura interna tiende a deteriorarse con el tiempo, por lo tanto, hace que la evolución y el mantenimiento sea muy pesado.

Para optar por los moduliths tendríamos que tenerlos formados por módulos y claramente definidos y aislados, pero esto no es gratis.

En los moduliths, se requieren habilidades de diseño para encontrar los limites y contenidos correctos del módulo, para decidir que concepto funcional implementar, dónde y cómo exponerlo al exterior.

También se requiere que todos se adhieran a las reglas, es decir, que no violen la encapsulación del módulo. Esto significa que se tiene que acceder a los módulos solo a través de sus interfaces oficiales.

Las actualizaciones tecnológicas pueden volverse difíciles para un modulith de implementación grande.

Si su diseño interno está impulsado por la reutilización, terminará con muchos módulos reutilizándose entre sí. Si lo hace bien obtendrá una estructura del módulo en capas internas donde cada módulo de una capa superior se basa en las funcionalidades proporcionadas para la capa de abajo.

El defecto de si no se crea una buena capa supone que la actualización sea difícil. La clave es que las actualizaciones no se vuelvan difíciles debido a la estructura de en tiempo de ejecución de los moduliths.

Otra opción sería impulsar el diseño interno por independencia mutua, que da lugar a una modularización impulsada por casos de uso donde diferentes módulos se encapsulan cosas de uso completos o interacciones de usuario.

En conclusión, las actualizaciones probablemente serán un poco más difíciles con los módulos que con los microservicios.

1.10. Microliths

Enlace: https://www.ufried.com/blog/microservices_fallacy_10_microliths/

En esta falacia se habla de los microliths como alternativa a los microservicios.

Los microliths pueden ser una opción si tiene requisitos de tiempo de ejecución especiales en términos de NFR muy dispares. También pueden ser una opción si se cumplen las condiciones previas para usar microservicios, pero no puede o no esta dispuesto a pagar el precio por los microservicios.

Un microlith es un servicio que se diseña utilizando los principios de diseño de módulos independientes que evita llamadas entre módulos o servicios mientras se procesa una solitud externa.

Se requiere un diseño de independencia mutua, ya que no se permite que los microliths se llamen entre si mientras procesan una solicitud externa que puede resultar desconocida para los ingenieros acostumbrados a pensar en capas, la reutilización y las técnicas de dividir y conquistar basadas en la pila de llamadas.

La pila de llamadas:

- Toda la funcionalidad necesaria para un caso de uso debe implementarse dentro de un microlith.
- Todos los datos necesarios para atender una solitud externa deben estar en su base de datos.
- La funcionalidad reutilizable debe proporcionarse a través de bibliotecas o alguna otra implementación o mecanismo de modularización del tiempo de construcción no a través de otros servicios.

Los microliths facilitan mejor cambios tecnológicos que los moduliths porque son más pequeños, por tanto, las unidades de migración son más pequeñas.

Por tanto, los microliths son una opción si no necesita ir rápido o no tiene varios equipos y no tiene NFR muy dispares para diferentes partes de su aplicación.