Speech Microservicios vs Monolíticos:

Introducción:

Como fue mencionado previamente en la introducción a la primera unidad de patrones que estuvimos viendo; los patrones, en este caso de arquitectura, son soluciones que ya fueron probadas y comprobadas para problemas comunes o frecuentes, en el diseño de software. Los patrones de arquitectura, explican cómo estructurar nuestras aplicaciones de manera eficiente y facilitar su mantenimiento a un plazo mediano-largo.
Estos patrones nos brindan un conjunto de reglas y directrices que nos ayudan a tomar decisiones informadas al diseñar nuestras aplicaciones, permitiéndonos abordar desafíos como la escalabilidad, la seguridad, la modularidad y la interoperabilidad.
En este caso, veremos en detalle lo que son los sistemas monolíticos y sistemas de microservicios, y cómo se pueden aplicar en diferentes contextos.

Monolíticos:

Para comenzar vamos a ver los sistemas con arquitectura monolítica.
La arquitectura monolítica es un enfoque tradicional de desarrollo de software en el que una aplicación se construye como una sola unidad indivisible.
En este tipo de arquitectura, todas las funcionalidades se desarrollan, despliegan y ejecutan como un solo componente.
En este enfoque, todo el código fuente se agrupa en un único monolito, y los diferentes módulos y componentes están estrechamente acoplados.
Esto significa que cualquier cambio en una parte del sistema puede tener efectos en otras áreas, lo que puede dificultar la escalabilidad y la mantenibilidad a medida que la aplicación crece en tamaño y complejidad.
Sin embargo, los sistemas monolíticos pueden ser más simples de desarrollar y desplegar en entornos más pequeños o para aplicaciones más simples.

Entonces, como características principales, o características a destacar podríamos tener:

1. **Alto acoplamiento:** Tenemos todo desplegado de manera conjunta, esto lleva a que se eleve la dificultad de escalar y mantener el sistema.
2. **Escalabilidad limitada:** Básicamente, si nosotros necesitamos escalar un componente específico, vamos a precisar escalar si o si, todo el sistema. Por otro lado, si un componente está generando problemas o tiene un mayor tráfico que los demás, de igual forma va a estar afectando a todo el sistema, ya que está todo conectado.
3. **Despliegue y cambios demorados:** Por último, vamos a tener demoras a la hora de desplegar el sistema ya que por más de que tengamos una parte completa, necesitamos terminar todos los componentes que tengamos dentro para poder tener la aplicación 100% funcional. Y por otro lado, a la hora de hacer un cambio o actualización, corremos riesgo de que afecte todo el sistema, y también para actualizar una parte, vamos a tener que bajar toda la app.

Microservicios:

Pasando a lo que son los microservicios, ya si vemos las dos imágenes, nos podemos dar una idea de por donde viene la mano, pero, para dejarlo claro vamos a explicarlo.
En este modelo, una aplicación se divide en pequeños servicios autónomos e independientes, cada uno con su propia funcionalidad y ejecutándose en su propio proceso. Estos servicios se comunican entre sí a través de interfaces bien definidas, generalmente a través de APIs.
Cada servicio puede ser desarrollado, desplegado y escalado de forma independiente, lo que permite una mayor flexibilidad y modularidad.
También permite a los equipos de desarrollo trabajar de manera más autónoma en diferentes servicios. Esta arquitectura favorece la escalabilidad y la mantenibilidad, ya que los microservicios se pueden escalar y actualizar de forma individual sin afectar a otros componentes.
Sin embargo, también introduce una mayor complejidad en cuanto a la gestión de la comunicación entre los servicios y puede requerir una infraestructura más robusta.

Entonces, como características principales, o características a destacar podríamos tener:

1. **Modularidad y desacoplamiento:** Por lo que veníamos comentando, todo está separado, por lo que obtenemos una mayor modularidad y también desacoplamiento ya que no está todo junto y dependiente.
2. **Favorece la escalabilidad:** Esto es debido a que todos los componentes están por separado, por lo que sí tenemos que sacar una actualización o un cambio, podemos hacerlo libremente sin riesgo de afectar los otros servicios. Además, no tenemos que tener offline más que el servicio a actualizar.
3. **Mayor complejidad en la gestión:** En pocas palabras, esto se debe a que en vez de tener un sistema, vamos a tener varios pero más pequeños, por lo que todo lo que es gestión de dependencias, monitoreo, observabilidad, despliegue y escalado, vamos a tener que realizarlos por cada servicio.

Comparación:

Acoplamiento:

- Sistema Monolítico: El acoplamiento es alto en un sistema monolítico, ya que todos los componentes y funcionalidades están estrechamente integrados dentro de una única aplicación. Los cambios en una parte del sistema pueden tener un impacto en otras áreas relacionadas.
- Sistema de Microservicios: El acoplamiento es bajo en un sistema de microservicios, ya que cada microservicio es independiente y se comunica a través de interfaces bien definidas. Los cambios en un microservicio no afectan directamente a otros, lo que permite la evolución y actualización individual de cada componente.

Escalabilidad más común:

- Sistema Monolítico: La escalabilidad en un sistema monolítico se logra generalmente mediante el escalado vertical, es decir, aumentando los recursos (por ejemplo, CPU, RAM) en el servidor donde se ejecuta la aplicación monolítica.
- Sistema de Microservicios: La escalabilidad en un sistema de microservicios se logra generalmente mediante el escalado horizontal, donde se agregan más instancias de microservicios según sea necesario. Cada microservicio puede escalar de forma independiente según su carga y requisitos.

Mantenibilidad:

- Sistema Monolítico: La mantenibilidad en un sistema monolítico puede ser desafiante, ya que todos los componentes están interconectados. Los cambios y correcciones pueden requerir pruebas exhaustivas de todo el sistema y pueden tener un impacto en áreas no deseadas.
- Sistema de Microservicios: La mantenibilidad en un sistema de microservicios suele ser más sencilla, ya que cada microservicio es independiente y se puede modificar, probar y desplegar de forma aislada. Los cambios en un microservicio no afectan a otros, lo que facilita el mantenimiento y la evolución del sistema.

Despliegue:

- Sistema Monolítico: El despliegue de un sistema monolítico implica desplegar toda la aplicación en un único proceso. Esto puede requerir tiempo y esfuerzo, y cualquier error en el despliegue puede afectar a todo el sistema.
- Sistema de Microservicios: El despliegue de un sistema de microservicios implica desplegar cada microservicio de forma independiente. Esto permite actualizaciones más rápidas y frecuentes, y reduce el impacto de los errores, ya que solo afectarían a un microservicio específico.

Comunicación:

- Sistema Monolítico: La comunicación entre componentes en un sistema monolítico es principalmente a través de llamadas internas en la misma aplicación. La comunicación es directa y rápida.
- Sistema de Microservicios: La comunicación entre microservicios en un sistema de microservicios se realiza a través de protocolos de comunicación (como HTTP/REST, mensajería, etc.). La comunicación puede ser más lenta debido a la necesidad de invocar microservicios remotos.

Complejidad de gestión:

- Sistema Monolítico: En un sistema monolítico, la gestión tiende a ser más simple y centralizada. Los componentes están estrechamente integrados y administrados dentro de una única aplicación, lo que facilita el seguimiento de cambios, pruebas, implementaciones y supervisión del sistema en su conjunto.
- Sistema de Microservicios: En un sistema de microservicios, la gestión puede ser más compleja y descentralizada. Se deben administrar y coordinar múltiples microservicios independientes, lo que implica realizar un seguimiento de sus versiones, implementaciones, comunicación entre ellos y supervisión individual de cada microservicio. Además, se necesita establecer una infraestructura adecuada para gestionar el descubrimiento, la escalabilidad y la resiliencia de los microservicios.

Tolerancia a fallos:

- Sistema Monolítico: En un sistema monolítico, un fallo en una parte del sistema puede afectar a todo el sistema, ya que todos los componentes están acoplados. Esto puede dificultar la recuperación y la tolerancia a fallos.
- Sistema de Microservicios: En un sistema de microservicios, debido a su naturaleza distribuida, los fallos en un microservicio tienen un impacto limitado a ese microservicio en particular. Otros microservicios pueden seguir funcionando correctamente, lo que facilita la tolerancia a fallos y la recuperación parcial del sistema.

Tiempos de desarrollo:

- Sistema Monolítico: En general, el desarrollo de un sistema monolítico puede requerir menos tiempo inicial, ya que todas las funcionalidades se desarrollan y prueban dentro de una única aplicación. Además, las pruebas y las integraciones son más sencillas, ya que no hay dependencias externas entre componentes.
- Sistema de Microservicios: El desarrollo de un sistema de microservicios puede llevar más tiempo inicialmente debido a la necesidad de diseñar y desarrollar varios microservicios independientes. Además, las pruebas y las integraciones entre microservicios pueden ser más complejas debido a las interacciones entre ellos.

Equipo de desarrollo:

- Sistema Monolítico: En un sistema monolítico, el equipo de desarrollo puede ser más pequeño y cohesionado, ya que todos los desarrolladores trabajan en la misma aplicación. La comunicación y la coordinación son más sencillas debido a la estrecha integración de los componentes.
- Sistema de Microservicios: En un sistema de microservicios, se requiere un equipo de desarrollo más grande y diverso, ya que cada microservicio puede requerir diferentes habilidades y conocimientos. La comunicación y la coordinación entre los equipos de desarrollo de los microservicios pueden ser más desafiantes debido a su naturaleza distribuida.

Mezcla entre los dos:

Sí, es posible combinar elementos de ambas arquitecturas en lo que se conoce como una estrategia híbrida. Por ejemplo, se puede tener una aplicación monolítica principal y utilizar servicios de microservicios para ciertas funcionalidades específicas o componentes complementarios. Esto permite aprovechar la **simplicidad y la cohesión** de una **arquitectura monolítica**, al tiempo que se obtiene la **flexibilidad y la escalabilidad** de los **microservicios** en áreas específicas de la aplicación.
Sin embargo, la combinación de ambas arquitecturas puede agregar complejidad adicional, por lo que se debe evaluar cuidadosamente si es la mejor opción en un caso particular.

Bibliografia:

- *¿Qué son los microservicios? | IBM*. (n.d.).

https://www.ibm.com/es-es/topics/microservices