T6: Microservicios

martes, 6 de agosto de 2024 20:04

Siguen sin ser la solución definitiva, es de hecho otro tipo de arquitectura; son bastante complejos de hecho.

Me quiero morir (son los primero apuntes que tomo en todo el curso de esta asignatura)

Los microservicios se basan en la idea de mayor cohesión (independencia de otros módulos) (principio de responsabilidad única, "un módulo debe ser responsable SOLO y UNICAMENTE de UNA funcionalidad") menos acoplamiento (alta dependencia entre módulos).

Una idea totalmente contraria son las arquitecturas monolíticas, donde el código del sistema se despliega como un único proceso, lo que nos lleva a problemas de:

- Escalabilidad
- Acoplamiento
- Conflictos entre desarrolladores o Delivery contention = Ownership y toma de decisiones…
- Acoplamiento en despliegue
- Complicaciones en CI/CD

Sin embargo no es una solución horrible… es solo otro tipo de arquitectura, que como problemas, tiene también ventajas:

- Proporciona una topología de despliegue sencilla… = todo se despliega como uno solo
- Monitorización sencilla
- End-to-end testing ya que el flujo de desarrollo es claro
- Rendimiento
- Reutilización del código

De este tipo de Arquitectura Monolítica hay además tipos:

- Single Process Systems
  - Aunque pueda haber varias instancias para que sea robusto, sigue siendo un único proceso… esto no es "escalable".
- Monolito Distribuido
  - Son múltiples servicios desplegados todos juntos --> Tiene lo peor de ambos mundos.
- Third Party black box system
  - Es SW que ha sido desarrollado por otros y no tenemos acceso a él.

Pag:

Microservicios:

Es una Arquitectura Orientada a Servicios (SOA), lo que se traduce en palabras sencillas en que las funcionalidades se modelan en un dominio de negocio y que desde fuera este proceso debe ser una caja negra (donde se oculta la mayor cantidad de información y endpoints); con objetivo principal de encapsular las funcionalidades.

Paréntesis - Caso Práctico-: En una empresa tradicional, con una arquitectura/organización de equipos tradicional, se divide en, módulos del sistema (3 por ejemplo, front, back, ddbb), y cuando se requiere que se añada/elimine/edite una funcionalidad estos 3 módulos se ven afectados, y los 3 equipos se deben coordinar…

La propuesta organizativa para que se puedan implementar microservicios, consiste en: un equipo que en las 3 verticales que deben ser abordadas por ellos para integrar esa nueva funcionalidad.

Otro asunto sobre los microservicios es que intentan resolver los problemas de acoplamiento:

- Implementation Coupling: Es cuando B ---depende de ---> A
  - Aunque es el peor tipo, es el más fácil de solucionar; por ejemplo colocar una API que gestione la interacción
- Temporal Coupling: Es cuando A --- tercero C ---> B
  - Consiste en peticiones anidadas, y su mayor problema, es que todas las partes deben estar siempre disponibles; aunque se puede solucionar, cacheando las peticiones y respuestas, siguiendo un modelo asíncrono.
- Deployment Coupling: Es cuando un cambio pequeño sugiere que todo el sistema tiene que ser re-desplegado.
- Domain Coupling: Es uno de los más comunes y casi más inevitables… dado que significa la interacción de servicios en distintos dominios

Además de resolver esto, los microservicios ofrecen ventajas como:

- La tecnología es heterogénea, pero eso significa que es apropiada y ajustada a cada caso
- Es escalable
- Fácil despliegue
- Contribuye con la alineación de los equipos y las necesidades del negocio
- Componibilidad (La funcionalidad se puede consumir para diferentes propósitos)

Para que estos funcionen depende de la arquitectura organización de la empresa y la distribución de los equipos para gestionar los sistemas desde un punto de vista de servicios.

Además no es perfecto, tiene ciertas desventajas, como:

- La experiencia de desarrollo
- El reporting por lo mismo de que es complicado el monitoring
- Testing
- Latencia, la comunicación entre los microservicios puede ser un problema
- La consistencia entre los datos, sobretodo si la base de datos es distribuida (ya que si fuera centralizada haría cuellos de botellas).
- Puede resultar más costoso, ya que habrán más recursos desplegados (máquinas, red, almacenamiento…)

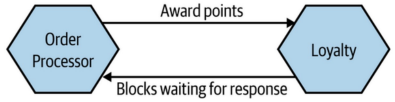
Sobre las comunicaciones en los microservicios:

Son mucho más ineficientes dado que la comunicación es ENTRE PROCESOS "from in-process to inter-process".

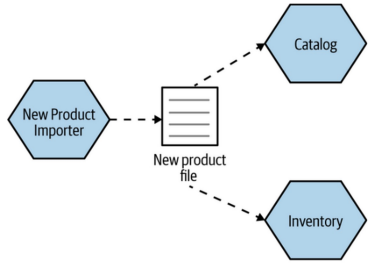
Tiene muy mal rendimiento ya que no se pueden optimizar los compiladores, hay retardos, puede "overheat" los sistemas… lo que also condiciona a que hay que tener cuidado con el nº de llamadas…

Los tipos de comunicación:

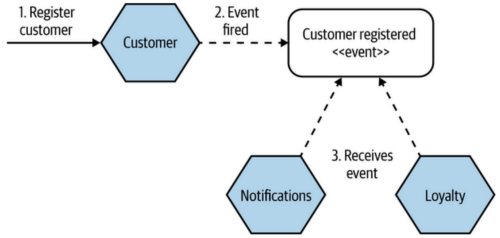
- Bloqueo síncrono: Es una llamada de toda la vida, que se bloquea hasta que obtiene la respuesta a la petición.



- Asíncronas no bloqueantes: Son la mejor opción (mi opinión) ya que el proceso no se bloquea ante la petición, aunque suponen un desarrollo más complicado.
- Common Data: Colocar a disposición de los microservicios, una fuente de datos común y centralizada de lectura y/o escritura (ya tu sabrás como gestionan los permisos…), esta puede ser un fichero, un datalake, un datawarehouse…



- Request-Response: Técnicamente es hacer una request previa, para solicitar la información, realizar la acción, o lo que sea, es como reservar un turno.
- Event Driven: (Es como un observer) cuando un "evento" ha sucedido/se ha completado, un microservicio notifica a los "clientes" que lo estén escuchando. Es asíncrono. Implemente "message brokers"



Algunas tecnologías para desplegar los microservicios: (Deben desplegarse cada uno como un proceso… pueden ser como contenedores, como FaaS (como las Lambdas de AWS…))

- REST
- Message Brokers
- Remote Procedure Calls (RPC)
  - SOAP
  - GRPC