***Microservicios***

**Teoría**:

1. *Introducción a arquitectura:*

En el mundo de la implementación de soluciones por software, anteriormente solo se programaban sistemas monolíticos, estos son sistemas o aplicaciones grandes que hacen muchas cosas a la vez pero no son mantenibles. De esta manera, cualquier pieza de software que escribamos va a tener un mal desempeño ya que conforme crezca su falta de mantenimiento también crecerá el costo de mantenerla funcionando, esto genera como consecuencia percibir menor beneficio de la solución que mayores utilidades.

Por esta razón, los desarrolladores y empresas ahora prefieren migrar de sistemas monolíticos a sistemas orientados a microservicios. Esto significa separar las funciones de un negocio o componentes de una aplicación en un contexto delimitado por servicio, así queda restringuido el funcionamiento de cada parte de la aplicación.

Los microservicios son amigables a las migraciones graduales. Esto toma bastante importancia cuando los clientes o sistemas son complejos o grandes. Además, este tipo de modelo es agnóstico a la tecnología ya que es posible utilizar cualquier lenguaje de de programación para desarrollar un microservicio. Por otro lado, estos son considerados ligeros *(lighterweight)* ya que trabajan aislados de otros componentes logrando consumir los recursos necesarios.

Los principios de microservicios son encapsulación, automatización, dominio céntrico, descentralización, independiente, a prueba de fallos y observable.

El punto principal de usar los microservicios es la **escalabilidad**, la cual consiste en descomponer las funciones en el eje X (duplicación horizontal), eje Y (Descomposicion funcional, escalar separando diferentes funciones) y eje Z (Particionamiento de datos, escalar separando funciones similares). Por ejemplo, al hacer una aplicación de e-commerce se puede tener las aplicaciones por separado (ventas, stock, clientes, etc.) permitiendo escalar sin problemas el componente que más recursos necesite.

**Cambiar a mantenibilidad:** A medida de que la industria evoluciona, la mantenibilidad es un factor clave en sistemas de software. Así, los principios de desarrollo SOLID y DRY se alinean con los fundamentos de Microservicios. Uno debería preocuparse por el tamaño y la calidad de la aplicación.

**Cambio de mar** *(Sea change)*:

* Un tiempo maduro
* Automatización.
* Contenedores
* Dev-ops.
* Abstracciones de alto nivel (tener una mente de enfoque abierto).

**Patrones y enfoques:**

* CQRS: Segregación de la responsabilidad de un command query.
* Abastecimiento de eventos.
* Proxy, APIs.
* API Orquestada.

1. *Enfoque al éxito:*

Para convertirse en exitoso se debe entender de manera correcta el giro del negocio, la estructura de la organización, el tamaño del código, el tamaño del equipo y tener en cuenta que la tecnología debe encajar con las metas de la organización. Además una aplicación de microservicio no debería tener defectos (hacer lo que debería hacer), debe satisfacer la necesidad de los usuarios, ser seguro, escalable (soportar gran carga de usuarios), robusto (resistir a fallas o cambios impredecibles), fácil de gestionar y lanzar, susceptible al cambio, debe tener un buen presupuesto y presentarse a tiempo.

**Estructura organizacional:**

Los equipos organizacionales deben construirse con el tiempo y los servicios deben ser propios de un equipo mas no de muchos. El diseño de software copia la estructura de comunicación de la organización que lo crea. La simplicidad es la clave para ser exitoso: principios DRY y SOLID.

**¿Cuál es el tamaño ideal de un equipo?**

El equipo deberia tener armonia *(good fit),* es decir, una buena combinación entre colaboradores web, backend, moviles,etc. El tamaño depende de las personas y el stack a seguir, además se recomienda bastante confianza y no debería haber transferencia entre quipos.

**Tecnologia / stack:**

* Basados en eventos, lenguajes inmutables (Erlang, Elixir, Scala)
* Bases de datos graficas (orientdb, neo4J, etc)
* Buscar índices optimizados (Lucene, solr)
* Java, .Net, NodeJS, Go, Rust, Julian.

1. *Beneficios/desafios:*

Beneficios de microservicios:

* Performance (en ejecución, desarrollo, time to market) , a long term benefit.
* Satisface las altas expectativas del cliente a través de la arquitectura (Robusto, escalable).
* Lenguaje ubicuo (usar un lenguaje específico al desarrollar un microservicio).
* Despliegue rápido.
* Fácil de probar.
* Más barato para escalar.
* Fallo de aislamiento mejorado.

Desafíos:

* Más complejidad en sistemas distribuidos.
* Pruebas del sistema debido a la dependencia de microservicios.
* Transacciones distribuidas (Consistencia eventual).
* Administración del microservicio.
* Mayor memoria a menos que se use containers.
* Organización y cultura
* Madurez

**Aplicación**:

1. Tecnologías:

* CQRS: Significa **Command Query Responsibility Segregation.** Separa el proceso de lectura de escritura.
* Event sourcing (Consistencia eventual): Es básico para diseños de negocios que necesiten confiabilidad y transparencia (Ej.: Contabilidad, Bancos, Blockchain). Repetición incorporada, es nuevo y no convencional y agnóstico a bases de datos. Guarda estados de objetos.
* DDD (Domain driven design)/ Contexto delimitado: cada contexto tiene su propio lenguaje.
* API Gateway: Resuelve la granularidad, diferentes clientes necesitan diferente data, el número y ubicación de las instancias del servicio cambian dinamicamente. El particionamiento de servicios debe ser invisible para los clientes.
* Lenguajes para problemas de espacio (NoSQL, Inmutable, Estadístico).