**Pregunta 1 – Martin Fowler y su exposición sobre Microservicios**

Martin Fowler comienza su presentación presentándose, a qué se dedicó los últimos años y como desarrolló microsevicios sin que se aún sean llamados como tales.

A continuación, habla sobre la agenda que seguirá su charla y comienza hablando sobre **la arquitectura de microservicios.** Aquí hace referencia al libro de Christopher Alexander ‘The Timeless Way of Building’, el cual es un libro sobre arquitectura de construcción de edificaciones, en el que se basaron los patrones de diseño de software.

Al entrar al detalle de la arquitectura, Fowler divide este tema en dos: **estandarización** y **metas**.

El expositor narra que desde el comienzo de los tiempos del desarrollo de software se ha trabajado con arquitecturas monolíticas. Sin embargo, en los últimos tiempos se está haciendo una transición de las arquitecturas monolíticas a las arquitecturas por microservicios. Esto se debe a que esta arquitectura permite separar las funciones de negocio en contextos delimitados, específicamente un contexto delimitado para un servicio.

Los microservicios son de cierta forma ‘amigables’ para realizar migraciones graduales ya que, como se explicó anteriormente, se separan contextos delimitados diferentes y eso hace que toda la arquitectura no sea dependiente entre sí.

El expositor afirma que los microservicios brinda a sus usuarios escalabilidad. Estos permiten descomponer las funciones por las cosas que hacen cada una y luego estas pueden escalar, independientemente de las demás.

Luego, Martin habla sobre los **principios** que cumplen los microservicios. Estos son: Encapsulación, automatización, dominio céntrico, descentralizado, independiente, a prueba de fallos y observable.

También habla de hacer el código de cierta forma ‘mantenible’ y de priorizar eso.

El siguiente tema importante a tratar en la exposición es el de **patrones.** Se presentan los siguientes patrones:

- CQRS (Command Query Responsibility Segregation): Indica que no se deben realizar las acciones de lectura y escritura en las mismas operaciones en el mismo momento.

- Event Sourcing: Se basa en la secuencia de eventos

- API Gateway / Proxy: Es una manera de extraer algunas de las complicaciones de seguridad que se puedan presentar.

- Orchestrated API: Es una manera de micro manejo de hacer las cosas.

Luego se realiza la siguiente pregunta. **¿Qué define el éxito de los microservicios?** Fowler dice que lo siguiente explica qué define su éxito:

* Es holístico, mientras más se enfoque de esta manera, más sentido tendrá.
* Hace lo que está destinado a hacer y nada más.
* Cumple con las necesidades de los usuarios, si no lo usan es porque hay algún problema.
* Es seguro.
* Es escalable.
* Es robusto, ósea puede aceptar cambios y errores.
* Fácil de dar soporte

Seguidamente se hablan de **los pros y contras** de los microservicios, los que serán detallados a continuación.

**Pros**:

* La performance es un beneficio a largo plazo. Se puede escalar fácilmente.
* Cumplir con las expectativas de los usuarios de alto nivel en base a la arquitectura.
* Ubiquidad del lenguaje alrededor de un problema, fácil de probar y barato de escalar.
* Si se tiene un cuello de botella se puede separar ese parte de la aplicación, escalarlo si modificar otras partes.

**Contras**:

* Es más complejo en los sistemas distribuidos, por las pruebas unitarias dadas la dependencia, por lo que se necesita una buena interfaz de trabajo.
* Las transacciones distribuidas. Solución: event outsourcing.
* Administración del sistema (tooling).
* Gran cantidad de uso de memoria.
* Dependencia en la organización, su cultura y la madurez de la compañía.

Finalmente se hablan sobre los detalles de **las tecnologías** que se usan al desarrollar microservicios. Estas son las siguientes:

**CQRS:** Separar la acción de lectura de la de escritura. Sirve como punto de partida para descomponer tu código.

**Event Sourcing:** Es secuencial, si se cambia algo en el medio del código se sabrá si afecta lo demás o no.

* + Evento céntrico en el enfoque de dominio de modelos
  + Guarda eventos, no objetos de estado
  + El almacenamiento de eventos es el intermediario de la base de datos y los mensajes
  + Consigue los eventos para las entidades
  + Guarda los eventos que se asociaran a las entidades y sus dependencias

**DDD / Bounded context (Domain Driven Design):** El diseño es complicado, se debe conocer el negocio, el problema y el alcance para así aislar los límites de cada problema y que pueda ser específico.

**API Gateway:** Esto soluciona los siguientes problemas:

* + Una granularidad muy fina.
  + Las distintas necesidades de información de los distintos clientes.
  + La cantidad y ubicación de los distintos servicios que cambian dinámicamente.
  + El particionamiento de servicios el cual debería estar oculto para los clientes.

**Lenguaje para los problemas de espacio**: Permite aplicar un lenguaje en el cual sea útil para un problema en específico lo cual minimiza el código a utilizar.