**Pregunta 1: Presentación de Martin Fowler sobre micro servicios:**

Martin Fowler es un experto que ha trabajado como consultor por 10 años en el tema de micro servicios.

Existen dos formas de desarrollar aplicaciones: De manera monolítica, donde todo esta en una sola aplicación, y a través de micro servicios donde todo se divide en pequeñas APIs dedicadas a una función específica. El problema del desarrollo monolítico es que al ir creciendo la aplicación se vuelve más costosa e insostenible, y si uno quiere hacer alteraciones al sistema se tiene que alterar prácticamente todo, teniendo que posiblemente parar el servicio.

El autor sostiene que “Vienen los micro servicios” al rescate y señala que tienen las siguientes características:

* Dividen las funciones del negocio según un contexto específico
* Es posible la migración gradual y amigable, mientras que una migración abrupta puede ser tediosa y costosa, se puede aplicar las nuevas funcionalidades a través de nuevas APIs separadas.
* Es tecnológicamente agnóstica
* Como cada API es independiente, se pueden utilizar varios lenguajes en el conjunto (un lenguaje por cada API), aunque es recomendado mantener uno por uniformidad.

Los micro servicios tienen los siguientes principios:

* Encapsulación: El exponente señala que es muy importante, si no encapsulas no realizas el aislamiento y no te enfocas bien en el problema.
* Automatización: Se genera en las pruebas, para facilitarlas.
* Dominio céntrico
* Independiente
* A prueba de fallos
* Observable

Un punto principal de los micro servicios es la escalabilidad, la cual permite el crecimiento del software desde tres (3) ejes:

* Primer eje (X): Duplicación horizontal, generación de varios servicios
* Segundo eje (Y): Descomposición funcional: Se escala para dividir diferentes tareas
* Tercer eje (Z): Particionamiento de la data: Se escala por dividir cosas similares.

Los micro servicios tienen las siguientes características:

* Filosofía: Se busca hacer una cosa pero bien, dividiendo las cosas para hacer servicios enfocados en algo.
* Cambio a la mantenibilidad: El costo del hardware se está reduciendo pero el de la gente está aumentando. Necesitas que el código sea mantenible y no solo aumentar la potencia del hardware. Los principios SOLID y DRY pueden ayudar a esto.
* Cambios abismales: El tiempo es maduro, se busca la automatización y las abstracciones de alto nivel.
* Patrones y alcances: Sourcing de eventos, tiene que existir una API que sea utilizada como Gateway.
* Éxito:
  + Tienes que entender el negocio, dividiendo problemas en partes más manejables, documentando el proceso de desarrollo (procesos, casos de uso, escenarios)
  + La tecnología tiene que ir de acorde a los objetivos (Ejemplo: El lenguaje según la situación -> Se puede usar uno basado en eventos o uno con grandes librerías)
  + El equipo de diseño tiene que saber la estructura organizacional.
    - Los equipos deben ser definidos y los servicios asociados a un solo equipo
  + El diseño de software debe cumplir la ley de Conway, tiene que copiar la estructura de la organización
  + Usar simplicidad y patrones: DRY, SOLID.
  + Tiene que cumplir con lo que piden los usuarios
  + El tamaño del equipo no debe de ser muy grande, máximo de 10-12 personas y tiene que ser gente con la que puedas tener confianza.

Los micro servicios tienen los siguientes beneficios:

* Performance: A largo plazo, puede demorar en el diseño pero por la escalabilidad de ayuda a futuro.
* Cumple con expectativas del usuario a través de la arquitectura
* Lenguaje omnipresente: Puedes tener varios lenguajes pero tener uno solo ayuda más para la uniformidad.
* Desarrollo más rápido
* Más fácil de probar como un solo objeto
* Más barato para escalar
* Aislamiento de fallas mejorada

Y también tienen retos:

* Mayor complejidad en sistemas distribuidos
* Prueba de sistemas: Debido a la dependencia de diversos micro servicios (equipos independientes)
* Transacciones distribuidas
* Manejo del sistema (tooling)
* Mayor uso de memoria
* La organización
* Madurez

Aplicación de tecnologías:

* CQRS
  + Patrón interesante, separa lecturas de escrituras.
  + Como separas tu programa para beneficios. La actualización de la data no depende del resto.
* Event Sourcing: Si capturas la data correcta y guardas los eventos, puedes volverla a reproducir y sub versionas todo… tienes una gran prueba de regresión: puedes ver la consistencia de todo.
  + Es seguro
  + Guardas eventos en la base de datos, siendo la DB usada para guardar eventos
    - Consigues eventos para una entidad
    - Aplicas eventos para la entidad
    - Procesas nuevos comandos
    - Guardas eventos para asociarlo a entidades
  + Nuevo y no convencional, confiable, consistente
* DDD: Delimitación de contextos: El diseño es complicado, cada contexto tiene su propio diseño. Buscas que cada contexto sea específico.
* Gateway API: Conecta a los clientes y los redirige a la data que necesitan