**Microservicios**

La meta de los patrones aplicados es comunicar. La estandarización que se utilizara genera un alto nivel de entendimiento del programa.

Busca sustituir los sistemas monolíticos, que son grandes sistemas / aplicaciones que realizan varias cosas dentro de su composición y son difíciles de dar soporte, lo que hace que el costo para el soporte de este sea mayor del beneficio que se generaría.

La razón principal por la cual se utilizan microservicios es por la escalabilidad. Permite descomponer las funciones por las cosas que hacen diferentes y luego estas se pueden escalar independientemente de los otros.

Se alinea con los principios SOLID y DRY.

**Características**:

* Descomponer las funciones del negocio agrupadas alrededor de un contexto.
* Es amigable para la migración gradual, lo que permite trabajar con los objetivos del negocio.
* Es agnóstico a la tecnología, permite aislar los problemas del negocio, codificar en la tecnología que desees para después poder testear si funciona o no.
* Es lo que SOA debió haber sido

**Principios**:

* Encapsulación: Si uno está encapsulando bien, entonces no se está aislando verdaderamente.
* Dominio céntrico: Se enfoca en un problema en específico.
* Independiente
* Seguridad contra fallos
* Autómata
* Descentralizado
* Observable

**Patrones**:

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

Event Sourcing: Basado en la secuencia de eventos

API Gateway / Proxy: Es una forma de extraer algunas de las complicaciones de seguridad

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

Que define su éxito:

* El éxito no es solo acerca de la tecnología, es un holístico, mientras más se enfoque de esta manera, más sentido tendrá.
* Que no tenga defectos, que haga lo que está destinado a hacer.
* Que cumpla con las necesidades de los usuarios, si no lo usan es porque hay algún problema.
* Seguro
* Escalable
* Robusto, que puede aceptar cambios y errores
* Fácil de dar soporte

**Pros**:

* Performance, al largo plazo es un beneficio. Se puede escalar fácilmente.
* Cumplir con las expectativas de los usuarios de alto nivel en base a la arquitectura (un software esta perpetuamente en beta por las modificaciones)
* 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, lo cual se soluciona con el event sourcing.
* Administración del sistema (tooling)
* Gran cantidad de uso de memoria
* Dependencia en la organización, su cultura y la madurez de la compañía.

**Tecnologías**:

1. CQRS: Separar la acción de lectura de la de escritura. Sirve como punto de partida para descomponer tu código.
2. Event Sourcing: Es secuencial, si se cambia algo en el medio del código se sabrá si afecta lo demás o no.
   1. Evento céntrico en el enfoque de dominio de modelos
   2. Guarda eventos no objetos de estado
   3. El almacenamiento de eventos es el intermediario de la base de datos y los mensajes
   4. Consigue los eventos para las entidades
   5. Guarda los eventos que se asociaran a las entidades y sus dependencias
3. 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.
4. API Gateway: Esto soluciona los siguientes problemas:
   1. Una granularidad muy fina
   2. Las distintas necesidades de información de los distintos clientes
   3. La cantidad y ubicación de los distintos servicios que cambian dinámicamente.
   4. El particionamiento de servicios el cual debería estar oculto para los clientes
5. 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.