Arquitectura de Software
Conceptos iniciales

Arquitectura de Software

[SA] ARQUITECTURA DE SOFTWARE

Elementos de tecnología que se relacionan entre sí para que los productos, los sistemas de información y los servicios de tecnología funcionen de la mejor forma posible para que el usuario final los use.

- Se ocupa de las interfaces (que separan lo privado de lo público), centrándose en la comunicación e interacción entre esos elementos.
- Se abstrae de los detalles internos y de cómo funcionan internamente esos elementos

Interesados en la SA

- Usuario Final → quien usará el sistema de información o producto SW.
    - Interesado en que los sistemas/productos funcionen de la mejor manera posible (se consideran criterios como rapidez, disponibilidad y confiabilidad) para que justamente el usuario final los quiera usar.
- Cliente → quien nos contrató, no necesariamente es el usuario final.
    - Interesado en que se implemente una arquitectura que respete calendario y presupuesto previamente seleccionados.
- Project Manager (PM) → quien define cómo se lleva adelante un proyecto.
    - Interesado en que los equipos trabajen en forma independiente interactuando con disciplina.
- Arquitecto → encargado de comunicar qué arquitectura se implementará.
    - Interactúa con el usuario final, el cliente y el PM.
    - Interesado en dejar conforme a los tres protagonistas mencionados.
- Administradores de Bases de Datos (DBA).
- Desarrolladores.
- *Testers*.
- Gente de seguridad.

Elección de la SA – Decisiones de Diseño:

- ¿Procesamiento distribuido o no?
- ¿SW dividido en capas? ¿Cuántas? ¿Qué patrones se usan?
- ¿Comunicación sincrónica o asincrónica?
- ¿Dependemos del sistema operativo que tenemos?
- ¿Dependemos del HW que tenemos?

Elección de la SA – Contextos y/o Aspectos:

- Técnico → conocimiento.
- Negocio → imposiciones de reglas de negocio que tiene la organización (como por ejemplo los convenios para usar ciertas tecnologías).
- Ciclo de Vida del proyecto → ¿encaja el ciclo de vida de un producto con la SA deseada?
- Profesional → ¿encajan las características del equipo de trabajo con la SA deseada?

Atributos de Calidad

ATRIBUTOS DE CALIDAD

Propiedades o medidas de testeo que permiten indicar qué tan bien funciona un sistema y cómo satisfacer las necesidades de los interesados.

- Atributos de Calidad referidos a los requerimientos:
  - Requerimientos Funcionales → qué hará el sistema de información o producto SW.
  - Requerimientos NO Funcionales → caracterizar las funcionalidades.
  - Restricciones de Negocio → reglas de negocio y convenios que tiene la organización.
- Disponibilidad → minimizar las interrupciones del servicio y mitigar posibles fallas.
  - Tácticas: detección, recuperación y prevención de fallas.
- Interoperabilidad → capacidad que tienen dos elementos dentro de la SA para poder relacionarse entre sí vía interfaces.
- Adaptabilidad → capacidad de no sentir resultados de cambios (de plataforma, de sistema operativo, etc.), costos o riesgos.
- Variabilidad → adaptación al contexto.
- *Performance* → referido al tiempo y a la habilidad.
- Seguridad → referido a la detección de ataques y a cómo resistirlos.
  - Detección de intrusos, denegación de servicios, verificación de integridad, autenticación de actores, límites de acceso, encriptación de datos, etc.
- Usabilidad → cuán fácil es para el usuario ejecutar una tarea deseada.
- *Testeabilidad* → buena parte del costo de una buena ingeniería en el desarrollo de los sistemas es absorbida por las pruebas.
- Portabilidad → capacidad para adaptarse a los cambios de plataforma.
- Desarrollo Distribuido → diseño del SW.
- Escalabilidad → capacidad de agregar más recursos.
- Monitoreo → controlar e investigar el sistema mientras trabaja.
- Comerciabilidad → si lo que terminamos construyendo se adaptó a lo que quería el negocio.

Patrones de Arquitectura

PATRONES DE ARQUITECTURA

Conjunto de soluciones de diseño que se dan bajo contextos y problemas similares.

Requerimientos de Arquitectura Significativos (ASR)

- Revisar documentación previa sobre qué arquitecturas se implementaron.
- Entrevistar a los interesados en el negocio por qué se eligieron esas arquitecturas.
- Interactuar con el usuario final para entender los objetivos del negocio

Características de la SA en Proyectos Ágiles

Existe una combinación de arquitecturas que se basan en proyectos ágiles y arquitecturas de paradigmas estructurados, no siempre hay que caer en lo que ofrece el mercado como solución.

No obstante, algunas características son:

- División del proyecto en intervalos de tiempo relativamente cortos (sprints).
- Entregas de SW entre semanas y meses.
- Interacción continua y fluidez de la comunicación entre el equipo de trabajo y el cliente.
- Alta satisfacción del cliente cuando se entrega una versión.

Gestión y Gobierno de la SA

- **Evaluación:**
    - El arquitecto debe interesarse por la gestión de proyectos.
    - PM y arquitecto deben trabajar en conjunto por la perspectiva de la organización.
    - A mayor complejidad de proyectos, más útil es la implementación de una arquitectura.
- **Planificación:**
    - Si bien la planificación sucede constantemente, existe un plan inicial para convencer a la dirección de construir el sistema y dar una idea de costo y agenda.
    - El PM debe educar a otros managers para que puedan corregir desvíos en el desarrollo del SW.
- **Organización:**
    - *Team Leader* → gestiona las tareas del equipo.
    - *Developer* → diseñan e implementan los subsistemas de código.
    - *Configuration Manager* → ejecutan y construyen *tests* de integración.
    - *System Test Manager* → testeo de sistema y *testing* de aceptación.
    - *Product Manage* → representan el marketing.

Arquitecturas Monolíticas

- La aplicación se construye como una unidad, un *todo*, una sola pieza, un monolito.
- Todos los módulos ejecutan dentro de un mismo proceso y sobre un mismo HW.
- Forma típica que adquieren:
    - Interfaz de Usuario del lado del cliente → formada por páginas HTML y código JavaScript que se ejecutan en el navegador web del cliente.
    - Base de Datos → generalmente es una BD relacional, pero puede ser otra.
    - Aplicación del lado del servidor → un único monolito que recibe solicitudes HTTP del cliente, ejecuta una lógica de negocio, recupera y actualiza datos de la BD y arma las vistas HTML que serán enviadas al navegador web.
- Los cambios en la aplicación están muy acoplados:
    - Todo cambio implica construir y desplegar una nueva versión de toda la aplicación.
    - Todo escalado requiere que escale toda la aplicación
    - Todo cambio condiciona la frecuencia de las entregas.
- Si bien pueden ser muy exitosas estas arquitecturas, suelen generar cierta frustración en los responsables, sobre todo a medida que la aplicación crece.

Ventajas

- Fáciles de desarrollar.
- Fáciles de testear → no significa que sean fáciles de corregir.
- Fáciles de desplegar → no significa que las tareas previas al despliegue sean sencillas, porque el armado de dichas tareas puede llevar bastante tiempo.
- Fáciles de escalar horizontalmente, replicando en distintos servidores.

Desventajas

- A medida que la aplicación crece:
    - Aumenta la complejidad del monolito → sigue siendo una sola pieza.
    - Aumenta la dificultad de mantener la modularidad inicial.
    - Aumenta la dificultad de incorporar nuevas tecnologías → es muy difícil cambiar la tecnología de un sector sin cambiar todo.
    - Disminuye la fiabilidad → un error en cualquier módulo puede potencialmente afectar toda la aplicación.
- El escalado es completo → se replica toda la aplicación.

Arquitectura Monolítica vs Arquitectura de Microservicios (MSA)

Arquitectura Monolítica	Arquitectura de Microservicios (MSA)
Pone toda la funcionalidad en un solo proceso.	Pone cada funcionalidad en un servicio separado.
Escala replicando el monolito en múltiples servidores.	Escala distribuyendo estos servicios entre servidores, replicando según sea necesario.

Arquitectura de Microservicios (MSA)

MICROSERVICIOS (MSA)

Estilo de arquitectura en el que una aplicación se desarrolla como un conjunto de pequeños servicios que:

- **Ejecutan en su propio proceso y se comunican con otros microservicios vía mecanismos ligeros.**
- **Se construyen en torno a capacidades de negocio.**
- **Pueden desplegarse de forma independiente mediante procesos automatizados.**
- **Poseen una mínima gestión centralizada.**
- **Pueden escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.**

Ventajas de MSA

- Facilita el despliegue continuo de aplicaciones grandes y complejas → la aplicación grande y compleja en realidad son aplicaciones chicas combinadas entre sí.
- Facilita el mantenimiento.
- Facilita la incorporación de nuevas tecnologías.
- Permite el despliegue y escalado independiente.
- Permite trabajar con equipos de desarrollo autónomos.

Desventajas de MSA

- Aumento significativo de la complejidad propia de un sistema distribuido.
- Requiere implementar mecanismos de comunicación entre servicios y el manejo de fallos.
- El *testing* de interacción es, en general, más complejo.
- Aumenta la complejidad de implementación, gestión y monitoreo.
- Se dificulta la detección de errores en tiempo de ejecución.
- Complejidad de la arquitectura de persistencia de datos particionada → son muy comunes las transacciones de negocio que requieren actualizaciones en repositorios pertenecientes a múltiples servicios.

¿Cuándo usar una MSA?

- Cuando el aprovisionamiento de infraestructura es rápido y, en lo posible, automatizado.
- Cuando hay herramientas adecuadas para el monitoreo correcto.
- Cuando hay mecanismos que permitan un despliegue rápido.

¿Cuándo NO usar una MSA?

- Si la aplicación es pequeña y no se espera que escale, **MSA** puede resultar una solución complicada y cara → una arquitectura monolítica es una solución mejor.

Características principales de MSA

- **Componentización a través de servicios** → las MSA usan bibliotecas, pero su forma principal de crear componentes de su propio SW es segmentarlos en servicios.
    - Componente → unidad de SW independiente, se puede reemplazar y actualizar.
    - Bibliotecas → componentes que corren dentro del mismo proceso → están vinculadas a un determinado SW que se invocan por medio de llamadas a funciones en memoria.
    - Servicios → componentes que corren fuera del proceso.
        - Se comunican vía peticiones a *web services* o RPC, por ejemplo.
        - Que el servicio corra en un proceso aparte lo dota de independencia → si se cae el servicio, no se cae todo el monolito → no tiene un único punto de falla.

- **Organizada en torno a funcionalidades de negocio** → el hecho de partir las funcionalidades en servicios puede generar dudas.
    - Los cambios simples pueden llevar a que un proyecto entre equipos tome tiempo y requiera aprobación presupuestaria.
    - Los equipos suelen intentar optimizar su rendimiento introduciendo lógica en la parte sobre la que tienen control directo.

- **Productos, no proyectos**:
    - Enfoque de Proyecto:
        - Tiene un objetivo determinado.
        - Se desarrollará en un marco temporal (tendrá fechas de inicio y fin planificadas).
        - El objetivo es entregar una pieza de SW que se considerará terminada en algún momento.
        - Al finalizar, el SW se asignará a una organización de mantenimiento y el equipo que lo creó se disuelve.
    - Enfoque de Producto → *somos los padres de la criatura, debemos hacernos cargo.*
        - Un equipo debe poseer un producto durante toda su vida útil.
        - Si nosotros lo construimos, entonces nosotros nos encargamos de él.
        - El equipo de desarrollo tiene un conocimiento profundo del producto → hay una relación más intensa con el negocio, que en general da mejores soluciones.
        - Se concibe al SW en una relación continua en la que se busca cómo este puede ayudar a sus usuarios a mejorar sus capacidades en el negocio.
        - Mientras la solución está disponible, no termina: termina cuando la retiremos.

- ***Smart endpoints and dumb pipes***
    - La lógica de la aplicación está en los servicios y no en los mecanismos de comunicación.
    - Los mensajes son coreografiados utilizando protocolos simples.

- **Gobierno descentralizado**
    - Los microservicios no están obligados a ninguna estandarización de plataformas.
    - Cada microservicio elige cuál es la tecnología que mejor se adapta a sus necesidades.

- **Gestión de Datos descentralizada**
    - Descentralización del modelo conceptual → se segmenta un dominio complejo en múltiples dominios delimitados y mapea las relaciones entre ellos.
    - Descentralización de las decisiones de persistencia de datos → cada servicio administra su propia DB.
    - **Teorema de CAP** → en un sistema distribuido, donde las partes deben comunicarse entre sí, existen 3 características cuyo cumplimiento no se puede asegurar en forma permanente: puede ser que las 3 características se cumplan la mayor parte del tiempo, pero *no siempre*; *siempre* solamente 2:
        - [C] Consistencia → si distintos usuarios se conectan a distintas partes del sistema, todos deben obtener la misma lectura.
        - [A] Disponibilidad → si un usuario se conecta al sistema, siempre obtiene una respuesta, aunque no se garantiza que esa respuesta esté actualizada.
        - [P] Tolerancia al Particionamiento → el sistema sigue funcionando incluso si ha sufrido particiones (por lo general, producto de una catástrofe).

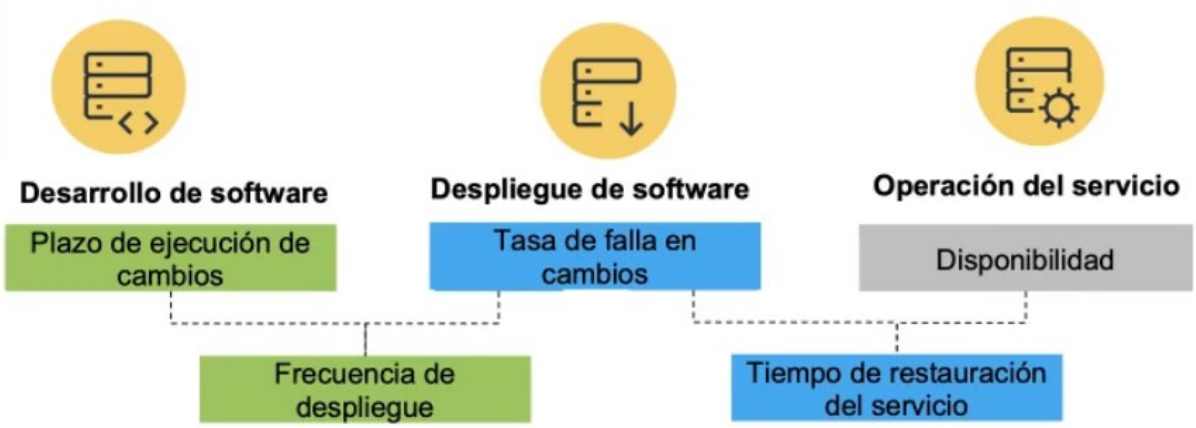
      Una buena y muy utilizada estrategia, aprovechando que las particiones son muy poco frecuentes, es asegurarse la consistencia y la disponibilidad permanentes. Respecto de la tolerancia al particionamiento, podemos prepararnos para, si sucede lo inevitable (una catástrofe que *particione* al sistema), recuperarnos rápidamente:
        - Pueden generarse particiones, ya que son inevitables.
        - Se tienen mecanismos para detectar tempranamente las particiones y, una vez recuperada la conectividad, se trabaja para corregir rápidamente las consecuencias de las particiones (un *merge* entre las dos particiones).

- **Automatización de infraestructura**.

- **Diseño tolerante a fallas** → se preparan soluciones que contemplan escenarios de fallas para lidiar con ellas y mantenerse operativo.
    - Las comunicaciones sobre redes son, por naturaleza, poco confiables.
    - Las aplicaciones se diseñan para ser resilientes y manejar errores, no solamente para prevenirlos.
    - Monitoreo en tiempo real de la aplicación, tanto de elementos de arquitectura como de métricas relevantes del negocio.

- **Diseño evolutivo**:
    - Los servicios evolucionan buscando reducir al mínimo el impacto de los cambios en sus consumidores.
    - Los servicios se diseñan buscando el mínimo acoplamiento posible al contrato de sus proveedores.
    - El uso de servicios como componentes posibilita planeamientos de despliegues más granulares.

Métricas de MSA



- Plazo de Ejecución de Cambios → mide el tiempo que va desde que empieza el análisis de los requerimientos hasta que todo se despliega y quede disponible para su uso.
- Tasa de Falla en Cambios → mide cuántos despliegues son exitosos respecto del total de despliegues realizados.
- Disponibilidad → mide qué tan disponible está un servicio.
- Frecuencia de Despliegue → lo ideal es que se las entregas se realicen muy seguido (varias veces por día), con una muy baja tasa de errores.
    - Es la combinación entre el plazo de ejecución de cambios y la tasa de falla en cambios.
- Tiempo de Restauración del Servicio → mide el tiempo que se tarda en recuperar el servicio, en volverlo a disponibilizar.
    - Es la combinación entre la tasa de falla en cambios y la disponibilidad.

Aspectos de la *performance* de la entrega de software:

	ÉLITE	ALTO	MEDIO	BAJO
Frecuencia de Despliegue	Bajo Demanda. Varios despliegues por día.	1 vez por día ~ 1 vez por semana	1 vez por semana ~ 1 vez por mes	1 vez por mes ~ 1 vez por semestre
Plazo de Ejecución de Cambios	Menos de 1 día	1 día ~ 1 semana	1 semana ~ 1 mes	1 mes ~ 6 meses
Tiempo de Restauración de Servicio	Menos de 1 hora	Menos de 1 día	Menos de 1 día	1 semana ~ 1 mes
Tasa de Falla en Cambios	0% ~ 15%	0% ~ 15%	0% ~ 15%	46% ~ 60%