

“**TIPOS DE ARQUITECTURA DE SOFTWARE**”

INGENIERIA DE SOFTWARE

**INGENIERÍA EN SISTEMAS COMPUTACIONALES**

PRESENTA:

**ISIDRO ARREDONDO CAMARENA**

**EDGAR CORTÉS RESÉNDIZ**

**JESÚS EDUARDO CÁRDENAS ÁVILA**

**Luis Antonio Diaz Rosales**

JIQUILPAN, MICHOACÁN, MARZO DE 2025

**1. Arquitectura Monolítica**

**Descripción:**  
Imagina que la arquitectura monolítica es como una casa donde todo está en un solo lugar: la cocina, el baño, la sala y los dormitorios están todos juntos en un solo edificio. En el mundo del software, esto significa que toda la aplicación (la interfaz, la lógica y la base de datos) está empaquetada en un solo programa. Es como tener una app que hace todo en un solo bloque.

**Ventajas:**

* **Es fácil de entender:** Como todo está en un solo lugar, no tienes que preocuparte por cómo se conectan las partes. Es como tener todo en una sola caja.
* **Rápido de desarrollar:** Al principio, es más sencillo construir algo pequeño y compacto.
* **Fácil de desplegar:** Solo tienes que subir un solo archivo o programa al servidor, como subir una foto a Instagram.

**Desventajas:**

* **Si algo falla, todo falla:** Imagina que se va la luz en tu casa. Como todo está en un solo lugar, no puedes usar nada. Lo mismo pasa con una app monolítica: si hay un error, toda la aplicación puede dejar de funcionar.
* **No escala bien:** Si tu app se vuelve muy popular y necesita crecer, es difícil hacer que solo una parte funcione mejor. Tienes que escalar todo el sistema, lo que puede ser caro y complicado.
* **Mantenimiento difícil:** Con el tiempo, el código puede volverse un lío, como un armario lleno de cosas sin orden. Cualquier cambio pequeño puede afectar todo el sistema.

**Ejemplo:**  
Un ejemplo clásico es WordPress. Es como una casa donde todo (el diseño, las entradas del blog, los plugins) está en un solo lugar. Es fácil de usar al principio, pero si quieres hacer cambios grandes, puede volverse complicado.

**2. Arquitectura en Capas**

**Descripción:**  
Ahora imagina que la arquitectura en capas es como un edificio con varios pisos. Cada piso tiene una función específica: el primer piso es la recepción (la interfaz de usuario), el segundo piso es donde se hacen las decisiones (la lógica de negocio), y el tercer piso es el almacén (la base de datos). Cada piso hace su trabajo y se comunica con los demás, pero no se mezclan.

**Ventajas:**

* **Organización:** Cada capa tiene su propio trabajo, lo que hace que sea más fácil entender y mantener el sistema. Es como tener un edificio bien organizado.
* **Reutilización:** Puedes usar la misma capa en diferentes proyectos. Por ejemplo, la capa de base de datos puede usarse en varias aplicaciones.
* **Facilidad de prueba:** Puedes probar cada capa por separado, como revisar cada piso del edificio uno por uno.

**Desventajas:**

* **Puede ser lento:** Como las capas tienen que comunicarse entre sí, puede haber un poco de retraso. Es como si tuvieras que subir y bajar escaleras para hacer algo.
* **Complejidad:** Si hay muchas capas, puede volverse complicado manejar todas las conexiones entre ellas.
* **Dependencia:** Aunque están separadas, las capas dependen unas de otras. Si una falla, puede afectar a las demás.

**Ejemplo:**  
Un sistema bancario es un buen ejemplo. La interfaz de usuario (el cajero automático o la app móvil) es una capa, la lógica de negocio (cómo se procesan las transacciones) es otra, y la base de datos (donde se guarda la información de las cuentas) es la última. Cada una hace su parte, pero trabajan juntas.

**3. Arquitectura de Microservicios**

**Descripción:**  
Piensa en la arquitectura de microservicios como un equipo de personas, donde cada uno tiene un trabajo específico. Por ejemplo, uno se encarga de los pagos, otro de las notificaciones, y otro de la base de datos. Cada uno trabaja de manera independiente, pero todos colaboran para que el sistema funcione. En el software, cada "persona" es un servicio pequeño que hace una tarea específica.

**Ventajas:**

* **Escalabilidad:** Si un servicio necesita más recursos (como el de pagos en temporada de rebajas), puedes escalar solo ese servicio sin tocar los demás. Es como contratar más cajeros en un supermercado solo para las cajas, sin necesidad de ampliar toda la tienda.
* **Flexibilidad:** Cada servicio puede estar escrito en un lenguaje diferente o usar tecnologías distintas. Es como tener un equipo donde cada uno usa la herramienta que mejor le funciona.
* **Resiliencia:** Si un servicio falla, los demás pueden seguir funcionando. Es como si en un equipo, si una persona se enferma, los demás pueden seguir trabajando.

**Desventajas:**

* **Complejidad:** Gestionar muchos servicios puede ser como coordinar un equipo grande. Necesitas herramientas para monitorear y asegurarte de que todo funcione bien.
* **Latencia:** Como los servicios tienen que comunicarse entre sí, puede haber un poco de retraso. Es como si en un equipo, las personas tuvieran que mandarse mensajes constantemente.
* **Consistencia de datos:** Mantener la información sincronizada entre servicios puede ser un desafío. Es como si cada persona del equipo tuviera su propia libreta de apuntes y tuvieras que asegurarte de que todos tengan la misma información.

**Ejemplo:**  
Netflix es un ejemplo clásico. Tienen un servicio para recomendaciones, otro para reproducir videos, otro para gestionar usuarios, etc. Cada uno funciona de manera independiente, pero juntos hacen que la plataforma funcione sin problemas.

**4. Arquitectura Orientada a Servicios (SOA)**

**Descripción:**  
La arquitectura orientada a servicios (SOA) es como un mercado donde diferentes vendedores ofrecen servicios específicos. Por ejemplo, uno vende frutas, otro verduras, y otro carne. En el software, cada "vendedor" es un servicio que hace algo específico, como procesar pagos o gestionar usuarios. Estos servicios pueden ser usados por diferentes aplicaciones, como si varios clientes compraran en el mismo mercado.

**Ventajas:**

* **Reutilización:** Los servicios pueden ser usados por diferentes aplicaciones. Es como si varios restaurantes compraran frutas en el mismo mercado.
* **Interoperabilidad:** Los servicios pueden comunicarse entre sí, incluso si están hechos con tecnologías diferentes. Es como si los vendedores del mercado hablaran diferentes idiomas pero aún pudieran hacer negocios.
* **Escalabilidad:** Si un servicio necesita más recursos, puedes escalarlo sin afectar a los demás. Es como si un vendedor contratara más ayudantes para atender a más clientes.

**Desventajas:**

* **Complejidad:** Gestionar muchos servicios puede ser complicado, como coordinar un mercado grande con muchos vendedores.
* **Rendimiento:** La comunicación entre servicios puede ser lenta, como si los vendedores tuvieran que mandarse mensajes constantemente.
* **Gobernanza:** Necesitas reglas claras para gestionar los servicios, como horarios y precios en un mercado.

**Ejemplo:**  
Un sistema de reservas de hoteles es un buen ejemplo. Tienes un servicio para buscar habitaciones, otro para hacer reservas, y otro para procesar pagos. Estos servicios pueden ser usados por diferentes aplicaciones, como una app móvil o una página web.

**5. Arquitectura Basada en Eventos**

**Descripción:**

Imagina un baile donde cada movimiento es un evento. Los componentes del sistema son como bailarines que reaccionan a la música (los eventos) de manera independiente, pero en perfecta sincronía. No necesitan hablar entre sí, solo seguir el ritmo.

**Ventajas:**

* **Escalabilidad:** Cada bailarín puede moverse a su propio ritmo, sin afectar a los demás.
* **Flexibilidad:** Si llega un nuevo bailarín, se integra fácilmente al baile.
* **Desacoplamiento:** No necesitan conocerse, solo seguir la música.

**Desventajas:**

* **Complejidad:** A veces, es difícil saber quién inició el movimiento o por qué.
* **Latencia:** Algunos pasos pueden tardar un poco en ejecutarse.
* **Consistencia:** Mantener a todos en el mismo compás puede ser un desafío.

**Ejemplo:**

Piensa en sistemas como Kafka o RabbitMQ, donde los eventos son como notas musicales que fluyen y los componentes bailan al ritmo de ellas.

**6. Arquitectura Cliente-Servidor**

**Descripción:**

Es como un restaurante. Tienes al cliente (tú) que hace una solicitud (pedir un plato) y al servidor (el chef) que prepara y entrega lo que has pedido. Todo gira en torno a esa relación de solicitud y respuesta.

**Ventajas:**

* **Centralización:** El chef tiene todo bajo control en la cocina.
* **Escalabilidad:** Puedes invitar a más amigos (clientes) sin que el chef se sienta abrumado.
* **Seguridad:** Solo el chef decide qué platos sirve y a quién.

**Desventajas:**

* **Dependencia:** Si el chef se enferma, nadie come.
* **Cuellos de botella:** Si todos piden al mismo tiempo, el chef puede estresarse.
* **Costo:** Mantener una cocina profesional no es barato.

**Ejemplo:**

Cuando usas Google, tu navegador es el cliente que pide información, y los servidores de Google son el chef que te sirve lo que necesitas.

**7. Arquitectura Peer-to-Peer (P2P)**

**Descripción:**

Es como una reunión de amigos donde todos comparten algo. No hay un anfitrión central; todos son iguales y colaboran entre sí. Si alguien tiene un archivo o un recurso, lo comparte directamente con los demás.

**Ventajas:**

* **Descentralización:** No hay un solo punto de fallo; si alguien se va, la reunión sigue.
* **Escalabilidad:** Cuantos más amigos lleguen, más se puede compartir.
* **Reducción de costos:** No necesitas un anfitrión que lo controle todo.

**Desventajas:**

* **Seguridad:** Es más difícil asegurar que todos jueguen limpio.
* **Complejidad:** Coordinar a tantos amigos puede ser un lío.
* **Consistencia:** A veces, no todos tienen la misma información al mismo tiempo.

**Ejemplo:**

Redes como BitTorrent, donde los usuarios comparten archivos directamente, sin depender de un servidor central.

**8. Arquitectura de Nube**

**Descripción:**

Es como tener un jardín mágico que crece según lo que necesitas. Si necesitas más flores (recursos), aparecen al instante. Si no, se reducen. Todo está en un lugar al que puedes acceder desde cualquier parte, siempre que tengas conexión a Internet.

**Ventajas:**

* **Escalabilidad**: Tu jardín crece o se reduce según tus necesidades.
* **Costo-eficiencia:** Solo pagas por las flores que usas.
* **Flexibilidad:** Puedes cuidar tu jardín desde cualquier lugar.
* **Alta disponibilidad:** Tu jardín siempre está floreciente, sin importar qué.

**Desventajas:**

* **Dependencia:** Estás confiando en el mago que cuida el jardín (el proveedor de nube).
* **Seguridad:** A veces, otros pueden intentar entrar en tu jardín sin permiso.
* **Latencia:** Si el mago vive lejos, puede tardar un poco en responder.

**Ejemplo:**

Servicios como Amazon Web Services (AWS) o Google Cloud Platform, que te ofrecen un jardín mágico de recursos informáticos.