UD2. Introducción a las arquitecturas web y servidores

2. Tipologías de arquitecturas web

Las aplicaciones web pueden organizarse de distintas maneras según su complejidad, sus necesidades de escalabilidad, el número de usuarios, los requisitos de seguridad, etc. Estas distintas formas de organización reciben el nombre de **tipologías de arquitectura**. A continuación, analizaremos las más comunes: monolítica, cliente-servidor, multicapa, y cloud o en la nube.

2.1 Arquitectura monolítica

La arquitectura **monolítica** es la más simple y tradicional. En este modelo, todos los componentes de la aplicación están agrupados en un solo bloque funcional: la lógica de negocio, la interfaz de usuario y el acceso a datos se ejecutan juntos, normalmente en un único servidor.

Ejemplo típico: una aplicación PHP alojada en un servidor Apache, donde el mismo servidor se encarga de recibir peticiones, ejecutar la lógica del programa y acceder a la base de datos.

Ventajas:

- Más fácil de desarrollar al principio.
- Menor complejidad para entornos pequeños.
- Todo el sistema puede desplegarse de una sola vez.

Desventajas:

- Difícil de escalar cuando crece la aplicación.
- Cualquier cambio puede afectar al resto del sistema.
- No favorece la reutilización ni el trabajo en equipo.

Uso actual: todavía es muy común en pequeñas webs y en sistemas legacy (antiguos), pero se tiende a evitar en desarrollos modernos de cierta envergadura.

2.2 Arquitectura cliente-servidor

La arquitectura **cliente-servidor** es un paso más allá respecto a la monolítica. En ella, el cliente (generalmente un navegador web) realiza peticiones al servidor, que se encarga de

procesarlas y devolver la respuesta. Aquí hay una separación clara entre la parte visible para el usuario (cliente) y la parte que realiza el trabajo (servidor).

Ejemplo típico: una aplicación donde el cliente envía una solicitud HTTP para obtener datos, y el servidor responde con una página HTML o un resultado en formato JSON.

Ventajas:

- Se separan responsabilidades entre cliente y servidor.
- El cliente puede ser ligero (por ejemplo, una SPA (Single Page Application) con JavaScript).
- Facilita el mantenimiento y la escalabilidad moderada.

Desventajas:

- El servidor central sigue siendo un punto de fallo.
- La escalabilidad horizontal es limitada. Esto quiere decir que, para atender a más usuarios, a menudo es necesario mejorar el hardware del servidor en lugar de añadir más servidores.

Uso actual: sigue siendo la base de muchas aplicaciones web actuales, aunque a menudo se combina con modelos más avanzados (por ejemplo, API RESTful en el servidor, cliente en React o Vue.js).

2.3 Arquitectura multicapa

La arquitectura **multicapa** (o de n capas) va un paso más allá y divide la aplicación en distintas capas, cada una con su responsabilidad específica. Aunque puede haber más, las tres capas más habituales son:

1. **Capa de presentación**: la interfaz gráfica que ve el usuario (HTML, CSS, JS).
2. **Capa de lógica de negocio**: el código que toma decisiones y ejecuta reglas de negocio.
3. **Capa de datos**: la base de datos o sistemas de almacenamiento.

Estas capas pueden estar en un mismo servidor o distribuidas en máquinas diferentes.

Ventajas:

- Favorece la modularidad y el mantenimiento.
- Permite que distintos equipos trabajen en paralelo.

- Escalable horizontalmente con más facilidad.

Desventajas:

- Aumenta la complejidad inicial.
- Requiere más recursos de infraestructura.

Uso actual: es el modelo de referencia para aplicaciones empresariales y profesionales. Permite adoptar buenas prácticas de diseño y facilita el uso de frameworks modernos y arquitecturas más avanzadas como microservicios.

2.4 Arquitectura cloud (en la nube)

La arquitectura **cloud** está pensada para desplegar aplicaciones directamente en plataformas en la nube como Amazon Web Services (AWS), Google Cloud Platform (GCP) o Microsoft Azure. Aquí, la infraestructura física desaparece de la vista del desarrollador, que trabaja con servicios virtuales escalables, gestionados y flexibles.

Modelos comunes en la nube:

- **IaaS** (Infraestructura como servicio): se alquilan máquinas virtuales (como si fueran servidores físicos).
- **PaaS** (Plataforma como servicio): se despliegan aplicaciones sin preocuparse del sistema operativo o la configuración del servidor.
- **SaaS** (Software como servicio): el usuario final accede al software ya desplegado (ej. Gmail, Dropbox).

Ventajas:

- Escalabilidad casi infinita.
- Pago por uso: se ajusta a las necesidades reales.
- Alta disponibilidad y recuperación ante desastres.
- Infraestructura gestionada por terceros.

Desventajas:

- Dependencia del proveedor.
- Puede tener costes ocultos si no se planifica bien.
- Requiere aprender nuevos entornos y herramientas.

Uso actual: cada vez más extendido. La mayoría de empresas modernas apuestan por arquitecturas híbridas o completamente en la nube, utilizando contenedores, funciones serverless y bases de datos distribuidas.

2.5 Comparativa entre modelos

Modelo	Complejidad	Escalabilidad	Mantenimiento	Uso típico
Monolítico	Baja	Limitada	Difícil en grandes sistemas	Webs pequeñas, MVPs
Cliente-servidor	Media	Moderada	Mejor que el monolítico	Apps tradicionales
Multicapa	Alta	Buena	Modular y mantenible	Empresas, apps escalables
Cloud	Muy alta	Excelente	Alta (si se planifica bien)	Sistemas distribuidos modernos

En el siguiente apartado abordaremos los **componentes fundamentales de una arquitectura web**, entendiendo cómo se conectan y qué papel juega cada uno en el procesamiento de una petición real.