Actividad 2: Modelos de Arquitectura de Software

1. Arquitectura Monolítica

Es un modelo en el que toda la aplicación está construida **como un bloque único e indivisible**. Todas sus funciones, como la interfaz de usuario, la lógica de negocio y la base de datos, están altamente **integradas en una misma estructura**.

1.1. ¿Cómo funciona?

1. **Flujo de información interno:**
   - Todos los módulos de la aplicación están integrados dentro de un mismo ejecutable o servicio.
   - La comunicación entre los módulos se realiza mediante llamadas directas a funciones o métodos dentro del mismo proceso.
   - No hay necesidad de mecanismos de comunicación externos (APIs, colas de mensajes, etc.), ya que los módulos comparten memoria y recursos.
2. **Interacción con los usuarios:**
   - Un usuario interactúa con la aplicación a través de una interfaz gráfica o una API.
   - La solicitud del usuario es procesada por la lógica de negocio dentro del mismo sistema.
   - Si se requiere acceder a datos, el sistema consulta directamente la base de datos y devuelve una respuesta al usuario.

1.2. Evaluación de la Arquitectura

La arquitectura monolítica ofrece una **comunicación interna eficiente y rápida**, ya que todos los módulos comparten recursos dentro de un mismo proceso. Su simplicidad en el desarrollo y despliegue la hace ideal para proyectos

pequeños y medianos, donde la rapidez de implementación es prioritaria.

Además, su f**acilidad para depurar y probar en entornos controlados** permite acelerar el desarrollo sin necesidad de gestionar múltiples servicios. Sin embargo, presenta serias dificultades en sistemas grandes, ya que cualquier cambio en una parte del código puede afectar a toda la aplicación, **complicando su mantenimiento**. La **escalabilidad es otro reto**, ya que al depender de una única instancia, los cuellos de botella pueden volverse un problema crítico. Esta arquitectura es **ideal para software** empresarial interno con pocos usuarios concurrentes y aplicaciones **donde la simplicidad sea más importante que la escalabilidad.**

Las tecnologías más comunes en arquitectura monolítica incluyen **Spring Boot (Java)** y **Django (Python)**, ya que están diseñadas para manejar proyectos en una sola base de código. **Ruby on Rails** es otra opción popular para desarrollos rápidos y cohesivos. En cuanto a bases de datos, PostgreSQL y MySQL son las más utilizadas debido a su integración directa con estos frameworks.

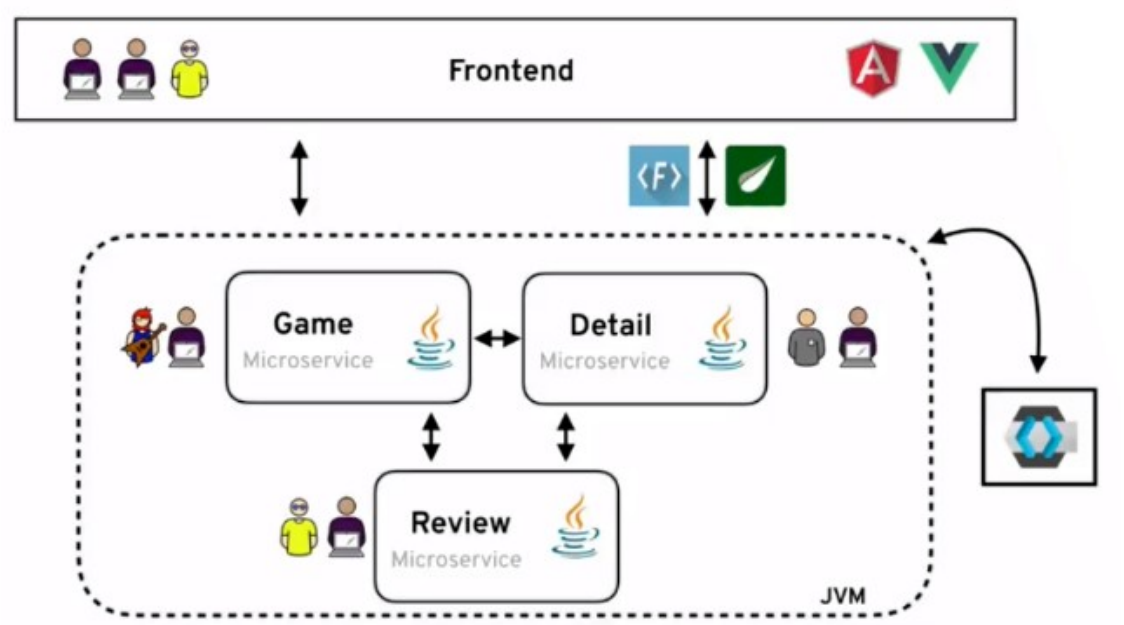

Figura 1: Arquitectura monolítica

2. Arquitectura Cliente-Servidor

Se basa en la comunicación entre dos entidades:

- **Cliente**: Solicita información o servicios (puede ser una app de escritorio o web).
- **Servidor**: Procesa las solicitudes y devuelve los resultados.

2.1. ¿Cómo funciona?

1. **Flujo de información:**
   - El cliente envía una solicitud al servidor a través de una red (usualmente HTTP/HTTPS).
   - El servidor recibe la solicitud, ejecuta la lógica de negocio y, si es necesario, consulta la base de datos.
   - El servidor envía una respuesta de vuelta al cliente con los datos solicitados o la acción ejecutada.
2. **Interacción entre cliente y servidor:**
   - El cliente puede ser un navegador web, una aplicación móvil o de escritorio.
   - La comunicación se realiza a través de APIs (REST, GraphQL, WebSockets, etc.).
   - La base de datos generalmente está centralizada en el servidor para garantizar consistencia y seguridad.

2.2. Evaluación de la Arquitectura

Este modelo permite una clara separación entre la interfaz de usuario y la lógica del sistema, lo que **facilita la escalabilidad y el mantenimiento del software.** La **centralización de los datos en el servidor mejora la seguridad**, reduciendo el riesgo de accesos no autorizados. **Sin embargo**, su mayor desventaja es la **dependencia del servidor**: si falla, toda la aplicación deja de funcionar. También puede haber **latencias debido a la comunicación en red** y se requiere una gestión eficiente de múltiples conexiones simultáneas para

garantizar un buen rendimiento. Esta arquitectura es **ideal para aplicaciones web, sistemas de gestión empresarial** (ERP, CRM) y aplicaciones que dependen de **bases de datos centralizadas**.

Las tecnologías más utilizadas incluyen **Node.js** con Express.js y Flask o Django en **Python** para el **backend**. **En el frontend**, los clientes pueden estar desarrollados en **React, Angular o Vue.js**. Los protocolos de comunicación más usados son **HTTP/HTTPS, mediante REST o GraphQL APIs**, mientras que la persistencia de datos se maneja con **MongoDB, MySQL o PostgreSQL**.

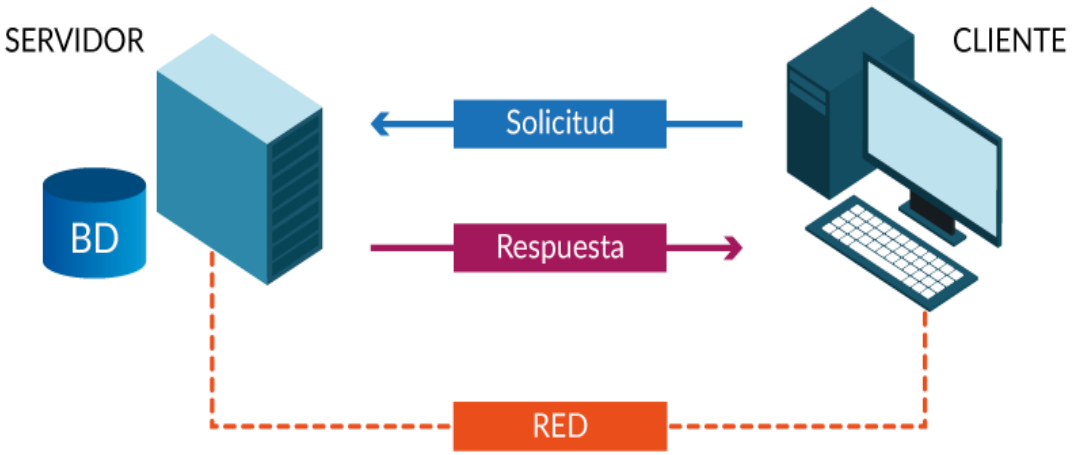

Figura 2: Arquitectura Cliente-Servidor

3. Arquitectura de Microservicios

Divide una aplicación en **múltiples servicios pequeños e independientes**, cada uno responsable de una funcionalidad específica y comunicándose entre sí mediante APIs.

3.1. ¿Cómo funciona?

1. **Flujo de información entre microservicios:**
    - Cada microservicio tiene una función específica y opera de manera independiente.

◦ Se comunican a través de protocolos como HTTP, REST, gRPC o colas de mensajes (Kafka, RabbitMQ).

◦ Pueden compartir datos a través de eventos o bases de datos independientes.

2. Interacción con los usuarios:

◦ Un usuario realiza una solicitud a través de una API Gateway, que distribuye la solicitud a los microservicios correspondientes.

◦ Cada microservicio procesa su parte de la solicitud y devuelve una respuesta.

◦ El API Gateway unifica las respuestas y las envía de vuelta al usuario.

3.2. Evaluación de la Arquitectura

La arquitectura de microservicios proporciona una **flexibilidad extrema** al permitir el desarrollo y despliegue independiente de cada servicio. Esto permite escalar partes específicas del sistema sin afectar el resto de la aplicación, además de mejorar la tolerancia a fallos. Sin embargo, introduce una **mayor complejidad en la comunicación entre servicios**, requiriendo herramientas adicionales como colas de mensajes y sistemas de monitoreo avanzados. La gestión de múltiples despliegues y bases de datos **puede dificultar la administración del sistema**. Se utiliza en **aplicaciones altamente escalables**, como plataformas de streaming, comercio electrónico y sistemas financieros con múltiples servicios independientes.

Las tecnologías más utilizadas incluyen **Docker y Kubernetes** para orquestación, **gRPC y Apache Kafka para comunicación**, y frameworks como Spring Boot (Java), FastAPI (Python) y Go para el desarrollo de los microservicios. Bases de datos como **MongoDB y Amazon DynamoDB** permiten una gestión distribuida eficiente, mientras que herramientas como **Prometheus y Jaeger ayudan en el monitoreo.**

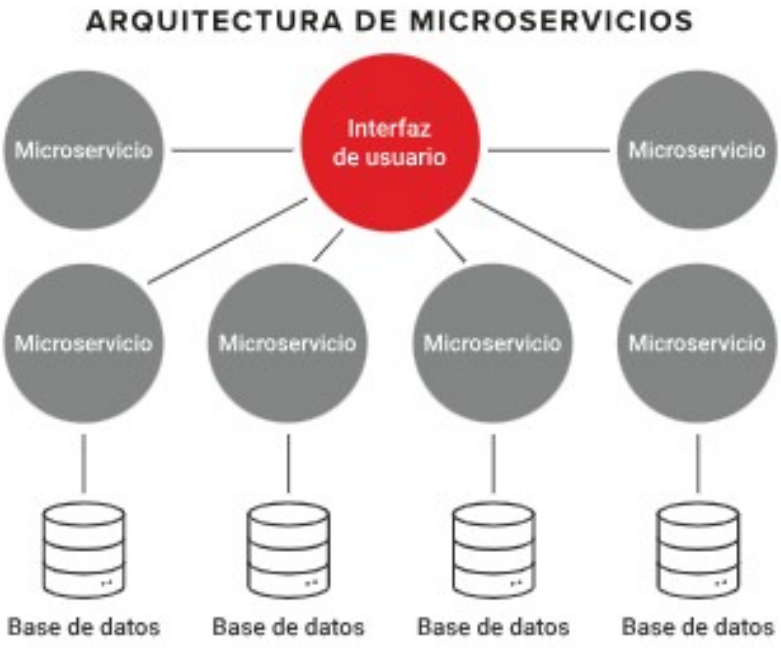

Figura 3: Arquitectura de microservicios

4. Otras Arquitecturas

4.1. Arquitectura en Capas (N-Tier)

Este modelo organiza la **aplicación en capas independientes (Presentación, Negocio y Datos)**, lo que mejora la modularidad y facilita el mantenimiento. Se usa en aplicaciones empresariales donde se necesita una separación estructurada. Tecnologías como .NET, Java EE y Angular son comunes en este modelo.

4.2. Arquitectura de Sistemas Distribuidos

En esta arquitectura, **múltiples nodos trabajan juntos para mejorar escalabilidad y rendimiento**. Se usa en plataformas en la nube como AWS, Google Cloud y Azure, junto con bases de datos distribuidas como Cassandra y CockroachDB.

4.3. Arquitectura Orientada a Servicios (SOA)

Los sistemas se dividen en **servicios reutilizables que se comunican mediante protocolos estándar como SOAP y REST**. Se usa en grandes corporaciones con múltiples sistemas interconectados, con herramientas como IBM WebSphere y WSO2.

4.4. Arquitectura Basada en Eventos

Este modelo permite que los componentes de una **aplicación reaccionen a eventos en lugar de hacer llamadas directas**, promoviendo la comunicación asíncrona. Es ideal para aplicaciones en tiempo real como **IoT, mensajería instantánea y analítica en streaming**, con herramientas como Apache Kafka y RabbitMQ.

5. Conclusión

Cada modelo de arquitectura tiene ventajas y desventajas que lo hacen adecuado para distintos escenarios. Mientras que la arquitectura monolítica es ideal para proyectos pequeños y rápidos, los microservicios permiten una escalabilidad avanzada, pero con mayor complejidad. Cliente-servidor sigue siendo la base de muchas aplicaciones, mientras que los sistemas distribuidos y basados en eventos han revolucionado la computación en la nube. **La elección de la arquitectura dependerá de los requisitos del proyecto, la escalabilidad deseada y las tecnologías disponibles.**