Interview Questions - Software Architecture Patterns & Styles

1. What is the difference between Monolithic and Microservices architecture?

- **Monolithic Architecture**:
    - **Definition**: In monolithic architecture, all components of the application (UI, business logic, database access, etc.) are tightly integrated into a single codebase. The system is deployed as a single unit.
    - **Pros**:
        - Simple to develop and deploy.
        - Easier to manage initially for small applications.
        - Less complex, as all components are in a single codebase.
    - **Cons**:
        - Difficult to scale because the entire application must scale together.
        - Tightly coupled components can make maintenance and updates harder over time.
        - A failure in one part of the system can bring down the whole application.
    - **Use Cases**: Suitable for small applications or startups where simplicity and quick deployment are priorities.
- **Microservices Architecture**:
    - **Definition**: Microservices architecture splits an application into small, independently deployable services, each responsible for a specific business capability. Each service communicates with others through APIs or messaging.
    - **Pros**:

        - Independent scaling, allowing for better resource allocation.
        - Fault tolerance: A failure in one service does not affect the entire system.
        - Flexibility in choosing technology stacks for different services.
    - **Cons**:
        - Increased complexity in communication and data management between services.
        - Requires a robust DevOps pipeline and automation for managing deployments.
        - Can be challenging to maintain consistent data across services.
    - **Use Cases**: Ideal for large-scale, complex applications, such as e-commerce platforms or cloud-based systems.

---

2. When would you choose a Layered architecture over Microservices?

- **Layered (N-Tier) Architecture** is often chosen over Microservices in cases where:
    - **Simpler systems** with fewer components are involved. If the application doesn't need to scale significantly or require independent services, a layered approach is easier and less complex.
    - **Cost and time**: Microservices require more infrastructure, DevOps pipelines, and management overhead. If the project timeline or budget is tight, a layered architecture might be the better choice.
    - **Existing systems**: If you are working with legacy systems that are already structured with layers, continuing with this approach can be more practical than refactoring into microservices.
    - **Use Case**: Enterprise applications, CRMs, or systems where different concerns (e.g., presentation, business logic, data access) need clear separation but not necessarily independent scaling.

---

3. What are the pros and cons of Event-Driven Architecture?

- **Event-Driven Architecture** uses events (messages) to trigger actions across various components, creating a decoupled system where components do not directly communicate with each other.

- **Pros**:

    - **Loose coupling**: Components do not directly depend on one another, which allows for more flexibility in making changes to individual components without affecting the rest of the system.

    - **Asynchronous processing**: Supports decoupled, asynchronous workflows, which can improve performance in systems that require real-time updates or high-volume data processing.

    - **Scalability**: Event-driven systems can easily scale, especially for systems handling a high number of events, such as IoT applications or financial systems.

- **Cons**:

    - **Complexity**: Debugging and tracing events across different services can be challenging, especially when issues arise in event streams or messaging.

    - **Data consistency**: Ensuring data consistency across multiple services or components that are asynchronously reacting to events can be difficult.

    - **Overhead**: Requires a messaging system and event buses, adding some operational overhead.

- **Use Cases**: Real-time systems, IoT platforms, financial trading applications, and systems that require high scalability and decoupling.

---

4. How does architecture impact system scalability and performance?

- **Scalability**:

    - The choice of architecture defines how well the system can handle increasing loads or expand in terms of resources. For example, **microservices** allow independent scaling of individual components, making them suitable for systems that expect high traffic or rapid growth.

    - **Monolithic** architectures can be harder to scale because the whole application must be scaled together, which can lead to inefficiencies and resource wastage.

- **Performance**:
    - The architecture determines how efficiently the system can process requests and handle concurrent users. For instance, **Event-Driven** architectures enable asynchronous processing, improving responsiveness for real-time systems.
    - **Layered** architectures may introduce performance overhead due to the communication between layers, whereas **microservices** might face latency due to inter-service communication.
- In short, architecture directly affects both how much traffic a system can handle (scalability) and how quickly it can respond to requests (performance).

---

5. How do business requirements influence architectural decisions?

- **Business Needs**: The architecture should align with the goals of the business. For example, if the business needs rapid development and flexibility in product offerings, **Microservices** might be the best choice, allowing teams to work independently on different services.
- **Speed to Market**: If a business needs to release a product quickly, a **Monolithic** architecture might be more suitable due to its simplicity and quicker deployment.
- **Scalability and Growth**: For businesses expecting significant growth or high volumes of users, **Microservices** or **Event-Driven** architectures will be more beneficial as they provide scalability and resilience.
- **Cost Considerations**: Smaller businesses or startups with limited resources may opt for a simpler **Monolithic** or **Layered** architecture, while larger businesses may invest in **Microservices** to scale efficiently.

---

6. Can you explain the key differences between Layered and Client-Server architectures?

- **Layered Architecture**:
    - Divides the system into layers (e.g., Presentation, Business Logic, Data), each with distinct responsibilities.

    - Promotes separation of concerns, making it easier to manage and maintain as the application grows.
    - Works well in enterprise applications where each layer is responsible for specific tasks.
- **Client-Server Architecture**:
    - Involves a server that provides resources or services, and a client that interacts with the server to request those services.
    - Typically simpler than layered, as it focuses on the interaction between a client and a server, but may not have the same separation of concerns as a layered approach.

---

7. What are the main challenges in maintaining a Monolithic system as it grows?

- **Tightly Coupled Components**: As the codebase grows, it becomes more difficult to update or modify specific components without affecting other parts of the system.
- **Scaling Issues**: The entire monolithic application must be scaled together, leading to inefficiencies if only certain components experience high traffic.
- **Deployment Complexity**: Any update or bug fix requires redeploying the entire system, which can be risky and time-consuming.
- **Slower Development**: As more developers work on a single codebase, coordination becomes harder, and it becomes difficult to make changes quickly.

---

8. How do you decide between Monolithic and Microservices for a new system?

- **Monolithic**:
    - Ideal for small to medium-sized applications where the complexity and scale do not justify the overhead of microservices.
    - A monolithic approach allows for faster initial development and simpler deployment.

- **Microservices**:
    - Best for large-scale applications with multiple independent modules that can scale and evolve separately.
    - Choose microservices when your system requires high scalability, independent development teams, and fault tolerance.

---

9. What role does fault tolerance play in Microservices architecture?

- **Fault Tolerance** is a key advantage of microservices. Since each service is independent, failures in one service do not affect others.
- **Circuit breakers**, **retry logic**, and **load balancing** are often used in microservices to ensure that failure in one service does not bring down the entire system.
- With fault tolerance, microservices can continue to operate even if individual services experience issues, improving overall system reliability.

---

10. How do event-driven systems handle real-time data?

- **Real-Time Data Processing**: Event-driven systems process data as soon as events are generated, allowing for real-time updates across the system.
- **Asynchronous Processing**: Events are published and subscribed to asynchronously, enabling the system to respond to high volumes of data quickly without blocking processes.
- **Use of Queues**: Event-driven systems often rely on message queues (e.g., Kafka, RabbitMQ) to handle and route events efficiently, ensuring that all components can react to events as they occur.