1. Layered Architecture

- **Characteristics**: Divides the system into layers (e.g., presentation, business logic, data access).
- **Benefits**: Clear separation of concerns, easy maintenance, reusable components.
- **Limitations**: Rigid structure, performance bottlenecks, scaling challenges.
- **Use Case**: Suitable for traditional enterprise or legacy applications.

2. Microservices Architecture

- **Characteristics**: Breaks the system into independent, small services, each with its own functionality and database.
- **Benefits**: Scalability, flexibility, fault isolation, and technology agnosticism.
- **Limitations**: High complexity in deployment and monitoring, data consistency issues.
- **Use Case**: Large-scale systems like e-commerce platforms, media streaming services (e.g., Netflix).

3. Event-Driven Architecture

- **Characteristics**: Components communicate via events (asynchronous), often using an event bus or message queue.
- **Benefits**: Loose coupling, real-time processing, fault tolerance, and scalability.
- **Limitations**: Complex event management, debugging difficulty, eventual consistency.
- **Use Case**: Real-time systems, such as financial platforms, IoT, and social media feeds.

When designing systems, principles like **modularity**, **scalability**, **maintainability**, and **flexibility** play a crucial role. Here's how they apply to a real-world system, such as an **e-commerce platform**:

Example: E-Commerce Platform Design (Microservices Style)

- **Modularity**:
  - Each feature (e.g., **User Authentication**, **Payment System**, **Product Management**) is a separate microservice. This modularity enables development teams to work independently on different services without causing disruption to others.
- **Scalability**:
  - Microservices can scale independently. If a high demand is expected on the **Payment Service**, it can be scaled without impacting the **Product Management** service.
- **Maintainability**:
  - Since each service is isolated, updating or maintaining a service becomes easier. The **Payment Service** can be updated without affecting the rest of the system.
- **Flexibility**:
  - New features or services (e.g., **Product Recommendations**, **Customer Support**) can be added independently.