Different Types of Common System Architecture

System architecture refers to the structure and design of a system, defining how its components interact to deliver functionality. There are several common types of system architectures, each suited to specific use cases, complexities, and business requirements.

1. Monolithic Architecture

- **Definition:** The entire system is built as a single, unified unit where all components (UI, business logic, data access) are tightly integrated.
- **Characteristics:**
  - One large codebase.
  - Simple to develop initially but harder to scale and maintain.
- **Use Cases:**
  - Small-scale applications.
  - Systems with low complexity and infrequent updates.
- **Examples:** Traditional web applications built on early frameworks like ASP.NET or Ruby on Rails.

---

2. Layered (Tiered) Architecture

- **Definition:** Divides the system into layers (tiers), each with a specific role, such as presentation, business logic, and data access.
- **Characteristics:**
  - Commonly follows the **3-tier model:** Presentation, Business Logic, Data.
  - Layers are loosely coupled.
- **Use Cases:**
  - Enterprise applications with structured workflows.
  - Scalable web apps requiring clear separation of concerns.
- **Examples:** E-commerce websites, ERP systems.

---

3. Client-Server Architecture

- **Definition:** A system where clients (front-end devices) request services or data from a server (back-end system).
- **Characteristics:**
  - Centralized control (server) with multiple clients.
  - Suitable for distributed environments.
- **Use Cases:**
  - File-sharing systems, email systems, or database-driven apps.
- **Examples:** Web browsers communicating with web servers.

---

4. Microservices Architecture

- **Definition:** A system design where functionalities are broken into small, independent services that communicate over APIs.
- **Characteristics:**
  - Highly modular and scalable.
  - Each service can use different technologies and be deployed independently.
- **Use Cases:**
  - Large-scale, complex applications requiring agility.
  - Applications with frequent updates and diverse functionalities.
- **Examples:** Netflix, Amazon, Uber.

---

5. Event-Driven Architecture

- **Definition:** A system design where components communicate by producing and consuming events, often using an event broker or bus.
- **Characteristics:**
  - Reactive and real-time communication.
  - Decouples event producers from consumers.
- **Use Cases:**
  - Systems requiring high responsiveness, like IoT applications.
  - Streaming platforms and financial transaction systems.
- **Examples:** Stock trading platforms, Kafka-based systems.

---

6. Service-Oriented Architecture (SOA)

- **Definition:** An architecture style where system components are delivered as reusable services, often exposed through a service bus.
- **Characteristics:**
  - Services are loosely coupled.
  - Focuses on reusability and interoperability.
- **Use Cases:**
  - Enterprise-level integration of legacy systems.
  - Scenarios requiring a focus on reusing business logic.
- **Examples:** Banking systems, healthcare systems with HL7.

---

7. Serverless Architecture

- **Definition:** A cloud-native architecture where the application is built on functions executed on demand, with no need for server management.
- **Characteristics:**
  - Pay-as-you-go model for execution.
  - Scales automatically based on demand.
- **Use Cases:**
  - Lightweight, event-driven applications.
  - Systems with sporadic workloads.
- **Examples:** AWS Lambda, Azure Functions.

---

8. Peer-to-Peer (P2P) Architecture

- **Definition:** A decentralized system where nodes (peers) interact directly with one another without a central server.
- **Characteristics:**
  - Resilient and fault-tolerant.
  - No single point of failure.
- **Use Cases:**
  - File-sharing systems, blockchain applications.
- **Examples:** BitTorrent, Bitcoin.

---

9. Distributed Architecture

- **Definition:** A system where components are distributed across multiple locations and communicate over a network.
- **Characteristics:**
  - High availability and fault tolerance.

    - Complex to manage and debug.
- **Use Cases:**
    - Systems requiring scalability and high reliability.
    - Real-time data processing systems.
- **Examples:** Hadoop, distributed databases like Cassandra.

---

10. Modular / Domain Architecture

- **Definition:** A system composed of interchangeable, self-contained modules that work together.
- **Characteristics:**
    - Highly maintainable and adaptable.
    - Supports plug-and-play functionality.
- **Use Cases:**
    - Systems requiring flexibility and adaptability.
    - Systems with evolving requirements.
- **Examples:** IoT systems with modular sensors.

---

11. Component-Based Architecture

- **Definition:** A design that organizes the system into reusable components that encapsulate functionality and interact through interfaces.
- **Characteristics:**
    - Encourages reusability and separation of concerns.
    - Components can be developed and tested independently.
- **Use Cases:**
    - Software with reusable parts, such as UI libraries.
- **Examples:** React.js, Angular.

---

12. Hybrid Architecture

- **Definition:** A combination of multiple architectural styles to meet specific requirements.
- **Characteristics:**
    - Tailored to the application's needs.
    - Can be complex to design and maintain.
- **Use Cases:**
    - Complex applications requiring diverse functionality.
    - Systems with varying workloads.

- **Examples:** A system using microservices for core functions and serverless for auxiliary tasks.

Choosing the Right Architecture

The selection depends on:

1. **Scale of the Application:** Small, medium, or large-scale.
2. **Complexity:** Single-functionality vs. multi-functional systems.
3. **Performance Needs:** Real-time processing vs. batch processing.
4. **Future Growth:** Scalability and modularity requirements.

Revision #1
Created 25 November 2024 16:05:42 by Admin
Updated 25 November 2024 16:11:16 by Admin