

Architectural Styles

Architectural Styles – Quick Reference Notes

1. What Is an Architectural Style?

- **Definition:** A set of design principles, patterns, and conventions that guide the overall structure of a software system.
 - **Purpose:**
 - Provide a common vocabulary for architects and developers.
 - Influence system qualities (e.g., scalability, maintainability, security).
 - Help decide trade-offs early in a project.
-

2. Classification by Scope

Scope	Typical Examples	Primary Concern
Enterprise	Service-Oriented Architecture (SOA), Micro-services	Business-level integration, governance
System	Layered (N-tier), Pipes-and-Filters, Client-Server	Internal component organization
Component	Model-View-Controller (MVC), Repository, Event-Sourcing	Reusability and testability of individual modules
Infrastructure	Cloud-Native, Serverless, Edge Computing	Deployment, elasticity, resource management

3. Core Architectural Styles

3.1 Layered (N-Tier) Architecture

- **Structure:** Stacked layers where each layer only communicates with the one directly below/above.
- **Typical Layers:** Presentation ? Application/Business ? Domain ? Persistence ? Infrastructure.

Architectural Styles

- **Benefits:** Separation of concerns, easy to replace or upgrade a layer, clear dependency direction.
- **When to Use:** Classic enterprise applications, systems with distinct UI, business logic, and data access concerns.

3.2 Client-Server

- **Structure:** One or more **clients** request services from a **server** that provides resources or processing.
- **Variants:** Two-tier (direct DB access), three-tier (adds an application server), multi-tier.
- **Benefits:** Centralized data management, easier security enforcement.
- **When to Use:** Web applications, mobile back-ends, any scenario with remote resource sharing.

3.3 Service-Oriented Architecture (SOA)

- **Structure:** Independent **services** expose well-defined interfaces (often SOAP/REST) and communicate via a **bus** or message broker.
- **Key Concepts:** Loose coupling, contract-first design, service composition, governance.
- **Benefits:** Reusability across business domains, technology heterogeneity, easier integration.
- **When to Use:** Large enterprises with many disparate systems, need for long-term integration.

3.4 Microservices

- **Structure:** System decomposed into many **small, autonomous services**, each owning its data and deployed independently.
- **Characteristics:** Bounded context, API-first, decentralized data management, containerization.
- **Benefits:** Independent scaling, fault isolation, rapid deployment cycles.
- **When to Use:** High-growth products, cloud-native environments, teams organized around business capabilities.

3.5 Event-Driven Architecture (EDA)

Architectural Styles

- **Structure:** Components communicate by **producing** and **consuming** events via an event broker (e.g., Kafka, RabbitMQ).
- **Patterns:** Event Notification, Event-Carried State Transfer, CQRS (Command Query Responsibility Segregation).
- **Benefits:** Loose coupling, natural fit for asynchronous processing, high scalability.
- **When to Use:** Real-time analytics, IoT, systems requiring eventual consistency.

3.6 Pipes and Filters

- **Structure:** Data flows through a **pipeline** of **filters** (processing steps) connected by **pipes** (data channels).
- **Benefits:** Reusability of filters, easy to add/replace processing steps, parallel execution.
- **When to Use:** Compilers