Software Architecture Overview

Key concepts and components of software architecture

Software architecture serves as the blueprint for structuring and organizing software systems. It encompasses several key concepts and components that define how the system functions as a whole:

Components: These are modular, self-contained units that encapsulate specific functionality. Components can be libraries, modules, or services that work together to fulfill the system's requirements.

Interfaces: Interfaces define how components interact with each other. Well-defined interfaces ensure proper communication and integration between components, promoting modularity and reusability.

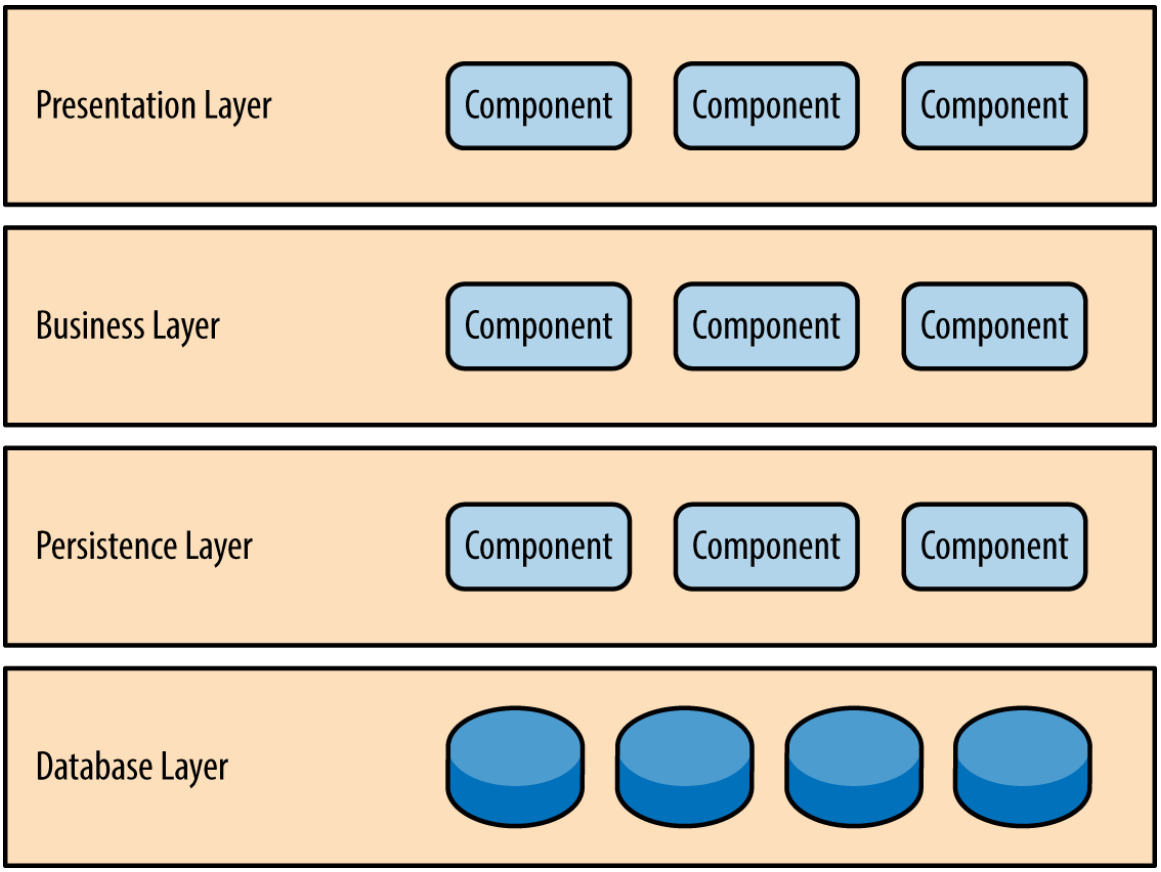
Architectural Patterns: These are established templates for solving common architectural problems. Examples include the layered architecture, where different responsibilities are segregated into distinct layers, and the client-server architecture, where client and server components interact over a network.

Design Patterns: These are reusable solutions for common design challenges within a specific context. Examples include the Singleton pattern, which ensures only one instance of a class exists, and the Factory pattern, which centralizes object creation.

Examples of different architectural styles

Layered Architecture: This style organizes components into distinct layers, each responsible for a specific aspect of functionality. It promotes separation of concerns and ease of maintenance. It's suitable for applications where clear separation of presentation, logic, and data layers is essential.

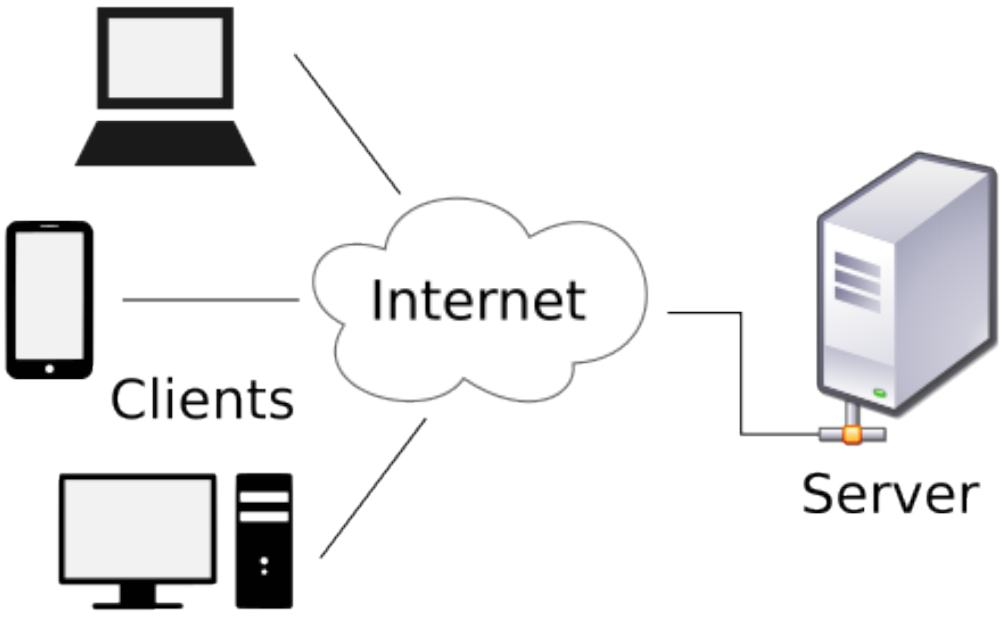
Graph: general example of layered architecture



Source: www.oreilly.com

Client-Server Architecture: In this style, clients (user interfaces) communicate with servers (centralized processing units) over a network. It's ideal for applications that require centralized data management, security enforcement, and scalability.

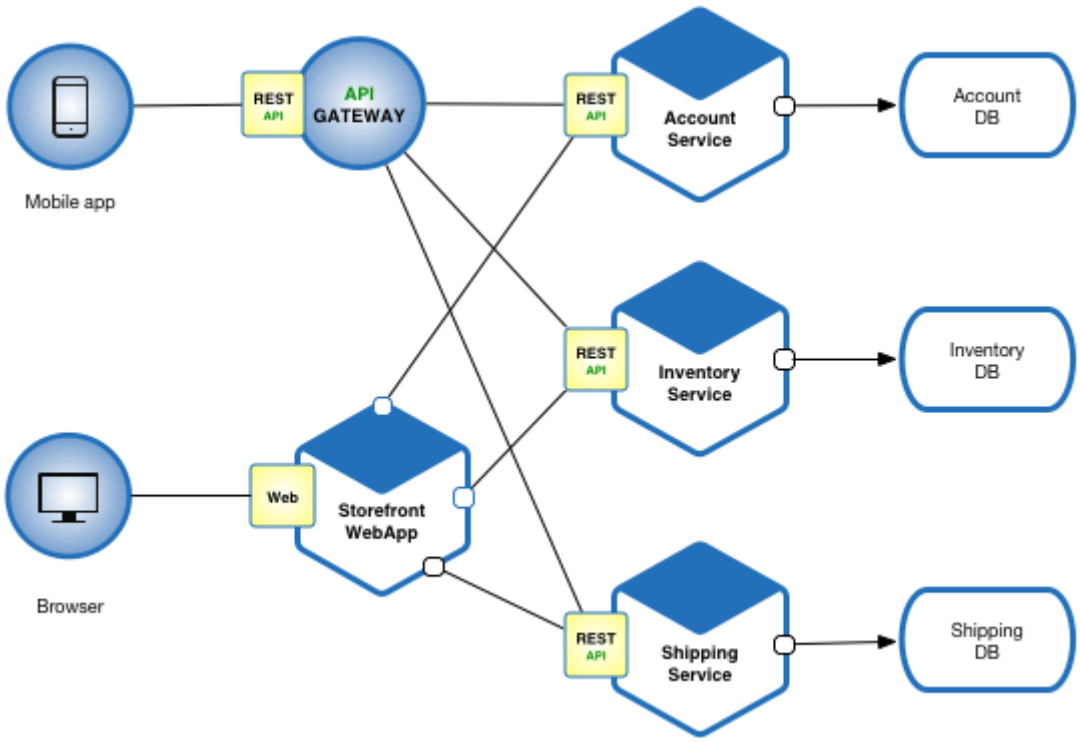
Graph: general example of client-server architecture



Source: www.en.wikipedia.org

Microservices Architecture: Microservices break down applications into small, independently deployable services, each responsible for a specific business capability. This promotes agility, scalability, and ease of maintenance, making it suitable for complex systems with varying demands.

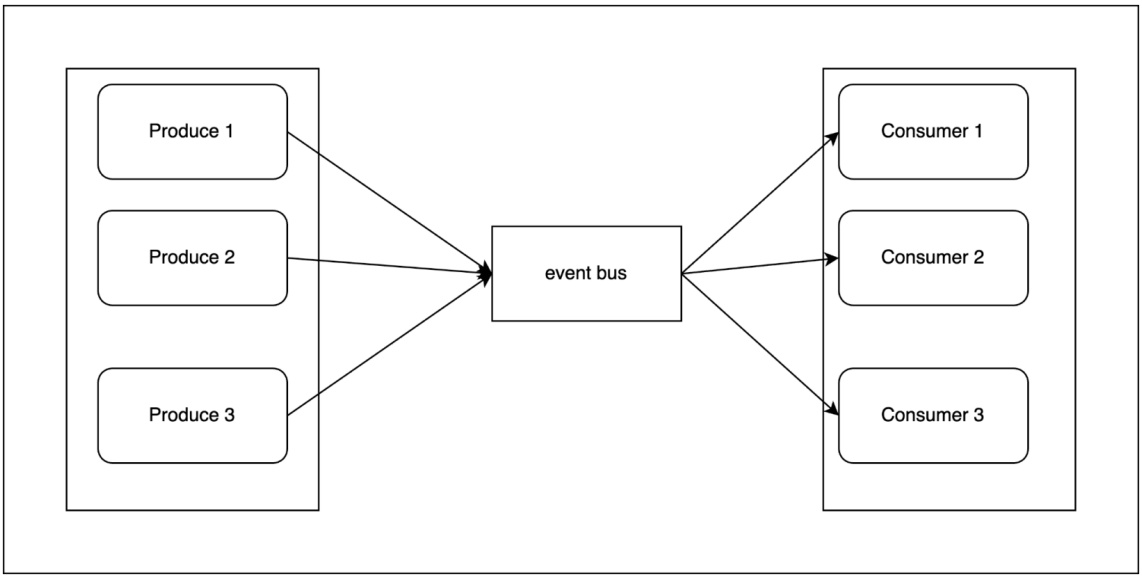
Graph: general example of microservices architecture



Source: www.microservices.io

Event-Driven Architecture: This style emphasizes communication between components through events and messages. It's suitable for systems with asynchronous and loosely coupled interactions, such as real-time updates, notifications, and complex workflows.

Graph: general example of event-driven architecture



Choosing the appropriate architectural style depends on factors like system complexity, scalability needs, inter-component interactions, and development team expertise.

Revision #1
Created 9 October 2025 11:44:05 by RISA
Updated 9 October 2025 11:53:07 by RISA