6 Software Architecture Patterns You Should Know.  
  
Choosing the right architecture isn’t about following trends.  
  
It’s about aligning with your application’s needs, your team’s expertise, and long-term scalability and maintainability.  
  
And it's one of the biggest decisions you'll make for your application.  
  
Let’s examine some of the most widely used architecture patterns:  
  
𝗠𝗼𝗻𝗼𝗹𝗶𝘁𝗵𝗶𝗰  
  
A traditional approach where the entire application is built as a single unit. Simple to develop and deploy, but scaling and maintaining large monolithic systems can become challenging over time. Some teams adopt a modular monolith to improve maintainability while keeping a single deployment.  
  
𝗦𝗲𝗿𝘃𝗲𝗿𝗹𝗲𝘀𝘀  
  
With serverless, developers focus on writing code while the cloud provider manages infrastructure, scaling automatically based on demand. Beyond functions (FaaS), serverless includes managed databases, authentication, and messaging. While it reduces operational overhead, it also has trade-offs.  
  
𝗘𝘃𝗲𝗻𝘁-𝗗𝗿𝗶𝘃𝗲𝗻 (𝗘𝗗𝗔)  
  
In event-driven systems, components communicate asynchronously through events, enabling reactive, real-time applications with better scalability and resilience. This pattern is great for microservices and streaming apps but can make debugging and event consistency challenging.  
  
  
𝗠𝗶𝗰𝗿𝗼𝘀𝗲𝗿𝘃𝗶𝗰𝗲𝘀  
  
A system built from independently deployable services, each responsible for a specific function. This pattern enables flexibility, independent scaling, and faster deployments but introduces complexity in communication, data consistency, monitoring, and orchestration.  
  
  
𝗗𝗼𝗺𝗮𝗶𝗻-𝗗𝗿𝗶𝘃𝗲𝗻 𝗗𝗲𝘀𝗶𝗴𝗻 (𝗗𝗗𝗗)  
  
DDD is not an architecture itself but a software design approach that shapes architectural decisions. It structures systems to align closely with business domains, emphasizing domain modeling, bounded contexts, and strategic design to reduce technical debt & improve maintainability.  
  
  
𝗟𝗮𝘆𝗲𝗿𝗲𝗱 (𝗡-𝗧𝗶𝗲𝗿)  
  
A structured approach where apps are divided into logical layers (eg; presentation, business logic, & data). Improves separation of concerns and maintainability, but rigid dependencies can introduce bottlenecks. More modern patterns are often favored today.  
  
No single architecture fits every use case. Each pattern has trade-offs, and the right choice depends on your system’s needs.  
  
💬 Which of these have you worked with the most? 💭

Activate to view larger image,

