# When to Use Different Architecture Styles

Below is a table that compares **Monolith**, **SOA**, **Microservices**, and **Serverless** architecture styles based on various criteria.

It provides guidance on when to use each style, taking into account architecture qualities, company stage/size, team expertise, scalability needs, and other factors.

| | Monolithic Architecture | Service-Oriented Architecture (SOA) | Microservices Architecture | Serverless Architecture |
|---|---|---|---|---|
| **Architecture Qualities** | - Simple, tightly coupled system<br>- Single deployment unit<br>- Easier to develop and test initially | - Decoupled services with shared communication protocols (e.g., HTTP, SOAP)<br>- Reusable services | - Highly decoupled, independent services<br>- Each service can be developed, deployed, and scaled independently | - Stateless, event-driven functions<br>- Fully managed by cloud providers<br>- Pay-per-use pricing model |
| **Company Stage/Size** | - Startups or small teams<br>- Early-stage projects with limited resources | - Medium-sized companies or enterprises<br>- Projects requiring integration of multiple systems or legacy applications | - Large enterprises with complex systems<br>- Mature organizations with established DevOps practices | - Companies leveraging cloud-native solutions<br>- Projects with sporadic or unpredictable traffic |

| | | | | |
|---|---|---|---|---|
| **Team Expertise** | - Small, cross-functional teams<br>- Limited need for specialized roles | - Teams familiar with distributed systems<br>- Requires expertise in integration patterns and protocols | - Experienced teams with knowledge of containerization, CI/CD, and distributed systems | - Teams experienced with cloud platforms and serverless frameworks |
| **Scalability Needs** | - Low to moderate scalability<br>- Scaling requires redeploying the entire application | - Moderate scalability<br>- Individual services can be scaled independently | - High scalability<br>- Independent scaling of services allows fine-grained resource allocation | - High scalability for specific functions<br>- Automatic scaling handled by the cloud provider |
| **Deployment Complexity** | - Simple deployment<br>- Single artifact to deploy | - Moderate complexity<br>- Multiple services to deploy and manage | - High complexity<br>- Requires orchestration tools (e.g., Kubernetes) and robust CI/CD pipelines | - Minimal deployment complexity<br>- Cloud provider manages deployment and scaling |
| **Development Speed** | - Fast initial development<br>- Slower as the codebase grows | - Moderate development speed<br>- Requires coordination between services | - Slower initial development<br>- Faster iteration once the infrastructure is set up | - Fast development for small, focused functions |
| **Maintenance Overhead** | - High maintenance overhead as the codebase grows | - Moderate maintenance overhead<br>- Changes are isolated to individual services | - Low maintenance overhead for individual services<br>- Higher operational complexity due to many services | - Minimal maintenance overhead<br>- Cloud provider handles |

| | | | | |
|---|---|---|---|---|
| | - Changes impact the entire system | | | infrastructure management |
| **Fault Tolerance** | - Low fault tolerance<br>- A failure in one part can bring down the entire system | - Moderate fault tolerance<br>- Failures are isolated to individual services | - High fault tolerance<br>- Failures are isolated to individual services | - High fault tolerance<br>- Functions are isolated and stateless |
| **Cost Efficiency** | - Cost-effective for small applications | - Moderate costs<br>- Requires infrastructure for service communication and management | - High costs for large-scale deployments<br>- Requires investment in DevOps tools and infrastructure | - Cost-effective for sporadic workloads<br>- Pay only for execution time |
| **Use Cases** | - Small applications or prototypes<br>- Applications with simple business logic | - Enterprise applications requiring integration of multiple systems<br>- Legacy system modernization | - Large-scale, complex applications<br>- Applications requiring independent scaling and frequent updates | - Event-driven applications (e.g., file processing, real-time analytics)<br>- APIs with low-latency requirements |
| **Examples** | - Blogging platform<br>- Simple e-commerce site | - Banking systems integrating multiple services<br>- Enterprise resource planning (ERP) systems | - Netflix, Uber, Amazon<br>- E-commerce platforms with modular components | - Image resizing service<br>- Real-time chatbots<br>- Scheduled tasks (e.g., sending emails) |

## Key Takeaways

1. **Monolithic Architecture**:
   - Best for small teams, startups, or early-stage projects.
   - Suitable for applications with simple business logic and low scalability needs.
   - Avoid for large, complex systems where scalability and maintainability are critical.
2. **Service-Oriented Architecture (SOA)**:
   - Ideal for medium-sized companies or enterprises integrating multiple systems.
   - Suitable for projects requiring reusable services and standardized communication protocols.
   - Avoid for small projects where the overhead of SOA is unnecessary.
3. **Microservices Architecture**:
   - Best for large enterprises with complex systems and mature DevOps practices.
   - Suitable for applications requiring high scalability, fault tolerance, and frequent updates.
   - Avoid for small teams or projects without the expertise to manage distributed systems.
4. **Serverless Architecture**:
   - Ideal for cloud-native applications with sporadic or unpredictable traffic.
   - Suitable for event-driven workloads and small, focused functions.
   - Avoid for long-running processes or applications requiring fine-grained control over infrastructure.