<https://www.talend.com/resources/microservices-vs-soa/>

<https://dzone.com/articles/microservices-vs-soa-whats-the-difference>

|  |  |  |
| --- | --- | --- |
|  | **Microservices** | **SOA** |
| **Architecture** | Designed to host services which can function independently | Designed to share resources across services |
| **Component sharing** | Typically does not involve component sharing | Frequently involves component sharing |
| **Granularity** | Fine-grained services | Larger, more modular services |
| **Data storage** | Each service can have an independent data storage | Involves sharing data storage between services |
| **Governance** | Requires collaboration between teams | Common governance protocols across teams |
| **Size and scope** | Better for smaller and web-based applications | Better for large scale integrations |
| **Communication** | Communicates through an API layer | Communicates through an ESB |
| **Coupling and cohesion** | Relies on bounded context for coupling | Relies on sharing resources |
| **Remote services** | Uses REST and JMS | Uses protocols like SOAP and AMQP |
| **Deployment** | Quick and easy deployment | Less flexibility in deployment |

**Major Differences Between Microservices and SOA in Detail**

* **Service Granularity**: Service components within a microservices architecture are generally single-purpose services that do one thing really, really well. With SOA, service components can range in size anywhere from small application services to very large enterprise services. In fact, it is common to have a service component within SOA represented by a large product or even a subsystem.
* **Component Sharing**: Component sharing is one of the core tenets of SOA. As a matter of fact, component sharing is what enterprise services are all about. SOA enhances component sharing, whereas MSA tries to minimize on sharing through “bounded context.” A bounded context refers to the coupling of a component and its data as a single unit with minimal dependencies. As SOA relies on multiple services to fulfill a business request, systems built on SOA are likely to be slower than MSA.
* **Middleware vs API layer**: The microservices architecture pattern typically has what is known as an API layer, whereas SOA has a messaging middleware component. The messaging middleware in SOA offers a host of additional capabilities not found in MSA, including mediation and routing, message enhancement, message, and protocol transformation. MSA has an API layer between services and service consumers.
* **Remote services**: SOA architectures rely on messaging (AMQP, MSMQ) and SOAP as primary remote access protocols. Most MSAs rely on two protocols – REST and simple messaging (JMS, MSMQ), and the protocol found in MSA is usually homogeneous.
* **Heterogeneous interoperability**: SOA promotes the propagation of multiple heterogeneous protocols through its messaging middleware component. MSA attempts to simplify the architecture pattern by reducing the number of choices for integration. If you would like to integrate several systems using different protocols in a heterogeneous environment, you need to consider SOA. If all your services could be exposed and accessed through the same remote access protocol, then MSA is a better option.

It is not that simple to tell which architecture is better than another. It mainly depends on the purpose of the application you are building. SOA is better suited for large and complex business application environments that require integration with many heterogeneous applications; smaller applications are not a good fit for SOA as they don’t need a messaging middleware component. Microservices, on the other hand, are better suited for smaller and well-partitioned, web-based systems in which microservices give you much greater control as a developer. The conclusion is that since they both have different architecture characteristics, it mainly depends on the purpose of the application you are building.