SERVICE-ORIENTED ARCHITECTURE VS MICROSERVICES ARCHITECTURE: COMPARING SOA TO MSA

In computing, a service refers to a single or collective units of software that perform repetitive, redundant tasks. In the era of cloud computing, applications are composed of a collection of services that collectively perform various functions to support the application's overall functionality.

In this article, we explore Microservices Architecture (MSA) and Service-Oriented Architecture (SOA) as two common service-based architectures—how they both rely on services as the main component and how they differ in terms of service characteristics.

Let's take a look.

What is a service-oriented architecture (SOA)?

A service-oriented architecture follows a design of multiple self-contained, discrete, and repeatable services that are collectively used to form a service mesh of an application's functionalities holistically.

This enables a framework of application components to interact and offer services with other components by leveraging a service interface (communication protocol).

SOA design principles

The principles of a Service Oriented Architecture may differ depending on your use case. Here are some common principles that segregate services to form an SOA:

- Abstraction
- Reusability
- Granularity
- Standardized contract
- Autonomy
- Statelessness
- Discovery

Features of SOA

One fundamental use case of an SOA is to allow you to build an application by using multiple distinct services collectively, where each service consists of a unique business or application logic.

Other than that, some common features of SOA include:

- "Share as much as possible" architecture
- Importance on business functionality reuse
- Common governance and standards
- Enterprise service bus (ESB) for communication
- Multiple message protocols
- Common platform for all services deployed to it
- Multi-threaded with more overheads to handle I/O
- Maximum application service reusability
- More likely to use traditional relational databases
- Not preferred in a DevOps model

What is a microservice architecture (MSA)?

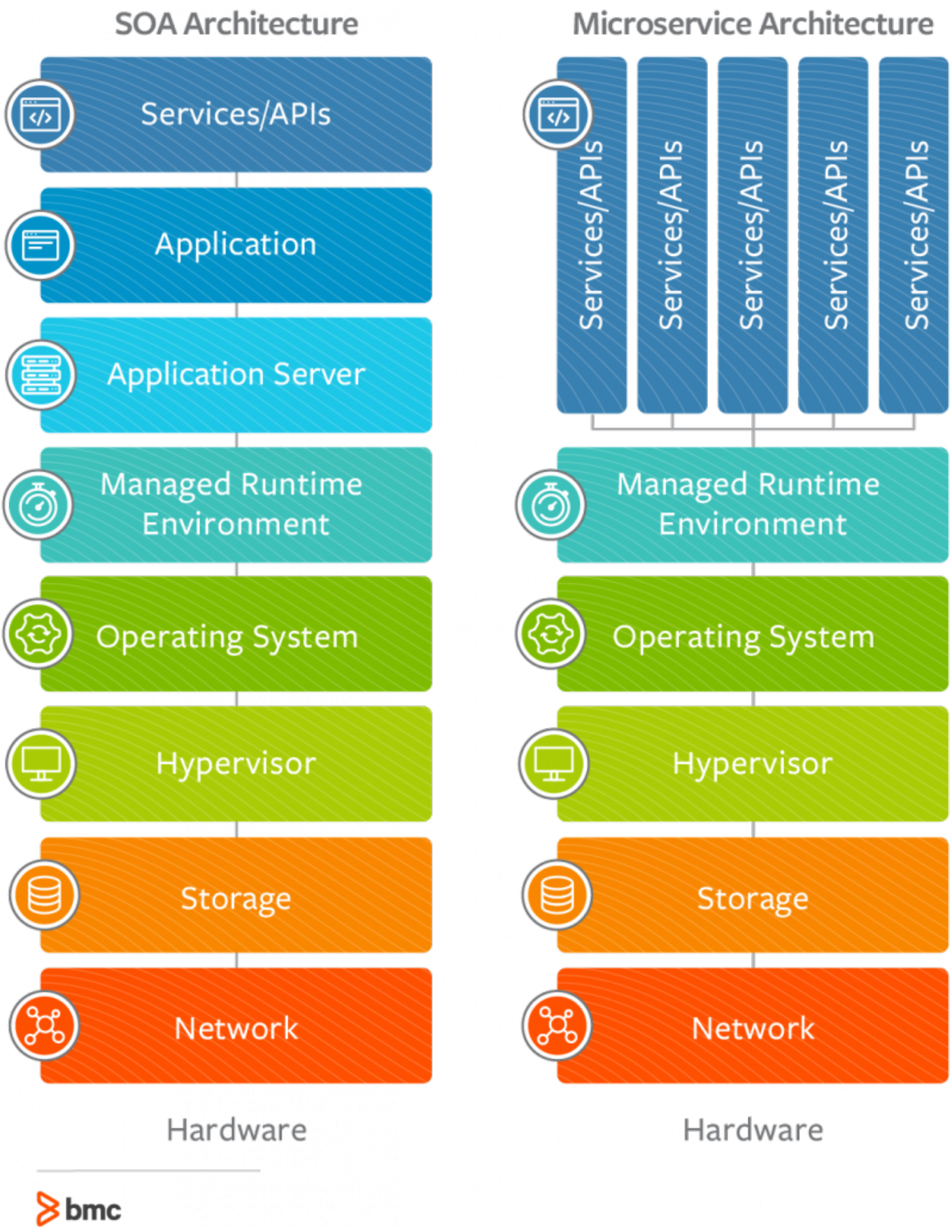
A microservice architecture, often known as microservices, follows an SOA pattern by breaking a single application into multiple *loosely coupled*, independent services yet working with each other.

Often considered the perfect use case of containerization, microservices are fairly routine for organizations to deploy each of such micro-services on separate containers. This enables an efficient framework of multiple services that are flexible, portable, and platform-agnostic—allowing each service to have different operating systems and databases while running in its own process.

Features of MSA

- "Share as little as possible" architecture
- Importance on the concept of bounded context
- Relaxed governance, with more focus on people
- Efficient collaboration and freedom in choosing platform and technologies
- Simple, less elaborate messaging system
- Lightweight protocols such as HTTP/REST and AMQP

- Single-threaded usually with the use of Event Loop features for non-locking I/O handling
- Containers work very well in MSA and are considered perfect for a DevOps model
- More focused on decoupling
- Uses modern, non-relational databases



Microservices vs SOA: key differences

Let's look at the key differences between SOA and MSA.

Coordination

SOA requires coordination with multiple groups to create business requests.

On the contrary, there is little or no coordination among services in an MSA. In the event coordination is needed among service owners, it is done through small application development teams, and services can be quickly developed, tested, and deployed.

Service granularity

The prefix “micro” in microservices refers to the granularity of its internal components. Service components within MSA are generally single-purpose services that do one thing really well.

Services in SOA usually include much more business functionality and are often implemented as complete subsystems.

Component sharing

SOA enhances component sharing, whereas MSA tries to minimize 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 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.

Contract decoupling

Contract decoupling is the holy grail of abstraction. It offers the greatest degree of decoupling between services and consumers. It is one of the fundamental capabilities offered within SOA—but MSA doesn't support contract decoupling.

Which architecture to choose?

Here are a few key considerations when opting for either of the patterns:

- **SOA is better suited for large and complex business application environments** that require integration with many heterogeneous applications. However, workflow-based applications that have a well-defined processing flow are a bit difficult to implement using SOA patterns. Small applications are also not a good fit for SOA as they don't need a messaging middleware component.
- **The MSA pattern is well suited for smaller and well partitioned web-based systems.** The lack of messaging middleware is one of the key factors that make MSA unfit for complex environments.
- **Control vs orchestration.** When developing an application from scratch, using MSA is considered a pragmatic choice as it offers greater control as a developer. On the other hand, if the goal is to orchestrate business processes, SOA is considered idle as it provides the right framework.
- **Early-stage vs more mature organizations.** Businesses that are in their early stages might find MSA as an ideal choice. As the business grows, organizations may require capabilities such as complex request transformation and heterogeneous systems integration. In such situations, organizations often turn to the SOA pattern to replace MSA.

Both SOA and MSA follow an identical pattern of services at different layers of an enterprise. The existence of MSA comes down to the success of the SOA pattern and is therefore often referred as a subset of SOA.

While both Microservices and a Service-Oriented Architecture functions entirely on breaking an application into multiple services, an MSA disaggregates services on an application level, while an SOA does so on an enterprise-level service-reusability.

Related reading

- BMC DevOps Blog
- Microservices vs Nanoservices: Weighing Framework Options
- Microservices vs Serverless: What's The Difference?
- Challenges of Microservices & When To Avoid Them
- Kubernetes vs Docker Swarm: Comparing Container Orchestration Tools
- The State of Containers Today: A Report Summary

Original reference images:

SOA Architecture

Microservices Architecture