SERVICE-ORIENTED ARCHITECTURE AND MICROSERVICES

SERVICE-ORIENTED ARCHITECTURE

- Service-oriented architecture (SOA) is an enterprise-wide approach to software development that takes advantage of reusable software components, or services. Each service is comprised of the code and data integrations required to execute a specific business function—for example, checking a customer's credit, signing in to a website, or processing a mortgage application.
    - Gartner - a software architecture that starts with an interface definition and builds the entire application topology as a topology of interfaces, interface implementations, and interface calls.
    - IBM - an application framework that takes everyday business applications and breaks them down into individual business functions and processes, processes, called services.
    - Microsoft - a worldwide mesh of collaborating services that are published and available for invocation on a Service Bus.
    - BearingPoint - a software design & implementation approach ("Architecture") of loosely coupled, coarse-grained, reusable artifacts ("Services"), which can be integrated with each other, through a wide variety of platform-independent service interfaces.
- An architectural framework that enables us to:
    - Create reusable services that are highly interoperable through the use of broadly supported standards.
    - Support a new generation of agile composite applications assembled from business, application, and technical services.
    - Create external / business partner interfaces to streamline inter-enterprise integration, replacing EDI, EDI, VANs, etc.

Characteristics of SOA

- Standardized Service Contracts: Services adhere to a service description. Services use service contracts to:
    - Express their purpose
    - Express their capabilities
    - Use formal, standardized service contracts.
- Loose Coupling: Services minimize dependencies on each other and create specific types of relationships within and outside of service boundaries with a constant emphasis on reducing ("loosening") dependencies between service contract, service implementation, and service consumers.
- Abstraction
    - Services hide the logic they encapsulate from the outside world.
    - Avoids the proliferation of unnecessary service information, metadata, or data
    - Hides as much of the underlying details of a service as possible
    - Enables and preserves the loosely-coupled relationships.
    - Plays a significant role in the positioning and design of service compositions
- Service Reusability: Logic is divided into services with the intent of maximizing reuse.
- Autonomy
    - Services should have control over the logic they encapsulate.
    - Represents the ability of a service to carry out its logic independently of outside influences
    - To achieve this, services must be more isolated
    - Primary benefits
        - Increased reliability
        - Behavioral predictability
- Statelessness
    - Ideally, services should be stateless.
    - Incorporate state management deferral extensions within a service design goal.
    - Increase service scalability.
    - Support the design of agnostic logic and improve service reuse.
- Discoverability
    - Services can be discovered (usually in a service registry).

- Service contracts contain appropriate metadata for discovery which also communicates purpose and capabilities to humans.
- Store metadata in a service registry or profile documents.

- Composability
  - Services break big problems into little problems. This is elated to the Reusability principle.
  - Service execution should efficient in that individual processing should be highly tuned.
  - Flexible service contracts to allow different types of data exchange requirements for similar functions.
- Interoperability
  - Services should use standards that allow diverse subscribers to use the service.
  - This is considered so obvious these days that it is often dropped as a principle.

SOA Conceptual Model

SOA Architecture

- For SOA there are three important architectural perspectives: the application architecture, the service architecture, and the component architecture.
- These architectures can be viewed from either the consumer or provider perspective.
  - The key to the architecture is that the consumer of a service should not be interested in the implementation detail of the service—just the service provided.
  - The consumer is focused on their application architecture, the services used, but not the detail of the component architecture.

- Similarly, the provider is focused on the component architecture, the service architecture, but not on the application architecture.

Service Architecture

- At the core of the SOA is the need to be able to manage services as first-order deliverables.
- It is the service that is constantly emphasized which is the key to communication between the provider and consumer.
- Service architectures that ensure that services don't get reduced to the status of interfaces, rather they have an identity of their own and can be managed individually and in sets.

Business Service Bus

- The BSB is a logical view of the available and used services for a particular business domain, such as Human Resources or Logistics.
- The purpose of the BSB is so that common specifications, policies, etc. can be made at the bus level, rather than for each individual service.
- It also facilitates the implementation of a number of common, lower-level business infrastructure services that can be aggregated into other higher-level business services on the same bus.

SOA Platforms

- The key to separation is to define a virtual platform that is equally relevant to a number of real platforms.
- The objective of the virtual platform is to enable the separation of services from the implementation to be as complete as possible and allow components built on various implementation platforms to offer services that have no implementation dependency.
- The virtual SOA platform comprises a blueprint that covers the development and implementation platforms.
- The blueprint provides guidance on the development and implementation of applications to ensure that the published services conform to the same set of structural principles that are relevant to the management and consumer view of the services.
- When a number of different applications can all share the same structure, and where the relationships between the parts of the structure are the same, then have what might be called a common architectural style.
- Example platform components of a virtual platform include:

- Host environment, consumer environment, middleware, integration and assembly environment, development environment, asset management, publishing & discovery, service level management, security infrastructure, monitoring & measurement, diagnostics & failure, consumer/subscriber management, web service protocols, identity management, certification, and deployment & versioning

MICROSERVICES

- Like SOA, microservices architectures are made up of loosely coupled, reusable, and specialized components. However, rather than being adopted enterprise-wide, microservices are typically used to build individual applications in a way that makes them more agile, scalable, and resilient.
- Microservices are a true cloud-native architectural approach, and by using them, teams can update code more easily, use different stacks for different components, and scale the component independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.
- Microservices (or microservices architecture) use a cloud-native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components or services. These services typically:
  - have their own stack, inclusive of the database and data model;
  - communicate with one another over a combination of REST APIs, event streaming, and message brokers; and
  - are organized by business capability, with the line separating services often referred to as a bounded context.
- While much of the discussion about microservices has revolved around architectural definitions and characteristics, their value can be more commonly understood through fairly simple business and organizational benefits:
  - Code can be updated more easily.
  - Teams can use different stacks for different components.
  - Components can be scaled independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.
- Microservices might also be understood by what they are *not*. The two comparisons drawn most frequently with microservices architecture are monolithic architecture and service-oriented architecture (SOA).
- The difference between microservices and monolithic architecture is that microservices compose a single application from many smaller, loosely coupled services as opposed to the monolithic approach of a large, tightly coupled application

How microservices benefit the organization

Microservices are likely to be at least as popular with executives and project leaders as with developers. This is one of the more unusual characteristics of microservices because architectural enthusiasm is typically reserved for actual engineers. The reason for this is that microservices better reflect the way many business leaders want to structure and run their teams and development processes. Put another way, microservices are an architectural model that better facilitates a desired operational model.

- **Independently deployable**
  - Perhaps the single most important characteristic of microservices is that because the services are smaller and independently deployable Microservices promise organizations an antidote to the visceral frustrations associated with small changes taking huge amounts of time.
  - The microservices model fits neatly with this trend because it enables an organization to create small, cross-functional teams around one service or a collection of services and have them operate in an agile fashion.
  - Finally, the small size of the services, combined with their clear boundaries and communication patterns, makes it easier for new team members to understand the code base and contribute to it quickly—a clear benefit in terms of both speed and employee morale.

- **Right tool for the job**
  - In traditional n-tier architecture patterns, an application typically shares a common stack, with a large, relational database supporting the entire application. This approach has several obvious drawbacks—the most significant of which is that every component of an application must share a common stack, data model, and database, even if there is a clear, better tool for the job for certain elements.
  - By contrast, in a microservices model, components are deployed independently and communicate over some combination of event streaming, and message brokers—so it's possible for the stack of every individual service to be optimized for that service. Technology changes all the time, and an application composed of multiple, smaller services is much easier and less expensive to evolve with more desirable technology as it becomes available.

- **Precise scaling**
  - With microservices, individual services can be individually deployed—but they can be individually scaled, as well. The resulting benefit is obvious: Done correctly, microservices require less infrastructure than monolithic applications because they enable precise scaling of only the components that require it, instead of the entire application in the case of monolithic applications.

Microservices and cloud services

- Microservices are not necessarily exclusively relevant to cloud computing but there are a few important reasons why they so frequently go together—reasons that go beyond microservices being a popular architectural style for new applications and the cloud being a popular hosting destination for new applications.
- Among the primary benefits of the microservices architecture are the utilization and cost benefits associated with deploying and scaling components individually. While these benefits would still be present to some extent with on-premises infrastructure, the combination of small, independently scalable components coupled with on-demand, pay-per-use infrastructure is where real cost optimizations can be found.
- Secondly, and perhaps more importantly, another primary benefit of microservices is that each individual component can adopt the stack best suited to its specific job. Stack proliferation can lead to serious complexity and overhead when you manage it yourself but consuming the supporting stack as cloud services can dramatically minimize management challenges. Put another way, while it's not impossible to roll your own microservices infrastructure, it's not advisable, especially when just starting out.

THE MAIN DIFFERENCE BETWEEN SOA AND MICROSERVICES: SCOPE

- SOA was an enterprise-wide effort to standardize the way all services talk to and integrate with each other, whereas microservices architecture is application-specific.
- The main distinction between the two approaches comes down to *scope*. To put it simply, the service-oriented architecture (SOA) has an enterprise scope, while the microservices architecture has an application scope (Figure 1).

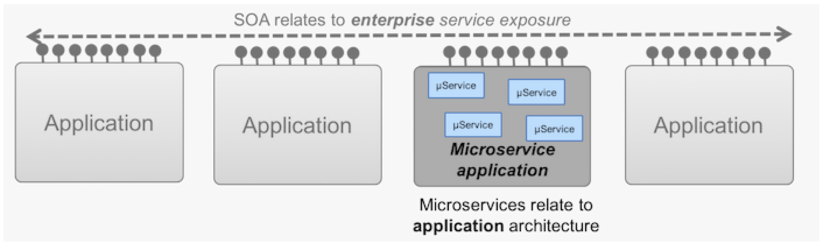
SOA relates to enterprise service exposure
Application
Application
µService
µService
µService
µService
Microservice application
Application
Microservices relate to application architecture

Figure 1. Distinction between SOA and microservices

Experts have filled a few thousands of print and digital pages comparing SOA and microservices and defining the subtleties of their relationship to one another. For the purposes of this article, the chief differences between the two are the coupling of components and scope of use:

- SOA is an enterprise-wide concept. It enables existing applications to be exposed over loosely coupled interfaces, each corresponding to a business function, that enables applications in one part of an extended enterprise to reuse functionality in other applications.

- Microservices architecture is an application-scoped concept. It enables the internals of a single application to be broken up into small pieces that can be independently changed, scaled, and administered. It does not define how applications talk to one another—for that we are back to the enterprise scope of the service interfaces provided by SOA.
- Many of the core principles of each approach become incompatible when the differences are neglected. If the differences are accepted in the scope, it becomes obvious that the two can potentially complement each other, rather than compete.
- Here are a few cases where this distinction comes into play.
  - **Reuse**
    - In SOA, reuse of integrations is the primary goal, and at an enterprise level, striving for some level of reuse is essential.
    - In microservices architecture, creating a microservices component that is reused at runtime throughout an application results in dependencies that reduce agility and resilience. Microservices components generally prefer to reuse code by copy and accept data duplication to help improve decoupling.
  - **Synchronous calls**
    - The reusable services in SOA are available across the enterprise using predominantly synchronous protocols such as APIs.
    - However, within a microservice application, synchronous calls introduce real-time dependencies, resulting in a loss of resilience. It may also cause latency, which impacts performance. Within a microservices application, interaction patterns based on asynchronous communication are preferred, such as event sourcing, in which a publish/subscribe model is used to enable a microservices component to remain up to date on changes happening to the data in another component.
  - **Data duplication**
    - A clear aim of providing services in an SOA is for all applications to synchronously get hold of and make changes to data directly at its primary source, which reduces the need to maintain complex data synchronization patterns.
    - In microservices applications, each microservice ideally has local access to all the data it needs to ensure its independence from other microservices, and indeed from other applications, even if this means some duplication of data in other systems. Of course, this duplication adds complexity, so it must be balanced against the gains in agility and performance, but this is accepted as a reality of microservices design.

Other Key Differences between SOA And Microservices

- **Communication:** In a microservices architecture, each service is developed independently, with its own communication protocol. With SOA, each service must share a common communication mechanism called an enterprise service bus (ESB). The ESB can become a single point of failure for the whole enterprise, and if a single service slows down, the entire system can be affected.
- **Interoperability:** In the interest of keeping things simple, microservices use lightweight messaging protocols. SOAs are more open to heterogeneous messaging protocols.
- **Service granularity:** Microservices architectures are made up of highly specialized services, each of which is designed to do one thing very well. The services that make up SOAs, on the other hand, can range from small, specialized services to enterprise-wide services.
- **Speed:** By leveraging the advantages of sharing a common architecture, SOAs simplify development and troubleshooting. However, this also tends to make SOAs operate more slowly than microservices architectures, which minimize sharing in favor of duplication.

REFERENCES:

Fastrack IT Academy (2019). *SAP business one – Basic.* Fastrack IT Academy.

Lam, W. (2017).*Enterprise architecture and integration: Methods, implementation, and technologies.* Information Science Reference.

SOA vs. microservices: What's the difference? Retrieved from: https://www.ibm.com/cloud/blog/soa-vs-microservices.

SOA (Service-Oriented Architecture). Retrieved from: https://www.ibm.com/cloud/learn/soa.

Microservices. Retrieved from: https://www.ibm.com/cloud/learn/microservices#toc-key-enabli-iyKOMbRc.