Software Architectures

Monolithic vs SOA vs Microservices

41492 – Engenharia de Software, Nuno Sá Couto e Rafael Direito

October 2nd 2023

Agenda

01 Monolithic Architectures

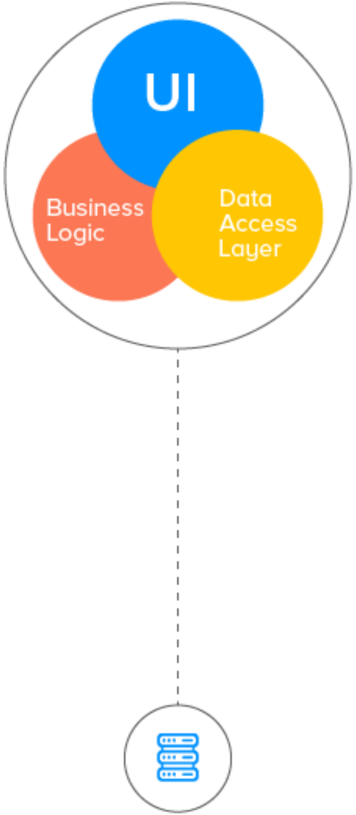
02 Service Oriented Architectures

03 Microservices

04 SOA vs Microservices

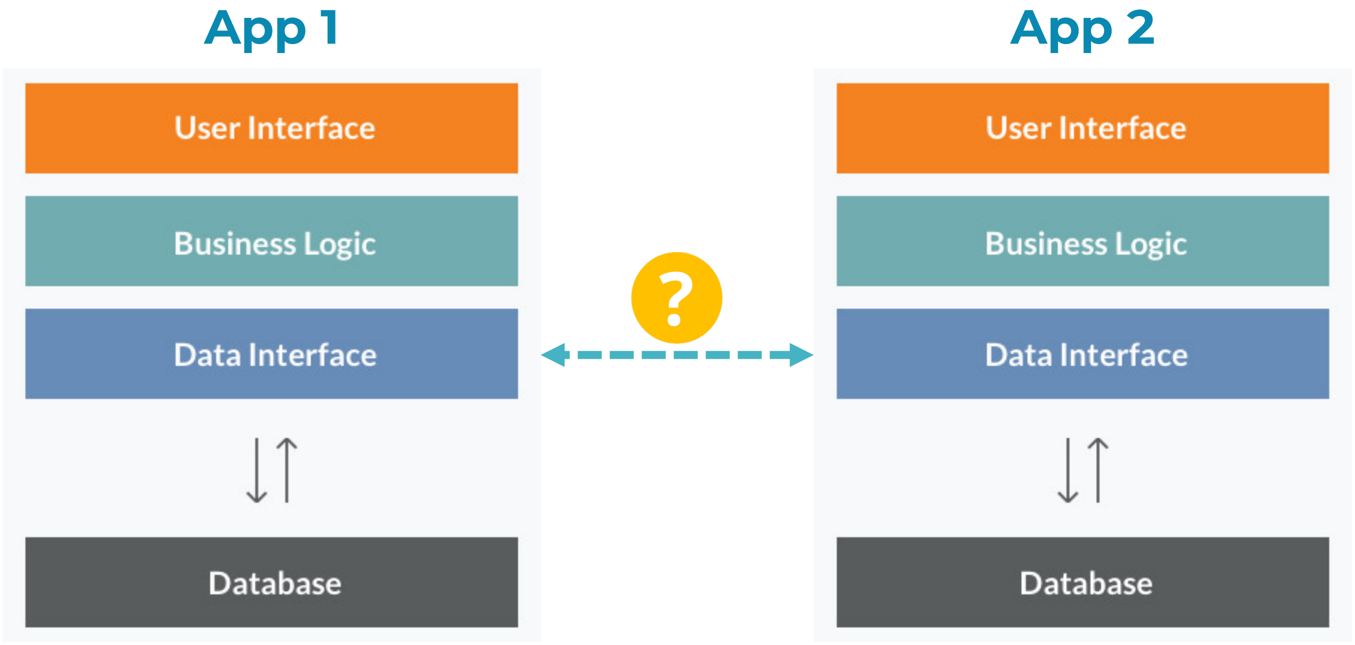
Monolithic Architecture

- The **“traditional” way** of building applications
- **One code base** couples all the business concerns together
- All software components are executed in a **single process**
- **No distribution** of resources
- **Strong coupling** between its components
  - To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface
- Usually **implemented as a Silo**

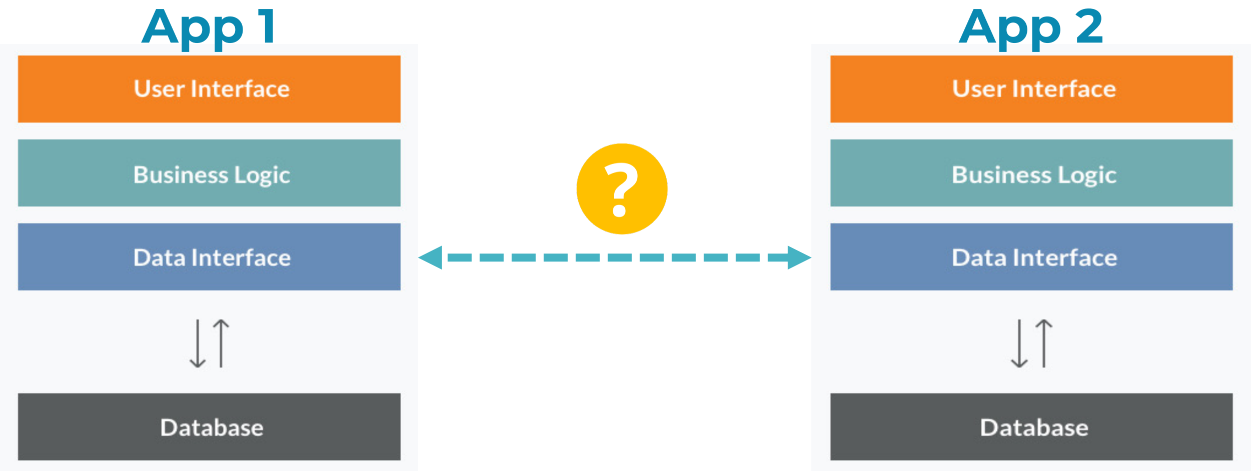


Monolithic Architecture

A **new requirement** is defined and the 2 apps have **to communicate with each other**.
How easy is it?



Monolithic Architecture



The **integration** between 2 monolithic applications **is very hard to achieve** and can be very **frustrating**.

Because the **applications were designed as independent and isolated silos** !

Monolithic Architecture

Pros?

Monolithic Architecture - Pros

Easier to Design

- It is easier to design an isolated and independent application
- **Easier debugging**
- **No queues nor messaging mechanisms**
- **No need to worry about integration** aspects

Performance

- Generally, it provides **better performance**
- **No need to have serialization/de-serialization** layers
- All **functionalities are in the same process**

Monolithic Architecture - Pros

Easier to Deploy

- Monolith applications are **packaged as a single artifact**, which makes them **easier to deploy**
- **We only have to deploy 1 artifact!**

Simplified Testing

- Since a monolith application is a single indivisible unit, **end-to-end testing is much faster**

Monolithic Architecture - Cons

Single Tech Platform

- **All the components must be developed using the same development platform**
- **Not always the best** for the task
- Can’t use specific platform for specific features
- **Future upgrade is a problem** – need to upgrade the whole app

Inflexible Deployment

- With monolith, **new deployment is always for the whole app**
- **No way to deploy only part of the app**
- Even when updating only one component – the whole codebase is deployed
- Forces rigorous testing for every deployment
- Forces long development cycles

Monolithic Architecture - Cons

Inefficient Compute Resources

- With monolith, compute **resources** (CPU and RAM) are **divided across all components**
- **If a specific component needs more resources – no way to do that**
- **Added resources will be made available to whole app**
- Very **inefficient**

Large and Complex

- With monolith, the **codebase is large and complex**
- Every little change can affect other components
- Testing not always detects all the bugs as full regressions are always required
- **Very difficult to maintain**
- Might make the system obsolete as developers will refrain changes in code as much as possible

Monolithic Architecture - Cons

Worse Reliability

- Since monolithic application are **packaged and deployed as a single** artifact, if there's **an error in any module, it will affect the entire application**
- If a specific module needs more resources, there's no way of guaranteeing that added resources will be available to that specific module. **The module might crash, and the application would become inoperable**

Barrier to Technology Adoption

- If we want to **use a different framework or language**, we would have to **update the entire application**
- **Making changes is often expensive and time-consuming**
- The application may become obsolete as **developers will refrain changes in code as much as possible**

From Monolithic to Microservices

In 2009 Netflix faced growing pains. Its infrastructure couldn’t keep up with the demand for its rapidly growing video streaming services*. The company* ***decided*** *to migrate its IT infrastructure from its private data centers to a public cloud and* ***replace its monolithic architecture with a microservices architecture. The only problem was, the term “microservices” didn’t exist and the structure wasn’t well-known.***

Netflix became one of the first high-profile companies to successfully migrate from a monolith to a cloud-based microservices architecture. It won the 2015 JAX Special Jury award in part due to this new infrastructure that internalized DevOps. Today, Netflix has more than a thousand microservices that manage and support separate parts of the platform, while its engineers deploy code frequently, sometimes thousands of times each day.

Netflix was an early pioneer in what has become increasingly common today: transitioning from a monolith architecture to a microservices architecture.

Source: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

From Monolithic to Microservices

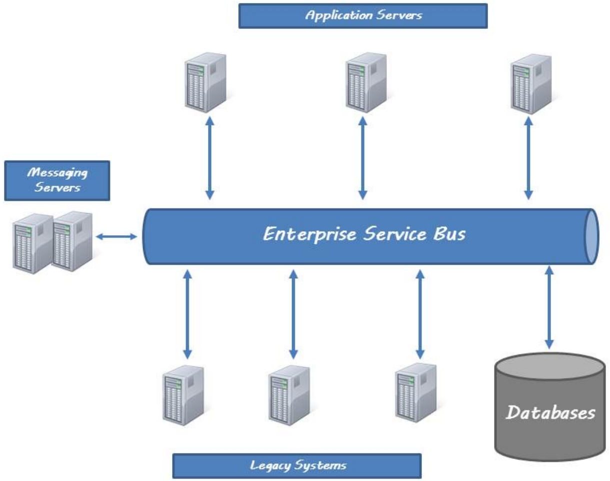
amazon.com
NETFLIX
GILT
UBER
PayPal
ebay
NORDSTROM
theguardian

But first we have to address the

Service Oriented Architecture

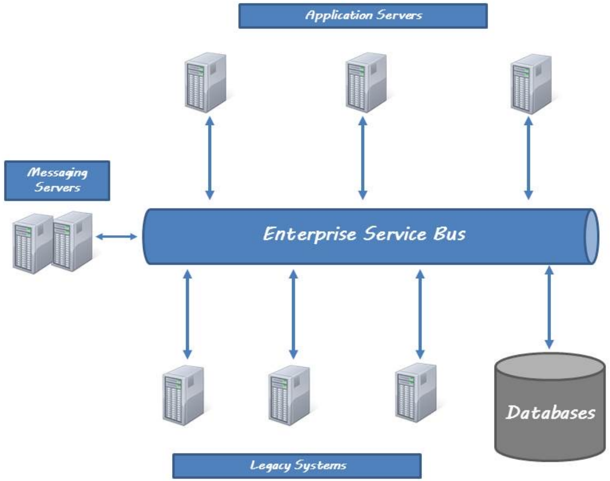
Service Oriented Architecture

- Relies on a series of **independently deployable Services**
- Each Service has its **own business logic**
- Separate tasks into **smaller processes (Services)**
- Each Service has its **own code base**
- Services **expose functionality to the other Services**



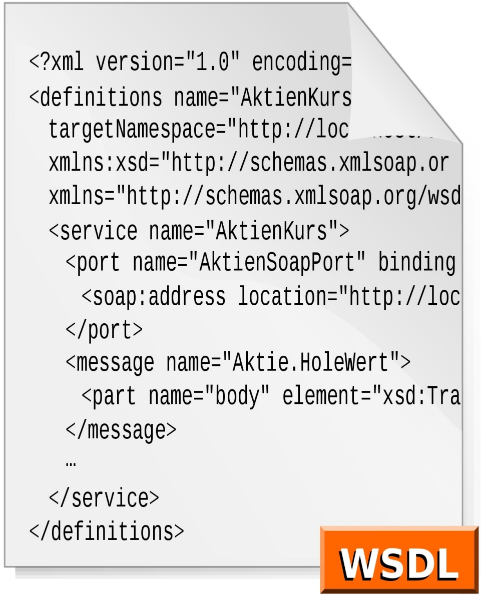
Service Oriented Architecture

- Usually, **Services communicate through an ESB** (Enterprise Service Bus)
- Usually, services register themselves on a **Service Registry**
- One of the most **popular Architecture Paradigms**
- **Not tied to specific technologies**



Service Oriented Architecture

- **Services expose metadata** to declare their functionality
- Usually implemented using **SOAP** (Simple Object Access Protocol) and **WSDL** (Web Services Description Language)
  - The first standards for Web Services
  - **The use of SOAP & WSDL was one of the main reasons for SOA failure**



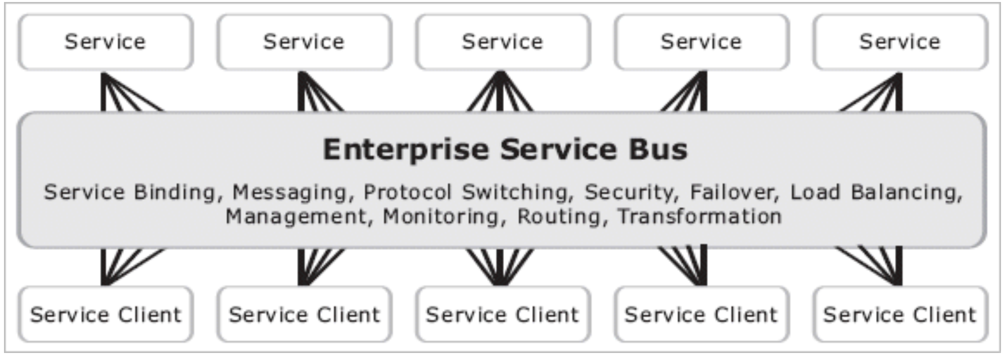
```
<?xml version="1.0" encoding=
<definitions name="AktienKurs
  targetNamespace="http://loc
  xmlns:xsd="http://schemas.xmlsoap.or
  xmlns="http://schemas.xmlsoap.org/wsd
  <service name="AktienKurs">
    <port name="AktienSoapPort" binding
      <soap:address location="http://loc
    </port>
    <message name="Aktie.HoleWert">
      <part name="body" element="xsd:Tra
    </message>
    …
  </service>
</definitions>
```

Service Oriented Architecture

- **Each Service consists of 3 components:**
  - The **interface**, which defines how a service provider will execute requests from a service consumer
  - The **contract**, which defines how the service provider and service consumer should interact
  - The **implementation**, which is the service code.

Service Oriented Architecture - ESB

- ESBs are **at the core of an SOA Architecture**
- ESBs provide a way to **decouple applications from each other**
- The ESB offers **a communication bus between all Services**, allowing **Services to talk to each other in a simple way**
- The **messages** travelling in the ESB are in a **canonical format** (e.g. XML)


Service
Service
Service
Service
Service
Enterprise Service Bus
Service Binding, Messaging, Protocol Switching, Security, Failover, Load Balancing, Management, Monitoring, Routing, Transformation
Service Client
Service Client
Service Client
Service Client
Service Client

Service Oriented Architecture - ESB

- **Clients don’t talk directly to the applications**, but rather to the ESB (through an intermediary. E.g.: the UI). The **ESB will then communicate with the applications.**

- **ESBs provide all the cross-cutting concerns of an SOA:**
    - Authorization
    - Authentication
    - Routing
    - Validation
    - Monitoring
    - Etc…

Service Oriented Architecture – Some Pros

Sharing Data and Functionality

- Monolith are silos: **not an easy task to create and expose services**
- With SOA, you just need:
  - To have access to the **WSDL document**
  - **Find out how their web methods are constructed**
  - **Construct your own clien**t for that method
- And voilá…. And there is a lot of tools that automates the generation of a good-looking client construct

Service Oriented Architecture – Some Pros

Polyglot Between Services

- **Avoids platform dependency**
- **Communication** between Services is done using **standard protocols**
- The **programming language** that the solution uses is **not relevant**
- The **underlying platforms are not relevant** for service execution

Service Oriented Architecture – Some Cons

Complicated and Expensive ESB

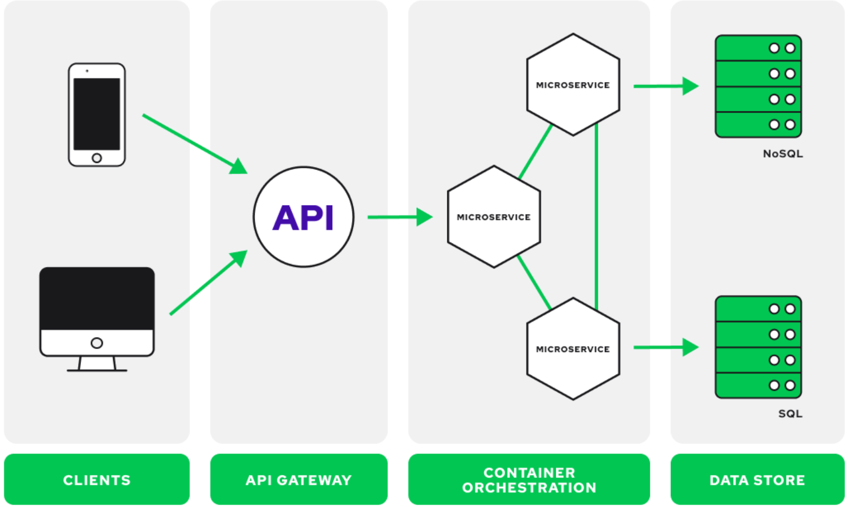
- The **ESB** can quickly become **bloated and expensive**
- It is very **difficult to maintain** the ESB due to the inherent complexity
- ESB complexity leads to **a lot of investment, time and money to maintain monstrous ESB instead of enjoying a lightweight and fast service oriented architecture**

Lack of Tooling

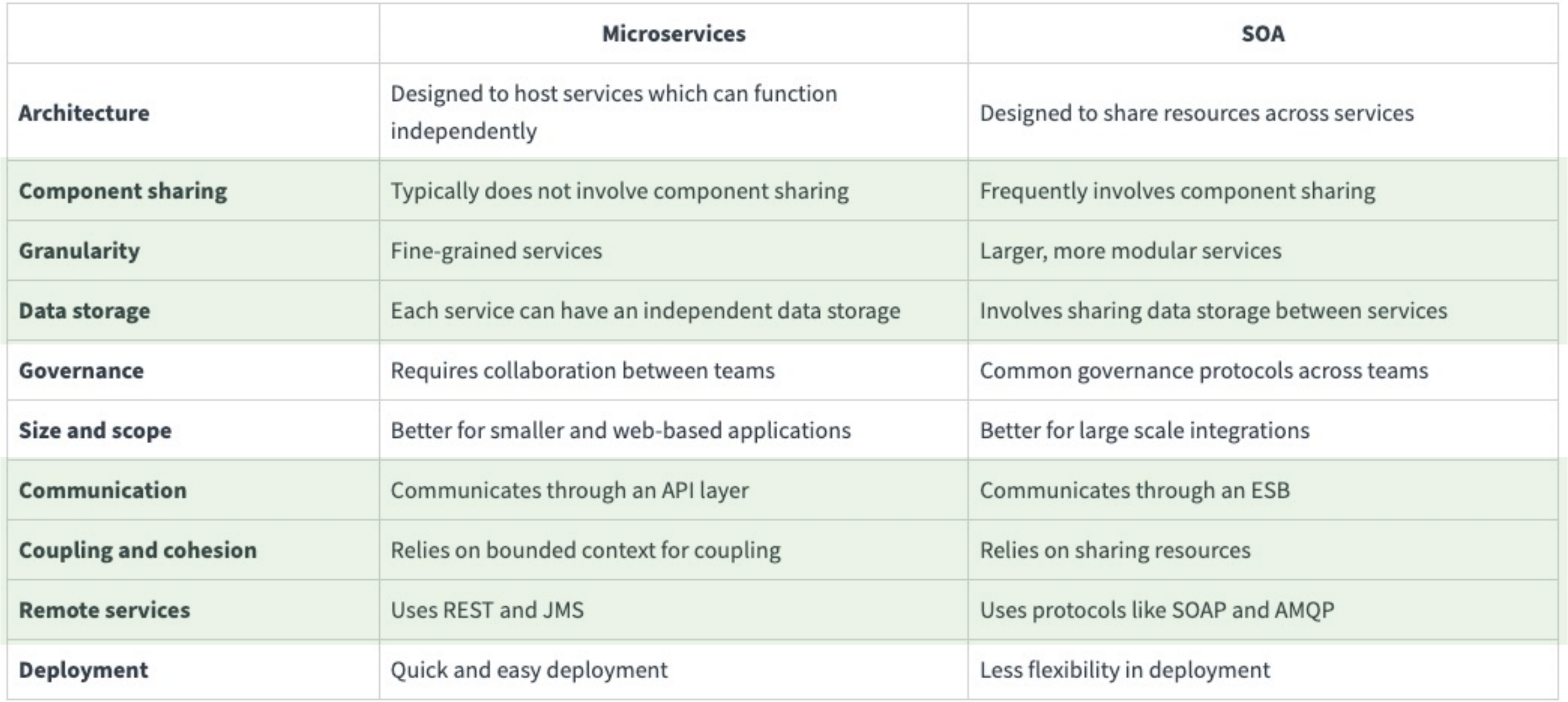
- For SOA to be effective, **short development cycles were needed**
- **SOA testing is much more complicated than Monolith testing**
- No tooling existed to support this testing, so mainly **manual testing is performed, leading to long test cycles..**
- **No time saving was achieved**

Microservices

- Introduced in 2014 by Martin Fowler and James Lewis
- **Adopted a lot of SOA's principles**
- But… these services must be **highly modular and with a simple API!**
- Can **rely on an API Gateway to hide the complexity of all the Services in a system**. The API Gateway is the single point of entry for the system
- Services are **loosely coupled**. Each Service has a specific purpose and should live independently of the other Services



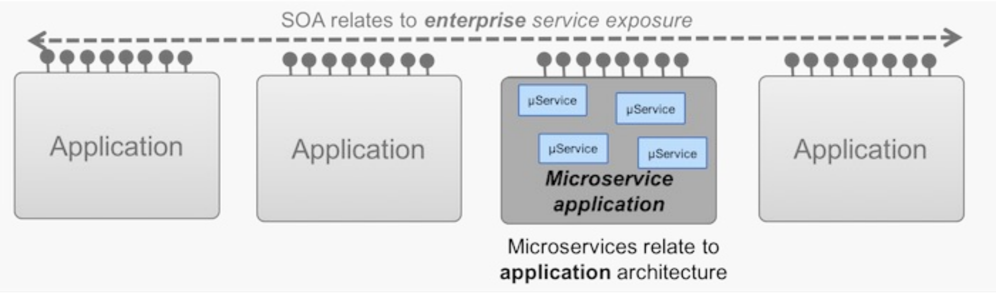
Microservices vs SOA



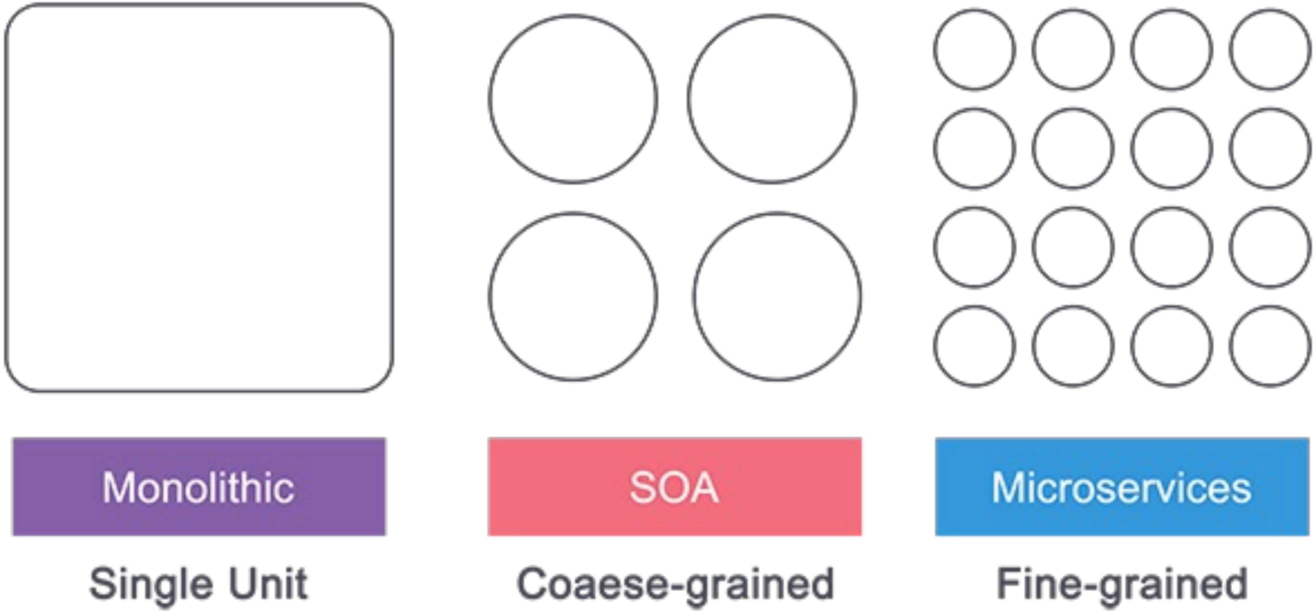
	Microservices	SOA
Architecture	Designed to host services which can function independently	Designed to share resources across services
Component sharing	Typically does not involve component sharing	Frequently involves component sharing
Granularity	Fine-grained services	Larger, more modular services
Data 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

Microservices vs SOA – Main Difference

The **main distinction between the two approaches comes down to *scope***. To put it simply, service-oriented architecture **(SOA) has an enterprise scope**, while the **microservices architecture has an application scope**.


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

Many of the core principles of each approach become incompatible when you neglect this difference. If you accept the difference in scope, you may quickly realize that the two can potentially complement each other, rather than compete.



Monolithic

Single Unit

SOA

Coaese-grained

Microservices

Fine-grained

Microservices - Pros

Agility

- Promote agile ways of working with small teams, resulting in frequent deployments and releases

Flexible Scaling

- It is possible to independently scale a single Service

Continuous Deployment

- Enable frequent and faster release cycles

Maintainability

- A change in a Service won't directly result in changes in the other Services

Testability

- It is easier to test smaller Services with a unique purpose

Technology Flexibility

- Different Services can be implemented in different ways, with different frameworks and languages

Microservices - Pros

Reliability

- A Service can be updated without the threat of bringing down the entire application

Happier Teams

- Teams are more autonomous, not having to rely on the teams developing the other modules

Independent Deployments

- A Service can be deployed independently

Data Sharing

- Services can easily change data between one another. They are not silos!

Microservices

Cons?

Microservices - Cons

Distributed Development

- If development sprawl isn’t properly managed, it results in slower development speed and poor operational performance

Infrastructure Costs

- Each new microservice can have **its own cost for test suite, deployment and hosting infrastructure**

Organization Overhead

- Teams need to add **another level of communication and collaboration to coordinate updates and interfaces**

Debugging Complexity

- Each microservice **has its own set of logs**, which makes **debugging more complicated**. Besides that, there are **replicated services**

Lack of Standardization

- Without a common platform, there can be a proliferation of languages, logging standards, and monitoring

Lack of Clear Ownership

- Sometimes it is **difficult to know which service should support a new operation**

Different Approaches for Different Problems

Ultimately, **all these 3 architectures are valid**

You just have **to take into account what is the problem at hand and use the best approach do solve it!**

Recap Quiz

Single and extensive code base	Monolithic
No platform dependency	SOA and Micro
Usually relies on REST based communication	Microservices
There are several Services sharing the same database	SOA (and Micro)
Applications are implemented as silos	Monolithic
Services are loosely coupled	Microservices
Service communication is usually achieved using an ESB	SOA
Service communication relies on SOAP and WSDL	SOA
Application is deployed as a single package	Monolithic
Services live completely independently of the other Services	Microservices
Each Service is composed of 3 components (interface, contract and implementation)	SOA
Fine-grained and independent Services	Microservices