Contents

[What are microservices ? 2](#_Toc56087203)

[Small, and Focused on Doing One Thing Well 2](#_Toc56087204)

[Autonomous 2](#_Toc56087205)

[Technology Heterogeneity 2](#_Toc56087206)

[Resilience 2](#_Toc56087207)

[Scaling 2](#_Toc56087208)

[Ease of Deployment 2](#_Toc56087209)

[Organizational Alignment 2](#_Toc56087210)

[Composability 2](#_Toc56087211)

[Optimizing for Replaceability 2](#_Toc56087212)

[What About Service-Oriented Architecture ? 2](#_Toc56087213)

[Other Decompositional Techniques 2](#_Toc56087214)

[Shared Libraries 2](#_Toc56087215)

[Modules 2](#_Toc56087216)

[No Silver Bullet 2](#_Toc56087217)

Microservices

# What are microservices ?

Microservices are small, autonomous services that work together.

## Small, and Focused on Doing One Thing Well

Cohesion – the drive to have related code grouped together – is an important concept when we think about microservices.

“Gather together those things that change for the same reason, and separate those things that change for different reasons.” – Robert C. Martin’s definition of Single Responsibility Principle.

We focus our service boundaries on business boundaries, making it obvious where code lives for a given piece of functionality.

Jon Eaves at *RealState.com.au* in Australia characterizes a microservice as something that could be rewritten in two weeks, a rule of thumb.

“the smaller the service, the more you maximize the benefits and downsides of microservices architecture.”

As you get smaller, the benefits around interdependence increase.

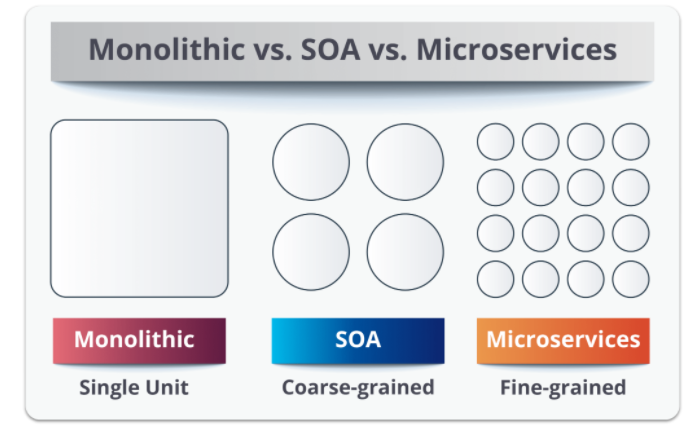
## Autonomous

Our microservice is a separate entity. It might be deployed as an isolated service on platform as a service (PAAS), it might be its own operating system process.

We try to avoid packing multiple services onto the same machine, although this isolation can add some overhead, as IPC will be over network to enforce separation between the services and avoid the perils of tight coupling.  
If there is too much sharing, our consuming services become coupled to our internal representations.   
This decreases our autonomy, as it requires additional coordination with consumers when making changes.

The golden rule: “Can you make a change to a service and deploy it by itself without changing anything else?”

# Microservices vs SOA

In layman’s terms, a **monolith** is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.

## Service-Oriented Architecture

**Service-oriented architecture** is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or some activity coordination between two or more services. Some means of connecting services to each other is needed.

**SOA** defines 4 four basic service types:

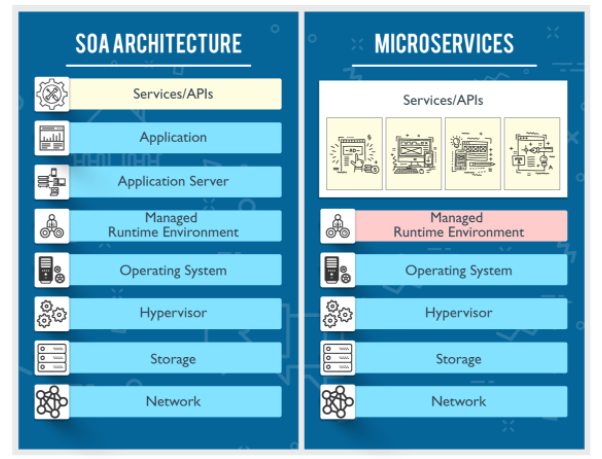
1. Functional/Business Services:
   1. This is related to *business users*.
   2. Coarse-grained services that define core business operations.
   3. Represented through XML, BPEL (Business Process Execution Language) and others.
2. Enterprise Services:
   1. This is related to *shared services team*.
   2. Implement the functionality defined by business services.
   3. Mainly rely on application services and infrastructure services to fulfil business requests.
3. Application Services:
   1. This is related to *application development team*.
   2. Fine-grained services that are confined to a specific application context.
   3. A dedicated user interface can directly invoke the services.
4. Infrastructure Services:
   1. This is related to *infrastructure services team*.
   2. Implement non-functional tasks such as authentication, auditing, security, and logging.
   3. Can be invoked from either application services or enterprise services.

## Microservices

**Microservices**, aka *microservice architecture*, is an architectural style that structures an application as a collection of small autonomous services modelled around a business domain.

Microservices have limited service taxonomy. They consist of two service types:

1. *Functional services* support specific business operations.
2. Accessing of services is done externally and these services are not shared with other services.
3. As in SOA, *infrastructure services* implement tasks such as authentication, auditing, security, logging.
4. In this, the services are not unveiled to the outside world.



## Major Differences

|  |  |  |
| --- | --- | --- |
| SOA | MSA |  |
| Follows “share-as-much-as-possible” architecture approach | Follows “share-as-little-as-possible” architecture approach | Sharing Granularity |
| Importance is on business functionality reuse | Importance is on the concept of “bounded context” | Component Sharing |
| They have common governance and standards | They focus on people, collaboration and freedom of other options |
| Uses Enterprise Service bus (ESB) for communication | Simple messaging system | Middleware vs API Layer |
| They support multiple message protocols | They use lightweight protocols such as HTTP/REST etc. | Remote Services |
| Multi-threaded with more overheads to handle I/O | Single-threaded usually with the use of Event Loop features for non-locking I/O handling |  |
| Maximizes application service reusability | Focuses on decoupling | Heterogenous Interoperability |
| Traditional Relational Databases are more often used | Modern Relational Databases are more often used |  |
| A systematic change requires modifying the monolith | A systematic change is to create a new service |  |
| DevOps / Continuous Delivery is becoming popular, but not yet mainstream | Strong focus on DevOps / Continuous Delivery |  |

# Key Benefits

Many of the benefits from microservices can be laid at the door of any distributed system. Microservices, however, tend to achieve these benefits to a greater degree primarily due to how far take the concepts behind the distributed systems and service-oriented architecture.

## Technology Heterogeneity

## Resilience

## Scaling

## Ease of Deployment

## Organizational Alignment

## Composability

## Optimizing for Replaceability

# What About Service-Oriented Architecture ?

## Other Decompositional Techniques

## Shared Libraries

## Modules

## No Silver Bullet