# Microservices – Definition, Principles and Benefits

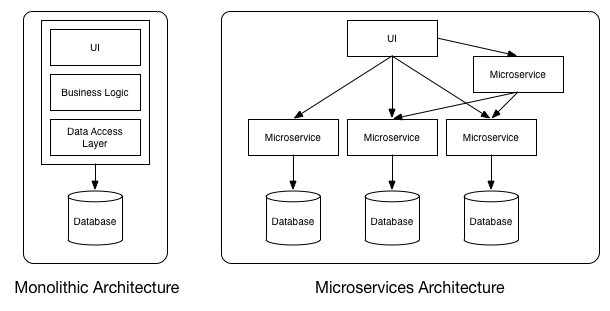
Microservices is latest buzzword in industry and everyone seems to be talking about it, in one way or another. Let’s understand **what are microservices**? In this tutorial, we will try to understand the definition, concepts and **principles of microservices**.

## Definition of Microservices

Today, microservices are one of the increasingly popular architecture patterns next to SOA (Services Oriented Architecture). If you are following industry trends, then you realize that today business houses are no longer interested in developing large applications to manage their end-to-end business functions as they did a few years ago, rather they opt for quick and agile applications which cost them less money as well.

Microservices help in breaking the boundaries of large applications and build logically independent smaller systems inside system. E.g. using [Amazon AWS](https://aws.amazon.com/) you can build a cloud application with minimum effort. It’s a good example of what microservices can do.



Monolithic vs MicroServices Architecture

As you can see in above diagram, each microservice has it’s own business layer and database. By doing so, changes to one microservice do not impact others.

In general, microservices communicate with each other using widely adopted lightweight protocols, such as HTTP and REST, or messaging protocols, such as JMS or AMQP. In specific scenarios, they can go for more specialized protocols as well.

## Principles of Microservices

Now let’s examine “must have” principles of a microservice.

#### Single responsibility principle

The single responsibility principle is one of the principles defined as part of the [SOLID design pattern](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#SRP). It implies that a unit, either a class, a function, or a microservice, should have one and only one responsibility.

At no point of time, one microservice should have more than one responsibility.

#### Built around business capabilities

**Microservices should focus on certain business function** and ensure that it helps in getting things done. A microservice shall never restrict itself from adopting appropriate technology stack or backend database storage which is most suitable for solving the business purpose.

This is often the constraint when we design monolithic applications where we try to solve multiple business solutions with some compromises in some areas. Microservices enable you to choose whats best for the problem in hand.

#### You build it, you own it!

Another important aspect of such design is related to responsibilities pre-and-post development. In large organization, usually one team develops the app location, and after some knowledge transfer sessions it hand over the project to maintenance team. In microservices, the team which build the service – owns it, and is responsible for maintaining it in future.

[You build it, you own it !!](https://aronatkins.github.io/2014/12/23/you-build-it-you-own-it.html)

This brings developers into contact with the day-to-day operation of their software and they better understand how their built product is used by customers in real world.

#### Infrastructure Automation

Preparing and building infrastructure for microservices is another very important need. **A service shall be independently deployable** and shall bundle all dependencies, including library dependencies, and even execution environments such as web servers and containers or virtual machines that abstract physical resources.

One of the major **differences between microservices and SOA** is in their level of autonomy. While most SOA implementations provide service-level abstraction, microservices go further and abstract the realization and execution environment.

In traditional application developments, we build a WAR or an EAR, then deploy it into a JEE application server, such as with JBoss, WebLogic, WebSphere, and so on. We may deploy multiple applications into the same JEE container. In ideal scenario, in the microservices approach, each microservice will be built as a [fat Jar](https://howtodoinjava.com/maven/maven-shade-plugin-create-uberfat-jar-example/), embedding all dependencies and run as a standalone Java process.

#### Design for Failure

A microservice shall be designed with failure cases in mind. What if the service fails, or go down for some time. These are very important questions and must be solved before actual coding starts – to clearly estimate **how service failures will affect the user experience**.

Fail fast is another concept used to build fault-tolerant, resilient systems. This philosophy advocates systems that expect failures versus building systems that never fail. Since services can fail at any time, it’s important to be able to detect the failures quickly and, if possible, automatically restore service.

Microservice applications put a lot of emphasis on real-time monitoring of the application, checking both architectural elements (how many requests per second is the database getting) and business relevant metrics (such as how many orders per minute are received). Semantic monitoring can provide an early warning system of something going wrong that triggers development teams to follow up and investigate.

## Benefits of Microservices

Microservices offer a number of benefits over the traditional multi-tier, monolithic architectures. Let’s list down them:

* With microservices, architects and developers can choose fit for purpose architectures and technologies for each microservice ([polyglot architecture](https://www.infoq.com/articles/paradigm-based-polyglot-prog)). This gives the flexibility to design better-fit solutions in a more cost-effective way.
* As services are fairly simple and smaller in size, enterprises can afford to experiment new processes, algorithms, business logic, and so on. It enables enterprises to do disruptive innovation by offering the ability to experiment and fail fast.
* Microservices enable to implement selective scalability i.e. each service could be independently scaled up or down and cost of scaling is comparatively less than monolithic approach.
* Microservices are self-contained, independent deployment modules enabling the substitution of one microservice with another similar microservice, when second one is not performing as per our need. It helps in taking right buy-versus-build decisions which are often the challenge for many enterprises.
* Microservices help us build systems that are organic in nature(Organic systems are systems that grow laterally over a period of time by adding more and more functions to it). Because microservices are all about independently manageable services – it enable to add more and more services as the need arises with minimal impact on the existing services.
* Technology changes are one of the barriers in software development. With microservices, it is possible to change or upgrade technology for each service individually rather than upgrading an entire application.
* As microservices package the service runtime environment along with the service itself, this enables having multiple versions of the service to coexist in the same environment.
* And finally, microservices also enable smaller, focused agile teams for development. Teams will be organized based on the boundaries of microservices.

Microservice Architecture - Introduction

Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units. Microservice is the process of implementing Service-oriented Architecture (SOA) by dividing the entire application as a collection of interconnected services, where each service will serve only one business need.

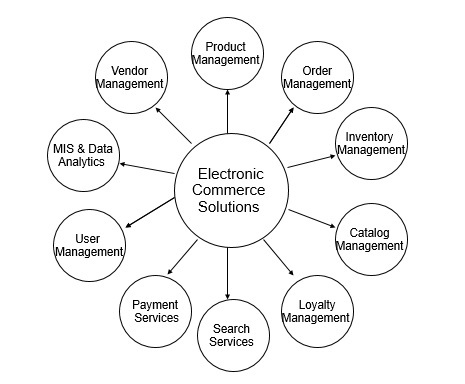
The Concept of Going Micro

In a service-oriented architecture, entire software packages will be sub-divided into small, interconnected business units. Each of these small business units will communicate to each other using different protocols to deliver successful business to the client. Now the question is, how Microservice Architecture (MSA) differs from SOA? In one word, SOA is a designing pattern and Microservice is an implementation methodology to implement SOA or we can say Microservice is a type of SOA.

Following are some rules that we need to keep in mind while developing a Microservice-oriented application.

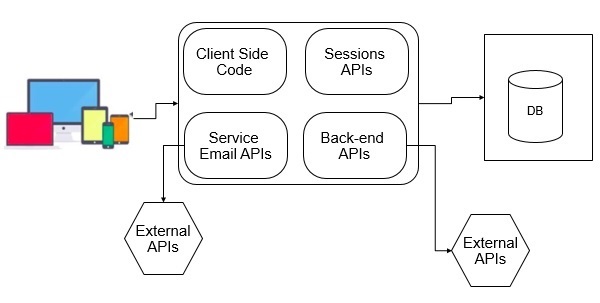
* **Independent** − Each microservice should be independently deployable.
* **Coupling** − All microservices should be loosely coupled with one another such that changes in one will not affect the other.
* **Business Goal** − Each service unit of the entire application should be the smallest and capable of delivering one specific business goal.

Let us consider an example of online shopping portal to understand microservice in depth. Now, let us break this entire E-commerce portal into small business units such as user management, order management, check-in, payment management, delivery management, etc. One successful order needs to proceed through all of these modules within a specific time frame. Following is the consolidated image of different business units associated with one electronic commerce system.



Each of these business modules should have its own business logic and stakeholders. They communicate with other third party vendor softwares for some specific needs, and also with each other. For example, order management may communicate with user management to get user information.

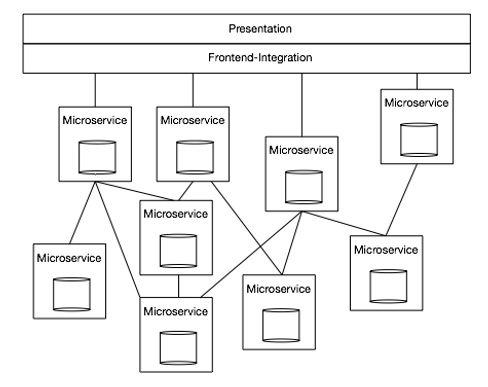
Now, considering you are running an online shopping portal with all of these business units mentioned earlier, you do need some enterprise level application consisting of different layers such as front-end, back-end, database, etc. If your application is not scaled and completely developed in one single war file, then it will be called as a typical monolithic application. According to IBM, a typical monolithic application should possess the following module structure internally where only one endpoint or application will be responsible to handle all user requests.



In the above image, you can see different modules such as Database for storing different users and business data. At the front-end, we have different device where we usually render user or business data to use. In the middle, we have one package that can be a deployable EAR or WAR file that accepts request form the users end, processes it with the help of the resources, and renders it back to the users. Everything will be fine until business wants any changes in the above example.

Consider the following scenarios where you have to change your application according to the business needs.

Business unit needs some changes in the “Search” module. Then, you need to change the entire search process and redeploy your application. In that case, you are redeploying your other units without any changes at all.



Now again your business unit needs some changes in “Check out” module to include “wallet” option. You now have to change your “Check out” module and redeploy the same into the server. Note, you are redeploying the different modules of your software packages, whereas we have not made any changes to it. Here comes the concept of service-oriented architecture more specific to Microservice architecture. We can develop our monolithic application in such a manner that each and every module of the software will behave as an independent unit, capable of handling a single business task independently.

Consider the following example.

In the above architecture, we are not creating any ear file with compact end-to-end service. Instead, we are dividing different parts of the software by exposing them as a service. Any part of the software can easily communicate with each other by consuming respective services. That's how microservice plays a great role in modern web application.

Let us compare our shopping cart example in the line of microservice. We can break down our shopping cart in the different modules such as “Search”, ”Filter”, “Checkout”, “Cart”, “Recommendation”, etc. If we want to build a shopping cart portal then we have to build the above-mentioned modules in such a manner that they can connect to each other to give you a 24x7 good shopping experience.

Advantages & Disadvantages

Following are some points on the advantages of using microservice instead of using a monolithic application.

Advantages

* **Small in size** − Microservices is an implementation of SOA design pattern. It is recommended to keep your service as much as you can. Basically, a service should not perform more than one business task, hence it will be obviously small in size and easy to maintain than any other monolithic application.
* **Focused** − As mentioned earlier, each microservice is designed to deliver only one business task. While designing a microservice, the architect should be concerned about the focal point of the service, which is its deliverable. By definition, one microservice should be full stack in nature and should be committed to delivering only one business property.
* **Autonomous** − Each microservice should be an autonomous business unit of the entire application. Hence, the application becomes more loosely coupled, which helps to reduce the maintenance cost.
* **Technology heterogeneity** − Microservice supports different technologies to communicate with each other in one business unit, which helps the developers to use the correct technology at the correct place. By implementing a heterogeneous system, one can obtain maximum security, speed and a scalable system.
* **Resilience** − Resilience is a property of isolating a software unit. Microservice follows high level of resilience in building methodology, hence whenever one unit fails it does not impact the entire business. Resilience is another property which implements highly scalable and less coupled system.
* **Ease of deployment** − As the entire application is sub-divided into small piece of units, every component should be full stack in nature. All of them can be deployed in any environment very easily with less time complexity unlike other monolithic applications of the same kind.

Following are some points on the disadvantages of microservice architecture.

Disadvantages

* **Distributed system** − Due to technical heterogeneity, different technologies will be used to develop different parts of a microservice. A huge set of skilled professionals are required to support this big heterogeneous distributed software. Hence, distributed and heterogeneity stands as a number one disadvantage of using microservice.
* **Cost** − Microservice is costly, as you have to maintain different server space for different business tasks.
* **Enterprise readiness** − Microservice architecture can be considered as a conglomerate of different technologies, as technology is evolving day-by-day. Hence, it is quite difficult to make a microservice application enterprise ready to compare to conventional software development model.

Microservice Over SOA

The following table lists certain features of SOA and Microservice, bringing out the importance of using microservice over SOA.

|  |  |  |
| --- | --- | --- |
| **Component** | **SOA** | **Microservice** |
| Design pattern | SOA is a design paradigm for computer software, where software components are exposed to the outer world for usage in the form of services. | Micro Service is a part of SOA. It is a specialized implementation of SOA. |
| Dependency | Business units are dependent on each other. | All business units are independent of each other. |
| Size | Software size is bigger than the conventional software. | Software size is small. |
| Technology | Technology stack is less than Microservice. | Microservice is heterogeneous in nature as exact technologies are used to perform a specific task. Microservices can be considered as a conglomerate of many technologies. |
| Autonomous and Focus | SOA applications are built to perform multiple business tasks. | Microservice applications are built to perform a single business task. |
| Nature | Monolithic in nature. | Full stack in nature. |
| Deployment | Deployment is time-consuming. | Deployment is very easy. Hence, it will be less time-consuming. |
| Cost-effectiveness | More cost-effective. | Less cost-effective. |
| Scalability | Less compared to Microservices. | Fully scaled. |
| Example | Let us consider one online CAB booking application. If we want to build that application using SOA, then its software units will be −   * GetPayments And DriverInformation And MappingDataAPI * AuthenticateUsersAnd DriversAPI | If the same application is built using microservice architecture, then its APIs will be −   * SubmitPaymentsService * GetDriverInfoService * GetMappingDataService * AuthenticateUserService * AuthenticateDriverService |