WHAT ARE MICROSERVICES?

Micro services - also known as the micro service architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The micro service architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack.

Micro services have many benefits for Agile and DevOps teams - as Martin Fowler [points out](http://martinfowler.com/articles/microservices.html), Netflix, eBay, Amazon, Twitter, PayPal, and other tech stars have all evolved from monolithic to micro services architecture. Unlike micro services, a monolith application is built as a single, autonomous unit. This make changes to the application slow as it affects the entire system.  A modification made to a small section of code might require building and deploying an entirely new version of software.  Scaling specific functions of an application, also means you have to scale the entire application.

Micro services solve these challenges of montholic systems by being as modular as possible. In the simplest form, they help build an application as a suite of small services, each running in its own process and are independently deployable. These services may be written in different languages and may use different data storage techniques. While this results in the development of systems that are scalable and flexible, it needs a dynamic makeover.

SIX CHARACTERISITICS OF MICROSERVICES

1. Multiple Components

Software built as micro services can, by definition, be broken down into multiple component services.  Why?  So that each of these services can be deployed, tweaked, and then redeployed independently without compromising the integrity of an application.  As a result, you might only need to change one or more distinct services instead of having to redeploy entire applications.

1. Built For Business

This is business oriented architecture means it develop on the base of needs

1. Simple Routing

Receive requests, process them, and generate a response accordingly. You could say that micro services have smart endpoints that process info and apply logic, and dumb pipes through which the info flows.

1. Decentralized

Micro service can work on both central database and decentralized database but it most innovative part is decentralized database. Each service can have its own database.

1. Failure Resistant

Like a well-rounded child, microservices are designed to cope with failure.  Since several unique and diverse services are communicating together, it’s quite possible that a service could fail, for one reason or another (e.g., when the supplier isn’t available).  In these instances, the client should allow its neighboring services to function while it bows out in as graceful a manner as possible. However, [monitoring microservices](https://smartbear.com/en-us/learn/performance-monitoring/monitoring-microservices/) can help prevent the risk of a failure. For obvious reasons, this requirement adds more complexity to microservices as compared to monolithic systems architecture.

1. Evolutionary

Microservices architecture is an evolutionary design and, again, is ideal for evolutionary systems where you can’t fully anticipate the types of devices that may one day be accessing your application

EXAMPLE OF MICROSERVICES

Netflix, amazon, ebay

PROS AND CONS OF MICROSERVICES

|  |  |
| --- | --- |
| PROS | CONS |
| Microservice architecture gives developers the freedom to independently develop and deploy services | Due to distributed deployment, testing can become complicated and tedious |
| A microservice can be developed by a fairly small team | Increasing number of services can result in information barriers |
| Code for different services can be written in different languages (though many practitioners discourage it) | The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing |
| Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.) | Being a distributed system, it can result in duplication of effort |
| Easy to understand and modify for developers, thus can help a new team member become productive quickly | When number of services increases, integration and managing whole products can become complicated |
| The developers can make use of the latest technologies | In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system |
| The code is organized around business capabilities | Developers have to put additional effort into implementing the mechanism of communication between the services |
| Starts the web container more quickly, so the deployment is also faster | Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams |
| When change is required in a certain part of the application, only the related service can be modified and redeployed- no need to modify and redeploy the entire application | The architecture usually results in increased memory consumption |
| Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system) | Partitioning the application into micro services is very much an art |
| Easy to scale and integrate with third-party services |  |
| No long-term commitment to technology stack |  |