## Microservices

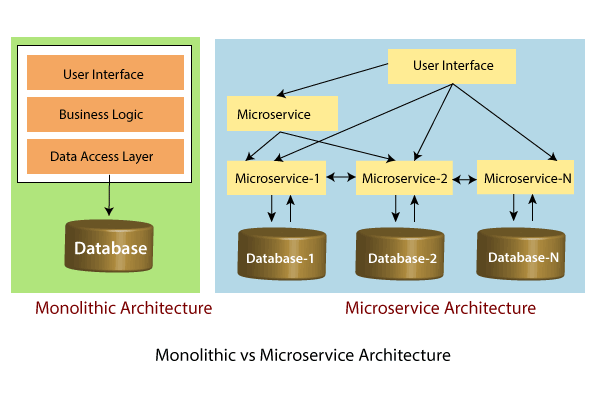
**Microservice Architecture** is a Service Oriented Architecture.

"Microservices are the small services that work together."

## Points to remember

* These are the services which are exposed by REST.
* These are small well-chosen deployable units.
* The services must be cloud-enabled.

The most important feature of the microservice-based architecture is that it can perform **continuous delivery** of a large and complex application.



In the above figure, each microservice has its own business layer and database. If we change in one microservice, it does not affect the other services. These services communicate with each other by using lightweight protocols such as HTTP or REST or messaging protocols.

## Principles of Microservices

There are the following principles of Microservices:

* Single Responsibility principle
* Modelled around business domain
* Isolate Failure
* Infrastructure automation
* Deploy independently

### Single Responsibility Principle

The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

### Modeled around business domain

Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

### Isolated Failure

The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

### Infrastructure Automation

Infrastructure automation is the process of scripting environments. With the help of a scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

### Deploy independently

Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other services.

# Advantages of Microservices

* Microservices are self-contained, independent deployment modules.
* The cost of scaling is comparatively less than the monolithic architecture.
* Microservices are independently manageable services. It can enable more and more services as the need arises. It minimizes the impact on existing service.
* It is possible to change or upgrade each service individually rather than upgrading in the entire application.
* Microservices allows us to develop an application which is organic (an application which later upgrades by adding more functions or modules) in nature.
* It enables event streaming technology to enable easy integration in comparison to heavyweight interposed communication.
* Microservices follow the single responsibility principle.
* The demanding service can be deployed on multiple servers to enhance performance.
* Less dependency and easy to test.
* Dynamic scaling.
* Faster release cycle.

## Disadvantages of Microservices

* Microservices have all the associated complexities of the distributed system.
* There is a higher chance of failure during communication between different services.
* Difficult to manage a large number of services.
* The developer needs to solve the problem, such as network latency and load balancing.
* Complex testing over a distributed environment.

Difference between Microservices Architecture (MSA) & Services-Oriented Architecture(SOA)

| **Microservice Based Architecture (MSA)** | **Service-Oriented Architecture (SOA)** |
| --- | --- |
| Microservices uses **lightweight protocols** such as **REST**, **HTTP**. | SOA supports **multi-message protocols**. |
| It focuses on **decoupling**. | It focuses on application service **reusability**. |
| It uses a **simple messaging system** for communication. | It uses **Enterprise Service Bus** (ESB) for communication. |
| Microservices follows a "**share as little as possible**" architecture approach. | SOA follows the "**share as much as possible architecture**" approach. |
| Microservices are much better in **fault tolerance** in comparison to SOA. | SOA is not better in fault tolerance in comparison to MSA. |
| Each microservice has an **independent** database. | SOA services share the **whole** data storage. |
| MSA used **modern** relational databases. | SOA used **traditional** relational databases. |
| It is better suited for the **smaller** and **well portioned**, web-based system. | It is better for a **large** and **complex** business application environment. |

# Microservices Monitoring

Monitoring is the control system of the microservices. As the microservices are more complex and harder to understand its performance and troubleshoot the problems. Given the vivid changes to software delivery, it is required to monitor the service. There are **five** principles of monitoring microservices, as follows:

* Monitor container and what's inside them.
* Alert on service performance.
* Monitor services that are elastic and multi-location.
* Monitor APIs.
* Monitor the organizational structure.

## Microservices Monitoring Tool

There are three monitoring tools are as follows:

* Hystrix dashboard
* Eureka admin dashboard
* Spring boot admin dashboard

## Microservice Virtualization

Microservices virtualization is the method to simulate the behavior of specific components in various component-based applications like cloud-based application, SOA, and API driven architecture. Service virtualization also reduces cost and save time. By combining service virtualization, an organization can develop the application which can be delivered from various locations and dissimilar environments.

### Spring Cloud Config Server

Spring Cloud Config Server provides the HTTP resource-based API for external configuration in the distributed system. We can enable the Spring Cloud Config Server by using the annotation **@EnableConfigServer**.

| Spring Cloud Config Server | 8888 |
| --- | --- |

1. limits-service.minimum=99
2. limits-service.maximum=9999

There is a better approach in Spring Boot to read values from the configuration using the annotation **@ConfigurationProperties**.

1. @Component
2. @ConfigurationProperties("limits-service")
3. **public** **class** Configuration
4. {
5. **private** **int** maximum;
6. **private** **int** minimum;

}