**Monolithic Architecture**

single database using single code

Monolithic Architecture is like a big container wherein all the software components of an application are assembled together and tightly packaged.

## **Challenges of Monolithic Architecture**

**Scenario 1: Scalability:** Let’s assume that the developers want to update the playlist according to most popular tv shows and also simultaneously want to update all videos to HD quality.

The developers cannot scale the application simultaneously. New instances of the same application have to be created every time a new feature has to be developed or deployed.

**Scenario 2: Agility:** Assume that developers want to make immediate changes in the application.

The monolithic application can definitely accommodate these changes. But, the problem here is that the developers have to rebuild the code for every small change.

**Scenario 3: Hybrid Technologies:** Suppose developers of this application are comfortable with various technologies like JAVA, C++,.NET, C#.

Even though they are comfortable with various technologies, they still have to build large and complex applications on a single technology.

**Scenario 4: Fault Tolerance:** Let’s suppose that a specific feature is not working in the application.

The complete system goes down because of this problem. In order to tackle this problem, the application has to be re-built, re-tested and also re-deployed.

So, how did the developers overcome these complexities?

=> microservices.

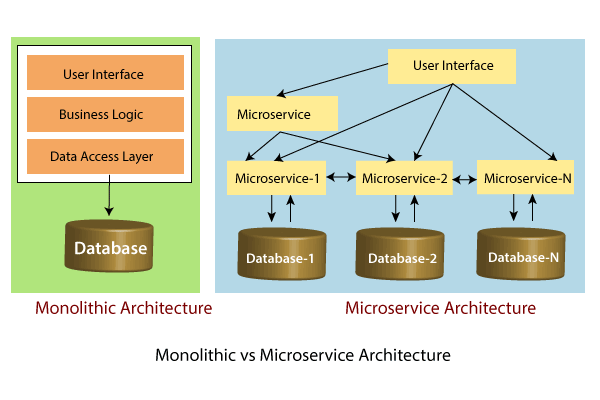
## What are Microservices

**Definition**: According to **Sam Newman**, "Microservices are the small services that work together."

According to **James Lewis and Martin Fowler**, "The microservice architectural style is an approach to develop a single application as a suite of small services. Each microservice runs its process and communicates with lightweight mechanisms. These services are built around business capabilities and independently developed by fully automated deployment machinery."

**Microservices**, is an architecture style that structures an application as a collection of small autonomous services, modelled around a Business Domain.

In Microservice Architecture, each service is self-contained and implements a single Business capability.



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**

The infrastructure automation is the process of scripting environments. With the help of 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 module.
* 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.
* Mixed technology stack
* Fault isolation
* Granular scaling - Services can be scaled independently

e.g. scaling only card service instead of entire application

* It is possible to change or upgrade each service individually rather than upgrading in the entire application.
* Microservices follows 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 has 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.
* Configuration Management - solution..centralized configuration
* Debugging - solution..centralized logging
* Consistency - to have some decentralized governance around the languages platforms technology and tools

Advantages of Scaling

* **Cost** − Proper scaling of a software will reduce the cost for maintenance.
* **Performance** − Due to loose coupling, the performance of a properly scaled software is always better than a non-scaled software.
* **Load distribution** − Using different technologies, we can easily maintain our server load.
* **Reuse** − Scalability of a software also increases the usability of the software.

**Eureka Server:**

**Eureka Server** is an application that holds the information about all client-service applications. Every Micro service will register into the **Eureka server** and **Eureka server** knows all the client applications running on each port and IP address. **Eureka Server** is also known as Discovery **Server**.