Microservices

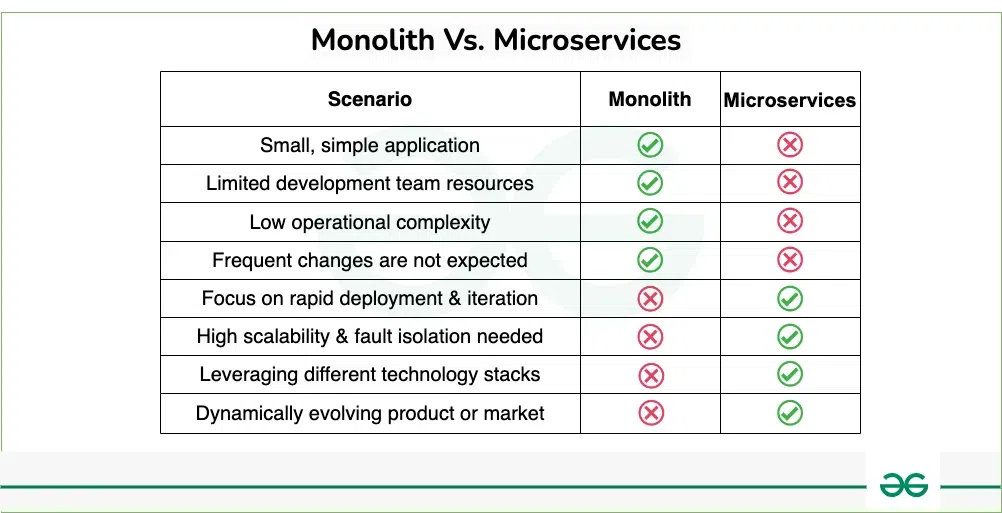
Microservices is an architectural style where an application is composed of loosely coupled, independently deployable services. Each service is designed to perform a specific business function and can be developed, deployed, and scaled independently.

# Benefits of Using Microservice Architecture

1. Scalability: Microservices architecture enables horizontal scaling, allowing individual services to be scaled independently based on demand
2. Agility: Microservices promote faster development and deployment cycles. Since each service is relatively small and focused on a specific business function, teams can work on them independently, reducing dependencies and bottlenecks.
3. Resilience: Failure in one service does not necessarily bring down the entire system.
4. Technology Diversity: Different services within a microservices architecture can be implemented using different technologies, programming languages, and frameworks
5. Easy Maintenance and Scaling: With smaller codebases and clear boundaries between services, maintenance becomes more manageable

# Difference Between Monolithic and Microservice

| **Aspect** | **Monolithic Architecture** | **Microservice Architecture** |
| --- | --- | --- |
| **Architecture** | Single-tier architecture | Multi-tier architecture |
| **Size** | Large, all components tightly coupled | Small, loosely coupled components |
| **Deployment** | Deployed as Single Unit | Individual Services Can be deployed independently |
| **Scalability** | Horizontal scaling can be challenging | Easier to Scale Horizontally |
| **Development** | Development is Simpler Initially | Complex due to managing multiple services |
| **Technology** | Limited Technology Choice | Freedom to choose best technology for each services |
| **Fault Tolerance** | Entire application may fail if a part fail | Individual services can fail without affecting others |
| **Maintenance** | Easier to maintain due to its simplicity | Requires more effort to manage multiple services |
| **Flexibility** | Less flexible as all components are tightly coupled | More flexible, as components can be developed, deployed, and scaled independently |
| **Communication** | Communication between components is faster | Communication may be slower due to network calls |



# Load Balancer

A load balancer is a device or software component that distributes incoming network traffic across multiple servers or resources. Its primary purpose is to improve the availability and reliability of applications, websites, or services by ensuring that no single server or resource becomes overwhelmed with requests, thus preventing performance degradation or downtime

## Different Load Balancer Algorithms

1. Round Robin: This is one of the simplest load balancing algorithms. Each new request is routed to the next server in a circular manner, distributing the load equally among all available servers.
2. IP Hash: With this algorithm, the client's IP address is used to determine which server receives its request. The request is consistently routed to the same server based on the client's IP address, ensuring session persistence. It's beneficial for applications that require maintaining session state.
3. Random Selection: Requests are distributed randomly among the available servers. While simple to implement, this approach may not distribute the load evenly and could lead to suboptimal resource utilization.
4. Weighted Round Robin: Similar to Round Robin, but with the capability to assign different weights to servers based on their capacities. Servers with higher capacities are assigned higher weights, allowing them to handle more traffic compared to servers with lower weights.

## Consumers and Producers

## Communication between Microservices

## Eureka Server and Eureka Client

@EnableEurekaServer - It is used with eureka server service to make it server

**Properties need to be added**

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

### Eureka Client

Properties to be added

spring.application.name=Microservice-2

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

eureka.instance.hostname=localhost