Microservices

What are Microservices

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."

There is a bare minimum of centralized management of these services, which may be written in different programming language and use different data storage technologies.

The microservice defines an approach to the architecture that divides an application into a pool of loosely coupled services that implements business requirements. It is next to **Service-Oriented Architecture (SOA)**. The most important feature of the microservice-based architecture is that it can perform **continuous delivery** of a large and complex application.

Microservice helps in breaking the application and build a logically independent smaller applications. For example, we can build a cloud application with the help of Amazon AWS with minimum efforts.

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

Principles of Microservices

There are the following principles of Microservices:

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

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.

- 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 latterly upgrades by adding more functions or modules) in nature.
- It enables event streaming technology to enable easy integration in comparison to heavyweight interposes communication.
- 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.

Challenges of Microservices Architecture

Microservice architecture is more complex than the legacy system. The microservice environment becomes more complicated because the team has to manage and support many moving parts. Here are some of the top challenges that an organization face in their microservices journey:

- Bounded Context
- Dynamic Scale up and Scale Down

- Monitoring
- Fault Tolerance
- Cyclic dependencies
- DevOps Culture

Bounded context: The bounded context concept originated in Domain-Driven Design (DDD) circles. It promotes the Object model first approach to service, defining a data model that service is responsible for and is bound to. A bounded context clarifies, encapsulates, and defines the specific responsibility to the model. It ensures that the domain will not be distracted from the outside. Each model must have a context implicitly defined within a sub-domain, and every context defines boundaries.

In other words, the service owns its data and is responsible for its integrity and mutability. It supports the most important feature of microservices, which is independence and decoupling.

Dynamic scale up and scale down: The loads on the different microservices may be at a different instance of the type. As well as auto-scaling up your microservice should auto-scale down. It reduces the cost of the microservices. We can distribute the load dynamically.

Monitoring: The traditional way of monitoring will not align well with microservices because we have multiple services making up the same functionality previously supported by a single application. When an error arises in the application, finding the root cause can be challenging.

Fault Tolerance: Fault tolerance is the individual service that does not bring down the overall system. The application can operate at a certain degree of satisfaction when the failure occurs. Without fault tolerance, a single failure in the system may cause a total breakdown. The circuit breaker can achieve fault tolerance. The circuit breaker is a pattern that wraps the request to external service and detects when they are faulty. Microservices need to tolerate both internal and external failure.

Cyclic Dependency: Dependency management across different services, and its functionality is very important. The cyclic dependency can create a problem, if not identified and resolved promptly.

DevOps Culture: Microservices fits perfectly into the DevOps. It provides faster delivery service, visibility across data, and cost-effective data. It can extend their use of containerization switch from Service-Oriented-Architecture (SOA) to Microservice Architecture (MSA).

Other challenges of microservices

- As we add more microservices, we have to be sure they can scale together. More granularity means more moving parts, which increase complexity.
- The traditional logging is ineffective because microservices are stateless, distributed, and independent. The logging must be able to correlate events across several platforms.

- When more services interact with each other, the possibility of failure also increases.