What is Monolithic Architecture?

In a Monolithic architecture, an application is built as a single unit and has a single logical executable. A standard enterprise application has a multi-layered architecture, generally presentation layer, service layer and data layer.

Benefits of Monolithic Architecture

1.Simple to develop (dividing it into multiple layers e.g. Presentation Layer, Service Layer, Data Access Layer just simplifies the development process).

2.Easy to test.

3.Easy to deploy.

4.Simple to scale.

Drawbacks of Monolithic Architecture

1.Even for a small change in the code, entire application needs to be re-built and re-deploy.

2.This type of Applications sometimes is not reliable. If any bug in the application, it may affect complete service down or application may not work as expected. Hence, one small problem may affect the entire application.

3.Adding new concept/technologies/new Integration may become very complex.

4. As the number of modules increase, then application size increases, downtime for re-deployment may also increase accordingly.

Microservices Architecture

The term Microservice Architecture is a "particular way of designing software applications. While applying microservices architecture in our application, we separate out business functionality into multiple independent modules that are independently responsible for performing specifically described, standalone tasks. These modules communicate with each other through simple, globally accessible application programming interfaces (APIs).

Microservice architectures offer you to break applications into distinct self-governing services, each managed by separate groups within your development organisation. It supports the separation of responsibilities critical for building highly scalable applications. Moreover, It allows a developer to work independently on individual services without impacting the work of other developers in other groups working on the same application as a whole.

How do internal services communicate with each other in Microservices Architecture?

Synchronous and Asynchronous

Benefits of Microservices Architecture

1. High flexibility for changes, If we make any change in one service, we don't need to stop other services.

2.If one Microservices is not working properly then it may not affect all other services.

3. Modification, Upgrade, Enhancements to new technologies are not so complex as it is in Monolithic architecture.

4. Scaling Service is easy, as Microservice architecture enables each service to be scaled independently. Memory will also be used in an Efficient way.

5. It makes continuous deployment possible for complex applications as Microservice architecture offers each microservice to be deployed independently.

6. It provides a choice of adopting new technologies in between the development instead of making choices at the start of the project.

7. There is always a possibility to replace small parts of the system easily, in case if any part is unreliable or the client demands it to be replaced.

Drawbacks of Microservices Architecture

1. Complexity arises when we need to implement the inter-process communication mechanism.

2. Testing microservices also becomes much more complex as compared to monolithic.

3. Sometimes a minor change becomes very difficult to implement in a Microservices Architecture based application. Suppose a change impacts multiple services, then you need to plan it carefully and coordinate changes to each of the services.

4. we need to configure, deploy, scale, and monitor each instance separately.

5. Even Runtime Environment needs very high configuration and tools such as AWS Cloud, Google Cloud, Azure Cloud, PCF Cloud etc. in Microservices Architecture.

6. Maintenance of Microservices Architecture based applications is also very Complex and Cost Effective.

Common Tools & Frameworks with Spring Cloud

Spring Cloud is the technology/framework which implements Microservices.

1. Generally, every module(service) is implemented as a single Project i.e. called as Microservices using a Spring Rest Controller concept.

2. Once we complete writing Microservices Application, it must be published by the Register and Discovery Server (R&D Server) i.e. Netflix Eureka. In short, we call it Eureka Server.

3. One Microservice can find and communicate with another Microservice using the Registration with Eureka only. When it gets published by Eureka Server, other Microservices can find it for further communication.

4. Common properties having similar values must be provided outside of all Microservices using the concept of Config Server. (via GitHub, GitLab etc.)

5. If any Microservice is throwing Exceptions repeatedly, then Stop Executing such Microservice for some duration and inform it to Admin Dashboard. In order to handle this, we use the Circuit Breaker concept(via Netflix Hystrix). However, the new concept for Fault tolerance is Resilience4j as Hystrix is deprecated now.

6. If any Microservice incorporates Load Balance Support (multi-instances), then we need to use Load balancer Client (via Ribbon or Feign Client)

7. Logging & Tracing in Microservices is possible via Sleuth and Zipkin or ELK Stack etc.

8. In order to use all tools, we need one entry and exit concept i.e. API Gateway (via Netflix Zuul). The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end. It supports Dynamic Microservice instance selection, request filtering, Single Sign On (SSO- One time login), HTTP support for Global Data in JSON format. The new implementation for this is Spring Cloud Gateway API as Zuul is deprecated now.

9) We can integrate with any type of HTTP client like Angular, Android Device, .Net REST API and many more.