**MICROSERVICES**

**What is monolithic architecture?**

   - entire software is composed into single piece (i.e.) designed to self-contained (i.e.) all programs are interconnected and interdependent

   - it will give tightly coupled application where each component and its associated component should be present in order to run the application

   - It is big container or it is not divided into small-small services or components, all the components are tightly coupled with each other

**Advantages**

   - very simple to develop

   - simple to deploy because we have to deploy single executable file (war file)

   - To scale your application, we have to run multiple copy of application behind the load balancer

**Drawbacks**

   - not suitable for large and complex application - If we have large application, it is really complex to understand and modify the application, as a result the development will slow down and quality of code also declined

   - Slow development - complex application, more team members, difficult to understand and modify the code

   - Blocks continuous development - to update one service we have to redeploy all the services, there also a chance the service hasn’t been updated will fail to start correctly because of some dependency issue

   - Un-scalable - Each copy of application service will get access to all of data which makes caching less effective and memory consumption also more

            - Different service requires different configuration

  - Unreliable - because of tightly coupled components, if one component failed then entire application will shut down

  - Inflexible - Monolithic architecture becomes difficult to adopt new languages and new framework

**Micro-service architecture**

   - If we have large application, it can be broken down into small-small multiple services that together form one large application

   - In micro-service architecture, all components are divided into separate modules which communicate with each other using well define interface using REST endpoints

   - Each micro-service is responsible for its own data because of which interaction with each micro-service is handled by different instance

   - Services are small, independent and loosely coupled. Each service has separate code and managed by separate team and deployed separately. The team can update the existing service without rebuilding and deploying the entire application

   - Service are responsible for persisting its own data, so its internal implementation of each service are hidden from other service

   - Services don’t need to share the same technology stack, libraries, framework etc.

**Micro-service components**

1. Service Registry/Service Discovery

       - maintain the list of services, it identifies each service through REST endpoints

       - Netflix Eureka, Consul, Zookeeper

2. API Gateway

       - entry points of client. The client doesn’t call all the services directly, it goes to API gateway which forwards the call to related services in backend

       - ZUUL Proxy

3. Fault tolerance -

      -Suppose an exception is happening in one service, then instead of throwing an exception we want to invoke some callback method

      - Netflix Hystrix

4. Client side load balancer

     - If multiple instance of micro-service is created, so whenever the request come, which micro-service instance have to be invoked

     - Netflix Ribbon

Advantages

1. Small Focused- simplicity - it can be rewritten and maintain without any extra effort from development team

2.Loosely coupled - Each micro-service is independent of each other

3. Language neutral - different service can be written in different language

4. Bounded Context - each micro-service doesn’t need to understand the implementation of other service

5. Independent deployment - each micro-service can be developed and deployed independently

6. Fault isolation - if one micro-service goes down, it won’t take the entire application down

7. Mixed technology stack - each service in different technology

8. Granular scaling - If we want to scale one service then only that service would be scaled

9. Amazon, Uber, E-bay, Pay pal, Netflix

Steps

1. We create Eureka server with one dependency EurekaServer

@EnableEurekaServer - used to make your Spring Boot application acts as a Eureka Server which in runs in port no 8761

2. Create EmployeeProducerService with Spring web and Eureka Discovery client

@EnableEurekaClient - will work only for Netflix eureka

@EnableDiscoveryClient - will be used for other service registry

application.properties

1. It stores data in sequential format as key value pairs

server.port=2000

2. It supports only key value pair with both as string

3. used only in Java

4. If u want to handle multiple profiles, we need to create different properties files

application.yml

1. It stores data in hiearchial format

server:

  port: 2000

2. It supports key value pair as well as map, list and scalar data type

3. can be used in Java, PHP, Python etc.,

4. We can have single yml file to define all profiles

DiscoveryClient used on the client side to search for services

RestTemplate is the class within the Spring framework for executing synchronous HTTP requests on the client side

Hystrix is a library created by Netflix used to handle failures and do some fallbacks when there is a failure.

@EnableCircuitBreaker

@EnableHystrixDashboard

Actuator endpoints let you monitor and interact with your application. Spring Boot includes a number of built-in endpoints and lets you add your own

Ribbon is a client-side load balancer that gives you a lot of control over the behavior of HTTP and TCP clients.