**Microservices**

**Centralized configuration:**

Create a new project spring cloud config server by adding ***spring-cloud-config-server*** dependency and add **@EnableConfigServer** annotation on the main class. Also add below properties in application.properties file.

*spring.application.name=spring-cloud-config-server*

*server.port=8888*

*spring.cloud.config.server.git.uri=file:///C:/Users/vpalla/Desktop/vishal/Microservices/git-localconfig-repo*

file. If another service say ***limits-service*** wants to consume properties(limit-service.minimum=10,limit-service.maximum=1000) from github using ***spring-cloud-config-server***, we need to add dependency ***spring-cloud-starter-config.*** . Also add below properties in application.properties file.

*spring.application.name=limits-service*

*spring.config.import=optional:configserver:http://localhost:8888*

**Spring-cloud-starter-openfeign:**

We use spring cloud feign in our services to consume other microservices rest endpoints. We need to add spring-cloud-starter-openfeign as dependency and add @EnableFeignClients annotation in main class and create a proxy interface and see below snapshots from Proxy and Controller.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Eureka Naming Sever or Service Registry**

To make any project or service as naming server we must add Eureka server dependency which is ***spring-cloud-starter-netflix-eureka-server***. Also add @EnableEurekaServer annotation in the main class. Give a good name for server spring.application.name=naming-server. We should add some config in application.properties so eureka server won’t register itself in registry.

*eureka.client.register-with-eureka=false*

*eureka.client.fetch-registry=false*

*server.port = 8761*

Now we go to browser and hit ***localhost:8761*** then we can see eureka server dashboard. To register other services in eureka open other microservices and dependency ***spring-cloud-starter-netflix-eureka-client***. And add below line in application.properties.

***eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka***

**Note** : ***Feign asks eureka server for instances of currency-exchange service and load balance between them. When we open dependency hierarchy, we can see spring-cloud-starter-loadbalancer in eureka naming server dependency. And feign uses this loadbalancer framework to distribute the load among the multiple instances which are returned by eureka. In the earlier versions of Spring Cloud, the load balancer which was used was Ribbon and in the recent versions, Spring Cloud shifted to using Spring Cloud Load Balancer as the load balancer. The great thing is, if you're using Eureka and Feign, then load balancing comes for free. This is client-side load balancing, and this comes for free for you.***

**API Gateway**

In previous versions we used Zuel API as gateway but now in latest we are using spring-cloud-API-gateway. We need to add dependency ***spring-cloud-starter-gateway*** and eureka client dependency in a new project to make it as a gateway.

#spring.application.name=api-gateway

#server.port=8765

#eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

And in application.properties file add below line so that it will talk eureka server to know the registered microservices. So, we can call currency exchange service with api gateway server with below URL and currency-exchange is the name of currency-exchange service in eureka so we can use it **http://localhost:8765/currency-exchange**/appendcorrectURL.

**http://localhost:8765/currency-exchange**/currency-exchange/from/{from}/to/{to}

#spring.cloud.gateway.discovery.locator.enabled=true

#spring.cloud.gateway.discovery.locator.lower-case-service-id=true

But URLS looks overwritten so we need to eliminate extra currency-exchange in the path. The way we can do that is configuring routes. Whenever we get a request from “/currency-exchange/\*\*” we can ask eureka to find that service and that service handle the request and load balance it. “lb://currency-exchange”

Graphical user interface, text, application

Description automatically generated

**Spring cloud Gateway**

* Simple, yet effective way to route to APIs.
* Provide cross cutting concerns:

1. Security
2. Monitoring/metrices

* Build on top of Spring web flux (Reactive Approach)
* Features :

1. Match routes on any request attributes.
2. Define Predicates and filters.
3. Integrates with Spring cloud Discovery client (Load Balancing).

**Circuit Breaker** :

Graphical user interface, text, application

Description automatically generated

To start we need to add spring-boot-starter-aop, actuator and resilience4j-spring-boot2 in the currency-exchange service.

**@Retry** : We use this annotation on a method if the method throws an error, we are saying to method please retry 3 more times and after give the response. We can also set fallBackMethod response in case after 3 times also if method throws error. We can also configure the no. of retries.

**@CircuitBreaker** : If a microservice or an API is down or failing for a period time, then circuit breaker says hey the API is failing for a certain period time now I wont call the API anymore, I will return default fall back method response.

**@RateLimiter** : We can configure like this, in 10 seconds(limit-refresh-period) allow only 2(limit-for-period) requests. We can do that in application.properties.

**@Bulkhead** : We can configure how may concurrent calls can be allowed trough application.properties.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated