microservice architecture

**We will require following microservices**

1. Config -server

Used to configuration global variables ( application.properties ) from git. Using spring-cloud-config-client.

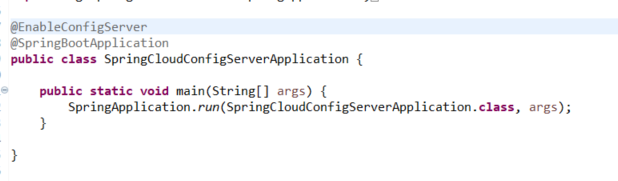
1. Eureka service registry   
    Used for registration of services so that microservices can talk with other microservices also it load balances between microservices instances.
2. API gateway  
   Used for making a centralized api where all the api calls will come and redirects to required microserve. Mostly used for proxy our original api’s. Add filters and common params to all api’s
3. Microservice 1
4. Microservice 2 …

**Important Dependencies and their use**

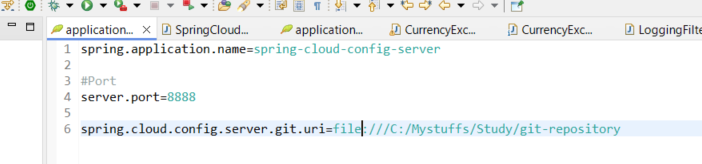
* **spring-boot-starter-actuator** -> Used for monitoring our spring boot application
* **spring-boot-starter-web** -> Used for making RestFull APIs.
* **spring-cloud-config-server** -> Used this dependency in our config server application. Here we config our server for global configurations.
* **spring-cloud-starter-config** -> Used to make configuration available for all other microservices.( In server project add @EnableConfigServer annotation and add git path in application.properties || In client where we want to use the config give application name and add config server path).
* **spring-boot-devtools** -> It is used for real-time hot reload of application when we save. (\*When we have changes in .properties file and pom file we have to do restart manually )
* **Lombok** : Used for generating boiler plate codes like builder, getter, setter, tostring and all.
* **spring-boot-starter-test** -> Used for doing test in our spring boot application
* **spring-cloud-starter-openfeign** -> Openfeign is a framwork which helps to call other microservice/ rest api for ease using proxy and also it auto oad balances using service registry.
* **spring-cloud-starter-netflix-eureka-client** -> Add this dependency helps to register your current service to eureka server with the name you are given in properties file.
* **spring-cloud-starter-netflix-eureka-server** -> This dependency is added to eureka naming server application for maintain eureka server. (<http://localhost:8761/>)
* **spring-cloud-starter-gateway** -> This dependency is added to our api gateway application. It helps to handle request using eureka server names.
* **spring-boot-starter-aop** -> This dependency is added in pom of project where we want to use resillence4j for circuit breaker.
* **spring-cloud-starter-circuitbreaker-resilience4j** -> This dependency is added for circuit braker and retires, timeout mechanism for microservices.

Config server configuration

LaunchApplication -> Add @EnableConfigServer annotation



application.properties



Dependencies

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

Eureka Service Registry

Dependencies

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

**Launch configuration**

**package** com.ankit.webservice;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer

@SpringBootApplication

**public** **class** EurekaNamingServerApplication {

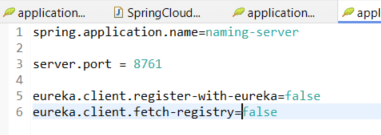
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(EurekaNamingServerApplication.**class**, args);

}

}

**Application.properties**



**API gateway**

­­­Dependencies

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

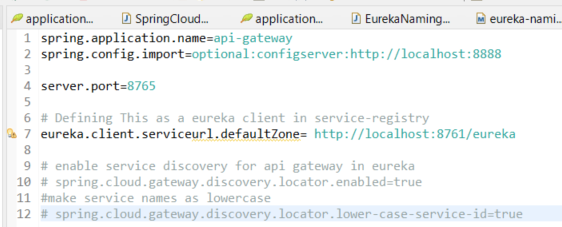
<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

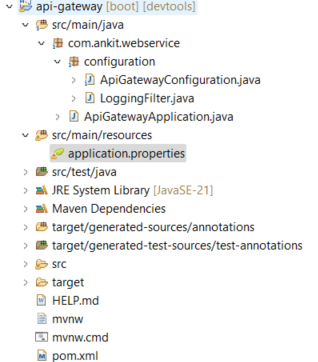
</dependency>

</dependencies>

Application.properties



File structure



ApiGateWayConfiguration.java



LoggingFilter.java (Adding global interceptor for api’s)



No specific condition to be added in launch

**Microservice 1 ( currency-conversion ) calling microsevice 2 ( currency-exchange )**

We will use feign framework to call api’s : It automatically handles load balancing usig eureka service registries

Dependencies

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

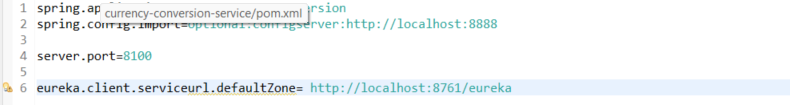
<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

</dependencies>

**Application.properties**



**Add @EnableEurekaClient in springlauch class**

**package** com.ankit.webservice;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.cloud.openfeign.EnableFeignClients;

**import** org.springframework.cloud.openfeign.FeignClient;

@SpringBootApplication

@EnableFeignClients

**public** **class** CurrencyConversionServiceApplication {

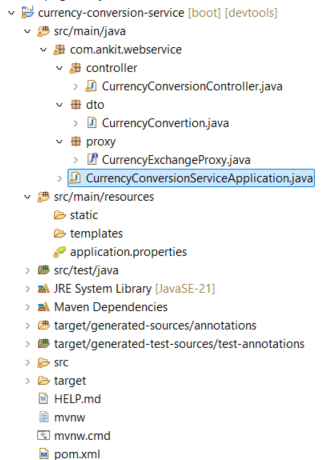
**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CurrencyConversionServiceApplication.**class**, args);

}

}

File structure



**CurrencyExchangeProxy.java**

**package** com.ankit.webservice.proxy;

**import** org.springframework.cloud.openfeign.FeignClient;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

**import** com.ankit.webservice.dto.CurrencyConvertion;

@FeignClient(name="currency-exchange")

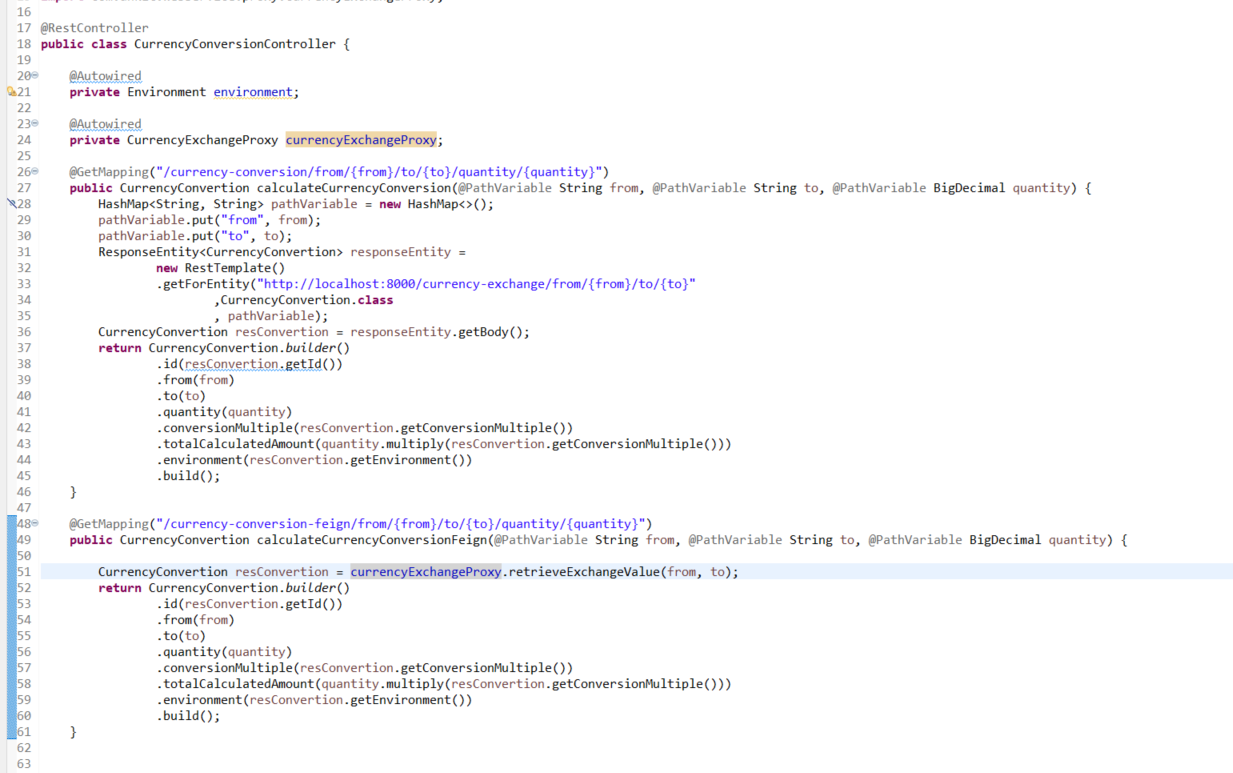
**public** **interface** CurrencyExchangeProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

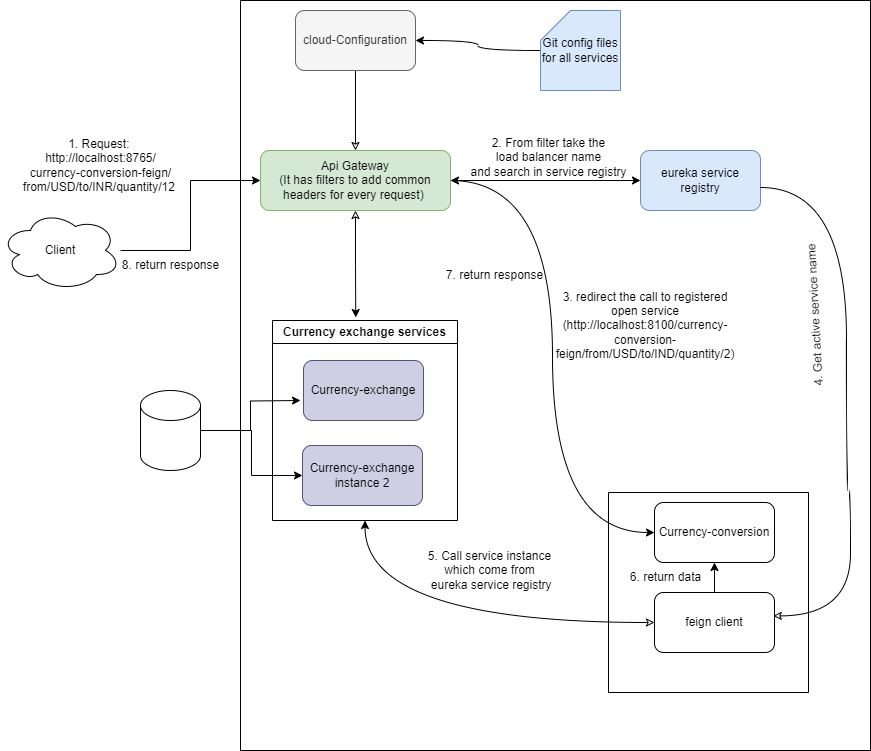
**public** CurrencyConvertion retrieveExchangeValue(@PathVariable String from, @PathVariable String to);

}

**CurrencyExcahngeController**



Flow diagram



Step1: A request from client to our api gateway url comes

Step2: From api configuration ( api gateway ) fetch the service registry information from eureka server and forward the request to required microservice

Step3: In our case, it forwards to currency conversion microservice.

Step4: In currency conversion microservice we are using feign client which works with eureka service registry to load balance between instances automatically. So, now the required instance of exchange service is called by conversion service and returns the data to api gateway.

Step5: Now api gateway will forward this reponse to the client.

Note: Using resillence4j in spring boot, we are using circuit breaker for retry, timeout, fallback mechanism. As our currency conversion microservice is calling currency exchange microservice. There might be a chance of exchange microservice is down for sometime then we required multiple tries within some interval so that our request will not stuck at one place. For this purpose we are using circuit breaker pattern with resillence4j in our spring boot application .

**Circuit breaker has 3 states**

‘Open: When the connection retry is stopped sue to lots of retry

Close : When the connection retry is happening continuously.

Half-close: When the connection retry will happen for some request only