***MicroServices***

**Step 1 :** Limits Service

Spring Cloud Starter Config -> if we have dependency for spring cloud starter config application will not start up

If we don’t have the below in application properties

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

Spring Cloud Config Client -> Dependency

**Step 2 :** Spring Cloud Config Server

Details provided in the centralized Git Hub repo can be accessed from spring cloud server and then passed on to the limit service / spring cloud config client.

**Step 3 :** Setting up GIT Repo/ Centralized config details

**Step 4 : Dependencies**

<dependency>

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

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

<version>4.0.3</version>

</dependency>

<dependency>

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

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

<version>4.0.3</version>

</dependency>

* Add above dependencies to the service from which we are going to connect to other service.

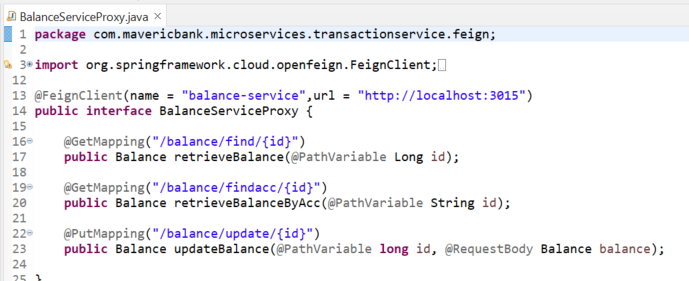
**Step 5 : App Properties**

* Add the following 2 to Application.properties in both from & to Service
* spring.application.name=balance-service Spring.cloud.config.enabled=false

**Step 6:** Add @Enable Feign Clients to From Service -> Main Method

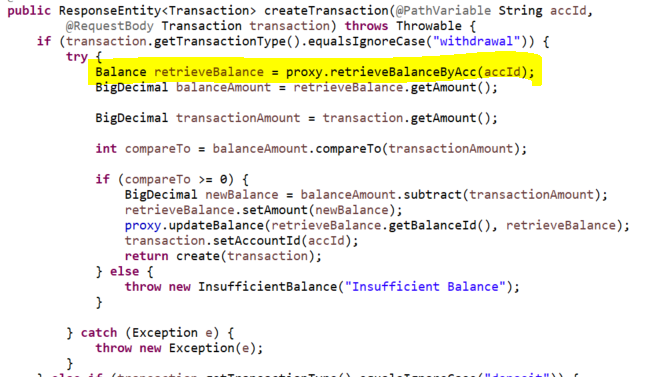
**Step 7:**

* Create a proxy in from service (Model Class of **from** and **to** service needs to be matched or copy and paste model class from **to** service to **from** service).

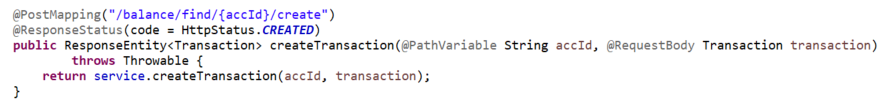


Copy the method signature from to service along with port URI

**Step 8: Calling Proxy from Service IMPL class**



**Step 9: Controller Class**



**Step 10: Naming Server or Service Registry to manage instances or load balancing**

A screenshot of a computer

Description automatically generated with medium confidence

**Add 🡪** @EnableEurekaServer in main method of naming server

**Add -> the below in application properties**

spring.application.name=naming-server

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**Step 11: Connecting microservices to naming server**

**Add the dependencies to microservices**

<dependency>

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

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

<version>4.0.2</version>

</dependency>

<dependency>

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

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

<version>3.1.1</version>

</dependency>

Launch : Localhost:8761 (Try launching)

**Step 12: in application properties of microservices add**

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

**to tag services to eureka**

**Step 13:API-Gateway**

**POM.XML**

<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>

**Application.properties**

spring.application.name=api-gateway

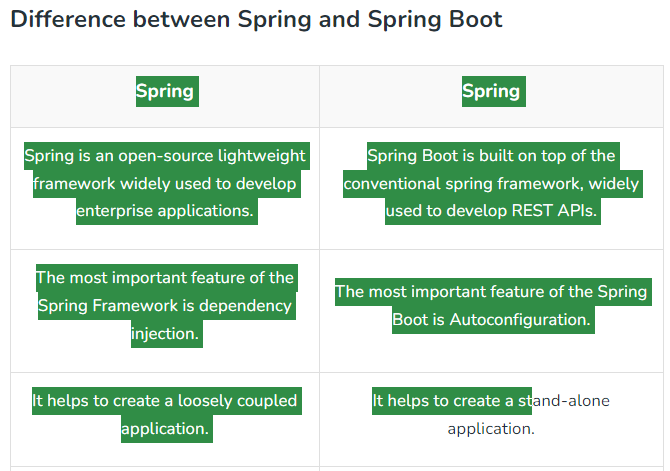
server.port=8765

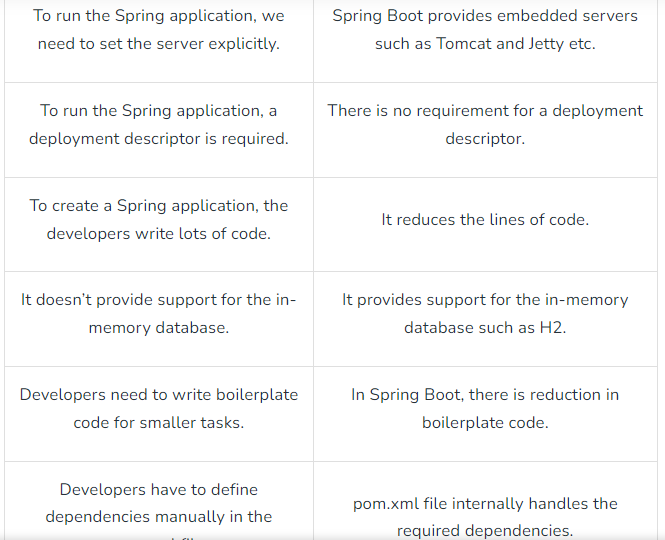
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

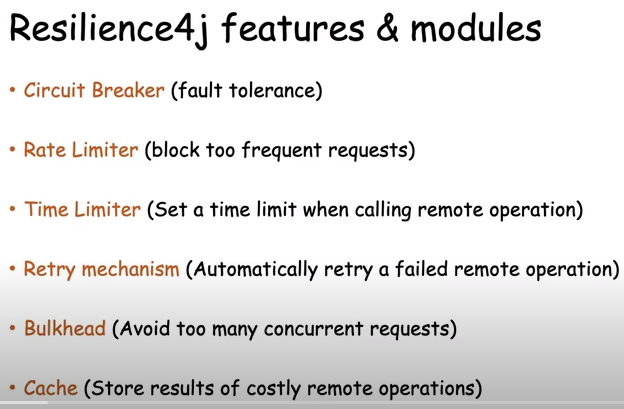
spring.cloud.gateway.discovery.locator.enabled=true

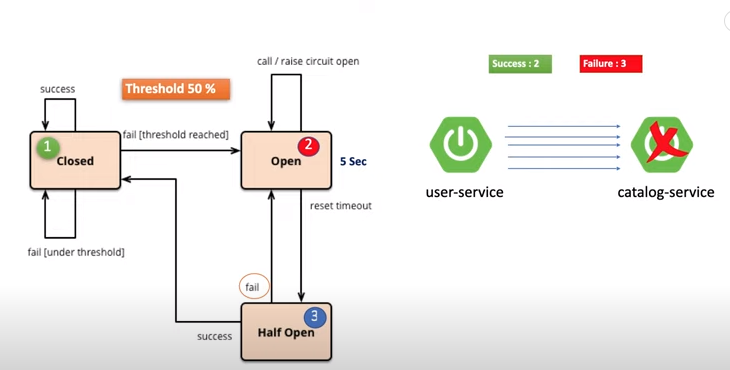
spring.cloud.gateway.discovery.locator.lower-case-service-id=true

Sample Code in Github -> Minnalll







****

**Load Balancer**

Basically, there are two ways to load balance the request

1. Client-Side Load Balancer
2. Server-Side Load Balancer

If you are keeping the load balancer on the client side and giving the load balancing responsibility to the client, then we called it Client-Side Load Balancing. And **Spring Cloud LoadBalancer is one of the most popular client-side load balancers provided by Spring Cloud.**

***Client Side Load Balancing***

[**https://www.geeksforgeeks.org/spring-cloud-client-side-load-balancer/**](https://www.geeksforgeeks.org/spring-cloud-client-side-load-balancer/)

* Note:Client-Side Load Balancer Provided by Spring Cloud***: Spring Cloud Load Balancer, Netflix Ribbon***

***Server Side Load Balancing***

[**https://www.geeksforgeeks.org/spring-cloud-server-side-load-balancer/**](https://www.geeksforgeeks.org/spring-cloud-server-side-load-balancer/)

* ***Server-Side Load Balancer Provided by Spring Cloud****: Spring Cloud Gateway, Netflix Zuul*

**Client Side Load Balancer vs Server Side Load Balancer**

| **Client Side Load Balancer** | **Server-Side Load Balancer** |
| --- | --- |
| If you are keeping the load balancer on the client side and giving the load balancing responsibility to the client, then it’s called Client-Side Load Balancing. | If you are keeping the load balancer on the server side and giving the load balancing responsibility to the server, then it’s called Server-Side Load Balancing. |
| No more single point of failure in Client Side Load Balancer. | The main disadvantage of Server-Side Load Balancing is the single point of failure. |
| Less network latency as the client can directly call the backend servers. | Network latency rises in Server-Side Load Balancing. |
| Cost Reduction as there is no need for server-side load balancing. | The cost is high to implement Server-Side Load Balancing in comparison to Client-Side Load Balancing. |
| In this case, the microservice code is combined with the load balancer’s logic. | Since each microservice will have a particular load balancer, the complexity of the system rises and it is hard to manage. |
| Client-Side Load Balancer Provided by Spring Cloud: **Spring Cloud Load Balancer, Netflix Ribbon** | Server-Side Load Balancer Provided by Spring Cloud: **Spring Cloud Gateway, Netflix Zuul** |