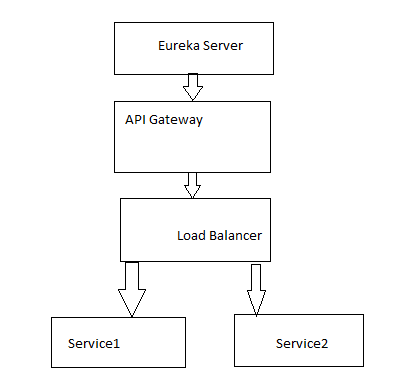
**Microservices Architecture with Spring Boot – Load Balancing with Microservices – API Gateway Integration**

**Summary**

Microservices means dividing the large applications into smaller parts called microservices. Each microservice will run in different port numbers. If we have many microservices, it is difficult to remember all port numbers of each microservice. Hence to access all microservices with single port number, it is possible using the concept called API Gateway. In this application, API Gateway will call internally Loadbalancer, it will get response from any one of the microservice using round robin fashion.



**Steps:**

1. Creating a Eureka Server in Springboot

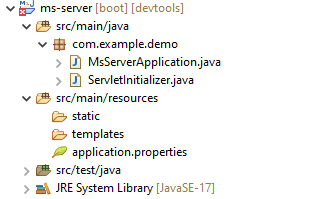
2. Building Microservices in Springboot

3. Usage of Loadbalancer in Springboot

4. Usage of API Gateway in Springboot

**4.3.1 Creating a Eureka Server in Springboot**

**Project Structure:**

****

**Dependencies to be add:**

spring-cloud-starter-netflix-eureka-server

spring-boot-starter-web

spring-boot-devtools

**Application.Properties**

**-------------------------------------**

spring.application.name=ms-server

server.port = 8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**MsServerApplication.java**

--------------------------------

@SpringBootApplication

@EnableEurekaServer

**public** **class** MsServerApplication {

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

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

System.out.println(“server is running…”);

}

}

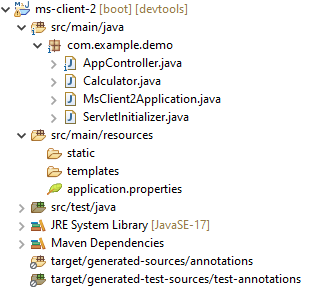
**4.3.2 Building Microservices in Springboot**

Here we are creating two microservices, that are:

Ms-client-2

Ms-client-3

**Project Structure of Ms-client-2**

****

**Dependencies to be add:**

spring-cloud-starter-netflix-eureka-client

spring-boot-starter-web

spring-boot-devtools

**MsClient2Application.java**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

@EnableDiscoveryClient

public class MsClient2Application {

public static void main(String[] args) {

SpringApplication.run(MsClient2Application.class, args);

System.out.println(“Client-2 is running…”);

}

}

**AppController.java**

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

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

import org.springframework.web.bind.annotation.RestController;

@RestController

public class AppController {

@GetMapping("/")

public String fun1() {

return "Client 2 is Running";

}

@GetMapping("/add/{a}/{b}")

public String fun2(@PathVariable("a") int a, @PathVariable("b") int b) {

Calculator c1 = new Calculator();

c1.setA(a);

c1.setB(b);

return "From Client 2: " + c1.add();

}

}

**Calculator.java**

public class Calculator {

int a, b, c;

public int getA() {

return a;

}

public void setA(int a) {

this.a = a;

}

public int getB() {

return b;

}

public void setB(int b) {

this.b = b;

}

public int getC() {

return c;

}

public void setC(int c) {

this.c = c;

}

public int add() {

c = a + b;

return c;

}

public int sub() {

c = a - b;

return c;

}

}

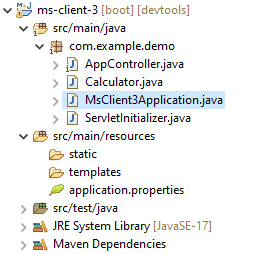
**Application.properties**

spring.application.name=ms-client-2

server.port = 8082

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

**Project Structure of Ms-client-3 (same as Ms-client-2)**



**4.3.3 Usage of Loadbalancer in Springboot**

**Dependencies to be add:**

spring-cloud-starter-netflix-eureka-client

spring-cloud-starter-loadbalancer

spring-boot-starter-web

spring-boot-devtools

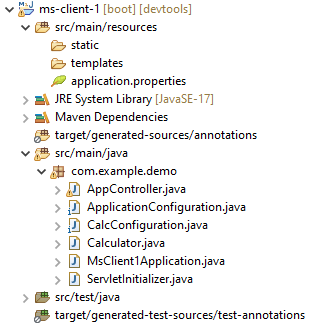
**Application.properties**

spring.application.name=ms-client-1

server.port = 8081

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

**Project\_Structure of ms-client-1:**

****

**MsClient1Application.java**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

@EnableDiscoveryClient

public class MsClient1Application {

public static void main(String[] args) {

SpringApplication.run(MsClient1Application.class, args);

System.out.println(“Client-1 LoadBalancer is running…”);

}

}

**AppController.java**

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.URL;

import org.springframework.cloud.loadbalancer.annotation.LoadBalancerClients;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Controller;

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

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

import org.springframework.web.bind.annotation.RestController;

@RestController

public class AppController {

private final Calculator calculator;

public AppController(Calculator calculator) {

this.calculator = calculator;

}

@GetMapping("/")

public String fun1() {

return "Welcome to Client 1";

}

//http://localhost:8081/10/20

@GetMapping("/add/{a}/{b}")

public String fun2(@PathVariable("a") int a, @PathVariable("b") int b) {

URL url;

String result = null;

try {

url = new URL("http://localhost:8082/add/" + a + "/" + b);

HttpURLConnection con = (HttpURLConnection) url.openConnection();

con.setRequestProperty("Accept", "application/json");

con.setUseCaches(false);

BufferedReader br = new BufferedReader(new InputStreamReader(con.getInputStream()));

result = br.readLine();

}

catch (Exception e) {

e.printStackTrace();

}

return "Result is: " + result;

}

@GetMapping("/load/{a}/{b}")

public String fun3(@PathVariable("a") int a, @PathVariable("b") int b) {

return calculator.calculate(a, b);

}

}

**Calculator.java**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.cloud.loadbalancer.annotation.LoadBalancerClient;

import org.springframework.stereotype.Component;

import org.springframework.web.client.RestTemplate;

@Component

@LoadBalancerClient(value = "calc", configuration = CalcConfiguration.class)

public class Calculator{

@Autowired

RestTemplate restTemplate;

public String calculate(int a, int b) {

return restTemplate.getForObject("http://calc/add/" + a + "/" + b, String.class);

}

}

**CalcConfiguration.java**

import org.springframework.cloud.client.DefaultServiceInstance;

import org.springframework.cloud.loadbalancer.core.ServiceInstanceListSupplier;

import org.springframework.cloud.loadbalancer.support.ServiceInstanceListSuppliers;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class CalcConfiguration {

@Bean

public ServiceInstanceListSupplier serviceInstanceListSupplier(ConfigurableApplicationContext context) {

ServiceInstanceListSupplier serviceInstanceListSupplier = ServiceInstanceListSuppliers.from("calc",

new DefaultServiceInstance("ms-client-2", "calc", "localhost", 8082, false),

new DefaultServiceInstance("ms-client-3", "calc", "localhost", 8083, false)

);

return ServiceInstanceListSupplier.builder()

.withBase(serviceInstanceListSupplier)

.build(context);

}

}

**ApplicationConfiguration.java**

import org.springframework.boot.web.client.RestTemplateBuilder;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.client.RestTemplate;

@Configuration

public class ApplicationConfiguration {

@LoadBalanced

@Bean

public RestTemplate restTemplate(RestTemplateBuilder builder) {

return builder.build();

}

}

**3.3.4 Usage of API Gateway in Springboot**

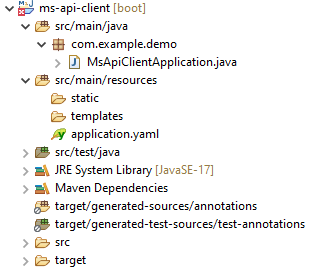
**Dependencies to be add:**

spring-cloud-starter-netflix-eureka-client

spring-cloud-starter-gateway

//spring-cloud-dependencies

**Project Structure of ms-api-client**

****

**Application.yml**

server:

port: 8085

spring:

application:

name: ms-api-client

cloud:

gateway:

routes:

- id: ms-client-1

uri: http://localhost:8081

predicates:

- Path=/load/\*\*

eureka:

client:

service-url:

default-zone: http://localhost:8761/eureka

**MsApiClientApplication.java**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

@EnableDiscoveryClient

public class MsApiClientApplication {

public static void main(String[] args) {

SpringApplication.run(MsApiClientApplication.class, args);

System.out.println(“API-Gateway is running…”);

}

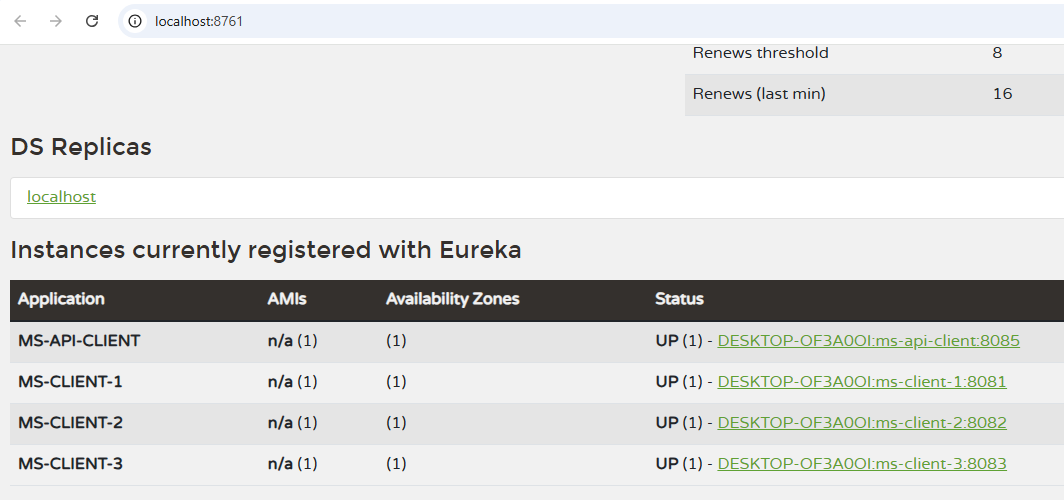
}

**Execution Process**

Restart the ms-server project (by right click on Boot Dashboard and click on Restart)

Run the ms-server project (by right click on Boot Dashboard and click on Open Web Browser)

Initially it will show “No instances are registered”



Restart and Run the ms-client1 (API-Gateway Project)

Refresh the ms-server browser. It will detect the client-1

Restart and Run the ms-client2

Refresh the ms-server browser. It will detect the client-2

Restart and Run the ms-client3

Refresh the ms-server browser. It will detect the client-3

Restart and Run the ms-api-client

Refresh the ms-server browser. It will detect the ms-api-client

<http://localhost:8081/add/12/25>

output: From Client 2: 37

<http://localhost:8081/load/20/25> (Loadbalancing by Round Robin Algorithm)

output: From Client 2: 45

output: From Client 3: 45

<http://localhost:8085/load/10/20>

(API Gateway will call loadbalancer, it will call either client2 or client3 using round robin fashion)

output: From Client 2: 30

output: From Client 3: 30

You have **two ways** to access the addition service:  
1️**Without Load Balancer:** fun2() calls http://localhost:8082/add/a/b **directly**.  
2️**With Load Balancer:** fun3() calls "http://calc/add/a/b", and **Spring LoadBalancer** automatically picks either ms-client-2 (8082) or ms-client-3 (8083).

Calculator.java

Calls "http://calc/add/a/b", and **Spring Load Balancer** selects either:

* http://localhost:8082/add/a/b
* http://localhost:8083/add/a/b

CalcConfiguration.java (Service Discovery): **Registers calc service manually with two instances:**

* ms-client-2 → localhost:8082
* ms-client-3 → localhost:8083
* **Spring LoadBalancer picks one randomly when calling http://calc/add/a/b.**

ApplicationConfiguration.java (Load Balanced RestTemplate):

**This enables load balancing for RestTemplate.**

* When Calculator calls "http://calc/add/a/b", Spring **automatically** picks an instance of calc (8082 or 8083).