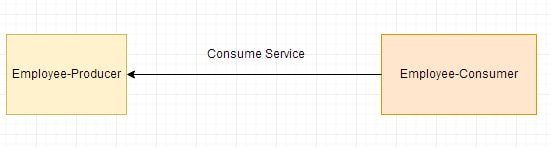
* Microservice Registration with Spring cloud using Netflix Eureka

In this post we develop 2 spring boot microservices named as employee-producer and employee-consumer.  
As the name suggests employee-producer will be exposing REST APIs which will be consumed by the employee-consumer.  
  
  
In the [next post we will develop a Eureka Server microservice using Spring Boot and register our employee-producer service to it.](https://www.javainuse.com/spring/spring_eurekaregister3)

* The pom.xml with spring batch dependencies is as follows-
* <?xml version="1.0" encoding="UTF-8"?>
* <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
* xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
* <modelVersion>4.0.0</modelVersion>
* <groupId>com.javainuse</groupId>
* <artifactId>employee-producer</artifactId>
* <version>0.0.1-SNAPSHOT</version>
* <packaging>jar</packaging>
* <parent>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-parent</artifactId>
* <version>1.4.1.RELEASE</version>
* <relativePath /> <!-- lookup parent from repository -->
* </parent>
* <properties>
* <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
* <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
* <java.version>1.8</java.version>
* </properties>
* <dependencies>
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-web</artifactId>
* </dependency>
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-test</artifactId>
* <scope>test</scope>
* </dependency>
* </dependencies>
* <build>
* <plugins>
* <plugin>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-maven-plugin</artifactId>
* </plugin>
* </plugins>
* </build>
* </project>

Define the domain class Employee

package com.javainuse.model;

public class Employee {

private String empId;

private String name;

private String designation;

private double salary;

public Employee() {

}

// getter setter

Expose the service using the Controller as-

package com.javainuse.controllers;

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

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

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

import com.javainuse.model.Employee;

@RestController

public class TestController {

@RequestMapping(value = "/employee", method = RequestMethod.GET)

public Employee firstPage() {

Employee emp = new Employee();

emp.setName("emp1");

emp.setDesignation("manager");

emp.setEmpId("1");

emp.setSalary(3000);

return emp;

}

}

Finally define the Spring Boot Main class

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootHelloWorldApplication {

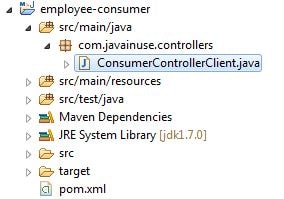
public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

Compile and the run the SpringBootHelloWorldApplication.java as a Java application.  
Go to **localhost:8080/employee**  


 Develop employee-consumer as follows- The maven project we will be creating is as follows-  
  


The pom.xml with spring batch dependencies is as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Define the controller to consume the service exposed by employee-producer above using the RESTTemplate as follows-

package com.javainuse.controllers;

import java.io.IOException;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.web.client.RestClientException;

import org.springframework.web.client.RestTemplate;

public class ConsumerControllerClient {

public void getEmployee() throws RestClientException, IOException {

String baseUrl = "http://localhost:8080/employee";

RestTemplate restTemplate = new RestTemplate();

ResponseEntity<String> response=null;

try{

response=restTemplate.exchange(baseUrl,

HttpMethod.GET, getHeaders(),String.class);

}catch (Exception ex)

{

System.out.println(ex);

}

System.out.println(response.getBody());

}

private static HttpEntity<?> getHeaders() throws IOException {

HttpHeaders headers = new HttpHeaders();

headers.set("Accept", MediaType.APPLICATION\_JSON\_VALUE);

return new HttpEntity<>(headers);

}

}

Finally create the Bean for the above controller, load it and call the getEmployee() Method.

package com.javainuse;

import java.io.IOException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestClientException;

import com.javainuse.controllers.ConsumerControllerClient;

@SpringBootApplication

public class SpringBootHelloWorldApplication {

public static void main(String[] args) throws RestClientException, IOException {

ApplicationContext ctx = SpringApplication.run(

SpringBootHelloWorldApplication.class, args);

ConsumerControllerClient consumerControllerClient=ctx.getBean(ConsumerControllerClient.class);

System.out.println(consumerControllerClient);

consumerControllerClient.getEmployee();

}

@Bean

public ConsumerControllerClient consumerControllerClient()

{

return new ConsumerControllerClient();

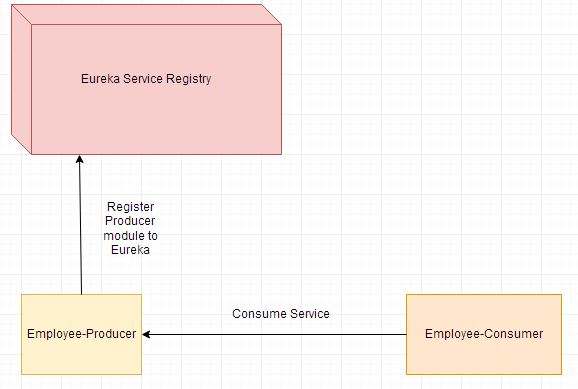
}

}

Define the application.properties as

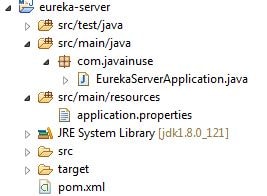
server.port=8091

Running the Rest Client we get the output as

 post we do service discovery using the employee-consumer module.  
  


Lets Begin-

* Develop the Eureka server Service as follows-

The maven project we will be creating is as follows-  
  
  
The pom.xml with spring cloud dependencies is as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>eureka-server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>eureka-server</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

**<dependency>**

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

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

**</dependency>**

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

**<dependencyManagement>**

**<dependencies>**

**<dependency>**

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

**<artifactId>spring-cloud-dependencies</artifactId>**

**<version>Camden.SR6</version>**

**<type>pom</type>**

**<scope>import</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Define the Spring Boot class with annotations @SpringBootApplication and @EnableEurekaServer.

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

**@EnableEurekaServer**

public class EurekaServerApplication {

public static void main(String[] args) {

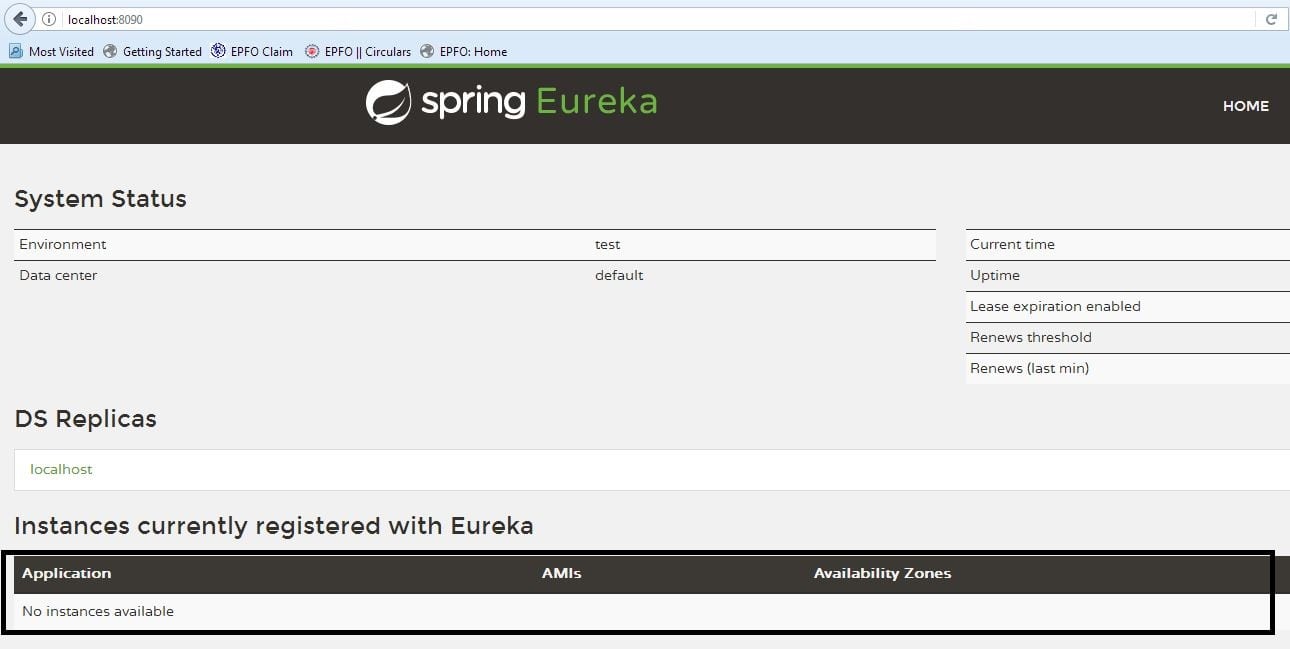
SpringApplication.run(EurekaServerApplication.class, args);

}

}

Finally specify the port on which the eureka application will get started -  
Next define the batch configuration for creating the tasklet as follows-

server.port=8090

Run this as java application. Go to URL-  
**http://localhost:8090/**  
We can see the Eureka Server page as follows-  
  


 Next we modify the [employee-producer module](https://www.javainuse.com/spring/spring_eurekaregister2) we defined previously to register to the Eureka server.

Add the Spring cloud dependencies to the pom.xml as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-producer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>SpringBootHelloWorld</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

**<dependency>**

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

**<artifactId>spring-cloud-starter-eureka</artifactId>**

**</dependency>**

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

**<dependencyManagement>**

**<dependencies>**

**<dependency>**

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

**<artifactId>spring-cloud-dependencies</artifactId>**

**<version>Camden.SR6</version>**

**<type>pom</type>**

**<scope>import</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Next define the Spring Boot class with annotations @SpringBootApplication and @EnableDiscoveryClient.

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

**@EnableDiscoveryClient**

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

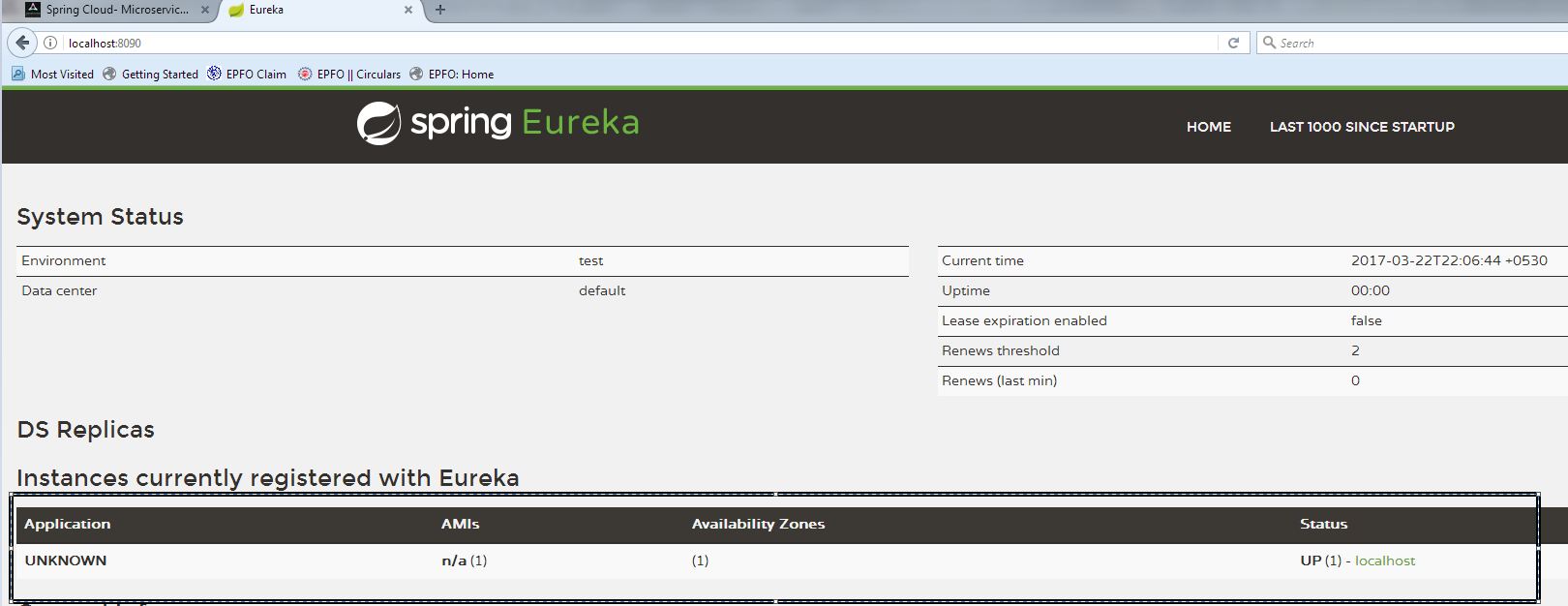
SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

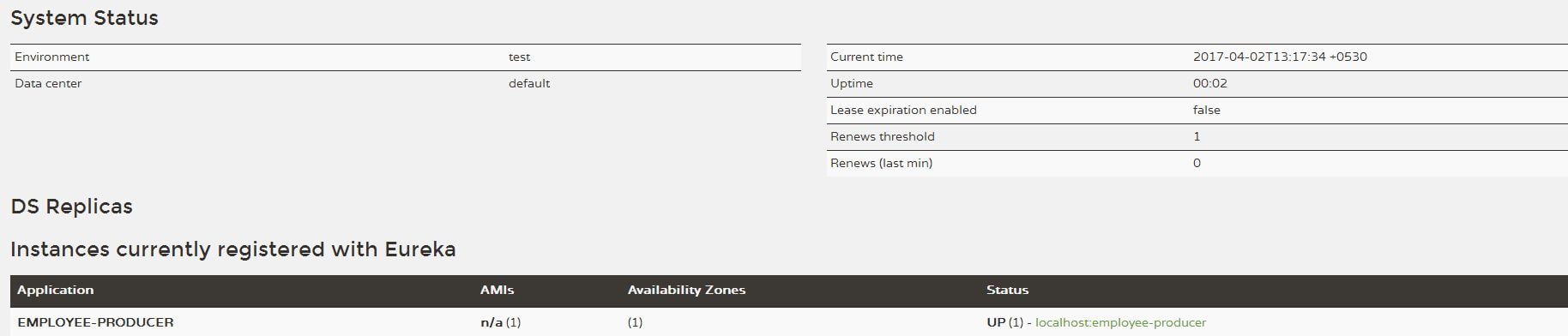
}

In the application.properties specify the url on which the Eureka server is up and running.

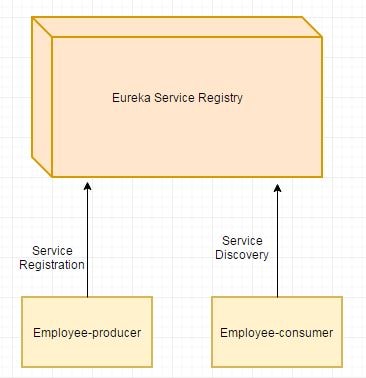
eureka.client.serviceUrl.defaultZone=http://localhost:8090/eureka

Now start this employee-producer by running this as a Java Application and go to url.  
**http://localhost:8090/**  
We can see the Eureka Server page with the employee-producer module registered as follows-  
  
  
We can see here that the registered application name is coming as UNKNOWN. To change this create in resources a file named bootstrap.properties

spring.application.name=employee-producer

We can now see the sevice registered as-  
  


 post we consume this service by discovering the employee-producer service from the Eureka server.



Lets Begin-

* Modify the employee-consumer service as follows-

Modify the pom.xml with spring cloud dependencies as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

**<dependency>**

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

**<artifactId>spring-cloud-starter-eureka</artifactId>**

**</dependency>**

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Camden.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

</project>

Modfy the ConsumerControllerClient class to autowire the **DiscoveryClient** dependency.

package com.javainuse.controllers;

import java.io.IOException;

import java.util.List;

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

import org.springframework.cloud.client.ServiceInstance;

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

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Controller;

import org.springframework.web.client.RestClientException;

import org.springframework.web.client.RestTemplate;

@Controller

public class ConsumerControllerClient {

**@Autowired**

**private DiscoveryClient discoveryClient;**

public void getEmployee() throws RestClientException, IOException {

**List<ServiceInstance> instances=discoveryClient.getInstances("employee-producer");**

**ServiceInstance serviceInstance=instances.get(0);**

**String baseUrl=serviceInstance.getUri().toString();**

baseUrl=baseUrl+"/employee";

RestTemplate restTemplate = new RestTemplate();

ResponseEntity<String> response=null;

try{

response=restTemplate.exchange(baseUrl,

HttpMethod.GET, getHeaders(),String.class);

}catch (Exception ex)

{

System.out.println(ex);

}

System.out.println(response.getBody());

}

private static HttpEntity<?> getHeaders() throws IOException {

HttpHeaders headers = new HttpHeaders();

headers.set("Accept", MediaType.APPLICATION\_JSON\_VALUE);

return new HttpEntity<>(headers);

}

}

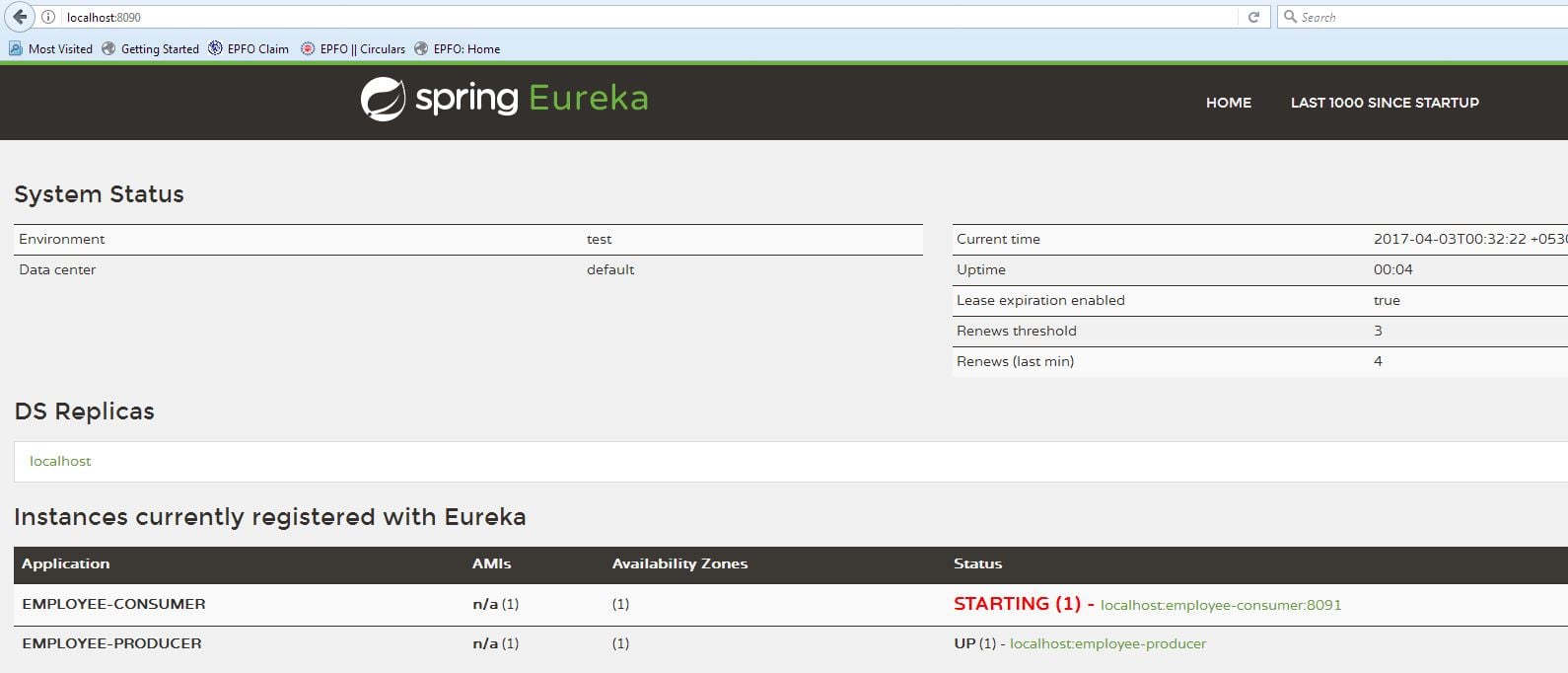
Modify the application.properties to include the eureka server url-

server.port=8091

eureka.client.serviceUrl.defaultZone=http://localhost:8090/eureka

Add the bootstrap.properties as follows-

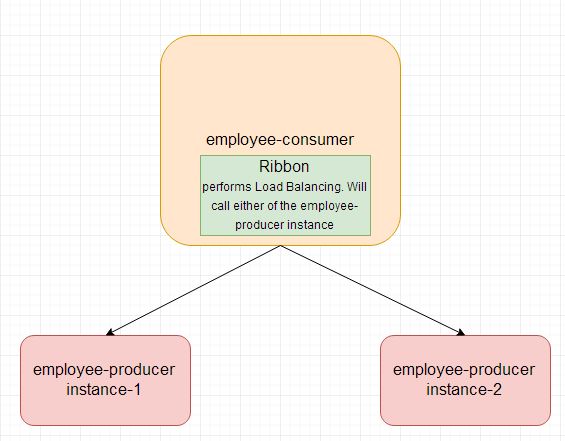
spring.application.name=employee-consumer

Run this as java application. We can see the the employee-producer is successfully consumed-  
  
  
Go to URL-  
**http://localhost:8090/**  
We can see the Eureka Server page as follows-  
  


 So the producer and client are successfully registered with Eureka Server

# Spring Cloud- Netflix Eureka + Ribbon Simple Example

What is Load Balancing? Need for Netflix Ribbon

In computing, load balancing improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.  
**In**[**previous posts we developed two services employee-producer and employee-consumer**](https://www.javainuse.com/spring/spring_eurekaregister)**. Suppose other modules are also calling and consuming employee-producer module services. So the load at employee-producer is high. To deal with this we this we deploy multiple instances of employee-producer. Suppose two in this case. Now we will have to use a Load Balancer to route any incoming requests to either one of these two services.**  


Lets Begin-

Start the [Eureka server module](https://www.javainuse.com/spring/spring_eurekaregister3) we developed in the previous tutorial.  
We will now be modifying the [employee-producer and employee-consumer modules](https://www.javainuse.com/spring/spring_eurekaregister4) we developed previously.

Code changes for employee producer-

In the application.properties add the instance id as follows-

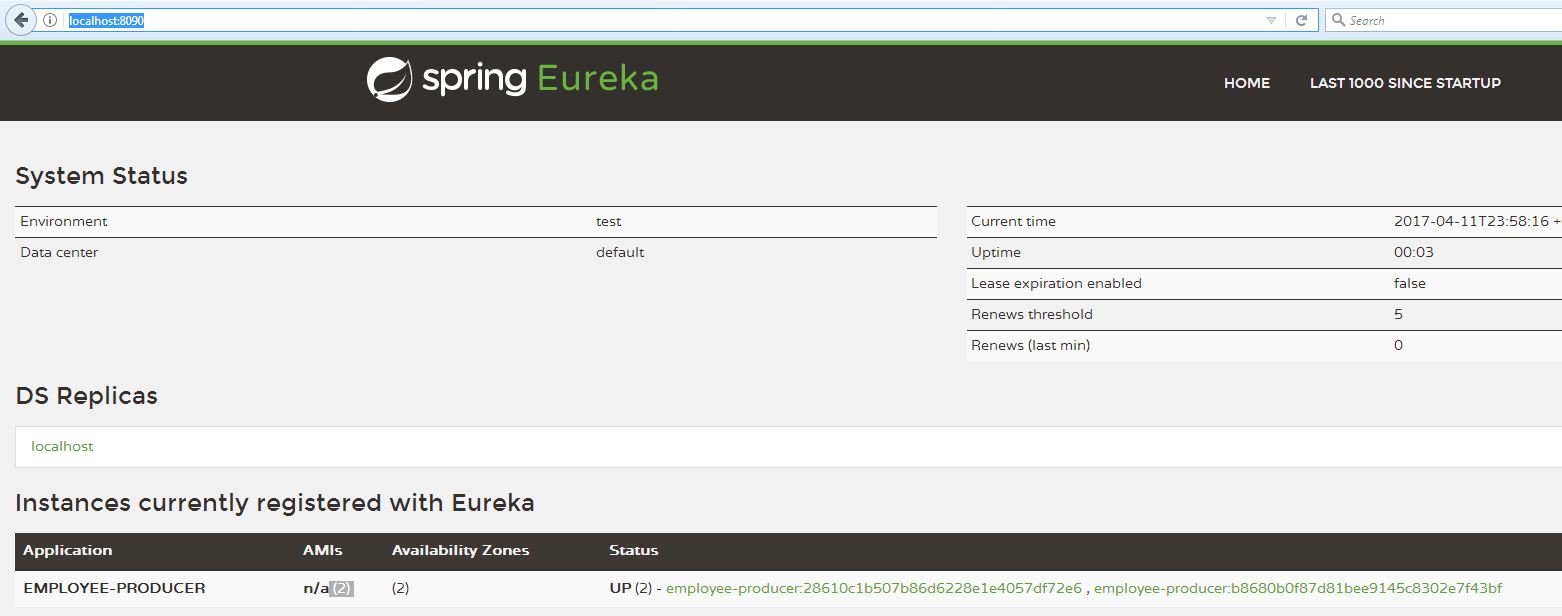
spring.application.name=employee-producer

**eureka.instance.instanceId=:**

We need to start the employee-producer instance twice. So start the employee-producer instance the first time.  
It will start on the default port 8080.  
Next in application.properties add the port as 8081 and start the employee-producer again.

eureka.client.serviceUrl.defaultZone=http://localhost:8090/eureka

**server.port=8081**

So now we have two instances of employee producer running, one on port 8080 and the other on 8081.  
If we go to eureka server url **http://localhost:8090/** we can see both instances of employee-producer registered.  


 Code changes for employee-consumer module

Modify the pom.xml to include the **spring cloud ribbon starter dependency**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**<dependency>**

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

**<artifactId>spring-cloud-starter-ribbon</artifactId>**

**</dependency>**

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Camden.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

</project>

Next modify the ConsumerControllerClient code. Previously we were autowiring an **DiscoveryClient** bean, replace it with **LoadBalancerClient**

package com.javainuse.controllers;

import java.io.IOException;

import java.util.List;

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

import org.springframework.cloud.client.ServiceInstance;

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

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

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.stereotype.Controller;

import org.springframework.web.client.RestClientException;

import org.springframework.web.client.RestTemplate;

@Controller

public class ConsumerControllerClient {

**@Autowired**

**private LoadBalancerClient loadBalancer;**

**public void getEmployee() throws RestClientException, IOException {**

**ServiceInstance serviceInstance=loadBalancer.choose("employee-producer");**

**System.out.println(serviceInstance.getUri());**

**String baseUrl=serviceInstance.getUri().toString();**

baseUrl=baseUrl+"/employee";

RestTemplate restTemplate = new RestTemplate();

ResponseEntity<String> response=null;

try{

response=restTemplate.exchange(baseUrl,

HttpMethod.GET, getHeaders(),String.class);

}catch (Exception ex)

{

System.out.println(ex);

}

System.out.println(response.getBody());

}

private static HttpEntity<?> getHeaders() throws IOException {

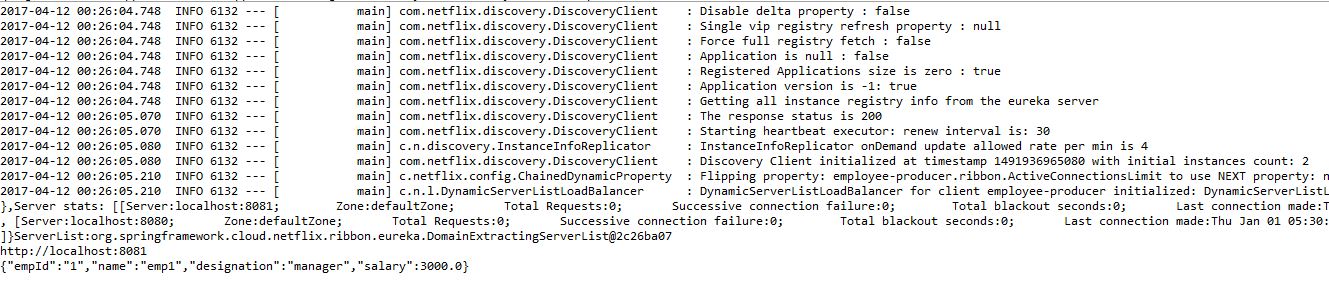
HttpHeaders headers = new HttpHeaders();

headers.set("Accept", MediaType.APPLICATION\_JSON\_VALUE);

return new HttpEntity<>(headers);

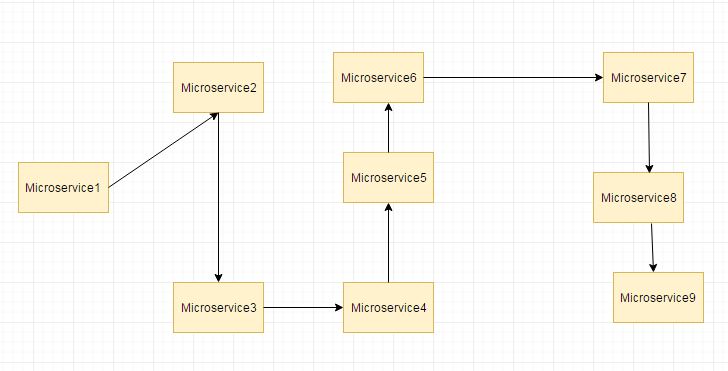
}

}

The load balancer will now call either of instances of employee-producer service depending on its internal algorithm to perform load balancing. Start the employee-consumer module.  
  


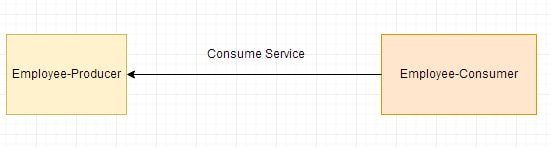
# Spring Cloud- Netflix Eureka + Ribbon + Hystrix Simple Example

What is Netflix Hystrix? Need for it?

**Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.**  
Usually for systems developed using Microservices architecture, there are many microservices involved. These microservices collaborate with each other.  
Consider the following microservices-  


Suppose if the microservice 9 in the above diagram failed, then using the traditional approach we will propagate an exception. But this will still cause the whole system to crash anyways.  
This problem gets more complex as the number of microservices increase. The number of microservices can be as high as 1000. This is where hystrix comes into picture-  
We will be using two features of Hystrix-

* Fallback method
* Circuit Breaker

In this post we will have a look at Hsytrix Fallback method. In next post we will have implement the Hystrix Circuit Breaker.  
  
[In previous posts we had two services-](https://www.javainuse.com/spring/spring_ribbon) employee-consumer consuming the service exposed by the employee-producer.  
The simplified diagram is as below-  
  
  
Now suppose due to some reason the employee-producer exposed service throws an exception. In this case using Hystrix we define a fallback method. This fallback method should have the same return type as the exposed service. In case of exception in the exposed service the fallback method will return some value.

Video

This tutorial is explained in the below Youtube Video.

Lets Begin-

Modify the employee-producer pom.xml to include the **spring cloud hystrix starter dependency**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>employee-consumer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>1.4.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**<dependency>**

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

**<artifactId>spring-cloud-starter-hystrix</artifactId>**

**</dependency>**

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-dependencies</artifactId>

<version>Camden.SR6</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

</project>

Next modify the SpringBootHelloWorldApplication code to add the EnableHystrix syntax

package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;

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

@SpringBootApplication

**@EnableCircuitBreaker**

@EnableDiscoveryClient

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

Next we modify the TestController.java to add the fallback method.

package com.javainuse.controllers;

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

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

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

import com.javainuse.model.Employee;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;

@RestController

public class TestController {

@RequestMapping(value = "/employee", method = RequestMethod.GET)

**@HystrixCommand(fallbackMethod = "getDataFallBack")**

public Employee firstPage() {

Employee emp = new Employee();

emp.setName("emp1");

emp.setDesignation("manager");

emp.setEmpId("1");

emp.setSalary(3000);

if(emp.getName().equalsIgnoreCase("emp1"))

throw new RuntimeException();

return emp;

}

**public Employee getDataFallBack() {**

**Employee emp = new Employee();**

**emp.setName("fallback-emp1");**

**emp.setDesignation("fallback-manager");**

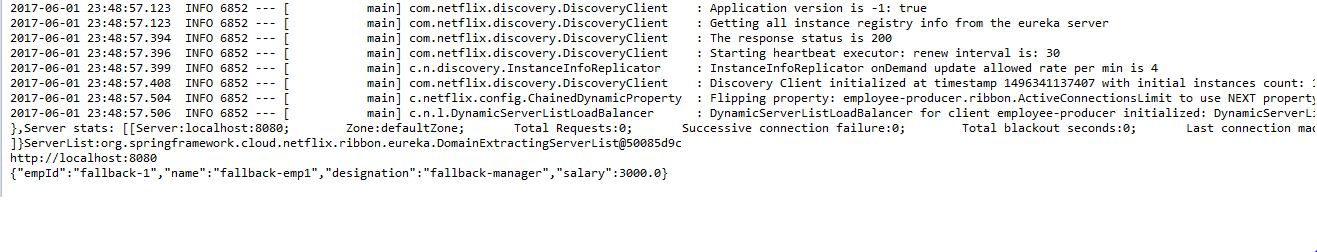
**emp.setEmpId("fallback-1");**

**emp.setSalary(3000);**

**return emp;**

**}**

}

Next start the Eureka server, employee-producer, employee-consumer modules. When we start the employee-consumer, the service exposed by the employee-producer gets called.  
  
**In the employee-consumer console we will see that the Employee object created in the fallback method of TestController class of employee-producer.**