# **Guided LAB - 310.2.1 - SP20 - Develop Microservices Using Spring Eureka and Spring Boot**

# 

# **Lab Overview**

# In this lab, we will demonstrate how to create [**microservice**](https://howtodoinjava.com/microservices/microservices-definition-principles-benefits/)s, based on [**Spring Cloud**](https://projects.spring.io/spring-cloud/) and [**Eureka**](https://github.com/Netflix/eureka) **registry server,** and create **microservices (Eureka clients)** register and discover services to call their APIs.

# 

# **Learning Objective**

# By the end of this lab, learners will be able to

# Create microservices for [Spring Cloud](https://projects.spring.io/spring-cloud/) and [Eureka](https://github.com/Netflix/eureka) registry server.

* Build a microservices using Eureka clients

# Build a discovery server using Eureka server.

**Introduction**

## We will create three microservices for this Spring Cloud Eureka example.

1. **Eureka Service Registry Server** – This microservice will provide the service registry and discovery server.
2. **Student Microservice** – This will give some functionality based on the Student entity. It will be a rest-based service, and most importantly, it will be a **eureka client service,** which will talk with eureka service to register itself in the service registry.
3. **School Microservice** – Same type as Student service – Which will give some functionality based on the Schools entity.

*Note that with Eureka, the discovery server is also known as a* ***Eureka server***. Meanwhile, the microservices are also known as ***Eureka clients***. I may use these terms interchangeably from here on.

# 

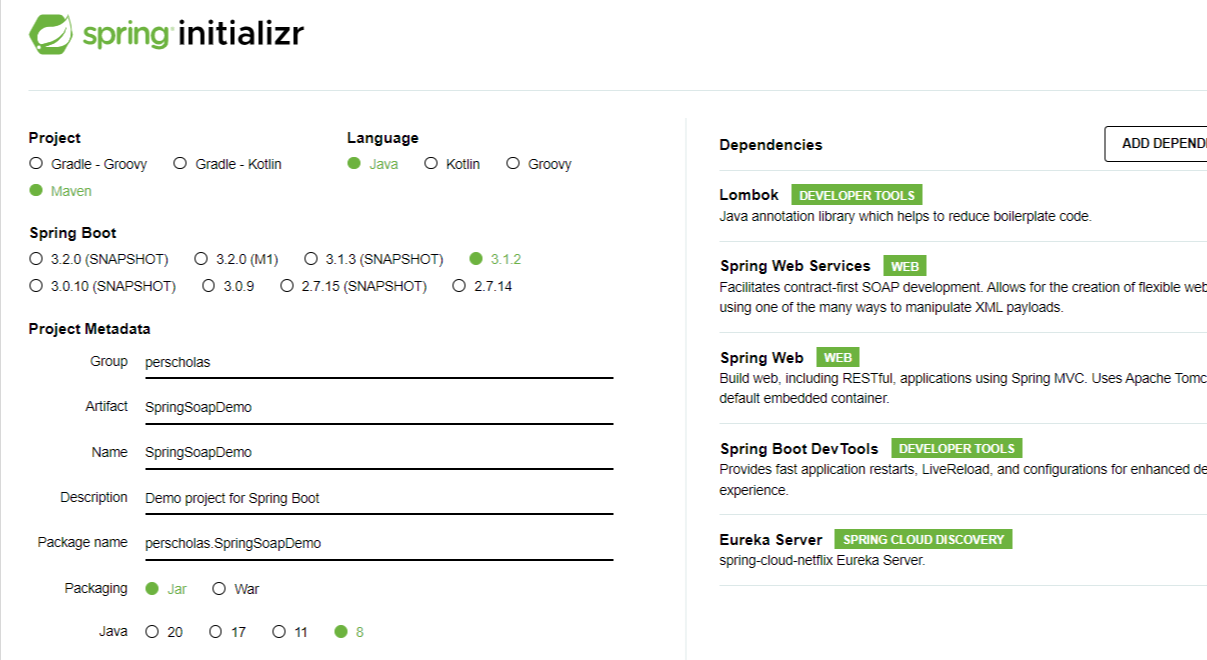
# **Steps to Create Eureka Server and Eureka Clients(microservices):**

1. Eureka Server Application Setup.
2. Service Registration for Eureka Server.
3. Setting application.properties:
   1. Running the Project.
   2. Create Eureka Client- Student Service (Microservices).
   3. Service Registration for Eureka Server.
   4. Setting Application.properties for Client Configuration.
   5. Create a “Student” class is a simple POJO/Entity class.
   6. Create a Controller class.
4. Testing - Eureka Client One- Student Service.
5. Create Eureka Client Two - School Service (Microservices).
6. Eureka Client Two Registration for Eureka Server.
7. Setting Application.properties for Client Configuration.
8. Create a “Schools” class is a simple POJO/Entity class.
9. Create a Controller class.
10. Testing - Eureka Client Two - School Service.

## 

## **Step 1: Eureka Server Application Setup**

## Create one **Spring boot** project from [the **SPRING INITIALIZR**](https://start.spring.io/) site with **Eureka Server (Spring Cloud Discovery).**



Download project in a zipped format. Unzip, and then import the project in IntelliJ / Eclipse as a Maven project. Let us go over the application and its important aspects. This is how our *pom.xm*l look like:

| <?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.5.6</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.perscholas</groupId>  <artifactId>MicroserviceDemo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>MicroserviceDemo</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>2020.0.4</spring-cloud.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>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  </dependency>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  </dependencies>  </project> |
| --- |

#### **Step 2: Service Registration for Eureka Server**

Open the **main method** and define **@EnableEurekaServer** annotation.

You first need a Eureka Service registry. You can use Spring Cloud’s **@EnableEurekaServer** to set up a registry with which other applications can communicate.

| package com.perscholasMicroservice.EurekaServerDemo;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  @SpringBootApplication  @EnableEurekaServer  public class EurekaServerDemoApplication {  public static void main(String[] args) {  SpringApplication.*run*(EurekaServerDemoApplication.class, args);  }  } |
| --- |

#### 

#### For this application, to become a Service Registration, you need to use @EnableEurekaServer annotation. @EnableEurekaServer marks your module as a Eureka Server — basically tying your microservices together.

#### **Step 3: Setting Application.properties:**

Add some properties to the **application.properties** file to handle all of these requirements, as shown below.

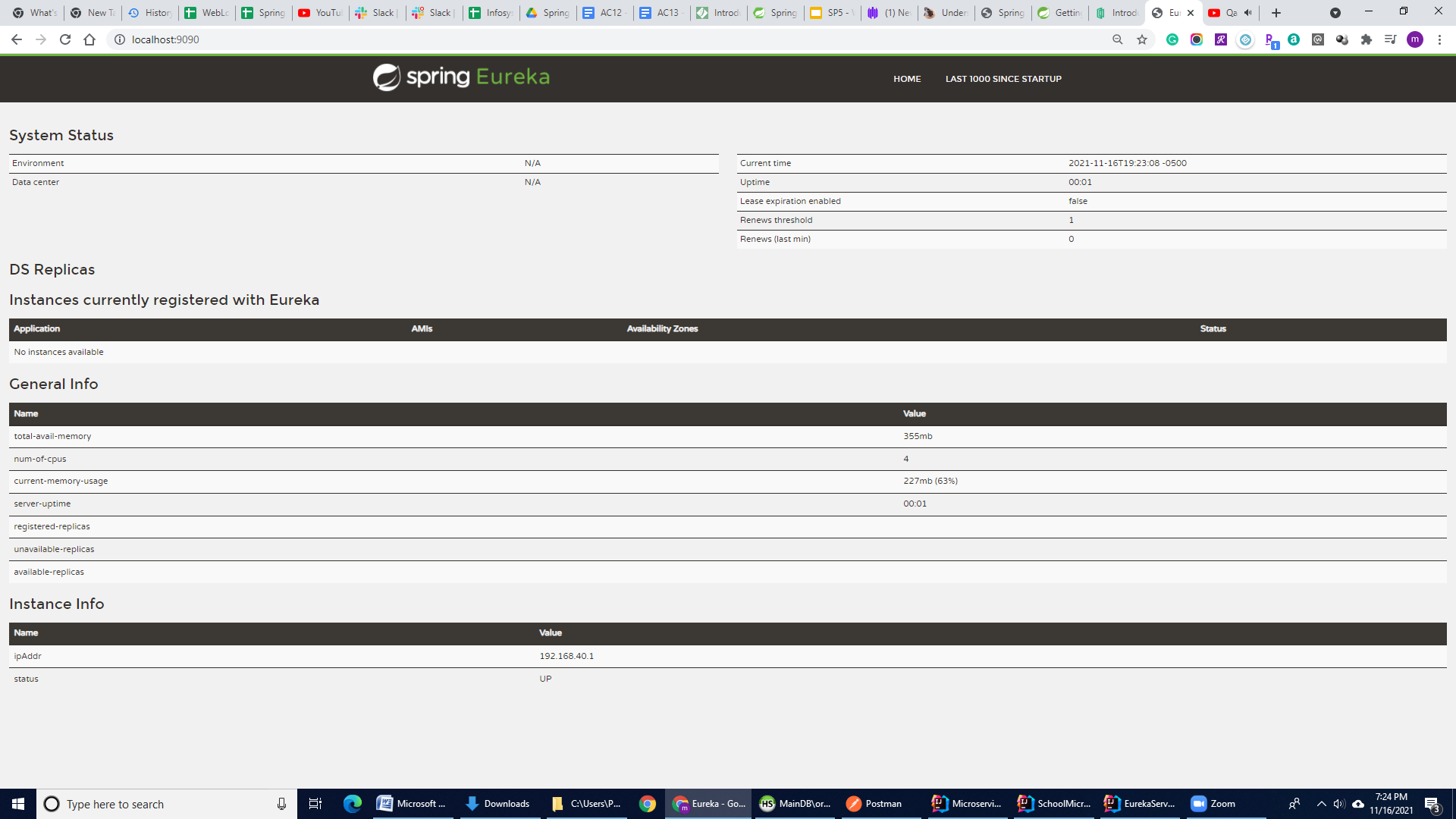
| spring.application.name=eurekaserver  server.port=9090  eureka.client.register-with-eureka=false  eureka.client.fetch-registry=false |
| --- |

**Let’s look at the above properties.**

* Spring.application.name: is a unique name for your application.
* Server.port:Port where the Eureka server will run.
* Register-with-eureka: It will be false as it is the server itself and it does not have to register itself.
* Fetch-registry: this is also client-specific and hence it will be false. The client uses this to fetch the registry information from the server and cache it locally.

**Step 4: Running the Project**

When you run the application and go to the URL ([http://localhost:9090](http://localhost:9090/)) in your browser, you will see the interface as shown in the screenshot below. **But you will notice that we do not have any Eureka clients or microservices listed.**

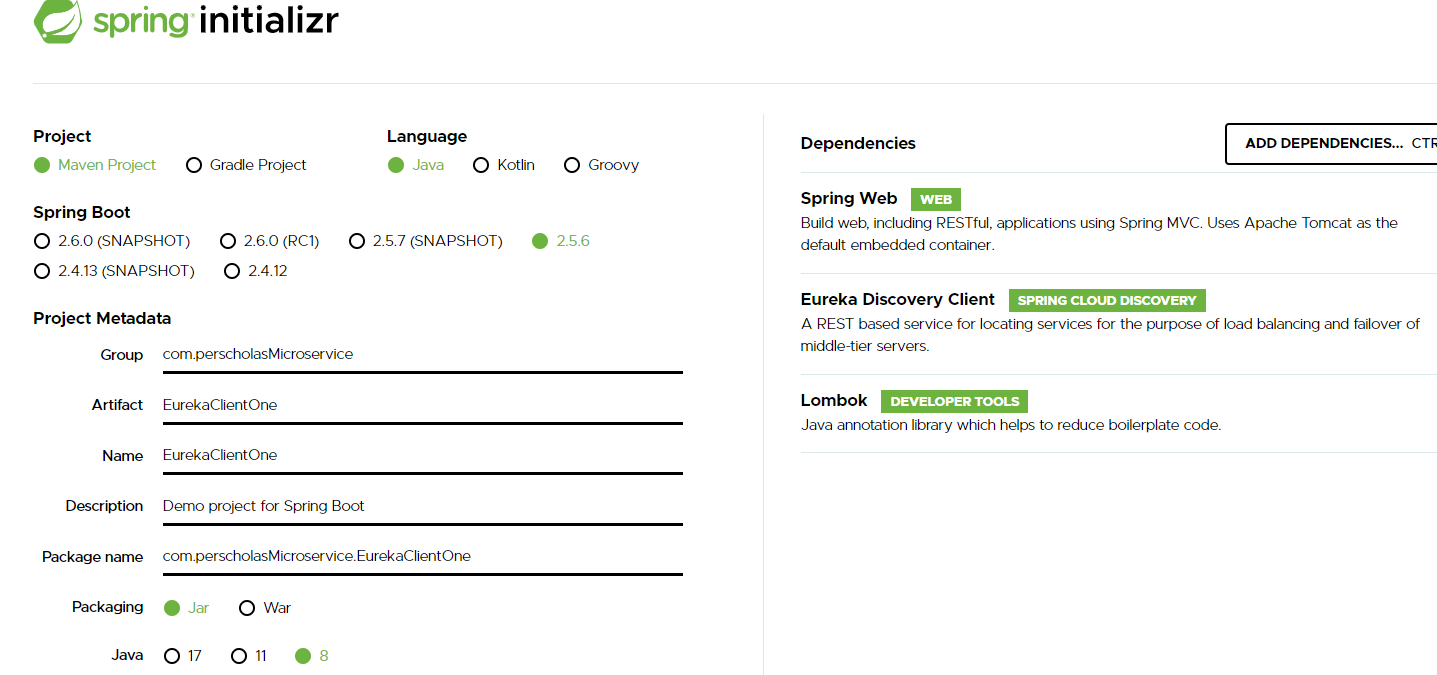


The Eureka Dashboard interface allows you to see the list of active and registered services (microservices) with this Eureka Server. This interface provides an overview of the overall health of the system and any other notifications we may need to know about. One key item to note here is that the instances panel is still empty, which means that we have not created an instance (a microservice) that connects to our discovery.

#### **Step 5: Create Eureka Client- Student Service (Microservices)**

## For the **Eureka Client one Student Service,** we will create another Spring application. This application will have a single endpoint that will return the string *Hello World*. To do so, it will retrieve the words *Hello* and *World* from our microservices.

## Create another **Spring boot** project from [the **SPRING INITIALIZR**](https://start.spring.io/) site with **Eureka client (Eureka Discovery Client) dependency.**



Download the project in a zipped format, and then import the project in IntelliJ / Eclipse as a Maven project. Let us go over the application and its important aspects. This is how our *pom.xm*l will look:

| <?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.5.6</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.perscholasMicroservice</groupId>  <artifactId>EurekaClientOne</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>EurekaClientOne</name>  <description>Demo project for Spring Boot</description>  <properties>  <java.version>1.8</java.version>  <spring-cloud.version>2020.0.4</spring-cloud.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-netflix-eureka-client</artifactId>  </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>  </dependencies>  <dependencyManagement>  <dependencies>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>${spring-cloud.version}</version>  <type>pom</type>  <scope>import</scope>  </dependency>  </dependencies>  </dependencyManagement>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <excludes>  <exclude>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </exclude>  </excludes>  </configuration>  </plugin>  </plugins>  </build>  </project> |
| --- |

#### Step 6: Service Reg**istration for Eureka Server .**

Open the **main method** and add the [***@EnableEurekaClient***](https://github.com/spring-cloud/spring-cloud-netflix/blob/master/spring-cloud-netflix-eureka-client/src/main/java/org/springframework/cloud/netflix/eureka/EnableEurekaClient.java) annotation. With this annotation, this artifact will act like a spring discovery client and will register itself in the Eureka server attached to this service.

| import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  @SpringBootApplication  **@**[**EnableEurekaClient**](https://github.com/spring-cloud/spring-cloud-netflix/blob/master/spring-cloud-netflix-eureka-client/src/main/java/org/springframework/cloud/netflix/eureka/EnableEurekaClient.java)  public class EurekaClientOneApplication {  public static void main(String[] args) {  SpringApplication.*run*(EurekaClientOneApplication.class, args);  }  } |
| --- |

#### Step 7: Setting Application.properties for Client Configuration

| spring.application.name=School-Microservice-clientOne  server.port=8081  eureka.client.service-url.defaultZone=http://localhost:9090/eureka/  eureka.instance.lease-renewal-interval-in-seconds=30 |
| --- |

* **Spring.application.name:** is a unique name for your client application.
* **Server.port:** Port where the Eureka Client will run.
* **Lease-renewal-interval-in-seconds:** It is the interval post, which the client will send the heartbeat to the server. But it is optional. **The default value is 30 seconds, which means that the client will send one heartbeat every 30 seconds.** But this flag is optional.
* **eureka.client.service-url.defaultZone=**[**http://localhost:9090/eureka/**](http://localhost:2000/eureka/)**:** This configuration is very important because of our **Microservice (Eureka clients). Eureka clients** will try to register or communicate to **Eureka Server on** port **8761 by default,** but our **Eureka Server** is running on port **9090**. If we do not specify Eureka Server running port (9090), it will cause an error.

#### Step 8: Create a Student Class is a simple POJO/Entity class.

**Student.java**

| package com.perscholasMicroservice.EurekaClientOne; public class Student {  private int id;  private String name;  private String className;   public Student(String name, String className) {  super();  this.name = name;  this.className = className;  }  public Student() {   }  public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }  public String getClassName() {  return className;  }  public void setClassName(String className) {  this.className = className;  } } |
| --- |

## 

## **Step 9: Create Controller Class**

Let’s define a controller class for our Eureka client by the name of **StudentServiceController.java**. This is a simple [Spring Boot REST controller](https://www.javadevjournal.com/spring-boot/spring-boot-rest-example/), and it defines a few endpoints to provide **Students** information to the calling client.

**StudentServiceController.java**

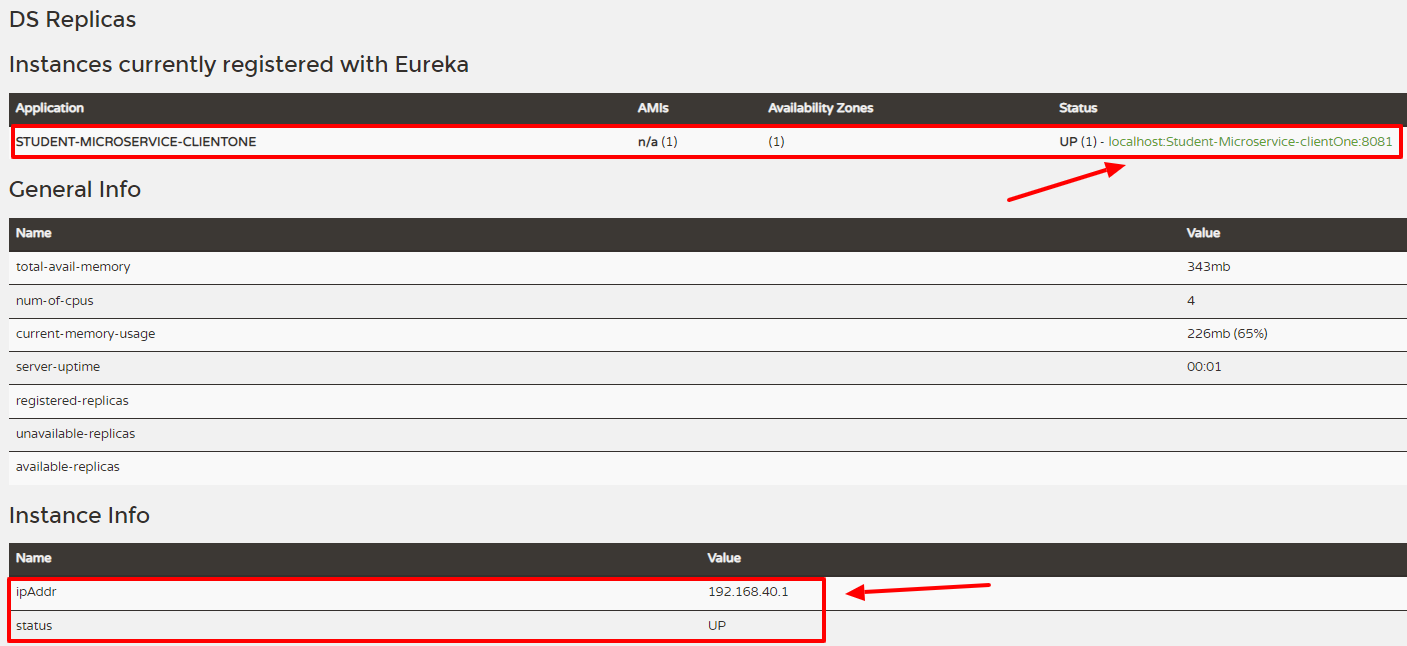
| package com.perscholasMicroservice.EurekaClientOne; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.PathVariable; import org.springframework.web.bind.annotation.RestController; import java.util.ArrayList; import java.util.List;  @RestController public class StudentServiceController {  @GetMapping("/liststudent/{classname}")  public List<Student> getStudents1(@PathVariable String classname) {  String classnames = classname;  // Assume that below list of student is coming from Database, later you can convert this into Spring Data Rest  ArrayList<Student> schooDB = new ArrayList<Student>();  List<Student> lst = new ArrayList<Student>();  Student std1 = new Student("Adam", classnames);  Student std2 = new Student("Tom", classnames);  Student std3 = new Student("Fink", classnames);  Student std4 = new Student("James", classnames);  lst.add(std1);  lst.add(std2);  lst.add(std3);  lst.add(std4);   return lst;  } } |
| --- |

## 

## **Step 10: Testing - Eureka Client One- Student Service**

Now, verify that this **Student Service** has been registered in the Eureka Server automatically. Go to the browser and type **Eureka Server** URL:[**http://localhost:**](http://localhost:9000/)**9090** and in your browser once both **Eureka Server and Eureka Client One- Student Service** are up and running. You can see our [**Student-Microservice-clientOne**](http://192.168.40.1:8081/actuator/info) is listed/registered under the Application section and with their IP addresses and ports.

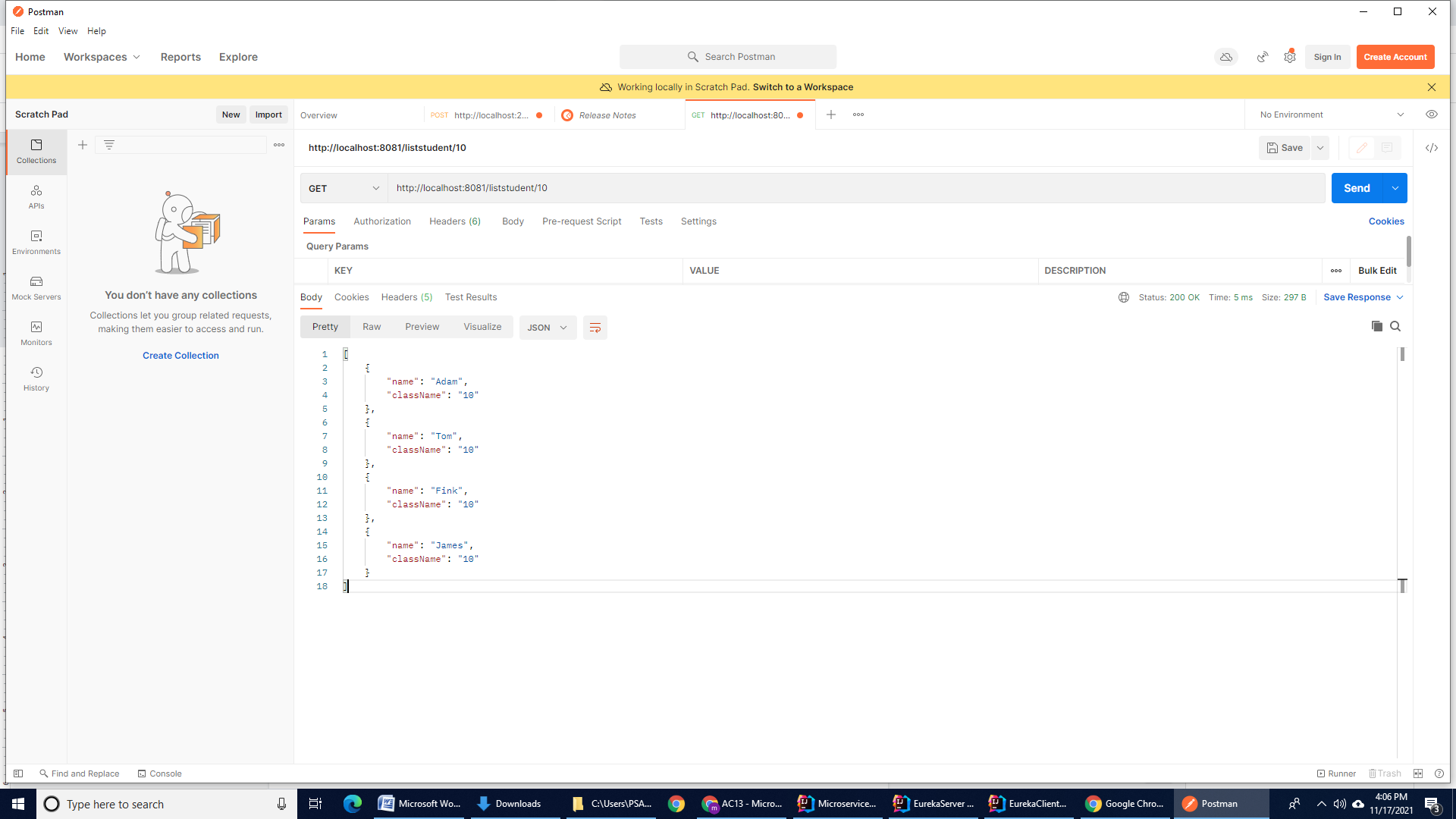
This indicates that both the Eureka server and client are aware of each other.



We will now verify that the /liststudent/{classname} **endpoint** for **Eureka Client- Student Service** is up and running.

Go to the POSTMAN and Type URL [**http://localhost:8081/liststudent/10**](http://localhost:8081/liststudent/10)

It will give the student details for a particular class (10).

****

Note: 10 is just a student’s class number. You can put/use any other number rather than 10.

You can see that the data is hardcoded in the controller class. Later, you can use either Spring Rest Data or Spring JPA for dynamic application.

## 

#### 

## **Step 11: Create Eureka Client Two - School Service (Microservices)**

## 

## For the **Eureka Client Two - School Service,** we will create another Spring application.

## Create a third **Spring boot** project from [the **SPRING INITIALIZR**](https://start.spring.io/) site with **Eureka client (Eureka Discovery Client) dependency as we created for previous client applications.**

Download project in zipped format from [**SPRING INITIALIZR**](https://start.spring.io/). Unzip, and then import the project in IntelliJ / Eclipse as a Maven project.

#### **Step 12: Eureka Client Two Registration for Eureka Server**

Open the **main method and** add the [@EnableEurekaClient](https://github.com/spring-cloud/spring-cloud-netflix/blob/master/spring-cloud-netflix-eureka-client/src/main/java/org/springframework/cloud/netflix/eureka/EnableEurekaClient.java) annotation. With this annotation, this artifact will act like a Spring discovery client and will register itself in the Eureka server attached to this service.

| import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.cloud.netflix.eureka.EnableEurekaClient;  @SpringBootApplication  @EnableEurekaClient  public class EurekaClientTwoApplication {  public static void main(String[] args) {  SpringApplication.*run*(EurekaClientTwoApplication.class, args);  }  } |
| --- |

#### **Step 13: Setting Application.properties for** **Client Configuration**

| spring.application.name=School-Microservice-clientTwo  server.port=8082  eureka.client.service-url.defaultZone=http://localhost:9090/eureka/  eureka.instance.lease-renewal-interval-in-seconds=30 |
| --- |

**Step 14: Create a class named *“Schools”*, it is a simple POJO/Entity Class**

**Schools.java**

| import java.util.UUID; public class Schools {  private UUID id;  private String Schoolname;  private String address;   public Schools(UUID id, String schoolname, String address) {  this.id = id;  this.Schoolname = schoolname;  this.address = address;  }   public UUID getId() {  return id;  }  public void setId(UUID id) {  this.id = id;  }   public String getSchoolname() {  return Schoolname;  }   public void setSchoolname(String schoolname) {  Schoolname = schoolname;  }   public String getAddress() {  return address;  }   public void setAddress(String address) {  this.address = address;  }  @Override  public String toString() {  return "Schools{" +  "id=" + id +  ", Schoolname='" + Schoolname + '\'' +  ", address='" + address + '\'' +  '}';  } } |
| --- |

#### Step 15: Create Controller Class

Let’s define a controller class for our Eureka client by the name of **SchoolServiceController.java**. This is a simple [Spring Boot REST controller](https://www.javadevjournal.com/spring-boot/spring-boot-rest-example/), and defines a few endpoints to provide **Schools** information to the calling client.

**SchoolServiceController**

| package com.perscholasMicroservice.EurekaClientOne; import org.springframework.http.HttpStatus; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.\*; import org.springframework.web.client.RestTemplate; import java.util.ArrayList; import java.util.List; import java.util.UUID;  @RestController public class SchoolServiceController {   private static List SchoolList = new ArrayList<>();  static{  SchoolList = new ArrayList<>();  Schools s1 = new Schools(UUID.randomUUID(), "PerscholasNY" , "NYC");  Schools s2 = new Schools(UUID.randomUUID(), "PerscholasNJ" , "NJ");  Schools s3 = new Schools(UUID.randomUUID(), "PerscholasDA" , "Dallas");  Schools s4 = new Schools(UUID.randomUUID(), "PerscholasAT" , "Atlanta ");  Schools s5 = new Schools(UUID.randomUUID(), "PerscholasBA" , "Baltimore ");  Schools s6 = new Schools(UUID.randomUUID(), "PerscholasBA" , "Boston ");  Schools s7 = new Schools(UUID.randomUUID(), "PerscholasDE" , "Detroit ");   SchoolList.add(s1);  SchoolList.add(s2);  SchoolList.add(s3);  SchoolList.add(s4);  SchoolList.add(s5);  SchoolList.add(s6);  SchoolList.add(s7);  }  @GetMapping("/ClientTwo-StudentService")  public ResponseEntity callEurekaClient1(){  return new ResponseEntity("Hello From Eureka Client Two- Student Service 2", HttpStatus.OK);  }   @GetMapping("/addschool/{id}")  public String showProduct(@PathVariable UUID id) {  Schools newSchool = new Schools(id, "PerscholasPHI" , "Philadelphia ");  return "Schools is going to add --> " + newSchool.toString();  }   @GetMapping("/allschools")  public List list() {  return SchoolServiceController.SchoolList;  }  } |
| --- |

## 

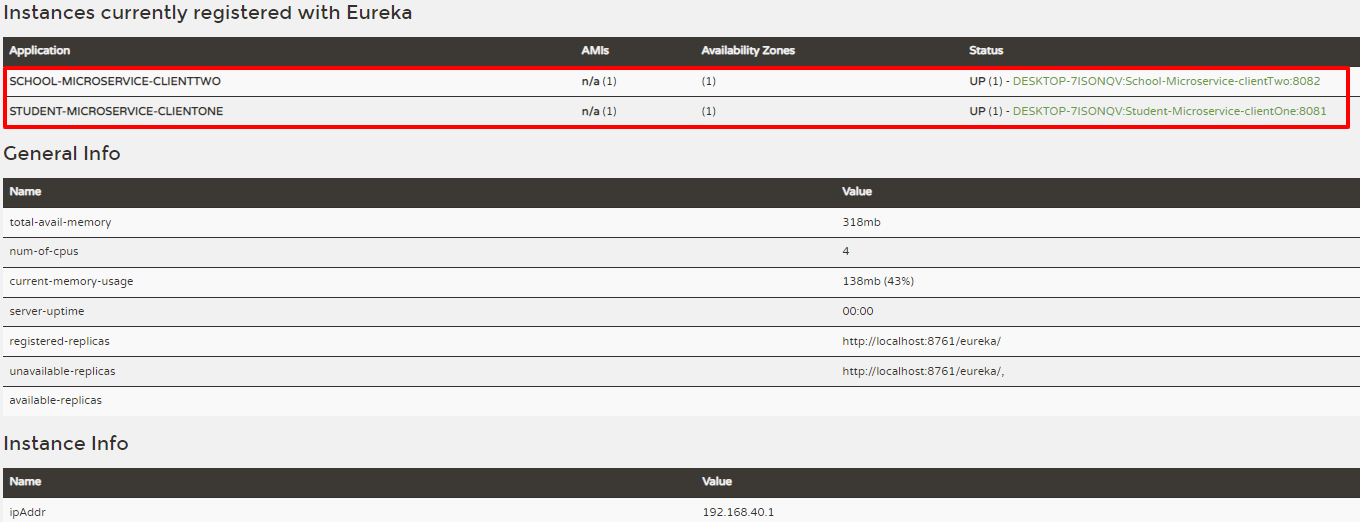
## 

## 

## **Step 16: Testing - Eureka Client Two - School Service**

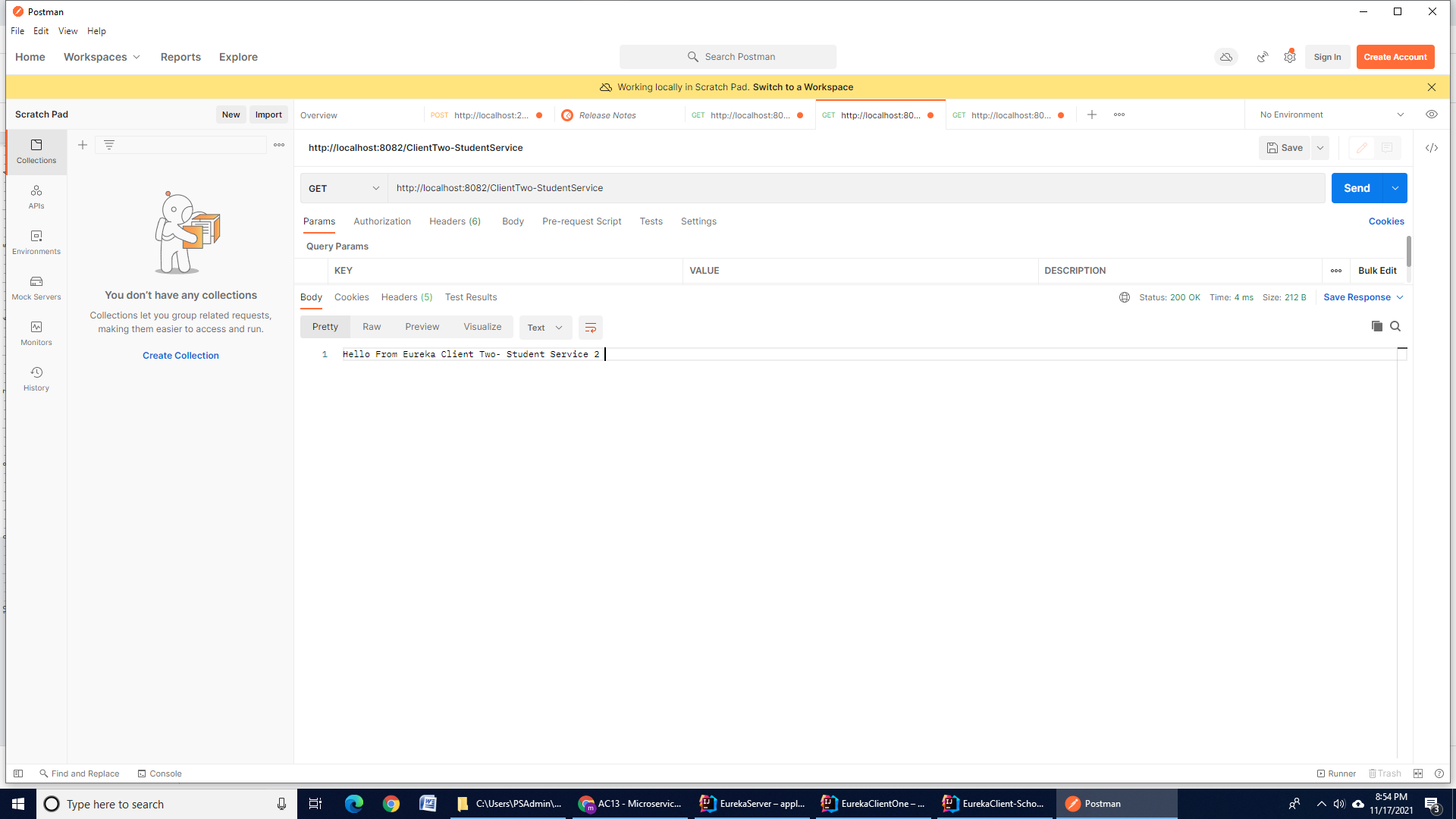
Now, verify that this **School Service** has been registered in the Eureka Server automatically. Go to the browser and type **Eureka Server** URL:[**http://localhost:**](http://localhost:9000/)9090. In your browser, once both **Eureka Server and Eureka Client Two School Service** are up and running, you can see both [**Student-Microservice-clientOne**](http://192.168.40.1:8081/actuator/info) and [**School-Microservice-clientTwo:8082**](http://192.168.40.1:8082/actuator/info)are listed/registered under Application section and with their IP addresses and ports.

This indicates that both the Eureka server and clients are aware of each other.

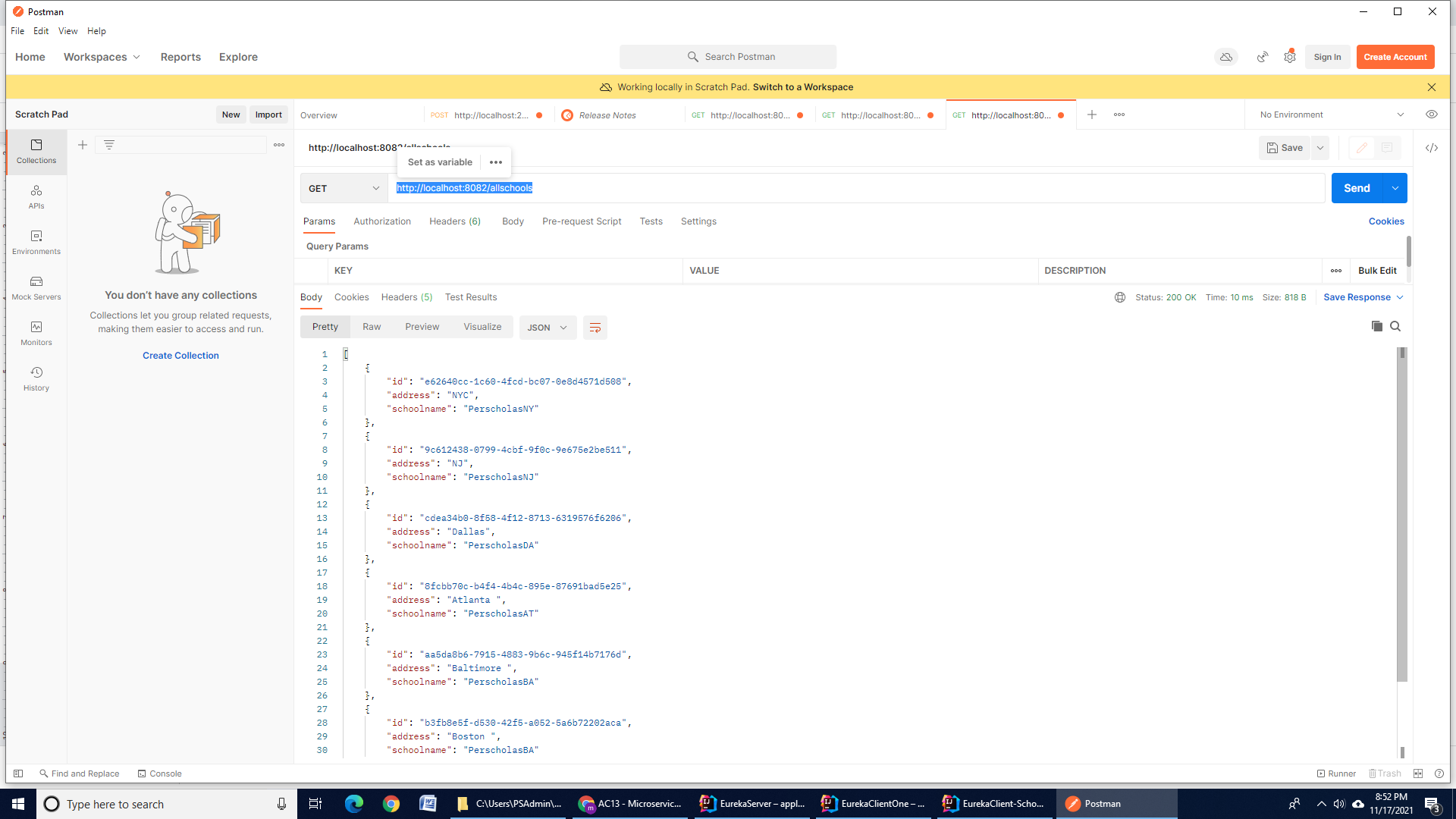


We will now verify that all of the **endpoints** for **Eureka Client two - Schools Service** are up and running.

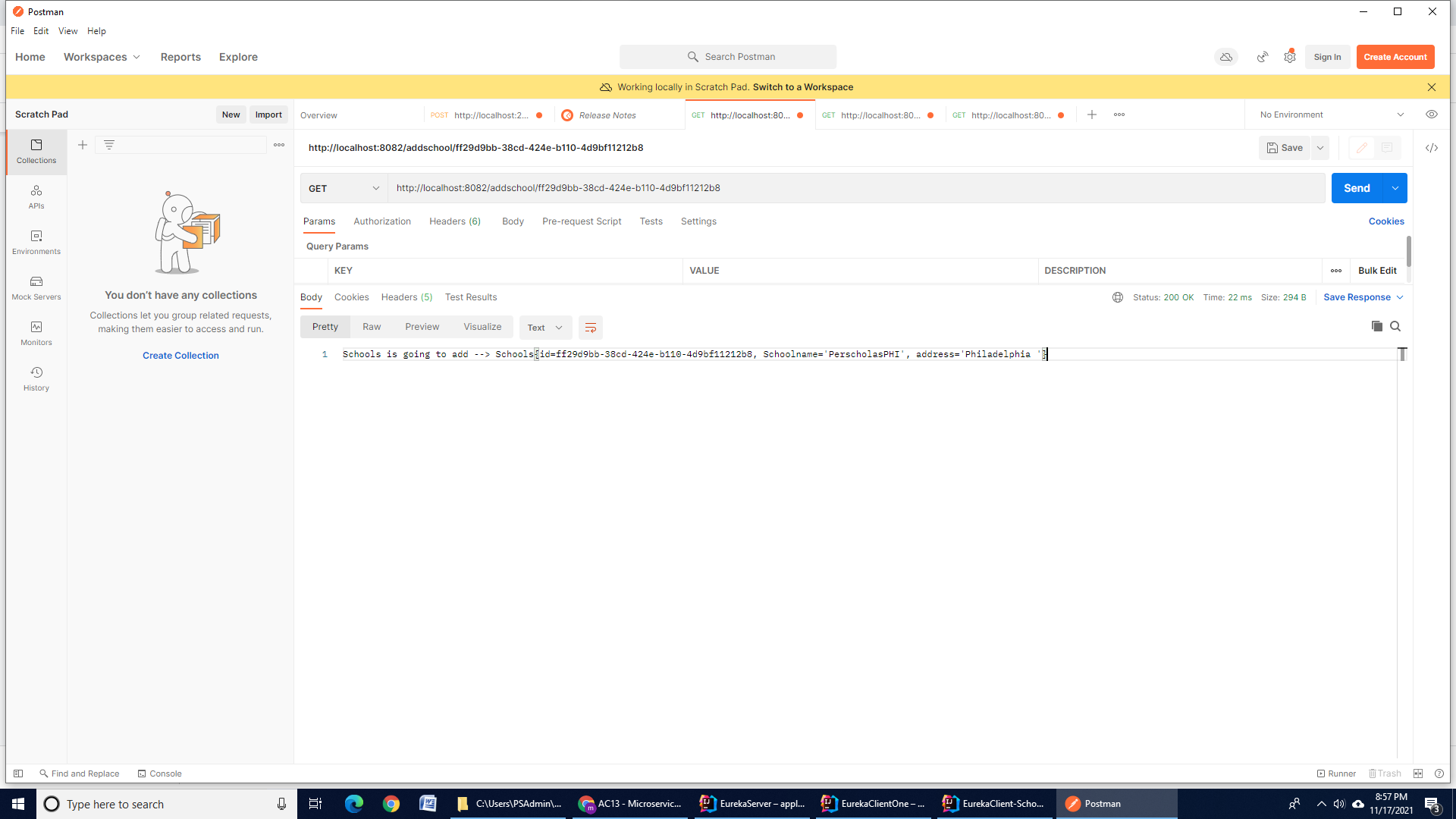
Go to the POSTMAN and type the URL: [**http://localhost:8082/ClientTwo-StudentService**](http://localhost:8082/ClientTwo-StudentService)



Go to the POSTMAN and type the URL: **http://localhost:8082/allschools**

****

Go to the POSTMAN and type URL: **http://localhost:8082/addschool/ff29d9bb-38cd-424e-b110-4d9bf11212b8**

****

**References**

<https://www.baeldung.com/eureka-self-preservation-renewal>

<https://www.section.io/engineering-education/introduction-to-microservices-with-spring-and-eureka/>

<https://www.javadevjournal.com/spring-boot/spring-cloud-netflix-eureka/>

**Submission Instructions:**

Include the following deliverables in your submission -

* Submit your code or screenshots using the Start Assignment button in the top-right corner of the assignment page in Canvas.