**Microservices**

* In this document we’re going see complete Microservice architecture in spring boot.
* So we will be creating couple of Microservices and we will be creating service registry and we will connect those Microservice to service registry.
* We will create API Gateway so all over request traverse through an API Gateway.
* We will create a Microservices, circuit breaker and also creating distributed log tracing as well.
* Let’s start creating the our Microservices
* This is the complete architecture we’re going to build it today.
* So we will be creating two Microservices i.e. Departmentservice and Userservice.
* Every user will be tag to Department. So we will be saving from user from one service and saving Department from one service.
* We will create a couple of methods to get the Department and to get the user information as well.
* We will create a method to fetch the user with its Department so we will have a call from Userservice to Departmentservice.

CREATING Departmentservice:

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.4**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

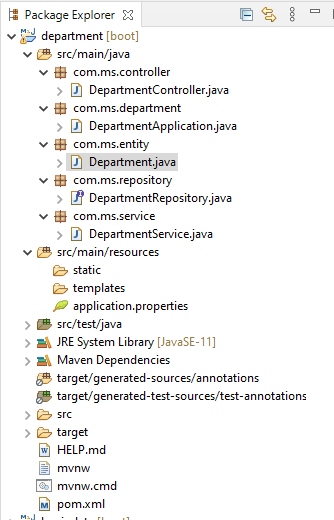
**Step 4:** Provide the **Artifact** Id. We have provided **department**

**Step 5:** Add the dependencies **Spring Web, Spring Data JPA,**

**H2 database, Lombok.**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application. The project structure be like:



**DepartmentApplication.java**

package com.ms.department;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication

@ComponentScan({"com.ms.Controller", "com.ms.Service"})

@EntityScan("com.ms.entity")

@EnableJpaRepositories ("com.ms.Repository")

public class DepartmentApplication {

public static void main (String[] args) {

SpringApplication.run (DepartmentApplication. Class, args);

}

}

**DepartmentController.java**

package com.ms.controller;

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

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

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

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

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

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

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

import com.ms.entity.Department;

import com.ms.service.DepartmentService;

@RestController

@RequestMapping("/departments/")

public class DepartmentController {

@Autowired

private DepartmentService departmentService;

@PostMapping("/save")

public Department saveDepartment(@RequestBody Department department) {

return departmentService.saveDepartment(department);

}

@GetMapping("/{id}")

public Department findDepartmentById(@PathVariable("id") Long departmentId) {

return departmentService.findDepartmentById(departmentId);

}

}

**DepartmentService.java**

package com.ms.service;

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

import org.springframework.stereotype.Service;

import com.ms.entity.Department;

import com.ms.repository.DepartmentRepository;

import lombok.extern.slf4j.Slf4j;

@Service

@Slf4j

public class DepartmentService {

@Autowired

private DepartmentRepository departmentRepository;

public Department saveDepartment(Department department) {

return departmentRepository.save(department);

}

public Department findDepartmentById(Long departmentId) {

return departmentRepository.findByDepartmentId(departmentId);

}

}

**DepartmentRepository**

package com.ms.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.ms.entity.Department;

@Repository

public interface DepartmentRepository extends JpaRepository<Department, Long> {

Department findByDepartmentId(Long departmentId);

}

**Application.properties**

server.port=9001

**Entity:**

package com.ms.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long departmentId;

private String departmentName;

private String departmentAddress;

private String departmentCode;

public Long getDepartmentId() {

return departmentId;

}

public void setDepartmentId(Long departmentId) {

this.departmentId = departmentId;

}

public String getDepartmentName() {

return departmentName;

}

public void setDepartmentName(String departmentName) {

this.departmentName = departmentName;

}

public String getDepartmentAddress() {

return departmentAddress;

}

public void setDepartmentAddress(String departmentAddress) {

this.departmentAddress = departmentAddress;

}

public String getDepartmentCode() {

return departmentCode;

}

public void setDepartmentCode(String departmentCode) {

this.departmentCode = departmentCode;

}

public Department() {

}

}

**pom.xml**

<?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>2.4.5</version>

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

</parent>

<groupId>com.ms</groupId>

<artifactId>department</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>department</name>

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

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</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>

<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>

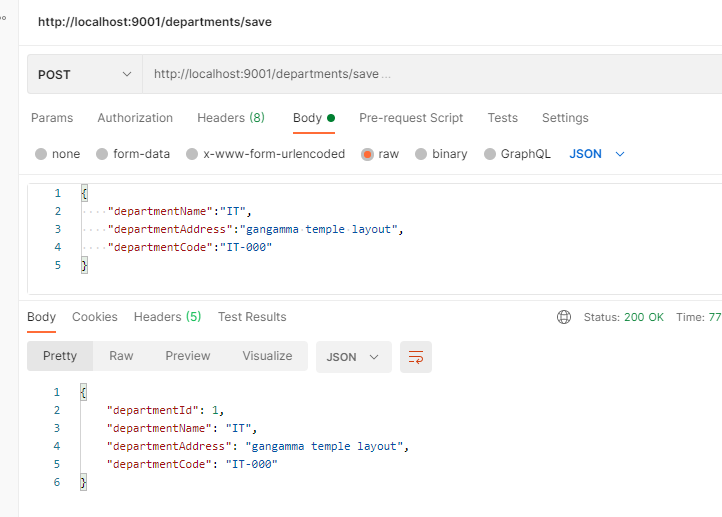
</plugins>

</build>

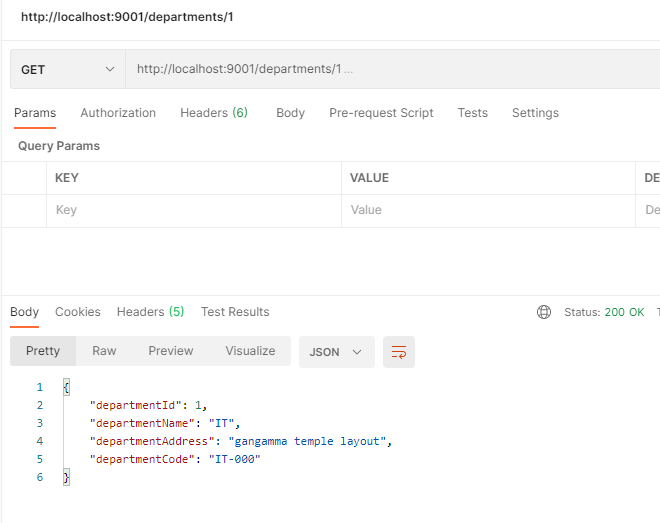
</project>

**Results:**

Post method: input &output



Get method:



CREATING Userservice:

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.4**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

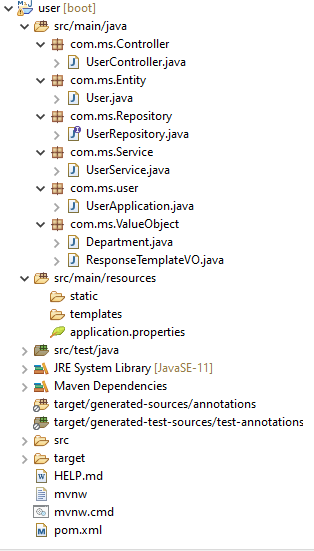
**Step 4:** Provide the **Artifact** Id. We have provided **user**

**Step 5:** Add the dependencies **Spring Web, Spring Data JPA,**

**H2 database, Lombok.**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application. The project structure be like:



**userApplication.java**

package com.ms.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

import org.springframework.web.client.RestTemplate;

@SpringBootApplication

@ComponentScan({"com.ms.Controller", "com.ms.Service"})

@EntityScan("com.ms.Entity")

@EnableJpaRepositories("com.ms.Repository")

public class UserApplication {

public static void main(String[] args) {

SpringApplication.run(UserApplication.class, args);

}

@Bean

RestTemplate restTemplate() {

return new RestTemplate();

}

}

**UserController.java**

package com.ms.Controller;

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

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

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

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

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

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

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

import com.ms.Entity.User;

import com.ms.Service.UserService;

import com.ms.ValueObject.ResponseTemplateVO;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@PostMapping("/save")

public User saveUser(@RequestBody User user) {

return userService.saveUser(user);

}

@GetMapping("/{id}")

public ResponseTemplateVO getUserwithDepartment(@PathVariable("id") Long userId) {

return userService.getUserwithDepartment(userId);

}

}

**UserService.java**

package com.ms.Service;

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

import org.springframework.stereotype.Service;

import org.springframework.web.client.RestTemplate;

import com.ms.Entity.User;

import com.ms.Repository.UserRepository;

import com.ms.ValueObject.Department;

import com.ms.ValueObject.ResponseTemplateVO;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

@Autowired

private RestTemplate restTemplate;

public User saveUser(User user) {

return userRepository.save(user);

}

public ResponseTemplateVO getUserwithDepartment(Long userId) {

ResponseTemplateVO vo = new ResponseTemplateVO();

User user = userRepository.findByUserId(userId);

Department department =

restTemplate.getForObject("http://localhost:9001/departments/" + user.getDepartmentId(), Department.class);

vo.setUser(user);

vo.setDepartment(department);

return vo;

}}

**UserRepository**

package com.ms.Repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.ms.Entity.User;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

User findByUserId(Long userId);

}

**Entity**

package com.ms.Entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long userId;

private String firstName;

private String lastName;

private String email;

private String departmentId;

public Long getUserId() {

return userId;

}

public void setUserId(Long userId) {

this.userId = userId;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getDepartmentId() {

return departmentId;

}

public void setDepartmentId(String departmentId) {

this.departmentId = departmentId;

}

public User() {

}

}

**Department.java**

package com.ms.ValueObject;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Department {

private Long departmentId;

private String departmentName;

private String departmentAddress;

private String departmentCode;

public Long getDepartmentId() {

return departmentId;

}

public void setDepartmentId(Long departmentId) {

this.departmentId = departmentId;

}

public String getDepartmentName() {

return departmentName;

}

public void setDepartmentName(String departmentName) {

this.departmentName = departmentName;

}

public String getDepartmentAddress() {

return departmentAddress;

}

public void setDepartmentAddress(String departmentAddress) {

this.departmentAddress = departmentAddress;

}

public String getDepartmentCode() {

return departmentCode;

}

public void setDepartmentCode(String departmentCode) {

this.departmentCode = departmentCode;

}

public Department() {

}

}

**ResponseTemplateVO.java**

package com.ms.ValueObject;

import com.ms.Entity.User;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class ResponseTemplateVO {

private User user;

private Department department;

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public Department getDepartment() {

return department;

}

public void setDepartment(Department department) {

this.department = department;

}

public ResponseTemplateVO() {

}

}

**Application.properties**

server.port=9002

**pom.xml**

<?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>2.4.5</version>

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

</parent>

<groupId>com.ms</groupId>

<artifactId>user</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>user</name>

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

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</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>

<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>

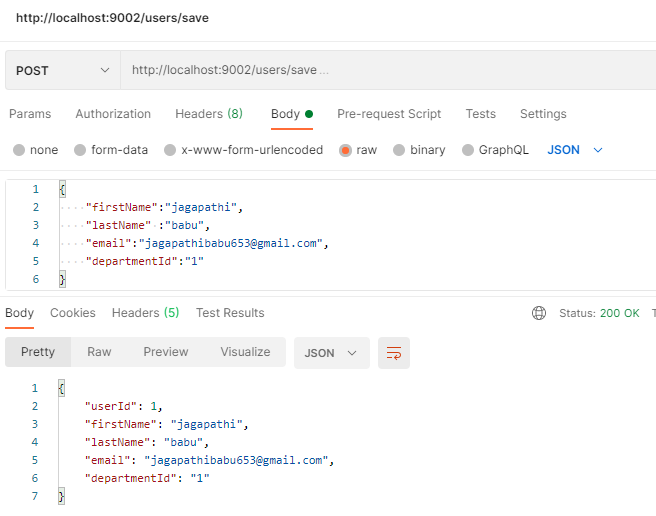
</configuration>

</plugin>

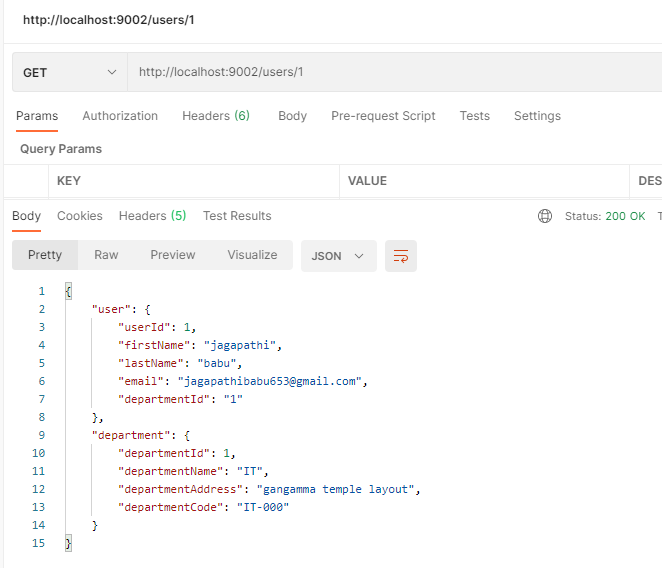
</plugins>

</build>

</project>

**Results:** Post method: input &output

Get method: input &output



* We’re calling another Microservice from our Userservice. So this way we connect and call the different Microservices from one Microservices.
* So we can see that it is decoupled. user object and department object are related to each other but they have different Microservices for their operations and we are calling each other using the RESTApi call and we’re getting the over here.
* Now considering we have a multiple Microservices like hundreds and thousands of Microservices then to manage all the Microservices it is be very difficult for ours. Now we have only two Microservice so we can think okay it is very easy.
* When ours application grows and vast state and we have thousands of Microservices then to maintain all those Microservice to get the port details and the url information it will be very difficult.
* So we want something that can manage all this Microservices. So it will be use in future. So all this Microservices and ports are maintaining is very difficult.
* We will create serviceregistry that maintain all the Microservices for it.
* We will create a serviceregistry that connected to Userservice and Departmentservice.

CREATING service-registry:

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.4**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

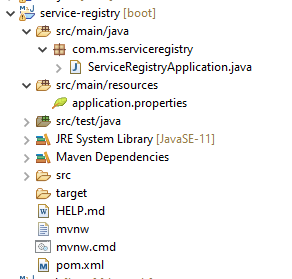
**Step 4:** Provide the **Artifact** Id. We have provided **service-registry**

**Step 5:** Add the dependencies **eureka server**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application.

The project structure be like:



**ServiceRegistryApplication.java:**

package com.ms.serviceregistry;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableEurekaServer

public class ServiceRegistryApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRegistryApplication.class, args);

}

}

**Application.properties:**

server.port=8761(default)

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

Because this my server so it doesn’t have to go to eureka server for the registering itself so I’m keeping it false now.

**pom.xml:**

<?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>2.4.5</version>

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

</parent>

<groupId>com.ms</groupId>

<artifactId>service-registry</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>service-registry</name>

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

<properties>

<java.version>11</java.version>

<spring-cloud.version>2020.0.2</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-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>${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>

</plugin>

</plugins>

</build>

</project>

* Now what we have to do is for the other Userservice and Departmentservice we have to implement the eureka client over there, and there we need to give the eureka client register with eureka is true and fetch registry is true, so that those two can be consider as a client then they can go head connected to the eureka server.
* So we need to add dependency is eureka discovery client in Userservice and Departmentservice.

**Pom.xml:**

<properties>

<java.version>11</java.version>

<spring-cloud.version>2020.0.2</spring-cloud.version>

</properties>

<dependency>

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

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

</dependency>

<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>

* Now we need to do the configuration, so that services go head and connect to eureka server
* After that we add in application.properties in Userservice and Departmentservice for configuration.

**User application.properties:**

server.port=9002

spring.application.name= user

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

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

eureka.client.instance.hostname = localhost

**Department application.properties:**

server.port=9001

spring.application.name= department

eureka.client.register-with-eureka=true

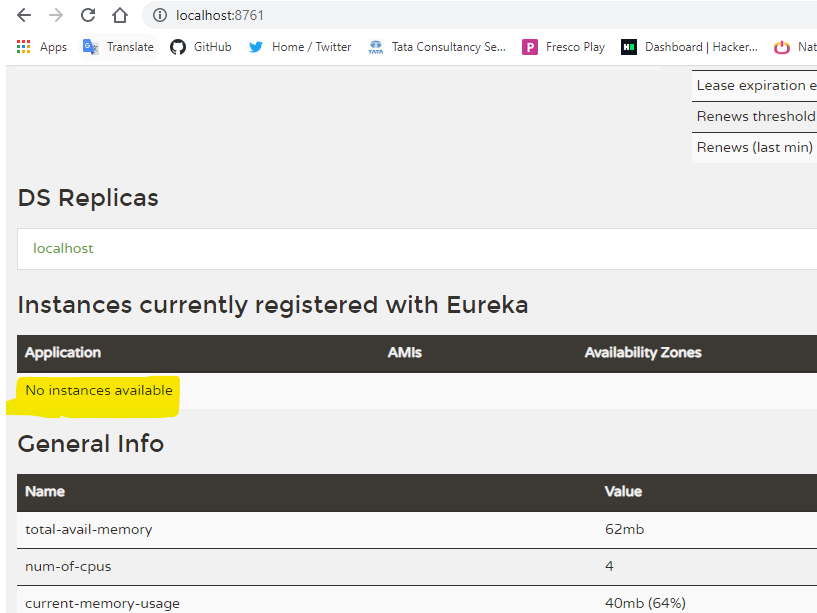
eureka.client.fetch-registry=true

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

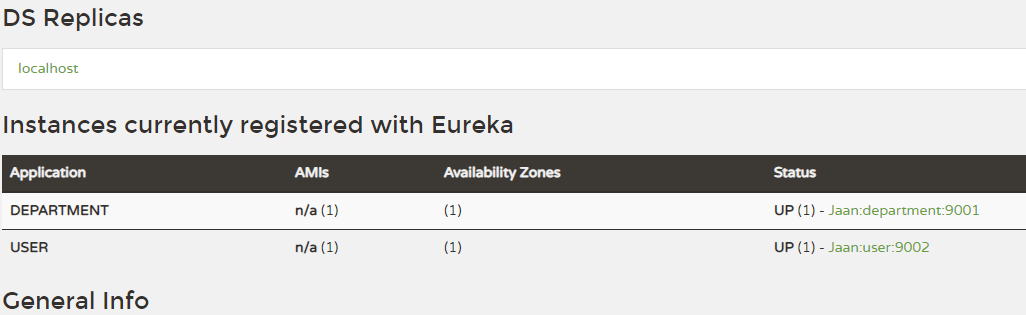
eureka.client.instance.hostname = localhost

* So this will connect to eureka server.
* We have given here eureka.client.register-with-eureka is true, eureka.client.fetch-registry is true and default port is 8761 what we given in the service registry.

**Now we run the service registry application.**

****

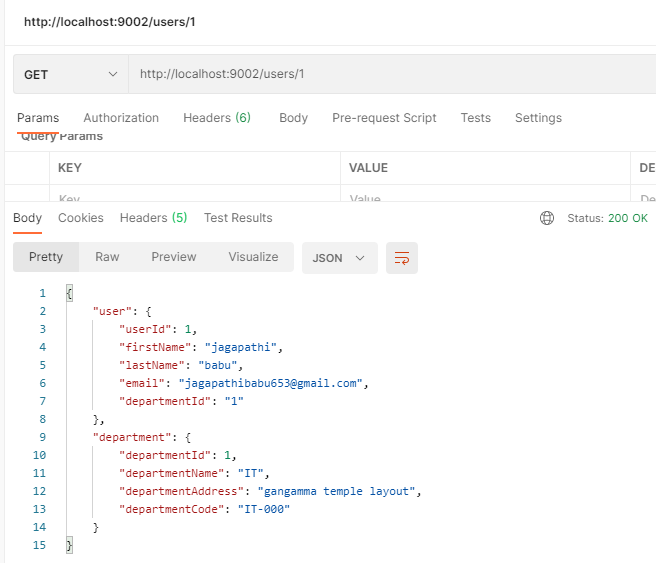
Now we can see no instances is available that means no service is connected to eureka. So now what we’ll do is start the user and Departmentservice.



So my department and user service is connected to eureka server.

In user Userservice we have to change the localhost: 9001 to Department. Ever this application is working any port or any service if it’s connected to service registry then I should get the information for the particular thing.

**Result:**

****

* So we see that we’re getting the user and department.
* So now what’s it’s doing is directly getting the department from the DEPARTMENT application. Not from the hostname and port information.
* So this was service registry helping us to identify the all Microservices are running and we can directly connect to different Microservices.
* Now create the API- GATEWAY, now currently we can see that all requests are directly going to the Microservices.
* So we create one API- GATEWAY from there only all the request should be traverse to the particular Microservices.
* All requests come from API-GATEWAY there only it traverses through particular Microservices based on URL pattern.

CREATING cloud-gateway:

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.4**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

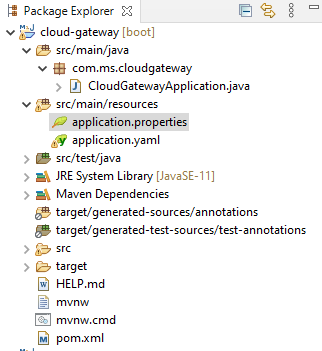
**Step 4:** Provide the **Artifact** Id. We have provided **cloud-gateway**

**Step 5:** Add the dependencies **eureka discovery client, gateway, spring boot actuator.**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application.

The project structure be like:



**CloudGatewayApplication.java:**

**package** com.ms.cloudgateway;

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

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

**import** org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

**public** **class** CloudGatewayApplication {

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

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

}

}

**Application.yaml:**

server:

port: 9191

eureka:

client:

register-with-eureka: true

fetch-registry: true

service-url:

defaultzone: http://localhost:8761/eureka/

instance:

hostname: localhost

spring:

application:

name: API-GATEWAY

cloud:

gateway:

routes:

- id: USER

uri:

lb://USER

predicates:

- Path=/users/\*\*

- id: DEPARTMENT

uri:

lb://DEPARTMENT

predicates:

- Path=/departments/\*\*

**Pom.xml:**

<?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>2.4.5</version>

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

</parent>

<groupId>com.ms</groupId>

<artifactId>cloud-gateway</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>cloud-gateway</name>

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

<properties>

<java.version>11</java.version>

<spring-cloud.version>2020.0.2</spring-cloud.version>

</properties>

<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-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>

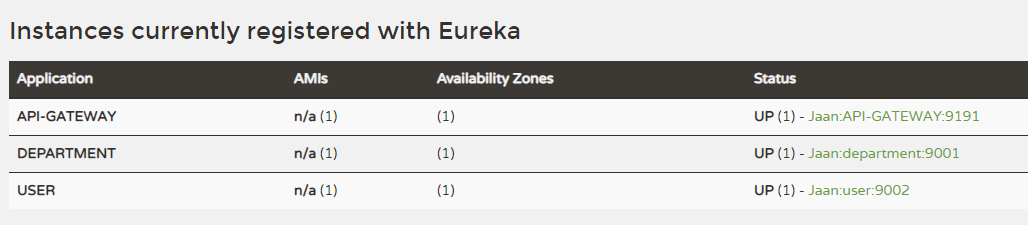
</plugin>

</plugins>

</build>

</project>

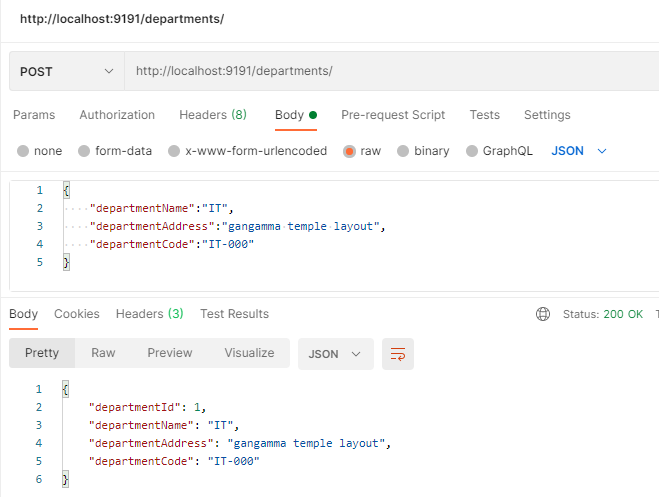
Now my API-GATEWAY is connected to eureka server.



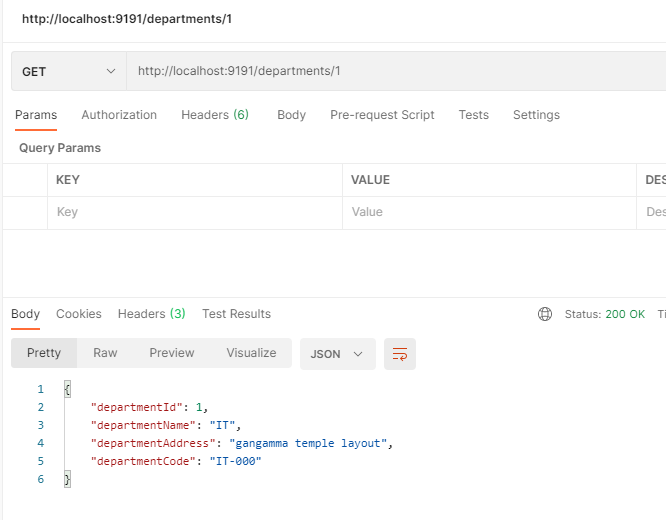
Now we can see the above fig. we have three services DEPARTMENT, USER and API-GATEWAY.

Now check our applications are working or not.

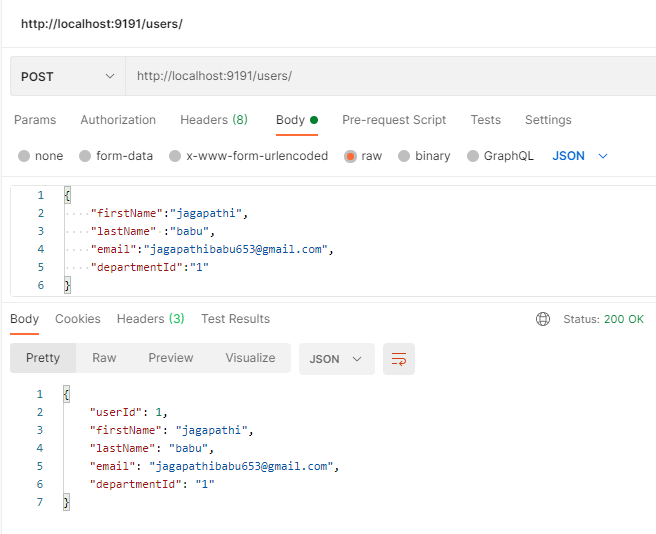
**Departmentservice:** post method



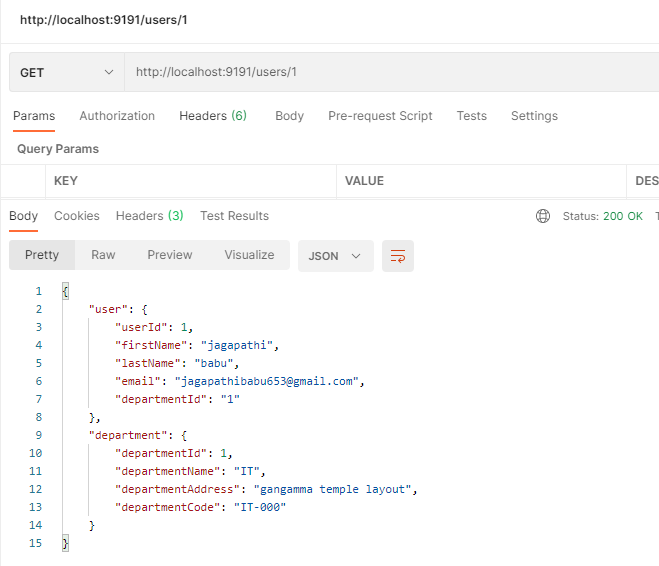
**Departmentservice:** Get method



**Userservice:** post method



**Userservice:** Get method



So my API-GATEWAY is working properly. All the requests are traverse to the API-GATEWAY with port: 9191.

* eureka.instance.prefer-ip-address=true

Add the above property in all application.properties file.

* Now we will be implementing the circuit breaker in our Microservices architecture.
* What circuit breaker will do is circuit breaker will identify which of the services is not running and it will run the fallback methods available and it will notify the users like these services are not working.
* So what will do as a circuit breaker we will be implementing the hystrix libraries and we will be implementing the hystrix dashboard. So that can we identify which all services are running and which all services are not running.
* Let’s go and implementing the hystrix library in our API cloud gateway and there we will implement the fallback methods for our Userservice and Departmentservice and we will do the configuration for that User and Departmentservice itself in the API-GATEWAY.

In our cloud gateway application we will add the hystrix dependency in pom.xml.

<dependency>

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

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

</dependency>

After that go to main application **CloudGatewayApplication.java** and enable the @EnableHystrix.

We will create FallBackMethodController whenever service is down we will redirect the request that particular controller

**FallBackMethodController.java:**

package com.ms.cloudgateway;

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

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

@RestController

public class FallBackMethodController {

@GetMapping("/userServiceFallBack")

public String userServiceFallBackMethod() {

return "User Service is taking longer than expected."+ "please try again later";

}

@GetMapping("/departmentServiceFallBack")

public String departmentServiceFallBackMethod() {

return "Department Service is taking longer than expected."+ "please try again later";

}

}

After that we will go to application.yaml and after predicates we will add these configuration.

filters:

- name: CircuitBreaker

args:

name: USER

fallbackuri: forward:/userServiceFallBack

filters:

- name: CircuitBreaker

args:

name: DEPARTMENT

fallbackuri: forward:/departmentServiceFallBack

Add the hystrix configuration:

hystrix:

command:

fallbackcmd:

execution:

isolation:

thread:

timeoutInMilliseconds: 4000

We enable the hystrix. stream we can use this information in the hystrix dashboard:

management:

endpoints:

web:

exposure:

include: hystrix.stream

Creating hystrix-dashboard :

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.3**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

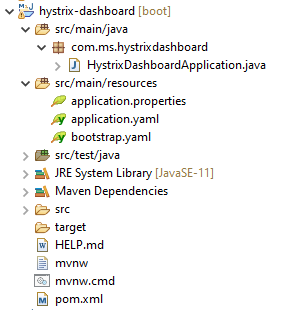
**Step 4:** Provide the **Artifact** Id. We have provided **hystrix-dashboard**

**Step 5:** Add the dependencies **eureka discovery client, hystrix dashboard**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application.

The project structure is like:



**HystrixDashboardApplication.java:**

package com.ms.hystrixdashboard;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;

@SpringBootApplication

@EnableHystrixDashboard

@EnableEurekaClient

public class HystrixDashboardApplication {

public static void main(String[] args) {

SpringApplication.run(HystrixDashboardApplication.class, args);

}

}

**Application.yaml:**

server:

port: 9295

spring:

application:

name: HYSTRIX-DASHBOARD

hystrix:

dashboard:

proxy-stream-allow-list: "\*"

**pom.xml:**

<?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>2.3.11.RELEASE</version>

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

</parent>

<groupId>com.ms</groupId>

<artifactId>hystrix-dashboard</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>hystrix-dashboard</name>

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

<properties>

<java.version>11</java.version>

<spring-cloud.version>Hoxton.SR11</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-config -->

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</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>

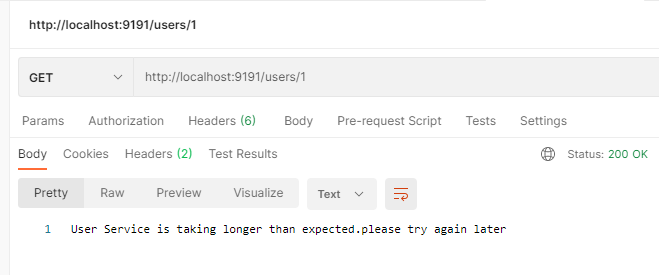
</plugin>

</plugins>

</build>

</project>

Run the all applications. after that we are checking the circuit breaker fall back method is working or not. I’m stopping the Departmentservice. I request the user and it goes department and now my department is down so it will show like some error message.



Creating cloud config-server :

**Step 1:** Open Spring Initializr.

**Step 2:** Select the Spring Boot version **2.3**

**Step 3:** Provide the **Group** name. We have provided **com.ms**

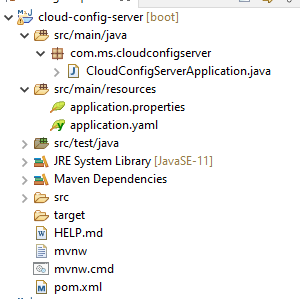
**Step 4:** Provide the **Artifact** Id. We have provided **cloud- config-server**

**Step 5:** Add the dependencies **eureka discovery client, config server**

**Step 6:** Click on the **Generate** button. When we click on the Generate button, it wraps the specifications in a **Jar** file and downloads it to the local system.

Once its downloded, import the project in spring boot application.

The project structure is like:



**CloudConfigServerApplication.java:**

package com.ms.cloudconfigserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.config.server.EnableConfigServer;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

@EnableEurekaClient

@EnableConfigServer

public class CloudConfigServerApplication {

public static void main(String[] args) {

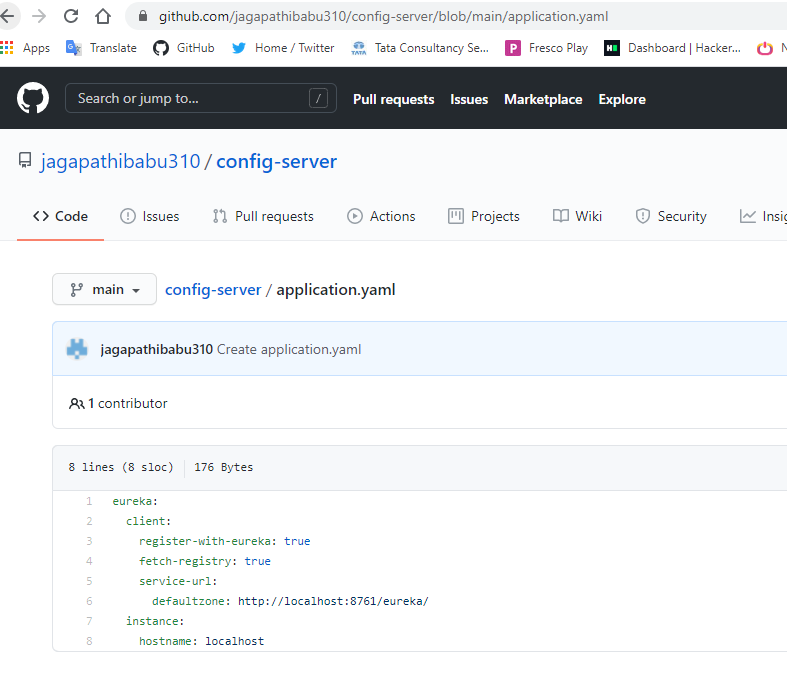
SpringApplication.run(CloudConfigServerApplication.class, args);

}

}

We will create one git repository and there we will create application.yaml file, here we store all default configuration there.

* Let’s go to the github.
* Create new repository. Repository name is config-server, and then click on create repository.
* Create a new file name as application.yaml. in this file I’ll store all default configurations.



The above configuration is the default configuration for all my services and I’ll give git repository link in application.yaml.

**Application.yaml:**

server:

port: 9296

spring:

application:

name: CONFIG-SERVER

cloud:

config:

server:

git:

uri: https://github.com/jagapathibabu310/config-server

clone-on-start: true

**pom.xml:**

<?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>2.3.11.RELEASE</version>

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

</parent>

<groupId>com.ms</groupId>

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

<version>0.0.1-SNAPSHOT</version>

<name>cloud-config-server</name>

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

<properties>

<java.version>11</java.version>

<spring-cloud.version>Hoxton.SR11</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</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>

</plugin>

</plugins>

</build>

</project>

* Application.yaml file is used for the application contexts.
* Whenever we are staring the application contexts spring boot will be using the application.yaml file.
* For getting the all cloud configurations we use bootstrap.yaml file.
* So that is been used bootstrap the configurations and it will start the applications context from the application.yaml file.

**bootstrap.yaml:**

spring:

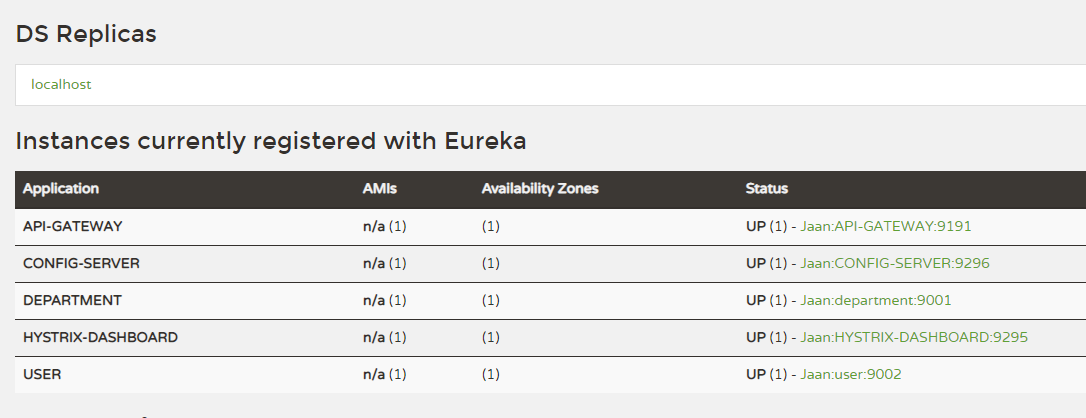
cloud:

config:

enabled: true

uri: <http://localhost:9296>

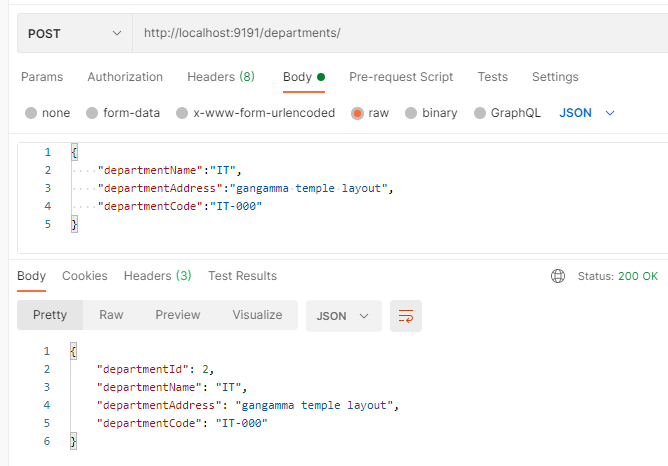
* Add the bootstrap.yaml file in our department, user, cloud-gateway, cloud-config-server, hystrix dashboard.
* Now let’s go and start the all services.
* Once all the services started now we go to eureka server and it’s look like this.



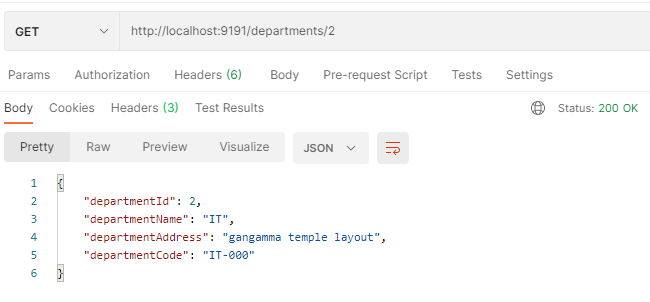
So all services are connected to my eureka server.

* All my configurations has been served from cloud-config-server. So let test our services.

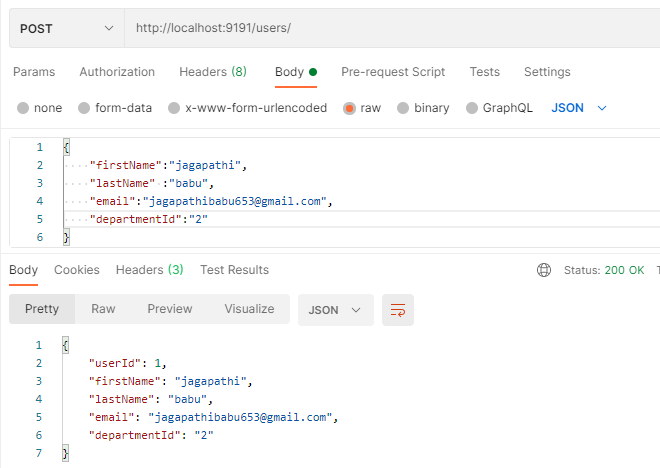
**Departmentservice:** post method



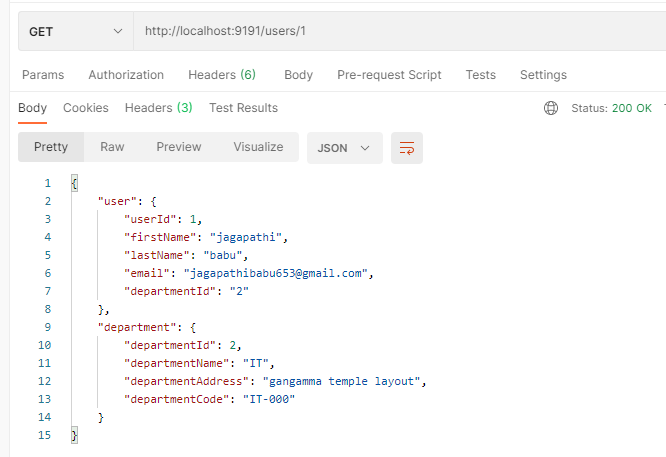
**Departmentservice:** Get method



**Userservice:** post method



**Userservice:** Get method



* Now we can see that everything is fine. So we add the configuration server which will serve all the default configurations.

So these are all the basic and necessary information about to start working on the Microservices architecture.