**Build Micro Services Using Spring Tool Suite, Spring Boot & Netflix-OSS**

Table Contents

[Introduction 4](#_Toc49964143)

[Revision History 4](#_Toc49964144)

[Prerequisites: 4](#_Toc49964145)

[Create Eureka Server Project 4](#_Toc49964146)

[1. Create a Spring Starter Project 4](#_Toc49964147)

[2. Default Package and Class 5](#_Toc49964148)

[2. Dependency (Pom.xml) files 6](#_Toc49964149)

[3. Configuring properties with application.yml 6](#_Toc49964150)

[Register Eureka Server 7](#_Toc49964151)

[1. Run as Spring Boot Application 7](#_Toc49964152)

[2. Access Eureka Server dashboard 8](#_Toc49964153)

[Create Micro service (drama-service) 8](#_Toc49964154)

[1. Create a Spring Starter Project 9](#_Toc49964155)

[2. Default Package & Class 9](#_Toc49964156)

[3. Dependency (Pom.xml) files 10](#_Toc49964157)

[4. Configuring properties with application.yml 11](#_Toc49964158)

[5. Implement Rest Resources 12](#_Toc49964159)

[Create Micro service (actor-service) 15](#_Toc49964160)

[1. Create a Spring Starter Project 15](#_Toc49964161)

[2. Default Package & Class 15](#_Toc49964162)

[3. Dependency (Pom.xml) files 16](#_Toc49964163)

[4. Configuring properties with application.yml 16](#_Toc49964164)

[5. Implement Rest Resources 17](#_Toc49964165)

[Create API-Gateway(Zuul-Proxy) 19](#_Toc49964166)

[1. Create a Spring Starter Project 19](#_Toc49964167)

[2. Default Package & Class 20](#_Toc49964168)

[3. Dependency (Pom.xml) files 21](#_Toc49964169)

[4. Configuring properties with application.yml 21](#_Toc49964170)

[Register Eureka Client with Eureka server Registry 22](#_Toc49964171)

[1. Run as Spring Boot Application 22](#_Toc49964172)

[2. Access Eureka Server dashboard 23](#_Toc49964173)

[3. Access H2-database 23](#_Toc49964174)

[Test the Microservices 24](#_Toc49964175)

[1. Create collection in Postman 24](#_Toc49964176)

[2. Create actor-service request 25](#_Toc49964177)

[3. Microservice Inter-communication 26](#_Toc49964178)

[4. Hystrix-breaker response 26](#_Toc49964179)

[5. Hystrix-dashboard 26](#_Toc49964180)

Introduction:

The tutorial provides a step-by-step guide for building and configuring a Microservice in Java using Spring boot & Netflix-OSS components. You will learn how easy it is to create your Spring boot application and test with Postman. Maven is used to build the tool. Postman is used to test the rest webservices.

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Changes Name** | **Updated By** | **Date** | **Version** |
| New Document | Sudhanshu Satyam | 2020/08/10 | 1.0.0 |
| Revision |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Prerequisites:

There is some software which should be installed on local machine before creating an application.

Spring Tools Suite (STS)

Java 1.8 or above

Apache Maven 3.5 or above

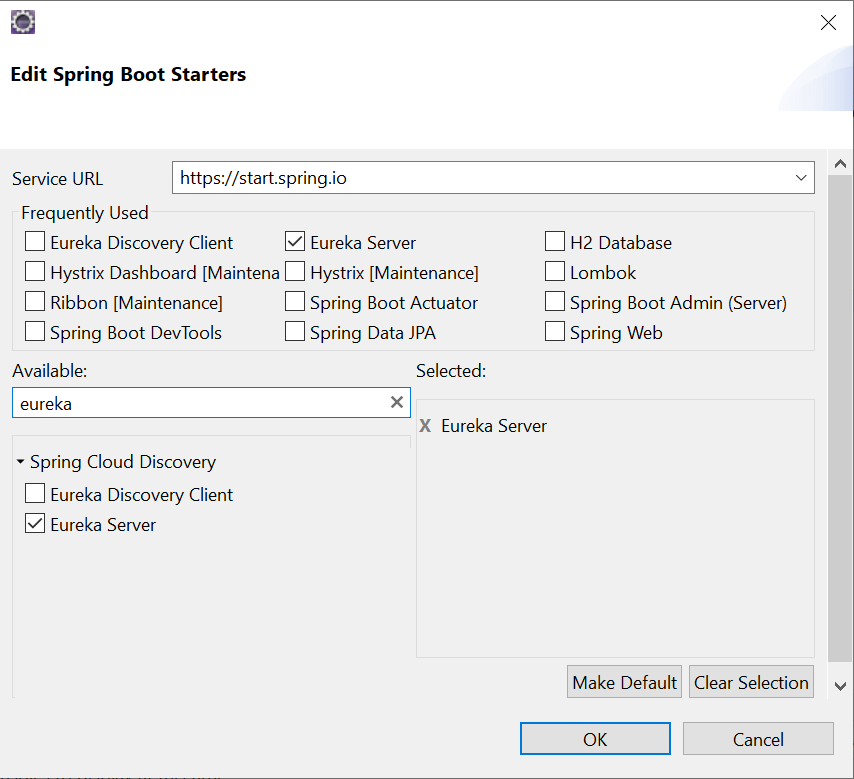
Postman

# Create Eureka Server Project

## Create a Spring Starter Project

Create a Spring Starter project and name your project discovery-server.

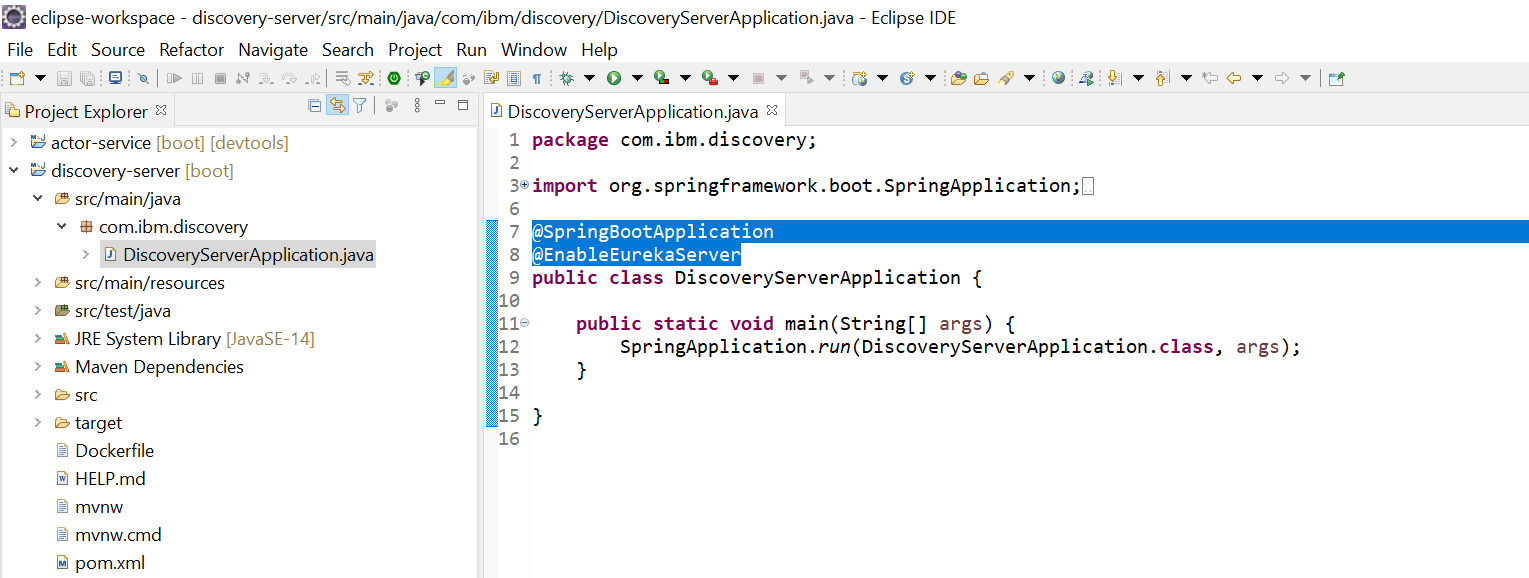
Add maven & maven-dependencies to create your Eureka Server as shown in Screen shot below:



1. Default Package and Class**:**

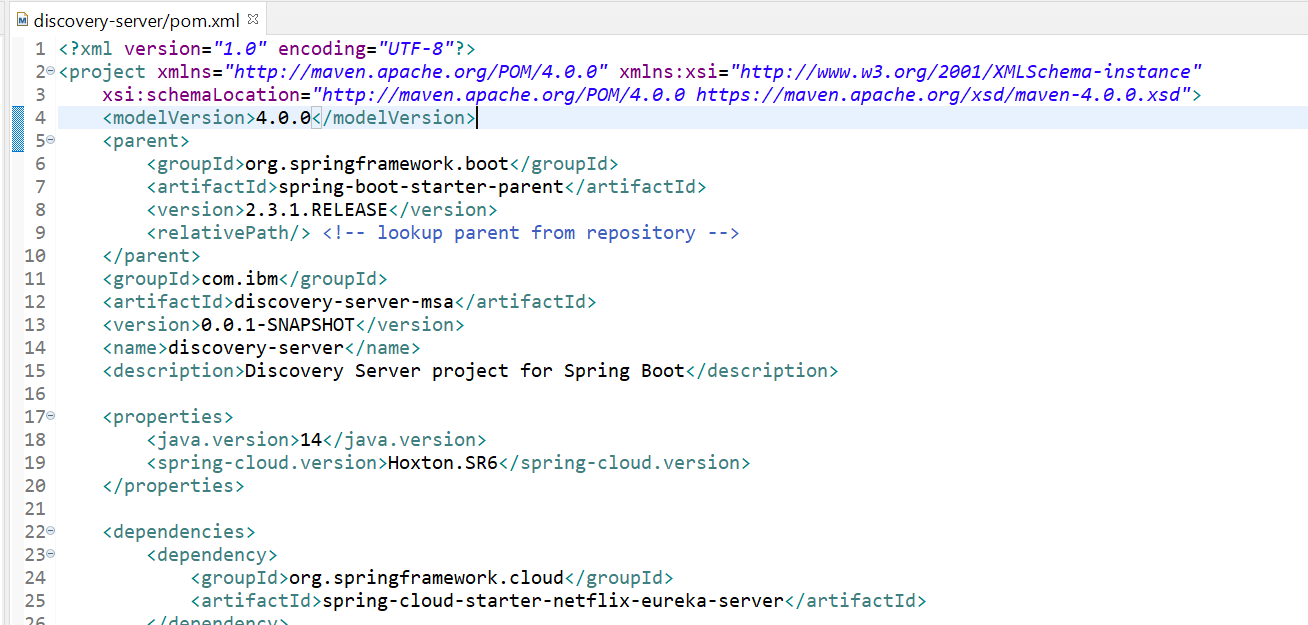
By Default, the project will create a package and one spring boot class which is entry point of spring boot project. Annotate Spring boot main class with @SpringBootApplication & @EnableEurekaServer.

Annotating class with @EnableEurekaServer treats this class as Eureka Server.



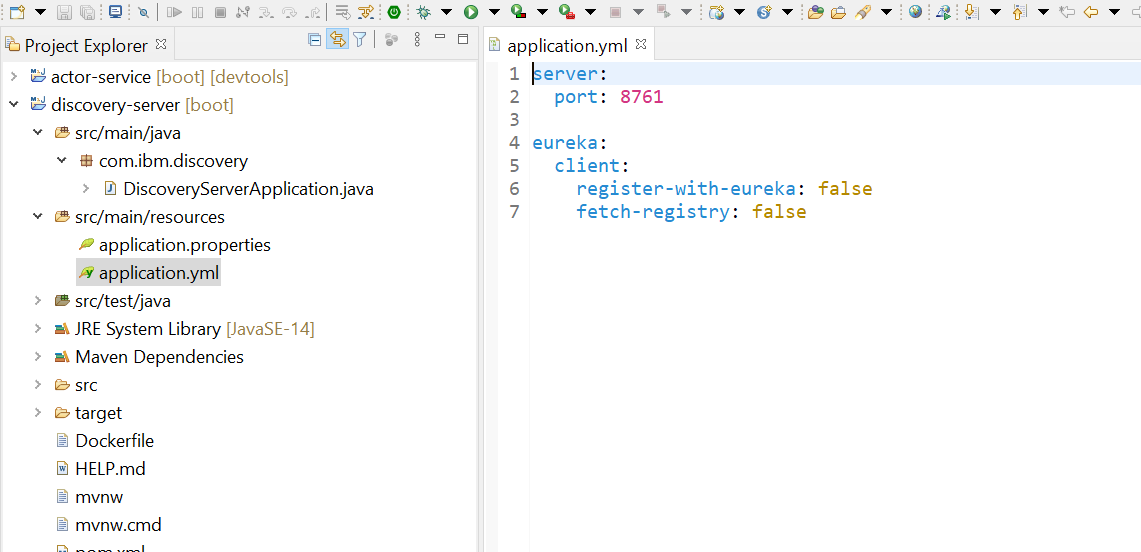
## Dependency (Pom.xml) files

By default, Spring Boot Create a pom.xml which contains all the dependency of discovery-server Project.



## Configuring properties with application.yml

Now , We have to configure Eureka Server properties by creating application.yml file in resource directory(/src/main/resources). This file contains server port and registry-related configuration as show below:



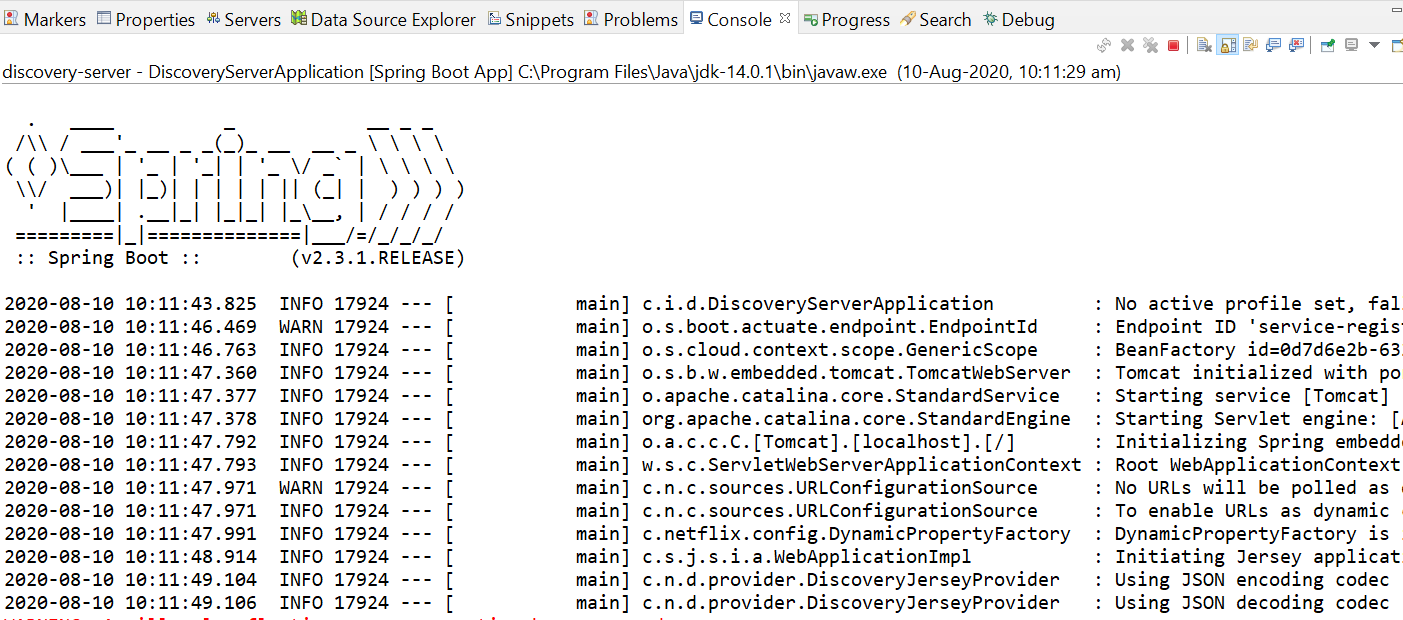
# Register Eureka Server

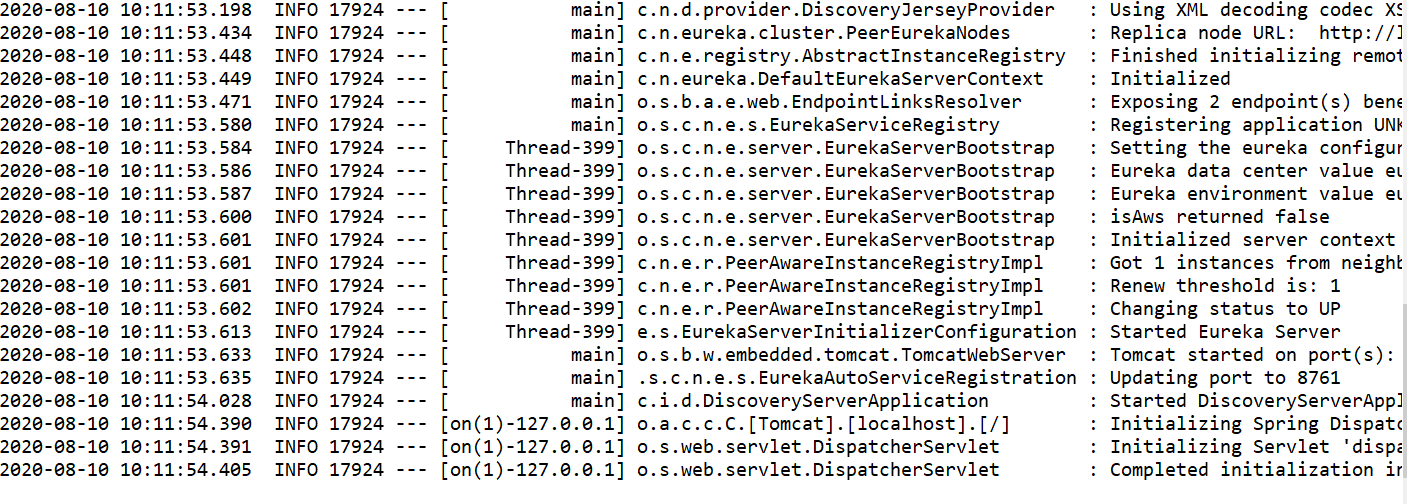
We have registered Eureka Server at port : 8761

* 1. Run as Spring boot application

## Run as Spring Boot Application

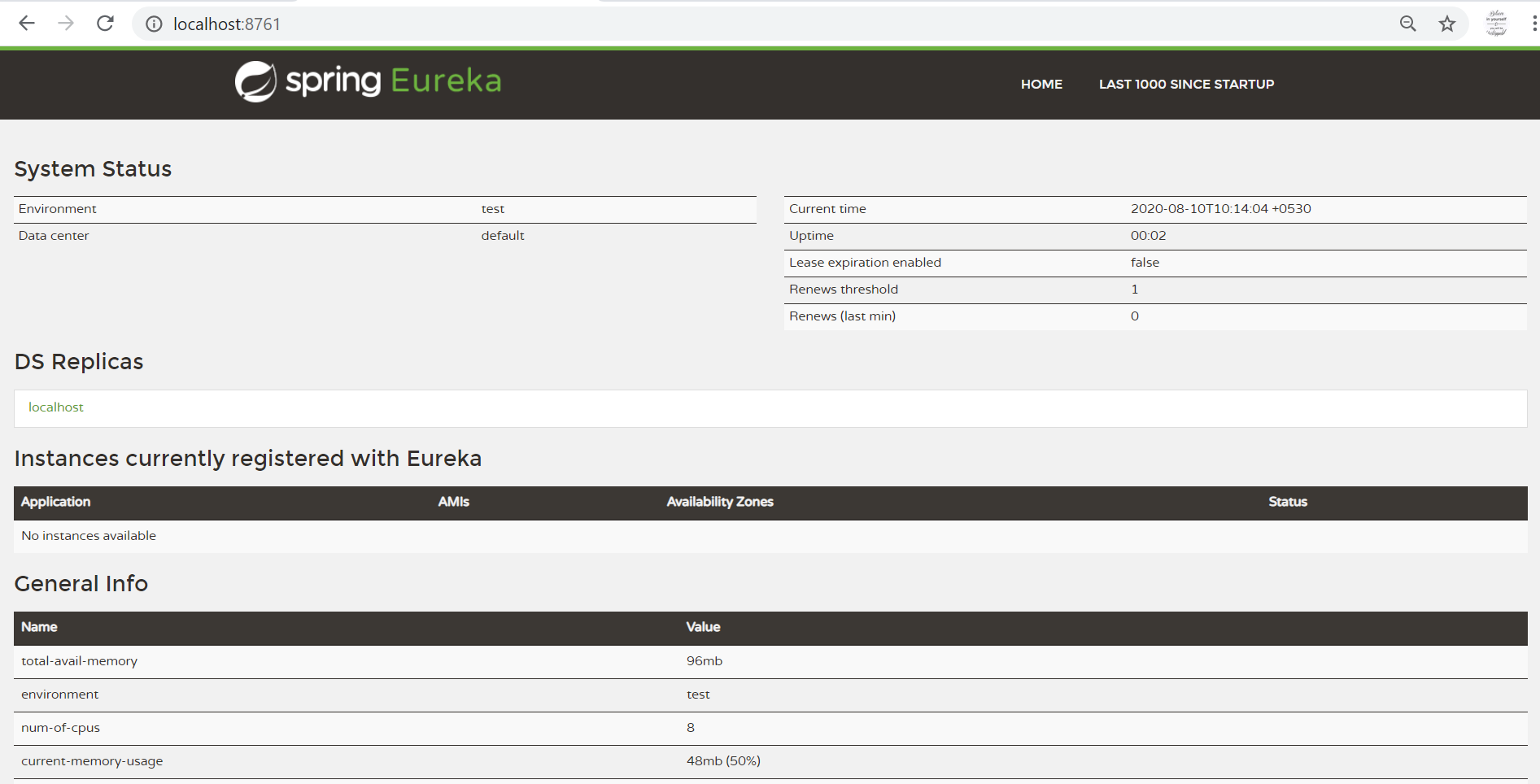
Right click on the project 🡪 Run As 🡪 Spring Boot App





## Access Eureka Server dashboard

Tomcat started with given port: 8761 and application started on JVM. Now the application can be accessed through browser to view Eureka Server dashboard at <http://localhost:8761/>



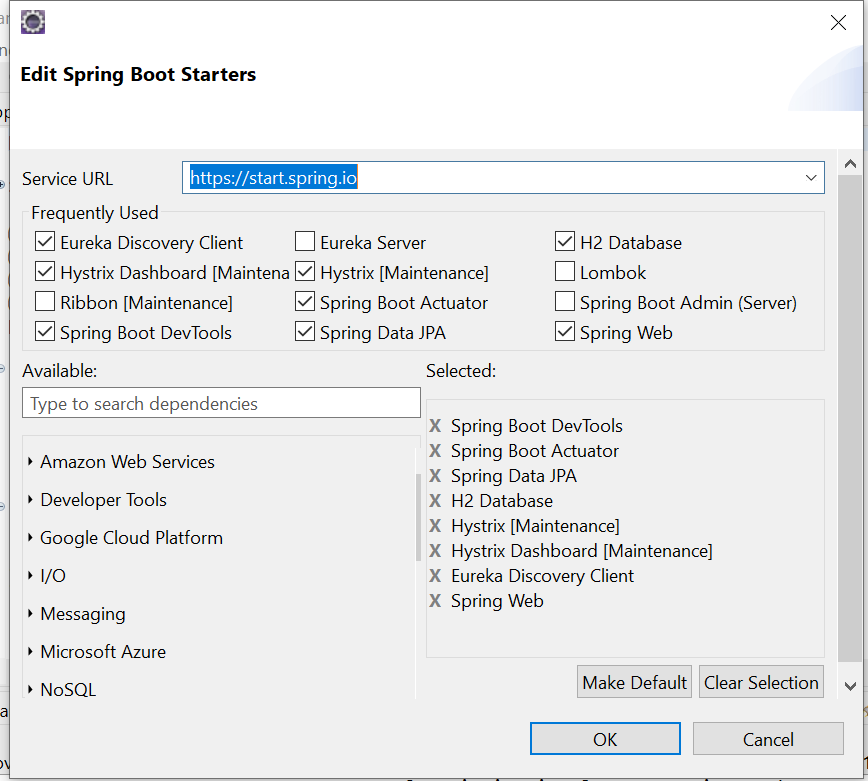


# Create Micro service (drama-service)

## Create a Spring Starter Project

Create a Spring Starter project and name your project drama-service.

Add maven & maven-dependencies to create your Micro service as shown in Screen shot below:



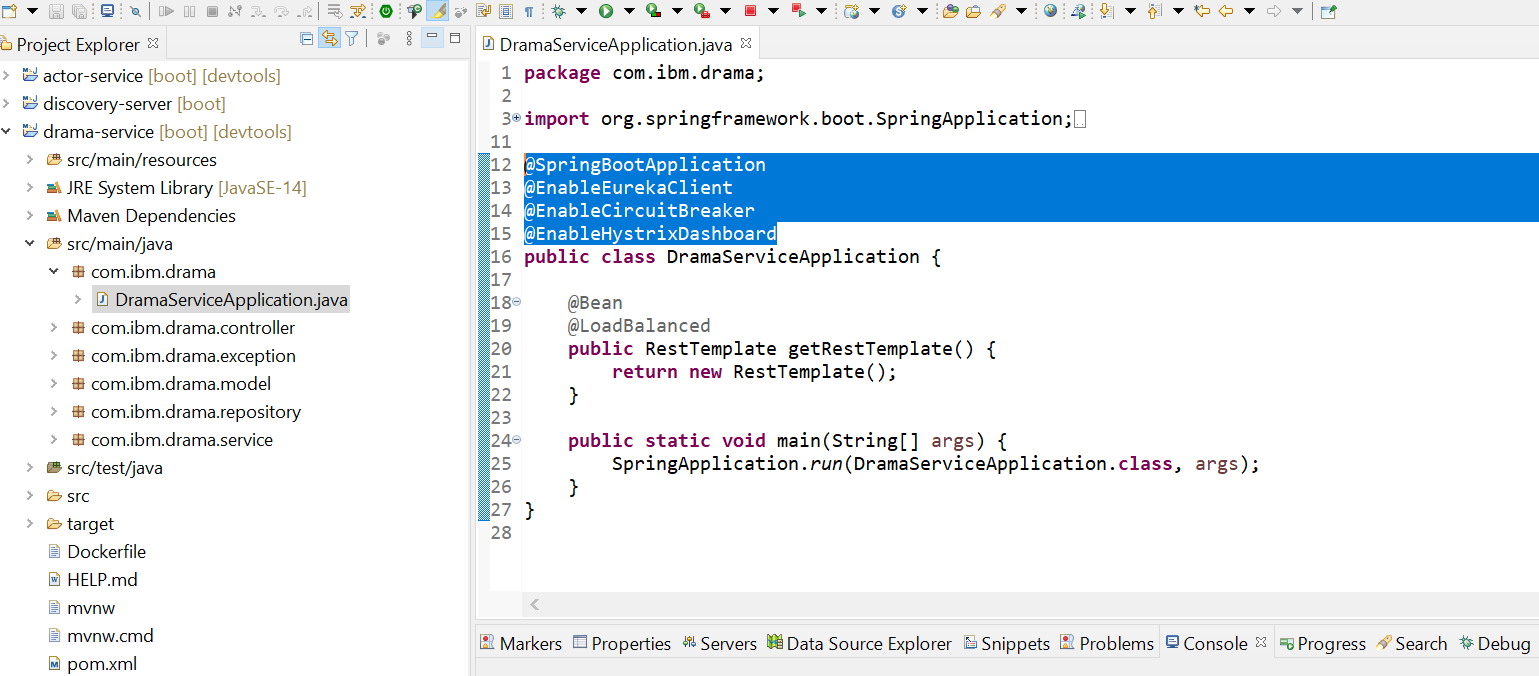
## Default Package & Class

By Default, the project will create a package and one spring boot class which is entry point of spring boot project. Annotate Spring boot main class to include following features :

* @EnableEurekaClient – This annotation registers Java class as discovery

client with Eureka Server.

* @EnableCircuitBreaker/@EnableHystrix - This annotation handles Fault Tolerance/Circuit-Breaker mechanism in a real-world Microservice application
* @EnableHystrixDashboard – This annotation visualize the application metrics on Hystrix dashboard for request/response analyzation.



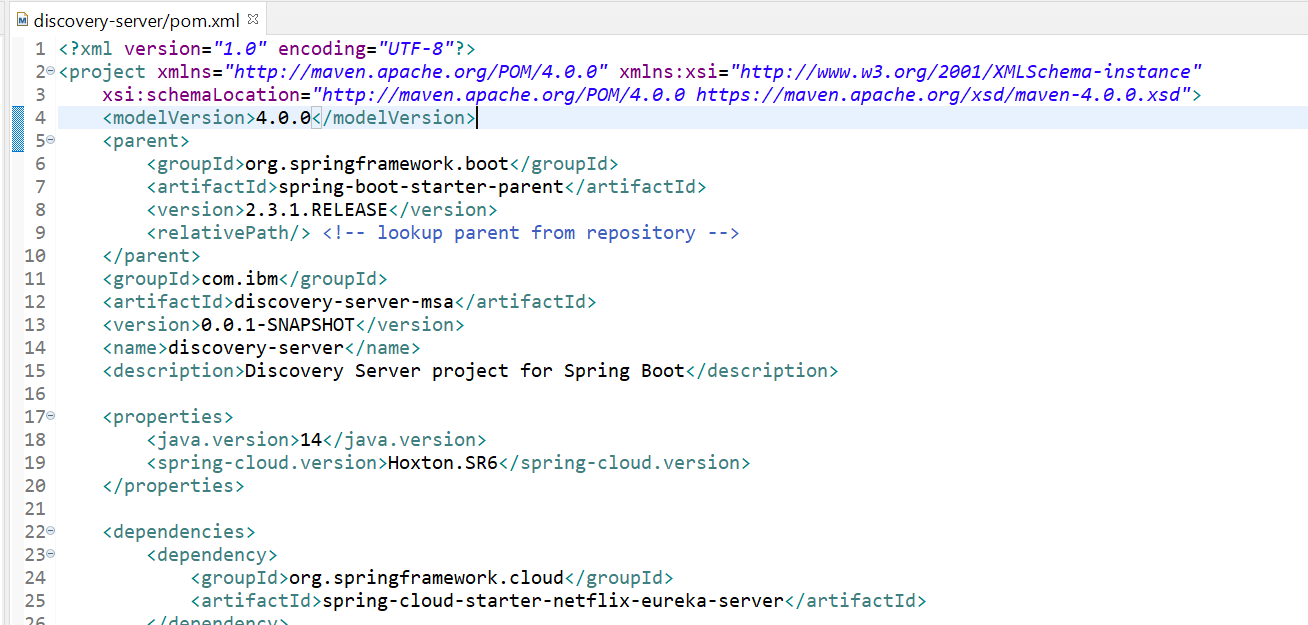
## Dependency (Pom.xml) files

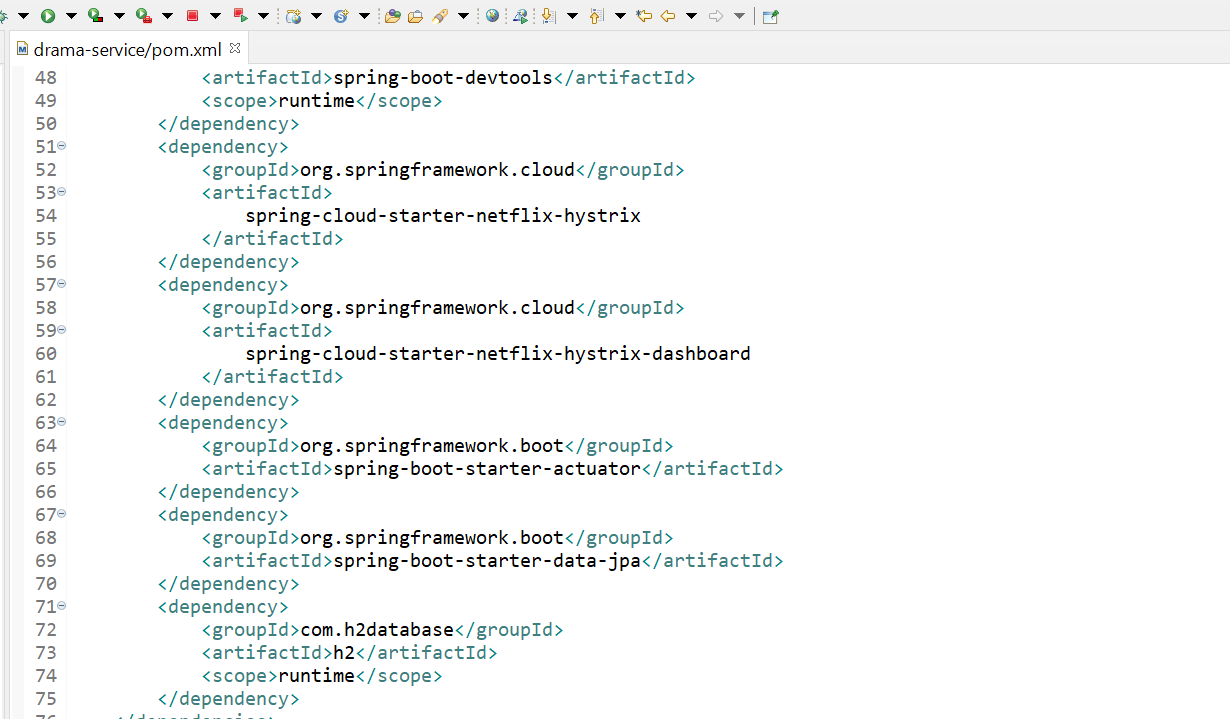
By default, Spring Boot Create a pom.xml which contains all the dependency of Discovery-Client, Circuit-Breaker, Hystrix & Hystrix-Dashboard Project.

Spring boot Actuator & Spring boot Devtools are for monitoring the application.

Spring-Data JPA is added for database interaction.

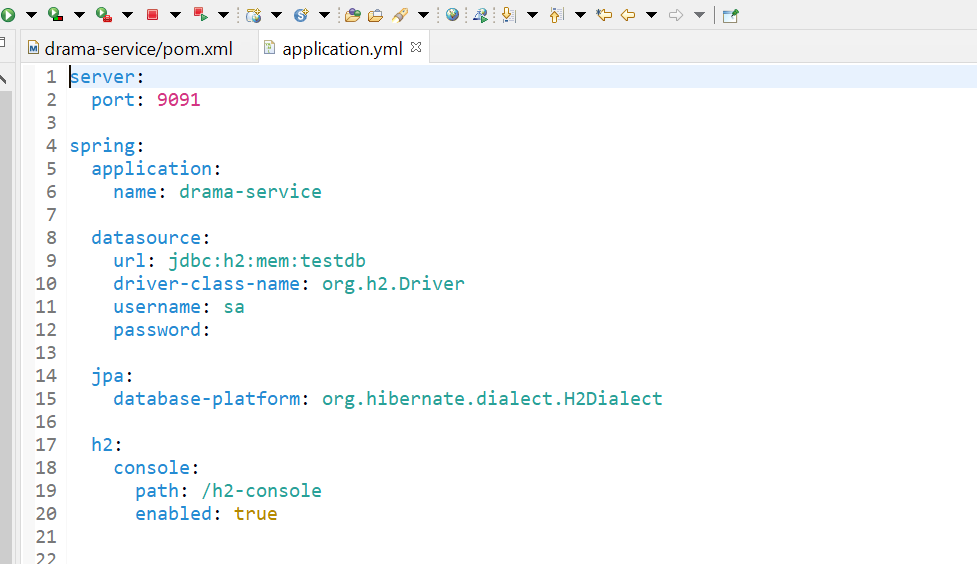
H2-runtime is in-memory database.





## Configuring properties with application.yml

Now , We have to configure Discovery Client properties to register with Eureka Server by creating application.yml file in resource directory(/src/main/resources). This file contains server port and registry-related configuration as show below:

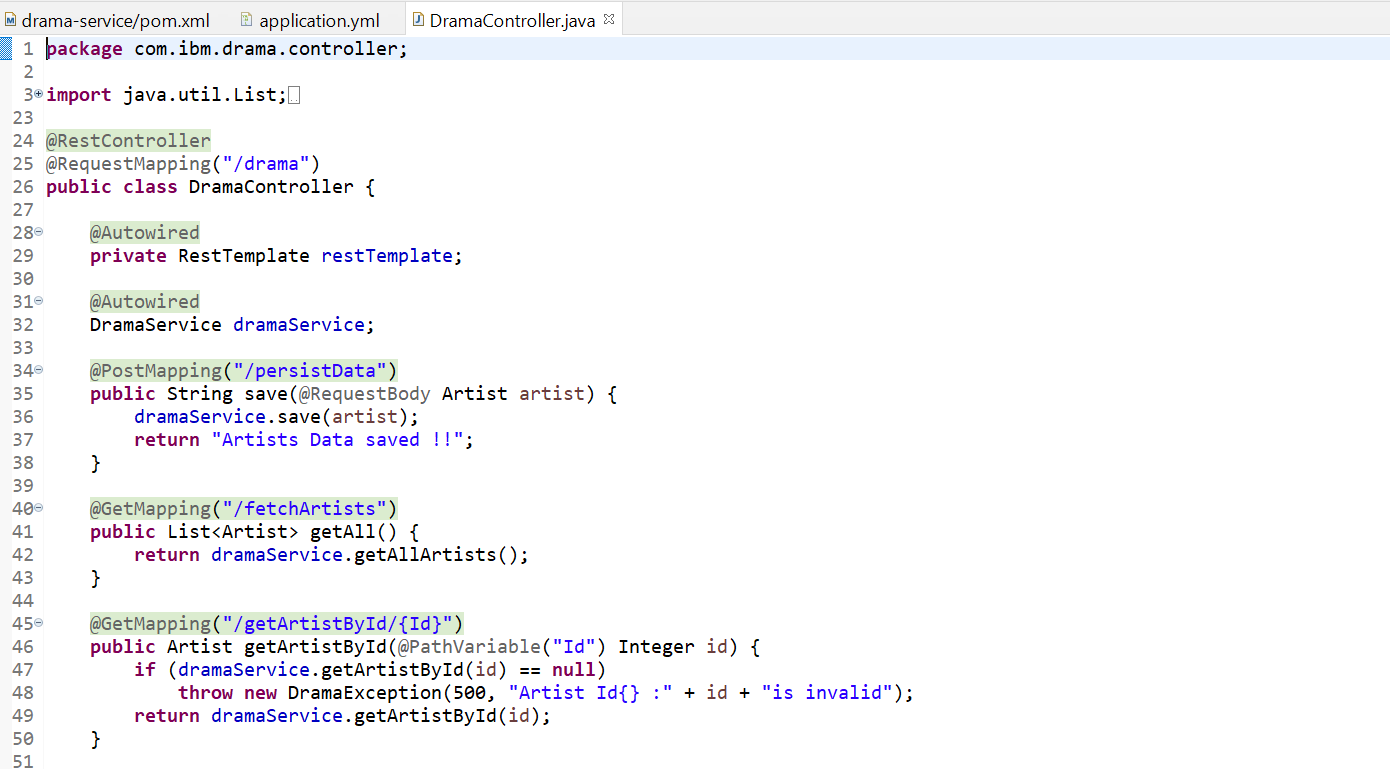


## Implement Rest Resources

This service implements REST resource using Spring boot. It accept JSON requests and respond back with JSON responses.



**@RestController:** this annotation marks the class as a Resource, it defines implicitly both @Controller and @ResponseBody MVC annotations, when annotating a class with @RestController, it's not necessary to write @ResponseBody beside the POJO classes returned from your methods.



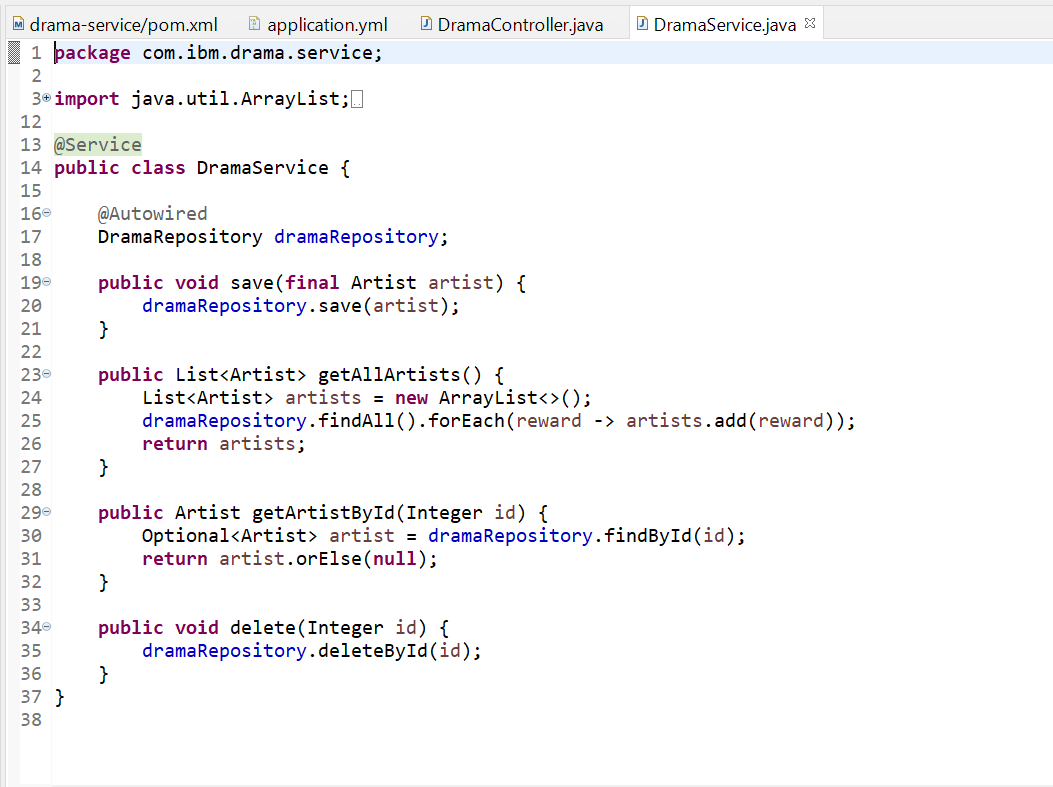
This method is designed to communicate with external Microservices. Circuit-Breaker is implemented to handle Fault occurrences, if external services goes down.

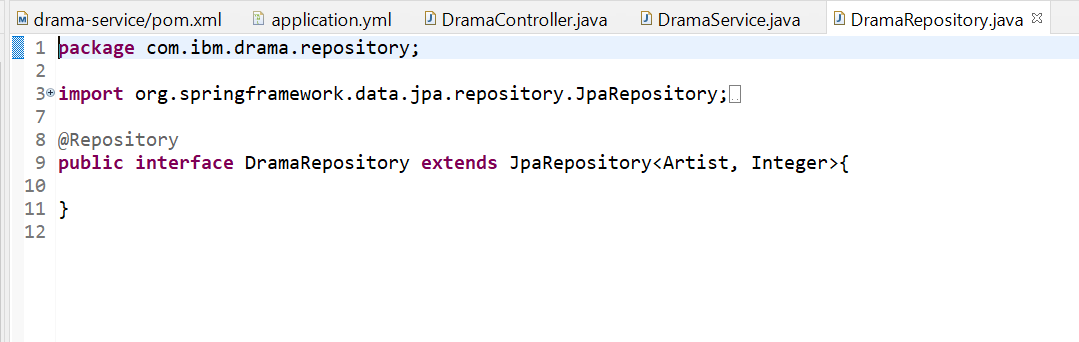
HystrixCommand(fallbackMethod = "fallbackMethodForGetReward", commandKey = "test-Rwd", groupKey = "test-Rwd")



This Java class serves as service layer that interacts with Repository/Dao to query databases.

Annotated with @Service and @Repository simultaneously. This class is mainly designed for performing CRUD operations on database.



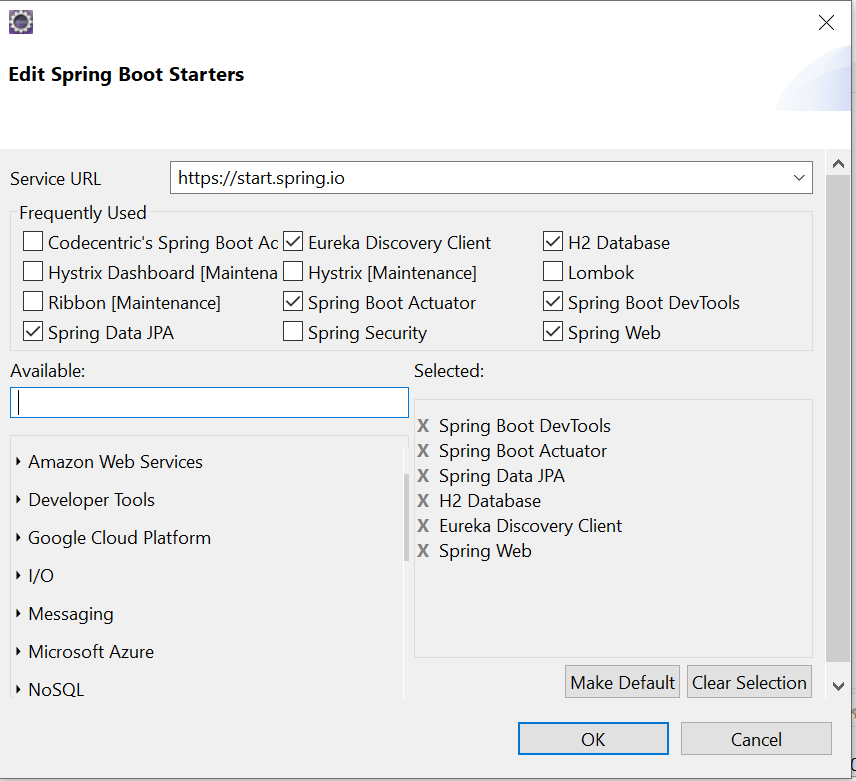


# Create Micro service (actor-service)

## Create a Spring Starter Project

Create a Spring Starter project and name your project actor-service.

Add maven & maven-dependencies to create your Micro service as shown in Screen shot below:

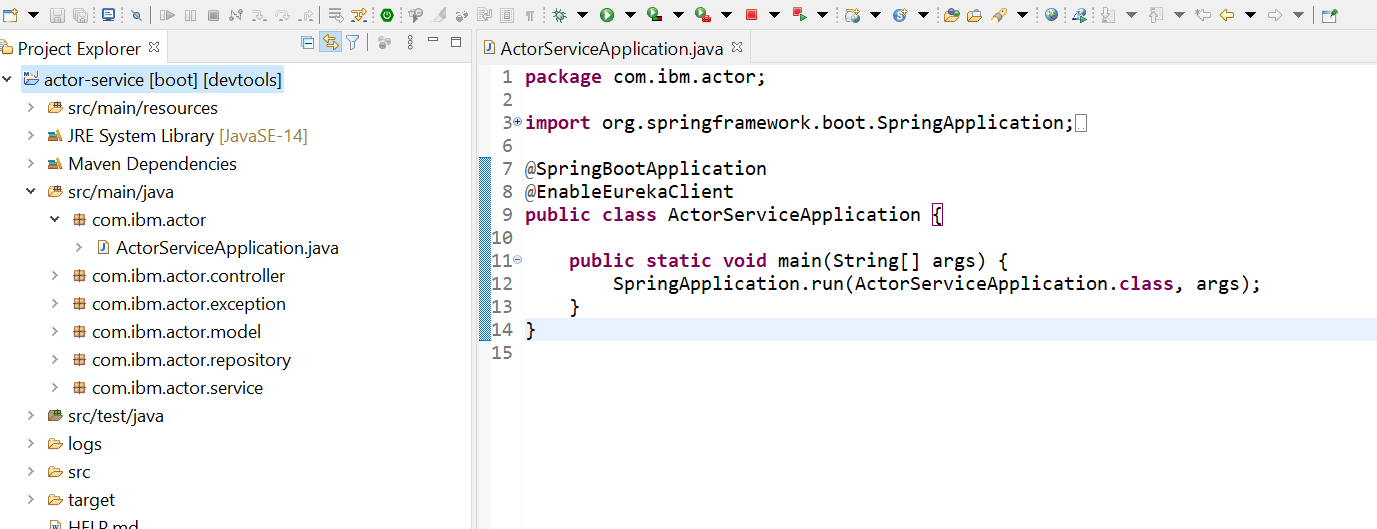


## Default Package & Class

By Default, the project will create a package and one spring boot class which is entry point of spring boot project. Annotate Spring boot main class to include following features :

* @EnableEurekaClient – This annotation registers Java class as discovery

client with Eureka Server.



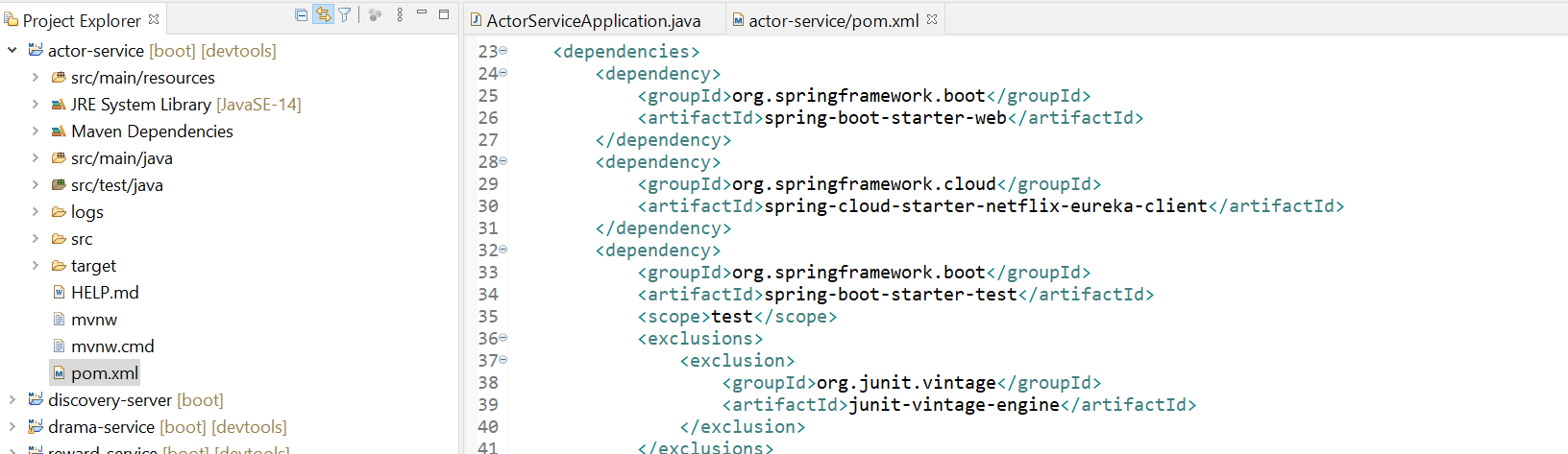
## Dependency (Pom.xml) files

By default, Spring Boot Create a pom.xml which contains all the dependency of Discovery-Client & Spring-web Project.

Spring boot Actuator & Spring boot Devtools are for monitoring the application.

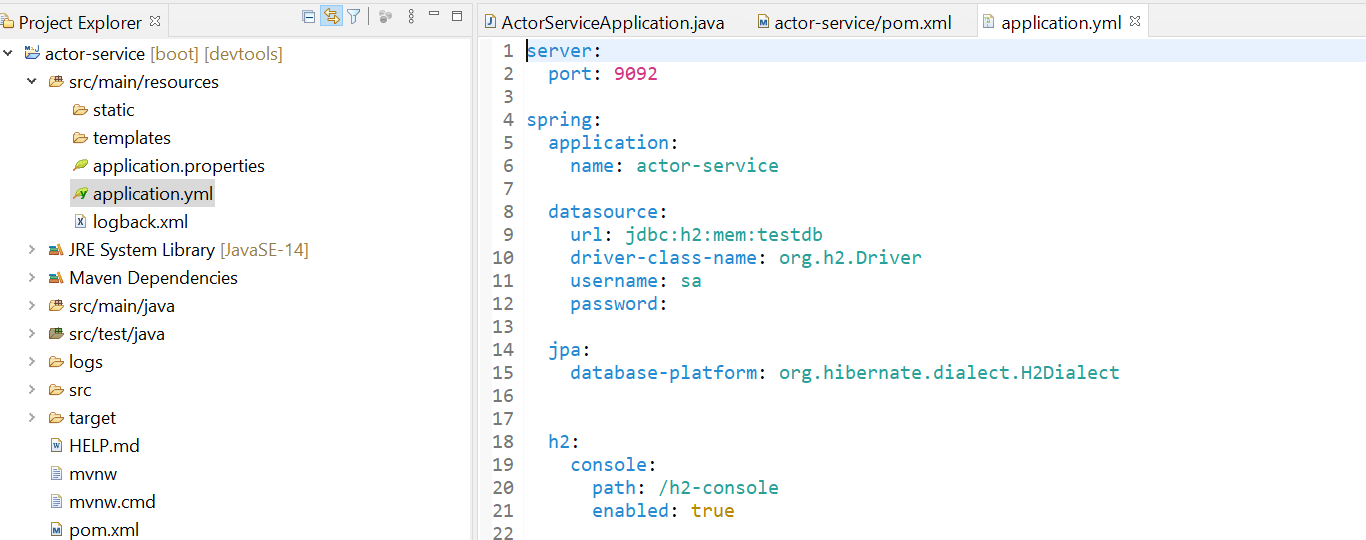
Spring-Data JPA is added for database interaction.

H2-runtime is in-memory database.



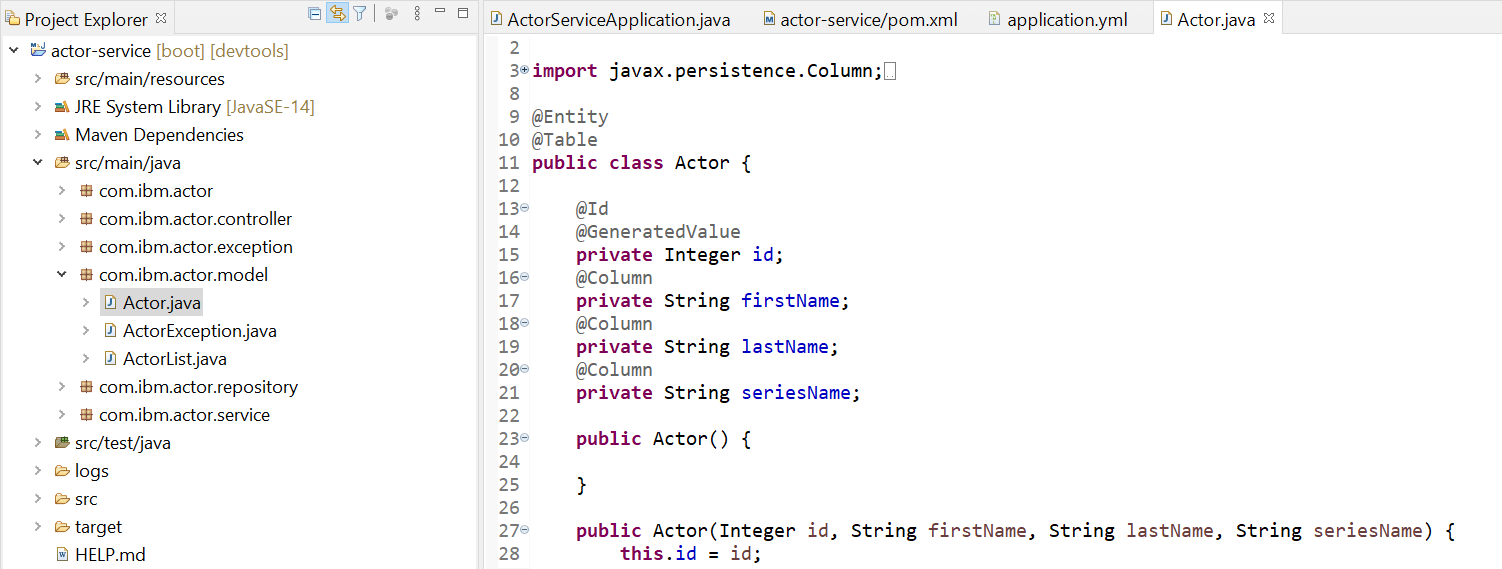
## Configuring properties with application.yml

Now , We have to configure Discovery Client properties to register with Eureka Server by creating application.yml file in resource directory(/src/main/resources). This file contains server port and registry-related configuration as show below:



## Implement Rest Resources

This service implements REST resource using Spring boot. It accept JSON requests and respond back with JSON responses.

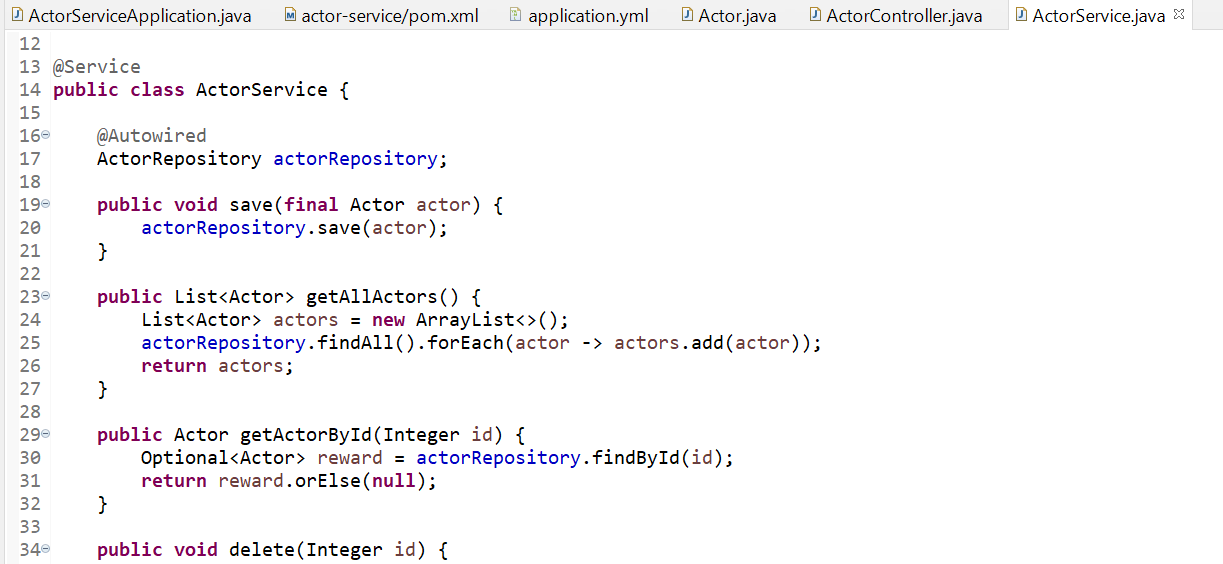


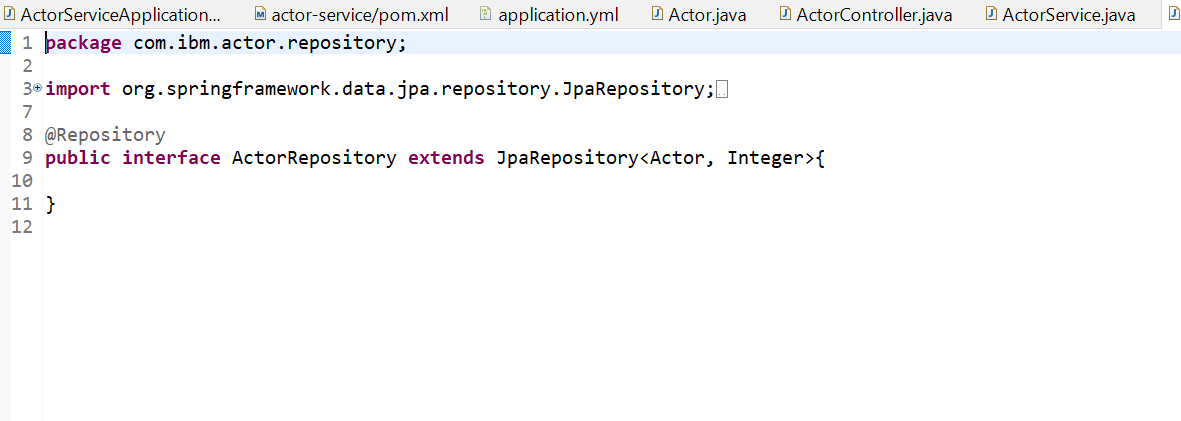
**@RestController:** this annotation marks the class as a Resource, it defines implicitly both @Controller and @ResponseBody MVC annotations, when annotating a class with @RestController, it's not necessary to write @ResponseBody beside the POJO classes returned from your methods.



This Java class serves as service layer that interacts with Repository/Dao to query databases.

Annotated with @Service and @Repository simultaneously. This class is mainly designed for performing CRUD operations on database.



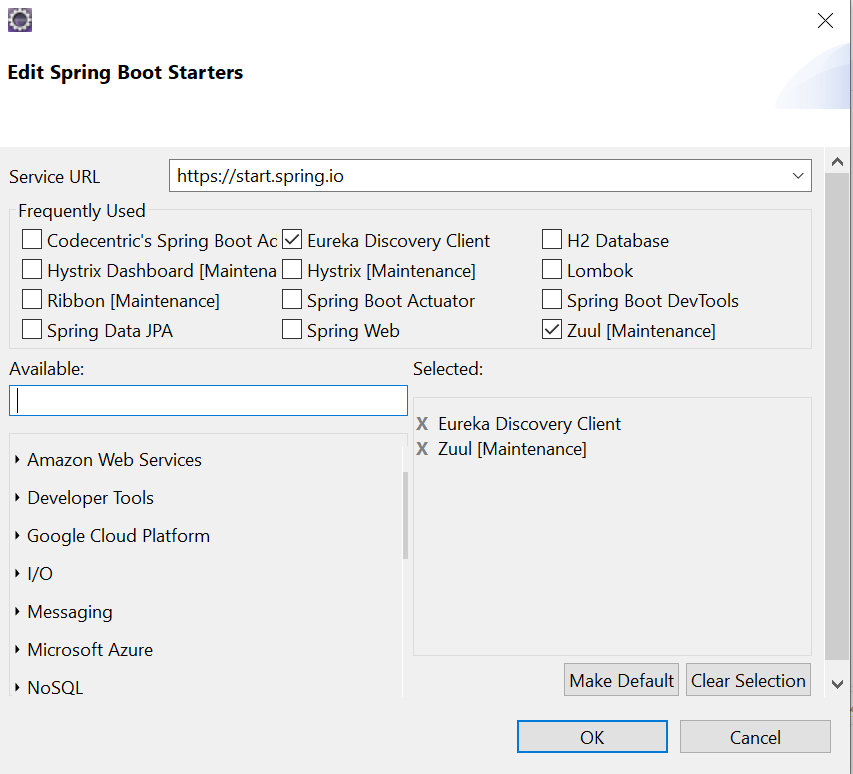


# Create API-Gateway(Zuul-Proxy)

## Create a Spring Starter Project

Create a Spring Starter project and name your project zuul-service.

Add maven & maven-dependencies to create your Micro service as shown in Screen shot below:



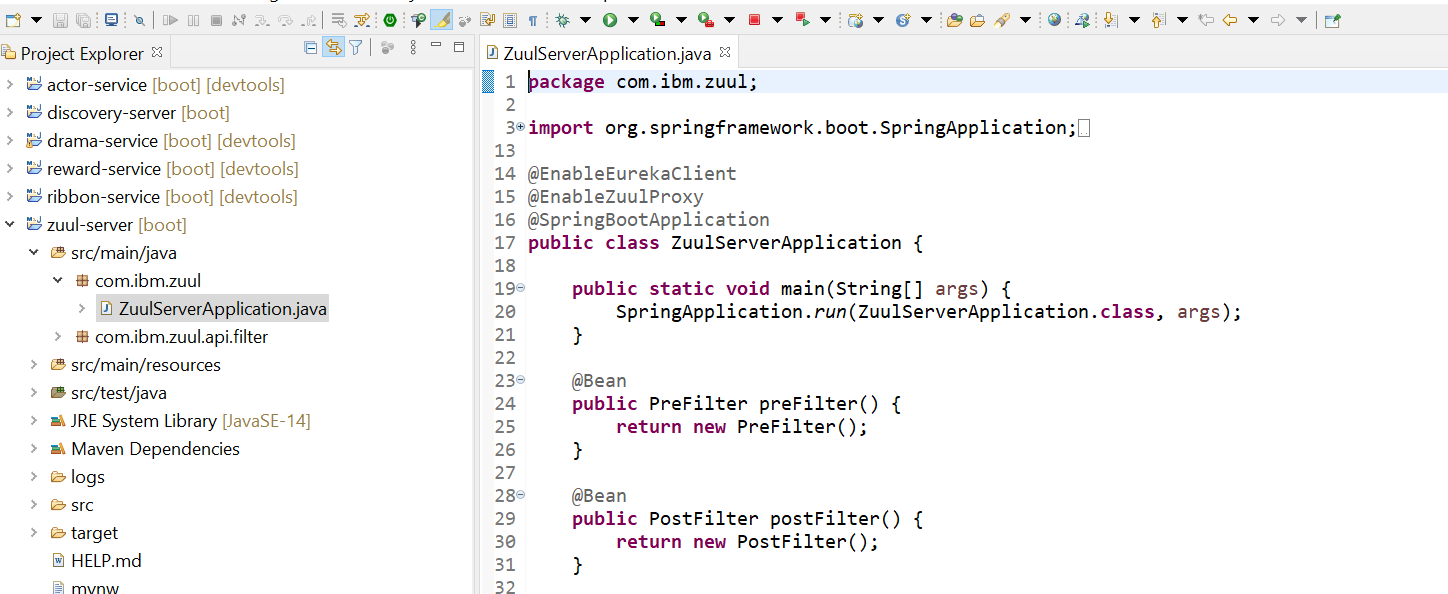
## Default Package & Class

By Default, the project will create a package and one spring boot class which is entry point of spring boot project. Annotate Spring boot main class to include following features :

* @EnableEurekaClient – This annotation registers Java class as discovery

client with Eureka Server.

* @EnableZuulProxy – This annotation registers Java class as api-gateway and serves as proxy server for application.
* Various Filters has been registered within this class to perform pre/post processing of request.



## Dependency (Pom.xml) files

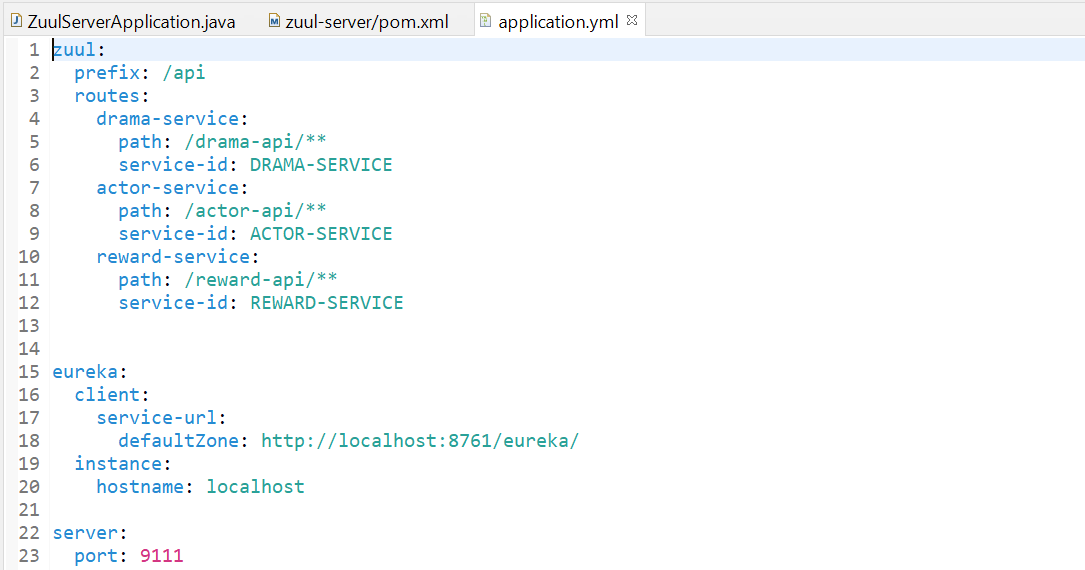
By default, Spring Boot Create a pom.xml which contains all the dependency of Discovery-Client & Netflix-zuul Project.

Spring boot Actuator & Spring boot Devtools are for monitoring the application.



## Configuring properties with application.yml

Now , We have to configure Discovery Client properties to register with Eureka Server by creating application.yml file in resource directory(/src/main/resources). This file contains server port and registry-related configuration as show below:



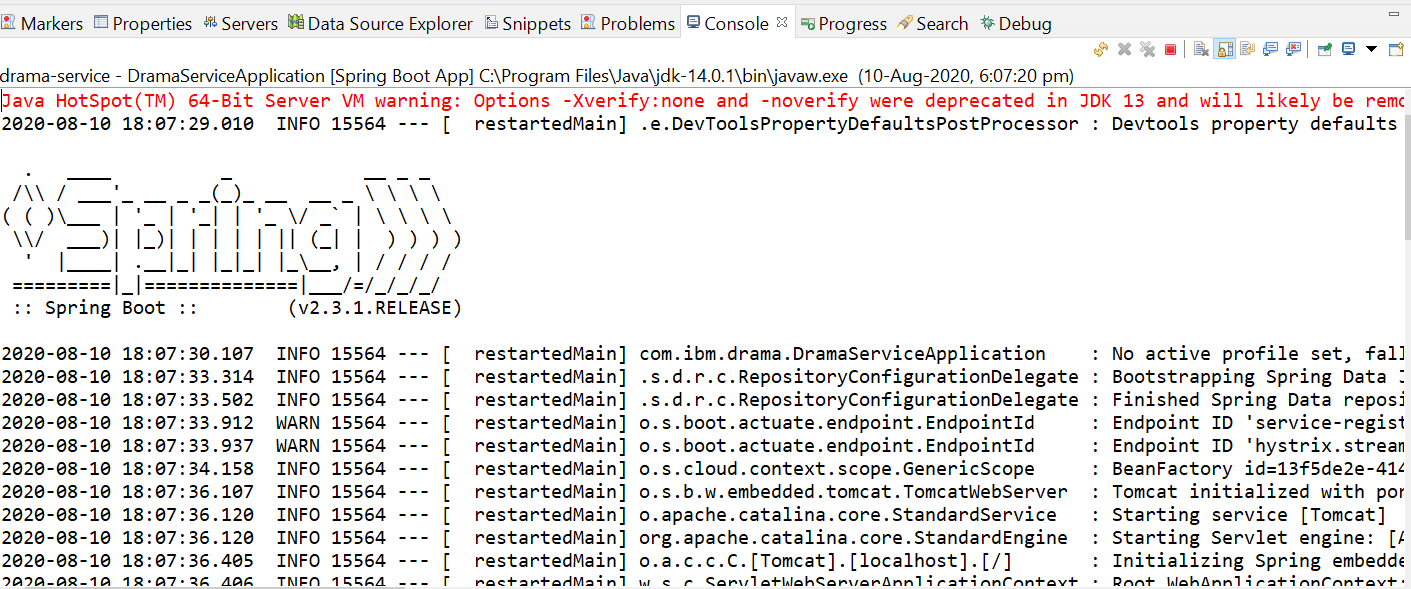
# Register Eureka Client with Eureka server Registry

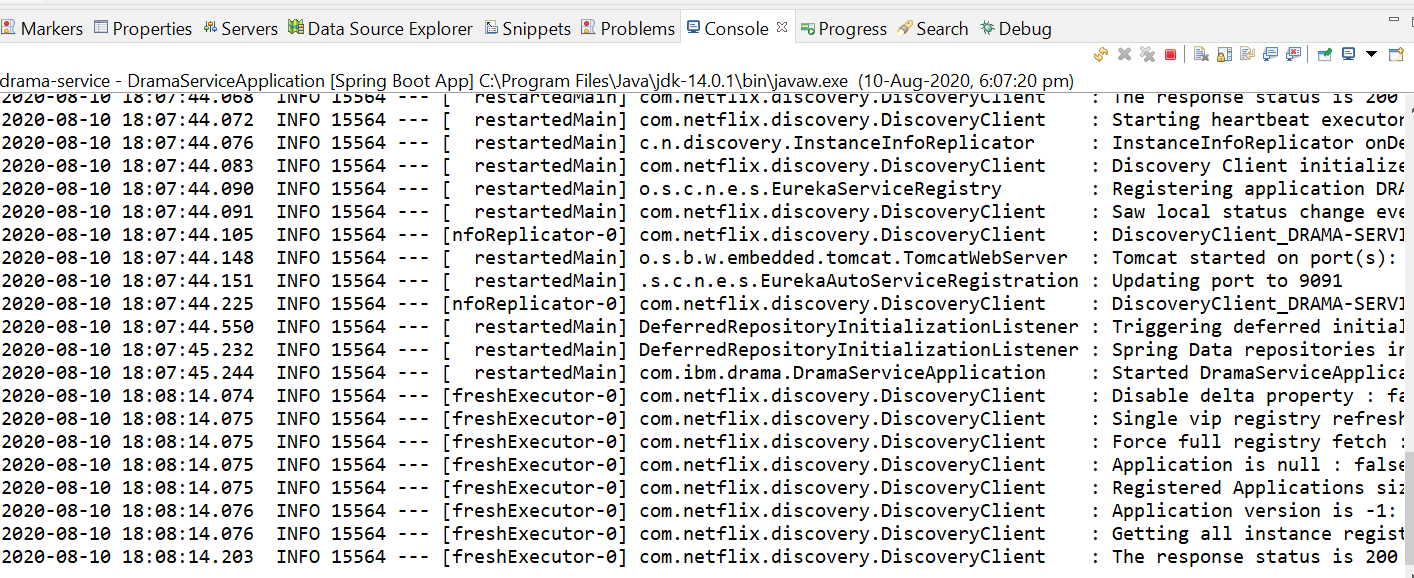
We have registered Eureka Client at port : 9091 & 9092

* 1. Run as Spring boot application

## Run as Spring Boot Application

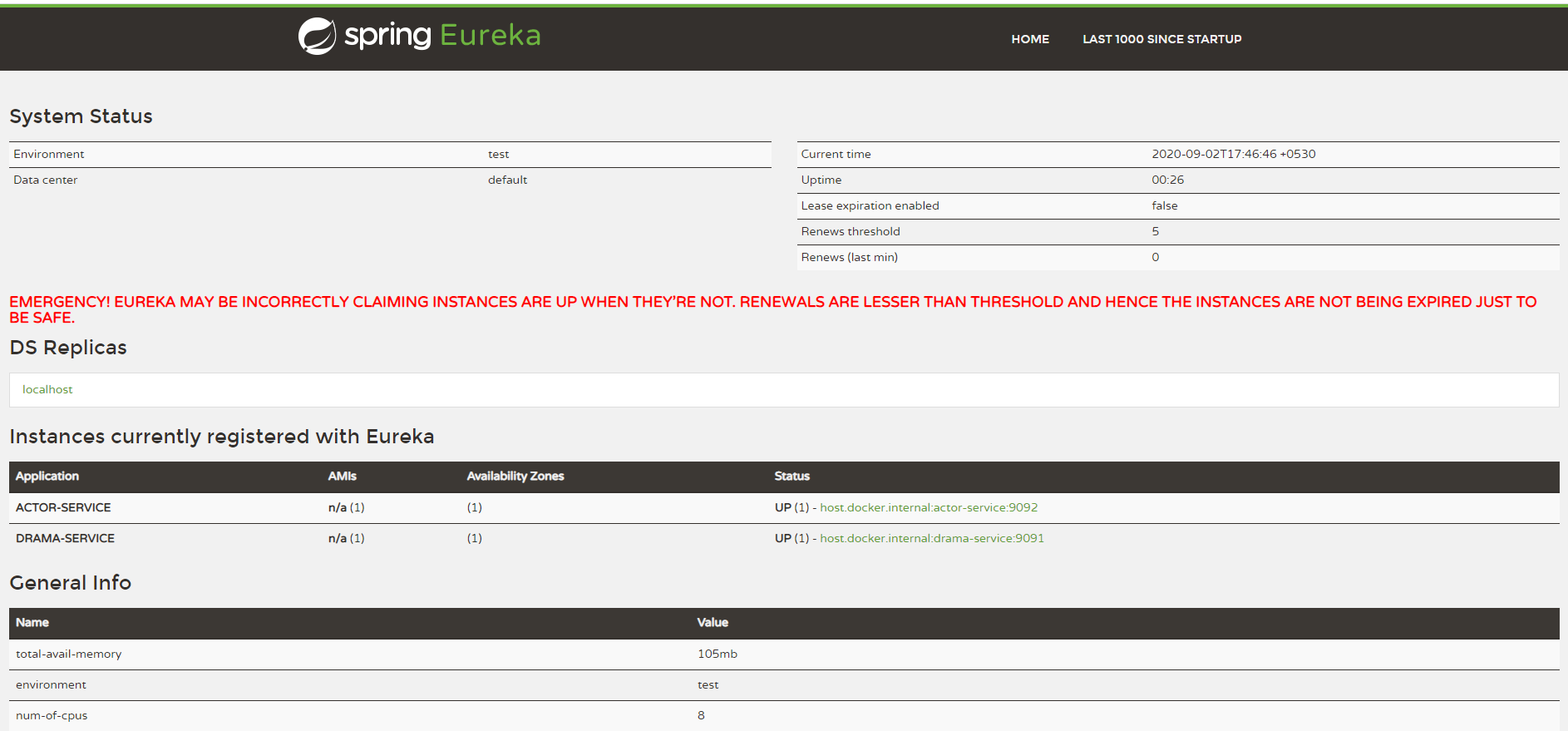
Right click on the project 🡪 Run As 🡪 Spring Boot App





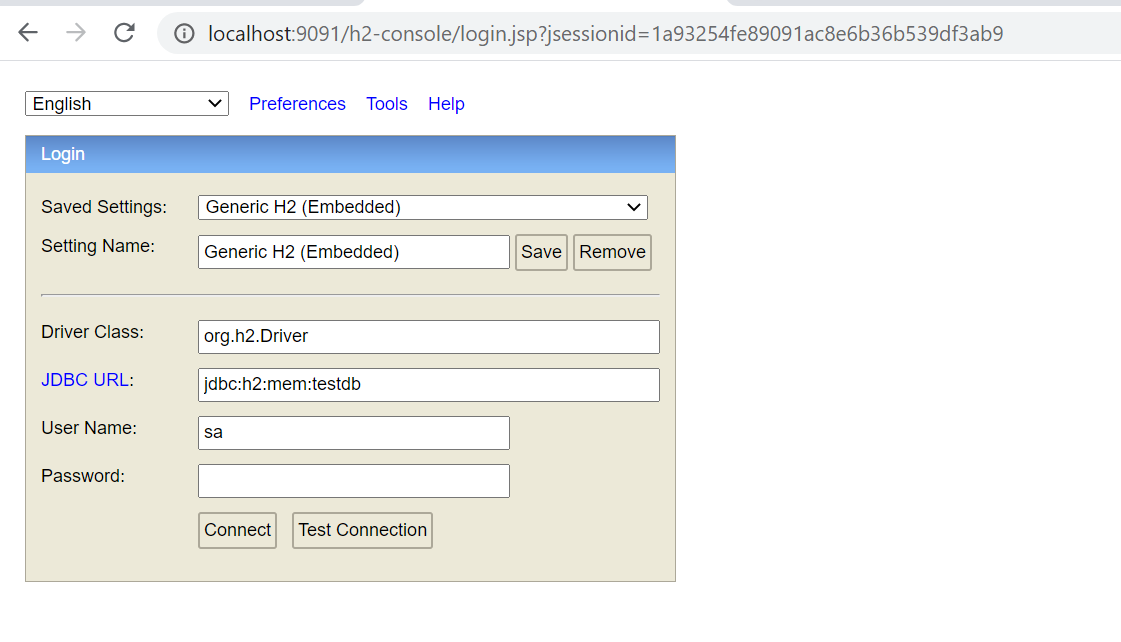
## Access Eureka Server dashboard

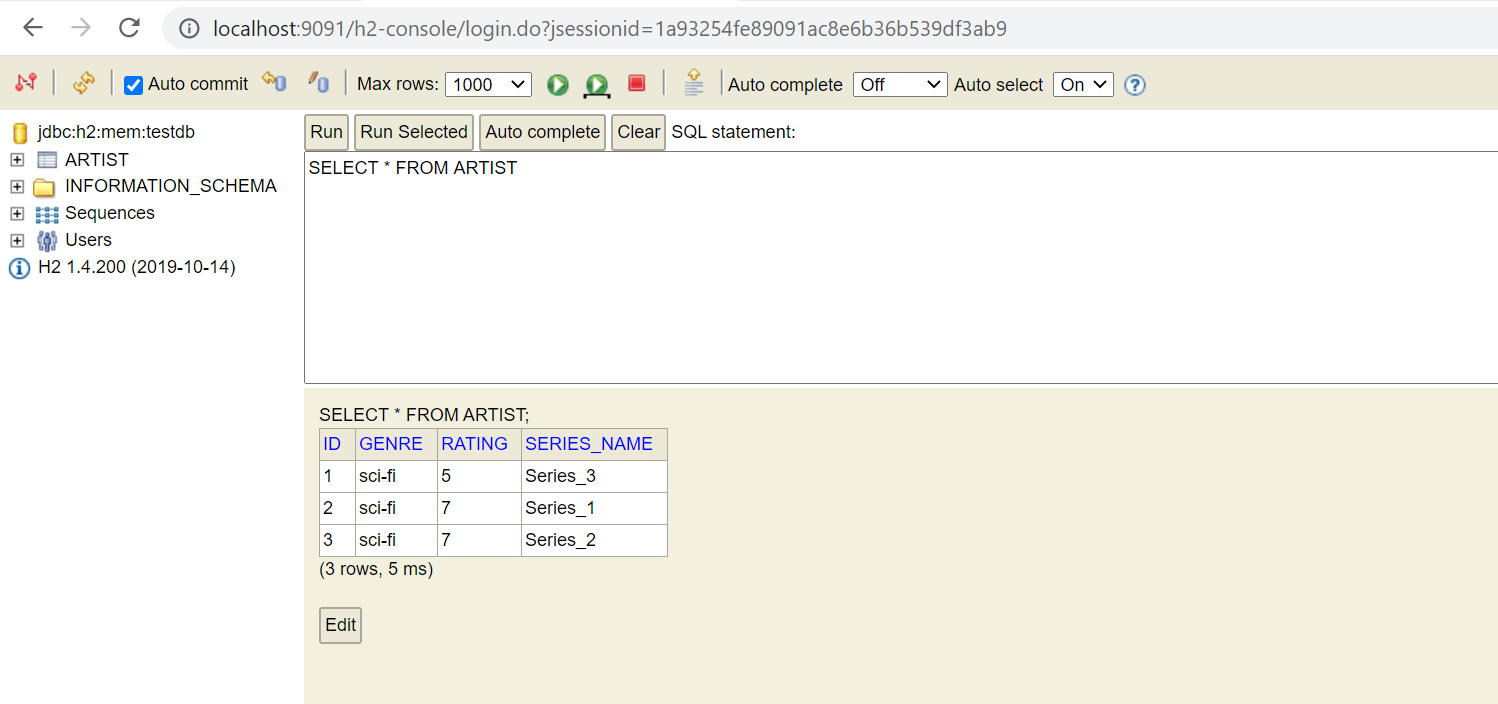
Tomcat started with given port: 9091 & 9092 and application started on JVM. Now the application can be accessed through browser to view Eureka Client registered with Eureka Server on dashboard at <http://localhost:8761/> in up & running state.



## Access H2-database

H2 is in-memory database that can be accessed from browser. This console is accessed to view data.



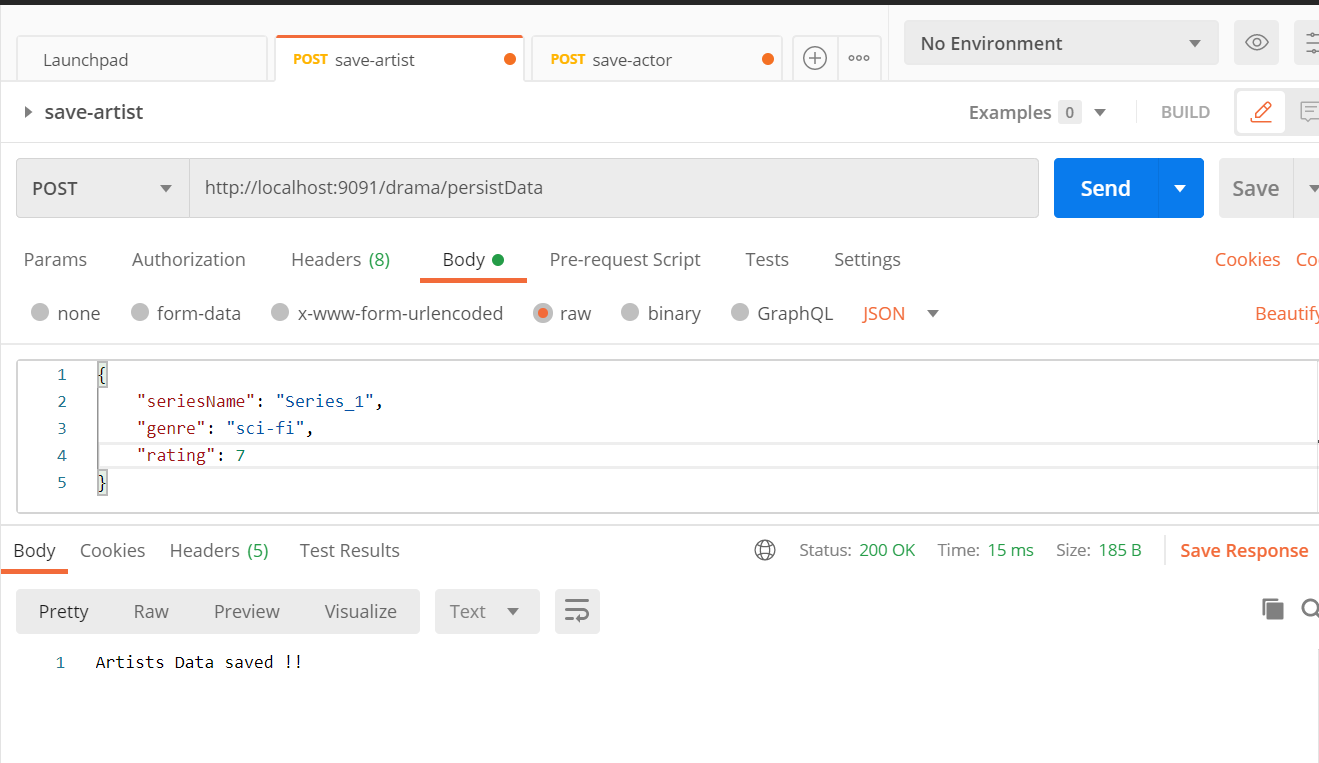


# Test the Microservices

After deploying on Spring boot, the web services can be tested through Postman.

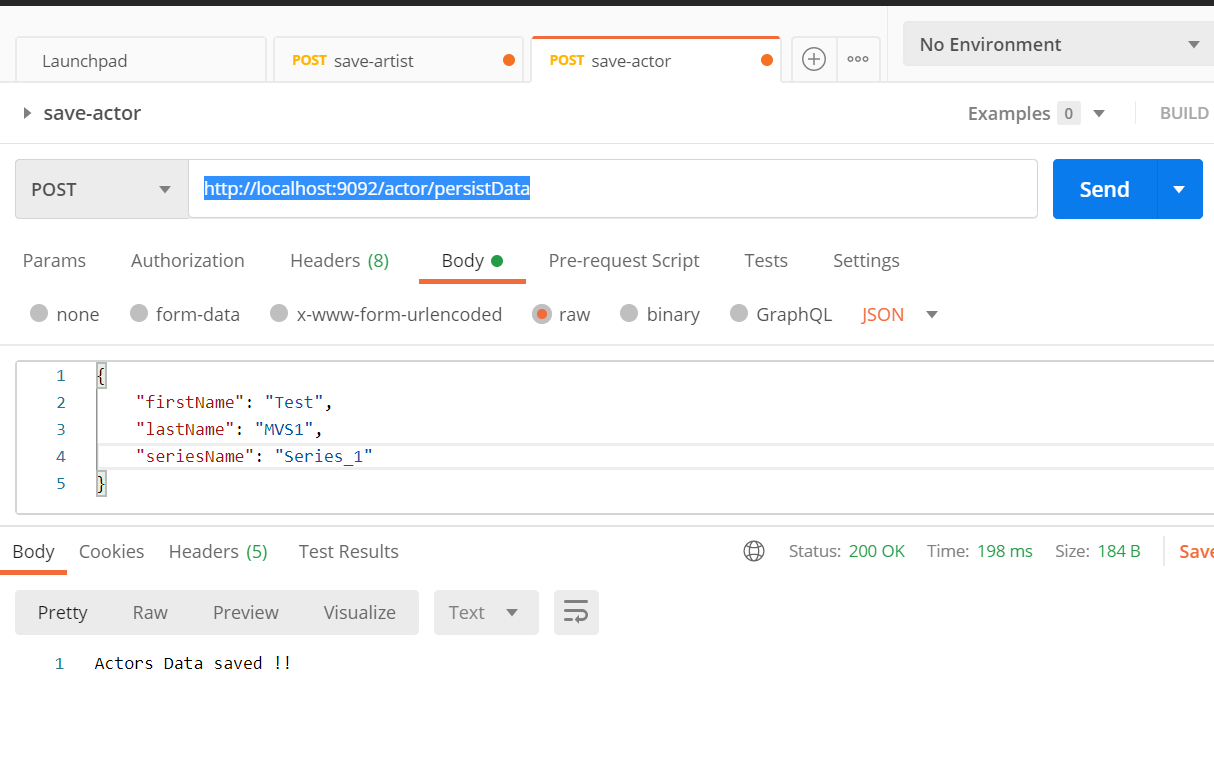
## Create collection in Postman

On Postman, create a new collection and under this collection, create a new request. Enter the URL: <http://localhost:9091/drama/persistData> and send the request in Json body as shown below:

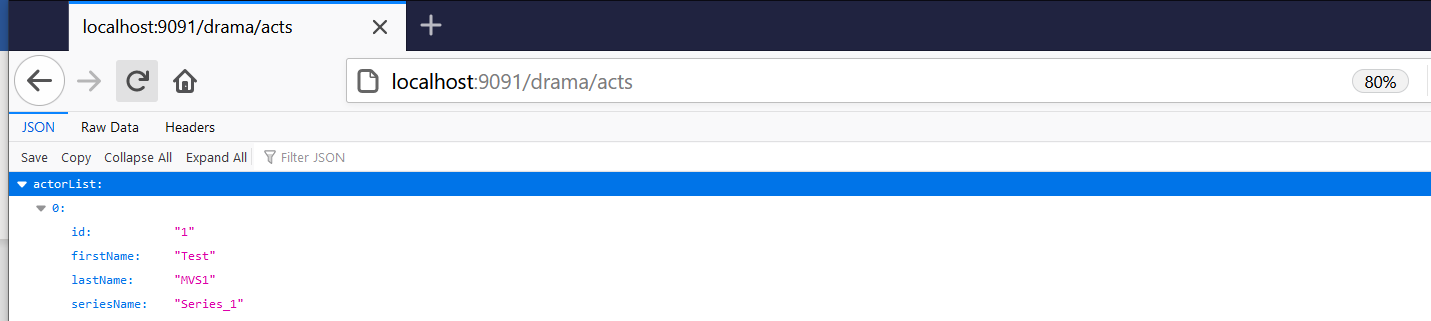


## Create actor-service request

Similarly, create a separate request and enter the URL : <http://localhost:9092/actor/persistData> And send the request in Json body as shown below:

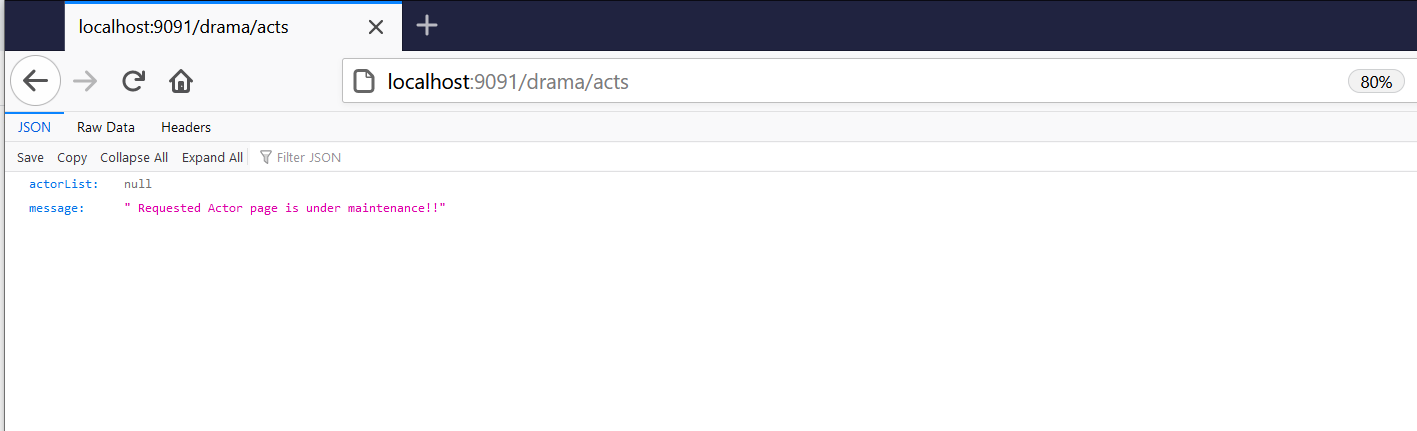


## Microservice Inter-communication

1. It can also be accessed from main service(drama-service) on browser by entering URL : <http://localhost:9091/drama/acts>
2. 

## Hystrix-breaker response

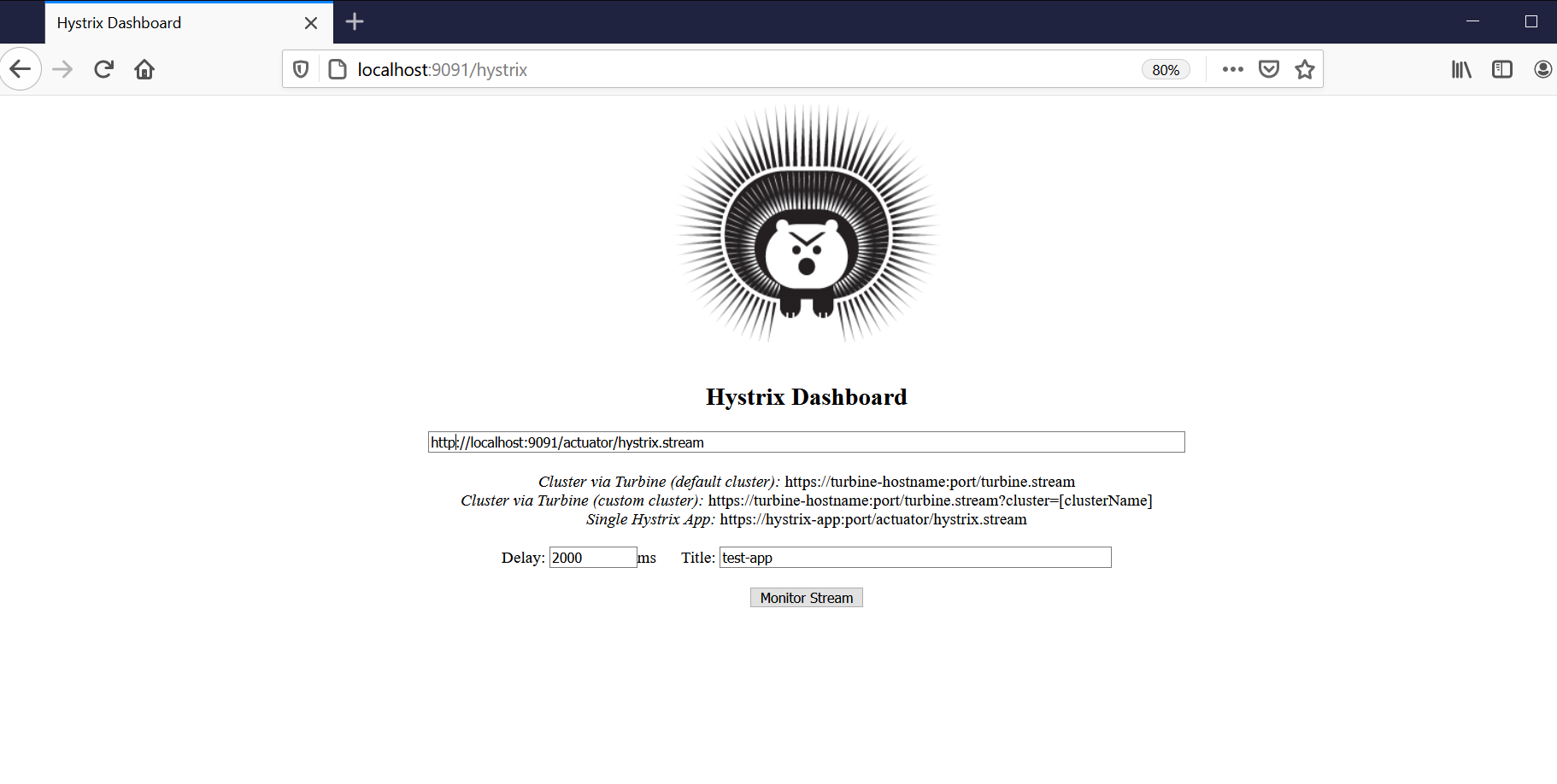
This response gets enabled only when any of the running Microservice goes down from running state. For our demo, We have stopped actor-service and analyzed response in postman.



After the service is restarted, the communication gets established.

## Hystrix-dashboard

Hystrix dashboard enables user to visualize performance metrics. It can be initiated by sending request on URL : <http://localhost:9091/hystrix> and providing input as stated below:



On clicking Monitor Stream, We can visualize performance metrics as shown below :

