**Spring Boot – Eureka Server**

Last Updated : 23 Oct, 2023

In modern microservices architectures, service registration and discovery are crucial components that enable seamless communication between different services. Netflix’s Eureka Server, integrated with Spring Boot, provides an elegant solution for managing these aspects. In this article, we will dive into the configuration and usage of Eureka Server, understanding why it’s essential, exploring its benefits, and considering alternatives.

**Why use Eureka Server?**

Eureka Server is a service registry that plays a central role in the automatic detection of devices and services on a network. It acts as the heart of your microservices ecosystem, allowing service instances to register themselves and facilitating service discovery. Key aspects of Eureka Server include:

* Client Registration: Instances of microservices automatically register themselves with Eureka Server.
* Service Discovery: Eureka Server maintains a registry of all client applications running on different ports and IP addresses.

**Benefits of using Eureka Server in Spring Boot Applications**

Eureka Server operates on a simple “Register, Lookup, Connect” principle, making it an excellent choice for managing microservices in a Spring Boot environment. Here are some compelling reasons to use Eureka Server:

1. Centralized Service Registry: Eureka Server knows about all client applications and their locations. This centralization simplifies service discovery.
2. Automatic Registration: Microservices automatically register themselves with Eureka Server, reducing manual configuration efforts.
3. Load Balancing: Eureka Server can help implement load balancing among service instances.
4. Health Checks: Eureka Server can perform health checks on registered services, ensuring robustness and reliability.
5. Integration with Spring Cloud: Eureka Server seamlessly integrates with the Spring Cloud ecosystem, enabling easy scaling and deployment.

**Use cases of Eureka Server**

Eureka Server finds applications in various scenarios, including:

1. Microservices Architecture: Eureka is a fundamental building block for microservices-based applications.
2. Distributed Systems: It simplifies the management of service discovery in complex, distributed systems.
3. Load Balancing: Eureka can be used in conjunction with load balancers to distribute traffic evenly among service instances.

**1. Configuring the Eureka Server**

**1.1 POM config**

You need to do the below configuration into your project’s pom.xml on server and client side, to get automatic registration and uninterrupted connection of client to server.

* XML

|  |
| --- |
| <!-- For Eureka Client -->  <**dependency**>    <**groupId**>org.springframework.cloud</**groupId**>    <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  </**dependency**>    <!-- For Eureka Server -->  <**dependency**>    <**groupId**>org.springframework.cloud</**groupId**>    <**artifactId**>spring-cloud-starter-netflix-eureka-server</**artifactId**>  </**dependency**> |

***Note****: Ensure that you also have spring-cloud-dependencies defined in your pom.xml below given dependencyManegement in case you are manually importing.*

* XML

|  |
| --- |
| <**dependencyManagement**>    <**dependencies**>      <**dependency**>        <**groupId**>org.springframework.cloud</**groupId**>        <**artifactId**>spring-cloud-dependencies</**artifactId**>        <**version**>2021.0.5</**version**>        <**type**>pom</**type**>        <**scope**>import</**scope**>      </**dependency**>    </**dependencies**>  </**dependencyManagement**> |

**1.2 Bean Configuration**

No bean configuration is required for simple use cases. You’ll mostly use annotations to enable Eureka functionality:

1. @EnableEurekaServer: Use this annotation to designate a Spring Boot application as the central instance where other services will get registered.
2. @EnableEurekaClient: Services use this annotation to enable service registration and discovery with the central Eureka Server.

**1.3 Application Properties**

Your application.properties or application.yml file should contain Eureka-related configuration. For example:

eureka:  
 instance:  
 prefer-ip-address: true  
 client:  
 fetch-registry: true  
 register-with-eureka: true  
 service-url:  
 defaultZone: http://localhost:8761/eureka

***Note****: To identify and locate other microservices with application name “spring.application.name=TEACHER-APP”*

This configuration ensures that the Eureka client instances prefer using IP addresses, fetch the registry, and register themselves with the Eureka Server running on the default “http://localhost:8761”

**Implementing Eureka Configuration**

For marking a Spring Boot Application as an Eureka Server

* Java

|  |
| --- |
| **package** org.geeksforgeeks.registry.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);      }    } |

For marking a Spring Boot Application as an Eureka Client to be used up by the server

* Java

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

**Eureka Dashboard**

When your Spring Boot application is up and running. Hit the default URL – http://localhost:8761. It shows that Eureka has discovered 3 microservices up and running on the device, that are:

* API-GATEWAY
* STUDENT-APP
* TEACHER-APP

*Dashboard of Eureka discovery client*

**Alternatives to Eureka Client**

Eureka is the most popular choice for service registration and discovery, but we do have some alternatives such as Consul and ZooKeeper that provide similar capabilities and may be found more useful for specific use cases. You can also refer to docs for integration-related issues.

**Conclusion**

We have been through the service registration and discovery of services with Eureka Server in your Spring Boot Application. By adding the Eureka Server dependency, configuring application properties, and use case for Eureka Annotations.