# Microservices Communication: Eureka Client

by Shamik Mitra MVB · Jul. 17, 17 · Java Zone · Tutorial

In my previous Java microservices tutorial example, I created a Eureka Server. In this tutorial, I will show you how microservice communication happens. In this tutorial, I will create a Eureka client that communicates with a Eureka server — a microservices service registry.

If you haven't gone through the previous articles, please click here.

Before diving into the code, let me tell you more about the about Eureka's client/server communication.

## Eureka Client/Server Communication

**Talk to the Eureka server**: At first, while the Eureka client is bootstrapped, it tries to talk with the Eureka server in the same Zone. So, suppose a Eureka client instance is in the Asia Zone — it tries to connect to the Asia Zone's Eureka server. If it is not available, then it fails over to another Zone.

It determines its targets using the `eureka.client.serviceUrl.defaultZone` property, so we need to give a URL of the same Zone's Eureka server. And for failing over to another Zone, the Eureka server should be peer connected.

**Register**: Next, the Eureka client/microservices share the instance information with the Eureka server and registers themselves with the {service-id } ( `spring.application.name` ).

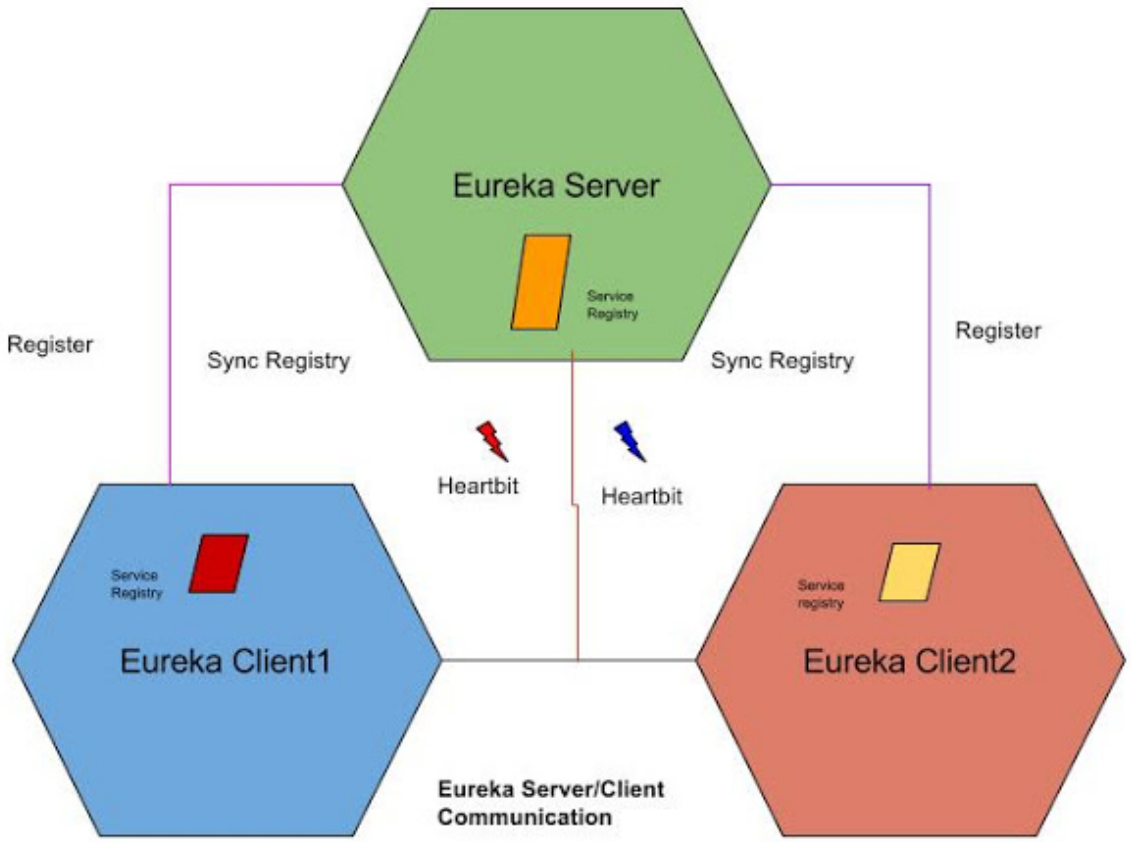
**Heartbeat**: After registration is successful, after every 30 seconds, the Eureka client sends a heartbeat to the Eureka server to renew its leases. So, if after 90 seconds the Eureka server does not get any heartbeat from the Eureka client, it unregisters the Eureka client instance from the service registry and sends the updated registry to all peers and Eureka clients.

**Fetching service registry**: Eureka clients fetch the service registry from the Eureka server so

it can discover other services and communicate with them. After every 30 seconds, this service registry information gets updated by receiving delta updates from the Eureka server. Please note that the Eureka server also caches the delta updates every 3 minutes, so the Eureka client can receive the same delta instances multiple times. After receiving delta updates, it again tries to compare its local registry with the server registry to check that delta is successfully applied. If there are any mismatches, it pulls all the registries again.

**Unregister**: When the client instances are going to shut down, they send a cancel signal to the Eureka client so the Eureka server unregisters it from its registry.

**Synchronization**: Be aware that there may be some time lag as the Eureka server and client manage the cache.




# Coding Time:

We will make the EmployeeSearchService a Eureka client. To do that, first, we need to add the discovery module for Spring Boot in the Maven pom.xml

```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-eureka</artifactId>
</dependency>
```

Now add the `@EnableDiscoveryClient` annotation on top of EmployeeSerachServiceApplication.java

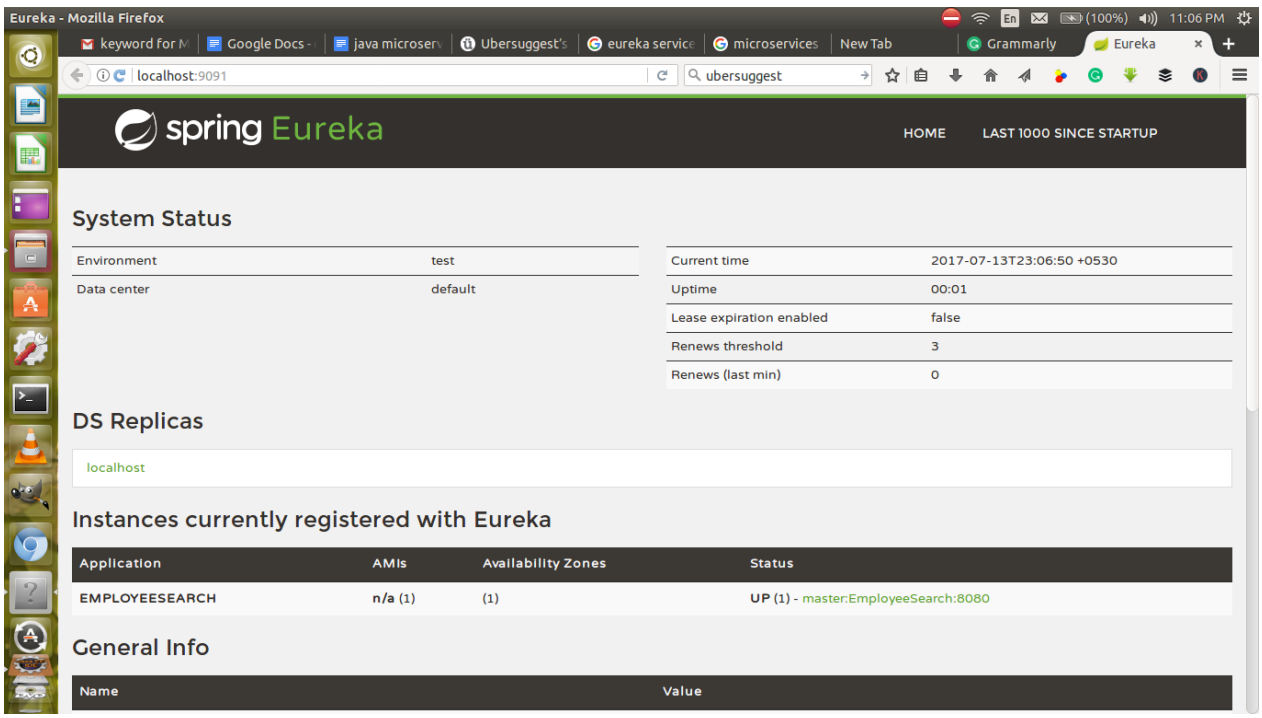
```
package com.example.EmployeeSerachService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.DiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class EmployeeSearchServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeSearchServiceApplication.class, args);
    }
}
```

After that, run the Eureka server first, then `EmployeeSearchApplication` . You will see that the Employee Search application is registered in the Eureka server with the service id EmployeeSearchService.




## Like This Article? Read More From DZone