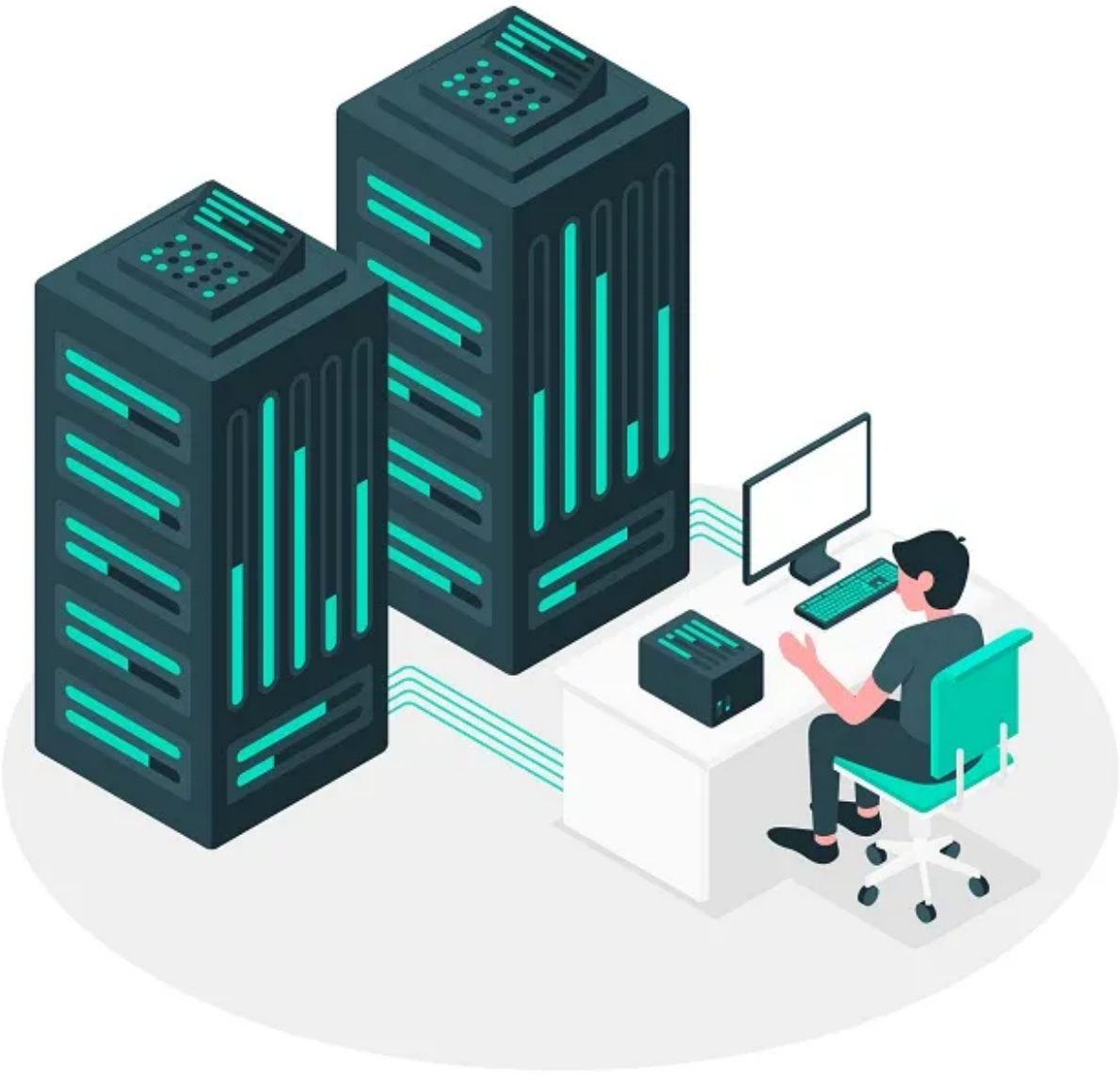
Member-only story

Spring Cloud: High Availability for Eureka

Sarindu Udagepala · Follow

Published in The Startup · 6 min read · Feb 17, 2020



In my previous post, I explained the fundamentals of Service Discovery and Netflix's Eureka. We implemented a stand-alone Eureka server and a client to understand how Eureka enables service discovery. Having a stand-alone Eureka server is not good enough for a production grade deployment. This piece is aimed at exploring how to make Eureka server highly available and resilient.

Throughout this article, I'm planning to cover the below topics.

- How Eureka Achieves High Availability
- Implementing an Eureka Server with High Availability
- Implementing an Eureka Client
- Starting Up Eureka Server Cluster and Client on Local Machine

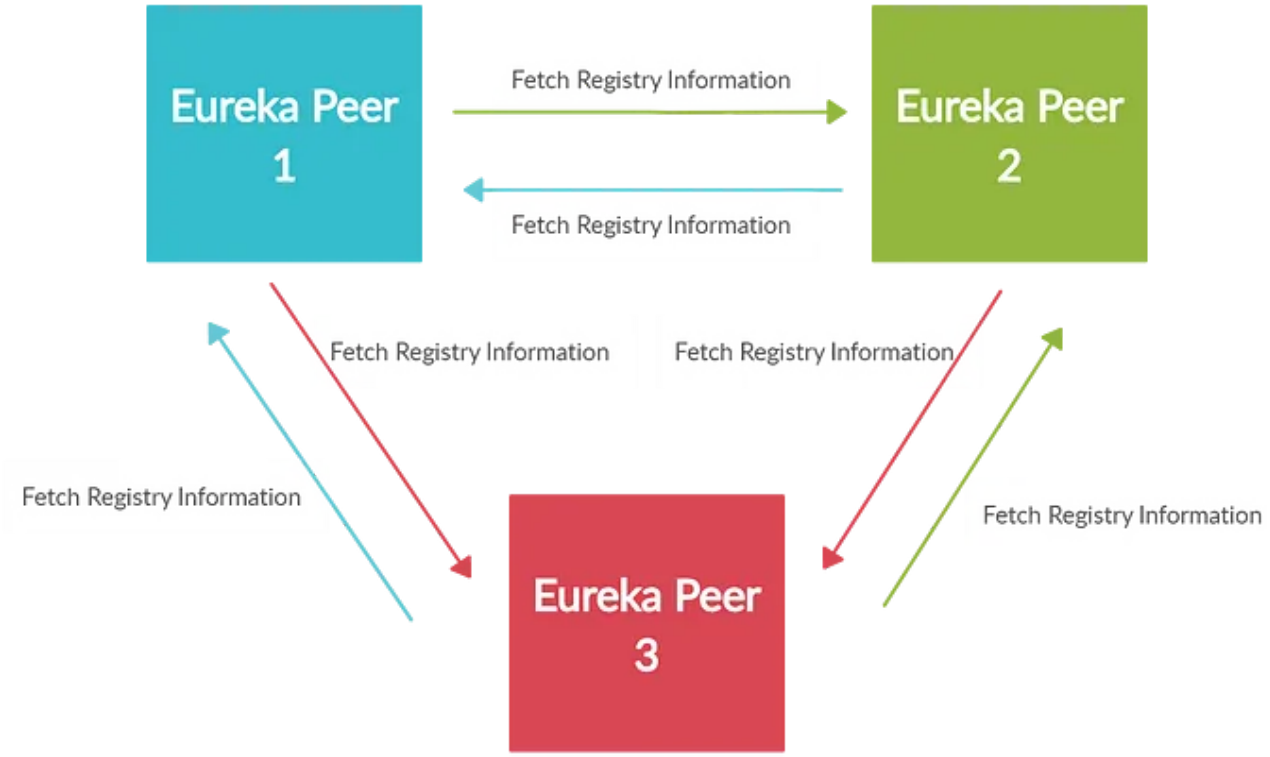
How Eureka Achieves High Availability

Eureka achieves high availability at two levels.

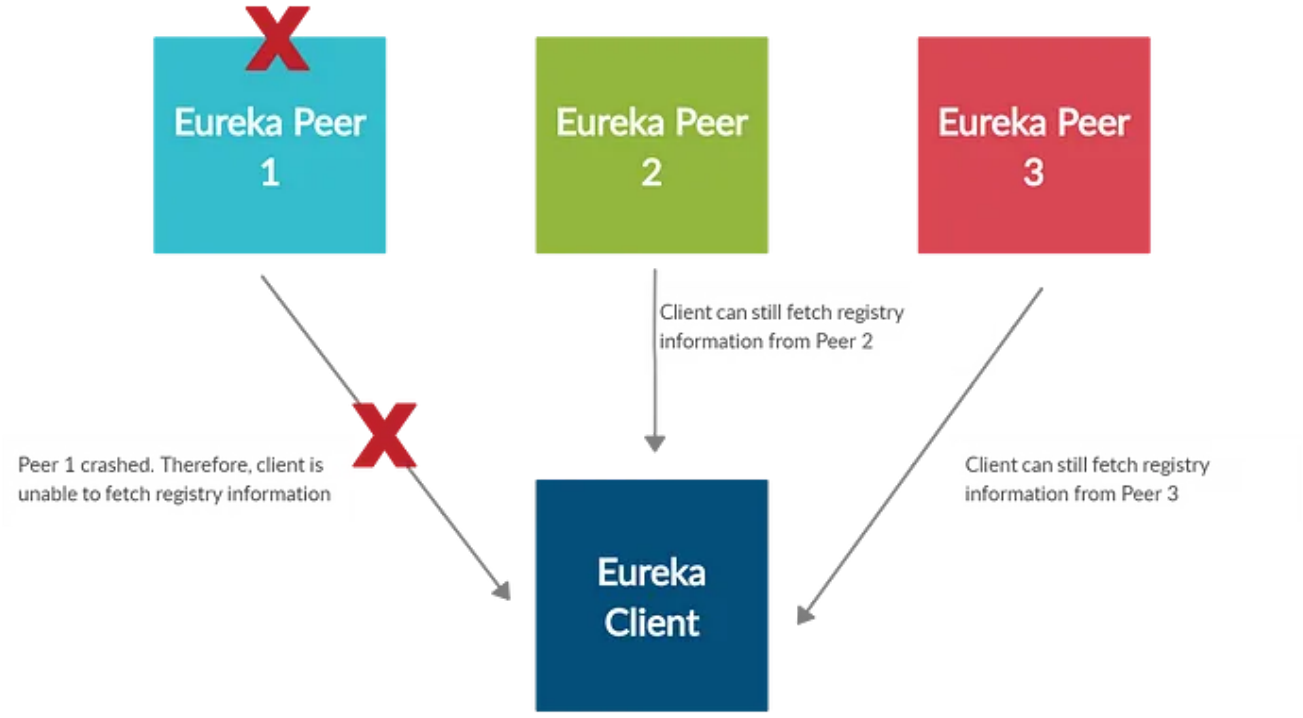
- **Server Cluster:** Eureka can be deployed as a cluster of servers. In case, one of these Eureka servers crash, clients can still connect to the remaining Eureka servers and discover other services.
- **Client Side Caching:** Clients retrieve and cache registry information from Eureka server. In case all Eureka servers crash, clients still posses the last healthy snapshot of the registry. This is the default behavior of Eureka clients and you don't have to make any additional configurations to enable client side caching.

High Availability via Server Cluster

As already stated above, Eureka can be deployed as a cluster of servers. Individual servers of this cluster are called **peers**. When these peers start up, they register with each other and synchronize registrations among themselves. This is also known as **peer awareness**. Following figure illustrates this concept.



Eureka clients are aware of all the available server peers and in case one server crashes, they connect to the remaining servers and fetch registry information.



Implementing an Eureka Server with High Availability

In this section, I will do a quick walk-though on enabling high availability in an Eureka server. Please note that I will not go through step by step but will hit the highlights. As usual, you can find the complete source code here.

“pom.xml” File

```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-i
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.2.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.medium.springcloud</groupId>
        <artifactId>eureka-server</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <name>eureka-server</name>
        <description>Eureka Server</description>

        <properties>
                <java.version>1.8</java.version>
                <spring-cloud.version>Hoxton.SR1</spring-cloud.version>
        </properties>

        <dependencies>
                <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>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
```

```
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>

</project>
```

ha-eureka-server-pom.xml hosted with ❤ by GitHub view raw

- You have to mainly import `spring-cloud` and `spring-cloud-starter-netflix-eureka-server` dependencies in this file.

“application.yml” File — This is where you enable high availability for Eureka server.

```yaml
---
# This default profile is used when running a single instance completely standalone:
spring:
  profiles: default
server:
  port: 9000
eureka:
  instance:
    hostname: default-eureka-server.com
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

---
spring:
  profiles: peer-1
  application:
    name: eureka-server-clustered
server:
  port: 9001
eureka:
  instance:
    hostname: peer-1-server.com
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://peer-2-server.com:9002/eureka/,http://peer-3-server.com:9003/eureka/

---
spring:
  profiles: peer-2
  application:
    name: eureka-server-clustered
server:
  port: 9002
eureka:
  instance:
    hostname: peer-2-server.com
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
```

```
      serviceUrl:
        defaultZone: http://peer-1-server.com:9001/eureka/,http://peer-3-server.com:9003/eureka/

---
spring:
  profiles: peer-3
  application:
    name: eureka-server-clustered
server:
  port: 9003
eureka:
  instance:
    hostname: peer-3-server.com
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://peer-1-server.com:9001/eureka/,http://peer-2-server.com:9002/eureka/
```

ha-application.yml hosted with ❤ by GitHub view raw

- This Eureka server is configured to be run in four different Spring profiles, namely `default`, `peer-1`, `peer-2` and `peer-3`.
- Default profile is used when running a single instance completely standalone. We are not using this profile, in this example.
- Each of the other profiles (`peer-1`, `peer-2` and `peer-3`) will run on separate hosts and ports. And they are also configured to register with each other via the following configuration. Each profile has a configuration similar to this.

```
eureka:
  ...
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://peer-2-server.com:9002/eureka/,http://peer-3-server.com:9003/eureka/
```

“EurekaServerApplication.java” File

```java
package com.medium.springcloud.eurekaserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

}
```

HAEurekaServerApplication.java hosted with ❤ by GitHub view raw

- The `@EnableEurekaServer` annotation is used to make our Spring Boot application acts as a Eureka Server.

Implementing an Eureka Client

As the next step, we have to implement an Eureka client. Key thing to note here is that the client should be aware of all the available server peers.

“pom.xml” File

```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-i
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.2.4.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.medium.springcloud</groupId>
        <artifactId>eureka-client-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>eureka-client-service</name>
        <description>Demo project for Spring Boot</description>

        <properties>
                <java.version>1.8</java.version>
                <spring-cloud.version>Hoxton.SR1</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.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>
```

```
        <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>
```

ha-eureka-client-pom.xml hosted with ❤ by GitHub view raw

- You have to use `spring-boot-starter-web` , `spring-cloud` and `spring-cloud-starter-netflix-eureka-client` dependencies. `spring-boot-starter-web` dependency is used as this application is web service which registers in the Eureka server.

“bootstrap.yml” File

```
spring:
  application:
    name: eureka-client-service
```

ha-eureka-client-bootstrap.yml hosted with ❤ by GitHub view raw

- `spring.application.name` property is used to indicate the service name. Eureka client service registers in the Eureka server with whatever the

name you have provided for this property. Service's network location will be attached to it's service name. This value will be used by other microservices to obtain the network location via the service registry.

- `bootstrap.yml` file is picked up before the `application.yml` file by Spring Boot. `spring.application.name` property is used in the earliest phases of service's configuration. Therefore, by convention, this property resides in the `bootstrap.yml` file.

"application.yml" File — This is where you mention all the available server peers

```yaml
eureka:
  client:
    serviceUrl:
      defaultZone: http://peer-1-server.com:9001/eureka, http://peer-2-server.com:9002/eureka, ht
```

ha-eureka-client-application.yml hosted with ❤ by GitHub view raw

- `eureka.client.serviceUrl.defaultZone` indicates the location of the Eureka server. Client service will use this list of URLs to access the Eureka server application. You have to mention the URLs of all available peers as comma separated values.

"EurekaClientServiceApplication.java" File

```java
package com.medium.springcloud.eurekaclientservice;

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

@SpringBootApplication
@EnableDiscoveryClient
public class EurekaClientServiceApplication {

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

}
```

HAEurekaClientServiceApplication.java hosted with ❤ by GitHub view raw

Starting Up Eureka Server Cluster and Client on Local Machine

As you must have seen already, each of the server peers (`peer-1` , `peer-2` and `peer-3`) will run on separate hosts and ports. In order to start this entire cluster in your local machine, you will have to do some additional configurations.

Starting Eureka Cluster

Step 1 — Edit the hosts file (Windows): You have to first edit the `C:\Windows\System32\drivers\etc\hosts` file and add the below configuration setup. This will allow you to access `peer-1-server.com` , `peer-2-server.com` and `peer-3-server.com` host names in your local machine.

```
127.0.0.1 peer-1-server.com
127.0.0.1 peer-2-server.com
```

```
127.0.0.1 peer-3-server.com
```

Step 2 — Build the Eureka server: Navigate to the home directory of Eureka server project and execute the below Maven command. This command will package your code into a `JAR` file and place it under `<eureka_server_root_directory>/target` folder.

```
mvn clean install
```

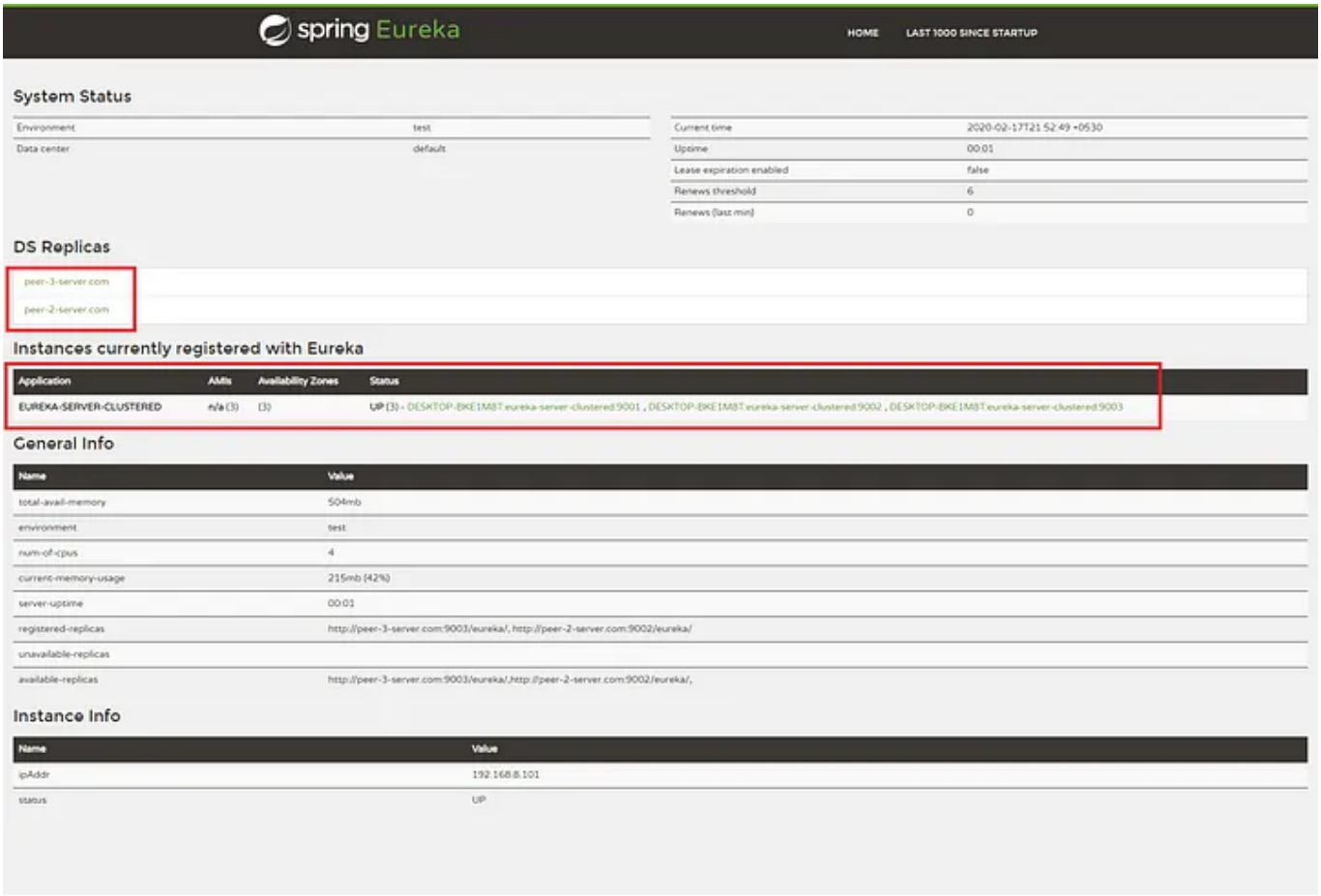
Step 3 — Start the Eureka Server: Navigate to `<eureka_server_root_directory>/target` folder via three different command prompts and and execute the below three commands separately in each command prompt. This will start three instance of Eureka server in `peer-1` , `peer-2` and `peer-3` profiles.

```
java -jar -Dspring.profiles.active=peer-1 eureka-server-1.0.0-
SNAPSHOT.jar
java -jar -Dspring.profiles.active=peer-2 eureka-server-1.0.0-
SNAPSHOT.jar
java -jar -Dspring.profiles.active=peer-3 eureka-server-1.0.0-
SNAPSHOT.jar
```

Step 4: Validating server startup: Navigate to below URLs from your browser and you should be able to see the Eureka server dashboard of each peer.

```
http://peer-1-server.com:9001/
http://peer-2-server.com:9002/
http://peer-3-server.com:9003/
```

Following figure shows how the `peer-1` dashboard looks like. You should be able to see similar dashboards for `peer-2` and `peer-2` as well. Key thing to note here is that all three Eureka server instances have registered in themselves with the name `EUREKA-SERVER-CLUSTERED` .



Starting Eureka Client

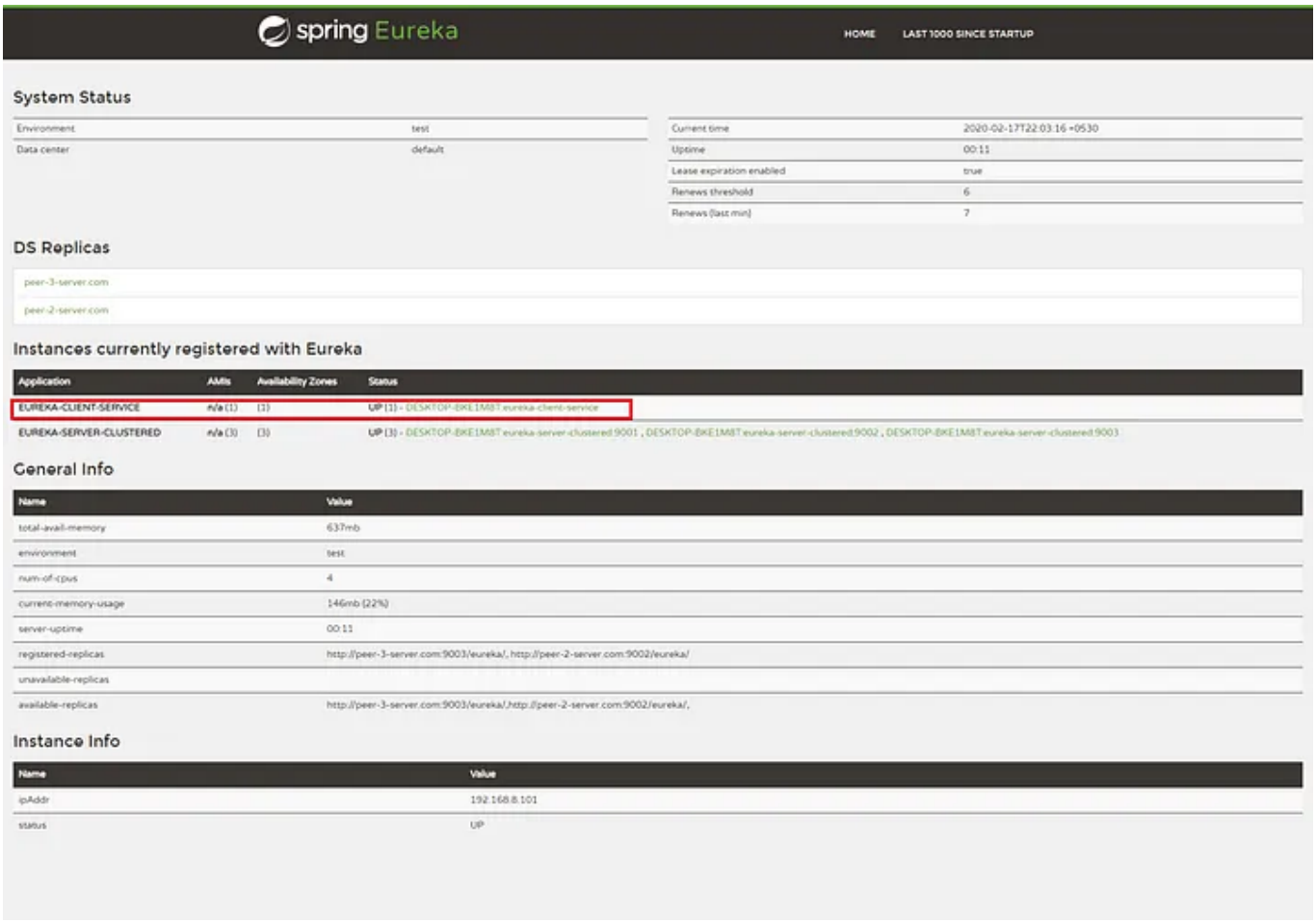
Step 1 — Build Eureka client: Navigate to the root directory of your Eureka client project using the command line, and execute the below Maven command. This command will package your code into a `JAR` file and place it under `<eureka_client_root_directory>/target` folder.

```
mvn install
```

Step 2 — Run Eureka Client: Navigate to the `<eureka_client_root_directory>/target` folder using the command line, and execute the below command to start the Eureka server.

```
java -jar <eureka_client_jar_file_name>
```

Step 3 — Validating client startup: Navigate to below Eureka server URLs from your browser and you should be able to see that the Eureka client information is available in all three Eureka server peers. Client information should appear on the dashboard as follows.



Conclusion

The purpose of this piece was to give you a quick walk-through of configuring a highly available and a resilient Eureka server cluster and test it on your local machine. I hope this knowledge will be helpful for your future microservices projects.

Programming Java Microservices Spring Cloud Eureka

Written by Sarindu Udagepala

341 Followers · Writer for The Startup

I'm a technical lead at WSO2, blogger, son, husband and father. Passionate about technology and learning.

Follow

More from Sarindu Udagepala and The Startup

Sarindu Udagepala in The Startup

Spring Cloud: Service Discovery With Eureka

Learn the fundamentals of Service Discovery and Spring Cloud Eureka with code examples.

· 8 min read · Feb 9, 2020

396 7

Kurtis Pykes in The Startup

Don't Just Set Goals. Build Systems

The Secret To Happiness And Achieving More

· 9 min read · Dec 21, 2022

22K 341

Martina D. in The Startup

11 Companies Making $1M+ That Are [Practically] Bedroom Businesses

Time to feed the ideation part of your brain with some real treats.

· 7 min read · Dec 31, 2023

2.5K 38

Sarindu Udagepala in The Startup

Angular: State Management with Akita

Understand the Fundamentals of Akita and Build a CRUD Application

· 9 min read · May 1, 2020

320 5

See all from Sarindu Udagepala

See all from The Startup

Recommended from Medium

Moussa NIANG

Spring Cloud Gateway with Eureka Service Registry in Spring Boot

This tutorial will walk you through how to use Spring Cloud Gateway to route to the...

3 min read · Sep 30, 2023

4 1

Gagan Kushwaha in Deutsche Telekom Digital Labs

Simplifying Microservices Communication with Spring Cloud...

Microservices have become the go-to architecture for developing modern...

6 min read · Oct 17, 2023

10

Lists

General Coding Knowledge

20 stories · 845 saves

Coding & Development

11 stories · 405 saves

Stories to Help You Grow as a Software Developer
19 stories · 750 saves

ChatGPT
23 stories · 421 saves

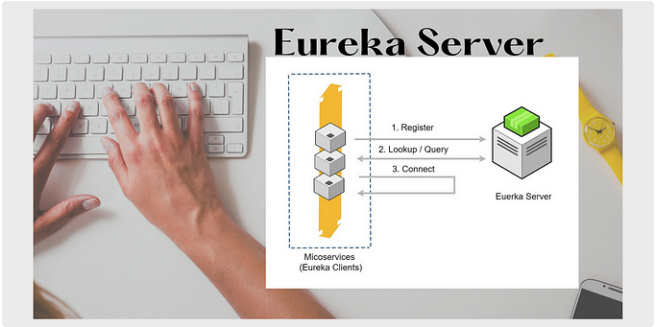
Ankitha Gowda

API gateway in Spring boot

APIs are a common way of communication between applications. In the case of...

7 min read · Oct 7, 2023

132 1



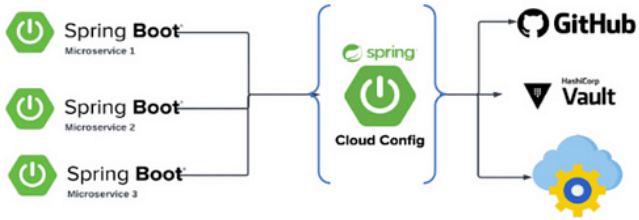
Balakrishnan vishnugan

Eureka Server: The Ultimate Service Registry for Microservices🚀💡🎅

Explore the benefits of using Eureka Server for service discovery in your microservices...

9 min read · Dec 18, 2023

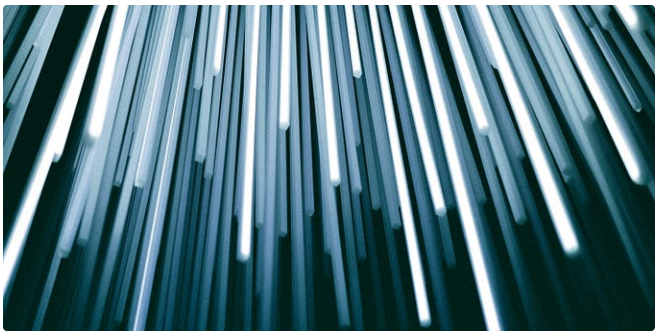
25



Pankaj Sharma in pankajtechblogs

Spring cloud config server — Auto reload config properties — zero-touch

In a distributed system, Spring Cloud Config provides server-side and client-side support...



Abhinav Sonkar in ITNEXT

Distributed Tracing with Spring Boot 3 — Micrometer vs OpenTelemetry

Spring Boot 3 recommends Micrometer for Distributed Tracing. But can OpenTelemetry...

3 min read · Sep 5, 2023

2

✨ · 9 min read · Jan 20

152 1

See more recommendations