

This member-only story is on us. [Upgrade](#) to access all of Medium.

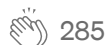
★ Member-only story

Spring Cloud: High Availability for Eureka



Sarindu Udagepala · Follow

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



285



5



Open in app ↗

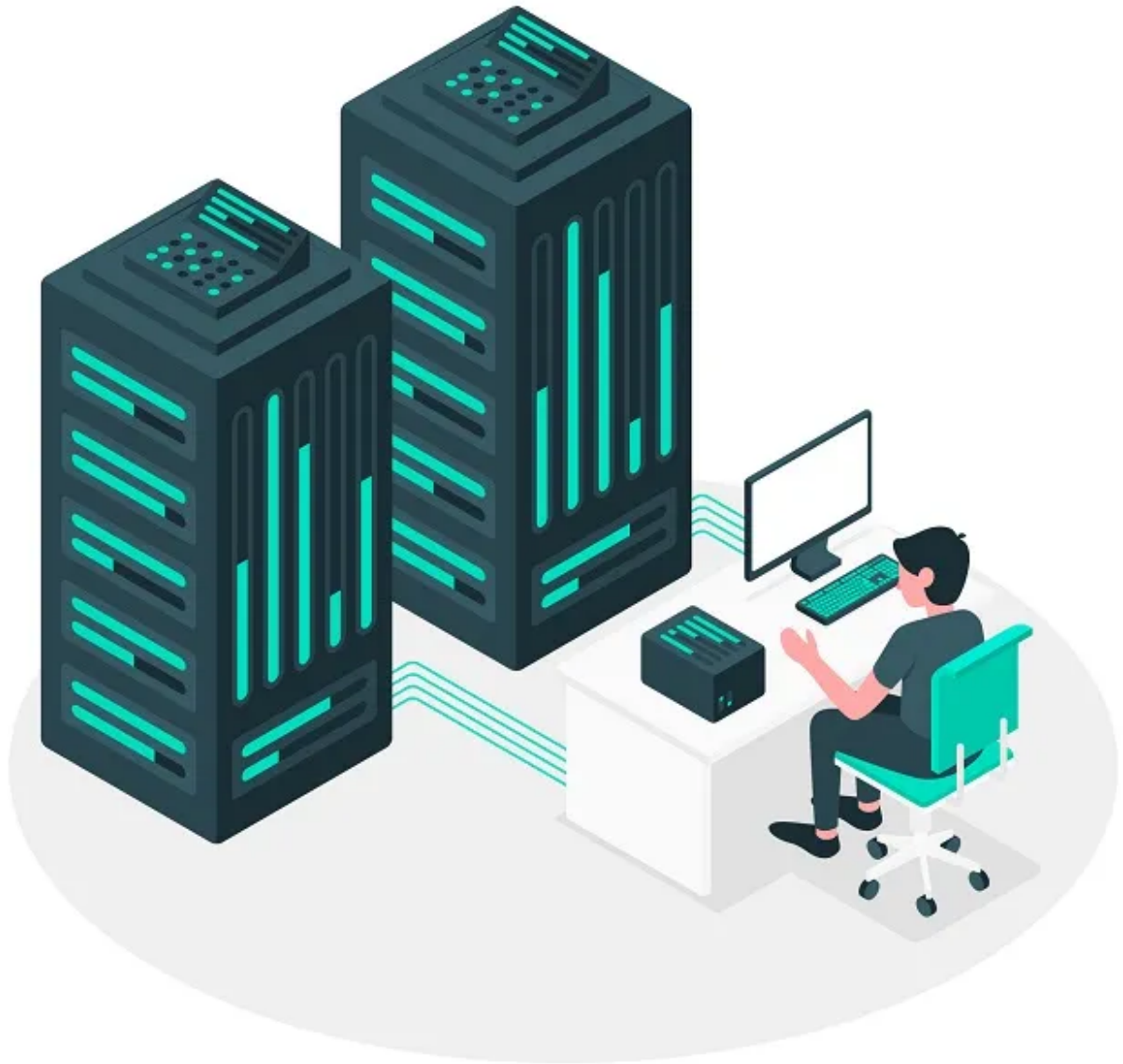


Search



Write





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

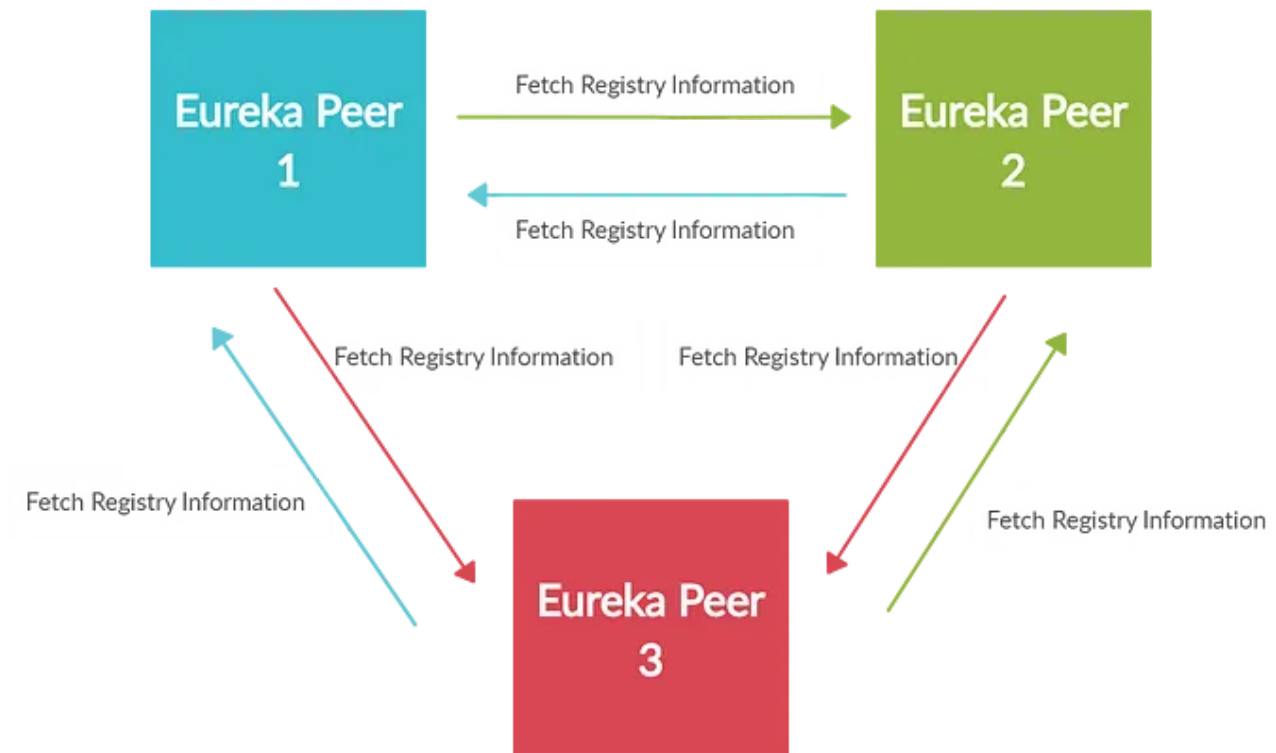
How Eureka Achieves High Availability

Eureka achieves high availability at two levels.

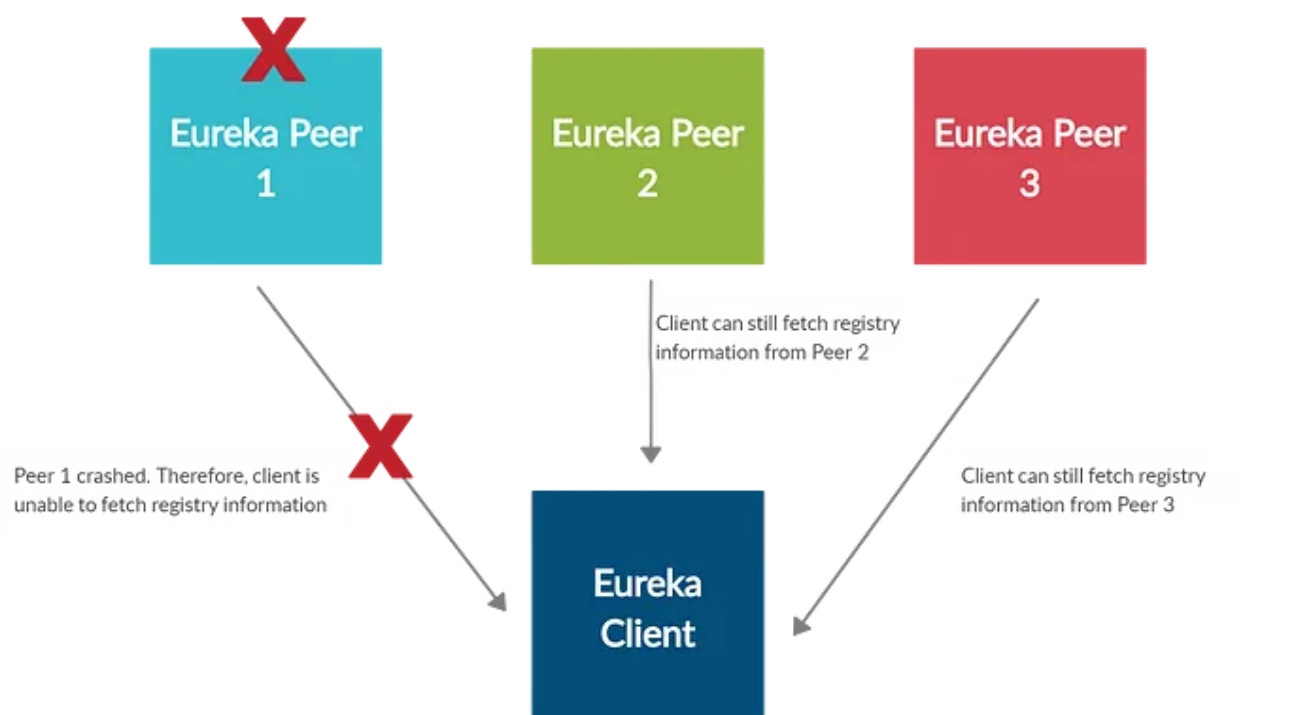
- **Server Cluster:** Eureka can be deployed as a cluster of servers. In case, one of these Eureka servers crash, clients can still connect to the remaining Eureka servers and discover other services.
- **Client Side Caching:** Clients retrieve and cache registry information from Eureka server. In case all Eureka servers crash, clients still possess 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-through 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

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.2.4.RELEASE</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.medium.springcloud</groupId>
12     <artifactId>eureka-server</artifactId>
13     <version>1.0.0-SNAPSHOT</version>
14     <name>eureka-server</name>
15     <description>Eureka Server</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19         <spring-cloud.version>Hoxton.SR1</spring-cloud.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.cloud</groupId>
25             <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
26         </dependency>
27     </dependencies>
28
29     <dependencyManagement>
30         <dependencies>
31             <dependency>
32                 <groupId>org.springframework.cloud</groupId>
33                 <artifactId>spring-cloud-dependencies</artifactId>
34                 <version>${spring-cloud.version}</version>
35                 <type>pom</type>
36                 <scope>import</scope>
37             </dependency>
38         </dependencies>
39     </dependencyManagement>
40
41     <build>
42         <plugins>
43             <plugin>
44                 <groupId>org.springframework.boot</groupId>
45                 <artifactId>spring-boot-maven-plugin</artifactId>

```

```
45         <groupId>org.springframework.boot</groupId>
46         <artifactId>spring-boot-maven-plugin</artifactId>
47     </plugin>
48 </plugins>
49
50 </build>
51
52 </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.

```
1  ---
2  # This default profile is used when running a single instance completely standalone:
3  spring:
4    profiles: default
5  server:
6    port: 9000
7  eureka:
8    instance:
9      hostname: default-eureka-server.com
10   client:
11     registerWithEureka: false
12     fetchRegistry: false
13     serviceUrl:
14       defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
15
16  ---
17  spring:
18    profiles: peer-1
19    application:
20      name: eureka-server-clustered
21  server:
22    port: 9001
23  eureka:
24    instance:
25      hostname: peer-1-server.com
26    client:
27      registerWithEureka: true
28      fetchRegistry: true
29      serviceUrl:
30        defaultZone: http://peer-2-server.com:9002/eureka/,http://peer-3-server.com:9003/eureka/
31
32  ---
33  spring:
34    profiles: peer-2
35    application:
36      name: eureka-server-clustered
37  server:
38    port: 9002
39  eureka:
40    instance:
41      hostname: peer-2-server.com
42    client:
43      registerWithEureka: true
44      fetchRegistry: true
45      serviceUrl:
```



```

46     defaultZone: http://peer-1-server.com:9001/eureka/,http://peer-3-server.com:9003/eureka/
47
48 ---
49 spring:
50   profiles: peer-3
51   application:
52     name: eureka-server-clustered
53   server:
54     port: 9003
55   eureka:
56     instance:
57       hostname: peer-3-server.com
58     client:
59       registerWithEureka: true
60       fetchRegistry: true
61       serviceUrl:
62         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

```
1  package com.medium.springcloud.eurekaserver;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7  @SpringBootApplication
8  @EnableEurekaServer
9  public class EurekaServerApplication {
10
11      public static void main(String[] args) {
12          SpringApplication.run(EurekaServerApplication.class, args);
13      }
14
15  }
```

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

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
3      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.2.4.RELEASE</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.medium.springcloud</groupId>
12     <artifactId>eureka-client-service</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>eureka-client-service</name>
15     <description>Demo project for Spring Boot</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19         <spring-cloud.version>Hoxton.SR1</spring-cloud.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.cloud</groupId>
29             <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
30         </dependency>
31
32         <dependency>
33             <groupId>org.springframework.boot</groupId>
34             <artifactId>spring-boot-starter-test</artifactId>
35             <scope>test</scope>
36             <exclusions>
37                 <exclusion>
38                     <groupId>org.junit.vintage</groupId>
39                     <artifactId>junit-vintage-engine</artifactId>
40                 </exclusion>
41             </exclusions>
42         </dependency>
43     </dependencies>
44
45     <dependencyManagement>

```

```
45 </dependencyManagement>
46 <dependencies>
47     <dependency>
48         <groupId>org.springframework.cloud</groupId>
49         <artifactId>spring-cloud-dependencies</artifactId>
50         <version>${spring-cloud.version}</version>
51         <type>pom</type>
52         <scope>import</scope>
53     </dependency>
54 </dependencies>
55 </dependencyManagement>
56
57 <build>
58     <plugins>
59         <plugin>
60             <groupId>org.springframework.boot</groupId>
61             <artifactId>spring-boot-maven-plugin</artifactId>
62         </plugin>
63     </plugins>
64 </build>
65
66 </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

```
1 spring:
2   application:
3     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 its 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

```
1  eureka:  
2    client:  
3      serviceUrl:  
4        defaultZone: http://peer-1-server.com:9001/eureka, http://peer-2-server.com:9002/eureka, http://peer-3-server.com:9003/eureka
```

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

```
1  package com.medium.springcloud.eurekaclient.service;  
2  
3  import org.springframework.boot.SpringApplication;  
4  import org.springframework.boot.autoconfigure.SpringBootApplication;  
5  import org.springframework.cloud.client.discovery.EnableDiscoveryClient;  
6  
7  @SpringBootApplication  
8  @EnableDiscoveryClient  
9  public class EurekaClientServiceApplication {  
10  
11      public static void main(String[] args) {  
12          SpringApplication.run(EurekaClientServiceApplication.class, args);  
13      }  
14  
15  }
```

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 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-3` as well. Key thing to note here is that all three Eureka server instances have registered in themselves with the name `EUREKA-SERVER-CLUSTERED`.

The screenshot displays the Spring Eureka dashboard for a peer-1 instance. The dashboard is divided into several sections:

- System Status:** A table showing environment details (test, default) and system metrics (Current time: 2020-02-17T21:52:49+0530, Uptime: 00:01, Lease expiration enabled: false, Renewal threshold: 6, Renewal (last min): 0).
- DS Replicas:** A table showing the list of DS replicas (peer-3-server.com, peer-2-server.com).
- Instances currently registered with Eureka:** A table showing the application name (EUREKA-SERVER-CLUSTERED), AMIs (n/a (3)), Availability Zones (3), and Status (UP (3) - DESKTOP-EKE1M8T-eureka-server-clustered9001, DESKTOP-EKE1M8T-eureka-server-clustered9002, DESKTOP-EKE1M8T-eureka-server-clustered9003).
- General Info:** A table showing various system metrics (Name, Value) such as total-avail-memory (504mb), environment (test), num-of-cpus (4), current-memory-usage (215mb (42%)), server-up-time (00:01), registered-replicas (http://peer-3-server.com:9003/eureka/, http://peer-2-server.com:9002/eureka/), unavailable-replicas, and available-replicas (http://peer-3-server.com:9003/eureka/, http://peer-2-server.com:9002/eureka/).
- Instance Info:** A table showing instance details (Name, Value) such as ipAddr (192.168.8.101) and status (UP).

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.

The screenshot displays the Spring Eureka dashboard. At the top, there's a navigation bar with the Spring Eureka logo and links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into several sections:

- System Status:** A table showing environment details.

Environment	test	Current time	2020-02-17T22:03:16+0530
Data center	default	Uptime	00:11
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	7
- DS Replicas:** A list of data center replicas.

peer-3-server.com
peer-2-server.com
- Instances currently registered with Eureka:** A table showing registered instances.

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-BKE1M8T eureka-client-service
EUREKA-SERVER-CLUSTERED	n/a (3)	(3)	UP (3) - DESKTOP-BKE1M8T eureka-server-clustered9001, DESKTOP-BKE1M8T eureka-server-clustered9002, DESKTOP-BKE1M8T eureka-server-clustered9003
- General Info:** A table showing system metrics.

Name	Value
total-avail-memory	637mb
environment	test
num-of-cpus	4
current-memory-usage	146mb (22%)
server-up-time	00:11
registered-replicas	http://peer-3-server.com:9003/eureka/, http://peer-2-server.com:9002/eureka/
unavailable-replicas	
available-replicas	http://peer-3-server.com:9003/eureka/, http://peer-2-server.com:9002/eureka/
- Instance Info:** A table showing instance details.

Name	Value
ipAddr	192.168.8.101
status	UP

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

Follow

341 Followers · Writer for The Startup

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

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



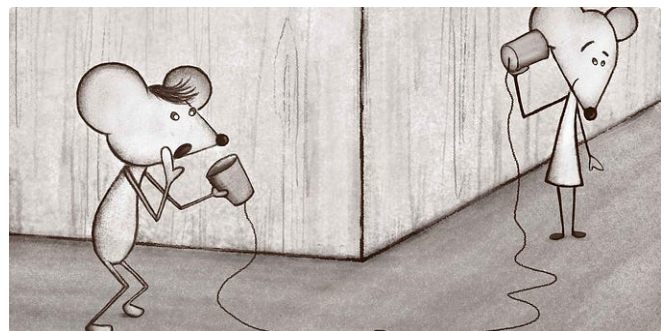
4



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



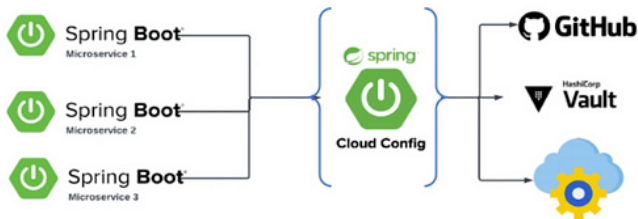
132



1



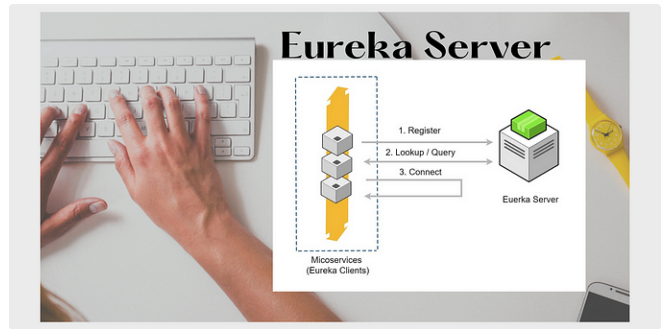
...



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...



Balakrishnan vishnugan

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

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

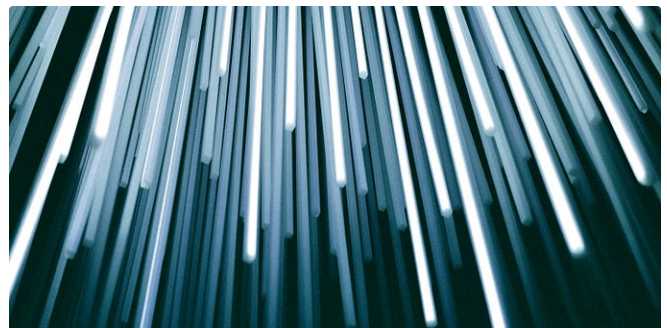
9 min read · Dec 18, 2023



25



...



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

★ · 9 min read · Jan 20



See more recommendations