**How to Create Microservices Architecture with Spring Boot**

<http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html>

**Table of Contents**

1. [**Introduction**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#1)
   1. [**What is Microservices Architecture?**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#1.1)
   2. [**Without Microservices**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#1.2)
   3. [**With Microservices**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#1.3)
2. [**Microservices Benefits**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#2)
3. [**Microservices Challenges**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#3)
4. [**Microservices Infrastructure**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#4)
5. [**Microservices Tooling Supports**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#5)
   1. [**Spring**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#5.1)
      1. [**Spring Boot**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#5.1)
   2. [**Spring Cloud**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#5.2)
      1. [**Eureka by Netflix**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#5.2)
6. [**Developing Simple Microservices Example**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#6)
   1. [**Creating Discovery Service**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#6.1)
   2. [**Creating MicroService (the Producer)**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#6.2)
   3. [**Create Microservice Consumers**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#6.3)
7. [**Summary**](http://www.dineshonjava.com/2017/01/microservices-with-spring-boot.html#7)

**1. Introduction**  
Microservices is not a new term. It coined in 2005 by Dr Peter Rodgers then called micro web services based on SOAP. It became more popular since 2010. Micoservices allows us to break our large system into number of independent collaborating processes. Lets see below microservices architecture.  
  
**1.1 What is Microservices Architecture?**  
Microservices architecture allows to avoid monolith application for large system. It provide loose coupling between collaborating processes which running independently in different environments with tight cohesion. So lets discuss it by an example as below.  
  
For example imagine an online shop with separate microservices for user-accounts, product-catalog order-processing and shopping carts. So these components are inevitably important for such a large online shopping portal. For online shopping system we could use following architectures.  
  
**1.2 Shopping system without Microservices (Monolith architecture)**  
In this architecture we are using Monolith architecture i.e. all collaborating components combine all in one application.



**Spring enables separation-of-concerns**

* ***Loose Coupling***- Effect of changes isolated
* ***Tight Cohesion***- Code perform a single well defined task

**Microservices provide the same strength as Spring provide**

* ***Loose Coupling***- Application build from collaboration services or processes, so any process change without effecting another processes.
* ***Tight Cohesion***-An individual service or process that deals with a single view of data.

There are a number of moving parts that you have to setup and configure to build such a system. For implementing this system is not too obvious you have to knowledge about spring boot, spring cloud and Netflix. In this post I will discuss one example for this architecture before the example lets first discuss about pros and cons of microservices architecture.  
 **2. Microservices Benefits**

* Smaller code base is easy to maintain.
* Easy to scale as individual component.
* Technology diversity i.e. we can mix libraries, databases, frameworks etc.
* Fault isolation i.e. a process failure should not bring whole system down.
* Better support for smaller and parallel team.
* Independent deployment
* Deployment time reduce

**3. Microservices Challenges**

* Difficult to achieve strong consistency across services
* ACID transactions do not span multiple processes.
* Distributed System so hard to debug and trace the issues
* Greater need for end to end testing
* Required cultural changes in across teams like Dev and Ops working together even in same team.

**4. Microservices Infrastructure**

* Platform as a Service like Pivotal Cloud Foundry help to deployment, easily run, scale, monitor etc.
* It support for continuous deployment, rolling upgrades fo new versions of code, running multiple versions of same service at same time.

**5. Microservices Tooling Supports**  
**5.1 Using Spring for creating Microservices**

* Setup new service by using Spring Boot
* Expose resources via a RestController
* Consume remote services using RestTemplate

**5.2 Adding Spring Cloud and Discovery server**  
***What is Spring Cloud?***

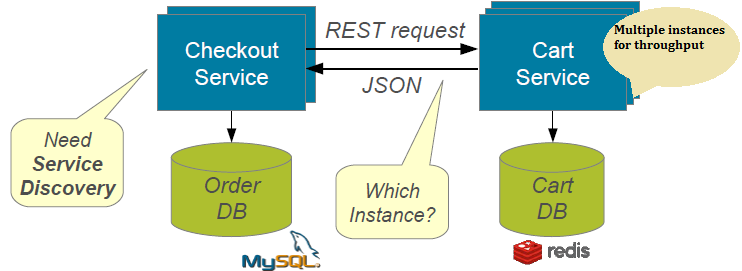
* It is building blocks for Cloud and Microservices
* It provides microservices infrastructure like provide use services such as Service Discovery, Configuration server and Monitoring.
* It provides several other open source projects like Netflix OSS.
* It provides PaaS like Cloud Foundry, AWS and Heroku.
* It uses Spring Boot style starters

There are many use-cases supported by Spring Cloud like Cloud Integration, Dynamic Reconfiguration, Service Discovery, Security,Client side Load Balancing etc. But in this post we concentrate on following microservices support

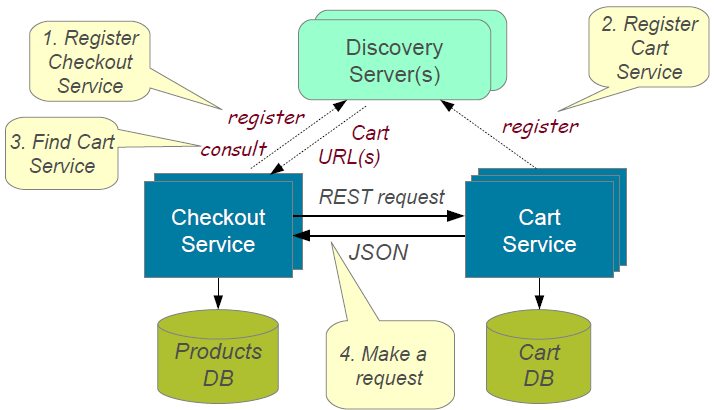
* Service Discovery (How do services find each other?)
* Client-side Load Balancing (How do we decide which service instance to use?)

**Service Discovery**  
**Problem without discovery**

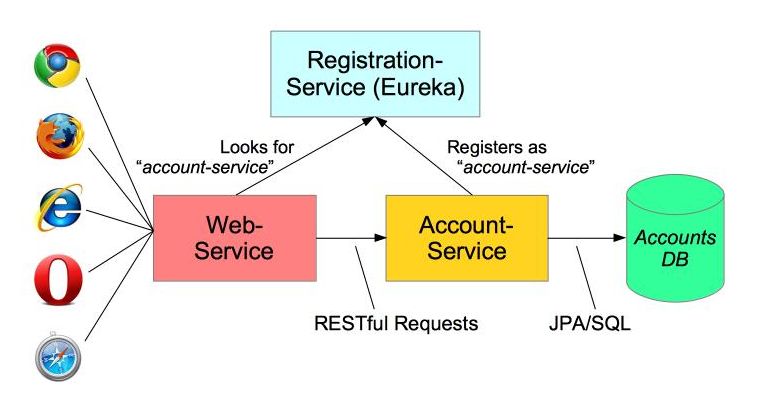
* How do services find each other?
* What happens if we run multiple instances for a service



**Resolution with service discovery**



**Implementing Service Discovery**  
Spring Cloud support several ways to implement service discovery but for this I am going to use Eureka created by Netflix. Spring Cloud provide several annotation to make it use easy and hiding lots of complexity.  
  
**Client-side Load Balancing**  
Each service typically deployed as multiple instances for fault tolerance and load sharing. But there is problem how to decide which instance to use?  
  
**Implementing Client-Side Load Balancing**  
We will use Netflix Ribbon, it provide several algorithm for Client-Side Load Balancing. Spring provide smart **RestTemplate**for service discovery and load balancing by using ***@LoadBalanced***annotation with **RestTemplate**instance.



*@ImageSource-Spring.io*

**6. Developing Simple Microservices Example**  
  
**For build a simple microservices system following steps required**

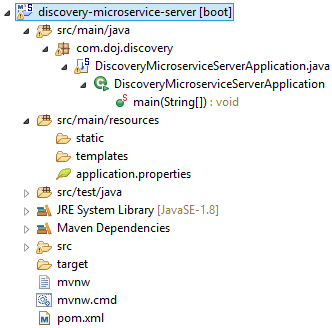
1. Creating Discovery Service (Creating Eureka Discovery Service)
2. Creating MicroService (the Producer)
   1. Register itself with Discovery Service with logical service.
3. Create Microservice Consumers find Service registered with Discovery Service
   1. Discovery client using a smart **RestTemplate**to find microservice.

**Maven Dependencies**

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter**</artifactId>**
5. **</dependency>**
6. **<dependency>**
7. **<groupId>**org.springframework.cloud**</groupId>**
8. **<artifactId>**spring-cloud-starter-eureka**</artifactId>**
9. **</dependency>**
10. **<dependency>**
11. **<groupId>**org.springframework.boot**</groupId>**
12. **<artifactId>**spring-boot-starter-web**</artifactId>**
13. **</dependency>**
14. **<dependency>**
15. **<groupId>**org.hsqldb**</groupId>**
16. **<artifactId>**hsqldb**</artifactId>**
17. **<scope>**runtime**</scope>**
18. **</dependency>**
19. **<dependency>**
20. **<groupId>**org.springframework.boot**</groupId>**
21. **<artifactId>**spring-boot-starter-test**</artifactId>**
22. **<scope>**test**</scope>**
23. **</dependency>**
24. **</dependencies>**

**Step 1: Creating Discovery Service (Creating Eureka Discovery Service)**

* Eureka Server using Spring Cloud
* We need to implement our own registry service as below.



**application.yml**

1. # Configure this Discovery Server
2. eureka:
3. instance:
4. hostname: localhost
5. client: #Not a client
6. registerWithEureka: **false**
7. fetchRegistry: **false**
8. # HTTP (Tomcat) port
9. server:
10. port: 1111

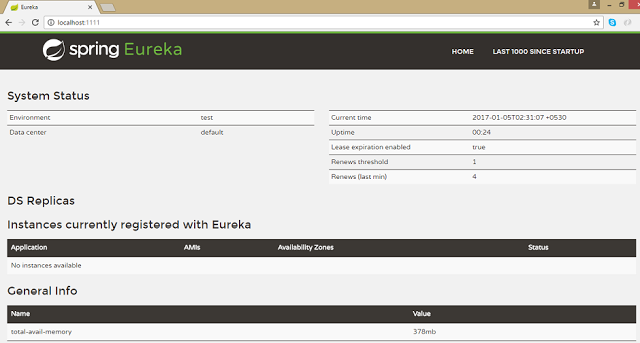
**DiscoveryMicroserviceServerApplication.java**

1. **package** com.doj.discovery;
2. **import** org.springframework.boot.SpringApplication;
3. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
4. **import** org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
5. @SpringBootApplication
6. @EnableEurekaServer
7. **public** **class** DiscoveryMicroserviceServerApplication {
8. **public** **static** **void** main(String[] args) {
9. SpringApplication.run(DiscoveryMicroserviceServerApplication.**class**, args);
10. }
11. }

**pom.xml**

1. <!-- Eureka registration server -->
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter-eureka-server**</artifactId>**
5. **</dependency>**

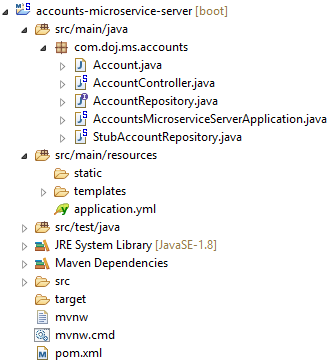
For Whole Source Code for the Discover Server Application you could download from github as below link.  
  
[**discovery-microservice-server**](https://github.com/DOJ-SoftwareConsultant/discovery-microservice-server)  
  
Run this Eureka Server application with right click and run as Spring Boot Application and open in browser **http://localhost:1111/**



**Step 2: Creating Account Producer MicroService**  
Microservice declares itself as an available service and register to Discovery Server created in **Step 1**.

* Using ***@EnableDiscoveryClient***
* Registers using its application name

Lets see the service producer application structure as below.



**application.yml**

1. ### Spring properties
2. # Service registers under this name
3. spring:
4. application:
5. name: accounts-microservice
6. # Discovery Server Access
7. eureka:
8. client:
9. serviceUrl:
10. defaultZone: http://localhost:1111/eureka/
11. # HTTP Server (Tomcat) Port
12. server:
13. port: 2222
14. # Disable Spring Boot's "Whitelabel" default error page, so we can use our own
15. error:
16. whitelabel:
17. enabled: **false**

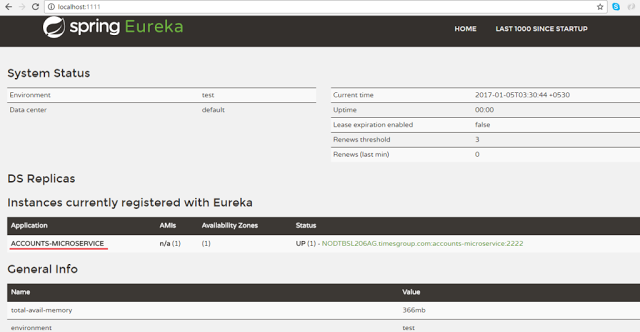
**AccountsMicroserviceServerApplication.java**

1. **package** com.doj.ms.accounts;
2. **import** org.springframework.boot.SpringApplication;
3. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
4. **import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;
5. @SpringBootApplication
6. @EnableDiscoveryClient
7. **public** **class** AccountsMicroserviceServerApplication {
8. **public** **static** **void** main(String[] args) {
9. SpringApplication.run(AccountsMicroserviceServerApplication.**class**, args);
10. }
12. }

**pom.xml**

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter**</artifactId>**
5. **</dependency>**
6. **<dependency>**
7. **<groupId>**org.springframework.cloud**</groupId>**
8. **<artifactId>**spring-cloud-starter-eureka**</artifactId>**
9. **</dependency>**
10. **<dependency>**
11. **<groupId>**org.springframework.boot**</groupId>**
12. **<artifactId>**spring-boot-starter-web**</artifactId>**
13. **</dependency>**
14. **</dependencies>**

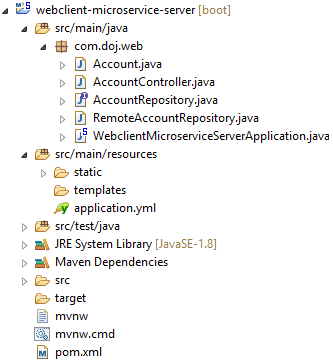
Other required source files related to this application you could download from github link as given below  
  
[**accounts-microservice-server**](https://github.com/DOJ-SoftwareConsultant/accounts-microservice-server)  
  
Now run this account service application as **Spring Boot application** and after few seconds refresh browser to the home page of **Eureka Discovery Server**at **http://localhost:1111/** in previous **Step 1**. Now one Service registered to the Eureka registered instances with Service Name "**ACCOUNT-MICROSERVICE**" as below



**Step 3: Consumer Service**

* Create Consumers to find the Producer Service registered with Discovery Service at Step 1.
* ***@EnableDiscoveryClient*** annotation also allows us to query Discovery server to find miroservices.

Lets see the consumer application structure as below.



**application.yml**

1. # Service registers under this name
2. # Control the InternalResourceViewResolver:
3. spring:
4. application:
5. name: accounts-web
6. mvc:
7. view:
8. prefix: /WEB-INF/views/
9. suffix: .jsp
10. # Discovery Server Access
11. eureka:
12. client:
13. serviceUrl:
14. defaultZone: http://localhost:1111/eureka/
15. # Disable Spring Boot's "Whitelabel" default error page, so we can use our own
16. error:
17. whitelabel:
18. enabled:  **false**

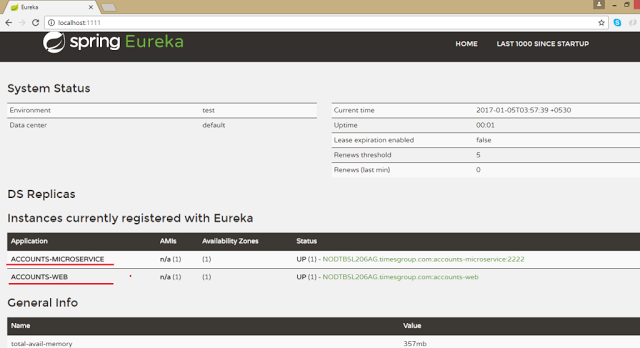
**WebclientMicroserviceServerApplication.java**

1. **package** com.doj.web;
2. **import** org.springframework.boot.SpringApplication;
3. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
4. **import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;
5. **import** org.springframework.cloud.client.loadbalancer.LoadBalanced;
6. **import** org.springframework.context.annotation.Bean;
7. **import** org.springframework.web.client.RestTemplate;
8. @SpringBootApplication
9. @EnableDiscoveryClient
10. **public** **class** WebclientMicroserviceServerApplication {
12. **public** **static** **final** String ACCOUNTS\_SERVICE\_URL = "http://ACCOUNTS-MICROSERVICE";
14. **public** **static** **void** main(String[] args) {
15. SpringApplication.run(WebclientMicroserviceServerApplication.**class**, args);
16. }
18. @Bean
19. @LoadBalanced
20. **public** RestTemplate restTemplate() {
21. **return** **new** RestTemplate();
22. }
23. @Bean
24. **public** AccountRepository accountRepository(){
25. **return** **new** RemoteAccountRepository(ACCOUNTS\_SERVICE\_URL);
26. }
27. }

**pom.xml**

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter-eureka**</artifactId>**
5. **</dependency>**
6. **<dependency>**
7. **<groupId>**org.springframework.cloud**</groupId>**
8. **<artifactId>**spring-cloud-starter-ribbon**</artifactId>**
9. **</dependency>**
10. **<dependency>**
11. **<groupId>**org.springframework.boot**</groupId>**
12. **<artifactId>**spring-boot-starter-web**</artifactId>**
13. **</dependency>**
14. **<dependency>**
15. **<groupId>**org.springframework.boot**</groupId>**
16. **<artifactId>**spring-boot-starter-actuator**</artifactId>**
17. **</dependency>**
18. **<dependency>**
19. **<groupId>**org.springframework.boot**</groupId>**
20. **<artifactId>**spring-boot-starter-test**</artifactId>**
21. **<scope>**test**</scope>**
22. **</dependency>**
23. <!-- These dependencies enable JSP usage -->
24. **<dependency>**
25. **<groupId>**org.apache.tomcat.embed**</groupId>**
26. **<artifactId>**tomcat-embed-jasper**</artifactId>**
27. **<scope>**provided**</scope>**
28. **</dependency>**
29. **<dependency>**
30. **<groupId>**javax.servlet**</groupId>**
31. **<artifactId>**jstl**</artifactId>**
32. **</dependency>**
33. **</dependencies>**

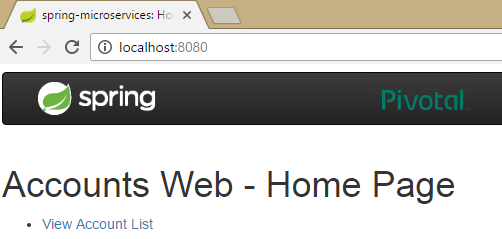
Other required source files related to this application you could download from github link as given below  
  
[**webclient-microservice-server**](https://github.com/DOJ-SoftwareConsultant/webclient-microservice-server)  
  
Now run this consumer service application as **Spring Boot application** and after few seconds refresh browser to the home page of **Eureka Discovery Server**at **http://localhost:1111/** in previous **Step 1**. Now one more Service registered to the Eureka registered instances with Service Name "**ACCOUNTS-WEB**" as below



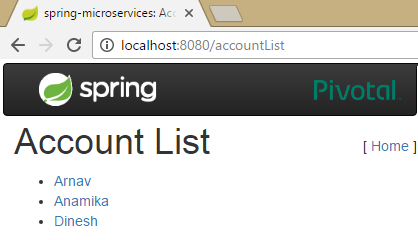
Lets our consumer consume the service of producer registered at discovery server.

1. **package** com.doj.web;
2. **import** java.util.Arrays;
3. **import** java.util.List;
4. **import** org.springframework.beans.factory.annotation.Autowired;
5. **import** org.springframework.web.client.RestTemplate;
6. /\*\*
7. \* @author Dinesh.Rajput
8. \*
9. \*/
10. **public** **class** RemoteAccountRepository **implements** AccountRepository {
12. @Autowired
13. **protected** RestTemplate restTemplate;
15. **protected** String serviceUrl;
17. **public** RemoteAccountRepository(String serviceUrl) {
18. **this**.serviceUrl = serviceUrl.startsWith("http") ? serviceUrl
19. : "http://" + serviceUrl;
20. }
22. @Override
23. **public** List<Account> getAllAccounts() {
24. Account[] accounts = restTemplate.getForObject(serviceUrl+"/accounts", Account[].**class**);
25. **return** Arrays.asList(accounts);
26. }
27. @Override
28. **public** Account getAccount(String number) {
29. **return** restTemplate.getForObject(serviceUrl + "/accounts/{id}",
30. Account.**class**, number);
31. }
32. }

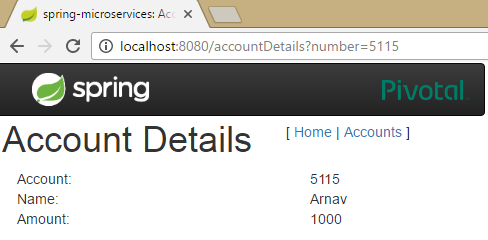
Lets open web application which is a consumer of the account microservice registered at Eureka Discovery Server.   
  
**http://localhost:8080/** as below 



Now click on **View Account List** then fetch all accounts from account microservice.  
  
**http://localhost:8080/accountList**



Now click on any account from the list of accounts to fetch the details of account for account number from account microservice.  
  
**http://localhost:8080/accountDetails?number=5115**



**Load Balanced *RestTemplate***  
**Create using** ***@LoadBalanced***- Spring enhances it to service lookup & load balancing

1. @Bean
2. @LoadBalanced
3. **public** RestTemplate restTemplate() {
4. **return** **new** RestTemplate();
5. }

**Must inject using same qualifier-**

* If there are multiple ***RestTemplate***you get the right one.
* It can used to access multiple microservices

1. @Autowired
2. @LoadBalanced
3. **protected** RestTemplate restTemplate;

**Load Balancing with Ribbon**  
Our smart RestTemplate automatically integrates two Netflix utilities

* ***Eureka***Service Discovery
* ***Ribbon***Client Side Load Balancer

***Eureka***return the URL of all available instances  
***Ribbon***determine the best available service too use  
  
Just inject the load balanced ***RestTemplate***automatic lookup by ***logical service-name***  
  
**7. Summary**  
After completion of this article you should have learned:

* What is the MicroServices Architecture
* Advantages and Challenges of MicroServices
* And some information about Spring Cloud such as Eureka Discover Server by Netflix and Ribbon.