WAY2LEARN

LAB- Using Eureka Server

In this lab you will be understanding how to decompose a monolith into multiple microservices

Running Eureka Server

In this section, you will be working on **02-01-eureka-server-start**

1) Open pom.xml and observe the dependencyManagement Tag. Which version of cloud dependency are we using ?

2) Observe that the following dependency is added .

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

3) Open Eureka.java and observe the code.

4) Add annotation on this class which will start Eureka Server Instance (Hint :@EnableEurekaServer)

5) This is a regular Spring Boot application with one annotation added to enable the service registry

6) What is the port on which Eureka Server is running?

   Open application.yml and observe that we have configured **server.port as 5001**

7) We are running only one Eureka Server. We can plan to run multiple Eureka Servers and in this case , each Eureka Server will act as a client to another Eureka Server and fetch the registry of the Eureka Server.

8) As we have One Eureka Server , in application.yml , we have to configure :- fetch-registry: false and register-with-eureka: false

Configure the following in application.yml .(Don't Copy paste the below code. Use CTRL+SPACE and type it. If you copy paste syntax errors may occur)

```
eureka:
  instance:
    hostname: localhost

  client:
    fetch-registry: false
    register-with-eureka: false
```

9) **Run Eureka.java file** (Embedded Tomcat will start) and Eureka Server will start listening on port 4001
10) Go to Browser and check http://localhost:4001/
11) No instance is currently registered with Eureka

Registering Client with Eureka Server.

Now, you will be working on 02-02-accounts-service-start

We want to start accounts service as a microservice and register it with Eureka Server

We have 2 Controllers in this Applications. Just open AccountController and AuthenticationController and observe the RequestMappings .(Dont go into Code level. You go into code level when you have enough time later)

Open AccountsApplication.java and complete TODO-1 (Hint : Use @EnableDiscoveryClient)

1) For spring cloud modules , bootstrap.yml is the default yml file

2) The property **spring.application.name** is Microservices Application Name with which it will be registered with Eureka Server

3) The property client.serviceUrl.defaultZone has URL to locate the Eureka Server

Open bootstrap.yml and add the following :

```
eureka:
  instance:
    instance-id: ${spring.application.name}:${random.value}

  client:
    serviceUrl:
      defaultZone: http://localhost:5001/eureka
server:
  port: 0
```

If we give server.port as 0, a random port will be used

4) Now **Run AccountsApplication.java** : An Embedded Tomcat will start and this microservices will be registered to the Eureka Server
5) Go to Browser and check eureka server console at http://localhost:5001/
6) You will find that AccountService is registered In that page, can you identify where **spring.application.name** and **eureka.instance.instance-id** are used?
7) Stop the AccountsApplication . Now observe Eureka Server Console . Is the Accounts-Service instance deregistered from Eureka ?

   By default Eureka Server will not deregister if it does not get heart beats from Services. EurekaServer by default starts in Self Preservation Mode to protect instance expiry incase of network /other problems

   You need to turn off the self preservation mode by configuring following property in application.yml

```
eureka:
  server:
       enable-self-preservation: false
```

   Also, on Eureka Server there is a eviction task which will evict the services from registry if it didn't receive heartbeats.

   But default, this eviction task is scheduled to run every 60 seconds. You can reduce the eviction interval by configuring as below : (Be careful to give appropriate value in production. I gave 5 secs just for testing.)

```
eureka:
  server:
    eviction-interval-timer-in-ms: 5000
```

8) Now Stop the eureka server and start it freshly.
9) Now Run **AccountsApplication.java** and observe that the service is registered with eureka server.
10) Now kill the AccountsApplication . Wait for 5 secs and refresh the eureka console page. Did you observe that the service got deregistered ?

    Not yet?

    Yes. By default, when a client is registering with Eureka server , client will sent following two parameters

```
lease-renewal-interval-in-seconds: 30
          lease-expiration-duration-in-seconds: 90
```

    Do you remember what they mean? If not, ask me .

11) Now kill both Eureka Server and AccountsApplication and start them again.
Now open the Eureka server console and
Observe that the service is registered in eureka server console at http://localhost:4001/

12) Now kill AccountsApplication . Wait for 5 secs and refresh the browser. Did you observe that the AccountsService is deregistered?

Eureka Server Clustering

Before you proceed, kill all the java applications running in your STS.

Open Notepad as Administrator and open C:\WINDOWS\System32\drivers\etc\hosts file and configure it as follows :

```
127.0.0.1        eureka-primary
 127.0.0.1        eureka-secondary
 127.0.0.1        eureka-tertiary
```

Now You will be working in 02-01-eureka-server-start project

Now, Configure profiles in application.yml as shown below :

```
eureka:
  server:
    enable-self-preservation: false
  instance:
    hostname: localhost
  client:
    fetch-registry: false
    register-with-eureka: false
server:
  port: 5001

---

spring:
  profiles: eurekaone
server:
  port: 5002
eureka:
  instance:
    hostname: eureka-primary
  client:
    fetch-registry: true
    register-with-eureka: false
    serviceUrl:
      defaultZone:
http://eureka-secondary:5003/eureka,http://eureka-tertiary:5004/eureka



---

spring:
  profiles: eurekatwo
server:
  port: 5003
eureka:
  instance:
    hostname: eureka-secondary
  client:
    fetch-registry: true
    register-with-eureka: false
    serviceUrl:
      defaultZone: http://eureka-primary:5002/eureka,http://eureka-tertiary:5004/eureka

---

spring:
  profiles: eurekathree
server:
  port: 5004
eureka:
  instance:
    hostname: eureka-tertiary
  client:
    fetch-registry: true
    register-with-eureka: false
    serviceUrl:
      defaultZone:
http://eureka-primary:5002/eureka,http://eureka-secondary:5003/eureka
```

Now Run Eureka.java as Spring Boot Application and select profile as eurekaone.

You should observe that Eureka server starts at port 5002.   You should see exceptions on the console as this eureka tries to fetch registries from secondary and tertiary eurekas**. Dont bother about the exceptions as of now.**

Run Eureka.java using profiles eurekatwo and eurekathree.

Go to browser and give requests to http://localhost:5002 , http://localhost:5003, http://localhost:5004 in three separate tabs.

Observe the DS replicas.

Now You will be working on 02-02-accounts-service-start again

Open bootstrap.yml and make sure it has eureka configuration as shown below :

```
eureka:
  instance:
    instance-id: ${spring.application.name}:${random.value}
    lease-renewal-interval-in-seconds: 1
    lease-expiration-duration-in-seconds: 3
  client:
    serviceUrl:
      defaultZone: http://localhost:5002/eureka
```

Now Open Accounts Application.java inside 02-02-accounts-service-start and run it.

So, AccountsApplication Should register to eureka running on 5002.
Since secondary and tertiary eurekas are configured to fetch registries, they should be fetching the details of this registered Accounts Application from primary Eureka.

Now refresh the three tabs on your browser pointing to http://localhost:5002 , http://localhost:5003, http://localhost:5004 . Observe that Accounts service is registered in all the eureka servers.

Congratulations !! You understood how to cluster Eureka servers .

WAY2LEARN

Now, you will be working on 02-03-quotes-service-start

We want to start Quotes service as a microservice and register it with Eureka Server

We   have 1 Controllers in this Application. Just open Quote Controller and observe the RequestMappings .(Dont go into Code level. You go into code level when you have enough time later)

Open QuotesApplication.java and complete TODO-1 (Hint : Use @EnableDiscoveryClient)

1) For spring cloud modules , bootstrap.yml is the default yml file

2) The property **spring.application.name** is Microservices Application Name with which it will be registered with Eureka Server

3) The property client.serviceUrl.defaultZone has URL to locate the Eureka Server

Open bootstrap.yml and add the following   :

```yaml
eureka:
  instance:
    instance-id: ${spring.application.name}:${random.value}

  client:
    serviceUrl:
      defaultZone: http://localhost:5001/eureka
server:
  port: 0
```

If we give server.port as 0, a random port will be used

4) Now **Run QuotesApplication.java** : An Embedded Tomcat will start and this microservices will be registered to the Eureka Server
5) Go to Browser and check http://localhost:5001/
6) You will find that QuoteService is registered

WAY2LEARN

Now, you will be working on 02-04-portfolio-service-solution

We want to start Portfolio service as a microservice and register it with Eureka Server

1) PortfolioService is dependent on another Microservice i.e. AccountService

2) Both services are registered in Eureka Server

3) Portfolio Service has to lookup Eureka Server and get the AccountService

4) In bootstrap.yml , name of all the services are mentioned under which these services are registered with Eureka Server : pivotal.quotesService.name , piovatal.accountsService.name and pivotal.portfolioService.name

   Open PortFolioService.java and complete TODO-1 (HINT :
`@Value("${pivotal.accountsService.name}"))`

5) To get the instance of AccountService , DiscoveryClient.getInstance can be used
   Suppose there are three instances of AccountService running . For all the instances, service name is same .

   when we are looking by name, we get 3 instances.

   As of now , in the logic , we want to get the first instance and get its URI(no load balancing as of now)
   Complete TODO-3 inside addOrder() method of PortfolioService using the below code

```
List<ServiceInstance> instances=  discoveryClient.getInstances(accountsService);
             URI uri=instances.get(0).getUri();
```

6) Further this URI is used to make Restful calls
 Complete TODO -4

Open QuoteRemoteCallService.java and Observe getQuotes() methos and see how we are using discovery client to lookup quoteservice and making restful calls to quoteservice.

7) **Run PortfolioApplication.java** : An Embedded Tomcat will start and this microservices will be registered to the Eureka Server
8) Go to Browser and check http://localhost:5001/
9) You will find that PortfolioService is registered

Now, you will be working on 02-05-way2learntrader-web-start

1) It has three controllers – UserController,TradeController and PortfolioController

2) And it has three Services – UserService , MarketService and MarketSummaryService

3) MarketService depends on QuoteService and PortfolioService (names are injected from bootstrap.yml thru @Value annotation)

   Again DiscoveryClient is autowired and a method getServiceURI is written to get the instance based on serviceName

   Based on the received URI , we are making restful call

   This way WebApplication is communicating with other microservices

4) If your database username/password is different – change in bootstrap.yml and application.yml
5) **Run Way2LearnTraderApp.java** : An Embedded Tomcat will start and this web application will start with embedded tomcat server
10) Go to Browser and check http://localhost:5001/
11) You will find that WebApplication is registered
6) Run the webapplication http://localhost:6060 as per the port in application.yml
7) As it points to same database , same username / password that you have registered last time will work