Technology details used while developing this project along with steps to run project is as below -

Step 1:

Unzip the **GloboMartApp.zip**. There will three folders inside this named -

a) **GlobomartServiceRegistry :** This app works as a service registry for rest of the microservices. I am using eureka library for it. It will also help in achieving scalability along with resiliency (We can start multiple instance for microservice and it will be automatically register with the service registry).

b) **ProductCatalogService:** This contains main microservice app which deal with requested functionality. I have used spring-boot along with spring-cloud, spring-jpa, restassured (for integration test). For backend hsqldb has been used.

c) **ProductCatalogClient:** This is a demo application used to demonstrate client side load balancing while using ProductCatalogService. I am using ribbon library to achieve this.

Step 2: Open terminal and moved to directory **GlobomartServiceRegistry** -

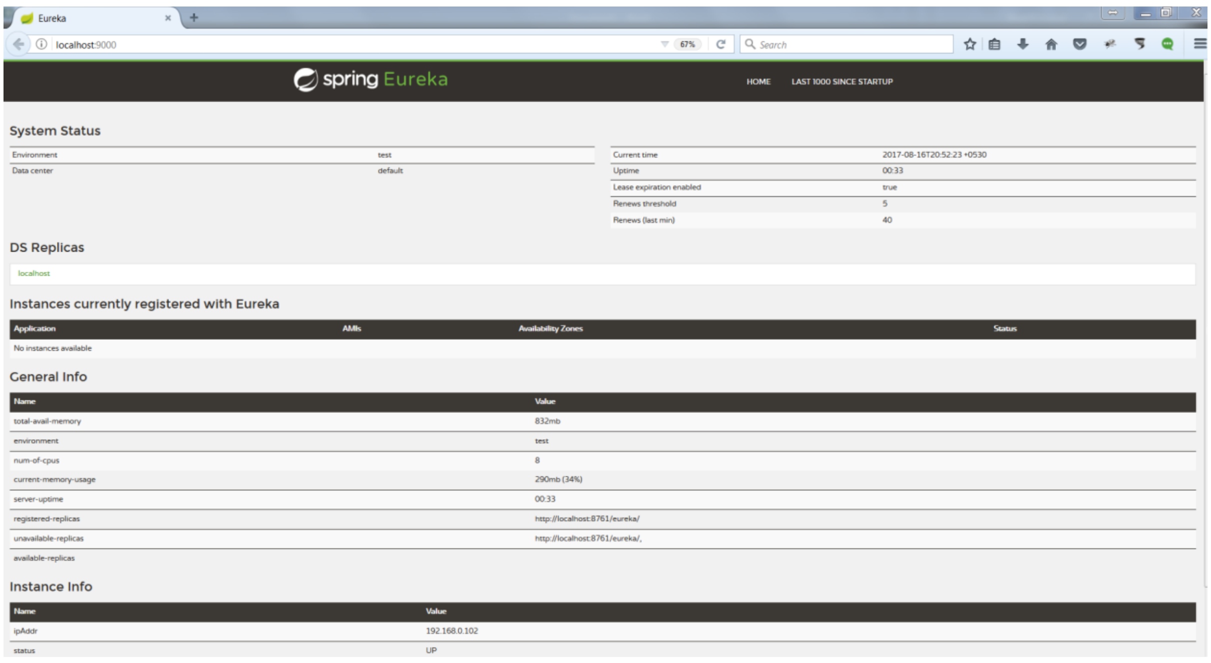
e.g.

```
$ cd GlobomartServiceRegistry

$ mvn clean install

$ mvn spring-boot:run  -Dserver.port=9000
```

This will make our service registry up and running. Ideally we should see following screen if we hit http://localhost:9000



Eureka

localhost:9000

spring Eureka — HOME — LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2017-08-16T20:52:23 +0530
Uptime	00:33
Lease expiration enabled	true
Renews threshold	5
Renews (last min)	40

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	832mb
environment	test
num-of-cpus	8
current-memory-usage	290mb (34%)
server-uptime	00:33
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	192.168.0.102
status	UP

Step 3: Let's start our product catalog service -

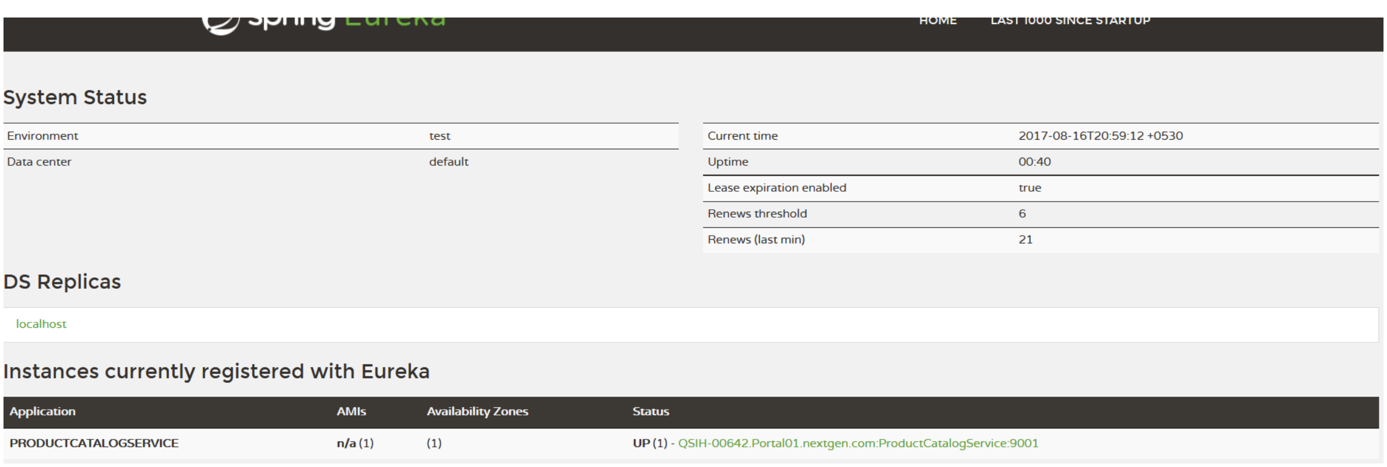
e.g -

```
$ cd ProductCatalogService

$ mvn clean install

$ mvn spring-boot:run  -Dserver.port= 9001
```

Now our product catalog service is up and running. We can verify it by refreshing our service registry URL e.g. - http://localhost:9000



System Status

Environment	test
Data center	default

Current time	2017-08-16T20:59:12 +0530
Uptime	00:40
Lease expiration enabled	true
Renews threshold	6
Renews (last min)	21

DS Replicas

localhost

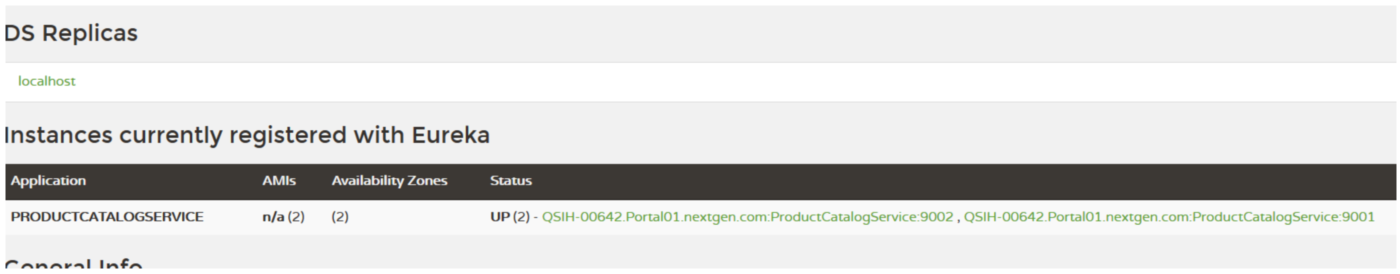
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCTCATALOGSERVICE	n/a (1)	(1)	UP (1) - QSIH-00642.Portal01.nextgen.com:ProductCatalogService:9001

For testing purpose we can start multiple instance by changing port. E.g. -

```
$ mvn spring-boot:run -Dserver.port= 9002
```

Newly started instance will be automatically register with service registry –



DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCTCATALOGSERVICE	n/a (2)	(2)	UP (2) - QSIH-00642.Portal01.nextgen.com:ProductCatalogService:9002 , QSIH-00642.Portal01.nextgen.com:ProductCatalogService:9001

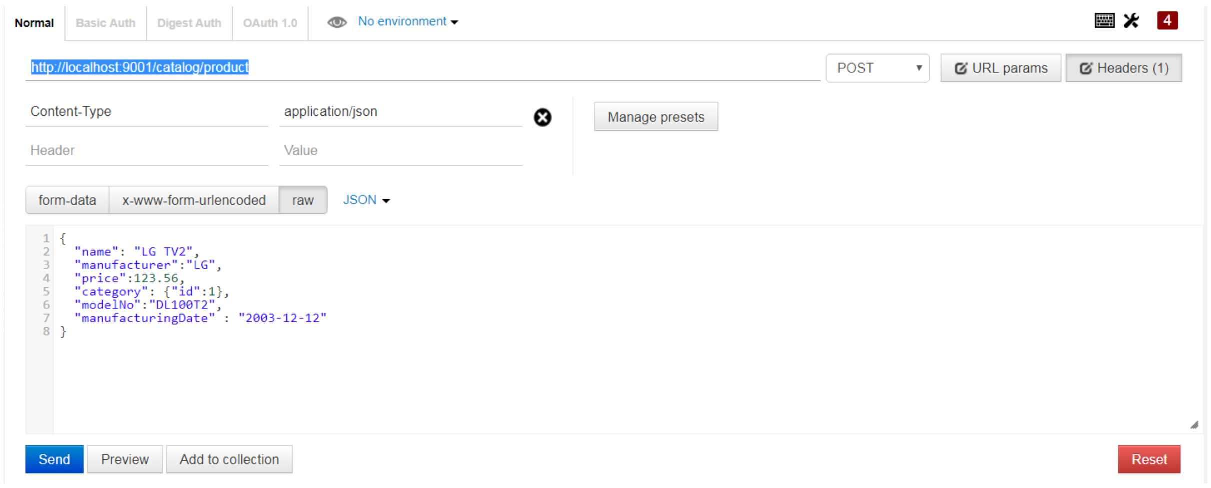
Some of the exposed end point is as below –

GET http://localhost:9001/catalog/products?page=0&size=3

GET http://localhost:9001//catalog/products?sort=id,desc

GET http://localhost:9001/catalog/products/{productId}

POST http://localhost:9001/catalog/products



http://localhost:9001/catalog/product

POST

Content-Type	application/json
Header	Value

```
{
  "name": "LG TV2",
  "manufacturer":"LG",
  "price":123.56,
  "category": {"id":1},
  "modelNo":"DL100T2",
  "manufacturingDate" : "2003-12-12"
}
```

GET http://localhost:9001//catalog/products/search?productType=books

Etc..

Step 4: Now test out the product catalog service by using an application client -

e.g -

```
$ cd ProductCatalogClient

$ mvn clean install

$ mvn spring-boot:run  -Dserver.port= 9003
```

Only one end point has been exposed to test out –

GET http://localhost:9003/productCatalog

This application client don't use direct endpoint for the product catalog service rather get the service instance from the service registry. I have included ribbon library to achieve client side load balancing if we have multiple instances running for product catalog service. We can easily verify it in log different call of the above rest endpoint will be served by different productcatalogservice instances. If we kill one of the instance ribbon will automatically fall back to running instance by contacting service registry. May be couple of call will fail if we try to call the endpoint frequently as service registry refresh takes some time.