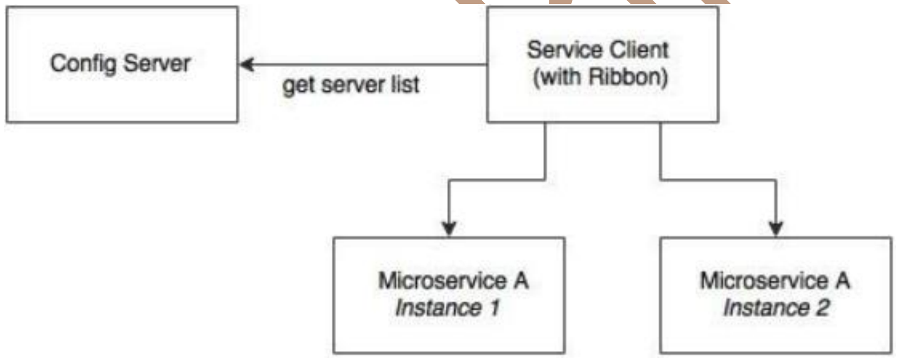
Ribbon For Load Balancing

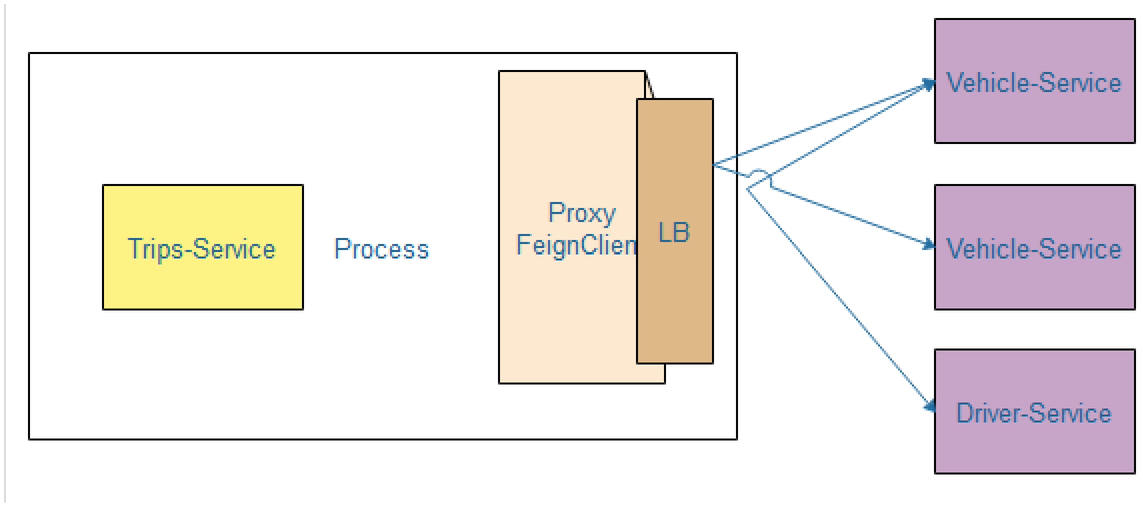
So far, we were always running with a single instance of the microservice.The URL is hard coded both in client as well as in the service-to-service calls.Since there could be more than one service instance in the real world, this is not a recommended approach.

If there are multiple instances, then ideally, we should use a load balancer or a local DNS server to abstract the actual instance locations, and configure an alias name or the load balancer address in the clients.

The load balancer then receives the alias name, and resolves it with one of the available instances. With this approach, we can configure as many instances behind a load balancer. It also helps us to handle server failures transparent to the client.

This is achievable with **Spring Cloud Netflix Ribbon. Ribbon is a client-side load balancer which can do round-robin load balancing across a set of servers**. There could be other load balancing algorithms possible with the Ribbon library. Spring Cloud offers a declarative way to configure and use the Ribbon client.



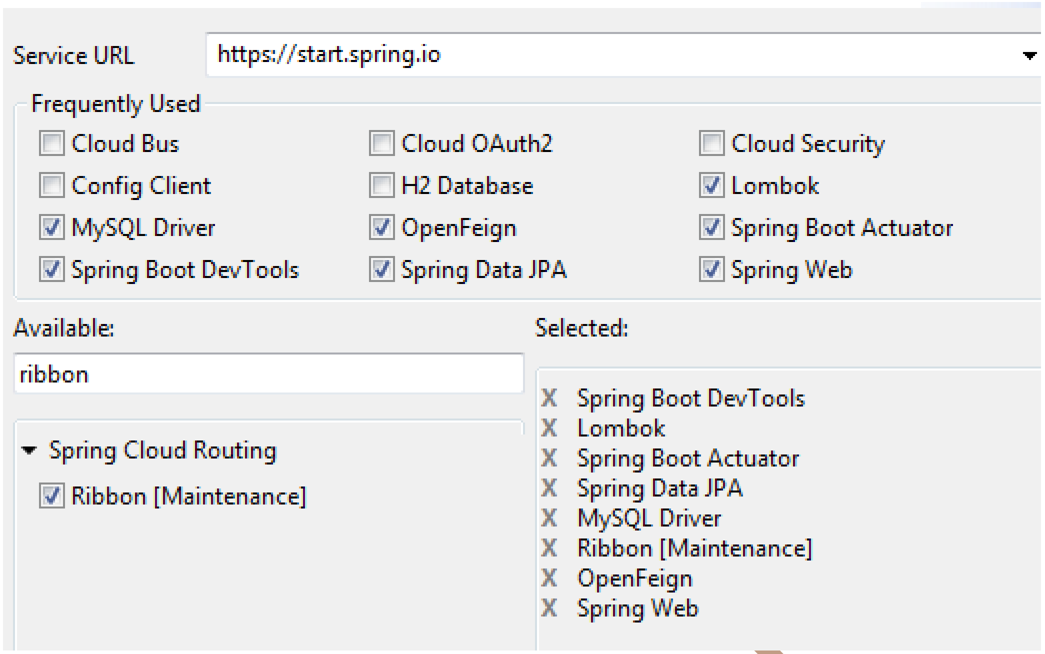


As shown in the preceding diagram, the Ribbon client looks for the Config server to get the list of available microservice instances, and, by default, applies a round-robin load balancing algorithm.

Step-1: In order to use the Ribbon client, we will have to add the following dependency to the pom.xml file:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
```

In case of development from ground up, this can be selected from the Spring Starter libraries, or from **http://start.spring.io/**. Ribbon is available under Cloud Routing:


Service URL https://start.spring.io

Frequently Used

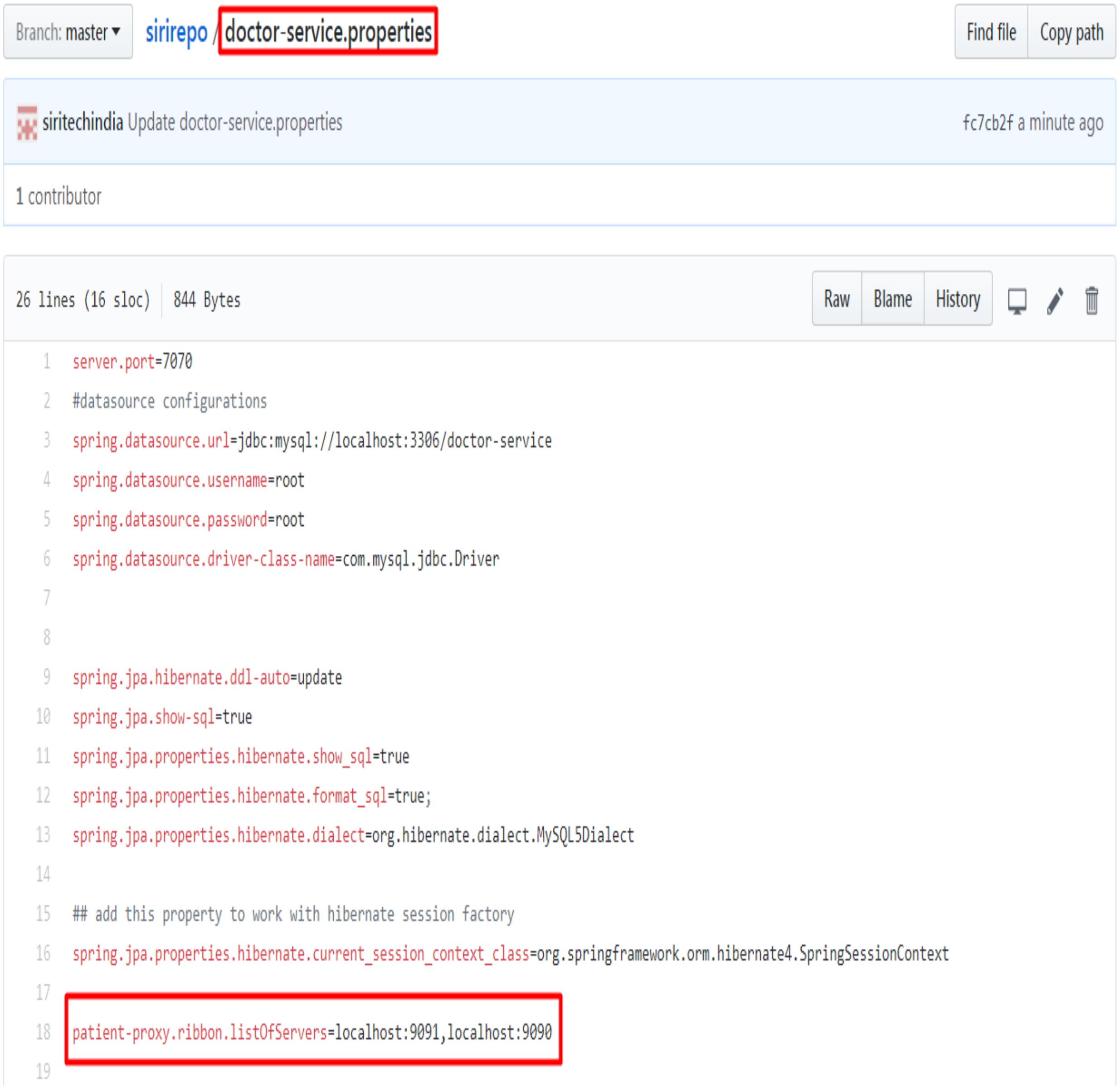
☐ Cloud Bus	☐ Cloud OAuth2	☐ Cloud Security
☐ Config Client	☐ H2 Database	☑ Lombok
☑ MySQL Driver	☑ OpenFeign	☑ Spring Boot Actuator
☑ Spring Boot DevTools	☑ Spring Data JPA	☑ Spring Web

Available: ribbon

Spring Cloud Routing
- ☑ Ribbon [Maintenance]

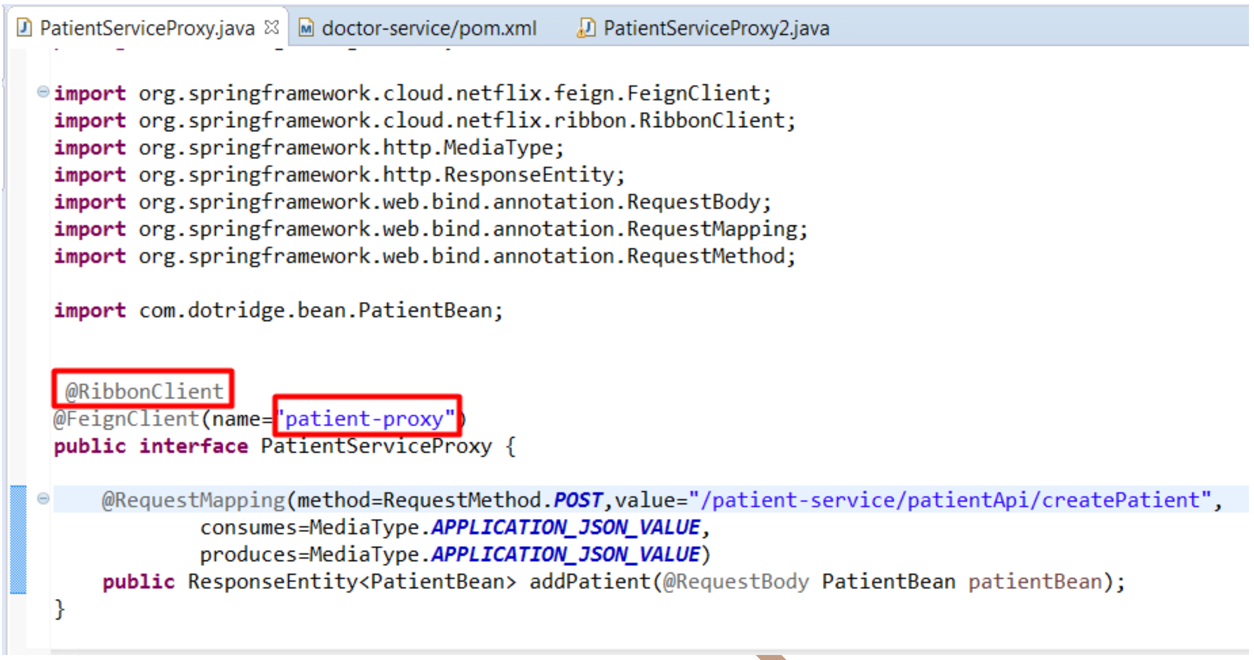
Selected:
- X Spring Boot DevTools
- X Lombok
- X Spring Boot Actuator
- X Spring Data JPA
- X MySQL Driver
- X Ribbon [Maintenance]
- X OpenFeign
- X Spring Web

Step-2: Update the doctor-service microservice configuration file, doctor-service.properties file in git repository(as we are using config server),to include a new property to keep the list of the patient-service MicroServices as:


Branch: master
sirirepo / doctor-service.properties
Find file
Copy path
siritechindia Update doctor-service.properties
fc7cb2f a minute ago
1 contributor
26 lines (16 sloc) 844 Bytes
Raw
Blame
History

```
server.port=7070
#datasource configurations
spring.datasource.url=jdbc:mysql://localhost:3306/doctor-service
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver


spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true;
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

## add this property to work with hibernate session factory
spring.jpa.properties.hibernate.current_session_context_class=org.springframework.orm.hibernate4.SpringSessionContext

patient-proxy.ribbon.listOfServers=localhost:9091,localhost:9090

```

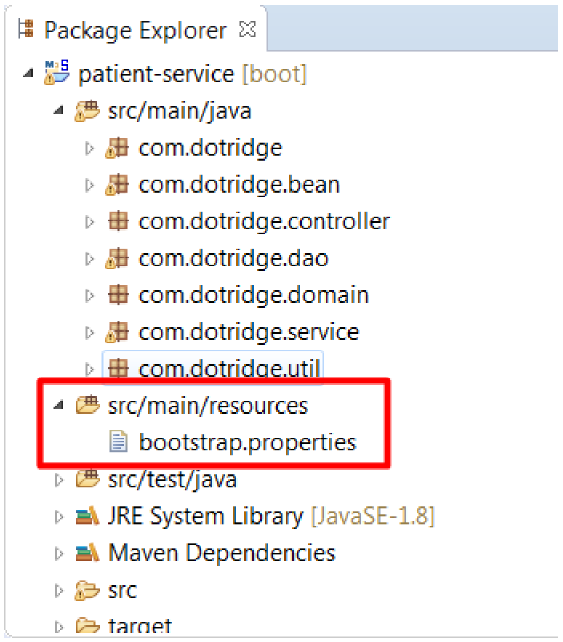
Going back and editing the PatientServiceProxy class to use the Ribbon client as below:


PatientServiceProxy.java | doctor-service/pom.xml | PatientServiceProxy2.java

```java
import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.dotridge.bean.PatientBean;


@RibbonClient
@FeignClient(name="patient-proxy")
public interface PatientServiceProxy {

    @RequestMapping(method=RequestMethod.POST,value="/patient-service/patientApi/createPatient",
            consumes=MediaType.APPLICATION_JSON_VALUE,
            produces=MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<PatientBean> addPatient(@RequestBody PatientBean patientBean);
}
```

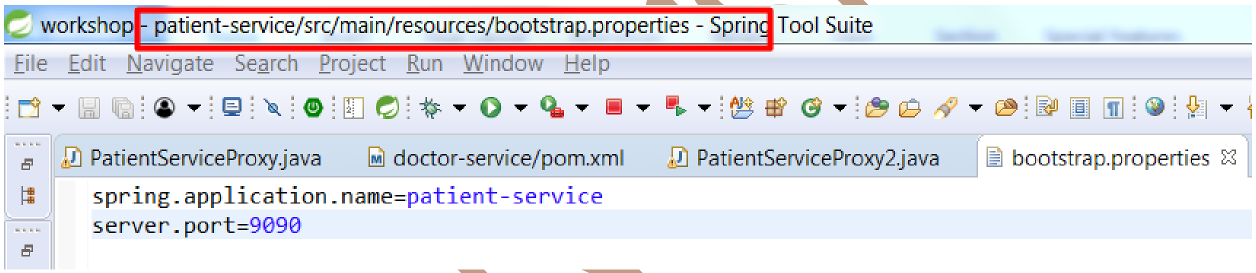
Here if we could observe, we have added **@RibbonClient** annotation to enable client side load balancing used by @FeignClient. In @FeignClient we have taken its name as **"patient-proxy"**. Now Make sure to add the **listOfServers** property with this name I.e.**<name of the feign client>.ribbon.lostOfServers**. Hence In our case it should be like **patient-proxy.ribbon.listOfServers=localhost:9091,localhost:9090**

Step-3: To See the effect of load balancing lets make the two instances of **patient-service** micro service. Patient-service micro service is as:


Package Explorer

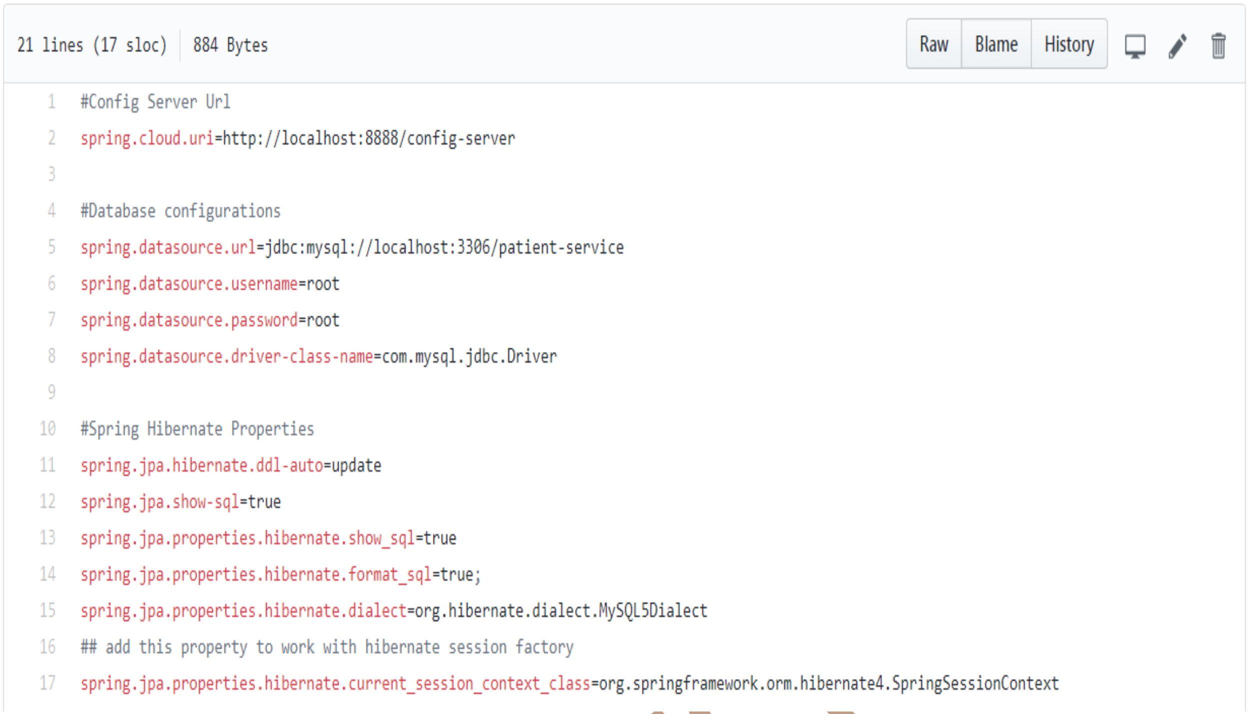
```
patient-service [boot]
  src/main/java
    com.dotridge
    com.dotridge.bean
    com.dotridge.controller
    com.dotridge.dao
    com.dotridge.domain
    com.dotridge.service
    com.dotridge.util
  src/main/resources
    bootstrap.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
```

The bootstrap.properties file of patient-service micro service is as below:


workshop - patient-service/src/main/resources/bootstrap.properties - Spring Tool Suite
File Edit Navigate Search Project Run Window Help
PatientServiceProxy.java
doctor-service/pom.xml
PatientServiceProxy2.java
bootstrap.properties

```
spring.application.name=patient-service
server.port=9090
```

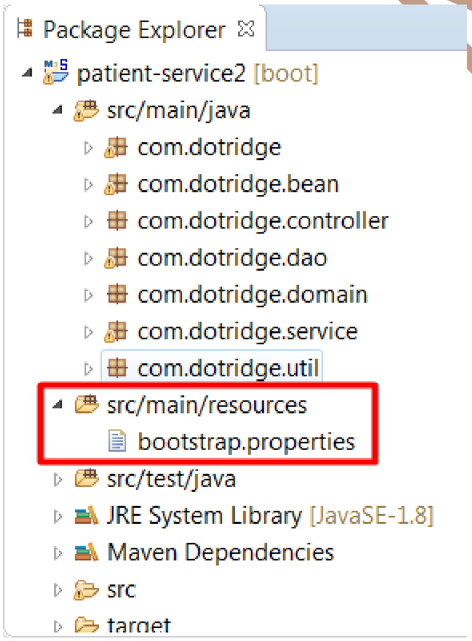
As patient-service connecting to config-server to fetch its configuration properties from git, it should have a properties file with the name **patient-server.properties(which is <spring.application.name>.properties)** in git repository as shown below:



21 lines (17 sloc) | 884 Bytes

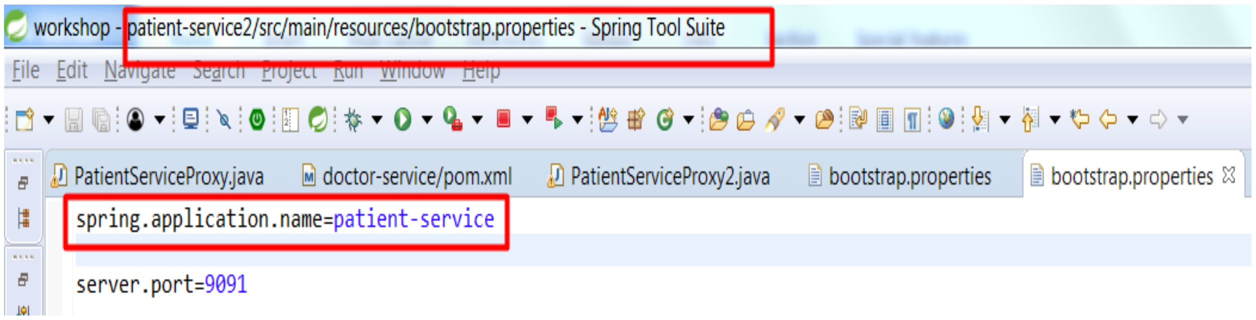
```
#Config Server Url
spring.cloud.uri=http://localhost:8888/config-server

#Database configurations
spring.datasource.url=jdbc:mysql://localhost:3306/patient-service
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

#Spring Hibernate Properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true;
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
## add this property to work with hibernate session factory
spring.jpa.properties.hibernate.current_session_context_class=org.springframework.orm.hibernate4.SpringSessionContext
```

To make another instance of patient-service, copy and paste the same patient-service project in Spring Too Suit, it should be as below:



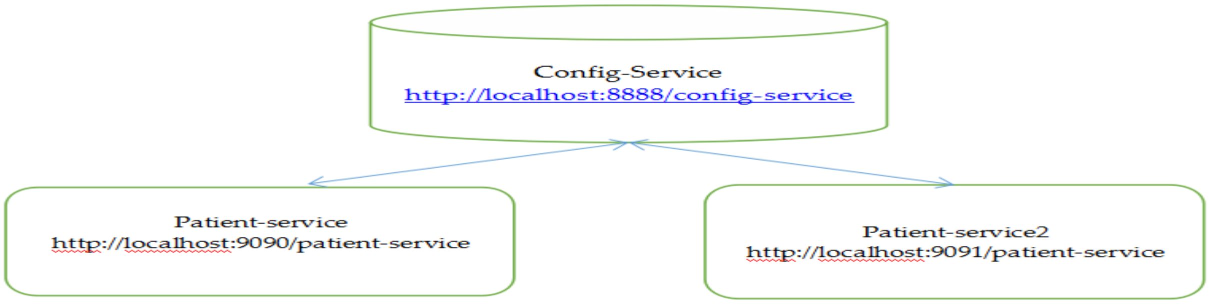
```
Package Explorer
patient-service2 [boot]
  src/main/java
    com.dotridge
    com.dotridge.bean
    com.dotridge.controller
    com.dotridge.dao
    com.dotridge.domain
    com.dotridge.service
    com.dotridge.util
  src/main/resources
    bootstrap.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
  target
```

This Service is completely identical to the patient-service except its port number. It should run on different port than patient-service,except this all the properties of the patient-service should also applies to this including spring.application.name. See the below screen shot.


workshop - patient-service2/src/main/resources/bootstrap.properties - Spring Tool Suite
File Edit Navigate Search Project Run Window Help
PatientServiceProxy.java
doctor-service/pom.xml
PatientServiceProxy2.java
bootstrap.properties
bootstrap.properties

```
spring.application.name=patient-service

server.port=9091
```

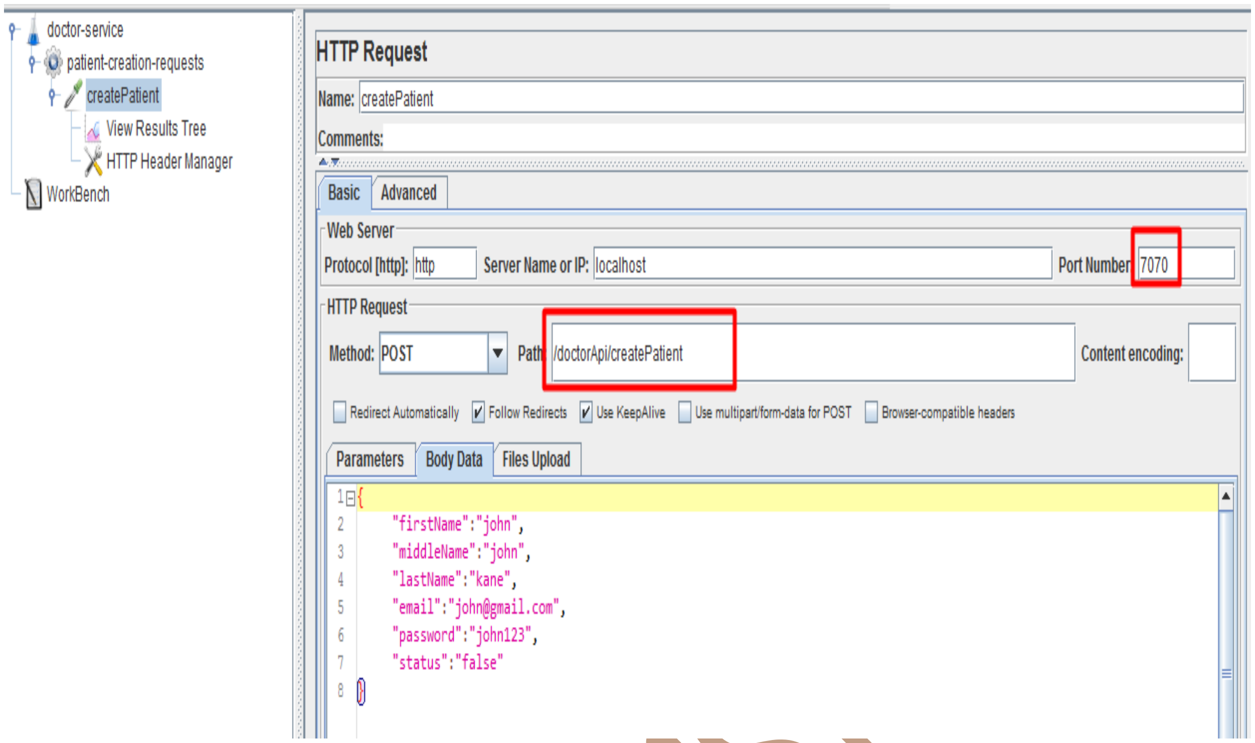
As it is also have the same name I.e. **sping.application.name=patient-service**, this will pull the same configuration parameters as patient-service have from the **config-service I.e.git**. Hence we have now two instance of patient-service one is running on port 9090 and other one is on 9091.


Config-Service
http://localhost:8888/config-service
Patient-service
http://localhost:9090/patient-service
Patient-service2
http://localhost:9091/patient-service

<u>**Step-3:**</u> Run the doctor-service microservice,When the it is bootstrapped,we see the following trace, which says there are two servers enlisted:
DynamicServerListLoadBalancer:{NFLoadBalancer:name=patient-proxy,currentlist of Servers=[localhost:9090, localhost:9091],Load balancer stats=Zone stats: {unknown=[Zone:unknown; Instance count:2; Active connections count: 0; Circuit breaker tripped count: 0; Active connections per server: 0.0;] }

When we are creating patients from doctor-service micro service, we could see the log as below:

<u>**Request:**</u>



HTTP Request

Name: createPatient

Comments:

Basic | Advanced

Web Server

Protocol [http]: http   Server Name or IP: localhost   Port Number: 7070

HTTP Request

Method: POST   Path: /doctorApi/createPatient   Content encoding:

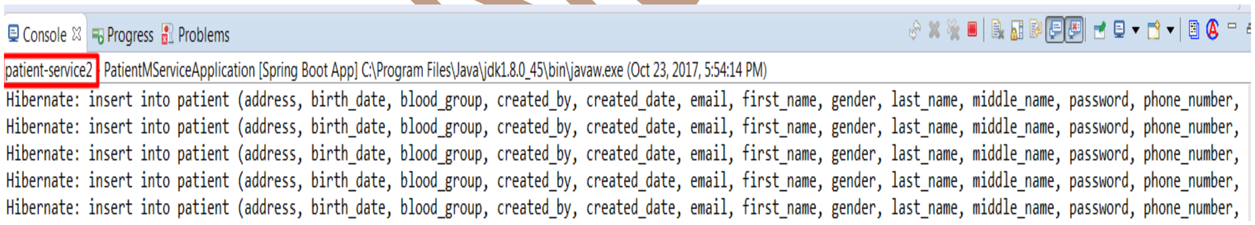
☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data for POST ☐ Browser-compatible headers

Parameters | Body Data | Files Upload

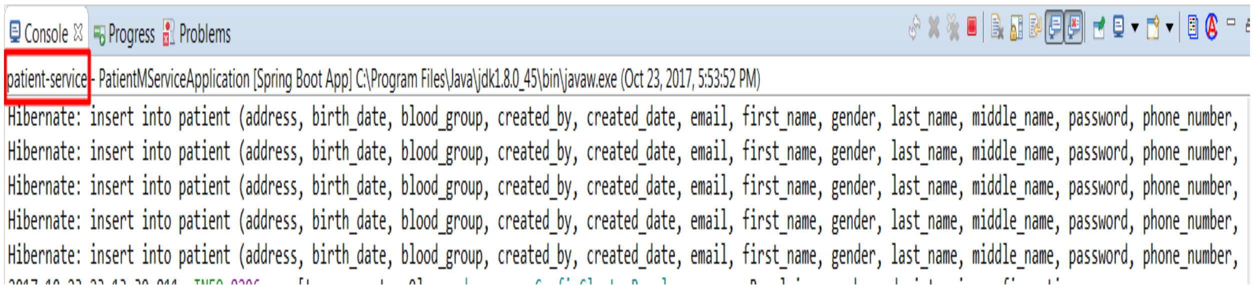
```
{
    "firstName":"john",
    "middleName":"john",
    "lastName":"kane",
    "email":"john@gmail.com",
    "password":"john123",
    "status":"false"
}
```

Here through Jmeter we are simulating this request for 10 users creation I.e. 10 patients records creation

Response:



```
patient-service2 PatientMServiceApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_45\bin\javaw.exe (Oct 23, 2017, 5:54:14 PM)
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
```



```
patient-service - PatientMServiceApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_45\bin\javaw.exe (Oct 23, 2017, 5:53:52 PM)
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
Hibernate: insert into patient (address, birth_date, blood_group, created_by, created_date, email, first_name, gender, last_name, middle_name, password, phone_number,
```

So here 5 users created by patient-service instance and another 5 users were created by patient-service instance2. Here Load balancer uses the Round Robbin algorithm to find the patient-service instance and to hand over the request

Note:

If we get any exception like "feign.RetryableException: Read timed out executing GET" please add the feign client time outs as below

```
feign.client.config.default.connectTimeout: 160000000

feign.client.config.default.readTimeout: 160000000
```