**Microservices with Spring Boot - Part 1 - Getting Started**

*Jan 5, 2018*  
*9 minute read*

This guide will help you learn the basics of microservices and microservices architectures. We will also start looking at a basic implementation of a microservice with Spring Boot.

*We will create a couple of microservices and get them to talk to each other using Eureka Naming Server and Ribbon for Client Side Load Balancing.*

This is a 5 Part Article Series

Microservices with Spring Boot

* Current Part - Part 1 - Getting Started with Microservices Architecture
* Part 2 - [Creating Forex Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-2-forex-microservice)
* Part 3 - [Creating Currency Conversion Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-3-currency-conversion-microservice)
* Part 4 - [Using Ribbon for Load Balancing](http://www.springboottutorial.com/microservices-with-spring-boot-part-4-ribbon-for-load-balancing)
* Part 5 - [Using Eureka Naming Server](http://www.springboottutorial.com/microservices-with-spring-boot-part-5-eureka-naming-server)

In part 1 of this series, lets get introduced to the concept of microservices and understand how to create great microservices with Spring Boot and Spring Cloud.

**You will learn**

* What is a Monolith?
* What is a Microservice?
* What are the Challenges with Microservices?
* How does Spring Boot and Spring Cloud make developing Microservices easy?
* How to implement client side load balancing with Ribbon?
* How to implement a Naming Server (Eureka Naming Server)?
* How to connect the microservices with Naming Server and Ribbon?

**Microservices Overview - A Big Picture**

In this series of articles, we would create two microservices:

* Forex Service - Abbreviated as FS
* Currency Conversion Service - Abbreviated as CCS

*Do not worry if you are not clear about a few things. The idea is to give a big picture before we get our hands dirty and create the microservices step by step*

**Forex Service**

Forex Service (FS) is the Service Provider. It provides currency exchange values for various currency. Let’s assume that it talks to a Forex Exchange and provides the current conversion value between currencies.

An example request and response is shown below:

GET to http://localhost:8000/currency-exchange/from/EUR/to/INR

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

port: 8000,

}

The request above is the currency exchange value for EUR to INR. In the response, conversionMultiple is 75.

*We will talk about port in the response a little later.*

**Currency Conversion Service**

Currency Conversion Service (CCS) can convert a bucket of currencies into another currency. It uses the Forex Service to get current currency exchange values. CCS is the Service Consumer.

An example request and response is shown below:

GET to http://localhost:8100/currency-converter/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

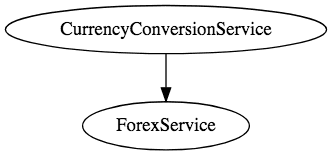
totalCalculatedAmount: 750000,

port: 8000,

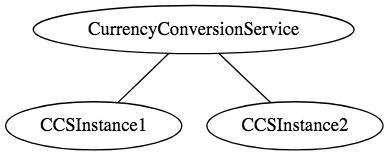
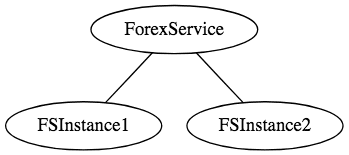
}

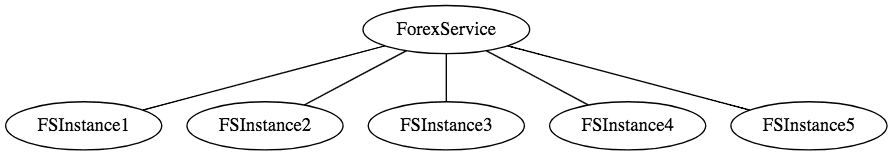
The request above is to find the value of 10000 EUR in INR. The totalCalculatedAmount is 750000 INR.

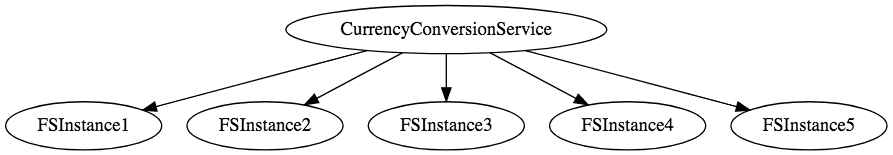
The diagram below shows the communication between CCS and FS.

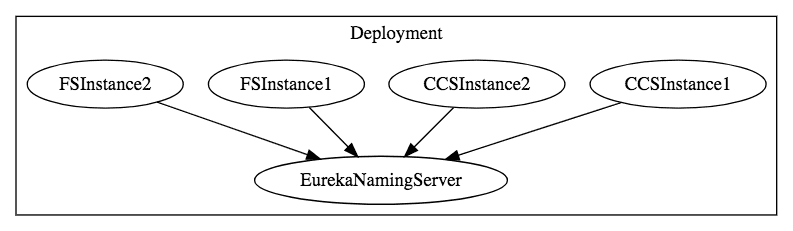


**Eureka Naming Server and Ribbon**

Based on the load, we can have multiple instances of the Currency Conversion Service and the Forex Service running.  

And the number of instances for each service might vary with time. Below picture shows a specific instance where there are 5 instances of the Forex Service. 

What needs to happen in the above situation is load should be uniformly distributed among these 5 instances. 

In this series of articles, we will use Ribbon for Load Balancing and Eureka Naming server for registering all microservices. 

*Do not worry if you are not clear about a few things. The idea is to give a big picture before we get our hands dirty and create the microservices step by step*

**What is a Monolith Application?**

Have you ever worked in a project

* Which is released (taken to production) once every few months
* Which has a wide range of features and functionality
* Which has a team of more than 50 working for it
* Where debugging problems is a big challenge
* Where bringing in new technology and new process is almost impossible

These are typical characteristics of a Monolith applications.

*Monolith applications are typically huge - more 100,000 line of code. In some instances even more than million lines of code.*

Monoliths are characterized by

* Large Application Size
* Long Release Cycles
* Large Teams

Typical Challenges include

* Scalability Challenges
* New Technology Adoption
* New Processes - Agile?
* Difficult to Automation Test
* Difficult ot Adapt to Modern Development Practices
* Adapting to Device Explosion

**Microservices**

Microservice Architectures evolved as a solution to the scalability and innovotation challenges with Monolith architectures.

There are a number of definitions proposed for Microservices

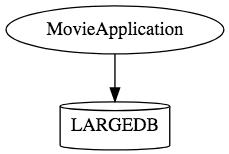
*Small autonomous services that work together - Sam Newman*

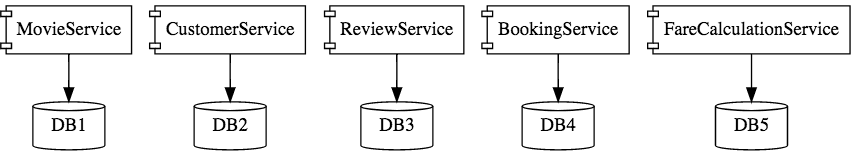
*Developing a single application as a suite of small services each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies - James Lewis and Martin Fowler*

While there is no single accepted definition for microservices, for me, there are a few important characteristics:

* REST - Built around RESTful Resources. Communication can be HTTP or event based.
* Small Well Chosen Deployable Units - Bounded Contexts
* Cloud Enabled - Dynamic Scaling

**How does Microservice Architecture look like?**

This is how a monolith would look like. One application for everything. 

This is how the same application would look like when developed using Microservices Architecture. 

Microservice Architectures involve a number of small, well designed, components interacting with messages. http://www.springboottutorial.com/images/Microservices-Chain-Example.png

**Advantages of Microservices**

Advantages

* New Technology & Process Adaption becomes easier. You can try new technologies with the newer microservices that we create.
* Faster Release Cycles
* Scaling with Cloud

**Challenges with Microservice Architectures**

While developing a number of smaller components might look easy, there are a number of inherent complexities that are associated with microservices architectures.

Lets look at some of the challenges:

* Quick Setup needed : You cannot spend a month setting up each microservice. You should be able to create microservices quickly.
* Automation : Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring etc.
* Visibility : You now have a number of smaller components to deploy and maintain. Maybe 100 or maybe 1000 components. You should be able to monitor and identify problems automatically. You need great visibility around all the components.
* Bounded Context : Deciding the boundaries of a microservice is not an easy task. Bounded Contexts from Domain Driven Design is a good starting point. Your understanding of the domain evolves over a period of time. You need to ensure that the microservice boundaries evolve.
* Configuration Management : You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution
* Dynamic Scale Up and Scale Down : The advantages of microservices will only be realized if your applications can scaled up and down easily in the cloud.
* Pack of Cards : If a microservice at the bottom of the call chain fails, it can have knock on effects on all other microservices. Microservices should be fault tolerant by Design.
* Debugging : When there is a problem that needs investigation, you might need to look into multiple services across different components. Centralized Logging and Dashboards are essential to make it easy to debug problems.
* Consistency : You cannot have a wide range of tools solving the same problem. While it is important to foster innovation, it is also important to have some decentralized governance around the languages, platforms, technology and tools used for implementing/deploying/monitoring microservices.

**Solutions to Challenges with Microservice Architectures**

**Spring Boot**

*Enable building production ready applications quickly*

Provide non-functional features

* embedded servers (easy deployment with containers)
* metrics (monitoring)
* health checks (monitoring)
* externalized configuration

**Spring Cloud**

*Spring Cloud provides solutions to cloud enable your microservices. It leverages and builds on top of some of the Cloud solutions opensourced by Netflix (Netflix OSS).*

**Important Spring Cloud Modules**

Dynamic Scale Up and Down. Using a combination of

* Naming Server (Eureka)
* Ribbon (Client Side Load Balancing)
* Feign (Easier REST Clients)

Visibility and Monitoring with

* Zipkin Distributed Tracing
* Netflix API Gateway

Configuration Management with

* Spring Cloud Config Server

Fault Tolerance with

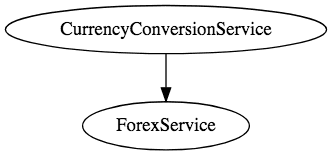
* Hystrix

**Microservice Series of Articles**

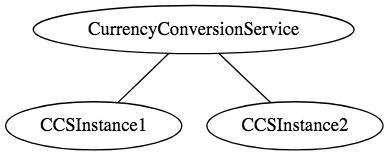
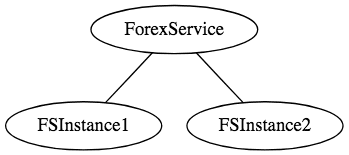
In this series of articles, we would create two microservices:

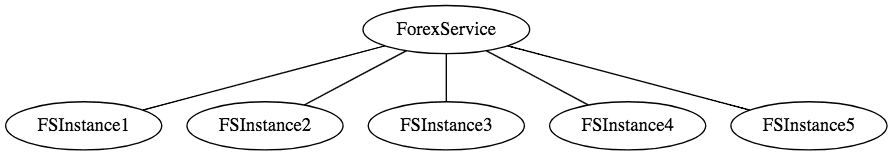
* Forex Service - Abbreviated as FS
* Currency Conversion Service - Abbreviated as CCS

The diagram below shows the communication between CCS and FS. We would establish communication between these two components.



We would want to be able to dynamically scale up and scale down the number of instances of each of these services.

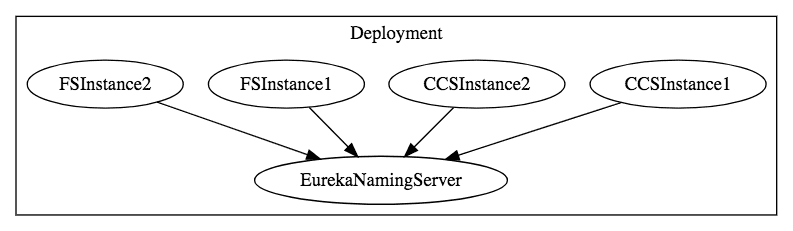
 

And the number of instances for each service might vary with time. Below picture shows a specific instance where there are 5 instances of the Forex Service. 

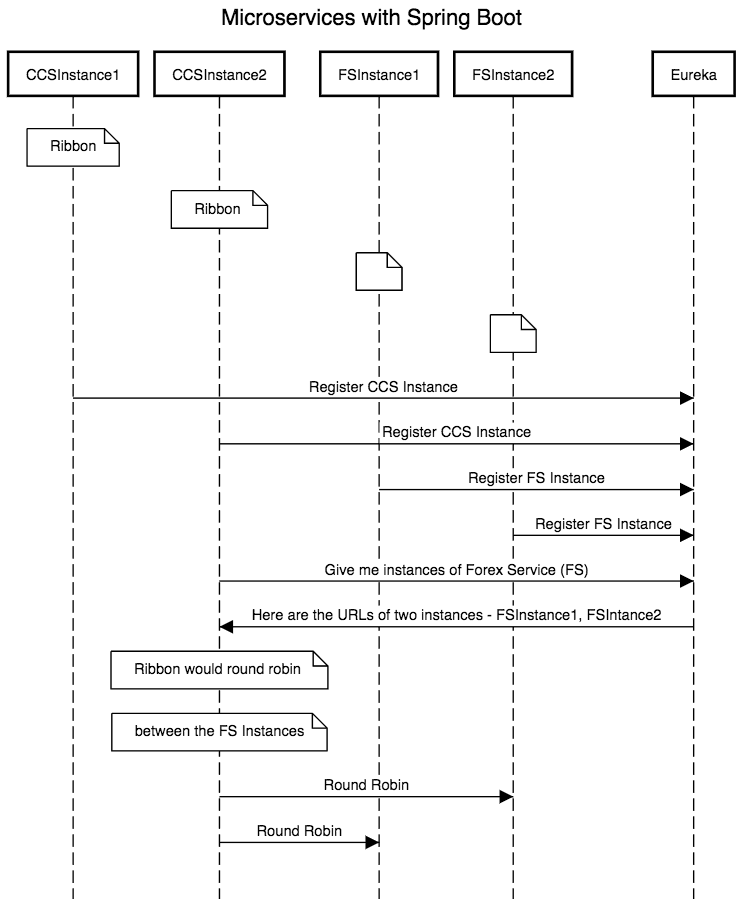
Implementing a solution for dynamic scale up and down needs to answer two questions

* How does the Currency Conversion Service (CCS) know how many instances of Forex Service (FS) are active?
* How does the Currency Conversion Service (CCS) distribute the load between the active instances.

Because we want this to be dynamic, we cannot hardcode the urls of FS in CCS. Thats why we bring in a Naming Server.



All instances of the components (CCS and FS) register with the Eureka Naming Server. When FS needs to call the CCS, it will ask Eureka Naming Server for the active instances. We will use Ribbon to do Client Side Load Balancing between the different instances of FS.

A high level sequence diagram of what would happen when there is a request from CCS to FS is shown below: 

Here are the next series of Articles

* Creating a Forex Microservice - We will create a simple rest service based on Spring Boot Starter Web and Spring Boot Started JPA. We will use Hibernate as JPA implmentation and connect to H2 database.
* Create the CCS - Currency Conversion Service - We will create a simple rest service using feign to invoke the Forex Microservice
* Use Ribbon for Load Balancing
* Implement Eureka Naming Service and connect FS and CCS through Eureka.

**Microservices with Spring Boot - Part 2 - Creating Forex Microservice**

*Jan 6, 2018*  
*7 minute read*

This is part 2 of this series. In this part, we will focus on creating the Forex Microservice.

**You will learn**

* How to create a microservice with Spring Boot?
* How to create a JPA Entity and Resource?
* How to get Spring MVC, Spring Boot, JPA, Hibernate and H2 to work together?

Forex Service (FS) is the Service Provider. It provides currency exchange values for various currency. Let’s assume that it talks to a Forex Exchange and provides the current conversion value between currencies.

An example request and response is shown below:

GET to http://localhost:8000/currency-exchange/from/EUR/to/INR

{

id: 10002,

from: "EUR",

to: "INR",

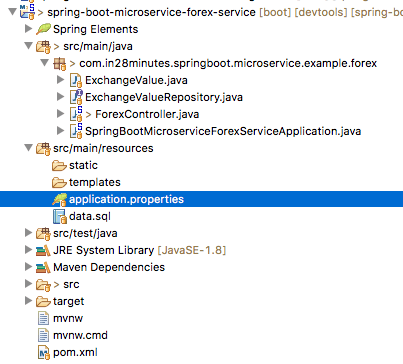
conversionMultiple: 75,

port: 8000,

}

The request above is the currency exchange value for EUR to INR. In the response, conversionMultiple is 75.

**Project Code Structure**

Following screenshot shows the structure of the project we will create. 

A few details:

* SpringBootMicroserviceForexServiceApplication.java - The Spring Boot Application class generated with Spring Initializer. This class acts as the launching point for application.
* pom.xml - Contains all the dependencies needed to build this project. We will use Spring Boot Starter Web and JPA.
* ExchangeValue.java - Exchange Value Entity
* ExchangeValueRepository.java - ExchangeValue JPA Repository. This is created using Spring Data JpaRepository.
* ForexController.java - Spring Rest Controller exposing the forex conversion service.
* data.sql - Initial data for the exchange\_value table. Spring Boot would execute this script after the tables are created from the entities.

**Tools you will need**

* Maven 3.0+ is your build tool
* Your favorite IDE. We use Eclipse.
* JDK 1.8+

**Complete Maven Project With Code Examples**

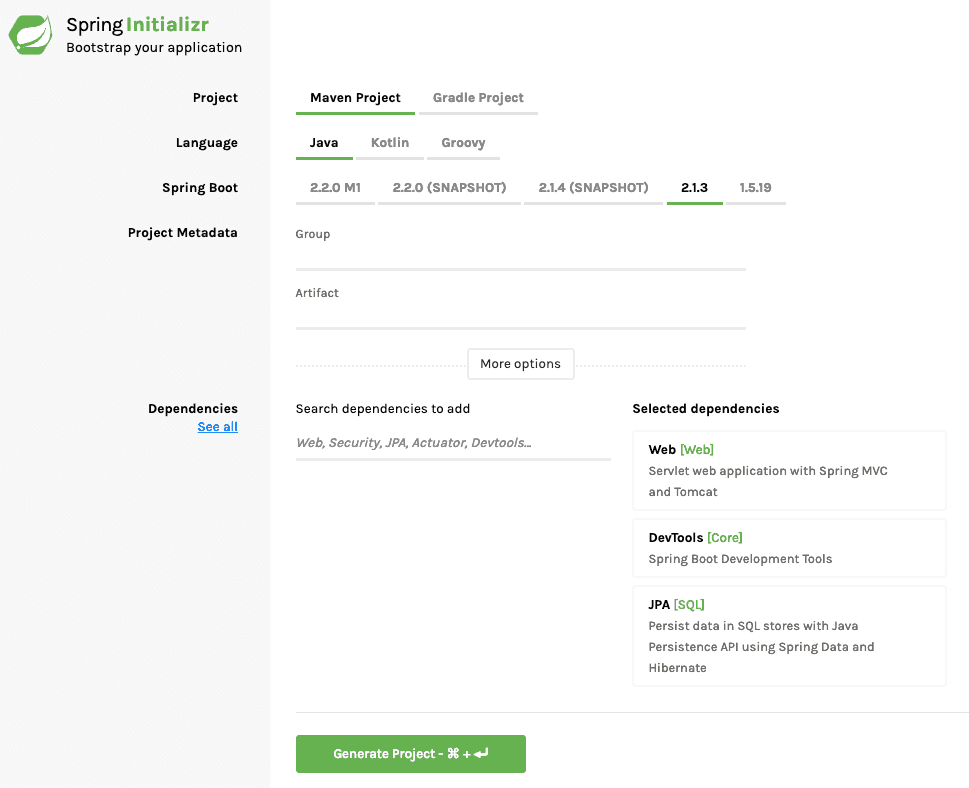
*Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-basic-microservice*

**Bootstrapping with Spring Initializr**

Creating a Microservice with Spring Initializr is a cake walk.

Spring Initializr <http://start.spring.io/> is great tool to bootstrap your Spring Boot projects.

You can create a wide variety of projects using Spring Initializr.



Following steps have to be done for a Web Services project

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot.microservice.example.forex as Group
  + Choose spring-boot-microservice-forex-service as Artifact
  + Choose following dependencies
    - Web
    - DevTools
    - Starter JPA
    - H2
* Click Generate Project.
* Import the project into Eclipse. File -> Import -> Existing Maven Project.

**Creating Exchange Value Entity**

@Entity

public class ExchangeValue {

@Id

private Long id;

@Column(name="currency\_from")

private String from;

@Column(name="currency\_to")

private String to;

private BigDecimal conversionMultiple;

private int port;

public ExchangeValue() {

}

public ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

}

public Long getId() {

return id;

}

public String getFrom() {

return from;

}

public String getTo() {

return to;

}

public BigDecimal getConversionMultiple() {

return conversionMultiple;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

Important things to note:

* @Entity: Specifies that the class is an entity. This annotation is applied to the entity class.
* @Id: Specifies the primary key of an entity.

**Creating Exchange Value JPA Repository**

/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ExchangeValueRepository.java

package com.in28minutes.springboot.microservice.example.forex;

import org.springframework.data.jpa.repository.JpaRepository;

public interface ExchangeValueRepository extends

JpaRepository<ExchangeValue, Long>{

ExchangeValue findByFromAndTo(String from, String to);

}

Notes

* public interface ExchangeValueRepository extends JpaRepository<ExchangeValue, Long> - We are extending JpaRepository using two generics - ExchangeValue & Long. ExchangeValue is the entity that is being managed and the primary key of ExchangeValue is Long.
* ExchangeValue findByFromAndTo(String from, String to); - We would want to query the conversion value from one currency to another. We are defining a query method for it.

**Create the Resource - ForexController**

/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ForexController.java

@RestController

public class ForexController {

@Autowired

private Environment environment;

@Autowired

private ExchangeValueRepository repository;

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public ExchangeValue retrieveExchangeValue

(@PathVariable String from, @PathVariable String to){

ExchangeValue exchangeValue =

repository.findByFromAndTo(from, to);

exchangeValue.setPort(

Integer.parseInt(environment.getProperty("local.server.port")));

return exchangeValue;

}

}

Notes

* @RestController public class ForexController { - Create a Controller to expose a Rest Service
* @Autowired private Environment environment - We would want to return the server port back. This will help us identify which instance service is giving the response back.
* @Autowired private ExchangeValueRepository repository - Autowire the repository.
* ExchangeValue exchangeValue = repository.findByFromAndTo(from, to) - Get the exchange value from the database.
* exchangeValue.setPort(Integer.parseInt(environment.getProperty("local.server.port")))- Get the port from environment and set it into the response bean.

**Configure Application Name and a few other configuration**

/spring-boot-microservice-forex-service/src/main/resources/application.properties

spring.application.name=forex-service

server.port=8000

spring.jpa.show-sql=true

spring.h2.console.enabled=true

We are assigning a port of 8000 for this application and enabling debug logging.

**Insert some test data into data.sql**

Let’s insert some test data by creating a file called data.sql. Spring Boot Auto Configuration ensures that this data is loaded up when application starts up.

/spring-boot-microservice-forex-service/src/main/resources/data.sql

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10001,'USD','INR',65,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10002,'EUR','INR',75,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10003,'AUD','INR',25,0);

**Test Forex Microservice**

GET to http://localhost:8000/currency-exchange/from/EUR/to/INR

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

port: 8000,

}

**Complete Code Example**

**/spring-boot-microservice-forex-service/pom.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.in28minutes.springboot.microservice.example.forex</groupId>

<artifactId>spring-boot-microservice-forex-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-microservice-forex-service</name>

<description>Microservices with Spring Boot and Spring Cloud - Forex Service</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath/> *<!-- lookup parent from repository -->*

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Finchley.M8</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ExchangeValue.java**

package com.in28minutes.springboot.microservice.example.forex;

import java.math.BigDecimal;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class ExchangeValue {

@Id

private Long id;

@Column(name="currency\_from")

private String from;

@Column(name="currency\_to")

private String to;

private BigDecimal conversionMultiple;

private int port;

public ExchangeValue() {

}

public ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

}

public Long getId() {

return id;

}

public String getFrom() {

return from;

}

public String getTo() {

return to;

}

public BigDecimal getConversionMultiple() {

return conversionMultiple;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ExchangeValueRepository.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.springframework.data.jpa.repository.JpaRepository;

public interface ExchangeValueRepository extends

JpaRepository<ExchangeValue, Long>{

ExchangeValue findByFromAndTo(String from, String to);

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ForexController.java**

package com.in28minutes.springboot.microservice.example.forex;

import java.math.BigDecimal;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.env.Environment;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class ForexController {

@Autowired

private Environment environment;

@Autowired

private ExchangeValueRepository repository;

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public ExchangeValue retrieveExchangeValue

(@PathVariable String from, @PathVariable String to){

ExchangeValue exchangeValue =

repository.findByFromAndTo(from, to);

exchangeValue.setPort(

Integer.parseInt(environment.getProperty("local.server.port")));

return exchangeValue;

}

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/SpringBootMicroserviceForexServiceApplication.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootMicroserviceForexServiceApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceForexServiceApplication.class, args);

}

}

**/spring-boot-microservice-forex-service/src/main/resources/application.properties**

spring.application.name=forex-service

server.port=8000

spring.jpa.show-sql=true

spring.h2.console.enabled=true

**/spring-boot-microservice-forex-service/src/main/resources/data.sql**

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10001,'USD','INR',65,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10002,'EUR','INR',75,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10003,'AUD','INR',25,0);

**/spring-boot-microservice-forex-service/src/test/java/com/in28minutes/springboot/microservice/example/forex/SpringBootMicroserviceForexServiceApplicationTests.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringBootMicroserviceForexServiceApplicationTests {

@Test

public void contextLoads() {

}

}

**Microservices with Spring Boot - Part 3 - Creating Currency Conversion Microservice**

*Jan 7, 2018*  
*8 minute read*

Let’s learn the basics of microservices and microservices architectures. We will also start looking at a basic implementation of a microservice with Spring Boot. We will create a couple of microservices and get them to talk to each other using Eureka Naming Server and Ribbon for Client Side Load Balancing.

Here is the Microservice Series Outline: Microservices with Spring Boot

* Part 1 - [Getting Started with Microservices Architecture](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-1-getting-started)
* Part 2 - [Creating Forex Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-2-forex-microservice)
* Current Part - Part 3 - Creating Currency Conversion Microservice
* Part 4 - [Using Ribbon for Load Balancing](http://www.springboottutorial.com/microservices-with-spring-boot-part-4-ribbon-for-load-balancing)
* Part 5 - [Using Eureka Naming Server](http://www.springboottutorial.com/microservices-with-spring-boot-part-5-eureka-naming-server)

This is part 3 of this series. In this part, we will focus on creating the Currency Conversion Microservice.

**You will learn**

* How to create a microservice with Spring Boot?
* How to use RestTemplate to execute a REST Service?
* How to use Feign to execute a REST Service?
* What are the advantages of Feign over RestTemplate?

**Resources Overview**

Currency Conversion Service (CCS) can convert a bucket of currencies into another currency. It uses the Forex Service to get current currency exchange values. CCS is the Service Consumer.

An example request and response is shown below:

GET to http://localhost:8100/currency-converter/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

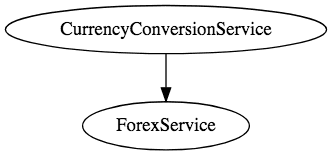
totalCalculatedAmount: 750000,

port: 8000,

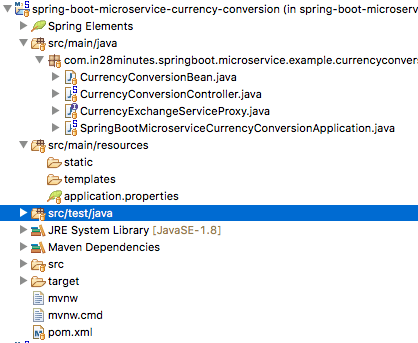
}

The request above is to find the value of 10000 EUR in INR. The totalCalculatedAmount is 750000 INR.

The diagram below shows the communication between CCS and FS.



**Project Code Structure**

Following screenshot shows the structure of the project we will create. 

A few details:

* SpringBootMicroserviceCurrencyConversionApplication.java - The Spring Boot Application class generated with Spring Initializer. This class acts as the launching point for application.
* pom.xml - Contains all the dependencies needed to build this project. We will use Spring Boot Starter Web.
* CurrencyConversionBean.java - Bean to hold the response that we want to send out.
* CurrencyExchangeServiceProxy.java - This will be the Feign Proxy to call the Forex Service.
* CurrencyConversionController.java - Spring Rest Controller exposing the currency conversion service. This will use the CurrencyExchangeServiceProxy to call the Forex Service.

**Tools you will need**

* Maven 3.0+ is your build tool
* Your favorite IDE. We use Eclipse.
* JDK 1.8+

**Complete Maven Project With Code Examples**

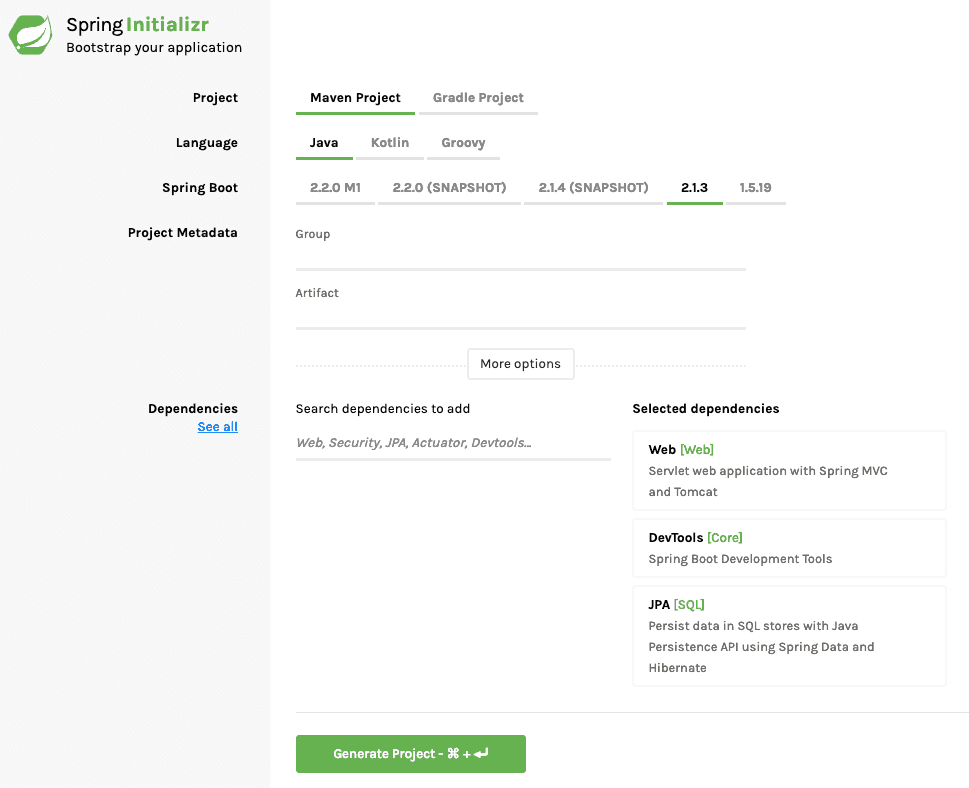
*Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-basic-microservice*

**Bootstrapping with Spring Initializr**

Creating a Microservice with Spring Initializr is a cake walk.

Spring Initializr <http://start.spring.io/> is great tool to bootstrap your Spring Boot projects.

You can create a wide variety of projects using Spring Initializr.



Following steps have to be done for a Web Services project

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot.microservice.example.currencyconversion as Group
  + Choose spring-boot-microservice-currency-conversion as Artifact
  + Choose following dependencies
    - Web
    - DevTools
    - Feign
* Click Generate Project.
* Import the project into Eclipse. File -> Import -> Existing Maven Project.

*Do not forget to choose Feign in the dependencies*

**Creating CurrencyConversionBean**

This is a simple bean for creating the response.

public class CurrencyConversionBean {

private Long id;

private String from;

private String to;

private BigDecimal conversionMultiple;

private BigDecimal quantity;

private BigDecimal totalCalculatedAmount;

private int port;

public CurrencyConversionBean() {

}

public CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal quantity,

BigDecimal totalCalculatedAmount, int port) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

this.quantity = quantity;

this.totalCalculatedAmount = totalCalculatedAmount;

this.port = port;

}

**Implement REST Client with RestTemplate**

The code below shows the implementation of REST Client to call the forex service and process the response. As you can see there is a lot of code that needs to be written for making a simple service call.

@RestController

public class CurrencyConversionController {

private Logger logger = LoggerFactory.getLogger(this.getClass());

@GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

Map<String, String> uriVariables = new HashMap<>();

uriVariables.put("from", from);

uriVariables.put("to", to);

ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate().getForEntity(

"http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversionBean.class,

uriVariables);

CurrencyConversionBean response = responseEntity.getBody();

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

**Configure application name and port**

/spring-boot-microservice-currency-conversion-service/src/main/resources/application.properties

spring.application.name=currency-conversion-service

server.port=8100

We are assigning an application name as well as a default port of 8100.

**Testing the Microservice**

Start the Spring Boot Application by launching SpringBootMicroserviceCurrencyConversionApplication.java

GET to http://localhost:8100/currency-converter/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

port: 8000,

}

**Creating a Feign Proxy**

Feign provide a better alternative to RestTemplate to call REST API.

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyExchangeServiceProxy.java

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.cloud.netflix.ribbon.RibbonClient;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="forex-service" url="localhost:8000")

public interface CurrencyExchangeServiceProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public CurrencyConversionBean retrieveExchangeValue

(@PathVariable("from") String from, @PathVariable("to") String to);

}

We first define a simple proxy.

* @FeignClient(name="forex-service" url="localhost:8100") - Declares that this is a Feign Client and the url at which forex-service is present is localhost:8100
* @GetMapping("/currency-exchange/from/{from}/to/{to}") - URI of the service we would want to consume

**Using Feign Proxy from the Microservice Controller**

Making the call using the proxy is very simple. You can see it in action in the code below. All that we had to do was to autowire the proxy and use to call the method.

@Autowired

private CurrencyExchangeServiceProxy proxy;

@GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to);

logger.info("{}", response);

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

**Enable Feign Clients**

Before we are able to use Feign, we need to enable it by using @EnableFeignClientsannotation on the appropriate package where the client proxies are defined.

@SpringBootApplication

@EnableFeignClients("com.in28minutes.springboot.microservice.example.currencyconversion")

@EnableDiscoveryClient

public class SpringBootMicroserviceCurrencyConversionApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceCurrencyConversionApplication.class, args);

}

}

**Testing the Microservice using Feign**

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

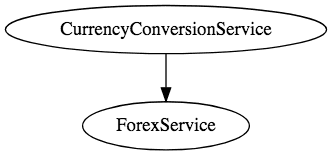
totalCalculatedAmount: 750000,

port: 8000,

}

**Summary**

We have now created two microservices and established communication between them.



However, we are hardcoding the url for FS in CCS. That means when new instances of FS are launched up we have no way to distribute load between them.

In the next part, we will enable client side load distribution using Ribbon.

**Complete Code Example**

**/spring-boot-microservice-currency-conversion-service/pom.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.in28minutes.springboot.microservice.example.currency-conversion</groupId>

<artifactId>spring-boot-microservice-currency-conversion</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-microservice-currency-conversion</name>

<description>Microservices with Spring Boot and Spring Cloud - Currency Conversion Service</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath /> *<!-- lookup parent from repository -->*

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Finchley.M8</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyConversionBean.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import java.math.BigDecimal;

public class CurrencyConversionBean {

private Long id;

private String from;

private String to;

private BigDecimal conversionMultiple;

private BigDecimal quantity;

private BigDecimal totalCalculatedAmount;

private int port;

public CurrencyConversionBean() {

}

public CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal quantity,

BigDecimal totalCalculatedAmount, int port) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

this.quantity = quantity;

this.totalCalculatedAmount = totalCalculatedAmount;

this.port = port;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFrom() {

return from;

}

public void setFrom(String from) {

this.from = from;

}

public String getTo() {

return to;

}

public void setTo(String to) {

this.to = to;

}

public BigDecimal getConversionMultiple() {

return conversionMultiple;

}

public void setConversionMultiple(BigDecimal conversionMultiple) {

this.conversionMultiple = conversionMultiple;

}

public BigDecimal getQuantity() {

return quantity;

}

public void setQuantity(BigDecimal quantity) {

this.quantity = quantity;

}

public BigDecimal getTotalCalculatedAmount() {

return totalCalculatedAmount;

}

public void setTotalCalculatedAmount(BigDecimal totalCalculatedAmount) {

this.totalCalculatedAmount = totalCalculatedAmount;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyConversionController.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import java.math.BigDecimal;

import java.util.HashMap;

import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.client.RestTemplate;

@RestController

public class CurrencyConversionController {

private Logger logger = LoggerFactory.getLogger(this.getClass());

@Autowired

private CurrencyExchangeServiceProxy proxy;

@GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

Map<String, String> uriVariables = new HashMap<>();

uriVariables.put("from", from);

uriVariables.put("to", to);

ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate().getForEntity(

"http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversionBean.class,

uriVariables);

CurrencyConversionBean response = responseEntity.getBody();

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

@GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to);

logger.info("{}", response);

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyExchangeServiceProxy.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.cloud.netflix.ribbon.RibbonClient;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="forex-service" url="localhost:8000")

public interface CurrencyExchangeServiceProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public CurrencyConversionBean retrieveExchangeValue

(@PathVariable("from") String from, @PathVariable("to") String to);

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/SpringBootMicroserviceCurrencyConversionApplication.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.feign.EnableFeignClients;

@SpringBootApplication

@EnableFeignClients("com.in28minutes.springboot.microservice.example.currencyconversion")

public class SpringBootMicroserviceCurrencyConversionApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceCurrencyConversionApplication.class, args);

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/resources/application.properties**

spring.application.name=currency-conversion-service

server.port=8100

**/spring-boot-microservice-currency-conversion-service/src/test/java/com/in28minutes/springboot/microservice/example/currencyconversion/SpringBootMicroserviceCurrencyConversionApplicationTests.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringBootMicroserviceCurrencyConversionApplicationTests {

@Test

public void contextLoads() {

}

}

**Microservices with Spring Boot - Part 4 - Using Ribbon for Load Balancing**

*Jan 8, 2018*  
*3 minute read*

Let’s learn the basics of microservices and microservices architectures. We will also start looking at a basic implementation of a microservice with Spring Boot. We will create a couple of microservices and get them to talk to each other using Eureka Naming Server and Ribbon for Client Side Load Balancing.

Here is the Microservice Series Outline: Microservices with Spring Boot

* Part 1 - [Getting Started with Microservices Architecture](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-1-getting-started)
* Part 2 - [Creating Forex Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-2-forex-microservice)
* Part 3 - [Creating Currency Conversion Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-3-currency-conversion-microservice)
* Current Part - Part 4 - Using Ribbon for Load Balancing
* Part 5 - [Using Eureka Naming Server](http://www.springboottutorial.com/microservices-with-spring-boot-part-5-eureka-naming-server)

This is part 4 of this series. In this part, we will focus on using Ribbon for Load Balancing.

**You will learn**

* What is the need for Load Balancing?
* What is Ribbon?
* How do you add Ribbon to your Spring Boot Project?
* How do you enable and configure Ribbon to do Load Balancing?

**Microservices Overview**

In the previous two parts, we created the microservices and established communication between them.

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

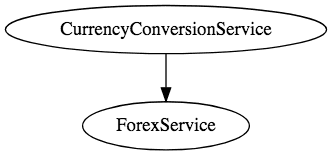
port: 8000,

}

*When we execute the above service, you would see that a request is also sent over to the forex-service.*

Thats cool!

We have now created two microservices and established communication between them.



However, we are hardcoding the url for FS in CCS component CurrencyExchangeServiceProxy.

@FeignClient(name="forex-service" url="localhost:8000")

public interface CurrencyExchangeServiceProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public CurrencyConversionBean retrieveExchangeValue

(@PathVariable("from") String from, @PathVariable("to") String to);

}

That means when new instances of Forex Service are launched up, we have no way to distributing load to them.

In this part, let’s now enable client side load distribution using Ribbon.

**Tools you will need**

* Maven 3.0+ is your build tool
* Your favorite IDE. We use Eclipse.
* JDK 1.8+

**Complete Maven Project With Code Examples**

*Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-basic-microservice*

**Enabling Ribbon**

Add Ribbon Dependency to pom.xml

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

Enable RibbonClient in CurrencyExchangeServiceProxy

@FeignClient(name="forex-service")

@RibbonClient(name="forex-service")

public interface CurrencyExchangeServiceProxy {

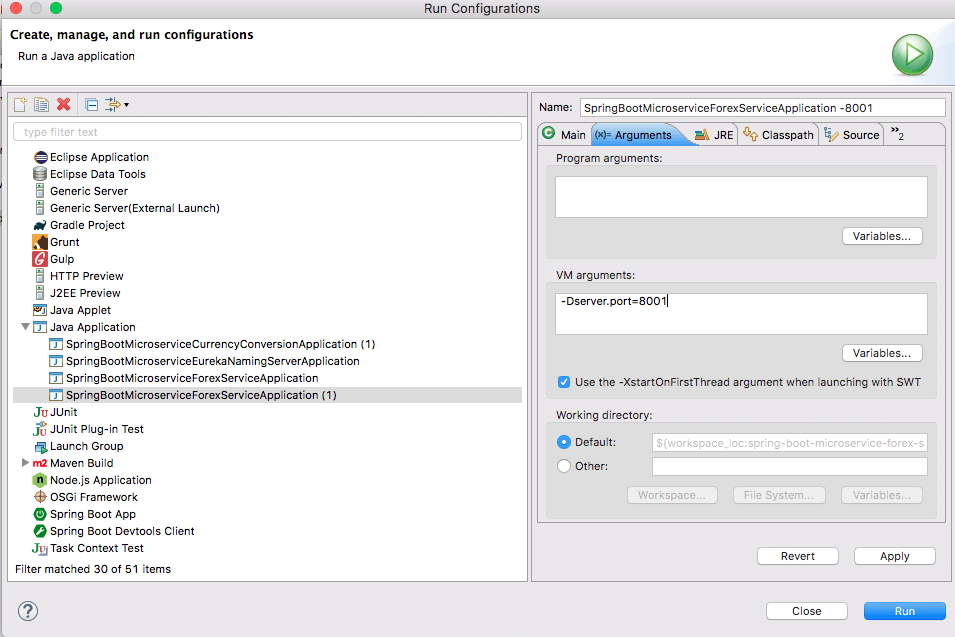
Configure the instances in application.properties

forex-service.ribbon.listOfServers=localhost:8000,localhost:8001

**Launch up Forex Service on 8001**

In the above step, we configured ribbon to distribute load to instances. However, we do not have any instance of Forex Service running on 8001.

We can launch it up by configuring a launch configuration as shown in the figure below.



**Ribbon in Action**

Currently we have the following service up and running

* Currency Conversion Micro Service (CCS) on 8100
* Two instances of Forex MicroService on 8000 and 8001

Now you would see that the requests to CCS would get distributed between the two instances of Forex Microservice by Ribbon

**Request 1**

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

port: 8000,

}

**Request 2**

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

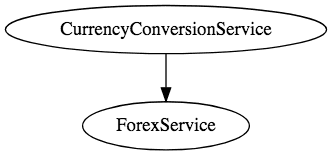
port: 8001,

}

You can see that the port numbers in the two responses are different.

**Summary**

We have now created two microservices and established communication between them.



We are using Ribbon to distribute load between the two instances of Forex Service.

However, we are hardcoding the urls of both instances of FS in CCS. That means every time there is a new instance of FS, we would need to change the configuration of CCS. Thats not cool.

In the next part, we will use Eureka Naming Server to fix this problem.

**Microservices with Spring Boot - Part 5 - Using Eureka Naming Server**

*Jan 9, 2018*  
*10 minute read*

Let’s learn the basics of microservices and microservices architectures. We will also start looking at a basic implementation of a microservice with Spring Boot. We will create a couple of microservices and get them to talk to each other using Eureka Naming Server and Ribbon for Client Side Load Balancing.

Here is the Microservice Series Outline: Microservices with Spring Boot

* Part 1 - [Getting Started with Microservices Architecture](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-1-getting-started)
* Part 2 - [Creating Forex Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-2-forex-microservice)
* Part 3 - [Creating Currency Conversion Microservice](http://www.springboottutorial.com/creating-microservices-with-spring-boot-part-3-currency-conversion-microservice)
* Part 4 - [Using Ribbon for Load Balancing](http://www.springboottutorial.com/microservices-with-spring-boot-part-4-ribbon-for-load-balancing)
* Current Part - Part 5 - Using Eureka Naming Server

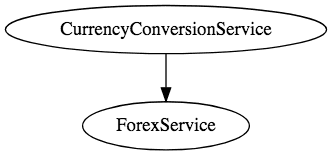
This is part 5 of this series. In this part, we will focus on enabling Eureka Naming Server and have the microservices communicate with it.

**You will learn**

* What is the need for Naming Server?
* What is Eureka?
* How does Naming Server enable location transparancy between microservices?

**Microservices Overview**

In Parts 2 & 3, we created two microservices and established communication between them.



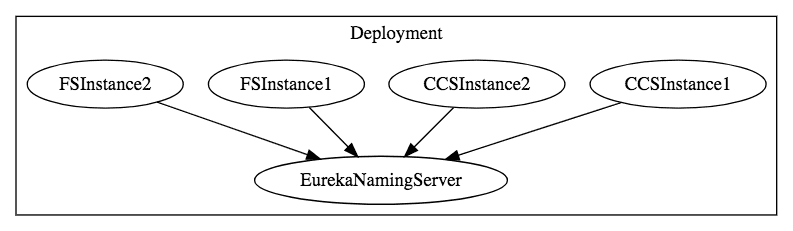
In Part 4, we used Ribbon to distribute load between the two instances of Forex Service.

However, we are hardcoding the urls of both instances of Forex Service in CCS.

forex-service.ribbon.listOfServers=localhost:8000,localhost:8001

That means every time there is a new instance of FS, we would need to change the configuration of CCS. Thats not cool.

In this part, we will use Eureka Naming Server to fix this problem.



**Tools you will need**

* Maven 3.0+ is your build tool
* Your favorite IDE. We use Eclipse.
* JDK 1.8+

**Complete Maven Project With Code Examples**

*Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-basic-microservice*

**Bootstrapping Eureka Naming Server with Spring Initializr**

Creating Eureka Naming Server with Spring Initializr is a cake walk.

Spring Initializr <http://start.spring.io/> is great tool to bootstrap your Spring Boot projects.

You can create a wide variety of projects using Spring Initializr.

Following steps have to be done for a Web Services project

* Launch Spring Initializr and choose the following
  + Choose com.in28minutes.springboot.microservice.eureka.naming.serveras Group
  + Choose spring-boot-microservice-eureka-naming-server as Artifact
  + Choose following dependencies
    - Eureka
    - DevTools
* Click Generate Project.
* Import the project into Eclipse. File -> Import -> Existing Maven Project.

*Do not forget to choose Eureka in the dependencies*

**Enabling Eureka**

EnableEurekaServer in SpringBootMicroserviceEurekaNamingServerApplication.

@SpringBootApplication

@EnableEurekaServer

public class SpringBootMicroserviceEurekaNamingServerApplication {

Configure the application name and port for the Eureka Server

/spring-boot-microservice-eureka-naming-server/src/main/resources/application.properties

spring.application.name=netflix-eureka-naming-server

server.port=8761

eureka.client.register-with-eureka=false

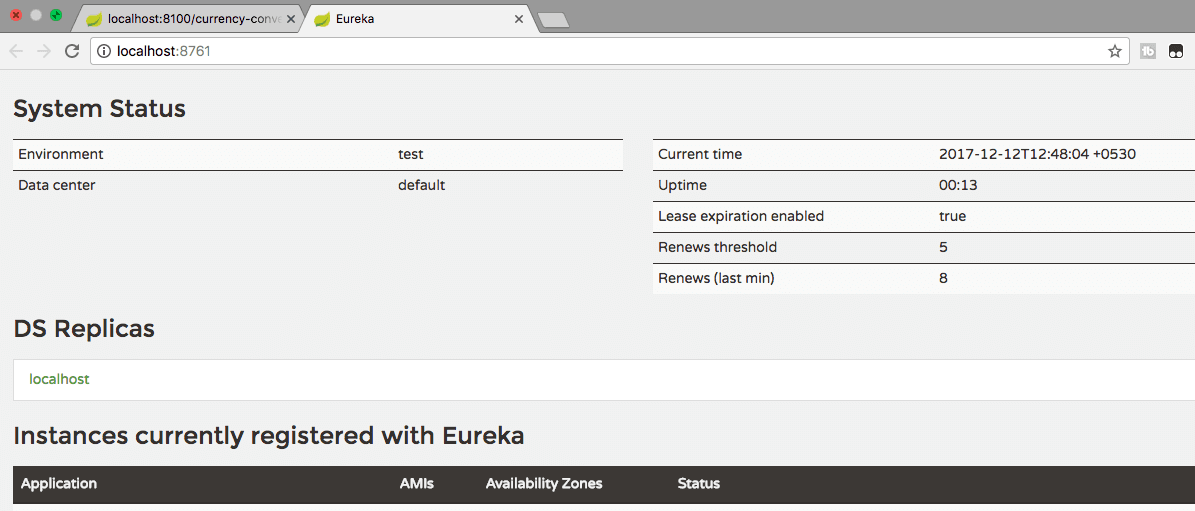
eureka.client.fetch-registry=false

**Launching Eureka Naming Server**

Launch SpringBootMicroserviceEurekaNamingServerApplication as a Java application.

You can launch up Eureka at http://localhost:8761

You would see that there are no instances yet connect to Eureka.



**Connect FS and CCS Microservices with Eureka**

Make these changes on both the microservices

Add to pom.xml

<dependency>

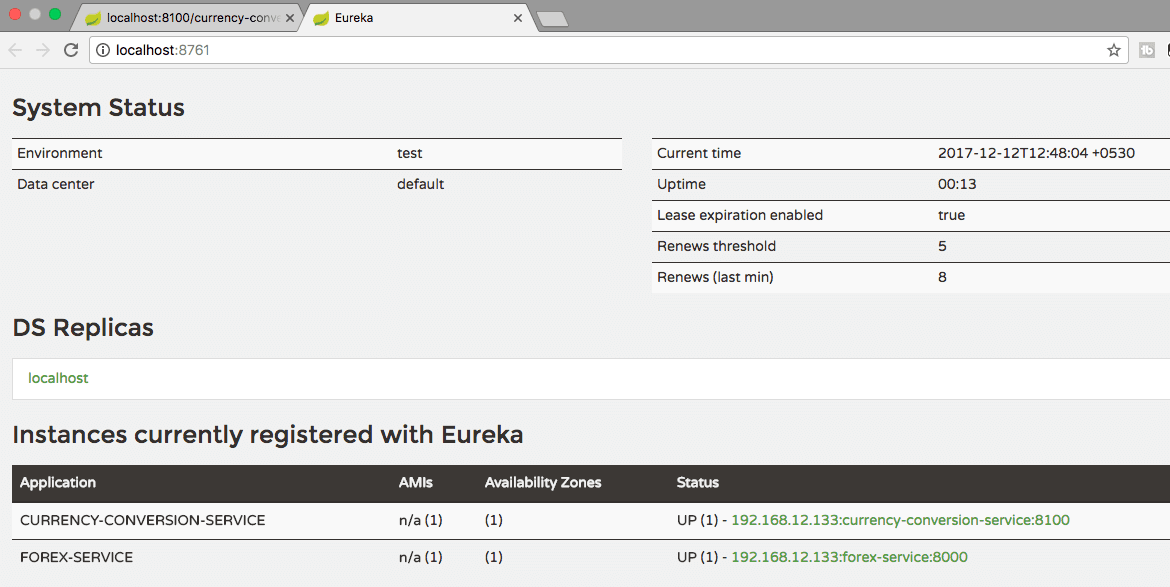
<groupId>org.springframework.cloud</groupId>

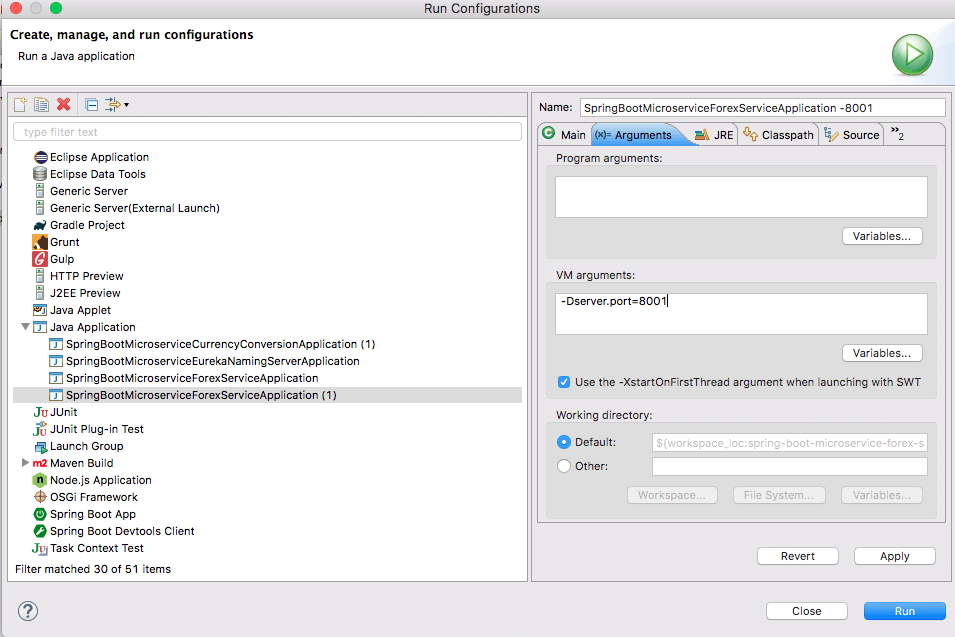
<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

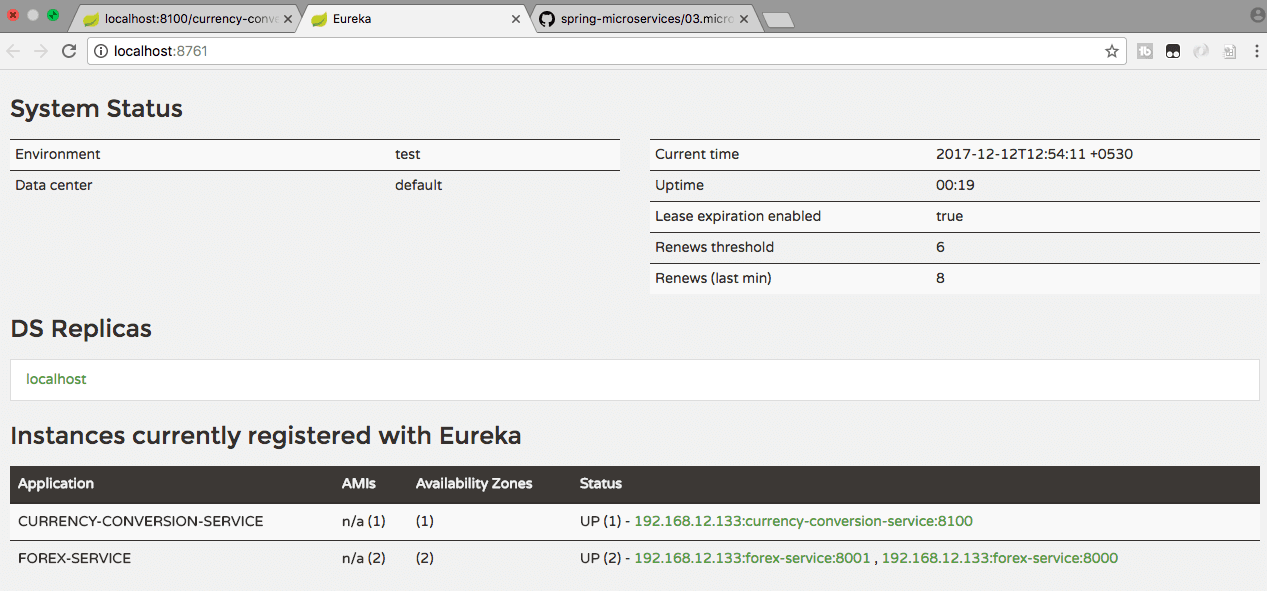
</dependency>

Configure Eureka URL in application.properties

eureka.client.service-url.default-zone=http://localhost:8761/eureka

Restart all the instances of CCS and FS. You would see that the CCS and FS microservices are registered with Eureka Naming Server. Thats Cool! 

Screenshot shows how to launch an additional instance of Forex Service on 8081. 

You would see that one instance of CCS and two instances of FS microservices are registered with Eureka Naming Server. 

**Routing Ribbon Requests Through Eureka**

All that you would need to do is to remove this configuration

Remove this configuration in application.properties

forex-service.ribbon.listOfServers=localhost:8000,localhost:8001

Restart the CCS instance.

**Eureka in Action**

Currently we have the following services up and running

* Currency Conversion Micro Service (CCS) on 8100
* Two instances of Forex MicroService on 8000 and 8001
* Eureka Server launched

Now you would see that the requests to CCS would get distributed between the two instances of Forex Microservice by Ribbon through Eureka.

**Request 1**

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

port: 8000,

}

**Request 2**

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

{

id: 10002,

from: "EUR",

to: "INR",

conversionMultiple: 75,

quantity: 10000,

totalCalculatedAmount: 750000,

port: 8001,

}

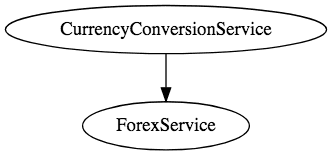
You can see that the port numbers in the two responses are different.

*Exercise : Launch up another instance of Forex Service on 8002. You would see that load gets automatically routed to it as well*

Cool! Thats awesome. Isn’t it.

**Summary**

We have now created two microservices and established communication between them.



We are using Ribbon to distribute load between the two instances of Forex Service and Eureka as the naming server.

When we launch new instances of Forex Service, you would see that load is automatically distribute to them.

The idea behind these series of 5 articles was to give a flavor of Spring Boot and Spring Cloud with Microservices.

There is a lot more ground to conver with Microservices. Until next time, Cheers!

**Complete Code Example**

**/spring-boot-microservice-currency-conversion-service/pom.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.in28minutes.springboot.microservice.example.currency-conversion</groupId>

<artifactId>spring-boot-microservice-currency-conversion</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-microservice-currency-conversion</name>

<description>Microservices with Spring Boot and Spring Cloud - Currency Conversion Service</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath /> *<!-- lookup parent from repository -->*

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Finchley.M8</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyConversionBean.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import java.math.BigDecimal;

public class CurrencyConversionBean {

private Long id;

private String from;

private String to;

private BigDecimal conversionMultiple;

private BigDecimal quantity;

private BigDecimal totalCalculatedAmount;

private int port;

public CurrencyConversionBean() {

}

public CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal quantity,

BigDecimal totalCalculatedAmount, int port) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

this.quantity = quantity;

this.totalCalculatedAmount = totalCalculatedAmount;

this.port = port;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFrom() {

return from;

}

public void setFrom(String from) {

this.from = from;

}

public String getTo() {

return to;

}

public void setTo(String to) {

this.to = to;

}

public BigDecimal getConversionMultiple() {

return conversionMultiple;

}

public void setConversionMultiple(BigDecimal conversionMultiple) {

this.conversionMultiple = conversionMultiple;

}

public BigDecimal getQuantity() {

return quantity;

}

public void setQuantity(BigDecimal quantity) {

this.quantity = quantity;

}

public BigDecimal getTotalCalculatedAmount() {

return totalCalculatedAmount;

}

public void setTotalCalculatedAmount(BigDecimal totalCalculatedAmount) {

this.totalCalculatedAmount = totalCalculatedAmount;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyConversionController.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import java.math.BigDecimal;

import java.util.HashMap;

import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.client.RestTemplate;

@RestController

public class CurrencyConversionController {

private Logger logger = LoggerFactory.getLogger(this.getClass());

@Autowired

private CurrencyExchangeServiceProxy proxy;

@GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

Map<String, String> uriVariables = new HashMap<>();

uriVariables.put("from", from);

uriVariables.put("to", to);

ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate().getForEntity(

"http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversionBean.class,

uriVariables);

CurrencyConversionBean response = responseEntity.getBody();

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

@GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")

public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,

@PathVariable BigDecimal quantity) {

CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to);

logger.info("{}", response);

return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,

quantity.multiply(response.getConversionMultiple()), response.getPort());

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyExchangeServiceProxy.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.cloud.netflix.ribbon.RibbonClient;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="forex-service")

@RibbonClient(name="forex-service")

public interface CurrencyExchangeServiceProxy {

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public CurrencyConversionBean retrieveExchangeValue

(@PathVariable("from") String from, @PathVariable("to") String to);

}

**/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/SpringBootMicroserviceCurrencyConversionApplication.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.feign.EnableFeignClients;

@SpringBootApplication

@EnableFeignClients("com.in28minutes.springboot.microservice.example.currencyconversion")

@EnableDiscoveryClient

public class SpringBootMicroserviceCurrencyConversionApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceCurrencyConversionApplication.class, args);

}

}

**/spring-boot-microservice-currency-conversion-service/src/main/resources/application.properties**

spring.application.name=currency-conversion-service

server.port=8100

eureka.client.service-url.default-zone=http://localhost:8761/eureka

**/spring-boot-microservice-currency-conversion-service/src/test/java/com/in28minutes/springboot/microservice/example/currencyconversion/SpringBootMicroserviceCurrencyConversionApplicationTests.java**

package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringBootMicroserviceCurrencyConversionApplicationTests {

@Test

public void contextLoads() {

}

}

**/spring-boot-microservice-eureka-naming-server/pom.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.in28minutes.springboot.microservice.eureka.naming.server</groupId>

<artifactId>spring-boot-microservice-eureka-naming-server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-microservice-eureka-naming-server</name>

<description>Microservices with Spring Boot and Spring Cloud - Eureka Naming Server</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath/> *<!-- lookup parent from repository -->*

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Finchley.M8</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

**/spring-boot-microservice-eureka-naming-server/src/main/java/com/in28minutes/springboot/microservice/eureka/naming/server/SpringBootMicroserviceEurekaNamingServerApplication.java**

package com.in28minutes.springboot.microservice.eureka.naming.server;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class SpringBootMicroserviceEurekaNamingServerApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceEurekaNamingServerApplication.class, args);

}

}

**/spring-boot-microservice-eureka-naming-server/src/main/resources/application.properties**

spring.application.name=netflix-eureka-naming-server

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**/spring-boot-microservice-eureka-naming-server/src/test/java/com/in28minutes/springboot/microservice/eureka/naming/server/SpringBootMicroserviceEurekaNamingServerApplicationTests.java**

package com.in28minutes.springboot.microservice.eureka.naming.server;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringBootMicroserviceEurekaNamingServerApplicationTests {

@Test

public void contextLoads() {

}

}

**/spring-boot-microservice-forex-service/pom.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.in28minutes.springboot.microservice.example.forex</groupId>

<artifactId>spring-boot-microservice-forex-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-boot-microservice-forex-service</name>

<description>Microservices with Spring Boot and Spring Cloud - Forex Service</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.0.RELEASE</version>

<relativePath/> *<!-- lookup parent from repository -->*

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

<spring-cloud.version>Finchley.M8</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</project>

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ExchangeValue.java**

package com.in28minutes.springboot.microservice.example.forex;

import java.math.BigDecimal;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class ExchangeValue {

@Id

private Long id;

@Column(name="currency\_from")

private String from;

@Column(name="currency\_to")

private String to;

private BigDecimal conversionMultiple;

private int port;

public ExchangeValue() {

}

public ExchangeValue(Long id, String from, String to, BigDecimal conversionMultiple) {

super();

this.id = id;

this.from = from;

this.to = to;

this.conversionMultiple = conversionMultiple;

}

public Long getId() {

return id;

}

public String getFrom() {

return from;

}

public String getTo() {

return to;

}

public BigDecimal getConversionMultiple() {

return conversionMultiple;

}

public int getPort() {

return port;

}

public void setPort(int port) {

this.port = port;

}

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ExchangeValueRepository.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.springframework.data.jpa.repository.JpaRepository;

public interface ExchangeValueRepository extends

JpaRepository<ExchangeValue, Long>{

ExchangeValue findByFromAndTo(String from, String to);

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/ForexController.java**

package com.in28minutes.springboot.microservice.example.forex;

import java.math.BigDecimal;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.env.Environment;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class ForexController {

@Autowired

private Environment environment;

@Autowired

private ExchangeValueRepository repository;

@GetMapping("/currency-exchange/from/{from}/to/{to}")

public ExchangeValue retrieveExchangeValue

(@PathVariable String from, @PathVariable String to){

ExchangeValue exchangeValue =

repository.findByFromAndTo(from, to);

exchangeValue.setPort(

Integer.parseInt(environment.getProperty("local.server.port")));

return exchangeValue;

}

}

**/spring-boot-microservice-forex-service/src/main/java/com/in28minutes/springboot/microservice/example/forex/SpringBootMicroserviceForexServiceApplication.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

@EnableDiscoveryClient

public class SpringBootMicroserviceForexServiceApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootMicroserviceForexServiceApplication.class, args);

}

}

**/spring-boot-microservice-forex-service/src/main/resources/application.properties**

spring.application.name=forex-service

server.port=8000

spring.jpa.show-sql=true

spring.h2.console.enabled=true

eureka.client.service-url.default-zone=http://localhost:8761/eureka

**/spring-boot-microservice-forex-service/src/main/resources/data.sql**

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10001,'USD','INR',65,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10002,'EUR','INR',75,0);

insert into exchange\_value(id,currency\_from,currency\_to,conversion\_multiple,port)

values(10003,'AUD','INR',25,0);

**/spring-boot-microservice-forex-service/src/test/java/com/in28minutes/springboot/microservice/example/forex/SpringBootMicroserviceForexServiceApplicationTests.java**

package com.in28minutes.springboot.microservice.example.forex;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)

@SpringBootTest

public class SpringBootMicroserviceForexServiceApplicationTests {

@Test

public void contextLoads() {

}

}

Comments :

1. For Feign to work, you need to add the annotation @EnableFeignClients in each of your microservice main application class along with @SpringBootApplication annotation.

2. For Eureka to work, you need to add the annotation @EnableEurekaClientin each of your microservice main application class along with @SpringBootApplication annotation. Also obviously for the Eureka Server spring boot application add: @EnableEurekaServer