

# Microservices With Spring Boot - Part 1 - Getting Started



by Ranga Karanam  MVB · Jan. 23, 18 · Microservices Zone

 Like (8)  Comment (0)  Save  Tweet

Learn the Benefits and Principles of [Microservices Architecture for the Enterprise](#)

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. In part 1 of this series, let's introduce the concept of microservices and understand how to create great microservices with Spring Boot and Spring Cloud.

## What You Will Learn

- What is a monolith?
- What is a microservice?
- What are the challenges with microservices?
- How do 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 the Naming Server and Ribbon.

## Resources Overview

In this guide, we will create a Student Resource exposing three services using the proper URIs and HTTP methods:

- Retrieve all Students - @GetMapping("/students")
- Get details of specific student - @GetMapping("/students/{id}")
- Delete a student - @DeleteMapping("/students/{id}")
- Create a new student - @PostMapping("/students")
- Update student details - @PutMapping("/students/{id}")

## 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>

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.

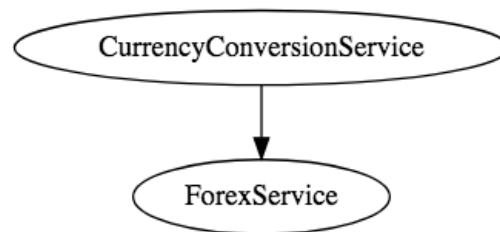
An example request and response is shown below:

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

```
{ id: 10002, from: "EUR", to: "INR", conversionMultiple: 75, port: 8000, }
```

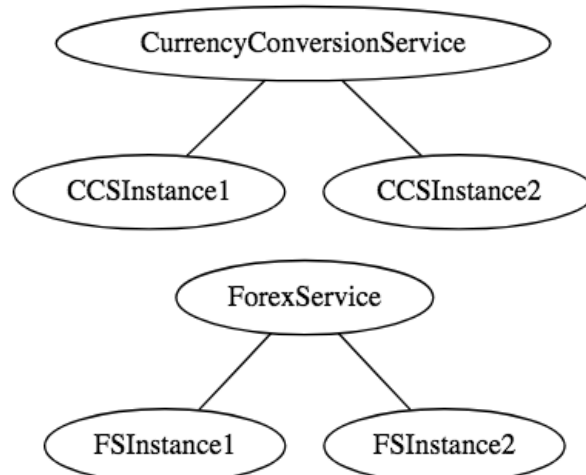
The request above is to find the value of 10000 EUR in INR. The totalCalculatedAmount is 750000 INR.

The diagram below shows the communication between CCS and FS.

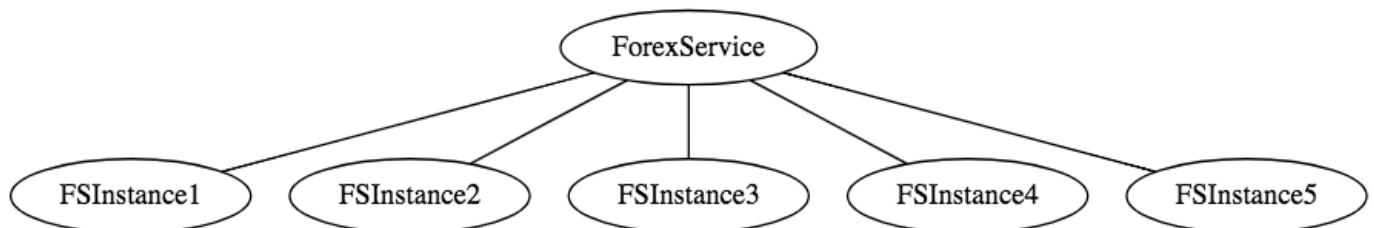


## Eureka Naming Server and Ribbon

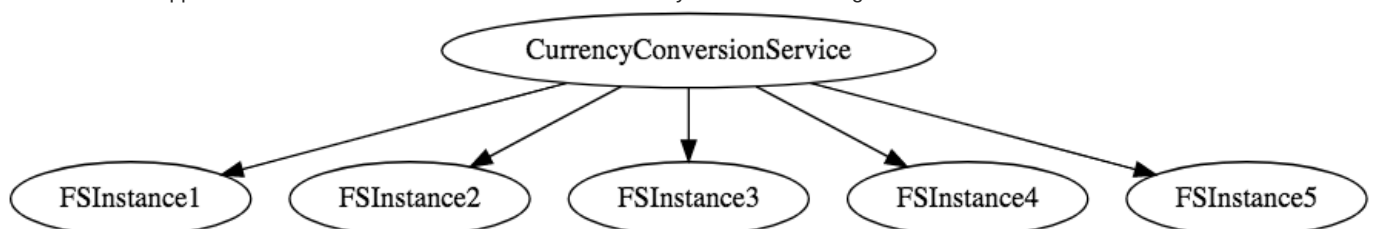
Based on the load, we can have multiple instances of the Currency Conversion Service and the Forex Service running.



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



What needs to happen in the above situation is load should be uniformly distributed among these five 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 on 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 on it?
- Where debugging problems is a big challenge?
- Where bringing in new technology and processes is almost impossible?

These are typical characteristics of a monolithic application. 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 perform automation tests
- Difficult to adapt to modern development practices
- Adapting to device explosion

## Microservices

Microservice architectures evolved as a solution to the scalability and innovation challenges with monolithic 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**

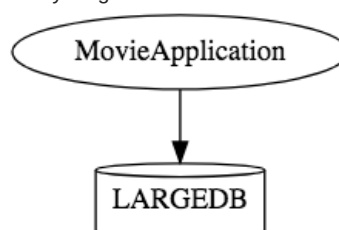
---

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

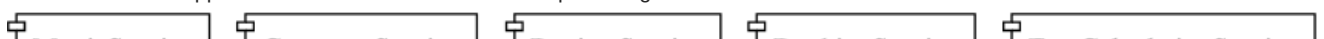
- REST - Built around RESTful Resources. Communication can be HTTP or event based.
- Small Well Chosen Deployable Units - Bounded Contexts
- Cloud Enabled - Dynamic Scaling

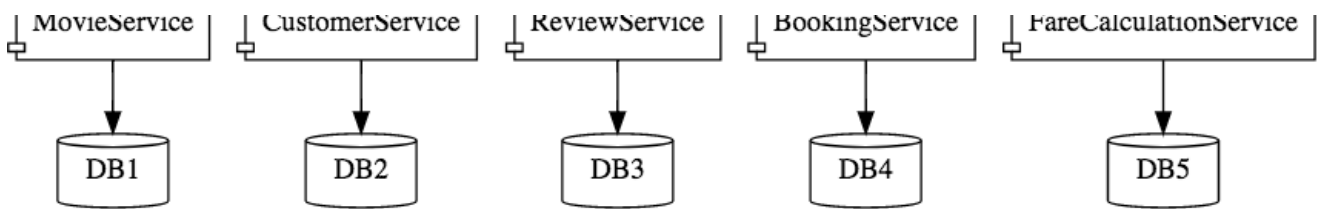
## What Does Microservice Architecture Look Like?

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

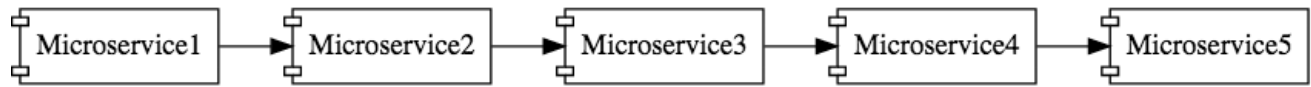


This is how the same application would look like when developed using microservices architecture.





Microservice architectures involve a number of small, well-designed components interacting with messages.



## Advantages of Microservices

Advantages of microservices include

- New technology and process adaption becomes easier. You can try new technologies with the newer microservices that we create.
- Faster release cycles.
- Scaling with the 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. Let's 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 are 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 be 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

Spring Boot enables building production-ready applications quickly and provides 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

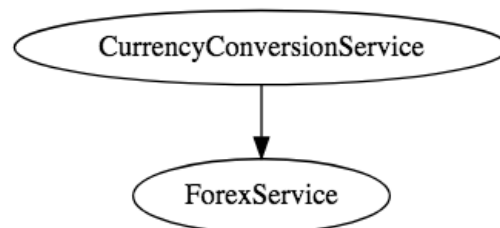
- Dynamically scale up and down. using a combination of
  - Naming Server (Eureka)

- 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.

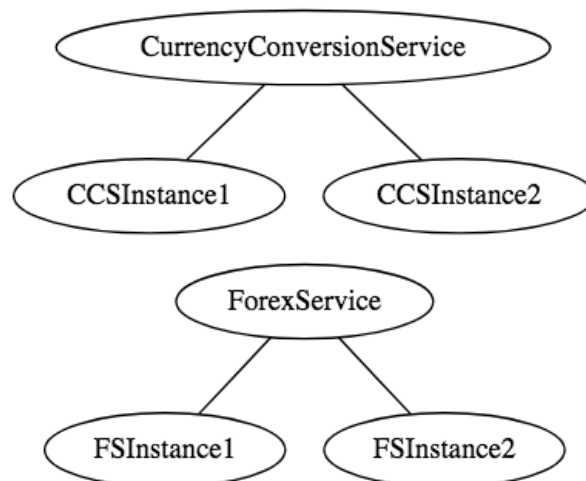
In this series of articles, we will create two microservices:

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

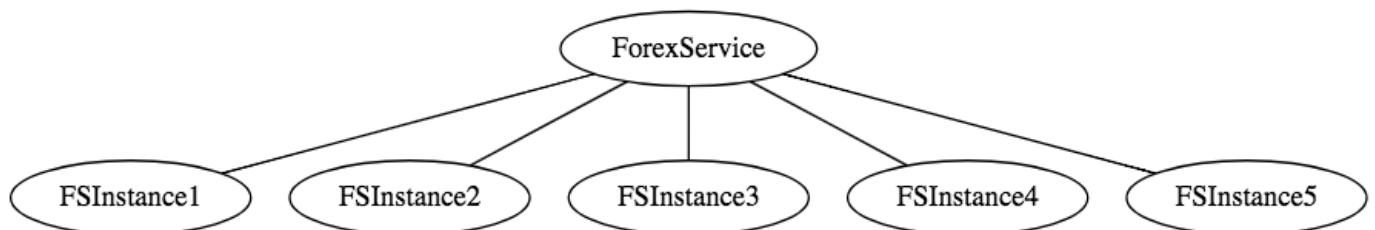
The diagram below shows the communication between CCS and FS. We would establish communication between these two components.



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

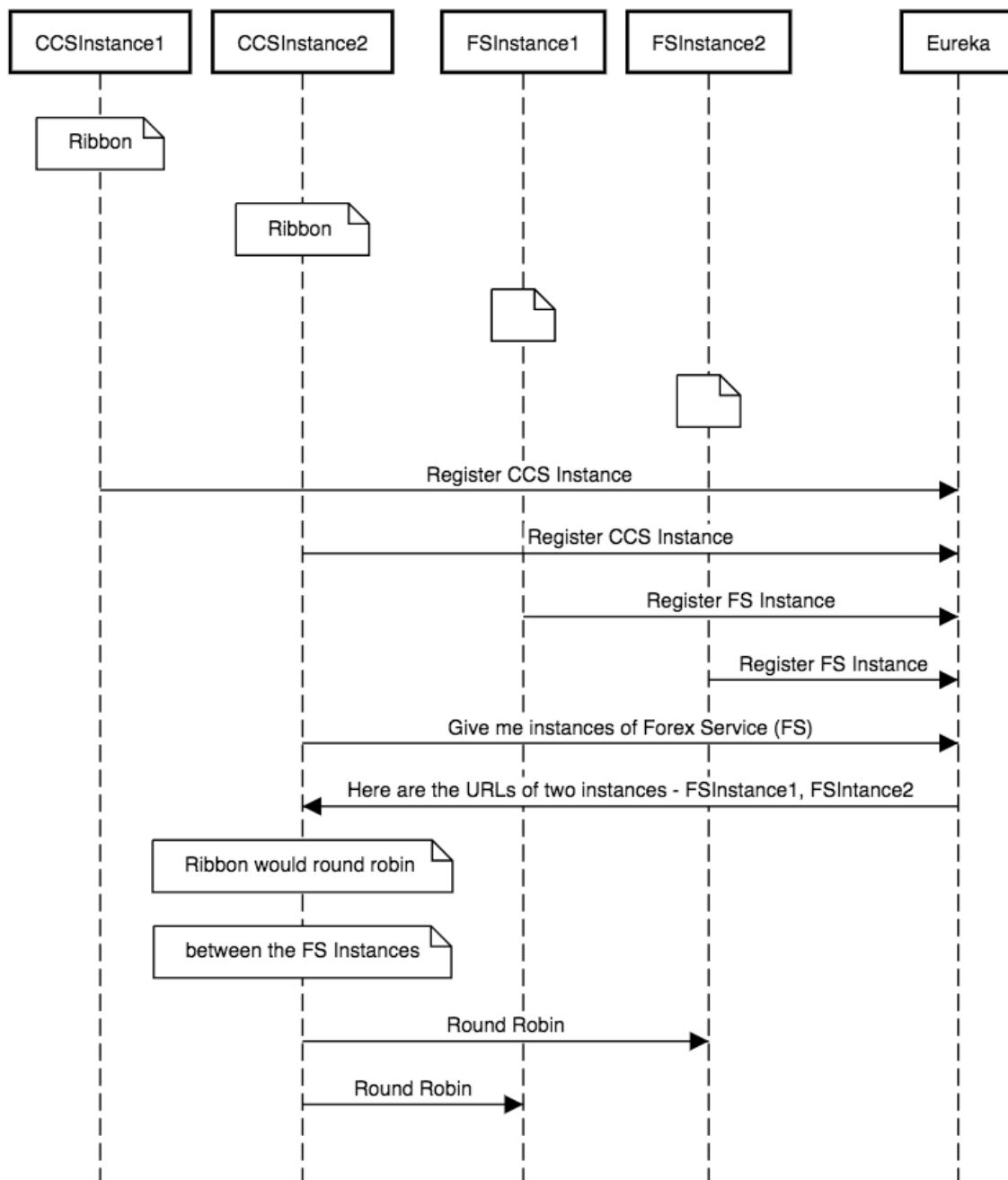
- How does the Currency Conversion Service (CCS) know how many instances of Forex Service (FS) are active?
- How does the Currency Conversion Service (CCS) distribute the load between the active instances?

Because we want this to be dynamic, we cannot hardcode the URLs of FS in CCS. That's 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:

### Microservices with Spring Boot



Next in this 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 implementation 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 for the Enterprise eBook: [Get Your Copy Here](#)

**Like This Article? Read More From DZone**

**Orchestrating Event Driven Microservices With Spring Dataflow**

Microservices With Spring Boot - Part 3 - Creating Currency Conversion Microservice

Microservices With Spring Boot - Part 5 - Using Eureka Naming Server


[Free DZone Refcard](#)  
Microservices in Java


Topics: MICROSERVICES , SPRING BOOT , SPRING CLOUD , TUTORIAL

 Like (8)  Comment (0)  Save  Tweet

Published at DZone with permission of Ranga Karanam, DZone MVB. [See the original article here.](#)  
Opinions expressed by DZone contributors are their own.

# Microservices Partner Resources

Microservices Architecture eBook: Download Your Copy Here   
CA Technologies

Learn the Benefits and Principles of Microservices Architecture   
CA Technologies

