# Microservices with Spring Boot and Spring Cloud

By: Prashant Gupta

## Agenda...

- Establishing Communication between Microservices
- Centralized Microservice Configuration with Spring Cloud Config Server
- Simplify communication with other Microservices using Feign REST Client
- Implement client side load balancing with Ribbon
- Implement dynamic scaling using Eureka Naming Server and Ribbon

## Microservices Introduction

In short, the microservice architectural style is an approach to 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.

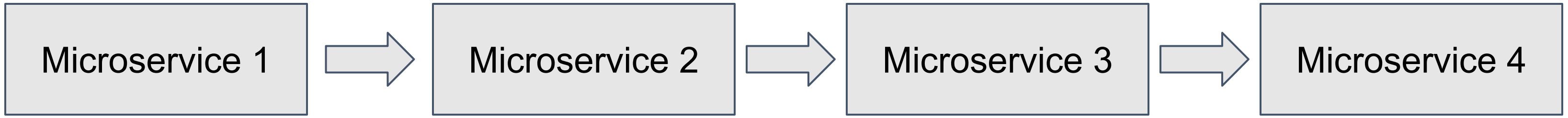
There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies **- James Lewis and Martin Fowler.**

Small autonomous services that work together **- Sam Newman.**

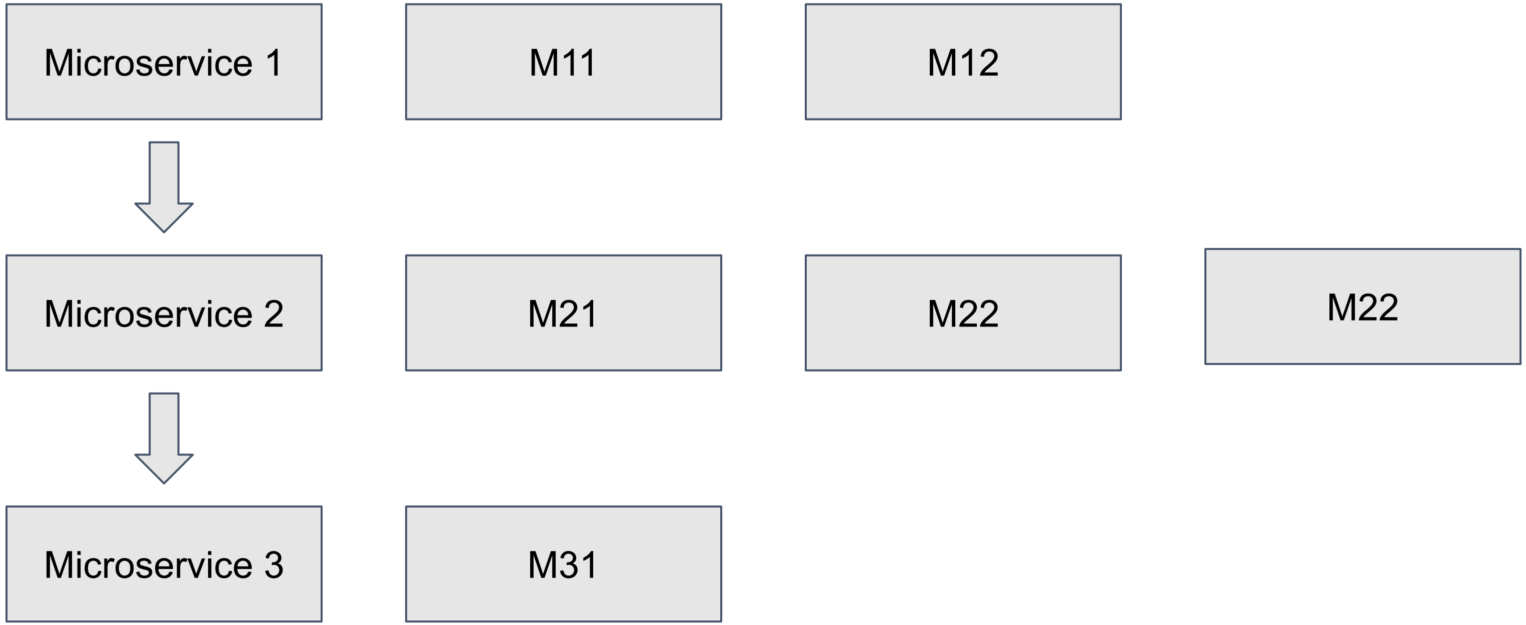
## Microservices

- REST
- Small Well Chosen Deployable Units
- Cloud Enabled

## Microservices



# Microservices

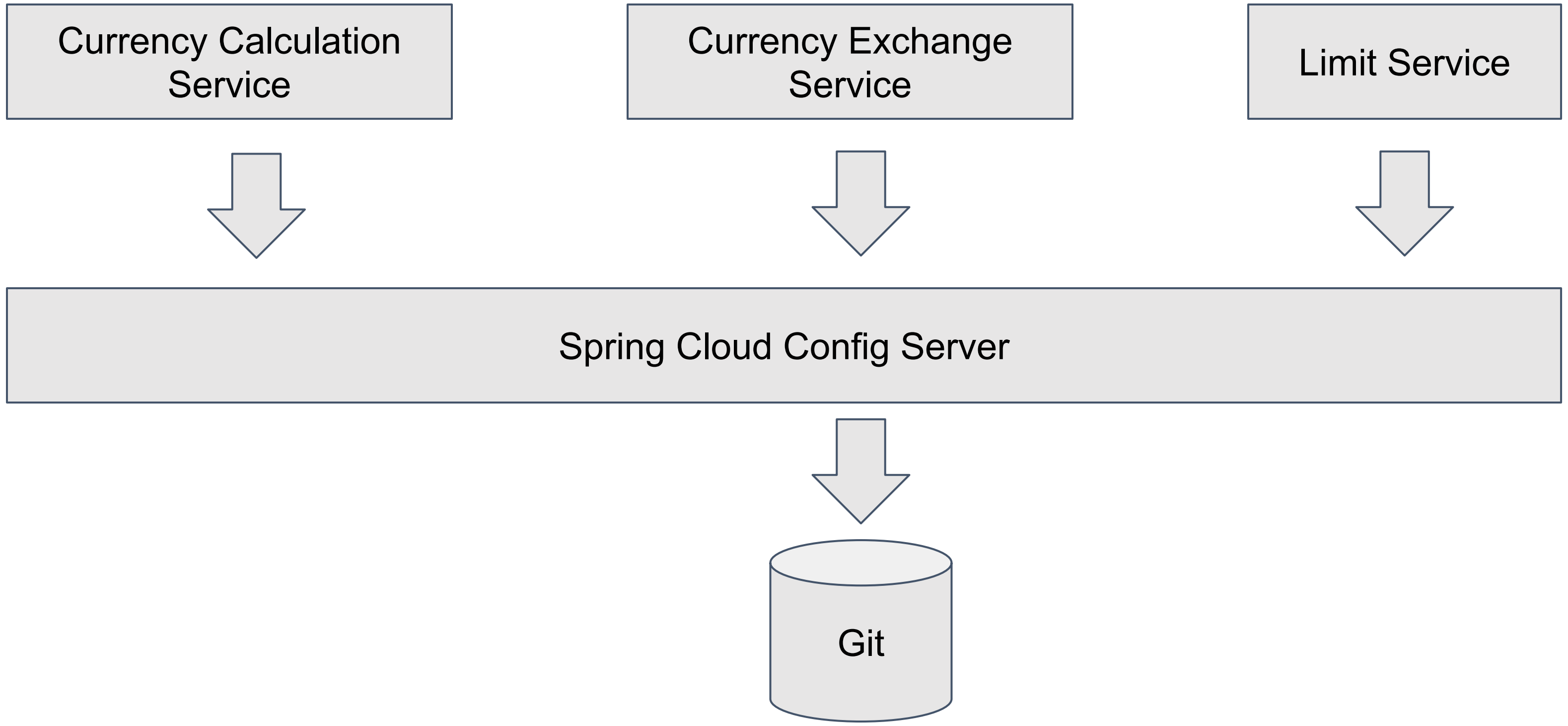


# Spring Cloud Introduction

## CENTRALIZED CONFIGURATION MANAGEMENT

Spring Cloud Config Server

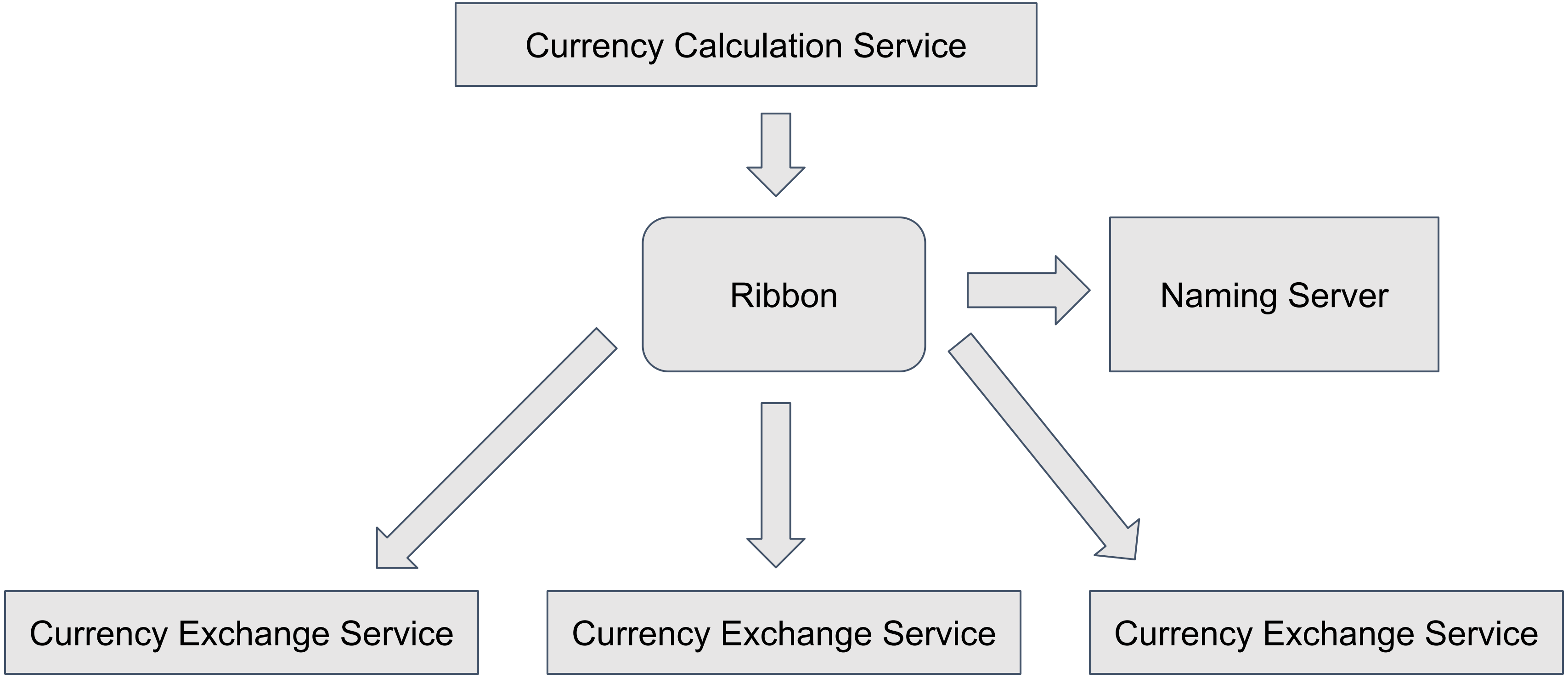
## Spring Cloud Config Server


Currency Calculation Service
Currency Exchange Service
Limit Service
Spring Cloud Config Server
Git

## Dynamic Scale Up and Scale Down

- Naming Server (Eureka) - LOCATION TRANSPARENCY
- Ribbon (Client Side) - LOAD DISTRIBUTION
- feign (Easier REST Clients)

# Ribbon Load Balancing

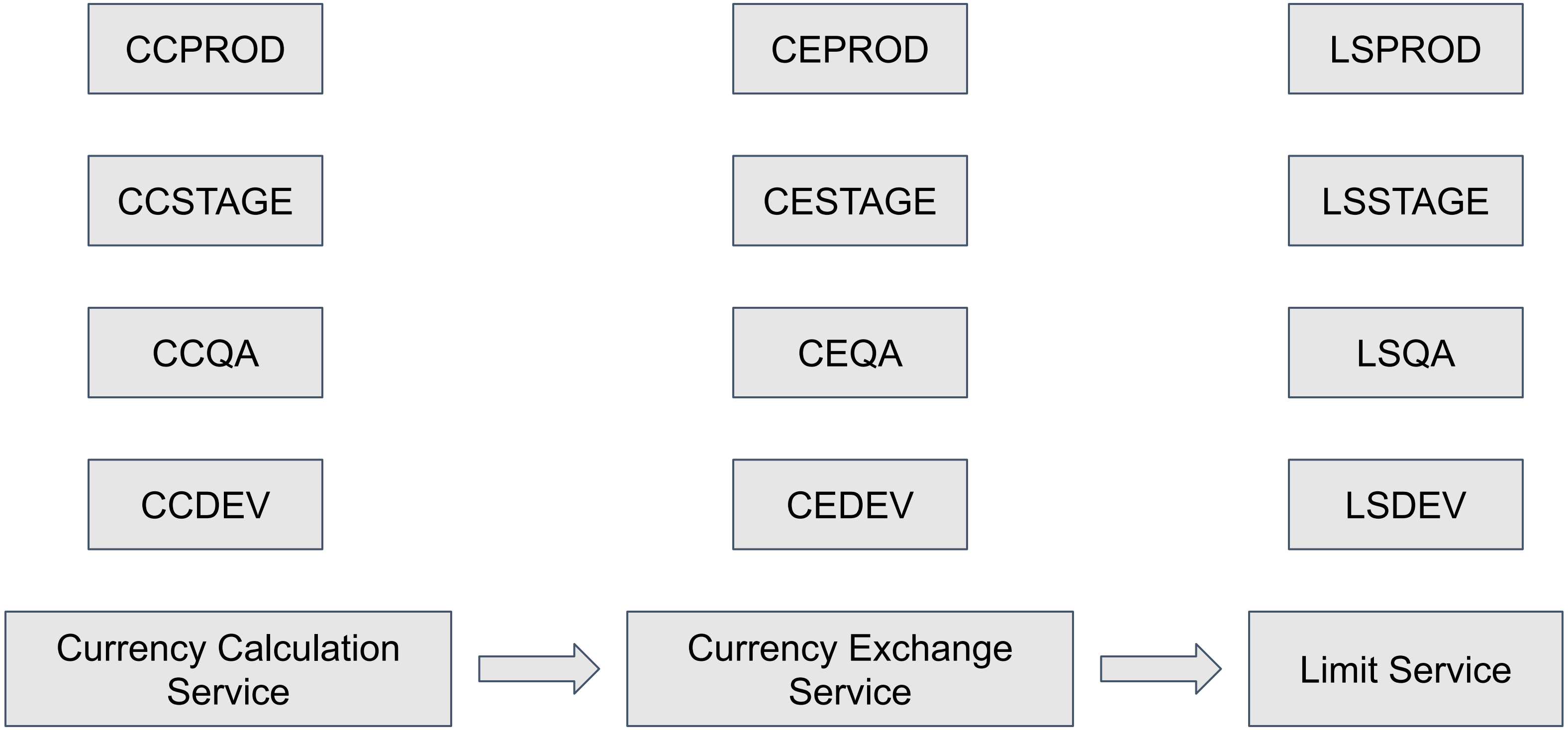

Currency Calculation Service
Ribbon
Naming Server
Currency Exchange Service
Currency Exchange Service
Currency Exchange Service

## Advantage of Spring Cloud

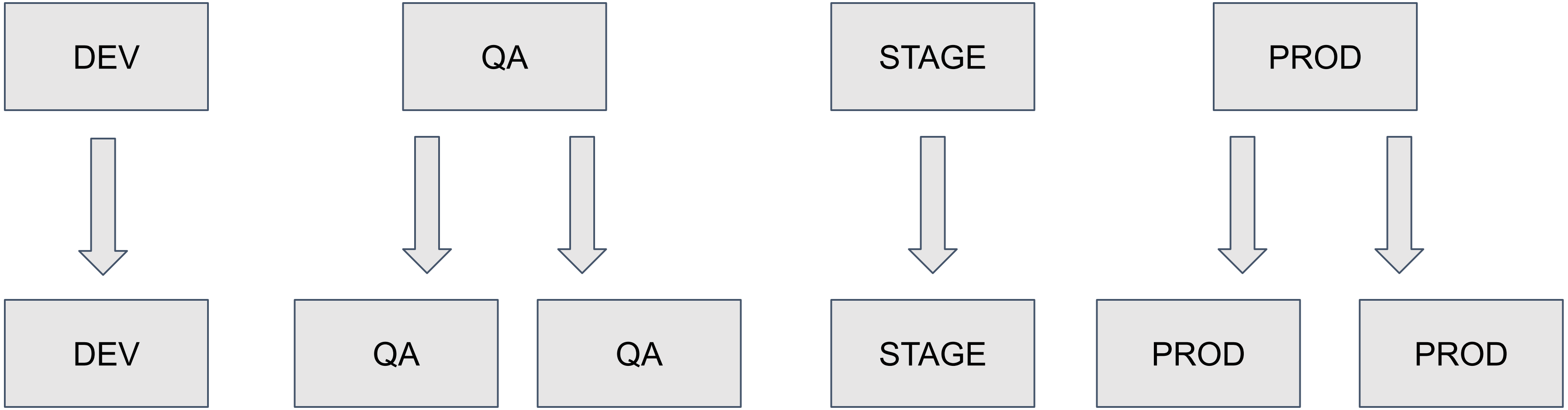
1. Adapt New Technology and process adaptation.
2. Dynamic Scaling.
3. faster release cycles.

## Spring Cloud Config

## Microservices Environments



# Currency Calculation Service



## Spring Cloud Config Implementation

## Spring Cloud Config Implementation

1. Create “limits-service” application and their configuration.
2. Create “spring-cloud-config-server” application and their configuration.

## Spring Cloud Config Implementation cont...

Limits Service App -

Dependency - Actuator, Web, Config Client, Dev Tools.

Spring Boot Version - 2.0.0

Spring Cloud Config App -

Dependency - Config Server, Dev Tools.

Spring Boot Version - 2.0.0

## Spring Cloud Config Implementation cont...

Git Spring cloud Config Folder - (Local commit these files)

limits-service.properties

```
limits-service.maximum=8888
limits-service.minimum=88
```

limits-service-dev.properties

```
limits-service.maximum=1111
limits-service.minimum=11
```

limits-service-qa.properties

```
limits-service.maximum=2222
limits-service.minimum=22
```

## Limit Service App (application.properties)

```
spring.application.name=limits-service
server.port=8080
limits-service.maximum=9999
limits-service.minimum=99
spring.cloud.config.uri=http://localhost:8888
spring.profiles.active=qa
management.security.enabled=false
management.endpoints.web.exposure.include=*
```

## Limit Service App (Limits Configuration Controller)

```
package com.springcloud.limitsservice;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class LimitsConfigurationController {

   @Autowired
   private Configuration configuration;

   @GetMapping("/limits")
   public LimitConfiguration retrieveLimitsFromConfigurations() {
       return new LimitConfiguration(configuration.getMaximum(),
               configuration.getMinimum());
   }

}
```

## Limit Service Limits App (Configuration)

```java
@Component
@ConfigurationProperties("limits-service")
public class Configuration {

   private int minimum;
   private int maximum;

   public void setMinimum(int minimum) {
       this.minimum = minimum;
   }

   public void setMaximum(int maximum) {
       this.maximum = maximum;
   }

   public int getMinimum() {
       return minimum;
   }

   public int getMaximum() {
       return maximum;
   }

}
```

## Spring Cloud Config App (application.properties)

```
spring.application.name=spring-cloud-config-server
server.port=8888
spring.cloud.config.server.git.uri=file:///home/prashant/Projects/GitHub/git-spring-cloud-config-server
spring.cloud.config.server.git.repos.limitsservicesimple.pattern=limitsservice
spring.cloud.config.server.git.repos.limitsservicesimple.uri=file:///home/prashant/projects/session/git-spring-cloud-config-server-simple
```

## Spring Cloud Config App (SpringCloudConfigServerApplication)

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@EnableConfigServer
@SpringBootApplication
public class SpringCloudConfigServerApplication {

   public static void main(String[] args) {
       SpringApplication.run(SpringCloudConfigServerApplication.class, args);
   }
}
```

## Refreshing environment properties

When the limit service app trying to get properties then on the running app application is not able to refresh the variable.

So there is another annotation which is available to refresh by actuator

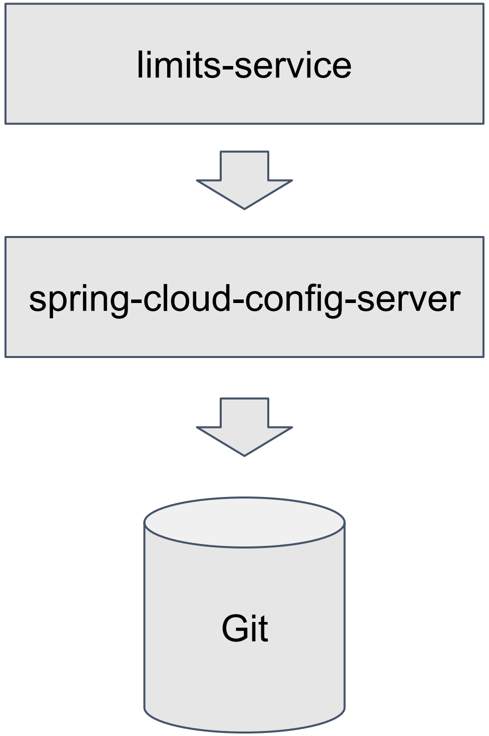
@RefreshScope

Add above annotation on controller.

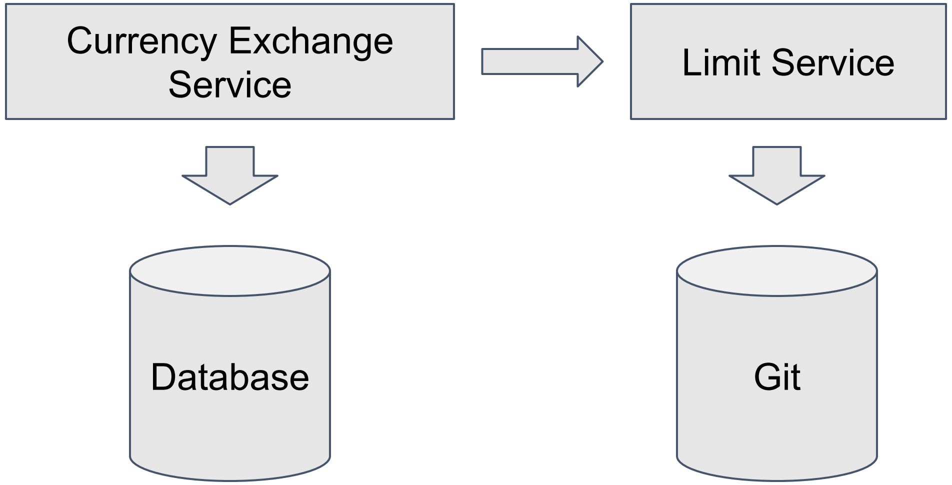
then run the below curl request to refresh.

curl -X POST http://localhost:8080/actuator/refresh

## Completed Spring Cloud Config Server



# Currency Exchange Service App



## Currency Exchange Service App

Dependencies - Actuator, Web, Config Client, Dev Tools, JPA, MYSQL.

Spring Boot: 2.0.0

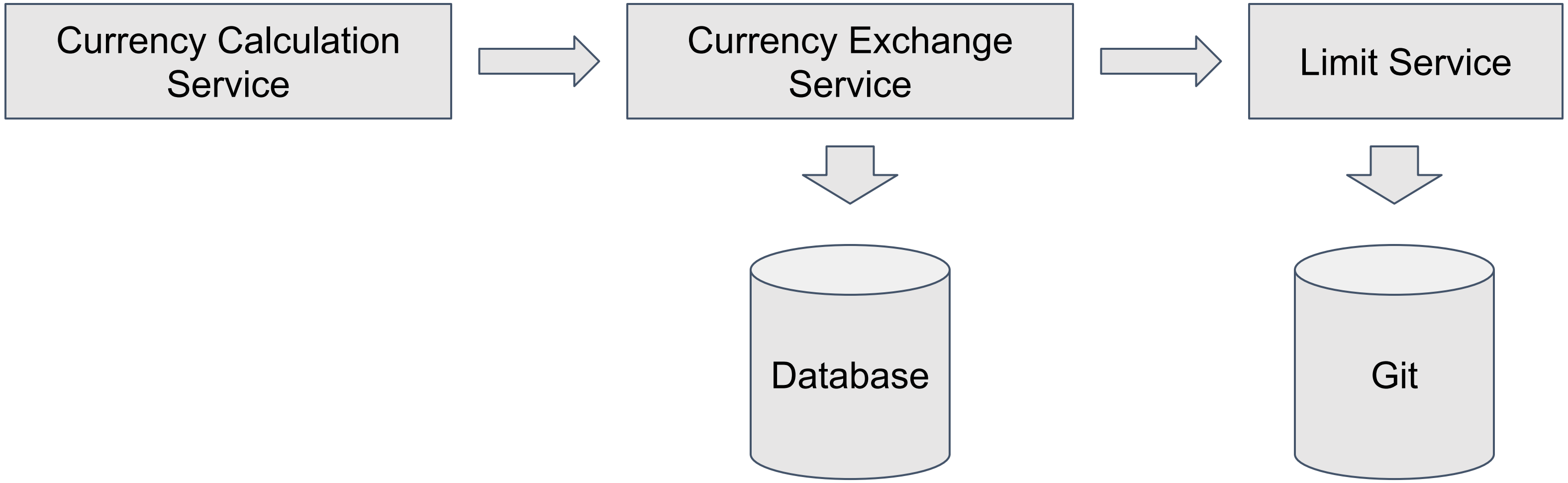
## CurrencyExchangeController

```
@RestController
public class CurrencyExchangeController {

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

   @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")));

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

       return exchangeValue;
   }

}
```

## application.properties

```
spring.application.name=currency-exchange-service
server.port=8000
```

# Currency Conversion Service



## Currency Conversion Service App

Dependencies - Actuator, Web, Config Client.

Spring Boot: 2.0.0

# CurrencyConversionController

```java
@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());
   }
}
```

## application.properties

```
spring.application.name=currency-conversion-service
server.port=8100
```

## Feign Client

Dependency - Feign Client

## Feign Client

```java
@EnableFeignClients("com.springcloud.currencyconversionservice")
@SpringBootApplication
public class CurrencyConversionServiceApplication {

   public static void main(String[] args) {
       SpringApplication.run(CurrencyConversionServiceApplication.class, args);
   }

}
```

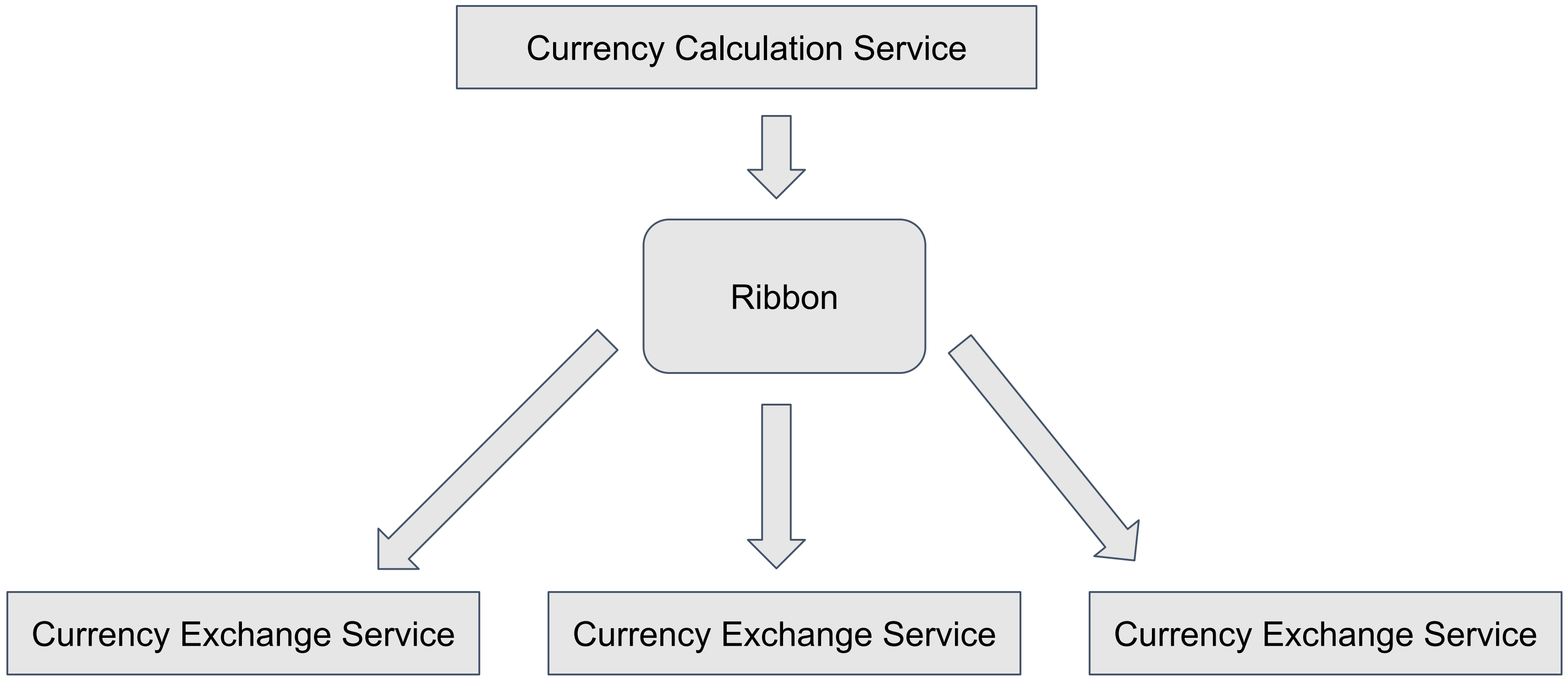
## CurrencyExchangeServiceProxy

```
@FeignClient(name = "currency-exchange-service", url =
"localhost:8000")
public interface CurrencyExchangeServiceProxy {
   @GetMapping("/currency-exchange/from/{from}/to/{to}")
   CurrencyConversionBean
retrieveExchangeValue(@PathVariable("from") String from,
@PathVariable("to") String to);
}
```

# CurrencyConversionController

```
@RestController
public class CurrencyConversionController {

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

   @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.toString());

         return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,
               quantity.multiply(response.getConversionMultiple()), response.getPort());
      }

}
```

## Ribbon Load Balancing


Currency Calculation Service
Ribbon
Currency Exchange Service
Currency Exchange Service
Currency Exchange Service

## Ribbon Load Balancing

Dependency - Ribbon Netflix

## application.properties

```
currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001
```

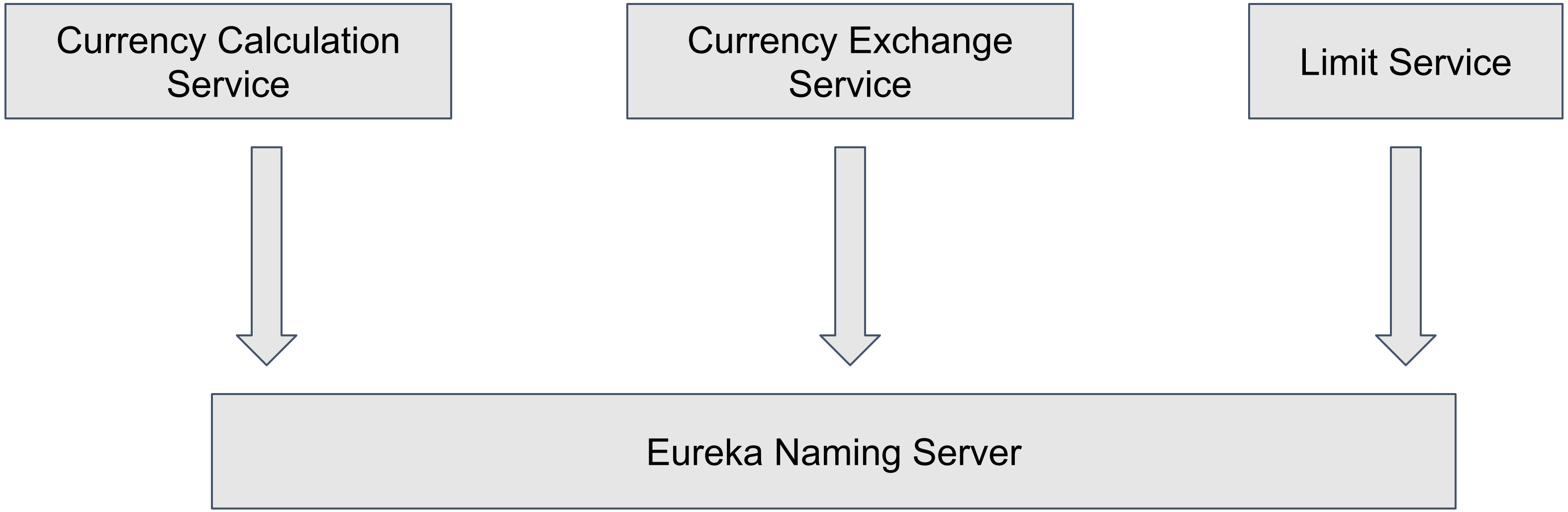
## build.gradle

```
compile('org.springframework.cloud:spring-cloud-starter-netflix-ribbon')
```

## CurrencyExchangeServiceProxy

```
@FeignClient(name = "currency-exchange-service")
@RibbonClient(name = "currency-exchange-service")
public interface CurrencyExchangeServiceProxy {
   @GetMapping("/currency-exchange/from/{from}/to/{to}")
   CurrencyConversionBean
retrieveExchangeValue(@PathVariable("from") String from,
@PathVariable("to") String to);
}


-Dserver.port=8001
```

## Eureka Naming Server



## Eureka Naming Server

Create new Application having name netflix-eureka-naming-server

Dependency - Eureka, Actuator, Config, Dev Tools

## application.properties

```
spring.application.name=netflix-eureka-naming-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

## NetflixEurekaNamingServerApplication

```
@EnableEurekaServer
@SpringBootApplication
public class NetflixEurekaNamingServerApplication {

   public static void main(String[] args) {
       SpringApplication.run(NetflixEurekaNamingServerApplication.class,
args);
   }
}
```

# Currency Conversion Service Application

Dependency - Eureka client/discovery

compile('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')

application.properties -

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

**CurrencyConversionServiceApplication -**

**Add anotation**

@EnableDiscoveryClient

## Currency Exchange Service Application

Dependency - Eureka client/discovery

```
compile('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')
```

application.properties -

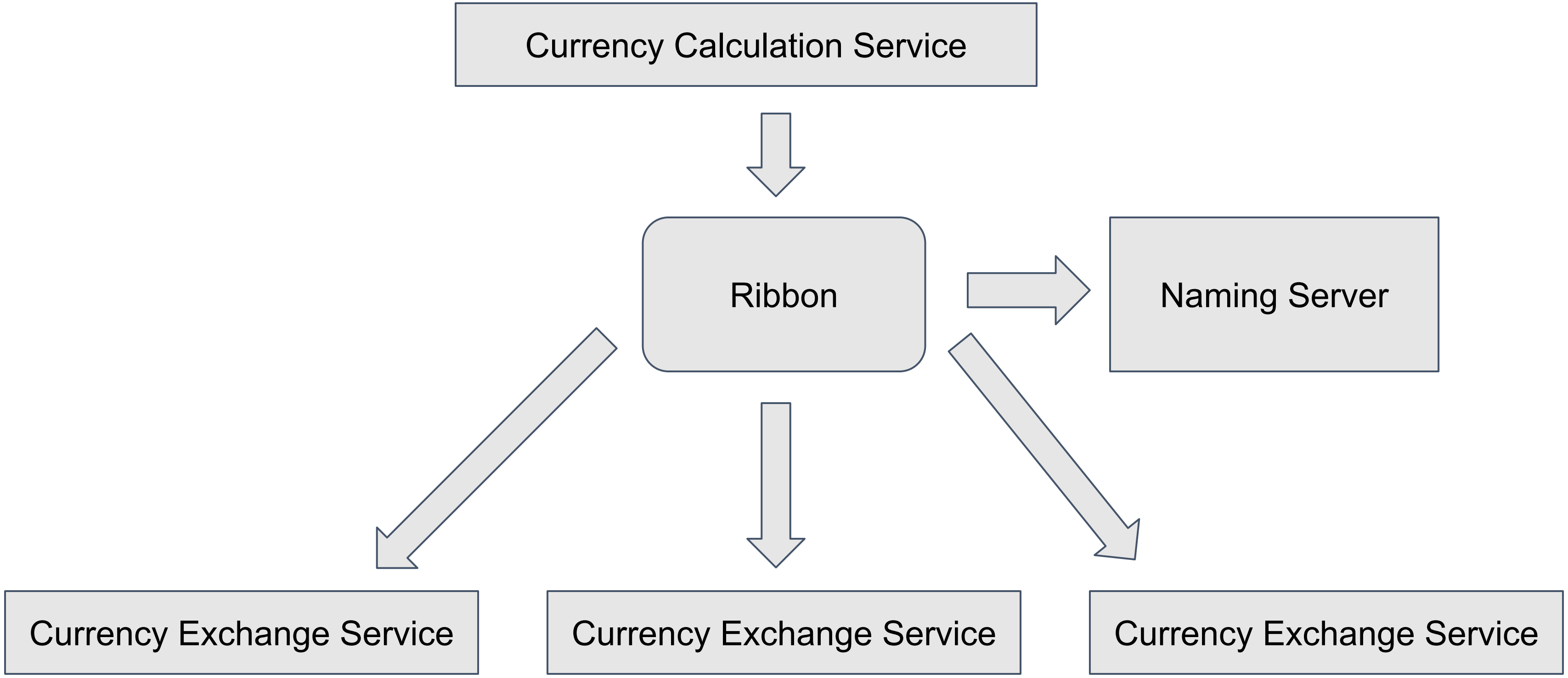
```
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

**CurrencyConversionServiceApplication -**

**Add anotation**

```
@EnableDiscoveryClient
```

# Ribbon Load Balancing


Currency Calculation Service
Ribbon
Naming Server
Currency Exchange Service
Currency Exchange Service
Currency Exchange Service

## Currency Conversion Service Application

application.properties -

```
#currency-exchange-service.ribbon.listOfServers=http://localhost:8000,http://localhost:8001
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

## Git Source

- git@github.com:prashantgupta123/microservices-spring-cloud-boot.git

# Thank You