Project URL

https://github.com/devops021/spring-microservices-v2

Debugging Guide

https://github.com/in28minutes/spring-microservices-v2/blob/main/03.microservices/01-step-by-step-changes/microservices-v2-1.md#spring-cloud-config-server---steps-01-to-08

Spring cloud

https://spring.io/projects/spring-cloud

https://spring.io/projects/spring-cloud-netflix

3.0.3

Centralized Configuration

Creating a hard coded limits service - V2

Getting hardcoded value

```
@GetMapping("/limits")
	public Limits retrieveLimits() {
		return new Limits(1,1000);
}
```

119. Step 01 - Setting up Limits Microservice - V2

To connect limit-service to spring-cloud-config-server we need below jar –

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

When we add in spring-cloud-starter-config what we would need to do is to configure how spring starter config needs to connect to spring cloud config server.

Application.properties
spring.config.import=optional:configserver:http://localhost:8888

122. Step 03 - Enhance limits service - Get configuration from application properties - V2

```
package com.in28minutes.microservices.limitsservice.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import com.in28minutes.microservices.limitsservice.bean.Limits;
import com.in28minutes.microservices.limitsservice.configuration.Configuration;

@RestController
public class LimitsController {

	@Autowired
	private Configuration configuration;

	@GetMapping("/limits")
	public Limits retrieveLimits() {
		return new Limits(configuration.getMinimum(),
				configuration.getMaximum());
//		return new Limits(1,1000);
	}
}
```

```
package com.in28minutes.microservices.limitsservice.configuration;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties("limits-service")//limits-service is name in application.properties before .
public class Configuration {
	private int minimum;
	private int maximum;

	public int getMinimum() {
		return minimum;
	}

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

	public int getMaximum() {
		return maximum;
	}

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

}
```

```
package com.in28minutes.microservices.limitsservice.bean;

public class Limits {
	private int minimum;
	private int maximum;

	public Limits() {
		super();
	}
```

```
	public Limits(int minimum, int maximum) {
		super();
		this.minimum = minimum;
		this.maximum = maximum;
	}

	public int getMinimum() {
		return minimum;
	}

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

	public int getMaximum() {
		return maximum;
	}

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

}
```

application.properties

```
limits-service.minimum=3
limits-service.maximum=9972
```

123. Step 04 - Setting up Spring Cloud Config Server - V2

Spring Cloud Config provides server-side and client-side support for externalized configuration in a distributed system. With the Config Server, you have a central place to manage external properties for applications across all environments. We can use Git, SVN, or HashiCorp.

```
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-config-server</artifactId>
		</dependency>
```

Installing Git and Creating Local Git Repository

Create a folder **git-localconfig-repo** and initialize git

```
git init
```

/git-localconfig-repo/limits-service.properties New

```
limits-service.minimum=1
limits-service.maximum=111
```

```
open git bash
```

```
git add *
git commit -m "First commit"
```

Connect Spring Cloud Config Server to Local Git Repository –

application.properties Modified

```
spring.application.name=spring-cloud-config-server
server.port=8888
#spring.cloud.config.server.git.uri=file:///in28Minutes/git/spring-microservices-v2/03.microservices/git-localconfig-repo
spring.cloud.config.server.git.uri=D:\\NEW_D_DRIVE\\References\\RnD\\DevOps\\MicroServices\\spring-microservices-v2-main\\spring-microservices-v2-main\\03.microservices\\git-localconfig-repo
```

SpringCloudConfigServerApplication.java

```
package com.in28minutes.microservices.springcloudconfigserver;

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);
	}
}
```

http://localhost:8888/limits-service/default


```
{
  "name": "limits-service",
  "profiles": [
    "default"
  ],
  "label": null,
  "version": "6848d10ff4506852c1553ecc2baa6210b239563f",
  "state": null,
  "propertySources": [
    {
      "name": "D:\\NEW_D_DRIVE\\References\\RnD\\DevOps\\MicroServices\\spring-microservices-v2-main\\spring-microservices-v2-main\\03.microservices\\git-localconfig-repo/file:C:\\Users\\MUKUL~1.ANA\\AppData\\Local\\Temp\\config-repo-5847434452671414885\\limits-service.properties",
      "source": {
        "limits-service.minimum": "1",
        "limits-service.maximum": "111"
      }
    }
  ]
}
```


127. Step 07 - Connect Limits Service to Spring Cloud Config Server - V2

In limit-service project add url of cloud config server and property file name which will be same as application name and jar `spring-cloud-starter-config is already added in pom`

application.properties

```
spring.application.name=limits-service
spring.config.import=optional:configserver:http://localhost:8888
```

http://localhost:8080/limits


```
{
  "minimum": 1,
  "maximum": 111
}
```

The above values are coming from cloud config server limits-server.properties default. The values in the application that properties have less priority compared
to the values which are present in your git repository.

Add few more properties file –
limits-service-qa.properties

limits-service.minimum=2
limits-service.maximum=222

limits-service-dev.properties

limits-service.minimum=3
limits-service.maximum=333

limits-service-prod.properties

limits-service.minimum=4
limits-service.maximum=444

http://localhost:8888/limits-service/prod

This would also return the default limits-service.properties file but the values which are configured in limits-service-dev will have higher priority than the values which are configured in limits-service.properties.


```
{
  "name": "limits-service",
  "profiles": [
    "prod"
  ],
  "label": null,
  "version": "922d7418db15475287e82bc8aa519f135bedd6e4",
  "state": null,
  "propertySources": [
    {
      "name": "D:\\NEW_D_DRIVE\\References\\RnD\\DevOps\\MicroServices\\spring-microservices-v2-main\\spring-microservices-v2-main\\03.microservices\\git-localconfig-repo/file:C:\\Users\\MUKUL~1.ANA\\AppData\\Local\\Temp\\config-repo-4839353651740153921\\limits-service-prod.properties",
      "source": {
        "limits-service.minimum": "4",
        "limits-service.maximum": "444"
      }
    },
    {
      "name": "D:\\NEW_D_DRIVE\\References\\RnD\\DevOps\\MicroServices\\spring-microservices-v2-main\\spring-microservices-v2-main\\03.microservices\\git-localconfig-repo/file:C:\\Users\\MUKUL~1.ANA\\AppData\\Local\\Temp\\config-repo-4839353651740153921\\limits-service.properties",
      "source": {
        "limits-service.minimum": "1",
        "limits-service.maximum": "111"
      }
    }
  ]
}
```


Note: We can pick different name of properties file from application name using below property in application.properties

```
spring.cloud.config.name=limits-service-002
```

We can configure the name suffix of configuration file we want to pick and use in application.properties –

```
spring.profiles.active=qa
spring.cloud.config.profile=qa
```

http://localhost:8080/limits

```
{
  "minimum": 2,
  "maximum": 222
}
```

```
spring.profiles.active=prod
spring.cloud.config.profile=prod
```

http://localhost:8080/limits

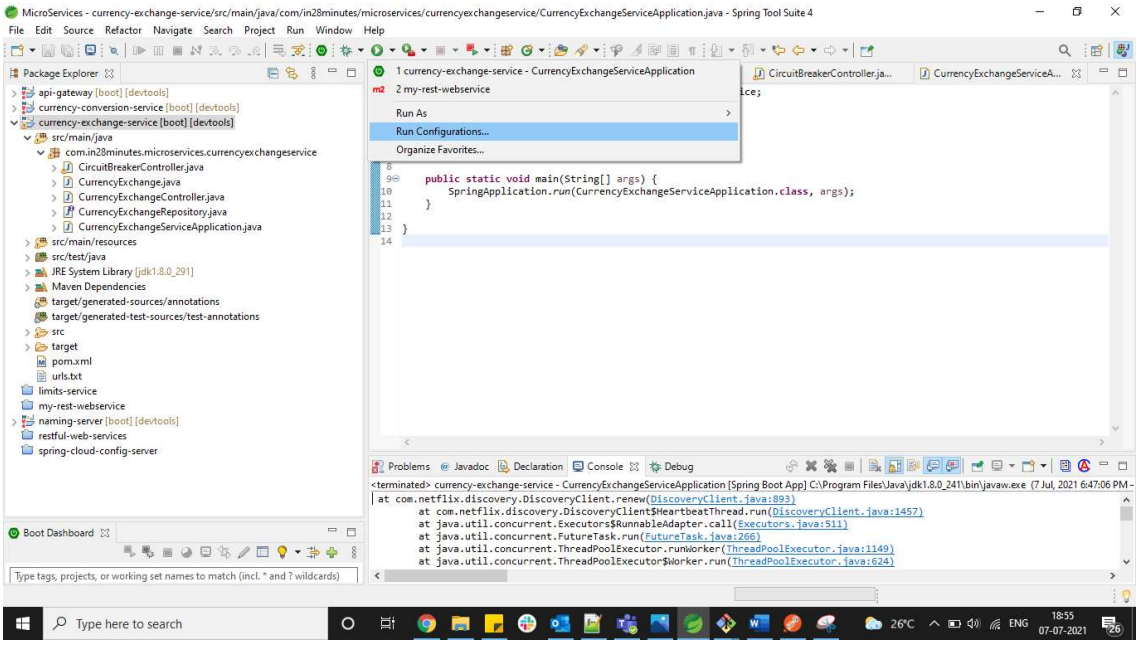
```
{
  "minimum": 4,
  "maximum": 444
}
```

---

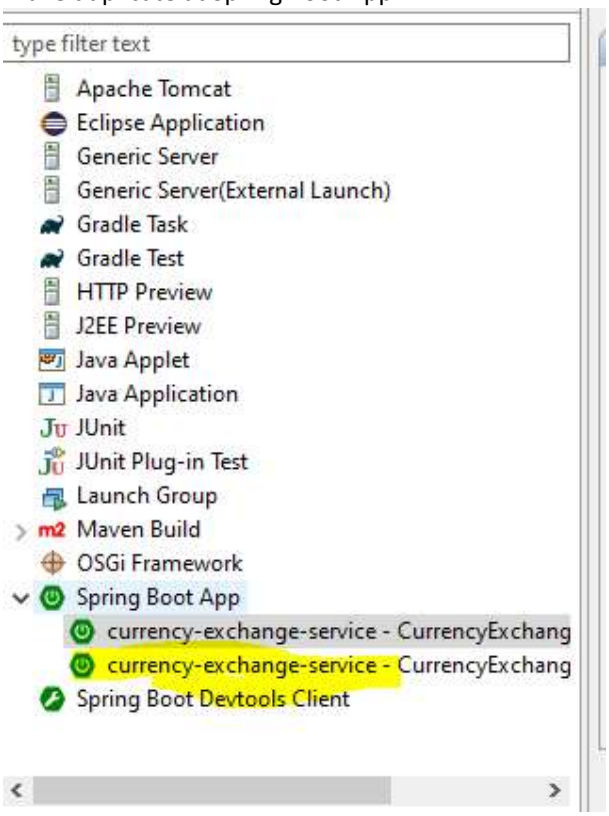
134. Step 12 - Setting up Dynamic Port in the the Response - V2

```
	@Autowired
	private Environment environment;

	@GetMapping("/currency-exchange/from/{from}/to/{to}")
	public CurrencyExchange retrieveExchangeValue(
			@PathVariable String from,
			@PathVariable String to) {

		logger.info("retrieveExchangeValue called with {} to {}", from, to);

		CurrencyExchange currencyExchange
				= repository.findByFromAndTo(from, to);

		if(currencyExchange ==null) {
			throw new RuntimeException
				("Unable to Find data for " + from + " to " + to);
		}

		String port = environment.getProperty("local.server.port");
		currencyExchange.setEnvironment(port);

		return currencyExchange;

	}
```

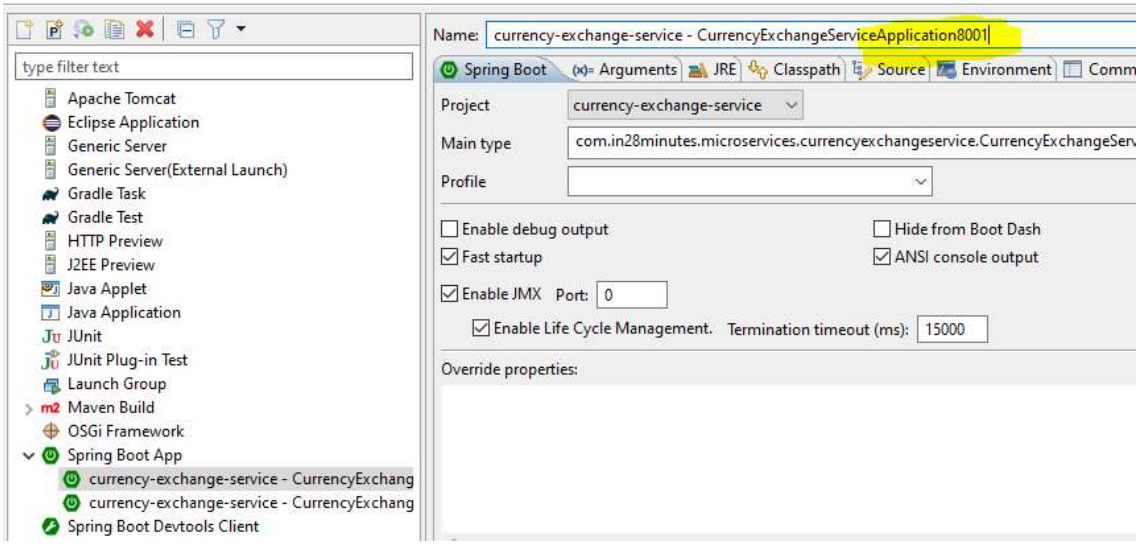
Go to Run configuration



Make duplicate at Spring Boot App

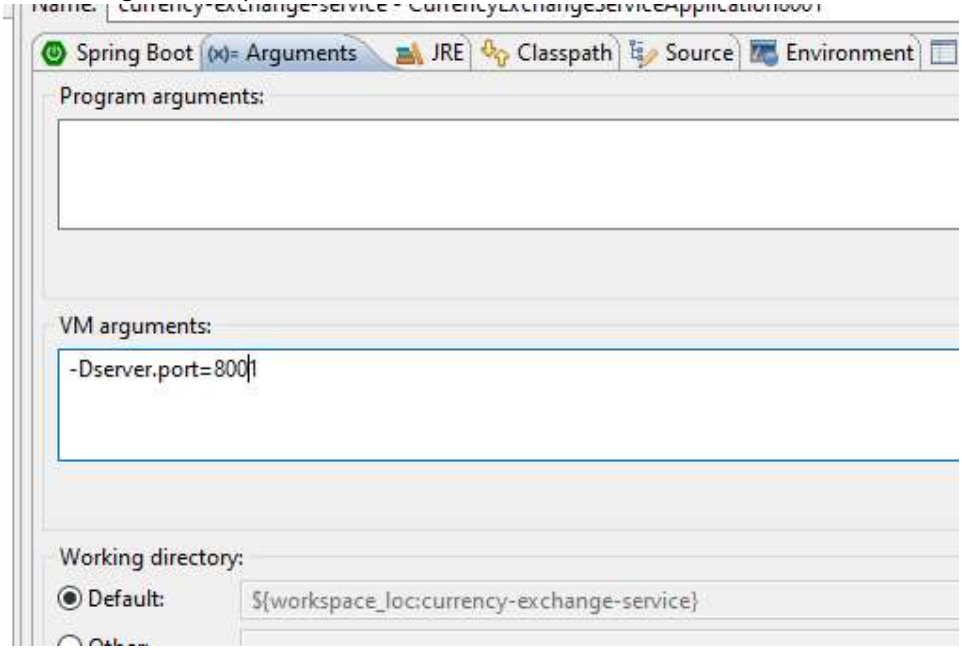


Rename with suffix 8000 and 8001



Apply

Provide Argument of port which will override 8000



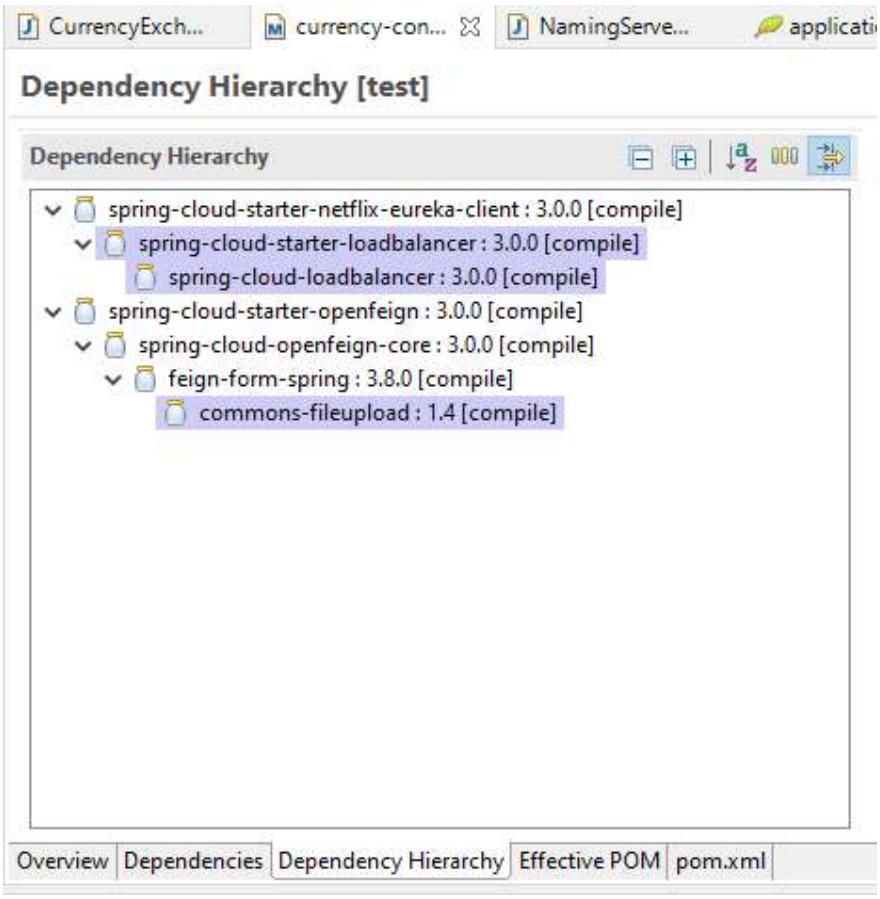
141. Step 17 - Invoking Currency Exchange from Currency Conversion Microservice - V2

```
@GetMapping("/currency-conversion/from/{from}/to/{to}/quantity/{quantity}")
	public CurrencyConversion calculateCurrencyConversion(
			@PathVariable String from,
			@PathVariable String to,
			@PathVariable BigDecimal quantity
			) {

		HashMap<String, String> uriVariables = new HashMap<>();
		uriVariables.put("from",from);
		uriVariables.put("to",to);
```

```
		ResponseEntity<CurrencyConversion> responseEntity = new
RestTemplate().getForEntity
		("http://localhost:8000/currency-exchange/from/{from}/to/{to}",
				CurrencyConversion.class, uriVariables);

		CurrencyConversion currencyConversion = responseEntity.getBody();

		return new CurrencyConversion(currencyConversion.getId(),
				from, to, quantity,
				currencyConversion.getConversionMultiple(),

	quantity.multiply(currencyConversion.getConversionMultiple()),
				currencyConversion.getEnvironment()+ " " + "rest
template");

	}
```

Feign (Client side load balancing)

Feign uses eureka clients spring-cloud-load-balancer for client side load balancing.



142. Step 18 - Using Feign REST Client for Service Invocation - V2

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

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

Client currency-conversion-service will have a proxy –

```
package com.in28minutes.microservices.currencyconversionservice;


import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="currency-exchange", url="localhost:8000")//name will be application name
configured in application.properties and URL will be base URI
public interface CurrencyExchangeProxy {

	@GetMapping("/currency-exchange/from/{from}/to/{to}")
	public CurrencyConversion retrieveExchangeValue(
			@PathVariable String from,
			@PathVariable String to); //Only method defination

}
```

Add an annotation in main class –

```
package com.in28minutes.microservices.currencyconversionservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class CurrencyConversionServiceApplication {

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

}
```

```
@Autowired
private CurrencyExchangeProxy proxy;

@GetMapping("/currency-conversion-feign/from/{from}/to/{to}/quantity/{quantity}")
	public CurrencyConversion calculateCurrencyConversionFeign(
			@PathVariable String from,
			@PathVariable String to,
			@PathVariable BigDecimal quantity
			) {
```

```
		CurrencyConversion currencyConversion =
proxy.retrieveExchangeValue(from, to);

		return new CurrencyConversion(currencyConversion.getId(),
				from, to, quantity,
				currencyConversion.getConversionMultiple(),

	quantity.multiply(currencyConversion.getConversionMultiple()),
				currencyConversion.getEnvironment() + " " + "feign");

}
```

Naming Server

naming-server add –

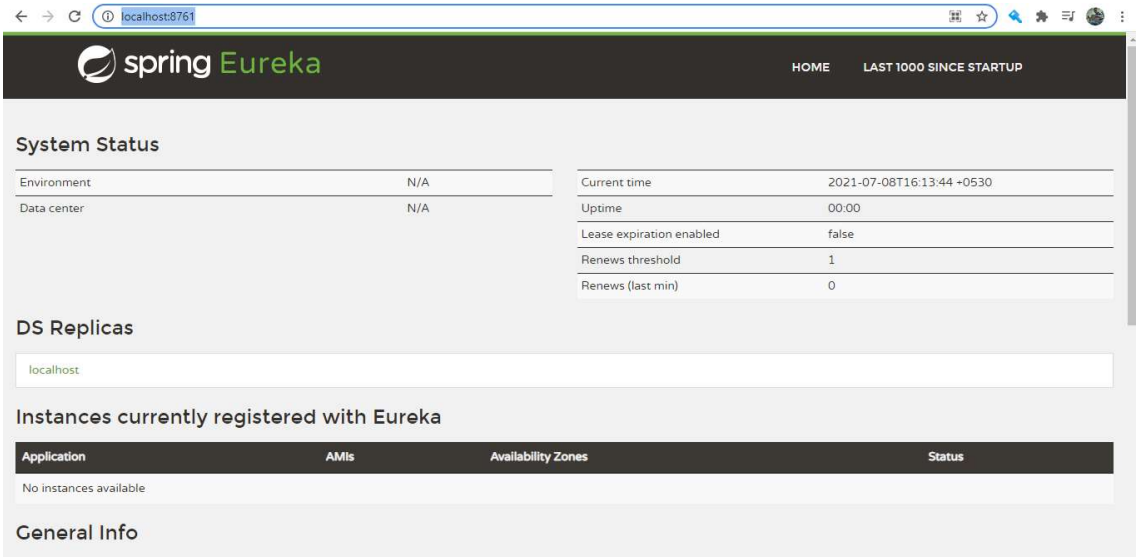
```
		<dependency>
			<groupId>org.springframework.cloud</groupId>
			<artifactId>spring-cloud-starter-netflix-eureka-
			server</artifactId>
		</dependency>
```

```
package com.in28minutes.microservices.namingserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer
@SpringBootApplication
public class NamingServerApplication {

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

}
```

Application.properties

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

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

http://localhost:8761/



145. Step 20 - Connect Currency Conversion & Currency Exchange Microservices - V2

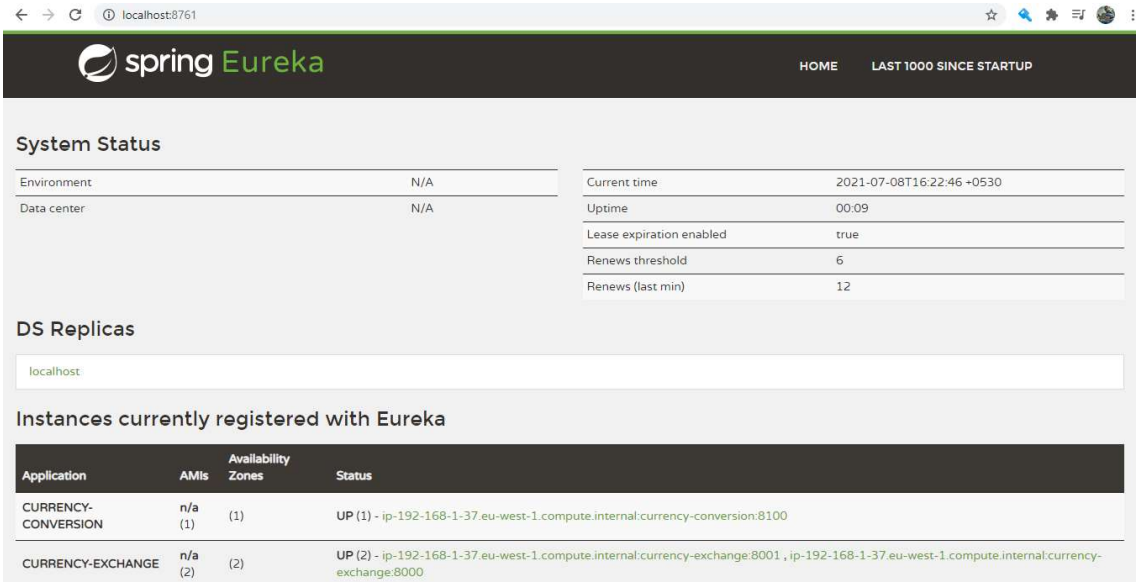
currency-conversion-service and currency-exchange-service add –

```
<dependency>
	<groupId>org.springframework.cloud</groupId>
	<artifactId>spring-cloud-starter-netflix-eureka-
	client</artifactId>
</dependency>
```

Add url of eureka server in all applications –

`Application.properties`

```
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```



General Info

Name	Value
total-avail-memory	377mb
num-of-cpus	6
current-memory-usage	220mb (58%)
server-uptime	00:09
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	192.168.1.37
status	UP

147. Step 22 - Load Balancing with Eureka, Feign & Spring Cloud LoadBalancer - V2

```
package com.in28minutes.microservices.currencyconversionservice;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;


//@FeignClient(name="currency-exchange", url="localhost:8000")
@FeignClient(name="currency-exchange")
public interface CurrencyExchangeProxy {

	@GetMapping("/currency-exchange/from/{from}/to/{to}")
	public CurrencyConversion retrieveExchangeValue(
			@PathVariable String from,
			@PathVariable String to);

}
```

Use feign URL of currency conversion service to see load balancing –

http://localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10

localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10


```
{
    "id": 10001,
    "from": "USD",
    "to": "INR",
    "quantity": 10,
    "conversionMultiple": 65.00,
    "totalCalculatedAmount": 650.00,
    "environment": "8000 feign"
}
```

148. Step 22 - Setting up Spring Cloud API Gateway

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

Application.properties

```
spring.application.name=api-gateway
server.port=8765
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
#To Allow the api gateway to discover other applications configured in eureka
spring.cloud.gateway.discovery.locator.enabled=true
#To allow url in lower case also
#spring.cloud.gateway.discovery.locator.lowerCaseServiceId=true
spring.sleuth.sampler.probability=1.0
```

All the common logic like authentication, logging can be implemented using api-gateway.

http://localhost:8765/CURRENCY-EXCHANGE/currency-exchange/from/USD/to/INR

153. Step 25 - Implementing Spring Cloud Gateway Logging Filter

```
package com.in28minutes.microservices.apigateway;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;

import reactor.core.publisher.Mono;

@Component
```

```
public class LoggingFilter implements GlobalFilter {

	private Logger logger = LoggerFactory.getLogger(LoggingFilter.class);

	@Override
	public Mono<Void> filter(ServerWebExchange exchange,
				GatewayFilterChain chain) {
		logger.info("Path of the request received -> {}",
					exchange.getRequest().getPath());
		return chain.filter(exchange);
	}

}
```

```
2021-07-08 17:12:23.259  INFO [api-gateway,,] 20788 --- [ctor-http-nio-3] c.i.m.apigateway.LoggingFilter          : Path of the request received -> /currency-conversion-feign/from/USD/to/INR/quantity/10
```

Circuit Breaker (Fault tolerance)

154. Step 26 - Getting started with Circuit Breaker - Resilience4j

currency-exchange-service add –

We need following dependencies –

```
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
		</dependency>


		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-aop</artifactId>
		</dependency>

		<dependency>
			<groupId>io.github.resilience4j</groupId>
			<artifactId>resilience4j-spring-boot2</artifactId>
		</dependency>
```

```
Application.properties
resilience4j.retry.instances.sample-api.maxRetryAttempts=5
resilience4j.retry.instances.sample-api.waitDuration=1s
resilience4j.retry.instances.sample-api.enableExponentialBackoff=true

#resilience4j.circuitbreaker.instances.default.failureRateThreshold=90

-------------------------------------------------------------------------------
```

```
package com.in28minutes.microservices.currencyexchangeservice;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

import io.github.resilience4j.bulkhead.annotation.Bulkhead;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import io.github.resilience4j.ratelimiter.annotation.RateLimiter;

@RestController
public class CircuitBreakerController {

	private Logger logger =
				LoggerFactory.getLogger(CircuitBreakerController.class);

	@GetMapping("/sample-api")
	//@Retry(name = "sample-api", fallbackMethod = "hardcodedResponse")
	//@CircuitBreaker(name = "default", fallbackMethod = "hardcodedResponse")
	//@RateLimiter(name="default")
	@Bulkhead(name="sample-api")
	//10s => 10000 calls to the sample api
	public String sampleApi() {
		logger.info("Sample api call received");
//		ResponseEntity<String> forEntity = new
RestTemplate().getForEntity("http://localhost:8080/some-dummy-url",
//					String.class);
//		return forEntity.getBody();
		return "sample-api";
	}

	public String hardcodedResponse(Exception ex) {
		return "fallback-response";
	}
}
```