1. **Microservices** : Small autonomous services that work together

🡪Rest

🡪Small well chosen boundries having Deployable units

🡪Cloud enabled(multiple instances are possible)

1. **Challenges with micro services:-**

🡪To identify the boundary of the microservices

🡪Configuration management(10 ms with 4 environments and 50 instances)

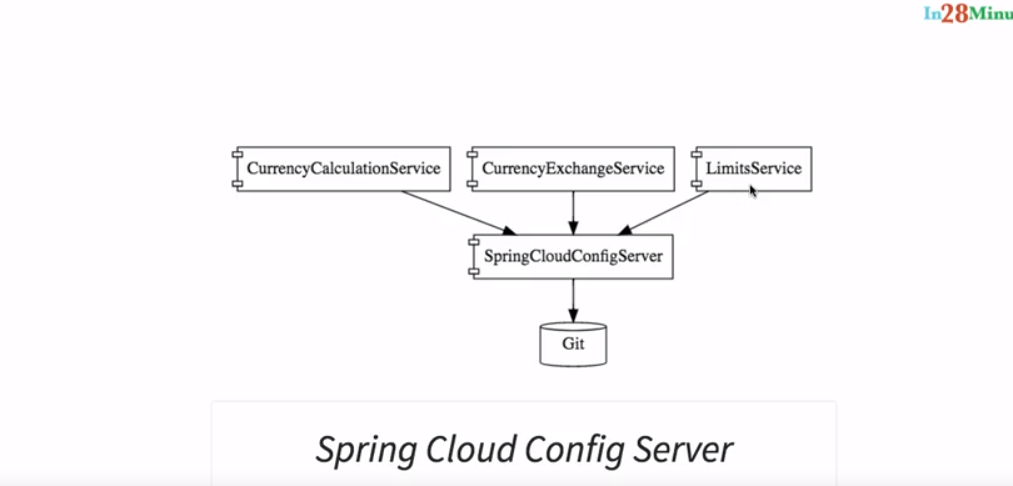
🡪upscaling the microservices and load balancing

🡪Visibility(identify the bug: soln centralised log)

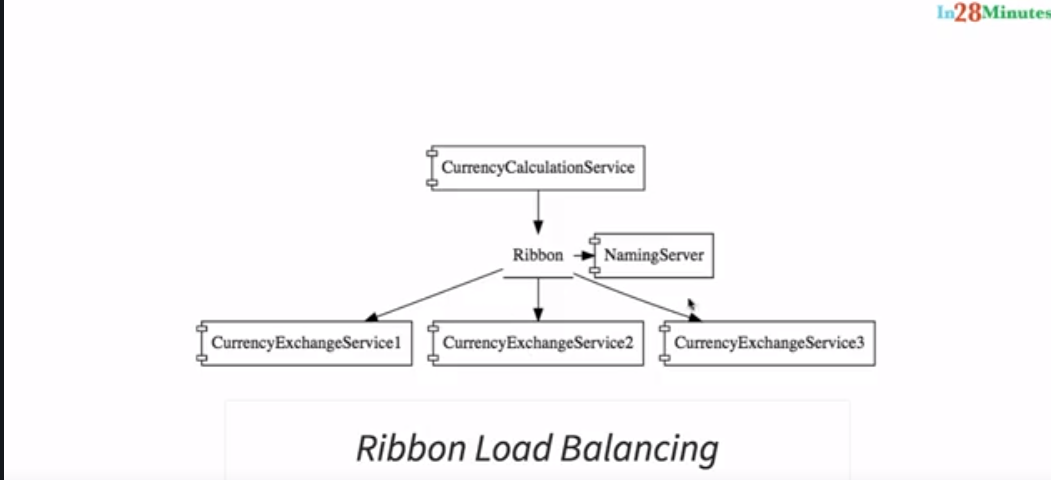
1. **Spring Cloud:-**

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, cluster state). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer’s own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

🡪**Configuration management**:-Spring cloud config server(git repo)



🡪**upscaling**: using naming server(eureka), Ribbon(client side load balancing), feign client(to call services)

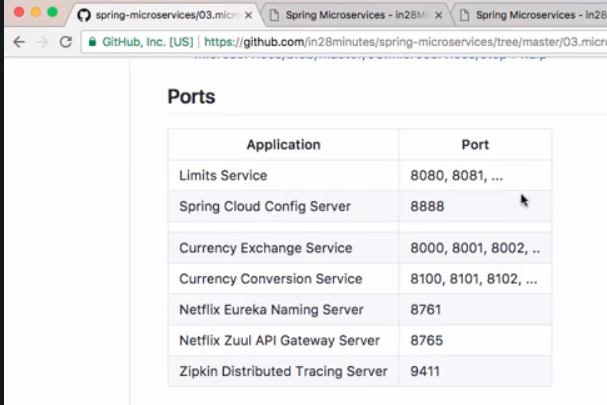


**🡪Visibility and monitoring**: Zipkin Distributed tracing, Netflix Api gateway(zull): for logging, security etc which is common for all micro services

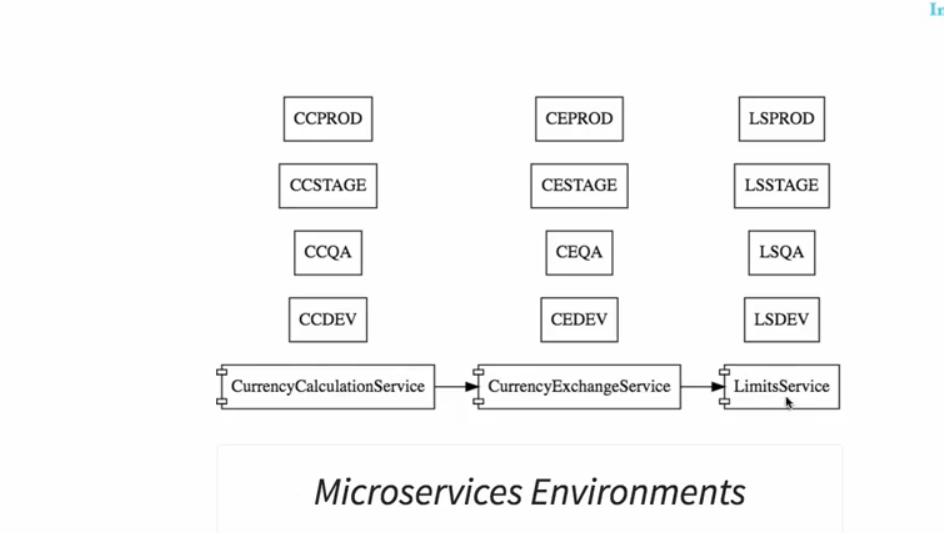
**🡪Fault tolerance**: Hysterix

1. **Advantages of microservices:-**
2. Use of multiple technologies
3. Dynamic scaling
4. Faster release

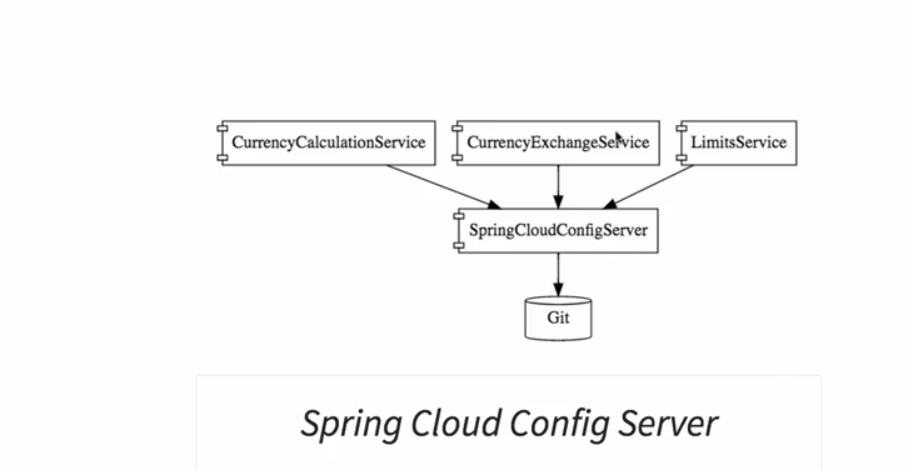
**5. Standarization of ports:-**



**6. Environments for services**

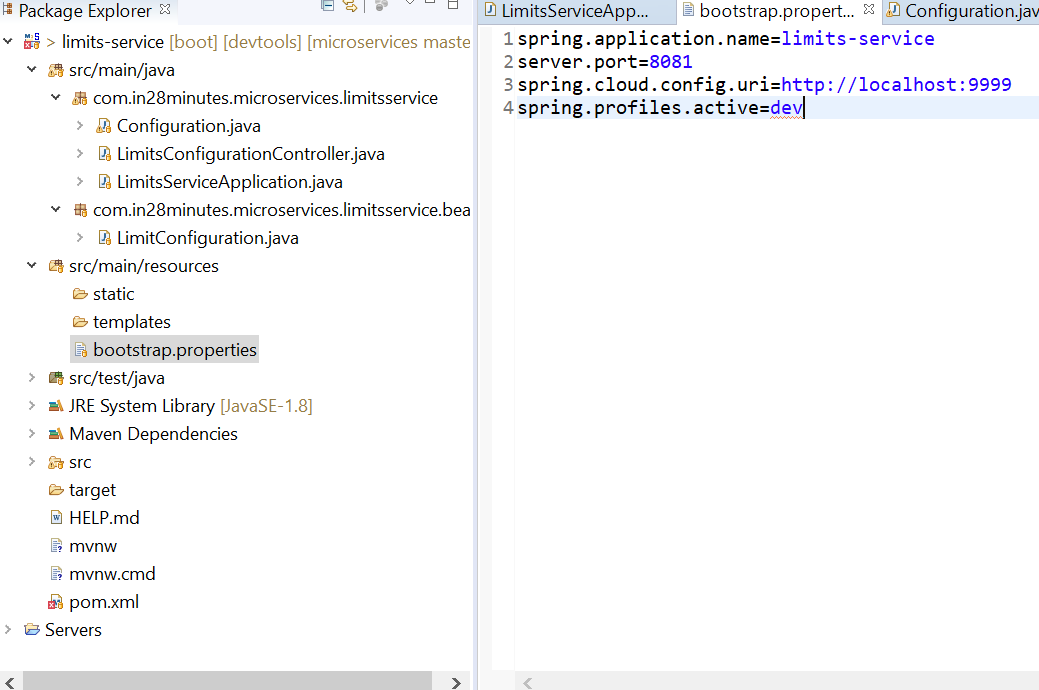


Centralised configuration is needed to handle all these problems



**Video 58:- creation of limits service**

To fetch the minimum and maximum of limit from the config sever for a particular environment



**Video 61**: **Setting up spring config server**

[http://localhost:8888/limits-service/default (watever](http://localhost:8888/limits-service/default%20(watever) is the file name in the git give that in uri and after slash environment to verify)

**{**"name": "limits-service","profiles": **[**"default"**]**,"label": **null**,"version": "a56f245b9dddbc2cf38ef38c5e020041fcb8ad1d","state": **null**,"propertySources": **[{**"name": "file://C:/microservice-git-localconfig-repo/limits-service.properties","source": **{**"limits-service.minimum": "2","limits-service.maximum": "222"**}}]}**

**Video 66: connecting limits service to the spring cloud config server**

spring.application.name=limits-service

server.port=8080

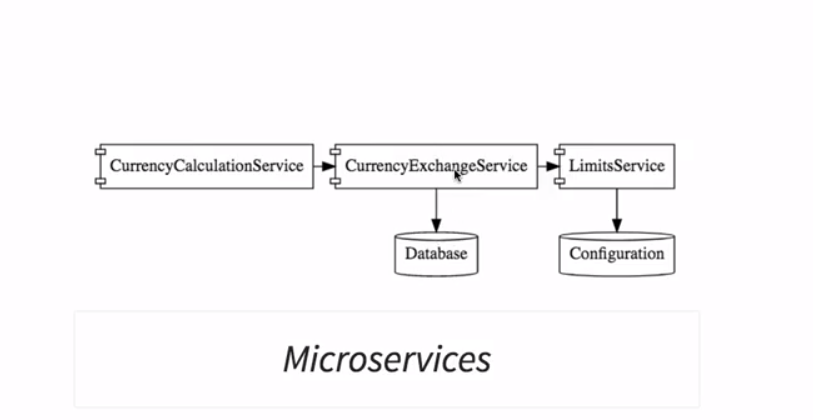
spring.cloud.config.uri=http://localhost:8888

spring.profiles.active=dev

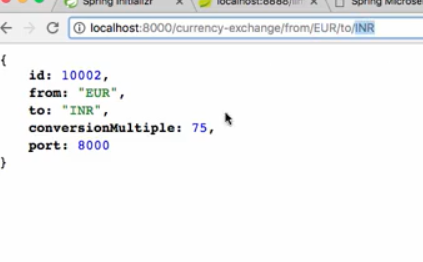
rest: <http://localhost:8080/limits>

response: **{**"maximum": **888**,"minimum": **8}**

**video 69: Introduction**



**Currency exchange service**



**Currency converter(calculation) service**



**Video 70:currency exchange service**

[http://localhost:8000/currency-exchange/from/{from}/to/{to}](http://localhost:8000/currency-exchange/from/%7bfrom%7d/to/%7bto%7d)

<http://localhost:8000/currency-exchange/from/IN/to/DOLLAR>

{

"from": "DOLLAR",

"to": "IN",

"id": 1000,

"conversionMultiple": 65

}

**Video 73: configuring JPA and initialized data**

spring.application.name=currency-exchange-service

server.port=8000

spring.jpa.show-sql=true

spring.h2.console.enabled=true

<http://localhost:8000/h2-console/login.do?jsessionid=93c35df4811cc046751e39427d21102d>

jdbc url: jdbc:h2:mem:testdb

**Video 74: Creating JPA Repository**

**public** **interface** ExchangeValueRepository **extends** JpaRepository<ExchangeValue, Long> {

ExchangeValue findByFromAndTo(String from, String to);

**Video 75 and 76 and 77: setting up currency converter service**

<http://localhost:8100/currency-converter/from/USD/to/INR/quantity/100> {

{

"id": 10001,

"from": "USD",

"to": "INR",

"conversionMultiple": 65,

"quantity": 100,

"totalCalculatedAmount": 6500,

"port": 8000

}

**Rest template**: @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());

}

**Video 78: feign**

<http://localhost:8100/currency-converter-feign/from/USD/to/INR/quantity/100>

->enable feign client

->

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

public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,@PathVariable BigDecimal quantity) {

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

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

}

🡪proxy:

@FeignClient(name="currency-exchange-service",url="localhost:8000")

**public** **interface** CurrencyExchangeServiceProxy {

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

**public** CurrencyConversionBean retrieveExchangeValue(@PathVariable String from,@PathVariable String to) ;

}

**Video 79:client side load balancing with ribbon**

Currency conversion service has one instance but currency exchange service has multiple instances

Make multiple instances of currency exchange service by adding –Dserver.port=8081

In VM arguments

Currency exchange service:

server.port=8100

spring.application.name=currency-conversion-service

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

**proxy class**

@FeignClient(name="currency-exchange-service")

@RibbonClient(name="currency-exchange-service")

**public** **interface** CurrencyExchangeServiceProxy {

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

**public** CurrencyConversionBean retrieveExchangeValue(@PathVariable String from,@PathVariable String to) ;

}

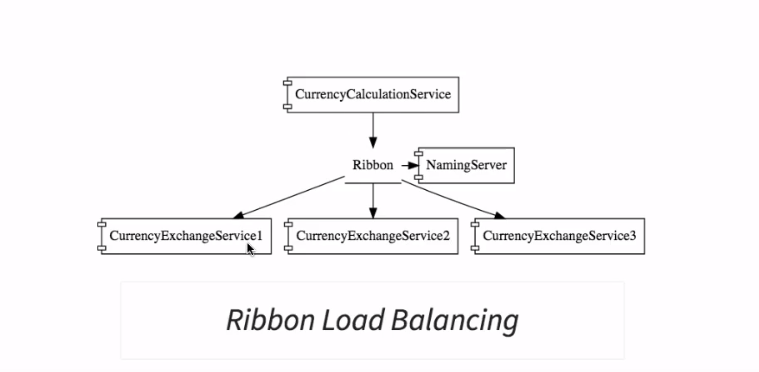
<http://localhost:8100/currency-converter-feign/from/USD/to/INR/quantity/100>

response comes load balanced in round robin manner

**{**"id": **10001**,"from": "USD","to": "INR","conversionMultiple": **65**,"quantity": **100**,"totalCalculatedAmount": **6500**,"port": **8000}**

**{**"id": **10001**,"from": "USD","to": "INR","conversionMultiple": **65**,"quantity": **100**,"totalCalculatedAmount": **6500**,"port": **8001}**

**Video 81-84 Eureka Server**



Use; instead of manually adding the instances in property files it automatically registers.

Server.properties

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

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

and @EnableEurrekaServer

clients.properties

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

and @EnableDiscoveryClient

**video 85-86:Distributing call using eureka and ribbon**

just remove that manual server information of exchange service from the app.properties of the conversion service thhen it will automatically pick the configurations from the eureka servers

**video 87: Introduction to zuul API gateways(proxy)**

🡪authentication and logging

🡪rate Limits

🡪fault tolerance

🡪service aggregation

@SpringBootApplication

@EnableEurekaClient

@EnableZuulProxy

Filter class

**package** com.in28minutes.microservices.netflixzuulapigatewayserver;

**import** javax.servlet.http.HttpServletRequest;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.servlet.support.RequestContext;

**import** com.netflix.zuul.ZuulFilter;

**import** com.netflix.zuul.exception.ZuulException;

@Component

**public** **class** ZuulLoggingFilter **extends** ZuulFilter {

**private** **static** **final** Logger ***log*** = LoggerFactory.*getLogger*(ZuulLoggingFilter.**class**);

@Override

**public** **boolean** shouldFilter() {

// **TODO** Auto-generated method stub

**return** **true**;

}

@Override

**public** Object run() **throws** ZuulException {

// logging filter core logic

HttpServletRequest request = com.netflix.zuul.context.RequestContext.*getCurrentContext*().getRequest();

***log***.info("request->{} request uri->{}",request,request.getRequestURI());

**return** **null**;

}

@Override

**public** String filterType() {

// pre,post or error which kind of filter

**return** "pre";

}

@Override

**public** **int** filterOrder() {

// priority order for multiple filters e.g log, security filter etc

**return** 1;

}

}

**Vid 90: executing request through zull api**

http://localhost:8765/{application-name}/{uri}

<http://localhost:8765/currency-exchange-service/currency-exchange/from/EUR/to/INR>

use the port of zull—then application name to which need to call—then uri of that application

so in currency converter we have modified the internall call to pass thru the zull. We can also make currency conversion to go through zull by using same above standard

**video 52: introduction to distributed tracing(sleuth nd zipkin)**

basically to log or to get info at one place

log sent to queue to be fed by zipkin server

spring cloud sleuth: it gives unique id to a request

**Steps to be done in every application:-**

@Bean

**public** Sampler defaultSampler() {

**return** Sampler.***ALWAYS\_SAMPLE***;

}

<dependency>

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

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

</dependency>

2019-06-26 17:45:12.370 INFO [netflix-zuul-api-gateway-server,9b82e5f0568485fa,d209920d32d35926,true] 12856 --- [nio-8765-exec-5] c.i.m.n.ZuulLoggingFilter : request->org.springframework.cloud.netflix.zuul.filters.pre.Servlet30RequestWrapper@290a26cb request uri->/currency-exchange-service/currency-exchange/from/USD/to/INR

**Vid-93 ZipKin Distributed Tracing**

