

MS

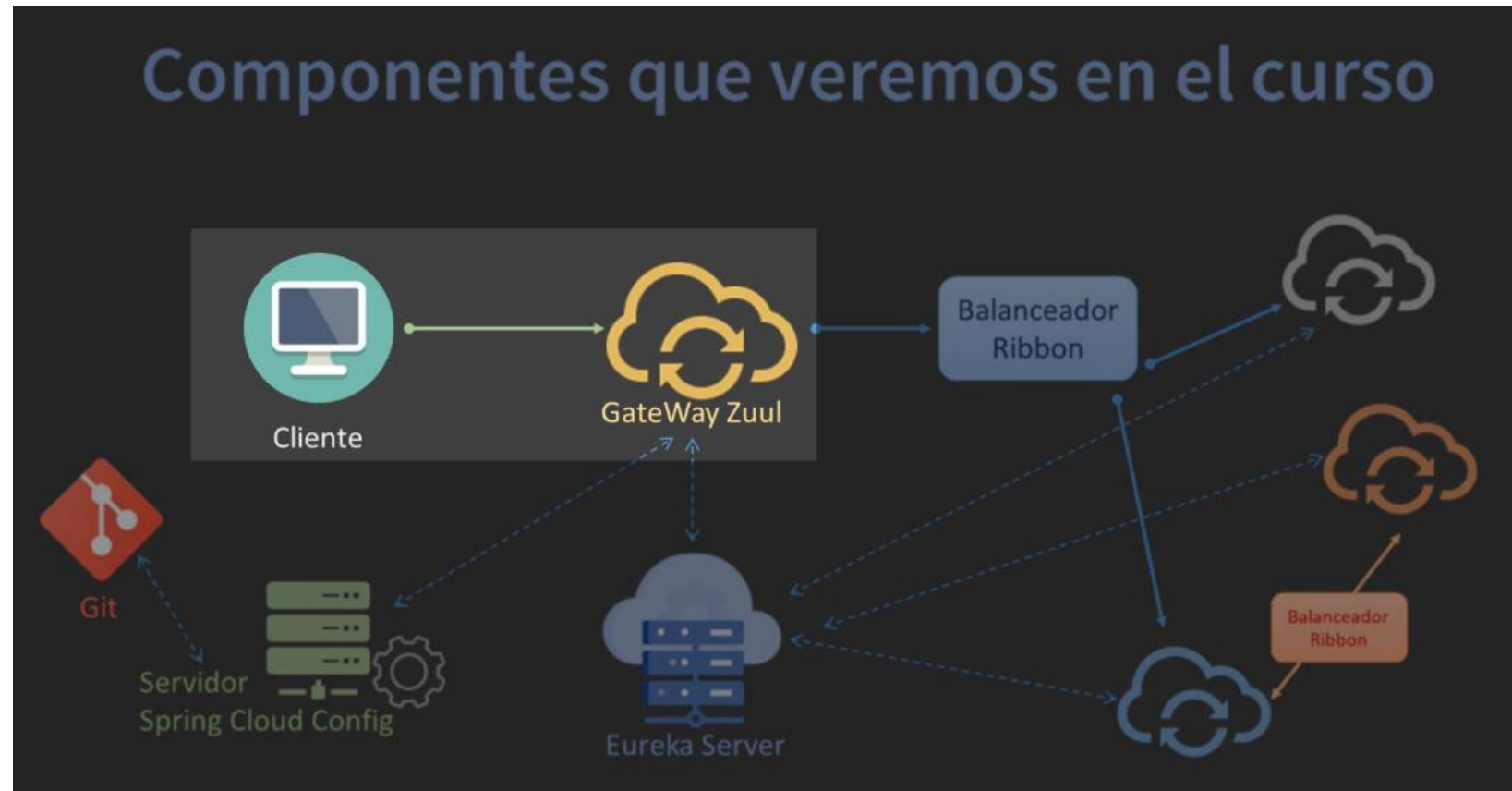
Spring

Microservicios

- Api rest
- Autonomia
- Especializado
- Registro automatico / descubrimiento.
 - registro en servidor de nombre
 - Ip
 - Puerto
 - Nombre de dominio
- Escalabilidad
 - Despliegue de instancias en un puerto diferente.tolerancia a fallos
 - Balanceo de carga

Microservicios

- Spring Boot
- Spring IoC
- Spring Data JPA y Hibernate
- API REST
- Spring Cloud
- Servidor Eureka Netflix
- Eureka Client
- Rest Template
- Feign
- Ribbon / balanceador de carga
- Hystrix
- Gateway Zuul puerta de enlace o entrada.



Ms

- Funcion unica.
- Independientes, despliegue independiente.escalar, monitorear
- Registro y auto descubrimiento de servicios.abstrae la ip y puerto.
 - El servidor de nombres registra los datos del microservicio- **Eureka**
 - Los MS consultan al **servidor de nombre** los datos del MS que va a consultar, pregunta por el nombre y recibe registro con la ubicacion- Autodescubrimiento, desacoplamiento y abstraccion.
 - Auto escalado y agilidad.aumento y decremento de despliegue de instancias.
 - Confiabilidad y tolerancia a fallos, API rest, http, **Hystrix**.
 - Balanceo de cargas. Por nombre el dns nos da las instancias disponibles de un ms y a travez de un algoritmo de balanceo de cargas nos da las instancias disponibles. **Ribbon**
 - Configuracion centralizada. Un MS puede tener muchas instancias corriendo en diferentes entornos (des, pre, prod). **Spring config** centraliza la configuracion. El repositorio de estas configuraciones puede estar en un **GIT o un properties**, en un directorio local.

Ventajas

- Cada aplicacio tiene sus modulos o MS, pueden compartir recursos o no.
- Nueva tecnologia y adopcion de procesos
- Reduccion de costo
- Liberaciones mas rapidas
- Equipos mas pequenos

Repaso

- Feing- cliente http (service discovery)
- Spring core
- Spring boot
- Spring JPA
- Spring Data
- Rest

MS

- Resumen:
 - @service
 - RestTemplate
 - Respository
 - Dao
 - @controller
 - @getmapping
 - @pathvariable
 - @autowired
 - @etity
 - @Table(name="")
 - @Id
 - @GeneratedValue(strategy = GenerationType.IDENTITY)
 - @Column(name = "create_at")
 - @Temporal(TemporalType.DATE)
 - CrudRepository
 - #properties
 - Server.port
 - Spring.applicaion.name
 - Import.sql<- insercion de datos de prueba

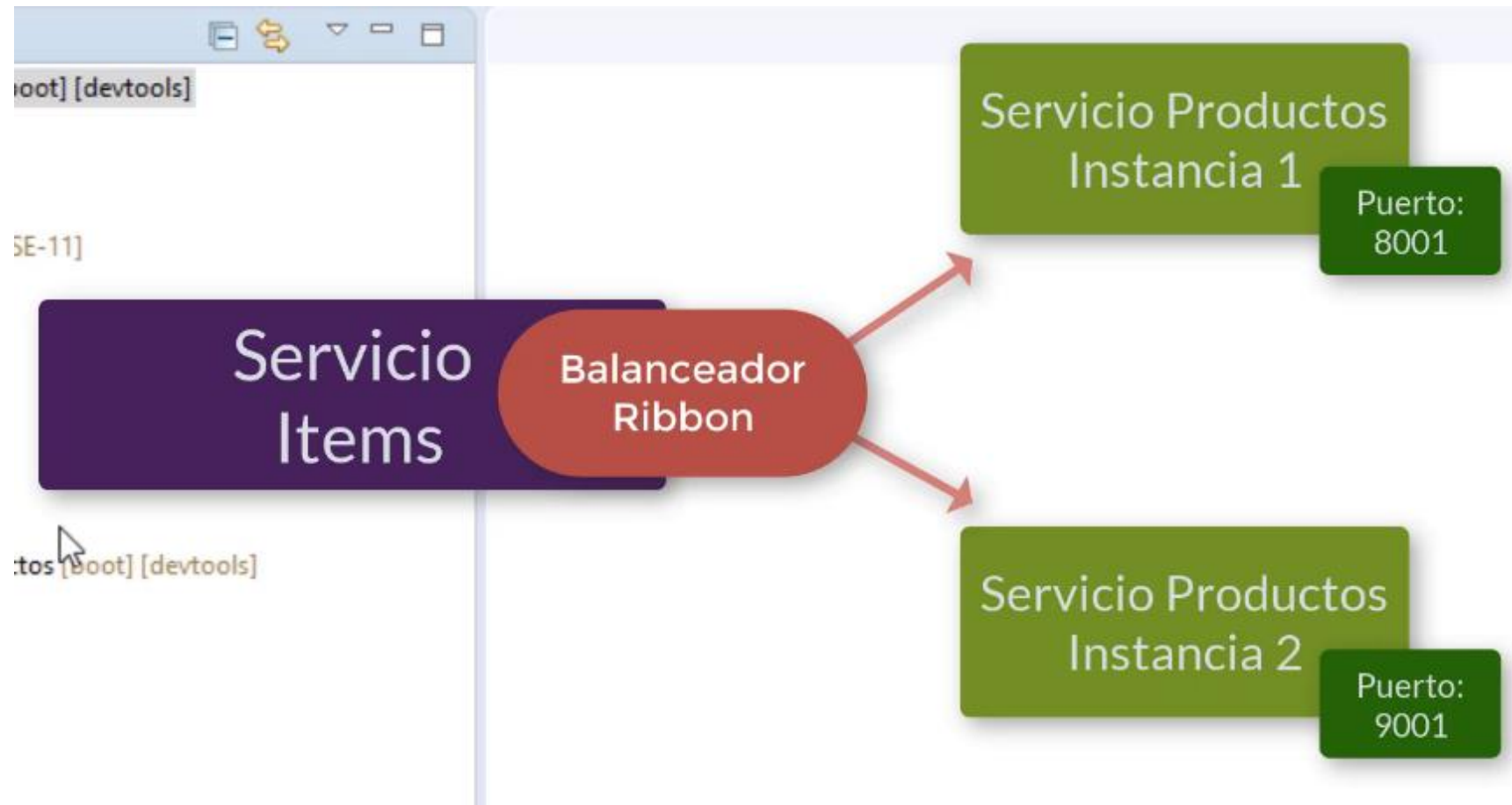
MS

- Resumen, configurando Rest Template
 - AppConfig , clase de configuracion en el paquete base para la inyeccion
 - @Bean
 - RestTemplate

MS

- Feing, http para comunicación entre MS, uso de interfaces, mas simple que restTemplate, es de netflix, por lo que se integra con ribbon, eureka
 - SpringBoot Application
 - @EnableFeingClients
 - Interface Clientes Feing
 - @FeingClient
 - Name = nombre de servicio, url
 - @GetMapping
 - Endpoints del MS que vamos a consumir
 - Interface Service feing Impl
 - Inyeccion de feing client
 - @primary si se tiene más de una alternativa de implementación
 - Controller
 - @qualifier si se tiene más de una implementacion posible para autowired

Balanceo de cargas

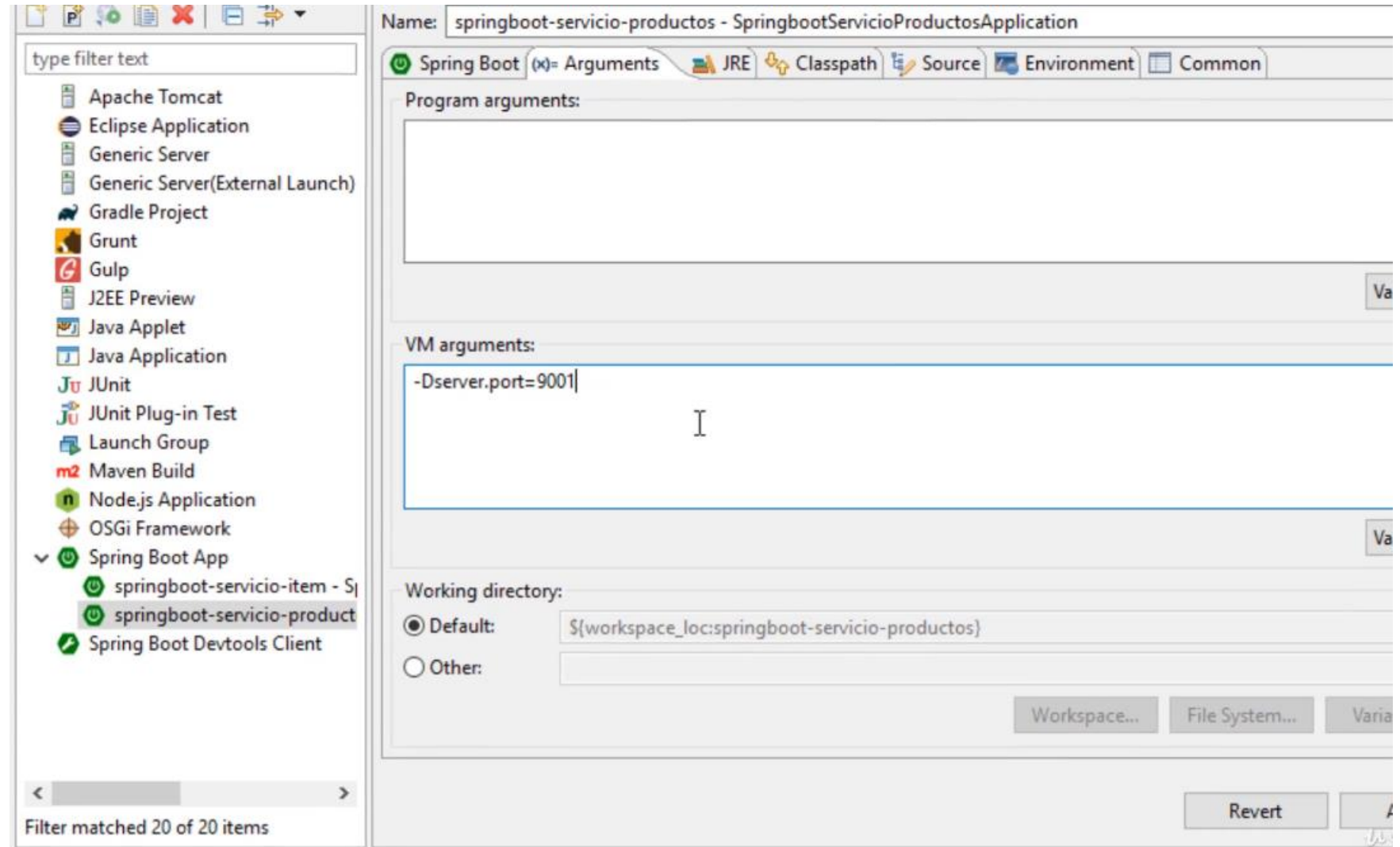


Balanceo de cargas- Feing

- Mas de una instancia disponible y ribbon determina cual es la mejor instancia para responder el request
- Dependencia en pom.xml
- `@RibbonClient(name = "servicio-productos")` Clase configuracion de spring
- Del cliente Feing eliminar la url, pues se configurará mediante properties
 - `@FeignClient(name="servicio-productos",url="localhost:8001")`
- Applicationproperties agregar la configuracion de las instancias donde se va a desplegar los MS que se van a consumir.
 - `Servicio-productos.ribbon.listOfServers=localhost:8001,localhost:9001`
 - Esta configuracion será reemplazada posteriormente por Eureka
- Se deben levantar las dos instancias del MS en los puertos para que ribbon pueda hacer el balanceo de carga
 - De manera manual levantar con el argumento: `arguments->VM arguments : -Dserver.port=9001`

Configuracion de puertos

- Run as



Balanceo de cargas - RestTemplate

- AppspringConfig
 - @RibbonClient(name ="servicio-productos-t")
- AppConfig
 - @configuration
 - @bean -> cliente rest
 - @loadBalanced por defecto usa ribbon para el balanceo
- SeriviceImpl, cambiar la URL por el nombre del servicio
 - <http://servicio-productos-t/listar>
- Applicationproperties
 - servicio-productos-t.ribbon.listOfServers=localhost:8001,localhost:9001

Eureka

- Nuevo proyecto el MS servidor Eureka
- Servidor Eureka, registra los servicios, ip, puerto,
- Los MS se configuran para que al ser desplegados se comuniquen con el servidor Eureka y registren el nombre de la maquina, ip , puerto
- Eureka distribuye la información sobre los MS del ecosistema, sin necesidad de saber informacion de puerto, etc.
- Basata con el nombre de MS, entonces se pueden escalar y comunicar del MS

Eureka server

- Dependencias
 - Devtools
 - Eureka server
- Applicationproperties
 - Spring.application.name
 - spring.application.name=
 - Server.port<- por defecto 8761
 - Eureka.client.register-with-eureka=false
 - Eureka se registra a si mismo como servidor y como cliente.para ello se especifica aquí que no sea así
 - Eureka.client.fetch-registry=false
- Pom.xml
 - Agregar JAX B para versiones de java de 9 en adelante
 - <dependency>
 - <groupId>org.glassfish.jaxb</groupId>
 - <artifactId>jaxb-runtime</artifactId>
 - </dependency>

Eureka Server

- @SpringBootApplication
 - @EnableEurekaServer

Cientes eureka

- En los MS que se quieren registrar en EurekaServer agregar la dependencia `eurekaDiscoveryclient`, para hacerlo cliente.
- Adicional al starter de spring (dependencia pom.xml) se puede agregar la anotación
 - `SpringBootApplication`
 - `@EnableEurekaClient`
 - Application properties
 - `eureka.client.service-url.defaultZone=http://localhost:8761/eureka`
 - Esta configuracion me dice cual es el servisor eureka, es opcional solo cuando los microservicios están en el mismo servidor, si estan en diferentes servidores se debe especificar, puesto que el MS envía un heartbeat, que es la señal de que el MS está habilitado y disponible; provee los datos de la ubicacion del mismo al servidor de Eureka y cada 30 seg sigue enviando esta señal para decir que sigue vivo, pasado ese tiempo si el servidor de eureka no recibe más señal, elimina el registro. Para volverse a activar, se tiene que recibir 3 heartbeat continuos.

Clientes eureka

- Se puede prescindir de la configuracion de ribbon puesto que eureka ya lo incluye.
 - Applicationproperties; Quitar la config de Ribbon de MS item
 - Pom.xml; quitar la dependeincia de ribbon
 - Springboot_application
 - @RibbonClient(name ="servicio-productos-t"), se quita eésta línea pues Eureka contiene ribbon

Escalar MS con puertos dinámicos

- Puertos asignados por spring, en vez de asignar los puertos manualmente.
- Application.properties
 - Server.port=\${PORT:0}
 - Eureka.instance.instance-id=\${spring.application.name}:\${spring.application.instance-id:\${random:value}}Configurar la instancia en Eureka

Tolerancia a fallos y latencia con hystrix

- Manejo de Errores, gestion de comunicación entre MS añadiendola logica de tolerancia a fallos, timeouts, latencia, cuando se alcanza un límite de errores ya no envía más peticiones al MS (se encarga de revisar que el MS esté funcional)
- Gestiona el camino alternativo en caso de que la instancia falle, se llama al callback que sule al MSevitando errores en cascada- patron circuit braker
- Se agrega al MS para especificar que se debe hacer cuando algún método falla o algo adicional pasa

Hystrix error conf

- Pom.xml
 - Agregar dependencia
- SpringBootApplication
 - `@EnableCircuitBreaker` – manejo de errores envuelve ribbon.
- Controller en el metodo donde puede ocurrir el error
 - `@HystrixCommand(fallbackMethod="metodoAlternativo")`- se indica el metodo que se debe ejecutar en caso de error

Timeout con hystrix y ribbon

- Para ocasionar el time out dormir el hilo en el controlador
- El tiempo por defecto en ribbon y hystrix para timeout es de 1 segundo a menos que se configure diferente
- Hystrix lanza una exception o bien si se tiene el fallbackmethod ejecuta el camino alternativo
- Tener un timeout tan bajo no es siempre la mejor alternativa, por lo que conviene configurarlo por ejemplo en casos de un upload de archivos.
 - `hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=11000`
 - `ribbon.ConnectTimeout:3000`
 - `ribbon.ReadTimeout:7000`
 - Recordar que Hystrix envuelve a ribbon por lo que se recomienda que su configuración sea mayor que la de ribbon.

Zuul API Gateway

- Se encarga del acceso a los demás MS registrados en Eureka
- Puerta de acceso, o de enlace, a las instancias y MS
- Enrutamiento dinámico
- Se integra con ribbon para balanceo de carga
- Filtros
 - Seguridad – en vez de spring security
 - Monitoreo
 - Errores
 - Authentication
 - Dynamic Routing
 - Service Migration
 - Load Shedding
 - Security
 - Static Response handling
 - Active/Active traffic management

Zuul appi gateway

- Dependencias
 - Eureka client
 - Devtools
 - Web
 - Zuul
- SpringBootApplication
 - @Enable Eureka Client
 - @EnableZuulProxy
- Application.properties
- zuul.routes.productos.service-id=servicio-productos-t
 - zuul.routes.productos.path=/api/productos/**
 - zuul.routes.productos.service-id=servicio-item-t
 - zuul.routes.productos.path=/api/items/**

Zuul filtro HTTP

- Ruteo dinámico
- Filtros
 - Pre -> antes de que se resuelva la ruta y el request sea enrutado, se usa para pasar datos al request. Para su uso en filtros posteriores como en los de la ruta.
 - Post -> se invoca justo despues del que el reques ha sido enrutado, se usa para modificar la respuesta. Cabeceras
 - Route -> Enruta elrequest, resuelve la ruta hacia el MS y se usa para la comunicación con el MS.
 - Ribbon router filter
 - Hystrix
 - Http (apache Http client)

Zuul filtro HTTP - pre

- Calculo de tiempo transcurrido en la comunicación con el MS es decir en el request
 - Filtro Pre para calculo de la hora inicial del request es decir antes de iniciar el enrutamiento y la comunicación con el MS
 - Filtro Post calcula el tiempo final de en que termina el enrutamiento es el tiempo de post- pre
- Los filtros implementaciones de zuul filter
 - @Component
 - Extends zuulFilter
 - FilterType(){ return "pre";...
 - FilterOrder(){ return 1; ...
 - ShouldFilter(){ // validaciones para en caso de true ejecutar el filtro , existe parametro ...
 - Run(){ // logica filtro





Zuul filtro HTTP - post

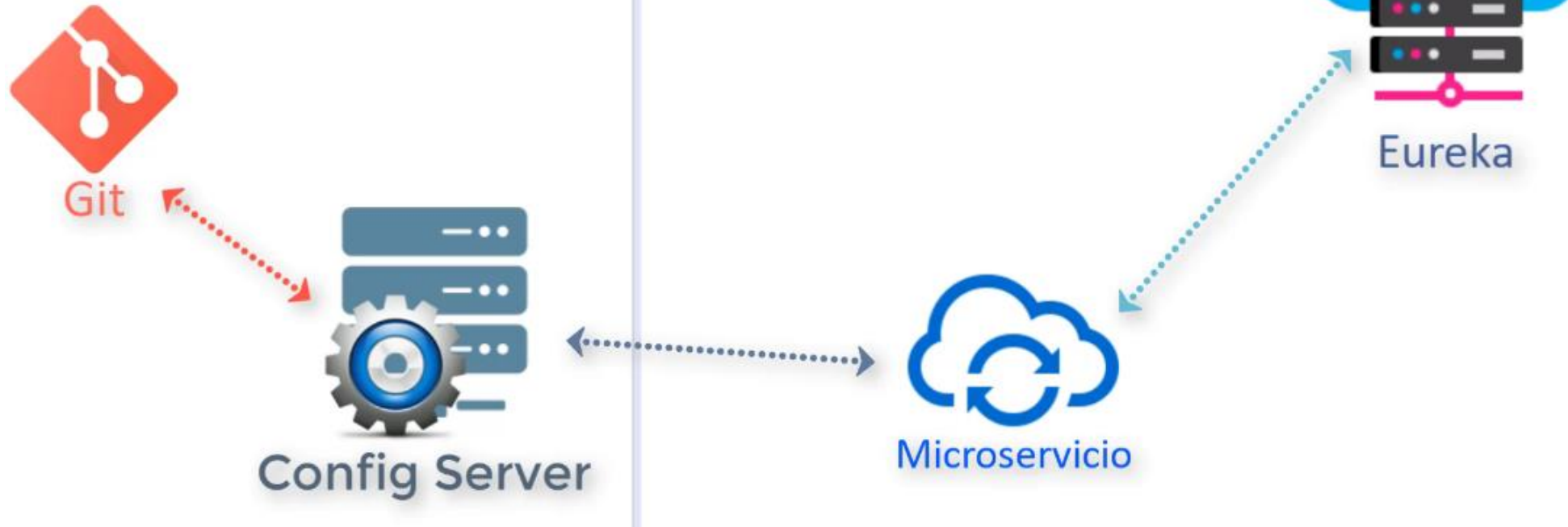
- Calcular el tiempo final, obtener el tiempo inicial y calcular la diferencia entre ambos, para así conocer el tiempo total que tarda en procesarse el request y se da la comunicación con el MS

Zuul timeout config

- `hystrix.commad.default.execution.isolation.thread.timeoutInMilliseconds=40000`
- `ribbon.ConnectTimeout:5000`
- `ribbon.ReadTimeout:15000`

Spring Cloud Config Server

- >  springboot-servicio-eureka-server
- >  springboot-servicio-item
- >  springboot-servicio-productos
- >  springboot-servicio-zuul-server



Spring Cloud Config Server

- MS que centraliza la configuración de todos los MS
- Independiente de la arquitectura, entorno...
- Repo GIT
- El MS al arrancar consulta al MS de Configuración, para saber todos los parámetros de configuración, una vez obtenido se registra en eureka y arranca
- Se puede tener una configuración, por ambiente, por perfil ...

Spring Cloud Config

- Pom.xml
 - Spring cloud config server
- Springboot Application
 - @EnableConfigServer
- Application.properties
 - spring.application.name=config-server
 - server.port=8888
 - #configuracion GIT
 - spring.cloud.config.server.git.uri=
- Repo config git
 - Un archivo nombre-servicio.properties con las configuraciones propias de cada MS

Config client

- Dependencias
 - Cloud config client
- Application properties
 - Copiar y renombrar como bootstrap.properties
- Bootstrap.properties
 - Config el servidor de config, nombre de Ms , url del servidor de configuración.
 - spring.application.name=servicio-item-t
 - spring.cloud.config.uri=h
- Configuraciones de DB, perfiles, texto, puerto, ruta de eureka ...
- Primero se arranca bootstrap y luego application, bootstrap tiene mayor prioridad que application. Se configura todo lo que tnga que ver con Spring y los ambientes o profiles
- Nombre de archivo {nombre MS}-{profile}.properties

Acceder a la configuracion desde el controlador

- @value

Profiles

- Agregar un archivo `servicio-profile.properties` al repositorio git para cada profile
- En el MS al archivo `bootstrap.properties` agregar
 - `Spring profiles.active=`

Refrescar actualizaciones configuracion

- Tomar las actualizaciones del repositorio GIT sin reiniciar el microservicio.
- Los controladores de configuracion implementan singleton, por ello la carga de la config se realiza una vez al iniciar.
- Para actualizar se debe hacer refresh
 - @RefreshScope
 - Spring Actuator->indica que el componente se va a poder actualizar. Expone un endpoint para refrescar los componenets anotados con@RefreshScope
 - Se deben anotar los componentes que estén consumiendo parámetros del archivo de configuracion, ejemplo @Value
 - Pom.xml, agregar la dependeincia Spring Actuator
 - management.endpoints.web.exposure.include=*
 - Solo se pueden modificar confiuraciones propias del MS, es decir, configuraciones de spring, DB, puertos, no se pueden reconfigurar, para eso si se debe reiniciar el MS.

Repositorio remoto

- En el repo local
 - Git remote add origin [https://github](https://github.com)....
 - Git push -u origin master
- En el MS
 - Modificar la uri
 - credenciales

CRUD

- Service
 - @Transactional
- Controller
 - HTTP
 - @Put
 - @Get
 - @Post
 - @Delete
 - @Response
 - @Request
 - Response entity

Librería commons

- Repositorios de objetos comunes o bien jar de utilería
- Proyecto nuevo
 - Eliminar el metodo de arranque de springboot application
 - Crear los pojos comunes
- POm.xml
 - Eliminar plugin maven
- Configurar Java Home y path en el SO
 - /etc/profile
 - Etc/environment
- Jar en targetcd
- Al usar JPs si queremos quitar la dependencia a la DB en controller
 - `@EnableAutoconfiguration(exclude = {DataSourceAutoConfiguration.class})`
 - **mvn install -DskipTests**

Librería commons

- `<groupId>com.besfactory.demoapp.commons</groupId>`
- `<artifactId>uds-servicio-commons</artifactId>`
- `<version>0.0.1-SNAPSHOT</version>`
- En el Pom del Ms producto pegar
- Cambiar las dependencias
- Agregar Entity scan a springbootapplication de productos para que busque en el otro package
 - `@EntityScan({"com.besfactory.demoapp.commons.model.entity"}),` los packages se separan por coma