# **Service Registry using eureka**

Service registry is the service which provide load balancing for the different services.

It creates instances for then services.

**Inside service registry project**

* Add depency in the pom

<dependency>

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

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

* Add eureka service configuration in the application starting point

@SpringBootApplication

@EnableEurekaServer

**public** **class** ServiceRegistryApplication {

* Inside application.yaml file

spring:

application:

name: service-registry

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

register-with-eureka: false --stops itself from registry to itself

fetch-registry: false --stop it from register itself

service-url:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/ -destination for registration

# **Config Service for all configuration files management**

* Add dependency in pom file

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

* Enable config server in application starting point

@SpringBootApplication

@EnableConfigServer

public class ConfigServerApplication {

* Inside application.yaml file

server:

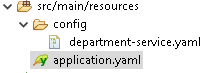
port: 8088

spring:

profiles:

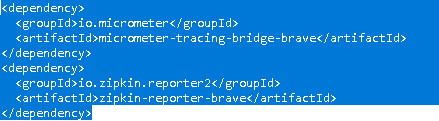
active: native --Makes it applicable for all the application registered to it matching the same name config file in the config folder

note- config folder is name specific folder in the same location as yaml file



* Add different service property or yaml files in the config folder
* Note- property file names should be same as the service name

# **First service department service**

* Add dependencies
  + Actuator for monitoring
    - 
  + Eureka client for service registry connection
    - 
  + Config service to connect with config service
    - Config client
    - 
  + Add Zipkin configuration for log management
    - 
    - By Default connects to port 9411
* Add inside main application
  + 
  + Discovery client for discovering the client from service discovery
* Inside property or yaml file
  + spring:
  + application:
  + name: department-service --service name should be same config server aml file
  + config:
  + import: optional:configserver:http://localhost:8088 --location to find the config properties
* Inside the config side stored property file

server:

port: 8081 --sets the port to 8081

eureka:

client:

service-url:

defaultZone: <http://localhost:8761/eureka/> --tells eureka client where to connect

management:

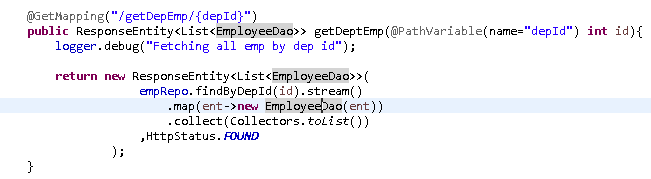
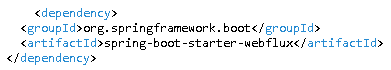
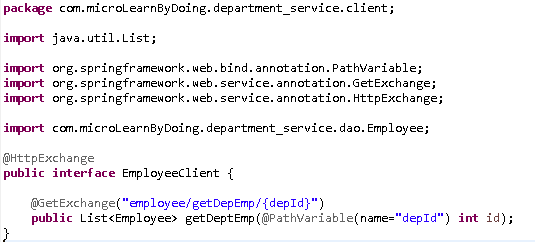
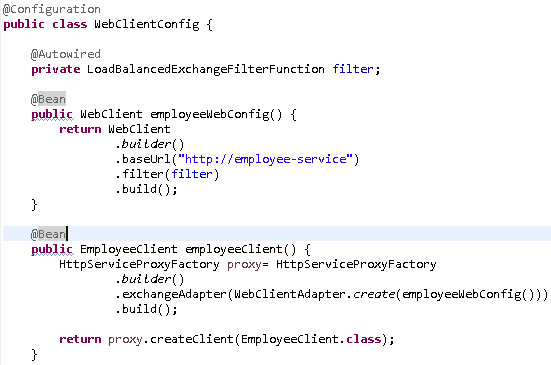
tracing:

sampling:

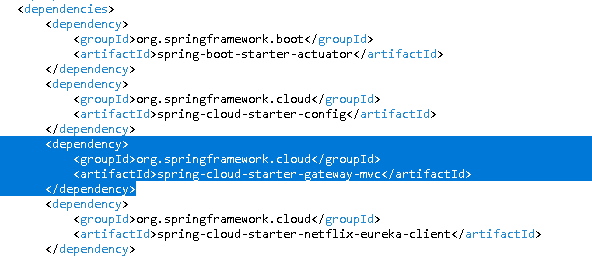
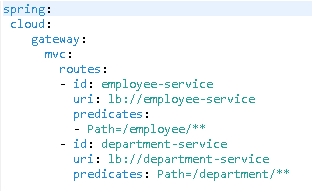
probability: 1.0 --sets probability of logs to be shared with zipkin

Note – 1.0 means 100 %

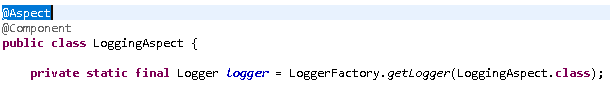
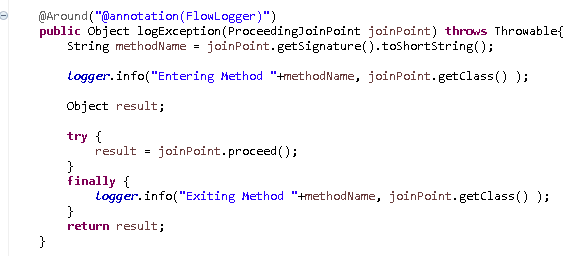
# **WebFlux using reactive web client to call employee service controller from department service**

* Inside Employee controller
  + 
* Add Webflux in the pom file of department service(From where the reactive call to be made)
  + 
* Create a client for Http Exchange
  + 
  + Create get Exchange with the end point of the service api to be called.
* Configure above client for the webclient
  + 
  + Create a webClient for the target service i.e- api call name for the service registry
  + Then use the created web client to create the target client proxy.
* Use the created client for calling the target service api
  + 

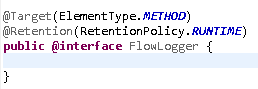
# **API GateWay**

* Create a new project for api gate way with below maven dependencies
  + 
  + Here gateway is specific for the Api Gateway
* Add Routing Properties in the property or yaml file
  + 
  + Here each id is specific to service name
  + Each uri is the new load balancing service point using service registry(i.e. name is specific to service registry)
  + Predicates define the path need to be mapped to the specific route

# **Aspect oriented prog(AOP) and custom annotation**

* Create Custom aspect class using @Aspect annotation
  + 
* Create a @Around aspect pointing to custom annotation
  + 
  + Here @annotation(FlowLogger) connects with custom annotation FlowLogger

Creation of Custom Annotation

* Create a class for custom annotation
  + 
  + Here @Target sets what type of anotation it is (like class method )
  + @Retention sets when it should run

# **Kafka via Docker**

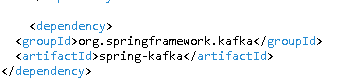
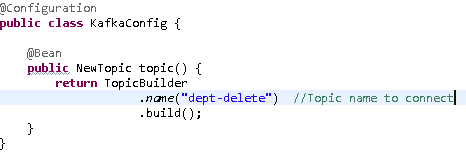
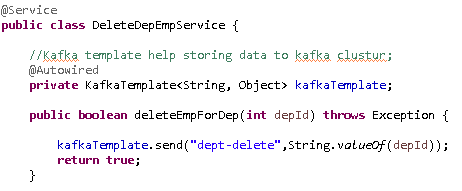
docker run -d -p 9092:9092 --name broker apache/kafka:latest

open kafka terminal

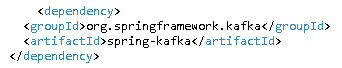
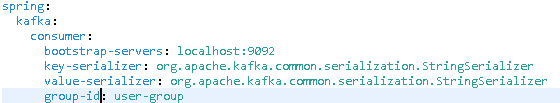
docker exec --workdir /opt/kafka/bin/ -it kafka sh

* To run zookeeper in docker
  + ./zookeeper-server-start.sh /opt/kafka/config/zookeeper.properties
* Run Kafka Server(Not for docker)
  + .\bin\windows\kafka-server-start.bat config\server.properties
* Create a new Topic
  + bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server localhost:9092
* Get All Topic
  + .\bin\windows\kafka-topics.bat --list --bootstrap-server localhost:9092
* Describe a Topic
  + bin/kafka-topics.sh --describe --topic quickstart-events --bootstrap-server localhost:9092
* Write Event on topic as a producer
  + bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server localhost:9092
* Read event as a consumer
  + bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning --bootstrap-server localhost:9092

## **Creating Kafka Producer in Department service**

* + Adding Kafka dependency in pom file
    - 
  + Configuration for kafka inside property/yaml file
    - 
  + Configuration in the config file(Create a config class for configuration)
    - 
    - Register Topic
  + Use kafka template to produce/send message
    - 
    - Topic name + msg data

## **Kafka consumer in Employee service**

* Kafka dependency inside pom file
  + 
* Kafka properties in yaml file
  + 
  + Added group id inside the consumer
* Creating a kafka listener for topic and group id
  + 