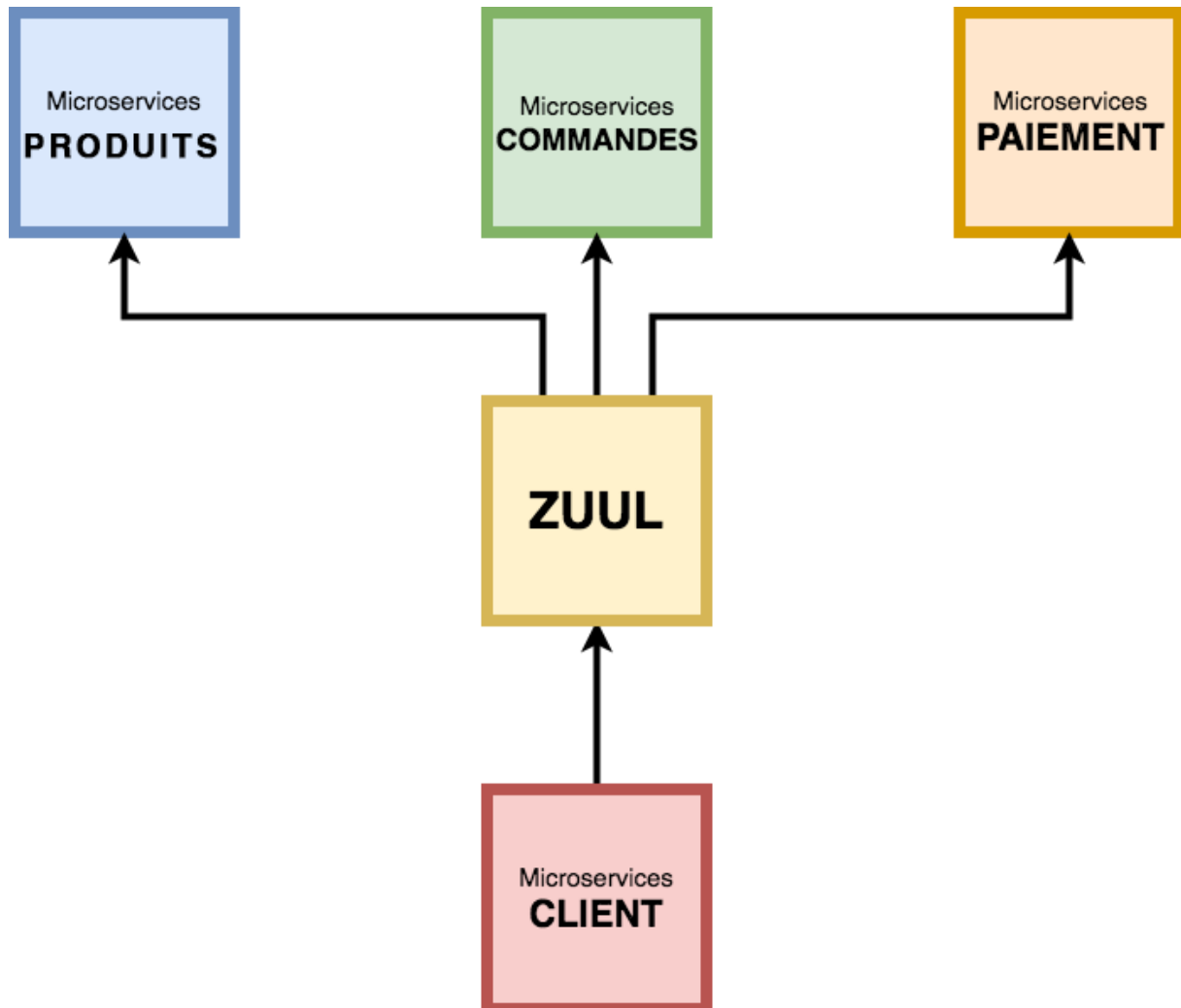


Atelier 7

Créer une API Gateway de l'application via Zuul

ZUUL va se positionner comme le point d'entrée unique de notre application. Ainsi, quand ClientUi, par exemple, voudra appeler les Microservices, il passera par **ZUUL**. Ce dernier s'occupe de dispatcher, modifier et appliquer des filtres et des règles à ces requêtes.



1. Ajoutez ZUUL à l'application

Rendez-vous sur Spring Initializr et choisissez les composants suivants :

Application	Addr	Availability Zones	Status
MICROSERVICE-CIENTUI	n/a (1)	(1)	UP (1) - 192.168.0.10:microservice-clientui:8080
MICROSERVICE-COMMANDES	n/a (1)	(1)	UP (1) - 192.168.0.10:microservice-commandes:9002
MICROSERVICE-PRODUITS	n/a (2)	(2)	UP (2) - 192.168.0.10:microservice-produits:9011 , 192.168.0.10:microservice-produits:9001
ZUUL-SERVER	n/a (1)	(1)	UP (1) - 192.168.0.10:zuul-server:9004

Instances currently registered with Eureka

ZUUL fonctionne nativement avec Eureka. Il récupère la liste de tous les Microservices disponibles dans Eureka et les expose via l'URL : localhost:9004/nom-du-microservice.

Ainsi, pour récupérer la liste des produits, il suffit d'appeler : localhost:9004/microservice-produits/Produits.

Vous récupérez alors la liste des produits comme si vous aviez appelé Microservice-produits directement.

2. Les filtres

Les filtres sont une des fonctionnalités les plus importantes de ZUUL. Quand un client appelle ZUUL, celui-ci vous offre la possibilité d'appliquer un filtre à cette requête avant de la passer au Microservice concerné.

Dans votre filtre, vous pouvez **manipuler, modifier ou adapter une requête selon les besoins**. Par exemple, vous pouvez faire un contrôle de sécurité afin de vérifier que le client a bien le droit d'appeler tel ou tel Microservice.

Créons alors un **premier filtre ZUUL**. Ce filtre va tout simplement utiliser un logger SL4j afin d'afficher un message dans la console.

Créez une classe, appelez-la **LogFilter** et mettez-la dans un package "**filters**" :

```
package com.mcommerce.zuulserver.filters;

import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.exception.ZuulException;
import org.springframework.stereotype.Component;

@Component
public class LogFilter extends ZuulFilter {

    @Override
    public String filterType() {
        return null;
    }

    @Override
    public int filterOrder() {
        return 0;
    }

    @Override
    public boolean shouldFilter() {
        return false;
    }
}
```

```
@Override
public Object run() throws ZuulException {
    return null;
}
}
```

Explications :

- Un filtre ZUUL est simplement une classe qui hérite de `ZuulFilter`. Vous devez implémenter les 4 méthodes obligatoires.
- `filterType` : cette méthode sert à déterminer le type de filtre à appliquer, elle propose 4 possibilités :
pre : permet d'exécuter du code avant la redirection de la requête vers sa destination finale.
post : permet d'exécuter du code après que la requête a été redirigée.
route : permet d'agir sur la façon de rediriger les requêtes.
error : permet d'agir en cas d'erreur lors de la redirection de la requête.
- `filterOrder` : dans votre API Gateway ZUUL, vous aurez forcément des dizaines de filtres. Cette méthode détermine l'ordre d'exécution de ce filtre.
- `shouldFilter` : permet d'écrire les conditions qui doivent être remplies pour que le filtre s'exécute.
- `run` : c'est ici que va la logique de votre filtre.

Modifiez alors le filtre afin de déterminer toutes ces options :

```
package com.mcommerce.zuulserver.filters;

import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.context.RequestContext;
import com.netflix.zuul.exception.ZuulException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

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

@Component
public class LogFilter extends ZuulFilter {

    Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public String filterType() {
        return "pre";
    }

    @Override
    public int filterOrder() {
        return 1;
    }

    @Override
    public boolean shouldFilter() {
        return true;
    }

    @Override
    public Object run() throws ZuulException {
```

```

HttpServletRequest req = RequestContext.getCurrentContext().getRequest();

log.info("**** Requête interceptée ! L'URL est : {} " , req.getRequestURL());

return null;
}
}

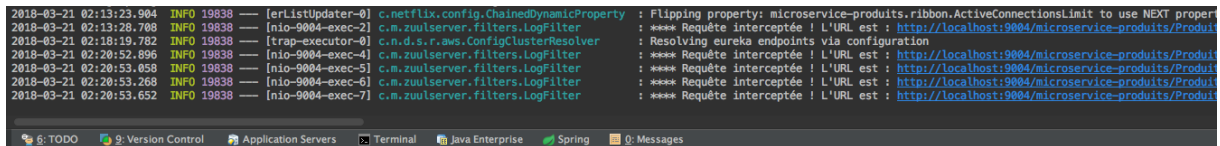
```

Explications :

- Afin de logger les requêtes reçues, on crée une instance du logger de SL4J.
- On retourne "pre" de `filterType` afin d'afficher notre message avant que la requête ne soit redirigée.
- On retourne true directement dans `shouldFilter` afin d'exécuter ce filtre sur toutes les requêtes sans conditions.
- `run()` : On récupère la requête grâce à `RequestContext` qui est utilisé par les filtres dans ZUUL afin de manipuler les requêtes en attendant leur redirection. On affiche ensuite un message avec l'URL de la requête reçue.

Lancez ZUUL et tous les Microservices, puis rendez-vous par exemple à `localhost:9004/microservice-produits/Produits` afin de tester ZUUL.

Vous devriez voir votre **message de log** s'afficher dans la console de ZUUL dans IntelliJ à chaque nouvelle requête :



```

2018-03-21 02:13:23.904 INFO 19838 [erListUpdater-0] c.netflix.config.ChainedDynamicProperty : Flipping property: microservice-produits.ribbon.ActiveConnectionsLimit to use NEXT proper
2018-03-21 02:13:28.708 INFO 19838 [nio-9004-exec-2] c.m.zuulserver.filters.LogFilter : **** Requête interceptée ! L'URL est : http://localhost:9004/microservice-produits/Produits
2018-03-21 02:18:19.782 INFO 19838 [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2018-03-21 02:20:52.896 INFO 19838 [nio-9004-exec-4] c.m.zuulserver.filters.LogFilter : **** Requête interceptée ! L'URL est : http://localhost:9004/microservice-produits/Produits
2018-03-21 02:20:53.050 INFO 19838 [nio-9004-exec-5] c.m.zuulserver.filters.LogFilter : **** Requête interceptée ! L'URL est : http://localhost:9004/microservice-produits/Produits
2018-03-21 02:20:53.268 INFO 19838 [nio-9004-exec-6] c.m.zuulserver.filters.LogFilter : **** Requête interceptée ! L'URL est : http://localhost:9004/microservice-produits/Produits
2018-03-21 02:20:53.652 INFO 19838 [nio-9004-exec-7] c.m.zuulserver.filters.LogFilter : **** Requête interceptée ! L'URL est : http://localhost:9004/microservice-produits/Produits

```

Vous pouvez également créer un autre filtre qui s'exécute à la réponse de la requête, par exemple :

```

package com.mcommerce.zuulserver.filters;

import com.netflix.zuul.ZuulFilter;
import com.netflix.zuul.context.RequestContext;
import com.netflix.zuul.exception.ZuulException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Component
public class ReponseFilter extends ZuulFilter {

    Logger log = LoggerFactory.getLogger(this.getClass());

    @Override
    public String filterType() {
        return "post";
    }

    @Override

```

```

public int filterOrder() {
    return 1;
}

@Override
public boolean shouldFilter() {
    return true;
}

@Override
public Object run() throws ZuulException {

    HttpServletResponse response = RequestContext.getCurrentContext().getResponse();

    response.setStatus(400);

    log.info(" CODE HTTP {}", response.getStatus());

    return null;
}
}

```

Ce filtre récupère toutes les réponses et change le code en 400. Redémarrez ZUUL et envoyez via Postman des requêtes vers n'importe quel Microservice ; vous recevrez invariablement un code HTTP 400 Bad Request.

N'oubliez pas de désactiver ce filtre (shouldFilter à false) pour qu'il ne vous bloque pas pour le reste du Lab.

3. Connectez le client à ZUUL

Nous avons mis en place un filtre qui intercepte les requêtes pour les logger avant de les rediriger vers leur destination finale. Nous souhaitons à présent que **ClientUI** puisse faire appel aux différents Microservice via ZUUL.

Afin que notre client passe par ZUUL, nous devons indiquer aux proxy Feign qu'il faut contacter ZUUL et non les Microservices directement.

Remplacez alors le nom du Microservice de destination par celui de ZUUL :

```

@FeignClient(name = "zuul-server")
@RibbonClient(name = "microservice-produits")
public interface MicroserviceProduitsProxy {

    @GetMapping(value = "/microservice-produits/Produits")
    List<ProductBean> listeDesProduits();

    /*
     * Notez ici la notation @PathVariable("id") qui est différente de celle qu'on utilise
     * dans le contrôleur
     */
    @GetMapping( value = "/microservice-produits/Produits/{id}")
    ProductBean recupererUnProduit(@PathVariable("id") int id);
}

```

On ajoute */microservice-produits/* devant toutes les URI. Feign ira alors contacter ZUUL avec l'URI *"/microservice-produits/Produits"*. Comme ZUUL extrait alors le nom du Microservice concerné de l'URL *"microservice-produits"*, il pourra rediriger la requête vers la destination voulue.

Redémarrez le client et testez. Vous devriez voir, dans la console de ZUUL, le log de toutes les requêtes et la liste des produits s'afficher.

Faites de même avec tous les proxy et votre application devrait fonctionner normalement tout en passant par ZUUL.

La branche pour ce chapitre est **ZUUL**.