**Tutoriel développer des Micro-Services avec Spring Boot et Spring Cloud dans l'IDE ECLIPSE**

Rédigé le : 20/10/2020

Table des matières

[1 Micro-Service de CONFIGURATION 2](#_Toc54114006)

[1.1 Développement du Micro-Service de CONFIGURATION CENTRALISEE 2](#_Toc54114007)

[1.1.1 Créer un nouveau projet SPRING-STARTER-PROJECT dans l'IDE 2](#_Toc54114008)

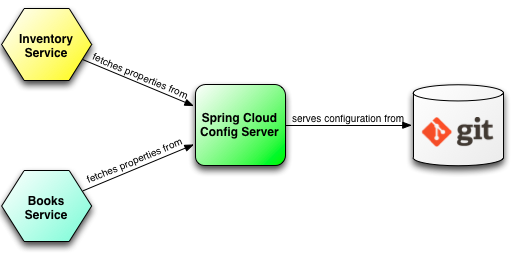
[1.2 Test du Micro-Service de CONFIGURATION CENTRALISEE 2](#_Toc54114009)

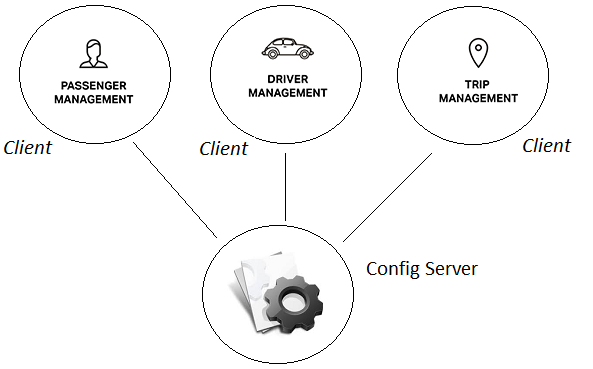
[1.3 Vérifier la couverture des tests du Micro-Service de CONFIGURATION CENTRALISEE 3](#_Toc54114010)

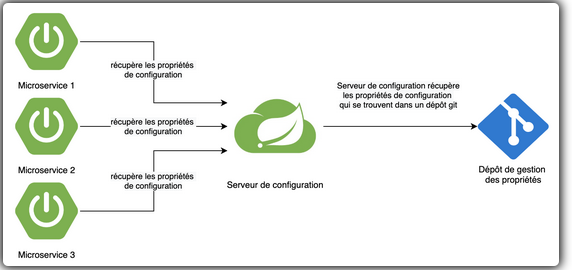
# Micro-Service TECHNIQUE de CONFIGURATION centralisée externalisée

**Objectif** : créer un micro-service TECHNIQUE **UNIQUE** chargé de gérer *toutes* les configurations de *tous* les micro-services utilisés par une application Web.

## Développement du Micro-Service de CONFIGURATION CENTRALISEE (Spring Cloud Config Server)







### Créer un nouveau projet SPRING-STARTER-PROJECT dans l'IDE

**Objectif** : Créer un nouveau projet SPRING-STARTER-PROJECT dans l'IDE ECLIPSE pour développer un Micro-Service de CONFIGURATION externalisée.

1. **Créer un nouveau SPRING STARTER PROJECT** dans l'IDE (ECLIPSE).

*MAVEN est automatiquement embarqué dans le projet ECLIPSE ainsi que le Conteneur WEB TOMCAT*.

**Le package choisi doit être le même quelque soit le micro-service** si on veut pouvoir créer une application *monolithique* en faisant juste glisser les packages ou classes provenant de différents projets ECLIPSE sous un explorateur Windows.

Je mets *systématiquement* levy.daniel.application à la fois comme groupId Maven et comme package Maven pour tous les micro-services que je développe afin de pouvoir facilement faire glisser des classes ou package d'un projet à un autre..

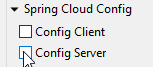
* Le groupId Maven indiqué dans le pom.xml du projet ECLIPSE MAVEN sert à **nommer les classes buildées par Maven (espace de nommage)**. Par exemple, si on met un groupId Maven = "toto" et que l'on crée une classe 'Exemple.java', le nom complet (aussi appelé nom qualifié en Java) de la classe buildée sera toto.Example.class.
* Le Package Maven indiqué dans le pom.xml du projet ECLIPSE sert à **créer automatiquement les packages des classes *sources*** .java puis ***binaires*** .class dans l'IDE lors de la génération automatique des packages dans le projet ECLIPSE par Maven.
* Par exemple, l'IDE ECLIPSE créera automatiquement les packages : src.main.java.levy.daniel.application et src.test.java.levy.daniel.application

Dans le projet dans l'IDE ECLIPSE si package = levy.daniel.application.

### Choisir les dépendances SPRING CLOUD d'un Micro-Service de Configuration

**Objectif** : **insérer dans le pom.xml** du projet MAVEN en cours de création les **dépendances** indispensables au micro-service de centralisation des configurations externalisées.

* Dans Spring Cloud Config > **Config Server**



* Dans Spring Cloud Discovery > **Eureka Discovery Client**



#### Tableau résumé des dépendances du micro-service de configuration

|  | **Groupe de dépendances Spring Boot + Cloud** | **Dépendance** |
| --- | --- | --- |
| 1 | **Spring Cloud Config** | **Config Server** |
| 2 | Spring Cloud Discovery (Netflix EUREKA) | Eureka Discovery Client |
| 3 | Developer Tools | Spring Boot Dev Tools |
| 4 | Web | Spring Web |
|  |  |  |

### Rajouter l'annotation @EnableConfigServer sur la classe applicative Spring Boot du micro-service de configuration

**Objectif** : indiquer à SPRING que le micro-service de CONFIGURATION joue un rôle de serveur de configuration pour tous les micro-services enregistrés auprès de lui.



### Créer un dossier GIT ou GITHUB Externe conteneur des configurations à servir

**Objectif** : L'intérêt du Micro-Service de Configuration est d'*externaliser les configurations* de tous les micro-services utilisés dans une application Web **dans un répertoire *distant***.

Ce répertoire distant situé *hors du classpath* sera donc librement accessible par l'hébergeur (centre-serveur) qui pourra venir y changer des valeurs au moment du *déploiement*.

On peut avantageusement utiliser un **repository GIT ou GITHUB externe** distant contenant les configurations afin de tracker toute modification sur un fichier de configuration.

1. **Créer un dossier** (par exemple 'configurations\_traficweb')
2. **Mettre dedans des fichiers de configuration de tous les micro-services** (et avec tous les différents profils) **utilisés par l'application :** application.properties, application-dev.properties, micro-service-metier1.properties, …
3. **Se positionner sur le dossier précédemment créé**
4. **Ouvrir un GitBash** sur le dossier

Puis taper dans la fenêtre GitBash :

* 1. git init // initialise le dépôt GIT sur le dossier
  2. git add -A . // ajoute (staged) tous les fichiers contenus dans le dossier dans l'index GIT
  3. git commit -m "COMMIT ORIGINEL DOSSIER configurations\_traficweb" // COMMIT ET PUSH tous les fichiers contenus dans le dossier dans le dépôt GIT

**Remarques** :

Les fichiers dans ce répertoire externalisé distant commençant par 'application' comme application.properties (profil DEFAULT), application-dev.properties (profil DEV), application-prod.properties (profil PROD) sont **communs à tous les micro-services enregistrés auprès du micro-service de configuration**. Tous ces micro-services enregistrés auprès du micro-service de configuration pourront donc bénéficier des propriétés définies dans application-{profil}.properties. C'est un point de factorisation des propriétés de factorisation.

#### Prendre en compte les modifications dans le dossier

Après avoir supprimé/ajouté/modifié des fichiers de configuration, il faut retaper dans GitBash sur le dossier :

1. git add -A . (attention, bien mettre le point)
2. git commit -m "nouveau\_commentaire"

Les modifications dans le répertoire distant contenant les fichiers de configuration de tous les micro-services sont désormais trackées par GIT.

#### Utiliser un simple dossier (Non GIT) comme conteneur des fichiers de configuration

On peut également utiliser un *simple* dossier (non dépôt GIT) mais il faut utiliser un PROFIL SPRING NATIVE.

Par exemple, dans un fichier application-native.properties :

* spring.profiles.active=native
* spring.cloud.config.server.native.searchLocations=file:///C:/donnees/eclipse/sts\_workspace/traficweb/configuration\_externalisee

### Compléter le fichier application.properties du Micro-Service de Configuration

**Objectif** :

1. **Préciser** *en profil DEV* le **port d'écoute** du Micro-Service de CONFIGURATION. *Ce port d'écoute doit pouvoir être redéfini par le centre-serveur lors du déploiement*.
2. **Nommer le Micro-Service de CONFIGURATION** (pour son enregistrement futur auprès du Micro-Service ANNUAIRE REGISRY).
3. **Indiquer au Micro-Service de CONFIGURATION** l'URI (Uniform Resource Identifier) du **répertoire externe** contenant les configurations de tous les micro-services utilisés par l'application.

Il faut préciser dans le application.properties du **Micro-Service de Configuration** :

* Le **port d'écoute** du Micro-Service de Configuration : server.port
* Le **nom** du Micro-Service de Configuration : spring.application.name
* **L'URI du répertoire** (GIT ou GITHUB) **contenant les fichiers .properties** externes servis par le Micro-Service de Configuration : spring.cloud.config.server.git.uri

Par exemple :

server.port=8888

spring.application.name=CONFIGURATION-SERVICE

spring.cloud.config.server.git.uri=file:///D:/Donnees/eclipse/sts\_workspace/traficweb\_fichiers\_externes/configurations\_traficweb

### Particularités du POM d'un Micro-Service de Configuration

Il peut être utile de sauter les tests unitaires lors du build Maven. Sinon, MAVEN lance l'application lors du Build.

Dans l'élément <properties>, rajouter :

<maven.test.skip>true</maven.test.skip>

### Accéder à la page du Service de configuration (URLS)

* /{application}/{profile}[/{label}]
* /{application}-{profile}.yml
* /{label}/{application}-{profile}.yml
* /{application}-{profile}.properties
* /{label}/{application}-{profile}.properties

application : nom du service (par exemple CONFIGURATION-SERVICE)

profile : Profil SPRING (dev, prod, test, ..)

label : nom de la branche GIT (par exemple : master, …)

#### Voir le application.properties en utilisant le nom du dossier GIT contenant les configurations

<http://localhost:8888/configurations_traficweb/master>

<http://localhost:8888/CONFIGURATION-SERVICE/configurations_traficweb>

<http://localhost:8888/CONFIGURATION-SERVICE/configurations_traficweb/master>

#### Voir le application.properties en utilisant le nom du micro-service de configuration

<http://localhost:8888/CONFIGURATION-SERVICE/default/master>

<http://localhost:8888/CONFIGURATION-SERVICE/default>

<http://localhost:8888/CONFIGURATION-SERVICE.properties>

<http://localhost:8888/CONFIGURATION-SERVICE-default.properties>

#### Voir le application-dev.properties et le application.properties en utilisant le nom du micro-service de configuration et le profil dev

<http://localhost:8888/CONFIGURATION-SERVICE/dev>

<http://localhost:8888/CONFIGURATION-SERVICE/dev/master>

<http://localhost:8888/CONFIGURATION-SERVICE-dev.properties>

#### Voir le application-prod.properties et le application.properties en utilisant le nom du micro-service de configuration et le profil prod

<http://localhost:8888/CONFIGURATION-SERVICE/prod>

<http://localhost:8888/CONFIGURATION-SERVICE/prod/master>

<http://localhost:8888/CONFIGURATION-SERVICE-prod.properties>

#### Voir les fichiers properties des Micro-Services métier via leur nom et le profil SPRING

**Attention** : fusionne TOUTES LES PROPRIETES contenues dans :

* application-{profil}.properties commun à tous les Micro-Services et
* METIER-SERVICE-{profil}.properties dévolu uniquement au service métier.

<http://localhost:8888/TEST-SERVICE/default>

<http://localhost:8888/TEST-SERVICE/dev>

<http://localhost:8888/TEST-SERVICE/prod>

### Documentation Swagger2

http://localhost:8888/v3/api-docs

## Test du Micro-Service de CONFIGURATION CENTRALISEE

## Vérifier la couverture des tests du Micro-Service de CONFIGURATION CENTRALISEE

## Tutoriels Micro-Service de Configuration

<https://www.baeldung.com/spring-cloud-configuration>

<https://xebia.developpez.com/tutoriels/spring/patterns-microservices-springcloud/>

<https://cloud.spring.io/spring-cloud-config/reference/html/>

<https://dzone.com/articles/intro-to-spring-cloud-config-server>

<https://javaetmoi.com/2018/10/architecture-microservices-avec-spring-cloud/>

<https://dzone.com/articles/spring-cloud-config-sever-and-client-article-2>

<https://openclassrooms.com/fr/courses/4668216-optimisez-votre-architecture-microservices/5176545-externalisez-la-configuration-de-vos-microservices>

<https://www.springboottutorial.com/creating-microservices-with-spring-boot-part-1-getting-started>

<https://www.springboottutorial.com/>

# Micro-Service TECHNIQUE ANNUAIRE (Registry)

# Micro-Service TECHNIQUE GATEWAY (Proxy)

## Développement du Micro-Service Gateway

### Dépendances

|  | Groupe de dépendances Spring Boot + Cloud | Dépendance |
| --- | --- | --- |
| 1 | **Spring Cloud Routing** | **Spring Cloud Gateway** |
| 2 | Spring Cloud Routing | Ribbon |
| 3 | Ops | Spring Boot Actuator |
| 4 | Ops | Codecentric's Spring Boot admin (Client) |
| 5 | Spring Cloud Circuit Breaker | Hystrix |
| 6 | Spring Cloud Discovery | Eureka Discovery Client |
| 7 | Spring Cloud Config | Config Client |
| 8 | Spring Cloud Tracing | Sleuth |
| 9 | Spring Cloud Tracing | Zipkin Client |
| 10 | Spring Cloud Security | Cloud Security |
| 11 | Developer Tools | Spring Boot DevTools |
| 12 | Developer Tools | Lombok |
| 13 |  | Swagger2 (documenter le micro-service) |
| 14 |  | ELK (Centraliser les Logs) |

## URL : localhost:9999/Micro-Service-Metier/products

localhost:9999/Micro-Service-Metier/products

# Micro-Service TECHNNIQUE HYSTRIX DASHBOARD

## Développement du Micro-Service Hystrix Dashboard

## Dépendances

|  | Groupe de dépendances Spring Boot + Cloud | Dépendance |
| --- | --- | --- |
| 1 | Spring Cloud Circuit Breaker | Hystrix Dashboard |
| 2 | Ops | Spring Boot Actuator |
| 3 | Spring Cloud Discovery | Eureka Discovery Client |
| 4 | Spring Cloud Config | Config Client |
| 5 | Developer Tools | Spring Boot DevTools |
| 6 | Developer Tools | Lombok |
| 7 |  | Swagger2 (documenter le micro-service) |
| 8 |  | ELK (Centraliser les Logs) |

## URL : localhost:10001/hystrix

localhost:10001/hystrix

## Tutoriels Circuit-Breaker

<https://arodrigues.developpez.com/tutoriels/architecture/design-pattern-circuit-breaker/>

<https://howtodoinjava.com/spring-cloud/spring-hystrix-circuit-breaker-tutorial/>

# Micro-Service METIER

## Développement d'un Micro-Service Métier

## Dépendances

|  | Groupe de dépendances Spring Boot + Cloud | Dépendance |
| --- | --- | --- |
| 1 | Web | Spring Web |
| 2 | Web | Rest Repositories |
| 3 | Web | Spring HATEOAS |
| 4 | SQL | Spring Data JPA |
| 5 | SQL | H2 Database |
| 6 | SQL | PostgreSQL Driver |
| 7 | Ops | Spring Boot Actuator |
| 8 | Spring Cloud Discovery | Eureka Discovery Client |
| 9 | Spring Cloud Config | Config Client |
| 10 | Developer Tools | Spring Boot DevTools |
| 11 | Developer Tools | Lombok |
| 12 |  | Swagger2 (documenter le micro-service) |
| 13 |  | ELK (Centraliser les Logs) |

## Tutoriels

Micro-Service

Micro-Service

Micro-Service