



# Projet Architecture Microservices – M. Menceur

03.07.2018

---

NAJI Mohammed Hamza / HILALY Nawfal

Master 2 MAGE IF Apprentissage

Université PARIS DAUPHINE

[naji.mohammedhamza@gmail.com](mailto:naji.mohammedhamza@gmail.com) / [nawfal.hilaly.9@gmail.com](mailto:nawfal.hilaly.9@gmail.com)

## Introduction

L'architecture MicroService (**MSA**) est apparue pour répondre à deux principaux challenges:

**1 - comment concevoir des applications ultra disponibles ne souffrant quasi jamais d'interruption et qui acceptent des mises à jour très fréquentes ?**

**2 - comment profiter du Cloud pour développer des applications Cloud-Native qui supportent n'importe les différentes montée en charges et qui sont très économiques ?**

La solution proposée par MSA est de découper une application en plusieurs multi-services nommés **micro-services** parfaitement autonomes et qui exposent des **API REST**. Chaque micro-service a une seule responsabilité (par ex : la gestion du pricing ou l'ajout d'utilisateurs ou la suppression d'utilisateurs, etc). Il s'ajoute à cela des **Edge Micro-services** qui sont des micro-services permettant la cohésion et le fonctionnement globale du système (par ex : la sécurité, la configuration des micro-services, etc).

Les avantages de la conception d'applications sur la base de MSA sont:

**1 - la scalabilité** de cette application, en augmentant et diminuant ses capacités **sans jamais interrompre le service.**

**2 - large choix de technologies** notre micro-service peut être développé en JAVA, C++, etc, tant qu'il expose une API REST aux autres micro-services.

**3 - cette flexibilité** offre des avantages énormes avec des grandes performances.

**4 - Réduction des coûts** de la maintenance et des mise à jour du à la simplicité des micro-services et les coûts des serveurs.

## Goals

Le but de ce projet est de se familiariser avec les notions des micro-services. En effet, il nous est demandé de créer une architecture d'une application permettant de répondre à deux besoins :

**1 - gérer (créer, récupérer, modifier et supprimer) des taux de change entre deux devises.**

**2 - gérer les opérations de change.**

Dans un premier temps nous allons commencer par exposer le diagramme de classes "**métiers**" à partir de notre problématique et de découper notre projet en deux micro-services, réalisés grâce à l'outil **Spring Boot**, afin de tirer profits de leur avantages.

## Projet

### Diagramme de classes “métiers”

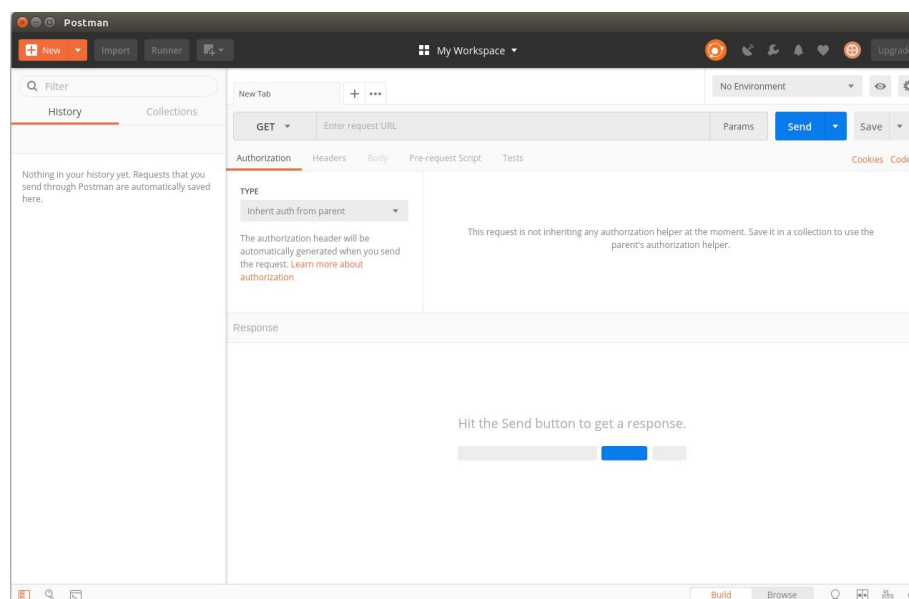
### Spring boot

Pour résumer Spring Boot par un seul mot, ce serait "**Simplicité**", il repose sur le principe **KISS (Keep It Simple Stupid)**. Spring Boot a été conçu pour rendre la vie du développeur plus simple et lui permettre de se concentrer sur le coeur de l'application et non pas sur les aspects annexes : configuration, tests, sécurité, déploiement...

La configuration complète par défaut de l'application se résume à l'annotation **@EnableAutoConfiguration**. Cette annotation va alors s'appuyer sur l'ensemble des dépendances de l'application (MVC, Tomcat, ...) pour configurer l'application. Il nous permet aussi d'affecter le port d'écoute à une valeur différente que celle par défaut du port HTTP (**8080**), en ajoutant au fichier **pom.xml** l'instruction **server.port 9090** par exemple.

### Post Man

**Post Man** est une application qui permet de tester les API REST, elle permet de personnaliser et d'envoyer des requêtes en utilisant tous les **verbes HTTP** possible (GET, POST, PUT, etc). Elle formate ensuite les réponses et nous offre la possibilité de les analyser très facilement.



**Figure 1 : Interface POST MAN**

## Spring Data JPA

FrameWork de la famille Spring Data. Il nous permet d'automatiser en grande partie la création de la couche d'accès aux données. En respectant les conventions de nommage, Spring Data va générer pour nous les requêtes nécessaires pour interagir avec notre BDD.

## Les microservices :

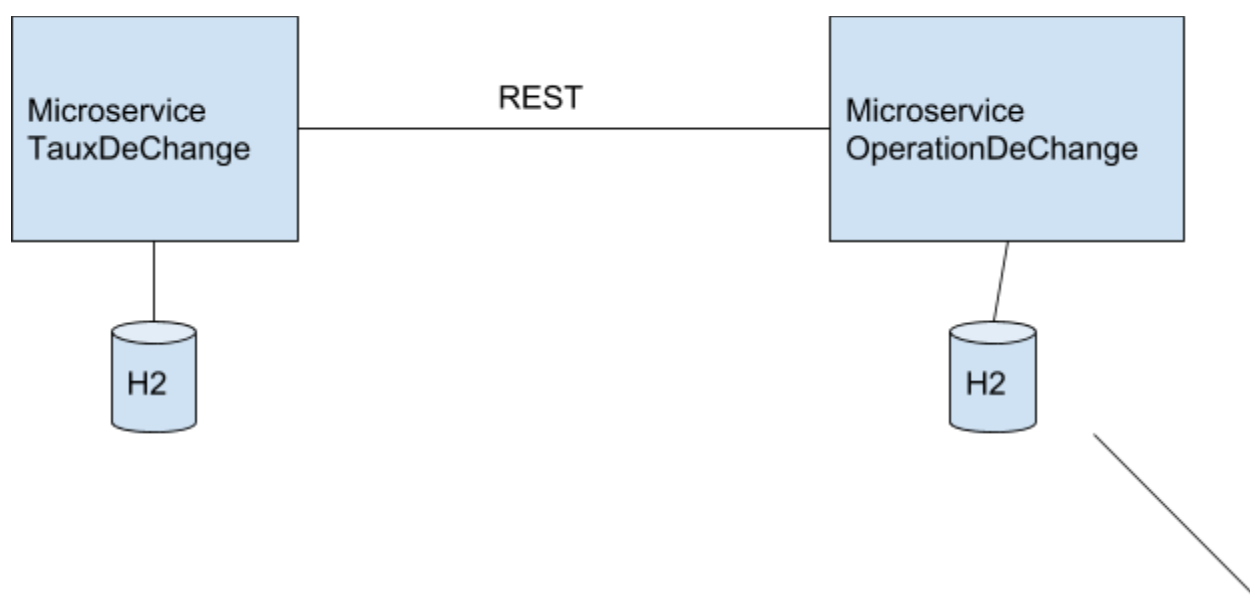
Nous découpons notre application en deux micro-services indépendants :

### 1 - microservice : TauxDeChange

Qui permet de fournir des informations concernant le taux de change entre deux devises à une date donnée. Il s'agit d'une application web service REST.

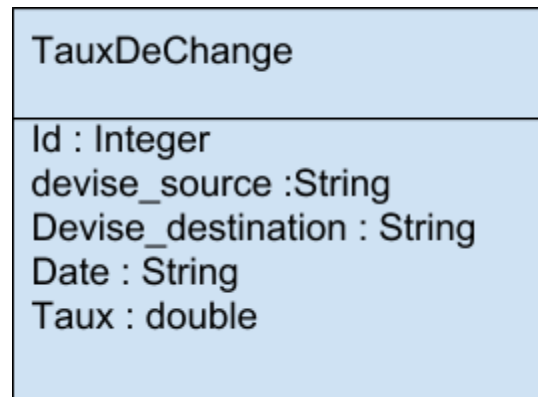
### 2- microservice : OperationDeChange

Qui permet de gérer des transactions de changes de devises. Ce micro service récupère le taux de change à partir du web service TauxDeChange en suivant le principe REST et en utilisant un protocole HTTP.



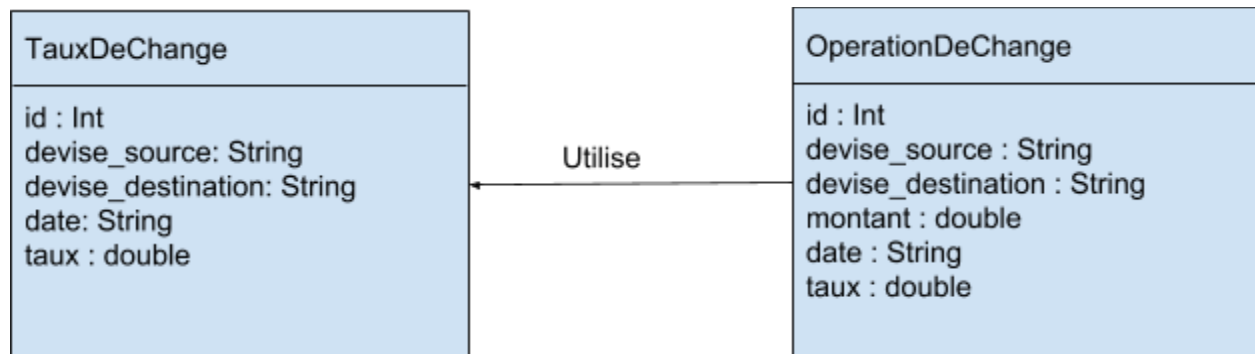
**Figure 2 : Les deux microservices**

## Premier microservice : “TauxDeChange”



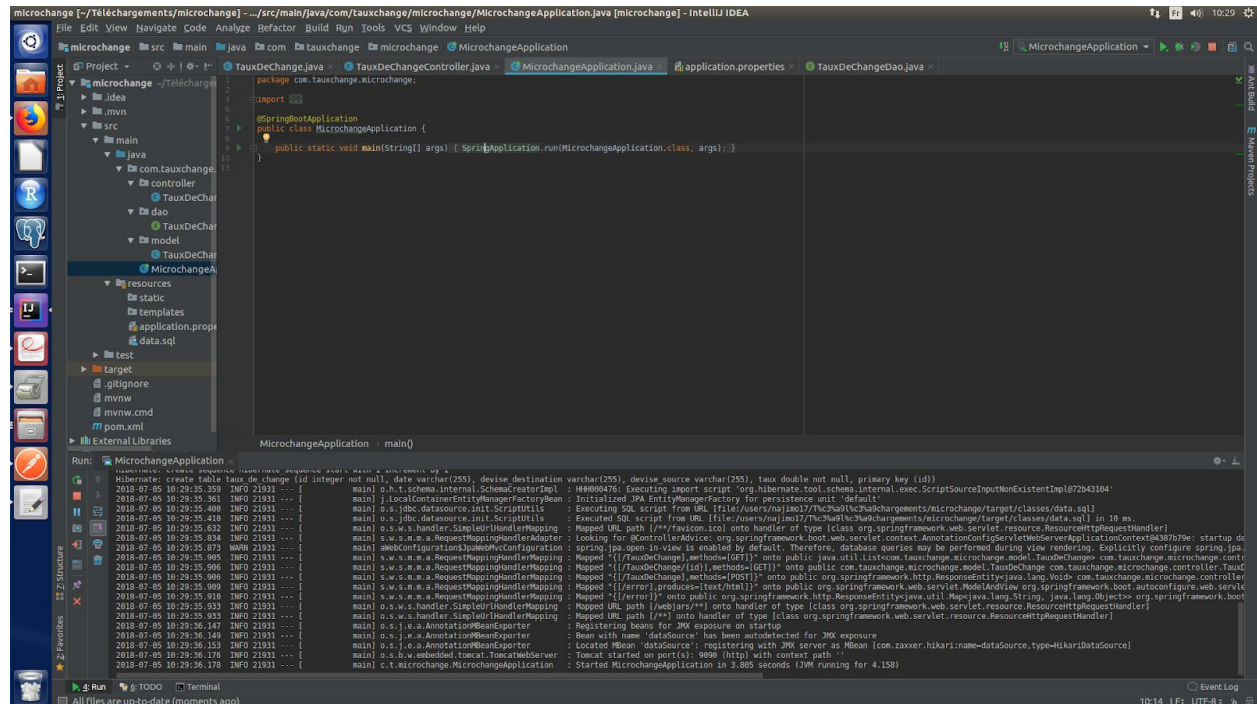
**Figure 3 : classe métier TauxDeChange**


## Deuxième microservice : “OperationDeChange”



**Figure 4 : classe métier OperationDeChange**

Nos deux microservices marchent correctement et nous pouvons le voir sur la console :





Il est à noter que l'adresse du micro service TauxDeChange est <http://localhost:9090/TauxDeChange>, alors que l'adresse du micro service OperationDeChange est <http://localhost:8080/OperationDeChange>.

## Conclusion

Ce projet nous a permis de mettre en pratique les notions vues en cours et de découvrir des notions très importantes tant sur le plan éducatif que sur le plan professionnel .