ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

IoT Cloud
TP3

TP3 – DataRouting

Partie 1: Déployer et manipuler le service

1.1 Introduction

Dans ce TP, vous allez jouer avec un outil de *DataRouting* afin d'en comprendre l'intérêt, les avantages et les limites. Vous allez en particulier découvrir *Nifi* [2].

Pour rappel, vous devez rendre un rapport individuel à la fin du semestre. Vous devez y mettre toutes les informations qui vous semblent pertinentes. Notez que les réponses aux questions encadrées en gris seraient à considérer comme un minimum à faire apparaitre dans ce rapport.

1.2 Déployer Nifi

Dans un premier temps, vous devez :

1. Déployer un serveur Nifi à partir du *docker-compose* proposé à la Figure 1.1.
2. À partir d'une *queue* RabbitMQ contenant des messages (vous pouvez réutiliser le TP1 pour déployer et remplir une **queue** RabbitMQ), créer un *flow* Nifi permettant de lire les messages de la *queue* RabbitMQ et de pousser les messages vers une base de données MongoDB (vous pouvez réutiliser le TP2 pour déployer un serveur MongoDB et voir que les messages s'ajoutent bien...)
3. Classe ! On va avoir besoin d'avoir un *flow* comme celui-là pour chaque client au moins, non ? Et si on créait un *template* ? Créer un *template* rassemblant votre *flow* et téléchargez-le sur votre ordinateur.
4. Imaginez maintenant... l'un des capteurs a dû être remplacé. Le nouveau fonctionne très bien mais génère des données d'une unité 1000 fois plus petite (par exemple, le capteur précédent générait des données en `kWh`, le nouveau en `Wh`). Pour éviter que notre base de données n'ait des données incohérentes, nous allons utiliser Nifi... Vous allez ajouter un processeur Nifi permettant de multiplier toutes les valeurs lues dans la *queue* RabbitMQ par `1000`. (j'entends une voix... elle chuchote `UpdateRecord` [7]... étrange, non ?! Il parait même que Nifi a un langage pour remplacer un `field.value` par un multiple [4])

```yaml
version: '3.7'
services:

  nifi:
    image: "apache/nifi:1.9.2"
    hostname: "nifi"
    environment:
      NIFI_WEB_HTTP_PORT: "8080"
    ports:
      - "8083:8080"
    networks:
      - iot-labs
    labels:
      NAME: "nifi"

networks:
  iot-labs:
    external: true
```

Figure 1.1: Exemple de *docker-compose* permettant de déployer un serveur Nifi avec son interface. Voir [1] pour plus de détails.

1.3 Récupérer les données d'une API ?

Super, mais maintenant vos clients voudraient avoir les données de météo historiques et spécifiques à leur ville... vous allez donc devoir récupérer et stocker tout ça dans votre MongoDB !
Vous devez donc :

5. Créer une nouvelle base de données MongoDB (par exemple : "`weather`") puis une *collection* pour la ville (par exemple : "`paris`").
6. Créer un compte gratuit sur WeatherStack.com afin d'obtenir une "*API Key*".
   Vous pourrez alors récupérer un JSON contenant les données météo actuelles avec un appel à l'adresse :
   http://api.weatherstack.com/current?access_key=YOUR_ACCESS_KEY&query=Paris
7. Ajouter un *flow* à Nifi permettant de récupérer les données de météo de Paris toutes les 10 minutes et de stocker le résultat dans la *collection* "`paris`" de la base de données "`weather`" sur MongoDB. (Indice : une API web est accessible par un appel `GET` sur une adresse accessible en `HTTP`)
8. Vous verrez que l'API vous renvoie de nombreuses informations sur la ville (nom, latitude, longitude etc)... il n'est peut-être pas nécessaire de tout stocker à chaque fois. Il vous faut modifier votre *flow* Nifi pour ne conserver que la partie "`current`" du retour de l'API. (*JSONPath* [3] ? `EvaluateJsonPath` [5] ?! Qui a dit ça ?!)

1.4 Et si les données sont en CSV [9] sur un FTP [10] ?

Ok. Maintenant, il se passe quoi si votre client a déjà une procédure pour rassembler ses données, ou qu'il utilise déjà un autre outil qui rassemble les données dans une fichier `CSV` et les rend accessibles au travers d'un serveur `FTP` ? Pour l'exemple, c'est notamment le cas dans des entreprises comme PSA qui ne permettent pas de collecter des données dans leurs entrepôts directement mais s'occupent de rassembler les données sur des serveurs FTP. Pourquoi ? Sécurité, sécurité...
Vous devez maintenant :

9. Déployer un serveur FTP à partir du *docker-compose* proposé à la Figure 1.2. Celui-ci créera un dossier "`ftp`" qui contiendra un dossier "`user`" (ce qui correspond au `FTP_USER`). Si vous vous connectez à ce serveur FTP à partir de votre hôte (par exemple, après avoir installé l'outil graphique "`filezilla`") avec les identifiants indiqués dans le *docker-compose*, vous pouvez déposer un fichier et constater que celui-ci se place bien dans le dossier "`ftp/user`". Sinon, vous pouvez aussi copier un fichier directement dans le dossier "`ftp/user`" à

partir de votre terminal `cp FICHIER_SOURCE DESTINATION`, soit par exemple `cp mon_fichier.csv ftp/user/`

10. Créer un fichier CSV contenant les informations présentées en Figure 1.3.
11. Créer un *flow* Nifi qui permet, à chaque fois qu'un nouveau fichier est déposé sur le serveur FTP, de :
    (a) Récupérer ce fichier avec le bon processeur sur Nifi. (Attention, dans le mode actuel de déploiement du serveur FTP, celui-ci utilisera un mode de connexion `Active` et non `Passive`.)
    (b) Découper chaque ligne grâce à un processeur `SplitRecord` [6]. Vous devrez alors y intégrer un "`RecordReader`" permettant de lire les CSV, et un "`RecordWriter`" permettant d'écrire des JSON. Attention à bien les configurer...
    (c) Envoyer les données sur une base de données MongoDB de votre choix.
12. Super ! Mais, heu... le nom de la colonne des valeurs n'est pas très joli. "`val`" ne correspond pas à nos attentes, nous voudrions que ce soit "`VALUE`". Trouver un processeur permettant de *remplacer ce texte* après avoir lu le fichier CSV. (Attendez, comment on dit *remplacer un texte* en anglais déjà ? 🔑)

```yaml
version: '3.7'
services:

  ftp:
    image: "fauria/vsftpd"
    hostname: "ftp"
    environment:
      FTP_USER: "user"
      FTP_PASS: "password"
      LOG_STDOUT: "YES"
    ports:
      - "21:21"
      - "20:20"
      - "21100-21110:21100-21110"
    volumes:
      - "./ftp:/home/vsftpd"
    networks:
      - iot-labs
    labels:
      NAME: "ftp"
    privileged: true
    restart: always

networks:
  iot-labs:
    external: true
```

Figure 1.2: Exemple de *docker-compose* permettant de déployer un serveur FTP. Voir [8] pour plus de détails.

1.5 Optionnel – Découvrir d'autres subtilités de Nifi

Pour aller plus loin, voici quelques pistes :

1. Créer un "*Process Group*" pour rassembler les *flows* d'un même client.
2. Explorer à quoi servent les "*Input Port*" et "*Output Port*".
3. Il y a un processeur "*ExecuteScript*"... il doit bien avoir un intérêt !
4. Et vous saviez que vous pouviez créer vos propres processeurs ?!
5. Et qu'il existe une API pour ne pas à avoir à tout faire avec l'interface ?

```
DATE;SENSOR;val
10/10/19 9:00;Conso1;2666,213967
10/10/19 9:10;Conso1;2137,396032
10/10/19 9:20;Conso1;2735,650848
10/10/19 9:30;Conso1;1553,431867
10/10/19 9:40;Conso1;3590,205401
10/10/19 9:50;Conso1;2466,49594
10/10/19 10:00;Conso1;2971,633414
10/10/19 10:10;Conso1;3964,156069
10/10/19 10:20;Conso1;1383,909196
10/10/19 10:30;Conso1;3195,50807
10/10/19 10:40;Conso1;1175,62607
10/10/19 10:50;Conso1;2386,020869
10/10/19 11:00;Conso1;2719,344498
10/10/19 11:10;Conso1;2740,257564
10/10/19 11:20;Conso1;1562,102106
10/10/19 11:30;Conso1;1914,720845
10/10/19 11:40;Conso1;1455,32196
10/10/19 11:50;Conso1;3127,209843
10/10/19 12:00;Conso1;1804,493063
10/10/19 12:10;Conso1;1004,90129
10/10/19 12:20;Conso1;2361,638979
10/10/19 12:30;Conso1;2656,268764
10/10/19 12:40;Conso1;1996,6814
10/10/19 12:50;Conso1;2862,780306
10/10/19 13:00;Conso1;1565,716749
10/10/19 13:10;Conso1;3050,976539
10/10/19 13:20;Conso1;2546,773871
10/10/19 13:30;Conso1;2478,743919
10/10/19 13:40;Conso1;2251,994654
```

Figure 1.3: Exemple de fichier CSV contenant des données de consommation au pas de 10 minutes.

Partie 2: Bibliographie

[1] Apache. *Image Docker officielle de Nifi*. URL: https://hub.docker.com/r/apache/nifi/tags/.

[2] Apache. *Nifi*. URL: https://nifi.apache.org/.

[3] Kazuki Hamasaki. *JSONPath Online Evaluator*. URL: https://jsonpath.com/.

[4] Nifi. *Apache NiFi Expression Language Guide*. URL: https://nifi.apache.org/docs/nifi-docs/html/expression-language-guide.html.

[5] Nifi. *EvaluateJsonPath documentation*. URL: https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.EvaluateJsonPath/index.html.

[6] Nifi. *SplitRecord documentation*. URL: https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.SplitRecord.

[7] Nifi. *UpdateRecord documentation*. URL: https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.UpdateRecord/.

[8] Fer Uria. *Docker permettant de déployer un serveur "vsftpd"*. URL: https://hub.docker.com/r/fauria/vsftpd/.

[9] Wikipedia. *Comma-separated values*. URL: https://fr.wikipedia.org/wiki/Comma-separated_values.

[10] Wikipedia. *File Transfer Protocol*. URL: https://fr.wikipedia.org/wiki/File_Transfer_Protocol.