

TP4 – PRESENTATION

1 Déployer et manipuler les services	1
1.1 Introduction	1
1.2 Présenter... mais quoi ?	1
1.3 Déployer... InfluxDB [4]	2
1.4 Déployer... Grafana [3]	5
2 Bibliographie	6

PARTIE 1: DÉPLOYER ET MANIPULER LES SERVICES

1.1 Introduction

Dans ce TP, vous allez jouer avec un outil de *Presentation* afin d'en comprendre l'intérêt, les avantages et les limites. Vous allez en particulier découvrir *Grafana* [3].

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 apparaître dans ce rapport.

1.2 Présenter... mais quoi ?

C'est cool, on a des "objets connectés", ils produisent des données, on a une architecture qui claque et permet d'absorber une quantité impressionnante de données (*RabbitMQ* [13]), de les router en fonction des besoins spécifiques (*Nifi* [1]) et de les stocker (*MongoDB* [9]).

Et après ? On paie des serveurs de plus en plus gros et on s'enorgueillit d'avoir plein plein de données ? Ou on en fait quelque chose ?

Vous allez aujourd'hui travailler sur un *Dashboard* permettant de suivre les données collectées. Dans un premier temps, vous devez :

1. Définir et expliquer les différentes informations pertinentes à présenter et la façon de les présenter (vous devez faire une maquette quoi...) qui vous permettra de mettre en avant les données des clients d'après les informations du *Fil Rouge*.

Pour vous aider, quelques questions à vous poser :

- Si je suis client, quelles sont les informations pertinentes qui je voudrais voir tous les jours ? (*Data storytelling...*)
- Si je vois une information étrange, comment je voudrais pouvoir fouiller la donnée, quelles représentations m'aideraient à comprendre une situation ? (Fouille de données)
- Si je suis un gestionnaire (énergie ? sécurité ?) et que j'ai plusieurs clients, quelles informations seraient pertinentes à voir ?

1.3 Déployer... InfluxDB [4]

L'outil que nous allons utiliser, *Grafana*, ne permet pas de se connecter simplement à une base de données *MongoDB* [12]. Nous allons donc commencer par déployer une nouvelle base de données orientée *Time Series*, *InfluxDB* [4]. Pour cela, vous devez :

2. Déployer un serveur *InfluxDB* [6] avec une interface graphique *Chronograf* [5] à partir du *docker-compose* proposé à la Figure 1.1.
3. Se connecter à l'interface *Chronograf* pour créer une base de données : **CREATE DATABASE "iot"**
4. Les données générées durant le TP1 qui sont envoyées sur une *queue RabbitMQ* sont sous format JSON (exemple : {"DATE": "1570985503", "SENSOR": "SDBPuiss", "VALUE": "201.54"}), or *InfluxDB* ne peut recevoir de fichiers JSON mais peut recevoir des données suivant le *InfluxDB line protocol* [8] (par exemple : **client1 SDBPuiss=201.54 1570985503**).
Nous allons utiliser un *flow Nifi* pour réaliser cette conversion. Ajouter un processeur *Nifi ExecuteScript* [10] avec **ECMAScript** comme *ScriptEngine* et basé sur le code proposé en Figure 1.2.
5. Finalement, comme présenté en Figure 1.3, ajouter un processeur *Nifi PutInfluxDB* [11] permettant d'envoyer les données vers une base de données *InfluxDB*.
6. Sinon, vous pouvez télécharger le *flow Nifi* mis à votre disposition sur votre *LMS*... Enfin, si vous lisez bien le sujet en entier avant de commencer...
7. Une fois que vous aurez envoyé quelques données à partir du générateur proposé au TP1 puis lancé votre *flow Nifi*, vous devriez donc avoir des données sur votre base de données *InfluxDB*. Pour vérifier, retournez sur l'interface *Chronograf* puis testez la commande **SELECT * FROM "iot"."autogen"."client1"** (si votre base de données est nommée "iot" et votre *measurement* [7] est nommé "client1"), comme présenté en Figure 1.4.

```

1 version: '3.7'
2 services
3
4 influxdb:
5   image: "influxdb"
6   hostname: "influxdb"
7   ports
8     - "8086:8086"
9   networks
10    - iot-labs
11   labels
12     NAME: "influxdb"
13
14 influxdbchronograf:
15   image: "chronograf"
16   hostname: "chronograf"
17   ports
18     - "8087:8088"
19   networks
20    - iot-labs
21   labels
22     NAME: "chronograf"
23   command: "--influxdb-url=http://influxdb:8086"
24
25 networks
26   iot-labs
27     external: true

```

Figure 1.1 – Exemple de *docker-compose* permettant de déployer un serveur *InfluxDB* [6] avec une interface graphique *Chronograf* [5].

```
1 flowFile = session.get();
2 if (flowFile != null) {
3     var StreamCallback = Java.type("org.apache.nifi.processor.io.StreamCallback");
4     var IOUtils = Java.type("org.apache.commons.io.IOUtils");
5     var StandardCharsets = Java.type("java.nio.charset.StandardCharsets");
6     var error = false;
7     var measure = "client1"
8     var line = "";
9     var sep = "\n"
10
11     // Get attributes
12     flowFile = session.write(flowFile, new StreamCallback(function (inputStream, outputStream) {
13         var content = IOUtils.toString(inputStream, StandardCharsets.UTF_8); // message or content
14         var message_content = {};
15         var sensor = "";
16         var date = "";
17         var value = "";
18
19         try {
20             message_content = JSON.parse(content);
21             for (key in message_content){
22                 if (key == 'SENSOR') {
23                     sensor = message_content[key]
24                 } else if (key == 'DATE') {
25                     date = message_content[key] * 1000 * 1000 * 1000
26                 } else if (key == 'VALUE') {
27                     value = message_content[key]
28                 }
29             }
30             line = measure + " " + sensor + "=" + value + " " + date + sep
31
32             // Write output content
33             outputStream.write(line.getBytes(StandardCharsets.UTF_8));
34
35             } catch (e) {
36                 error = true;
37                 log.error('Something went wrong', e)
38                 outputStream.write(content.getBytes(StandardCharsets.UTF_8));
39             }
40         }));
41
42         if (error) {
43             session.transfer(flowFile, REL_FAILURE)
44         } else {
45             session.transfer(flowFile, REL_SUCCESS)
46         }
47     }
```

Figure 1.2 – Code ECMAScript utilisable sur un processeur *Nifi ExecuteScript* [10] permettant de convertir un JSON (exemple : {"DATE": "1570985503", "SENSOR": "SDBPuiss", "VALUE": "201.54"}) vers un texte compatible avec le *InfluxDB line protocol* [8] (par exemple : *client1 SDBPuiss=201.54 1570985503*).

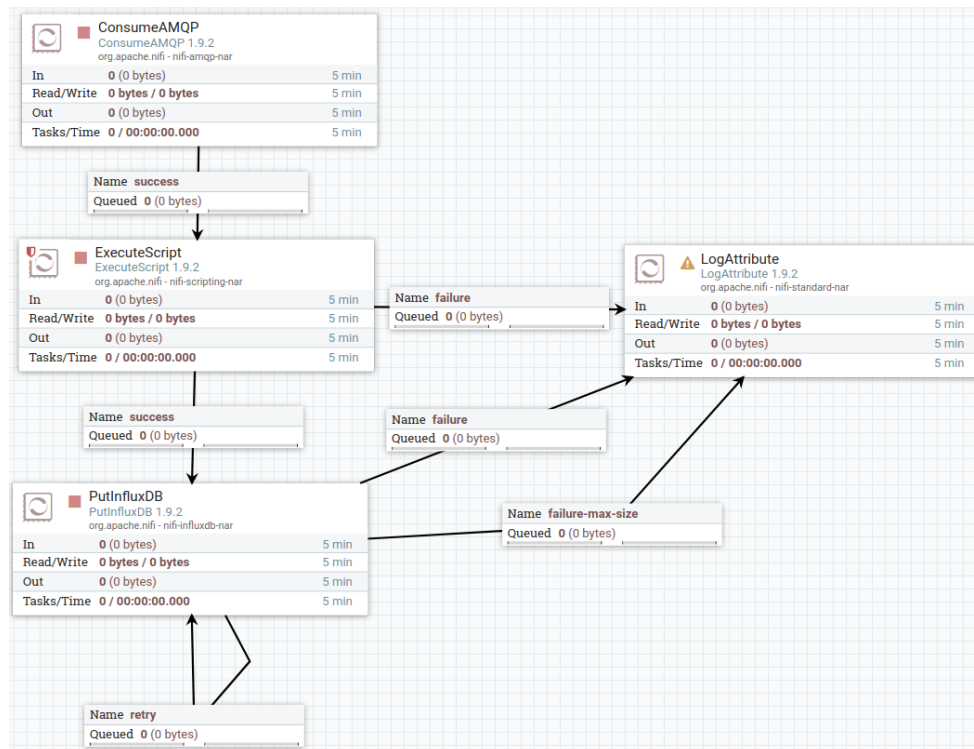


Figure 1.3 – *Flow NiFi* permettant de récupérer des données JSON sur une *queue RabbitMQ*, de les convertir avec un code comme présenté en Figure 1.2 puis de les envoyer vers une base de données *InfluxDB*.



Figure 1.4 – Résultat d'une requête sur *Chronograf* permettant de vérifier les données d'une base de données *InfluxDB*.

1.4 Déployer... Grafana [3]

Vous devez maintenant :

8. Déployer un serveur *Grafana* [3] à partir du *docker-compose* proposé à la Figure 1.5.
9. Une fois connecté à l'interface, ajouter une source de données *InfluxDB* en configurant bien l'*URL* et la *Database*.
10. Nous considérons seulement deux capteurs, un capteur de puissance instantanée et un capteur de température. Créer un dashboard comme présenté en Figure 1.6 contenant :
 - Une gauge indiquant la température actuelle (en vert jusqu'à 20°C, orange jusqu'à 25°C puis en rouge jusqu'à 35°C).
 - Une gauge indiquant la température moyenne.
 - Un *singlestat* indiquant la consommation moyenne à l'heure en kWh.
 - Un graphique contenant deux courbes :
 - La courbe de puissance instantanée indiquant en légende les valeurs min/max/moyenne/courante. Vous utiliserez l'axe Y de gauche pour cette courbe.
 - La courbe de la température avec la même légende. Vous utiliserez l'axe Y de droite pour cette courbe en le limitant entre 18°C et 30°C.
 - Vous ajouterez à la courbe de puissance un seuil à 1000kW permettant de voir rapidement les dépassements.

```
1 version: '3.7'
2 services:
3   grafana
4     image: "grafana/grafana"
5     hostname: "grafana"
6     environment:
7       GF_SECURITY_ADMIN_PASSWORD: "secret"
8     ports:
9       - "8084:3000"
10    networks:
11      - iot-labs
12    labels:
13      NAME: "grafana"
14  networks:
15    iot-labs
16    external true
```

Figure 1.5 – Exemple de *docker-compose* permettant de déployer un serveur *Grafana* [2].

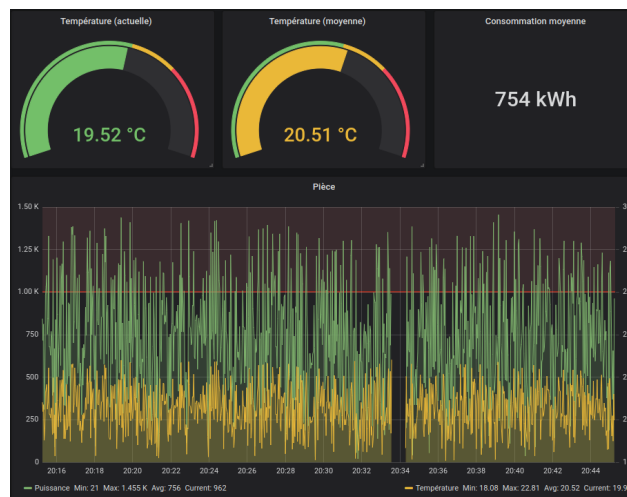


Figure 1.6 – Dashboard à reproduire sur *Grafana*.

PARTIE 2: BIBLIOGRAPHIE

- [1] Apache. *Nifi*. URL: <https://nifi.apache.org/>.
- [2] Grafana. *Image Docker officielle de Grafana*. URL: <https://hub.docker.com/r/grafana/grafana/>.
- [3] Grafana. *Site web principal*. URL: <https://grafana.com/>.
- [4] InfluxDB. *Site web principal*. URL: <https://www.influxdata.com/>.
- [5] InfluxData. *Image Docker officielle de Chronograph*. URL: https://hub.docker.com/_/chronograf.
- [6] InfluxData. *Image Docker officielle de InfluxDB*. URL: https://hub.docker.com/_/influxdb.
- [7] InfluxData. *InfluxDB key concepts: measurement*. URL: https://docs.influxdata.com/influxdb/v1.7/concepts/key_concepts/#measurement.
- [8] InfluxData. *InfluxDB line protocol tutorial*. URL: https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_tutorial/.
- [9] MongoDB. URL: <https://www.mongodb.com/>.
- [10] Nifi. *ExecuteScript documentation*. URL: <https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-scripting-nar/1.6.0/org.apache.nifi.processors.script.ExecuteScript>.
- [11] Nifi. *PutInfluxDB documentation*. URL: <https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-influxdb-nar/1.6.0/org.apache.nifi.processors.influxdb.PutInfluxDB/>.
- [12] James Osgood. *MongoDB datasource for Grafana*. URL: <https://github.com/JamesOsgood/mongodb-grafana>.
- [13] RabbitMQ. URL: <https://www.rabbitmq.com/>.