

RAPPORT

15 JANVIER 2021

---

# Développement d'applications et webservices pour l'IoT

---

*Écrit par :*  
Bouvet Xavier



ÉCOLE  
**D'INGÉNIEURS**  
PARIS-LA DÉFENSE

# Table des matières

0.1	Introduction et présentation	2
<b>1</b>	<b>TP1 Ingestion</b>	<b>3</b>
1.1	RabbitMQ	3
1.1.1	Définition de l'architecture	3
1.1.2	Déployer votre architecture	5
1.1.3	Découvrir l'envoi et la réception de messages	5
1.1.4	Automatisation du déploiement	8
1.1.5	Génération automatique de messages	9
<b>2</b>	<b>TP2 Stockage</b>	<b>10</b>
2.1	MongoDB	10
2.1.1	Définition de l'architecture	10
2.1.2	Déployer votre architecture	12
2.1.3	Découvrir l'envoi et la réception de données	13
2.1.4	Automatisation du déploiement	14
2.1.5	Génération automatique de messages	14
<b>3</b>	<b>TP3 Data Routing</b>	<b>16</b>
3.1	Nifi	16
3.1.1	Déployer Nifi	16
3.1.2	Récupérer les données d'une API ?	19
3.1.3	Et si les données sont en CSV [9] sur un FTP [10] ?	22
<b>4</b>	<b>TP4 Présentation</b>	<b>26</b>
4.1	Introduction	26
4.1.1	Présenter... mais quoi ?	26
4.2	InfluxDB	26
4.2.1	Déployer... InfluxDB [4]	27
4.3	Grafana	29
4.3.1	Déployer... Grafana [3]	30
<b>5</b>	<b>TP5 Principe d'API</b>	<b>32</b>
5.1	OpenAPI	32
5.1.1	Prise en main de Swagger Codegen [6]	32
5.1.2	How to "do some magic" ? – Remplir le code de Swagger Codegen	33
5.1.3	Et maintenant... on fait le lien avec le TP2 ?	33

## Introduction et présentation

Dans le cadre du cours de développement d'application et webservice , nous nous sommes positionné en tant que jeune entreprise du milieu de l'IoT . Notre but étant d'offrir un dashboard concernant la consommation énergétique afin de réduire graduellement leur consommation .

Nous représentons un groupe utilisant les capteurs et actionneurs de différents partenaires et ne nous occupons pas de ces problèmes directement .

Afin de satisfaire l'ensemble de nos clients au nombre de 2 , nous tenons à offrir un service sans égal ayant un potentiel de croissance très vaste . C'est pourquoi même avec 2 clients nous considérons un flux de données très large .

# TP1 Ingestion

## RabbitMQ

RabbitMQ : <http://192.168.99.100:15672/>

### 1.1.1 Définition de l'architecture

Définir et expliquer l'architecture Exchanges/Queues qui vous suivra durant les TPs pour répondre au Fil Rouge (réception des données).

Nous utiliserons une architecture simple avec 1 queue par client, chaque capteur sera liée à un virtual host faisant référence au client liée **Pour vous aider, quelques questions à vous poser :**

Est-ce que les données d'un client doivent être cloisonnées de celles des autres clients ?

Pour une question de logique et de sécurité le cloisonnement me semble important. En effet une simple recherche sur un tag associé à un client nous renverra toutes les informations concernant ce client.

Est-ce qu'il faut un exchange par capteur, par maison, par client, par quartier... ?

Dans le cas où nous nous intéressons à la consommation énergétique, il me semble logique de regrouper les exchange par propriété. On peut aussi penser au cas où un client contrôlerait 2 lieux différents et voudrait garder ces 2 entités distinctes.

Quel impact cela va-t-il avoir sur la configuration de mes capteurs / objets connectés ?

Le regroupement fait précédemment facilitera l'installation et transférera la charge plus loin sur la chaîne du traitement des informations.

Comment gérer les droits d'accès aux queues ?

Des droits d'accès seront mis en place pour chaque virtual host/queue.

Est-ce que, si les messages se retrouvent dans la même queue de messages, je pourrai les ré-identifier facilement ?

Des méta-data pourrons être intégrer au message en provenance des capteurs pour augmenter leurs traçabilités .

Du point de vue des consommateurs de données, sera-t-il plus simple et/ou plus sécurisé d'avoir une ou plusieurs queues de messages ?

Une seul queue est plus simple et plus sécurisé . L'intérêt d'avoir plusieurs queue serait de fait un tri préalable des informations . Mais dans le cadre de petit capteur pouvant être bouger a volonté il faudrait ajusté les queues sur lequel les capteurs postent leurs message pour en profiter pleinement ce qui n'est pas une action que tout le monde veux avoir a faire régulièrement .

Quel sera limpact si je perds complètement une queue / un noeud de mon architecture ?

L'impact seras important mais pas tant que cela , perdre une queue représente perdre 100% de l'afflux de donnée , cependant ce n'est qu'une perte de donnée s'étalant sur la période d'échec de la queue . Le reste des informations sont sauvegarder sur une base de donnée .

Si, demain, j'ai vraiment beaucoup de clients, est-ce que mon architecture sera scalable [11] ? Est-ce quelle passera à l'échelle [10] ?

La création de virtual host et de queue est simple et nous n'allons pas surcharger ni virtual host ni queue car l'afflux sur ces dernières est indépendant du nombre de client (il y auras toujours 1 queue et 1 virtual host par client).

Si, demain, j'ai des clients internationaux qui ne veulent pas que leurs données sortent de leur pays, est-ce que mon architecture globale reste viable ?

Oui si l'on installe l'infrastructure dans le dit pays comme dit précédemment "il y auras toujours 1 queue et 1 virtual host par client" donc si tout est près a accueillir ces dernier dans un autre pays cela est faisable

Et sils veulent héberger cette partie de ma solution directement chez eux pour contrôler plus finement ce qui sort de chez eux ?

Faut-il que je me répète , votre queue votre vie .

Définir et expliquer l'architecture Exchanges/Queues qui vous servirait dans ce cas à "redescendre" des informations vers des objets chez le client.

Il faudra dans un premier temps router le message vers le bon client identifiable par son virtual host et sa queue puis se tourner vers le bon objet grâce à son identifiant retrouvable dans les méta-datas .

### 1.1.2 Déployer votre architecture

Déployer un serveur RabbitMQ à partir du docker-compose proposé à la Figure 1.1.

Liste des commandes à exécuter dans l'ordre :

- **docker network create –attachable iot-labs**
- **docker-compose up**

Welcome to windows : J'ai besoin d'utiliser mon adresse ip pour accéder à l'interface web localhost ne marche pas . Welcome to http ://192.168.99.100 :15672/

Les identifiants sont donnés dans le docker-compose.yml

- **Username : rabbitmq**
- **Password : rabbitmq**

Implémenter votre architecture de remontée de données à partir de l'interface web proposée par RabbitMQ

Overview					Messages			Message rates			
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
Client1	Client1	classic	D Args	Idle	0	0	0				
Client2	Client2	classic	D Args	Idle	0	0	0				

Le binding est simple car c'est une liaison directe entre le virtual host et la queue , cela correspond au binding par défaut .

### 1.1.3 Découvrir l'envoi et la réception de messages

Jouer avec votre architecture en générant des données à la main à partir de l'interface web. Testez par exemple en générant quelques données pour 2 capteurs différents de 2 pièces différentes pour chacun des clients .

Après mettre arracher les cheveux et avoir voulu utilisé pyrabbit suite au conseil d'un "amis" , j'ai découverts que pyrabbit était a l'abandon mais un successeur peu connu et entretenu par la communauté . Ce dernier facilite grandement les interaction avec rabbitMQ .

**Publish message**

Message will be published to the default exchange with routing key **Client1**, routing it to this queue.

Delivery mode: **1 - Non-persistent**

Headers: **location** = **entrée** String

Properties: **delivery\_mode** = **1** String

Payload: 

```
NomCapteur=Détecteurs de présence
DateCapture=01/01/2021
ValeurCapture=0
```

Publish message

**Get messages**

Warning: getting messages from a queue is a destructive action. ?

Ack Mode: **Nack message requeue true**

Encoding: **Auto string / base64** ?

Messages: **1**

Get Message(s)

Message 1

The server reported 0 messages remaining.

Exchange	(AMQP default)
Routing Key	Client1
Redelivered	0
Properties	delivery_mode: 1 headers: location: entrée
Payload	NomCapteur=Détecteurs de présence
76 bytes	DateCapture=01/01/2021
Encoding: string	ValeurCapture=0

**Publish message**

Message will be published to the default exchange with routing key **Client1**, routing it to this queue.

Delivery mode: **1 - Non-persistent**

Headers: **location** = **cuisine** String

Properties: **NomCapteur=Capteur d'activation de lumière**  
**DateCapture=01/01/2021**  
**ValeurCapture=1**

**Get messages**

Warning: getting messages from a queue is a destructive action. ?

Ack Mode: **Nack message requeue true**

Encoding: **Auto string / base64** ?

Messages: **1**

**Get Message(s)**

Message 1

The server reported 1 messages remaining.

Exchange	(AMQP default)
Routing Key	Client1
Redelivered	•
Properties	delivery_mode: 1 headers: location: cuisine
Payload	NomCapteur=Détecteurs de présence
76 bytes	DateCapture=01/01/2021
Encoding: string	ValeurCapture=1

Créer un script simple (avec le langage de programmation que vous voulez) permettant de lire les messages d'une queue de messages et de les afficher sur la console.



```
from pyrabbit2.api import Client

cl = Client('192.168.99.100:15672', 'rabbitmq', 'rabbitmq')
cl.is_alive()

while(1):
    message=cl.get_messages('Client1', 'Client1')
    if(len(message)>0):
        message=message[0]['payload']
    print(message)
    input()
```

Créer un script simple (avec le langage de programmation que vous voulez) permettant de générer des données pour 1 capteur.

```
from pyrabbit.api import Client

cl = Client('192.168.99.100:15672', 'rabbitmq', 'rabbitmq')
cl.is_alive()

while(1):
    message=input()
    cl.publish('Client1', 'amq.default', 'Client1', message)

    print('Sent :'+message)
```

#### 1.1.4 Automatisation du déploiement

Créer un script (avec le langage de programmation que vous voulez) permettant de déployer automatiquement l'architecture définie en Section 1.2. Votre script doit répondre aux deux situations suivantes (qui sont peut être, suivant votre architecture, la même situation pour vous) : Le monde vient de sécrouter (pour vous, cela veut dire que vous avez perdu tous vos serveurs) et vous devez redéployer l'architecture de base permettant d'ajouter des clients à l'avenir. Le monde s'ouvre à vous, un nouveau client arrive ! Vous devez déployer le nécessaire pour commencer à recevoir les messages des objets connectés de ce client.

```
from pyrabbit2.api import Client

cl = Client('192.168.99.100:15672', 'rabbitmq', 'rabbitmq')
x=len(cl.get_vhost_names())
print("Combient de nouveau client ?")
y=int(input())+x

while(x<y):
    name="Client"+str(x)
    cl.create_vhost(name)
    cl.create_queue(vhost=name, name=name)
    cl.publish('Client1', 'amq.default', 'Client1', 'Welcome On Board')
    x=x+1
```

Cet méthode n'est pas la plus propre mais dans le cas ou aucun utilisateur ne résigne sont abonnement ou si en cas de désabonnement on garde la queue ainsi que le virtual host , il n'y aura pas de problème de chevauchement de nom . On peut supposer dans le monde professionnelle une génération de clé utilisateur unique .

### 1.1.5 Génération automatique de messages

Utiliser le code proposé en Figure 1.2 pour générer des données cohérentes pour tous les capteurs du Fil Rouge.

Afin de générer des données logique , il suffit d'éditer les paramètres dans le fichier yml fourni par M.Courbin

```
(SENZING_RECORD_MIN,SENZING_RECORD_MAX,SENZING_DATA_TEMPLATE)}
```

Je n'ai cependant jamais réussi a envoyer les données générer vers les virtual host comme je le voulais . Pour la suite je vais donc utiliser le virtual host par défaut ("/") afin de simuler le Client1 .

# TP2 Stockage

## MongoDB

Mongo Express : <http://192.168.99.100:8081/>

### 2.1.1 Définition de l'architecture

Définir et expliquer l'architecture de stockage utilisant MongoDB (au niveau matériel et logiciel) Databases / Collections / Serveurs qui vous suivra durant les TPs pour répondre au Fil Rouge. Comment allez-vous stocker ?

J'ai tout d'abord pensé avoir une collection par client, cependant un tel recoupement n'est pas forcément judicieux si l'on prend en compte le besoin de faire des requêtes sur l'ensemble des clients nécessitant de jointure complexe en cas de sharding des données. Il suffit de stocker toutes les données dans la même collection en ajoutant les bons identifiants à chaque valeur.

Définir et expliquer la structure des données (modèle du document stocké sur MongoDB) qui vous suivra durant les TPs pour répondre au Fil Rouge. Quest-ce que vous allez stocker ?

Les données seront écrites avec un  $ID_{client}$ , une date, le nom du capteur ainsi que la valeur mesurée. L'unité de la valeur devrait être trouvable via une autre collection répertoriant les différents capteurs existants.

Définir et expliquer les indexes que vous allez utiliser sur votre base de données MongoDB afin de pouvoir accéder au mieux à vos données. Comment allez-vous y accéder ?

Il est logique d'indexer notre collection sur  $ID_{client}$  car c'est le facteur que nous allons régulièrement utiliser. Il est aussi envisageable d'indexer sur la date pour effectuer des recherches temporelles.

Est-ce que les données d'un client doivent être cloisonnées de celles des autres clients ?

Je n'en vois pas l'intérêt.

Est-ce qu'il faut une collection par capteur, par maison, par client, par quartier... ?

La séparation des données en de nombreuses collections ne me semble pas viable . Sur mongoDB les jointures entre collection shardé n'est pas recommandé et shardé les collections semble logique une fois la quantité de donnée suffisamment large .

Comment gérer les droits d'accès aux databases/collections ?

On peut donner les accès à des views n'affichant que les informations concernant l'utilisateur en question .

Du point de vue des consommateurs de données, sera-t-il plus simple et/ou plus sécurisé d'avoir une ou plusieurs databases/collections de messages ?

Une brèche au niveau de la database ferait dans tous les cas fuiter les informations de tous les clients même si nous cloisonnons les datas

Quel sera l'impact si je perds complètement un nœud de mon architecture matérielle ?

En cas de database shardé et répliqué la perte d'un nœud est nulle car les serveurs secondaires prennent presque instantanément la relève .

Est-ce qu'il faut que je mette toutes les databases sur la même grappe de serveurs, ou il me faut plusieurs grappes de serveurs ?

La logique veut que l'on distribue les shards sur différents serveurs en cas de problème physique sur l'un des serveurs .

Si, demain, j'ai vraiment beaucoup de clients, est-ce que mon architecture sera scalable [18] ? Est-ce qu'elle passera à l'échelle [17] ? Est-ce que mes données seront bien réparties ou est-ce que je risque de créer des nœuds plus remplis que d'autres ?

Dans l'architecture que j'entrevois toutes les données sont dans la même collection mais shardé et donc répartie sur de nombreux serveurs . Ainsi une augmentation du nombre d'utilisateur n'est pas si problématique que cela car il suffit de rajouter des shards supplémentaires . Il reste néanmoins le problème de la vaste quantité de donnée à interroger lors des requêtes , mais cela est sûrement préférable à l'utilisation de jointures .

Si, demain, j'ai des clients internationaux qui ne veulent pas que leurs données sortent de leur pays, est-ce que mon architecture globale reste viable ?

L'architecture devient complexe car avec cet architecture toutes les données sont accessible à l'administrateur de la database , peu importe le pays . Il faudrait le cas échéant créer une collection par pays pour les pays ne souhaitant pas faire sortir les informations en dehors de leurs pays .

Et s'ils veulent héberger cette partie de ma solution directement chez eux pour contrôler plus finement ce qui sort de chez eux ?

comme précédemment mais à plus petite échelle . Je tiens à rappeler que l'interaction avec les données externes peut être un problème mais si le sous-ensemble (ici l'utilisateur) représente une fraction suffisamment petite de l'ensemble des données , on peut considérer son impact sur la moyenne comme négligeable et comparer ces statistiques moyennes à celle du plus grand ensemble .

Est-ce que j'ai besoin que du nom du capteur, de sa valeur et de la date de relevé de la valeur ? Quid de l'unité ?

On peut penser à une collection en parallèle avec l'ensemble des capteurs , une courte description et l'unité de la valeur donnée par le capteur .

Que se passe-t-il si un capteur doit être remplacé ? Est-ce que je garderais le même nom de capteur ? Et s'il ne produit pas les données avec la même unité ?

Il faudra enregistrer un nouveau capteur avec le nom du nouveau capteur afin d'obtenir la bonne unité en provenance de l'autre collection .

Quelles seront les requêtes les plus courantes pour accéder à mes données ? Est-ce qu'il faut que j'ajoute un index sur les dates ? Sur les noms de capteurs ? Sur les valeurs ? Sur chaque paramètre ?

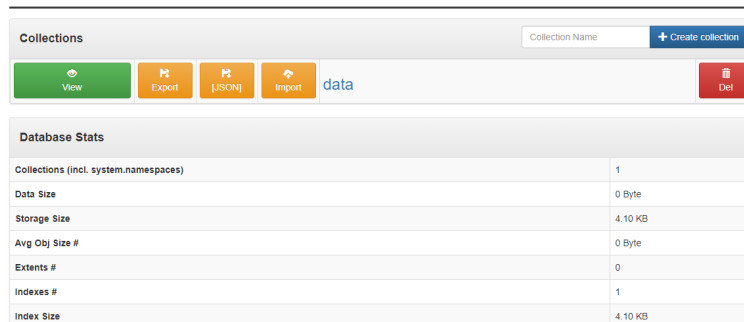
Je pense que les requêtes les plus courantes seront des moyennes glissantes (personnelles et globales) afin de savoir si l'on consomme plus que les autres . Les noms de capteurs n'ont d'intérêt que de donner l'unité mesurée . L'indexation sur les valeurs n'a pas d'intérêt car nous faisons des moyennes . L'index sur les dates semble intéressant car les requêtes basées sur les dates vont être très fréquentes .

### 2.1.2 Déployer votre architecture

Déployer un serveur MongoDB à partir du docker-compose proposé à la Figure

Implémenter votre architecture de stockage de données à partir de l'interface web proposée par Mongo-Express (mais si, vous allez la trouver...). Vous pouvez sinon utiliser MongoDB Compass comme interface de gestion

#### Viewing Database: FilRouge



Collections (incl. system.namespaces)	1
Data Size	0 Byte
Storage Size	4.10 KB
Avg Obj Size #	0 Byte
Extents #	0
Indexes #	1
Index Size	4.10 KB

### 2.1.3 Découvrir l'envoi et la réception de données

Créer un script simple (avec le langage de programmation que vous voulez) permettant d'envoyer des données pour 1 capteur sur une collection.

```
#!/usr/bin/env python
from pymongo import MongoClient
import datetime, random, time

client = MongoClient('mongodb://192.168.99.100:27017/')
db = client.FilRouge
collection = db.data
value=20
while(1):
    post = {"SENSOR":"Templ","DATE":datetime.datetime.utcnow(), "VALUE":value}
    post_id = collection.insert_one(post).inserted_id
    print(post_id)
    value=value+random.uniform(-0.5, 0.5)
    time.sleep(1)
```

Créer un script simple (avec le langage de programmation que vous voulez) permettant de lire les données d'une collection et de les afficher sur la console.

```
#!/usr/bin/env python
from pymongo import MongoClient
import datetime, random, time, pprint

client = MongoClient('mongodb://192.168.99.100:27017/')
db = client.FilRouge
collection = db.data
pprint.pprint(collection.find_one())
time.sleep(1)
```

Créer des scripts plus avancés permettant de : (a) récupérer la dernière valeur connue d'un capteur. (b) calculer la moyenne des valeurs d'un capteur entre deux dates fixées. (c) récupérer la valeur minimale d'un capteur entre deux dates fixées.

```
#!/usr/bin/env python
from pymongo import MongoClient
import datetime, time, pprint
from bson import ObjectId

client = MongoClient('mongodb://192.168.99.100:27017/')
db = client.FilRouge
collection = db.data

pipeline = [{"$match": {"SENSOR": "Temp1"}}, {"$sort": {"DATE": -1}}, {"$limit": 1}]
pprint.pprint(list(collection.aggregate(pipeline)))

d1 = datetime.datetime.utcnow()
time.sleep(10)
d2 = datetime.datetime.utcnow()

pipeline = [{"$match": {"DATE": {"$gte": d1, "$lte": d2}, "SENSOR": "Temp1"}}, {"$group": {"_id": {"SENSOR": "SENSOR"}, "average": {"$avg": "$VALUE"}}}]
pprint.pprint(list(collection.aggregate(pipeline)))

pipeline = [{"$match": {"DATE": {"$gte": d1, "$lte": d2}, "SENSOR": "Temp1"}}, {"$group": {"_id": {"SENSOR": "SENSOR"}, "minimum": {"$min": "$VALUE"}}}]
pprint.pprint(list(collection.aggregate(pipeline)))
```

### 2.1.4 Automatisation du déploiement

Créer un script (avec le langage de programmation que vous voulez) permettant de déployer automatiquement l'architecture définie en Section 1.2. Votre script doit répondre aux deux situations suivantes (qui sont peut être, suivant votre architecture, la même situation pour vous) : Le monde vient de sécrouter (pour vous, cela veut dire que vous avez perdu tous vos serveurs) et vous devez redéployer l'architecture de base permettant d'ajouter des clients à l'avenir. Le monde s'ouvre à vous, un nouveau client arrive ! Vous devez déployer le nécessaire pour commencer à recevoir les messages des objets connectés de ce client.

```
#!/usr/bin/env python
from pymongo import MongoClient
from bson.objectid import ObjectId

client = MongoClient('mongodb://192.168.99.100:27017/')
db = client.FilRouge
collection = db.data

post = {"Faux": "Faux"}
post_id = collection.insert_one(post).inserted_id
x=post_id
result = collection.delete_one({'_id': ObjectId(x)})
print(result)
```

### 2.1.5 Génération automatique de messages

Reprenons le générateur de mock data [15] et votre RabbitMQ du TP précédent. Un projet de Marcel Maatkamp [4], basé sur une librairie NodeJS de Ruquay K Calloway [1] devrait vous permettre de connecter une queue de messages à une collection. Un exemple vous est donné en Figure 1.2, n'hésitez pas à l'adapter avec la documentation. 10. Utiliser le code proposé en Figure 1.2 pour envoyer des données cohérentes générées

précédemment sur une queue RabbitMQ vers une collection MongoDB pour quelques capteurs du Fil Rouge.

```
chuve@PC-Xavier MINGW64 ~/Desktop/shared/docker/RabbitToMongo
$ docker-compose up
Starting rabbitomongo_amqp2mongo1_1 ... done
Attaching to rabbitomongo_amqp2mongo1_1
amqp2mongo1_1 | [Fri Jan 15 2021 07:17:27 GMT+0000 (UTC)] Saving to 'data' in 'mongodb://mongo/FilRouge'
amqp2mongo1_1 | [Fri Jan 15 2021 07:17:27 GMT+0000 (UTC)] Consuming Queue: Client1 (cTag=amq.ctag-qgXW01f4wvcDV-yu4t4_Tg)
```



# TP3 Data Routing

## Nifi

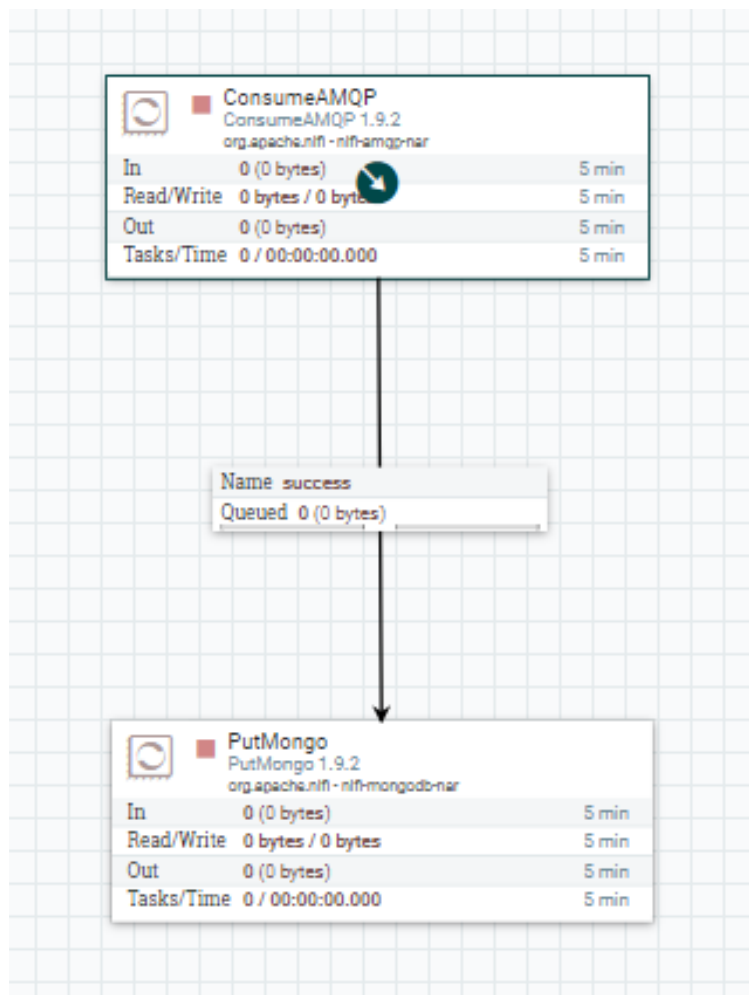
NiFi : <http://192.168.99.100:8083/nifi/>

### 3.1.1 Déployer Nifi

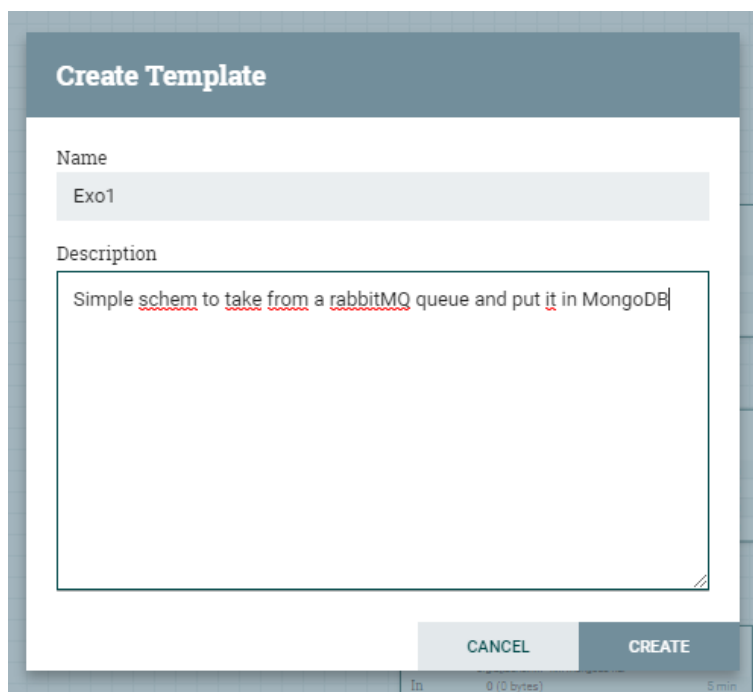
Déployer un serveur Nifi à partir du docker-compose proposé à la Figure 1.1

```
MINGW64 ~/c:/Users/Share/Desktop/Share/docker7/nifi
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1895)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1894)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1812)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1178)
[1]  at org.apache.nifi.amp.processors.AbstractANQPProcessor.createConnection(AbstractANQPProcessor.java:252)
[1]  ... 14 common frames omitted
[1]  2021-01-15 07:44:22,786 MAIN [RabbitMQ Error On Write Thread] c.r.c.impl.forgivingExceptionHandler An unexpected connection driver error occurred (Exception me
[1]  ssaage: Socket closed)
[1]  2021-01-15 07:44:22,786 MAIN [ANQP Connection 192.168.99.100:5672] c.r.c.impl.forgivingExceptionHandler An unexpected connection driver error occurred (Excepti
[1]  on message: Socket closed)
[1]  2021-01-15 07:44:23,514 MAIN [RabbitMQ Error On Write Thread] c.r.c.impl.forgivingExceptionHandler An unexpected connection driver error occurred (Exception me
[1]  ssaage: Socket closed)
[1]  2021-01-15 07:44:23,519 MAIN [ANQP Connection 192.168.99.100:5672] c.r.c.impl.forgivingExceptionHandler An unexpected connection driver error occurred (Excepti
[1]  on message: Socket closed)
[1]  2021-01-15 07:44:23,517 ERROR [Timer-Driven Process Thread-3] o.a.nifi.amp.processors.ConsumerANQP ConsumerANQP[d47f625e-92d4-3d86-f43d-8da99f9ec9f6] Consum
[1]  erANQP[d47f625e-92d4-3d86-f43d-8da99f9ec9f6] failed to process session due to java.lang.IllegalStateException: failed to establish connection with ANQP Broker: com.ra
[1]  bbitmq.client.ConnectionFactory@25082f65; Processor Administratively Vielded for 1 sec: java.lang.IllegalStateException: Failed to establish connection with ANQP Broker
[1]  c.com.rabbitmq.client.ConnectionFactory@25082f65
[1]  java.lang.IllegalStateException: failed to establish connection with ANQP Broker: com.rabbitmq.client.ConnectionFactory@25082f65
[1]  at org.apache.nifi.amp.processors.AbstractANQPProcessor.createConnection(AbstractANQPProcessor.java:155)
[1]  at org.apache.nifi.amp.processors.AbstractANQPProcessor.createSource(AbstractANQPProcessor.java:181)
[1]  at org.apache.nifi.amp.processors.AbstractANQPProcessor.onTrigger(AbstractANQPProcessor.java:156)
[1]  at org.apache.nifi.processor.AbstractProcessor.onTrigger(AbstractProcessor.java:27)
[1]  at org.apache.nifi.controller.StandardProcessorNode.onTrigger(StandardProcessorNode.java:1182)
[1]  at org.apache.nifi.controller.tasks.ConnectableTask.run(ConnectableTask.java:180)
[1]  at org.apache.nifi.controller.scheduling.TimerDrivenSchedulingAgent$1.run(TimerDrivenSchedulingAgent.java:117)
[1]  at org.apache.nifi.engine.FlowEngine$2.run(FlowEngine.java:119)
[1]  at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
[1]  at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:288)
[1]  at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$380(ScheduledThreadPoolExecutor.java:180)
[1]  at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:204)
[1]  at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
[1]  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
[1]  at java.lang.Thread.run(Thread.java:748)
[1]  Caused by: com.rabbitmq.client.AuthenticationFailureException: ACCESS_REFUSED - Login was refused using authentication mechanism PLAIN. For details see the br
[1]  ower logFile.
[1]  at com.rabbitmq.client.impl.AMQConnection.start(AMQConnection.java:362)
[1]  at com.rabbitmq.client.impl.recovery.RecoveryAwareAMQConnectionFactory.newConnection(RecoveryAwareAMQConnectionFactory.java:64)
[1]  at com.rabbitmq.client.impl.recovery.AutorecoveringConnection.init(AutorecoveringConnection.java:156)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1895)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1894)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1812)
[1]  at com.rabbitmq.client.ConnectionFactory.newConnection(ConnectionFactory.java:1178)
[1]  at org.apache.nifi.amp.processors.AbstractANQPProcessor.createConnection(AbstractANQPProcessor.java:252)
[1]  ... 14 common frames omitted
```

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



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.



**Create Template**

Name  
Exo1

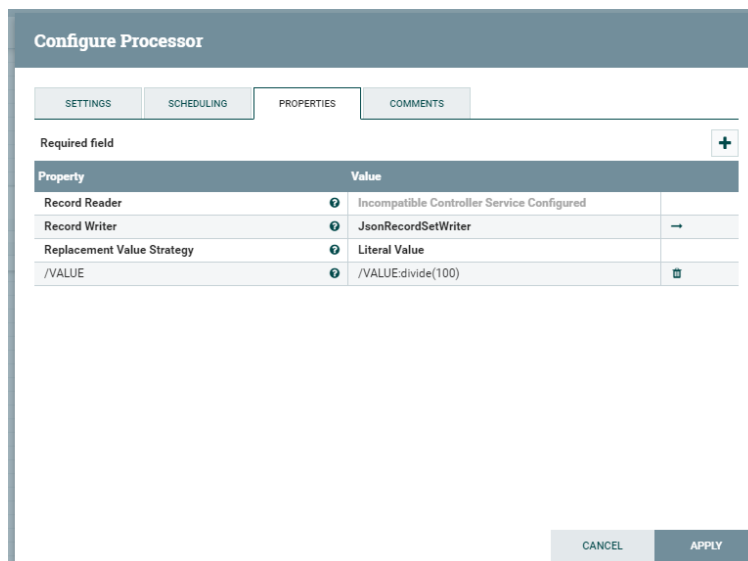
Description  
Simple schem to take from a rabbitMQ queue and put it in MongoDB

CANCEL CREATE

In 0 (0 bytes) 5 min

Imaginez maintenant... lun 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 nait 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'ai divisé les valeurs de température par 100 plutôt que de les multiplier par 1000 car avoir des valeurs entre 500 000 et 700 000 ne me semblait vraiment pas logique alors qu'entre 5 et 7, fait pas chaud mais c'est viable .



The 'Configure Processor' window has four tabs: SETTINGS, SCHEDULING, PROPERTIES, and COMMENTS. The PROPERTIES tab is active, showing a 'Required field' section with a '+' icon. Below this is a table with two columns: 'Property' and 'Value'.

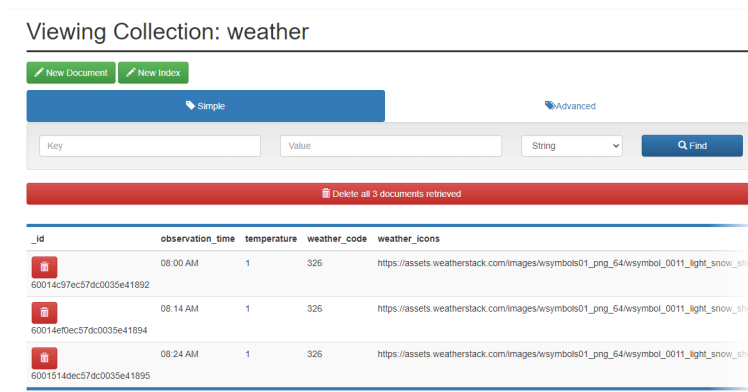
Property	Value
Record Reader	Incompatible Controller Service Configured
Record Writer	JsonRecordSetWriter
Replacement Value Strategy	Literal Value
/VALUE	/VALUE:divide(100)

At the bottom right are 'CANCEL' and 'APPLY' buttons.

### 3.1.2 Récupérer les données d'une API ?

Créer une nouvelle base de données MongoDB (par exemple : "weather") puis une collection pour la ville (par exemple : "paris").

Les requête lancer par le processus putMongo créer les éléments manquant rendent cet étape redondante .



The 'Viewing Collection: weather' interface shows a 'Simple' view selected. It includes a search bar with 'Key' and 'Value' fields, a 'String' dropdown, and a 'Find' button. A red banner indicates 'Delete all 3 documents retrieved'. Below is a table of documents:

_id	observation_time	temperature	weather_code	weather_icons
60014c97ec57dc0035e41892	08:00 AM	1	326	https://assets.weatherstack.com/images/wsymbols01_png_64/wsymbol_0011_light_snow_showing.png
60014e0ec57dc0035e41894	08:14 AM	1	326	https://assets.weatherstack.com/images/wsymbols01_png_64/wsymbol_0011_light_snow_showing.png
6001514dec57dc0035e41895	08:24 AM	1	326	https://assets.weatherstack.com/images/wsymbols01_png_64/wsymbol_0011_light_snow_showing.png

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}`

Afin de ne pas user l'api (et mon accès limité) j'ai ralenti le processus avec 30s de sommeil entre chaque tâche durant les phase de teste .

The image shows two screenshots of the Apache NiFi web interface. The top screenshot is the 'Processor Details' window, and the bottom is the 'Configure Processor' window.

**Processor Details**

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Scheduling Strategy ⓘ Timer driven Run Duration ⓘ 00:00:00.000

Concurrent Tasks ⓘ 1 Run Schedule ⓘ 10 min

Execution ⓘ All nodes

OK

**Configure Processor**

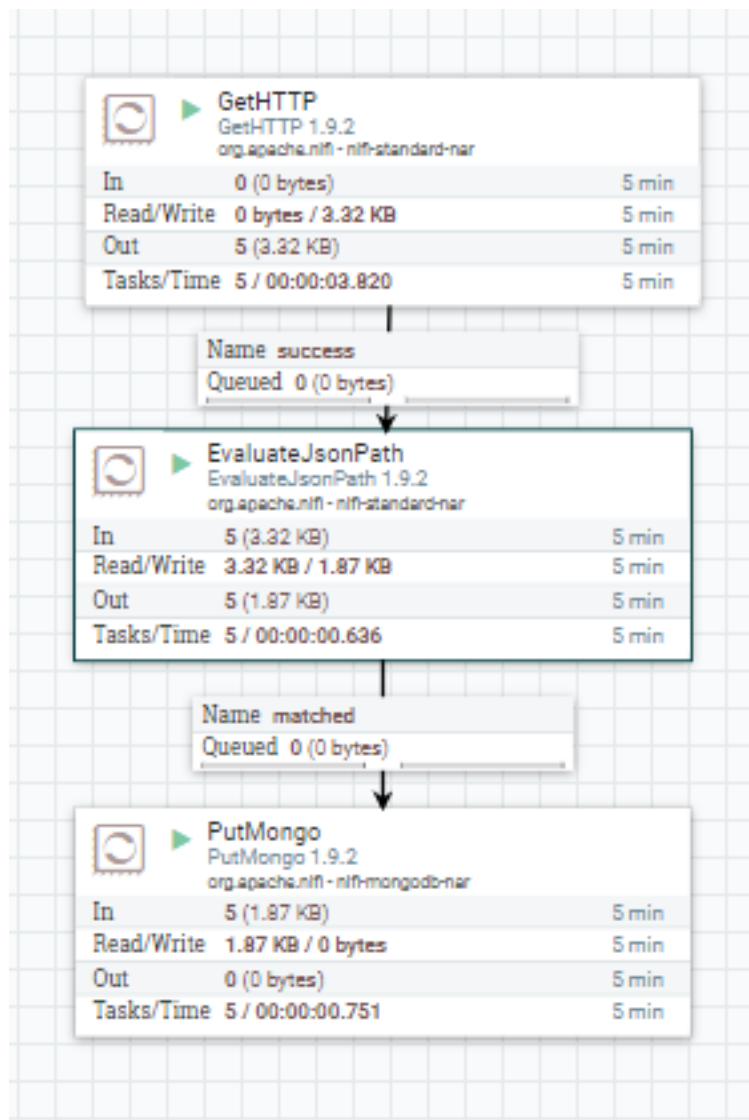
SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field +

Property	Value
URL	http://api.weatherstack.com/current?access_key=ab9...
Filename	\$(.uuid)
SSL Context Service	No value set
Username	No value set
Password	No value set
Connection Timeout	30 sec
Data Timeout	30 sec
User Agent	No value set
Accept Content-Type	No value set
Follow Redirects	false
Redirect Cookie Policy	default
Proxy Configuration Service	No value set
Proxy Host	No value set
Proxy Port	No value set

CANCEL APPLY

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

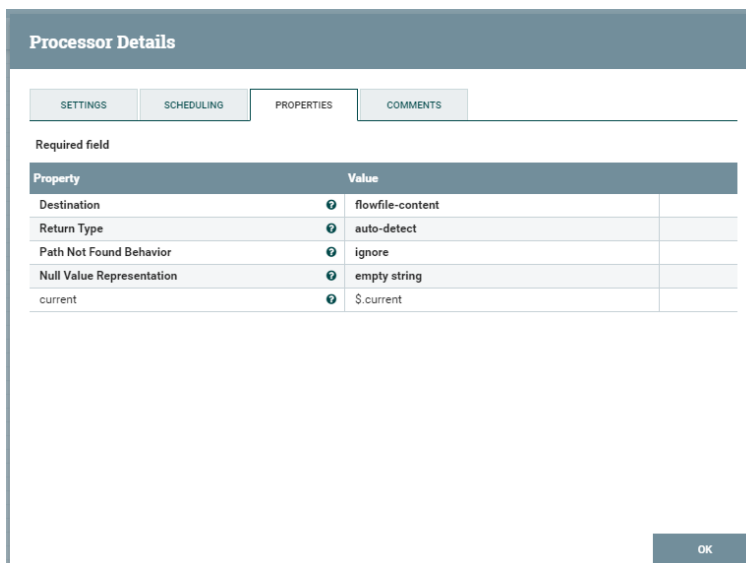


Viewing Collection: weather

Delete all 3 documents retrieved

_id	observation_time	temperature	weather_code	weather_icons
60014c97ec57dc0035e41892	08:00 AM	1	325	https://assets.weatherstack.com/images/symbols01_png_64/symbol_0011_light_snow_showing.png
60014ef8ec57dc0035e41894	08:14 AM	1	325	https://assets.weatherstack.com/images/symbols01_png_64/symbol_0011_light_snow_showing.png
6001514dec57dc0035e41895	08:24 AM	1	325	https://assets.weatherstack.com/images/symbols01_png_64/symbol_0011_light_snow_showing.png

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.



**Processor Details**

SETTINGS SCHEDULING **PROPERTIES** COMMENTS

Required field

Property	Value
Destination	flowfile-content
Return Type	auto-detect
Path Not Found Behavior	ignore
Null Value Representation	empty string
current	\$.current

OK

### 3.1.3 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 un 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 :

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 ce serveur FTP à partir de votre hôte (par exemple, après avoir installé l'outil

La joie de Windows m'empêche de pouvoir apprécier l'outil formidable qu'est sûrement filezilla .

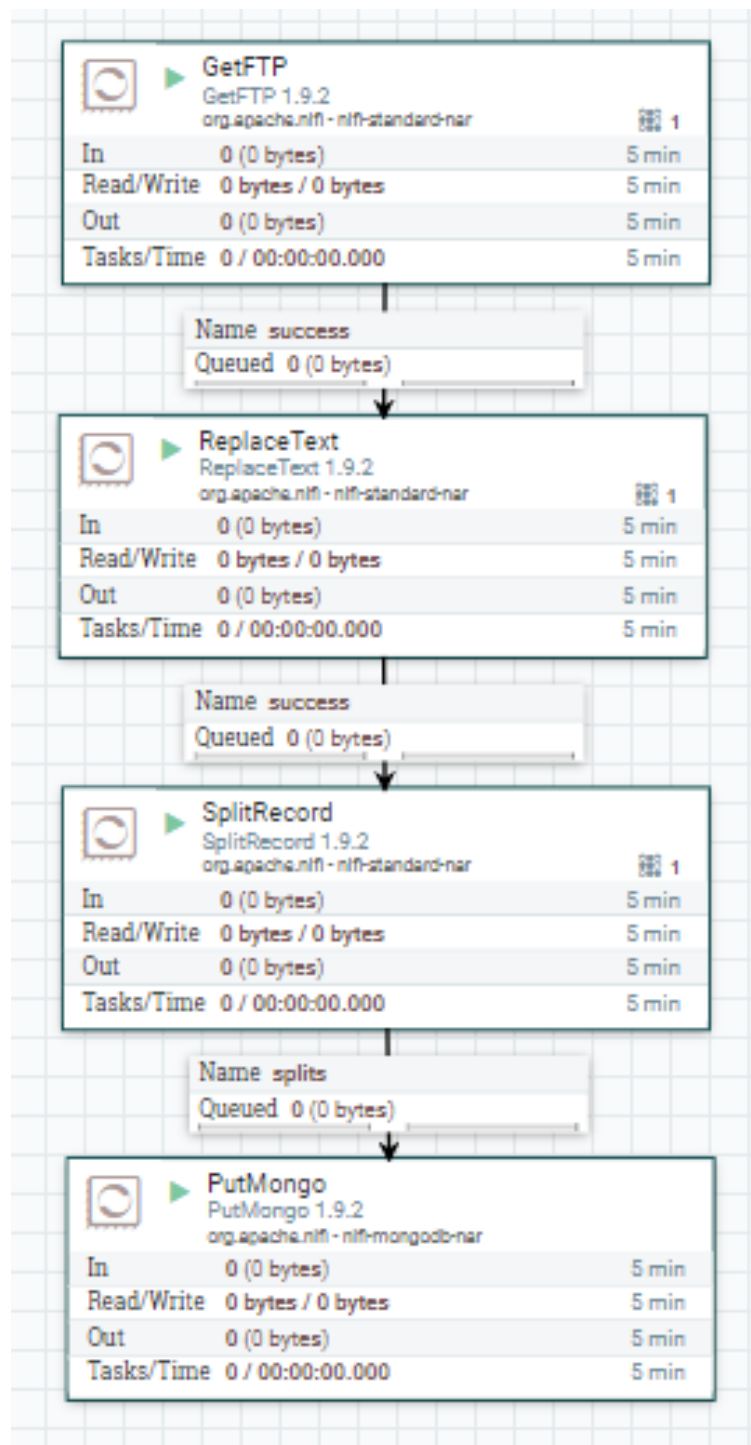
Créer un fichier CSV contenant les informations présentées en Figure

Presse-papiers			Police			Align
A2			10/10/2019 09:00:00			
	A	B	C	D	E	F
1	DATE	SENSOR	val			
2	#####	Conso1	2666,214			
3	#####	Conso1	2137,396			
4	#####	Conso1	2735,651			
5	#####	Conso1	1553,432			
6	#####	Conso1	3590,205			
7	#####	Conso1	2466,496			
8	#####	Conso1	2971,633			
9	#####	Conso1	3964,156			
10	#####	Conso1	1383,909			
11	#####	Conso1	3195,508			
12	#####	Conso1	1175,626			
13	#####	Conso1	2386,021			
14	#####	Conso1	2719,344			
15	#####	Conso1	2740,258			
16	#####	Conso1	1562,102			
17	#####	Conso1	1914,721			
18	#####	Conso1	1455,322			
19	#####	Conso1	3127,21			
20	#####	Conso1	1804,493			
21	#####	Conso1	1004,901			
22	#####	Conso1	2361,639			
23	#####	Conso1	2656,269			
24	#####	Conso1	1996,681			

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.





Viewing Collection: conso

[New Document](#) [New Index](#)

[Simple](#) [Advanced](#)

Key  Value  String

Delete all 30 documents retrieved

[First](#) [Prev](#) [Next](#) [Last](#)

_id	DATE;SENSOR;VALUE
5f7c94336f5e5d0032ebf768	10/10/19 13:00;Conso1;1565.716749
5f7c94336f5e5d0032ebf769	10/10/19 13:30;Conso1;2478.743919
5f7c94336f5e5d0032ebf76a	10/10/19 12:00;Conso1;1804.493063

Super ! Mais, heu... le nom de la colonne des valeurs nest 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.

Configure Processor

SETTINGS SCHEDULING PROPERTIES COMMENTS

Required field [+](#)

Property	Value
Search Value	val
Replacement Value	VALUE
Character Set	UTF-8
Maximum Buffer Size	1 MB
Replacement Strategy	Regex Replace
Evaluation Mode	Entire text

[CANCEL](#) [APPLY](#)

# TP4 Présentation

## Introduction

### 4.1.1 Présenter... mais quoi ?

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.

Si je suis client, quelles sont les informations pertinentes que je voudrais voir tous les jours ? (Data storytelling...)

Dans le contexte des économies d'énergie on peut afficher la différence de consommation énergétique entre hier et aujourd'hui ou encore mieux entre hier et l'an dernier même jour . Peut être un comparatif avec ses voisins ?

Si je vois une information étrange, comment je voudrais pouvoir fouiller la donnée, quelles représentations m'aideraient à comprendre une situation ? (Fouille de données)

Un graphique permet une vue globale mais peu précise , les logs de certains événements peuvent aussi être intéressants pour certains .

Si je suis un gestionnaire (énergie ? sécurité ?) et que j'ai plusieurs clients, quelles informations seraient pertinentes à voir ?

Des comparatifs entre les différents individus ainsi que des historiques d'événements particuliers (dépassement d'un seuil préalablement fixé ou autre) .

## InfluxDB

Cronograph : <http://192.168.99.100:8087/>

### 4.2.1 Déployer... InfluxDB [4]

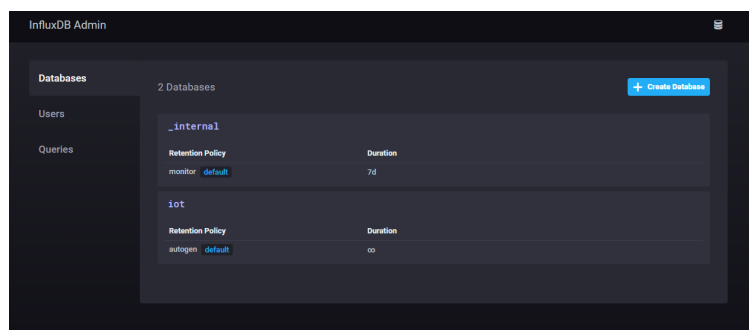
L'outil que nous allons utiliser, Grafana, ne permet pas de se connecter simplement à une base de données MongoDB [12]. Nous allons donc commencer par déployer une nouvelle base de données orientée Time Series, InfluxDB [4]. Pour cela, vous devez :

```

MINGW64 ~/c:/Users/roune/Desktop/shard/docker/influxdb
[httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=epoch=ms&q=SHOW%20ABASES&rp= HTTP/1.1" 200 115 "-" "Go-http-client/1.1" 812c0e
[InfluxDB_1] ts=2021-01-15T08:41:47.813522Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW USERS"
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=epoch=ms&q=SHOW%20USERS&rp= HTTP/1.1" 200 108 "-" "Go-http-client/1.1" 812c0f1-5
76b-11eb-8864-0242ac150802 4468
[InfluxDB_1] ts=2021-01-15T08:41:47.844957Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW GRANTS FOR xavier"
[InfluxDB_Chrono_1] time="2021-01-15T08:41:47Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time=1.88238
7ms status=204
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=epoch=ms&q=SHOW%20GRANTS%20FOR%20xavier%20&rp= HTTP/1.1" 200 96 "-" "Go-http-client/1.1" 812daf423-576b-11eb-8864-0242ac150802 2249
[InfluxDB_1] ts=2021-01-15T08:41:47.845512Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW RETENTION POLICIES ON internal"
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=internal&epoch=ms&q=SHOW%20RETENTION%20POLICIES%20ON%20internal%20&rp= HTTP/1.1" 20
0 153 "-" "Go-http-client/1.1" 812db07a-576b-11eb-8864-0242ac150802 1822
[InfluxDB_1] ts=2021-01-15T08:41:47.864291Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW RETENTION POLICIES ON iot"
[InfluxDB_Chrono_1] time="2021-01-15T08:41:47Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time=13.24219ms sta
tus=200
[InfluxDB_Chrono_1] time="2021-01-15T08:41:47Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time=18.28335ms sta
tus=200
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=iot&epoch=ms&q=SHOW%20RETENTION%20POLICIES%20ON%20iot%20&rp= HTTP/1.1" 200 149 "-" "G
o-http-client/1.1" 813914eb-576b-11eb-8864-0242ac150802 1813
[InfluxDB_1] ts=2021-01-15T08:41:47.909401Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW DATABASES"
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=epoch=ms&q=SHOW%20DATABASES&rp= HTTP/1.1" 200 115 "-" "Go-http-client/1.1" 81388e
ac-576b-11eb-8864-0242ac150802 1610
[InfluxDB_1] ts=2021-01-15T08:41:47.912352Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW RETENTION POLICIES ON internal"
[InfluxDB_1] [httpd] 172.21.0.1 - - [15/Jan/2021:08:41:47 +0000] "POST /query?db=internal&epoch=ms&q=SHOW%20RETENTION%20POLICIES%20ON%20internal%20&rp= HTTP/1.1" 20
0 153 "-" "Go-http-client/1.1" 8138f34b-576b-11eb-8864-0242ac150802 1243
[InfluxDB_1] ts=2021-01-15T08:41:47.915892Z lvl=info msg="executing query" log_id=8Rlntw000 service=query query="SHOW RETENTION POLICIES ON iot"
[InfluxDB_Chrono_1] time="2021-01-15T08:41:47Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="87.326µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:41:47Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time=12.453972ms sta
tus=200
[InfluxDB_Chrono_1] time="2021-01-15T08:41:51Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="26.782µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:02Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="35.727µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:13Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="48µs" status=
200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:24Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="58.486µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:35Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61693" response_time="51.5129µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:46Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61700" response_time="38.293µs" statu
s=200
[InfluxDB_Chrono_1] time="2021-01-15T08:42:57Z" level=info msg="Response: OK" component=server method=GET remote_addr="192.168.99.1:61701" response_time="58.56µs" statu
s=200
  
```

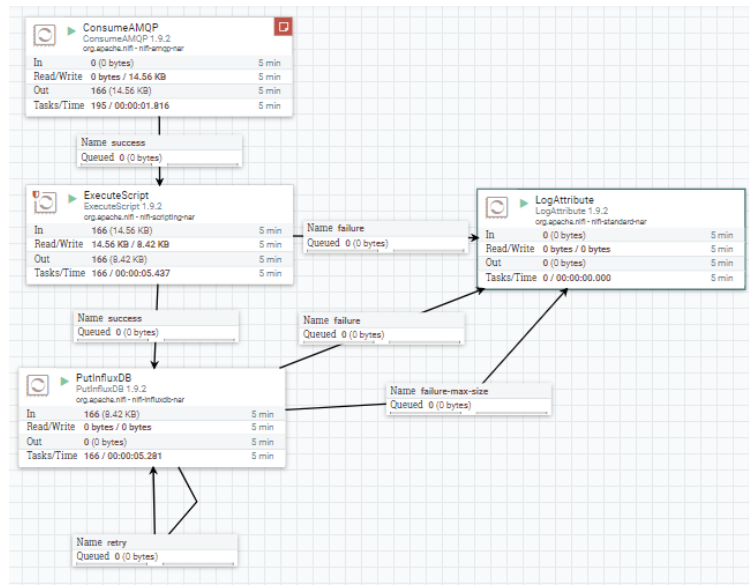
Déployer un serveur InfluxDB [6] avec une interface graphique Chronograf [5] à partir du docker-compose proposé à la Figure 1.1.

Se connecter à l'interface Chronograf pour créer une base de données : CREATE DATABASE "iot"



Les données générées durant le TP1 qui sont envoyées sur une queue RabbitMQ sont sous format JSON (exemple : "DATE" : "1570985503", "SENSOR" : "SDBPuiss", "VA-LUE" : "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.



Finalement, comme présenté en Figure 1.3, ajouter un processeur Nifi PutInfluxDB [11] permettant d'envoyer les données vers une base de données InfluxDB.

J'aime windows qui me force à chercher la petite bête à chaque fois, ici ainsi que dans une nombreuse partie de ce TP, des valeurs que certains n'auront sûrement jamais vues. C'était caler sur localhost ce que mon windows n'accepte absolument pas.

**Processor Details**

SETTINGS SCHEDULING PROPERTIES COMMENTS

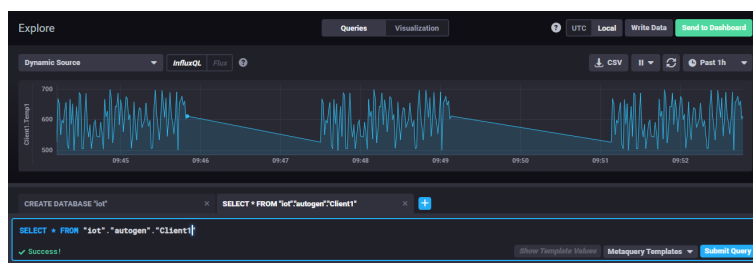
Required field

Property	Value
Database Name	iot
InfluxDB connection URL	http://192.168.99.100:8086/
InfluxDB Max Connection Time Out (seconds)	3 seconds
Username	xavier
Password	Sensitive value set
Character Set	UTF-8
Consistency Level	One
Retention Policy	autogen
Max size of records	1 MB

OK

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...

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.



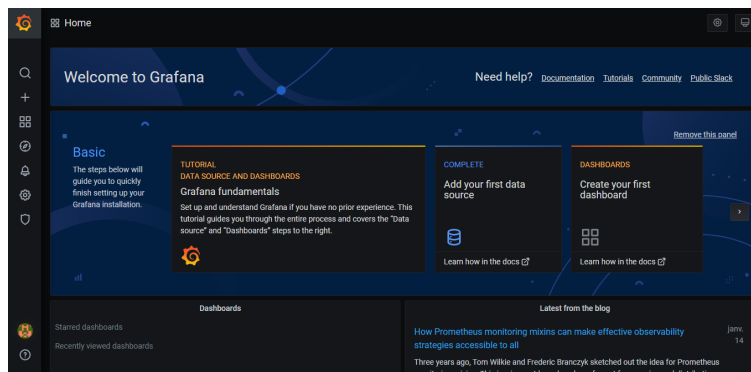
## Grafana

Grafana : <http://192.168.99.100:8084/>

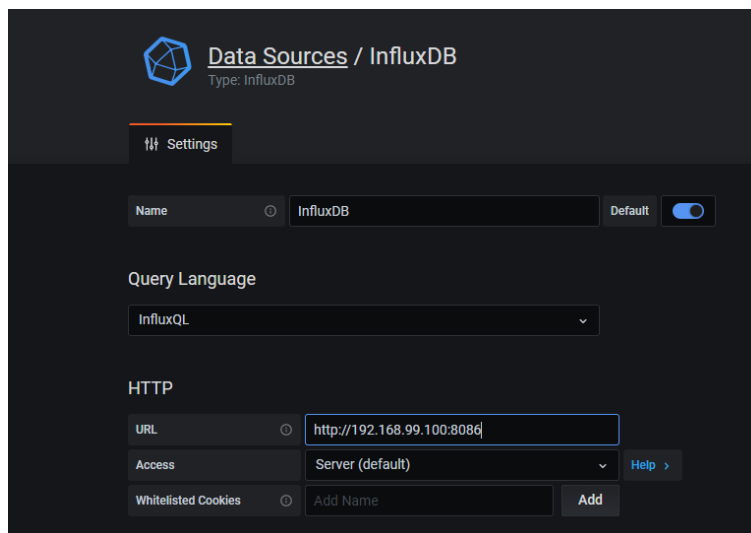
- Username : admin
- Password : secret

#### 4.3.1 Déployer... Grafana [3]

Déployer un serveur Grafana [3] à partir du docker-compose proposé à la Figure 1.5.



Une fois connecté à l'interface, ajouter une source de données InfluxDB en configurant bien l'URL et la Database.



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.





# TP5 Principe dAPI

## OpenAPI

Swagger : `http ://192.168.99.100 :8080/v1/ui/`

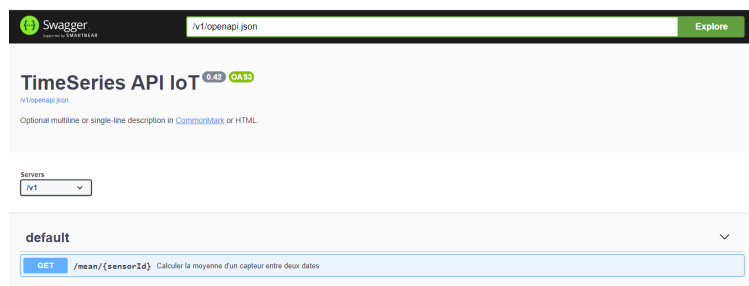
### 5.1.1 Prise en main de Swagger Codegen [6]

Vous pouvez utiliser la commande basée sur le Docker préparé par Swagger : `docker run --rm -v $PWD`

pour : générer un code à partir du fichier `"timeseriesiot.yaml"` (qui contiendrait par exemple... le code proposé)

Vous avez donc maintenant, dans le dossier `"out/python-ts-iot"`, une structure de fichiers permettant de déployer une API basée sur votre définition OpenAPI et s'appuyant sur le framework Python-Flask. (Non, n'applaudissez pas encore, c'est encore plus beau après...) Nous allons donc maintenant déployer cette API avec... Docker! (Bah non, pourquoi vous fuyez maintenant?) (a) Vous devriez voir un fichier `"Dockerfile"` dans le dossier `"out/python-ts-iot"` généré. (b) À partir de la Figure 1.2, créez un fichier `"docker-compose.yml"` dans le dossier pour faciliter la création du Docker et son ajout à notre réseau virtuel `"iot-labs"`. (c) Créez l'image Docker avec la commande `docker-compose build` (d) Lancez le service Docker avec la commande `docker-compose up -d` (e) Ouvrez votre navigateur à l'adresse `http ://localhost :8080/v1/ui` pour voir cette belle interface et jouer avec votre API déployée! Elle devrait ressembler à la Figure 1.3.

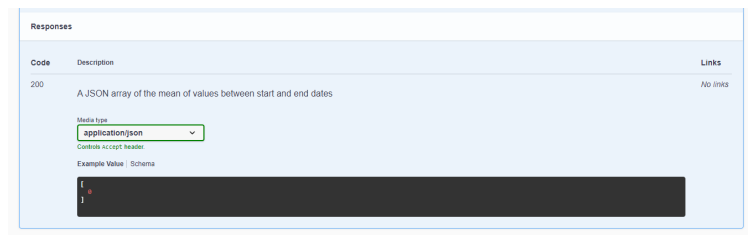
- `docker-compose build`
- `docker-compose up -d`
- `http ://192.168.99.100 :8080/v1/ui/`
- `docker-compose up -d --force-recreate`



### 5.1.2 How to "do some magic" ? Remplir le code de Swagger Codegen

Ouvrez le fichier "controllers/default\_controller.py", c'est lui qui contient les fonctions qui sont appelées quand

Modifiez cette ligne de retour par `return 'Sensor : ' + sensor_id + '[start : end] : [' + str(start_date) + ' : ' + str(end_date) + ']'`



### 5.1.3 Et maintenant... on fait le lien avec le TP2 ?

Ajouter dans la fonction "mean\_sensor\_id\_get" le code permettant de requêter votre base de données MongoDB et calculer la moyenne des valeurs d'un capteur entre deux dates fixées.

```
import datetime
import six
from pymongo import MongoClient
from TimeSeriesIoT import util

def mean_sensor_id_get(sensor_id, start_date=None, end_date=None): # noqa: E501
    """Calculer la moyenne d'un capteur entre deux dates

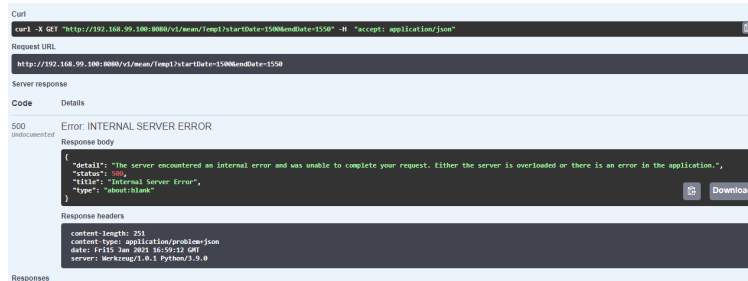
    Optional extended description in CommonMark or HTML. # noqa: E501

    :param sensor_id: String Id of the sensor to get
    :type sensor_id: str
    :param start_date: Integer/timestamp of the start date
    :type start_date: int
    :param end_date: Integer/timestamp of the end date
    :type end_date: int

    :rtype: List(int)
    """
    client = MongoClient('mongodb://192.168.99.100:27017/')
    db = client.FillRough
    collection = db.data
    pipeline = [{"$match": {"DATE": {"$gte": start_date, "$lte": end_date}, "SENSOR": "Temp1"}},
                {"$group": {"_id": {"$SENSOR": "$SENSOR"}, "average": {"$avg": "$VALUE"}}}]
    x=list(collection.aggregate(pipeline))
    return 'Sensor : ' + sensor_id + ' [start: ' + str(start_date) + ' : ' + str(end_date) + ']' Average : ' + x[0]['average']
```

Modifier votre définition OpenAPI et le code généré pour ajouter un chemin permettant de récupérer la valeur minimale d'un capteur entre deux dates fixées à partir des données de votre base de données MongoDB.

Je n'ai pas réussi à résoudre cet erreur. Je pense qu'elle est due à mon utilisation de windows créant une incompatibilité entre la base de données MongoDB et l'image servant à l'API



Modifier votre définition OpenAPI et le code généré pour ajouter un chemin permettant de récupérer la dernière valeur connue d'un capteur à partir des données de votre base de données MongoDB.

J'ai tout de même mis en place l'architecture demandée afin de visualiser le potentiel de Swagger.

