

# Rapport TPs

Développement d'applications et webservices  
pour l'IOT

Github: [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---  
Fil-rouge](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge)

12-01-2020  
ESILV IBO A5

BOUCAUD Stéphane

# SOMMAIRE

Introduction.....	4
L'entreprise Home Energy Awareness.....	4
Clients.....	4
Capteurs du Client1 (5 pièces).....	4
Capteurs du Client2 (2 pièces).....	5
Format des données des capteurs.....	5
Détails Projet.....	7
Ingestion (TP1).....	7
Description générale du problème.....	7
Solutions existantes.....	7
Introduction.....	7
Patterns d'intégration.....	8
Solutions de message brokers.....	8
Protocoles.....	8
Choix de la solution.....	9
Message broker.....	9
Architecture Exchanges/Queues et containers.....	9
Implementation.....	10
Docker-compose.....	10
Interactions avec le container.....	11
Paramétrage Rabbit-MQ (Graphique).....	12
Premiers messages (Manuel sur RabbitMQ).....	17
Envoi/Reception messages via Python.....	18
Automatisation du déploiement.....	20
Génération automatique de messages.....	23
Stockage (TP2).....	26
Description du problème.....	26
Choix de la solution.....	26
Base de données.....	26
Architecture.....	26
Implémentation.....	28
Docker compose.....	28
Architecture mongodb via mongo express.....	29
Architecture MongoDB via script python.....	29
Script de lecture simple de données (python).....	30
Envoi de données pour un capteur (python).....	30
Scripts avancés de requêtes (avec données du json generé).....	31
Connection avec Rabbitmq.....	32
Scripts avancés de requêtes (avec données retirées depuis la queue rabbitmq).....	33
Indexes.....	36
Etat du contenu des collections.....	36
Cle de sharding.....	37
Datarouting (TP3).....	37
Description du problème.....	37
Choix de la solution.....	37

Implémentation.....	37
Docker-compose.....	37
Templates nifi.....	38
Consommation Rabbitmq / Push dans la collection mongodb.....	38
JSON Path evaluation.....	42
Nifi Expression language.....	43
Nifi RecordPath: Domain Specific Language (DSL).....	44
Modifier valeur en utilisant un Processeur UpdateRecord.....	44
Connection avec weatherstack.....	48
Récupération CSV depuis Serveur FTP.....	52
Présentation (TP4).....	56
Choix de solution.....	56
Implémentation.....	56
Docker-compose.....	56
Première config de influxDB.....	57
Configuration nifi.....	58
Premier affichage de données.....	59
Grafana.....	60
API (TP5).....	63
Conclusion générale.....	67
Questions Pour le professeur.....	67
Vague 1.....	67
Vague 2.....	68
Vague 3 (Après lecture).....	68

# INTRODUCTION

---

## L'ENTREPRISE HOME ENERGY AWARENESS

Home Energy Awareness, sauvons la planète en étant des consommateurs d'énergie avertis !

L'entreprise Home Energy Awareness vous aide à avoir une vue globale sur votre maison, pour une meilleure gestion énergétique.

Nos services consistent en la récupération d'informations depuis les objets connectés (domotique / énergie) et leur visualisation. Via nos dashboards vous pourrez avoir une vue sur votre consommation en énergie en temps réel, l'interpréter et la faire changer.

## CLIENTS

Nos deux principaux clients sont « Client1 » et « Client2 ».

Leurs besoins diffèrent un peu, au vu du nombre de capteurs : Le Client1 en a plus.

Voici la liste complète de leurs capteurs :

### CAPTEURS DU CLIENT1 (5 PIÈCES)

#### 1. Entrée

- 1 détecteur d'ouverture de porte d'entrée
- 1 ampoule connectée

#### 2. Salon

- 2 détecteurs d'ouverture sur les porte fenêtre
- 2 ampoules connectées
- 1 détecteur de présence
- 1 capteur de température
- 2 radiateurs connectés avec fil pilote et capteur de puissance

#### 3. Chambre

- 2 détecteurs d'ouverture sur les porte fenêtre
- 1 ampoules connectées
- 1 détecteur de présence
- 1 capteur de température
- 1 radiateurs connectés avec fil pilote

#### 4. Cuisine

- 1 détecteur de présence
- 1 capteur de température
- 1 radiateurs connectés avec fil pilote
- 1 prise connectée avec un compteur de consommation (le grille pain est branché dessus, ils avaient peur qu'il soit un peu vieux et ils voudraient vérifier s'il consomme trop)
- 2 appareil avec un capteur de puissance (machine à laver, four)

#### 5. Salle de bain

- 1 appareil avec un capteur de puissance (ballon d'eau chaude électrique)
- 1 radiateurs connectés avec fil pilote
- 1 ampoules connectées

### CAPTEURS DU CLIENT2 (2 PIÈCES)

#### 1. Chambre/Salon/Cuisine

- 1 ampoules connectées
- 1 détecteur de présence
- 1 capteur de température
- 1 radiateurs connectés avec fil pilote

#### 2. Salle de bain

- 1 appareil avec un capteur de puissance (ballon d'eau chaude électrique)
- 1 radiateurs connectés avec fil pilote
- 1 ampoules connectées

### FORMAT DES DONNÉES DES CAPTEURS

#### 1. Détecteurs de présence (porte d'entrée/pièce)

- NomCapteur
- DateCapture
- ValeurCapture - (0/1)

#### 2. Capteur de luminosité

- NomCapteur
- DateCapture
- ValeurCapture - (Double, en Lux)

#### 3. Capteur d'activation de lumière (Exemple : Philips Hue)

- NomCapteur
  - DateCapture
  - ValeurCapture - (0 : éteinte, 1 : allumée)
4. Capteur de température
- NomCapteur
  - DateCapture
  - ValeurCapture - (Double, en °C)
5. Capteur de Position
- NomCapteur
  - DateCapture
  - ValeurCaptureLongitude
  - ValeurCapteurLatitude
6. Capteur d'activation de chauffage
- NomCapteur
  - DateCapture
  - ValeurCapture - (6 états possible : Confort, Confort -1°C, Confort -2°C, Eco, Hors gel, Arrêt. Voir le principe des "fil pilote)
7. Capteur d'activation de climatisation
- NomCapteur
  - DateCapture
  - ValeurCapture - (Double, Température cible en °C)
8. Capteur de consommation énergétique
- NomCapteur
  - DateCapture
  - ValeurCapture - (Double, en kWh)
9. Capteur de puissance énergétique
- NomCapteur
  - DateCapture
  - ValeurCapture - (Integer, en kW)
10. Capteur ordre ouverture volet
- NomCapteur

- DateCapture
- ValeurCapture - (Integer, 0-100 en %. 0% : volet ouvert, 100% : volet fermé)

#### 11. Capteur position volet butée

- NomCapteur
- DateCapture
- ValeurCapture - (0 ou 1. 0 : volet ouvert, 1 : volet fermé)

## DÉTAILS PROJET

Le projet :

- consiste en l'élaboration de la solution en partant de l'ingestion de données jusque leur affichage dans des dashboards pour une meilleure interprétation.
- concerne les deux clients fictifs Client1 et Client2. L'entreprise Home Energy Awareness étant également fictive.
- s'exécute dans le cadre de mes études au sein de l'école ESILV en Année 5
- Prend fin le 12 janvier 2020.

Au fil de ce rapport, sera abordée l'exécution de son projet, à savoir les moyens techniques mises en œuvre, questionnements et propositions de solutions.

Vous pourrez accéder au code et aux scripts via mon github : <https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge>

## INGESTION (TP1)

---

### DESCRIPTION GÉNÉRALE DU PROBLÈME

Le but du projet est de récupérer les données de la domotique ou énergétiques de la maison du client (comme décrit dans l'introduction).

Il sera donc premièrement question de récupérer ces données, de manière à les regrouper avant de les traiter dans un second temps.

Intéressons nous donc aux solutions d'ingestion qui existent afin de choisir la meilleure pour notre cas spécifique.

### SOLUTIONS EXISTANTES

#### INTRODUCTION

L'ingestion des données consiste en la mise en place d'un Message Broker qui va directement récupérer les données générées depuis les objets connectés. L'intérêt est

de pouvoir supporter les grosses quantités de données générées par les objets, avoir un lien asynchrone entre les producteurs et consommateurs de données et être capable de garder ces messages sans les perdre, pour une utilisation future (ou pas).

L'utilisation d'un message broker permet également une utilisation plus simple des données générées du fait de l'abstraction de celles-ci à la fois pour le producteur et le consommateur.

## PATTERNS D'INTÉGRATION

Il existe plusieurs types de patterns d'intégration de données dans le message broker :

- Content based Router : Se base sur le contenu du message pour router la donnée vers le bon récipient
- Point-to-Point Channel : Ou l'on s'assure qu'un seul consommateur reçoit les données à partir d'un même producteur.
- Publish/Subscribe Channel : Délivre la copie d'événements particuliers à différents Subscribers (Abonnés). Ces événements sont filtrés soit par leur Topic (Topic-based), auquel on peut s'abonner ; ou filtrés par leur contenu (Content-based) où l'abonné définira lui même à quelle règle il s'abonne.

## SOLUTIONS DE MESSAGE BROKERS

Nous avons également le choix de différents message brokers :

- RabbitMQ (propose notamment du MQTT et du AMQP)
- Mosquitto (MQTT)
- Apache Kafka (protocole spécifique)

## PROTOCOLES

**MQTT** est un protocole plus léger et efficace en terme de latence, performances et d'énergie ; il est donc plus utilisé sur des systèmes embarqués proches des objets connectés.

Il permet la mise à disposition simple des données dans un graphe de topics.

**AMQP** (le A pour advanced) est utilisé plutôt dans des contextes où l'on est moins limités par les ressources de l'hôte de la solution.

Il permet donc de mettre en place des architectures plus complètes avec la notion « d'Exchanges/Queues ». Le consommateur va s'abonner à un topic mis à disposition. Le consommateur peut également être l'initiateur de la communication.

Le protocole AMQP est plutôt focalisé sur la fiabilité que sur la latence contrairement à l'MQTT.



Le protocole de **Kafka** est plus utile dans un contexte où l'on a vraiment besoin de garder toute trace des messages reçus (sous forme de logs) ; ce qui peut également être un avantage dans certains contextes.

En revanche il n'utilise que le principe de topics pour la mise à disposition des données.

## CHOIX DE LA SOLUTION

### MESSAGE BROKER

Dans un contexte de domotique, il n'y a pas énormément de capteurs et il n'y a pas besoin d'une bonne latence forcément. Nous pourrions donc nous concentrer sur la fiabilité de l'ingestion de données avec le protocole AMQP et la solution RabbitMQ.

### ARCHITECTURE EXCHANGES/QUEUES ET CONTAINERS

Un producteur se connecte sur un Exchange pour publier ses messages tandis que un consommateur va se connecter sur une Queue dans le principe général.

Pour la récupération des données des clients :

- **1 Vhost par Client** : Permet une sécurité pour le client : Les données sont séparées pour chaque client directement sur RabbitMQ et chaque client aura son propre identifiant afin de se connecter et d'envoyer des données. Cela évite de créer un container de RabbitMQ par client ce qui serait trop coûteux. Le fait d'avoir 1 Vhost par client permet de lui assurer que ses données restent en circuit uniquement dans ce Vhost, et donc sont séparées des autres flux de données.
- **1 Exchange par Maison** : Cela permet d'avoir une séparation au cas où l'on voudrait faire des modifications spécifique à une maison. Cela permet donc une évolutivité. De plus, on pourra y donner accès à deux utilisateurs donc (deux identifiants) différents, ce qui pourrait ajouter de la sécurité en cas d'attaque. Nous allons nous contenter d'avoir un seul identifiant pour les différents exchanges.

L'échange sera de type **Fanout** car nous n'avons pas spécialement besoin de routing key. On ne va pas trier les informations car ce travail sera fait plus tard dans le stockage en BDD. En effet, nous n'avons pas besoin d'une faible latence pour du temps réel, donc on ne se connectera probablement pas directement à une queue de rabbitMQ pour récupérer les données et les traiter immédiatement. Donc autant ne pas trier les données tout de suite.

D'autant plus que rajouter une telle information (routing key) directement dans les objets connectés créerait une redondance de données (impact réseau, perfs, énergie et stockage) et demanderait plus de travail si l'on fait une modification sur un objet connecté en particulier. **Toute forme d'opération pour enrichir, modifier, vérifier ou dispatcher les données se fera plus tard avec le Data Routing ou lors du stockage des données.**

- **1 Queue par Client** : Pas de tri de données à ce stade on a dit, donc on ne se complique pas la vie. Mélanger tous les messages ne pose pas de problème car via l'identifiant du capteur nous avons déjà suffisamment d'infos pour plus tard faire le tri. D'autant plus que cela facilite la tâche de n'avoir qu'un identifiant pour récupérer la donnée pour la suite du processus.
- La question est aussi de savoir identifier la **provenance** et la **nature** du message :  
  
Actuellement dans un message, on a : (NomCapteur, DateCapture, ValeurCapture)  
  
On a donc le nom du capteur donc indirectement aussi la nature de l'information, qui sera plus tard stockée sur une BDD. Pour se faciliter la tâche on va également supposer que **chaque capteur a un identifiant strictement unique**.
- **1 container de RabbitMQ lancé par Pays** sur un network de type bridge afin d'isoler les données de différents pays. (ici france) De cette manière on assure que les données ne sortent pas d'un pays dans tous les cas.

Pour renvoyer des données aux clients :

- **1 nouveau Vhost par client** : Le fait de séparer les deux directions d'envoi / réception de données sécurise la chose, notamment pour que si un identifiant se fait « hacker » on n'aura pas accès à tout.
- **1 nouveau Exchange par client** : ce serait supplémentaire pour faire revenir des données chez le client. Ces Exchanges seraient de type **Direct**. Le routing\_key sera l'identifiant exact de la maison du client.
- **1 nouvelle Queue par Maison**: Cela permettra de directement envoyer les messages vers une maison en particulier ; c'est plus pratique et permet également une fiabilité et évolutivité des données si l'on veut séparer les choses dans des clusters différents.

## IMPLEMENTATION

### DOCKER-COMPOSE

J'ai créé un réseau bridge pour chaque pays (annulé car trop coûteux en temps) ; donc je suis resté sur la commande standard :

```
docker network create iot-labs
```

Pour le fichier docker compose, voici le script nécessaire pour lancer le container de RabbitMQ :

```

version: '3.3'
services:
  rabbitmq1:
    image: "rabbitmq:3-management"
    hostname: "rabbitmq1"
    environment:
      RABBITMQ_ERLANG_COOKIE: "SWQOKODSQALRPCLNMEQG"
      RABBITMQ_DEFAULT_USER: "rabbitmq"
      RABBITMQ_DEFAULT_PASS: "rabbitmq"
      RABBITMQ_DEFAULT_VHOST: "/"
    ports:
      - "15672:15672"
      - "5672:5672"
    networks:
      - iot-labs
    labels:
      NAME: "rabbitmq1"
networks:
  iot-labs:
    external: true

```

La version a été changée de 3.7 en 3.3 car cela ne fonctionnait pas.

Le docker-compose est lancé de la manière suivante :

```
docker-compose -f docker-compose.yml up -d
```

Ce docker compose ainsi que la commande pourront donc être réutilisés pour chaque pays.

Deux ports sont ouverts, un pour les requêtes sur l'API de RabbitMQ et l'autre pour se connecter au terminal de configuration.

C'est donc l'adresse 0.0.0.0:15672 que nous utiliserons pour se connecter à l'interface.

L'adresse de l'API est 0.0.0.0:5672

Nous utiliserons les identifiants présents dans le docker compose (rabbitmq)

## INTERACTIONS AVEC LE CONTAINER

Pour lancer des commandes directement dans le container lancé, nous utiliserons la commande :

```
docker exec -it rabbitmq_rabbitmq1_1 /bin/bash
```

Ensuite, pour lancer des commandes rabbitmq dans le bash, nous utiliserons la doc :

<https://www.rabbitmq.com/management-cli.html>

## PARAMÉTRAGE RABBIT-MQ (GRAPHIQUE)

Doc Rabbit-MQ :

<https://blog.eleven-labs.com/fr/rabbitmq-partie-1-les-bases/>

Création des users pour les Producteurs (clients), les consumers de messages et les émetteurs de messages (Retour à la maison) :

### Users

▼ All users

Filter:  ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
client1		No access	•
client2		No access	•
consumer1		No access	•
consumer2		No access	•
rabbitmq	administrator	/, Client1, Client1-BackHome, Client2, Client2-BackHome	•
transmitter1		No access	•
transmitter2		No access	•

=> Pour les Vhosts normaux (récupération de données clients) :

Les clients ont le pouvoir d'envoyer des messages sur leurs Vhosts (API) Mais pas besoin de tags (Web) (Ce paramètre de tag se trouve tout en bas du paramétrage des users)

Les consommateurs n'ont le droit qu'à la visualisation.(API)

=> Pour les Vhosts BackHome :

Les clients ont le droit de récupérer les messages dans les Vhosts depuis les queues (API)

Les émetteurs ont le droit d'envoyer des messages dans les échanges (API)

Un seul compte admin : rabbitmq ; il est donc le seul à pouvoir configurer via l'interface Web.

Si l'entreprise est plutôt grande et que l'on a besoin d'un maximum de sécurité ; il serait préférable de créer un nouvel utilisateur admin pour chaque Vhost et supprimer du Vhost rabbitmq. De cette manière si le mot de passe rabbitmq se fait hacker, le client reste intouchable.

Dans ce TP nous allons nous contenter de travailler avec le compte rabbitmq

Voici l'ensemble des Virtual hosts pour les deux clients :

## Virtual Hosts

▼ All virtual hosts

Filter:  ☐ Regex ?

Overview			Messages			Network		Message rates		+/-
Name	Users ?	State	Ready	Unacked	Total	From client	To client	publish	deliver / get	
/	rabbitmq	running	NaN	NaN	NaN					
Client1	rabbitmq	running	NaN	NaN	NaN					
Client1-BackHome	rabbitmq	running	NaN	NaN	NaN					
Client2	rabbitmq	running	NaN	NaN	NaN					
Client2-BackHome	rabbitmq	running	NaN	NaN	NaN					

Les Vhosts BackHome vont servir à séparer les récupérations des données de clients et l'envoi de données vers les clients

Maintenant la partie intéressante est l'attribution des droits d'accès aux utilisateurs.

Voici la configuration type d'un Vhost normal :

User	Configure regexp	Write regexp	Read regexp	
client1	.*	.*	.*	Clear
consumer1	.*		.*	Clear
rabbitmq	.*	.*	.*	Clear

Et voici la configuration type d'un Vhost BackHome :

User	Configure regexp	Write regexp	Read regexp	
client1	.*	.*	.*	Clear
rabbitmq	.*	.*	.*	Clear
transmitter1	.*	.*	.*	Clear

On pourrait faire des droits plus restreints mais je n'ai pas le temps d'apprendre à manier les regexp.

Voici donc la nouvelle config :

## Users

▼ All users

Filter:  ☐ Regex ?

Name	Tags	Can access virtual hosts	Has password
client1		Client1, Client1-BackHome	•
client2		Client2, Client2-BackHome	•
consumer1		Client1	•
consumer2		Client2	•
rabbitmq	administrator	/, Client1, Client1-BackHome, Client2, Client2-BackHome	•
transmitter1		Client1-BackHome	•
transmitter2		Client2-BackHome	•

## Virtual Hosts

▼ All virtual hosts

Filter:  ☐ Regex ?

Overview			Messages			Network		Message rates	
Name	Users ?	State	Ready	Unacked	Total	From client	To client	publish	deliver / get
/	rabbitmq	running	NaN	NaN	NaN				
Client1	client1, consumer1, rabbitmq	running	NaN	NaN	NaN				
Client1-BackHome	client1, rabbitmq, transmitter1	running	NaN	NaN	NaN				
Client2	client2, consumer2, rabbitmq	running	NaN	NaN	NaN				
Client2-BackHome	client2, rabbitmq, transmitter2	running	NaN	NaN	NaN				

On peut désormais avoir un minimum de sécurité car les choses ont été bien séparées en Vhosts.

Pour la suite des opérations, il faudra se logger en tant que rabbitmq :



Username:  \*

Password:  \*

Login

Ensuite, en se plaçant à tour de rôle dans les différents Vhosts, on configure les Exchanges et les Queues dont on a besoin.

Dans chaque Vhost normal, pour chaque Maison :

▼ Add a new exchange

Virtual host:	<input type="text" value="Client1"/>
Name:	<input type="text" value="Client1-Maison1"/>
Type:	<input type="text" value="fanout"/>
Durability:	<input type="text" value="Durable"/>
Auto delete: ?	<input type="text" value="No"/>
Internal: ?	<input type="text" value="No"/>
Arguments:	<input type="text"/> = <input type="text"/> <input type="text" value="String"/>
<a href="#">Add</a> <a href="#">Alternate exchange</a> ?	

▼ Add a new queue

Virtual host:	<input type="text" value="Client1"/>
Type:	<input type="text" value="Classic"/>
Name:	<input type="text" value="Client1"/>
Durability:	<input type="text" value="Durable"/>
Auto delete: ?	<input type="text" value="No"/>
Arguments:	<input type="text"/> = <input type="text"/> <input type="text" value="String"/>

## Exchange: Client1-Maison1 in virtual host Client1

### ▼ Overview

Message rates **last minute** ?

Currently idle

### Details

Type	fanout
Features	durable: true
Policy	

### ▼ Bindings

This exchange



... no bindings ...

Add binding from this exchange

To queue ▼ :	<input type="text" value="Client1"/>	*
Routing key:	<input type="text"/>	
Arguments:	<input type="text"/> = <input type="text"/>	String ▼

Dans chaque Vhost BackHome, pour chaque Client:

### ▼ Add a new exchange

Virtual host:	<input type="text" value="Client1-BackHome"/>	▼
Name:	<input type="text" value="Client1-BackHome"/>	*
Type:	<input type="text" value="direct"/>	▼
Durability:	<input type="text" value="Durable"/>	▼
Auto delete: ?	<input type="text" value="No"/>	▼
Internal: ?	<input type="text" value="No"/>	▼
Arguments:	<input type="text"/> = <input type="text"/>	String ▼
Add <b>Alternate exchange</b> ?		



▼ Add a new queue

Virtual host: Client1-BackHome ▼

Type: Classic ▼

Name: Client1-Maison0-BackHome \*

Durability: Durable ▼

Auto delete: ? No ▼

Arguments:  =  String ▼

## Exchange: Client1-BackHome in virtual host Client1-BackHome

▼ Overview

Message rates last minute ?

Currently idle

Details

Type	direct
Features	durable: true
Policy	

▼ Bindings

This exchange

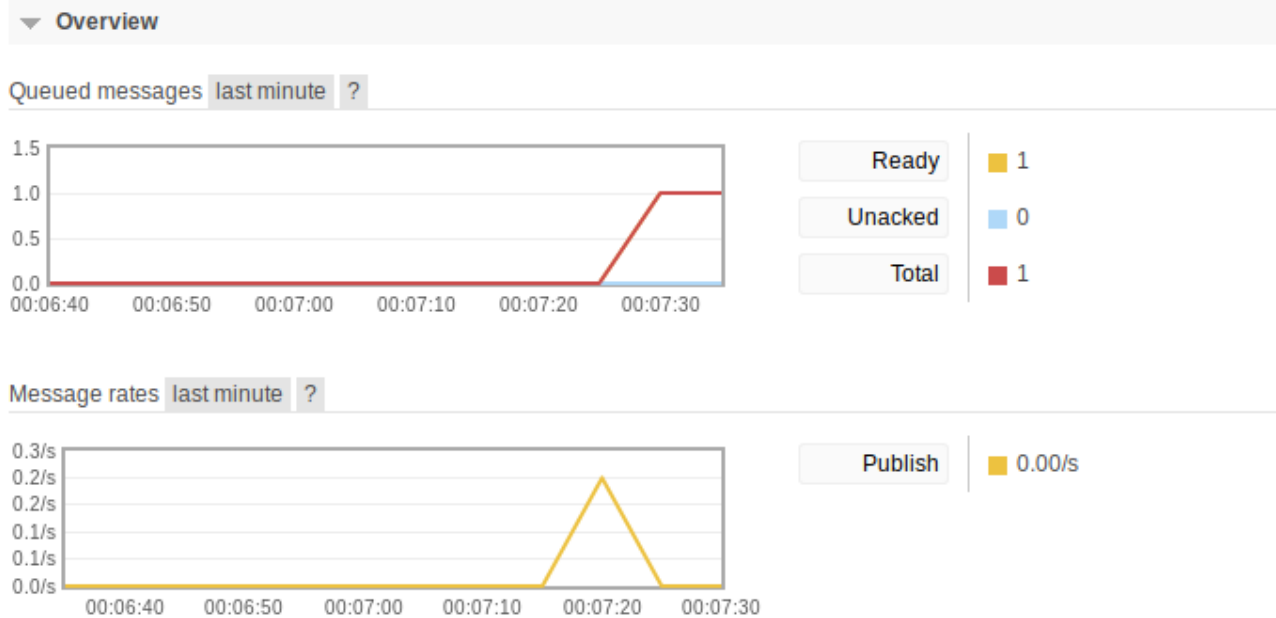
⇓

To	Routing key	Arguments	
Client1-Maison0-BackHome	client1maison0		Unbind

## PREMIERS MESSAGES (MANUEL SUR RABBITMQ)

Dans le Vhost normal de recuperation de données de client ; après avoir publié un message dans l'échange Client1-Maison1 :

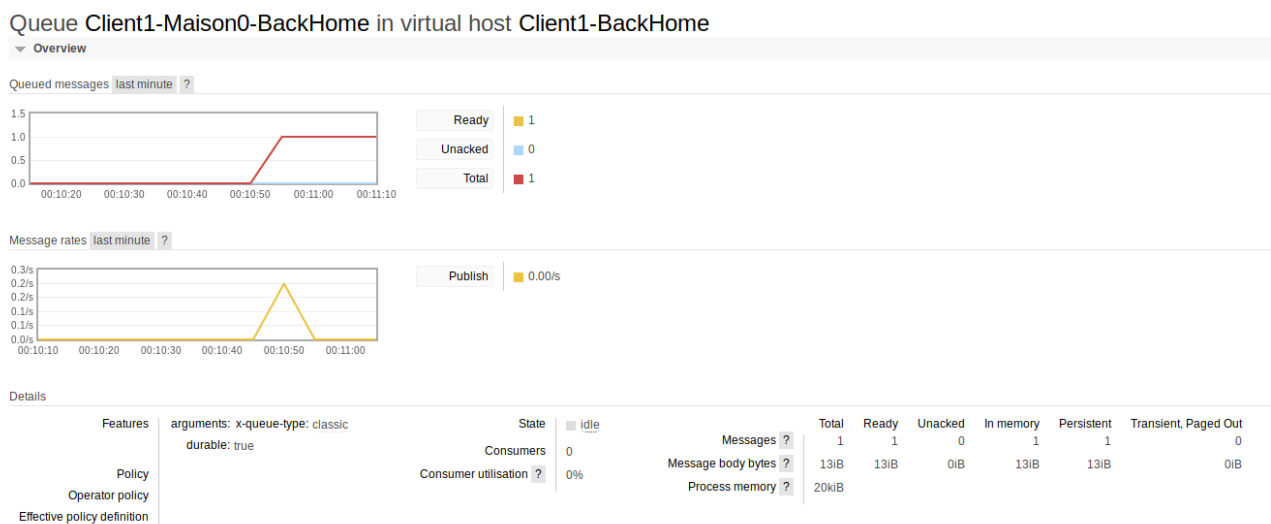
## Queue Client1 in virtual host Client1



Et Dans le Vhost d'envoi de données au client, en publiant un message sur l'Exchange Client1-BackHome:

**On précise le routing key " client1maison0 "**

**On reçoit bien également un message dans la queue de la maison 0 :**



## ENVOI/RECEPTION MESSAGES VIA PYTHON

Tout d'abord dans la console installer la librairie :

```
python -m pip install pika --upgrade
```

Mais avant faire alias py3=/usr/bin/python3

et utiliser py3 au lieu de python :

py3 -m pip install pika -upgrade

(plus tard j'ai finalement configuré pour que python3 comme alias fonctionne)

Voici le petit script python qui résulte :

```
import pika
import time

'''
    Function to get a message and print it in console
'''
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
    time.sleep(body.count(b'.'))
    print(" [x] Done")

'''
    Connecting to Rabbitmq
'''
credentials = pika.PlainCredentials('rabbitmq', 'rabbitmq')
parameters = pika.ConnectionParameters('localhost',
                                         5672,
                                         'Client1',
                                         credentials)

connection = pika.BlockingConnection(parameters)
channel = connection.channel()

'''
post a message from a client
'''

channel.basic_publish(exchange='Client1-Maison1',
                     routing_key='',
                     body='Hello World!')

'''
    Get message from client1 queue
'''

channel.basic_consume(queue='Client1',
                     on_message_callback=callback,
                     auto_ack=True
                     )

'''
    Disconnecting from Rabbitmq
'''
```

```
...  
connection.close()
```

Vous le trouverez ici : <https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/rabbitmq/rabbitmq-envoi-reception%5Bold%5D.py>

Mon souci est tout de même que je n'arrive pas à afficher les messages dans la console.. Mais j'arrive bien à les afficher dans l'interface web en tous cas;donc ce ne sera pas un point bloquant pour la suite.

Et dans ce script j'arrive tout de même bien à consommer les messages.

## AUTOMATISATION DU DÉPLOIEMENT

Apparemment la librairie utilisée précédemment donc pika, ne contient pas toutes les fonctionnalités dont on a besoin comme la creation de Vhosts.

On va donc le faire en shell en se basant sur la documentation <https://pulse.mozilla.org/api/>:

```
# Creating Vhosts  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT  
http://localhost:15672/api/vhosts/Client1  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT  
http://localhost:15672/api/vhosts/Client1-BackHome  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT  
http://localhost:15672/api/vhosts/Client2  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT  
http://localhost:15672/api/vhosts/Client2-BackHome  
  
# Creating users  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"client1","tags":""}' \  
http://localhost:15672/api/users/client1  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"client2","tags":""}' \  
http://localhost:15672/api/users/client2  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"consumer1","tags":""}' \  
http://localhost:15672/api/users/consumer1  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"consumer2","tags":""}' \  
http://localhost:15672/api/users/consumer2  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"transmitter1","tags":""}' \  
http://localhost:15672/api/users/transmitter1  
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -  
d'{"password":"transmitter2","tags":""}' \  
http://localhost:15672/api/users/transmitter2
```

```

# Put permissions to vhosts for each user
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client1/client1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client1-BackHome/client1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client2/client2
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client2-BackHome/client2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": "", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client1/consumer1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": "", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client2/consumer2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client1-BackHome/transmitter1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"configure": ".*", "write": ".*", "read": ".*"}' \
    http://localhost:15672/api/permissions/Client2-BackHome/transmitter2

# Create Exchanges
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type": "fanout", "auto_delete": false, "durable": true, "internal": false, "arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client1/Client1-Maison1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type": "fanout", "auto_delete": false, "durable": true, "internal": false, "arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client1/Client1-Maison2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type": "fanout", "auto_delete": false, "durable": true, "internal": false, "arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client2/Client2-Maison1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type": "fanout", "auto_delete": false, "durable": true, "internal": false, "arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client2/Client2-Maison2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type": "direct", "auto_delete": false, "durable": true, "internal": false, "arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client1-BackHome/Client1-BackHome

```

```

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"type":"direct","auto_delete":false,"durable":true,"internal":false,"arguments":
{}}' \
    http://localhost:15672/api/exchanges/Client2-BackHome/Client2-BackHome

# Create Queues
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client1/Client1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client2/Client2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client1-BackHome/Client1-Maison1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client1-BackHome/Client1-Maison2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client2-BackHome/Client2-Maison1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPUT -
d'{"auto_delete":false,"durable":true,"arguments":{}}' \
    http://localhost:15672/api/queues/Client2-BackHome/Client2-Maison2

# Creating bindings
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"","arguments":{}}' \
    http://localhost:15672/api/bindings/Client1/e/Client1-Maison1/q/Client1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"","arguments":{}}' \
    http://localhost:15672/api/bindings/Client1/e/Client1-Maison2/q/Client1

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"","arguments":{}}' \
    http://localhost:15672/api/bindings/Client2/e/Client2-Maison1/q/Client2
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"","arguments":{}}' \
    http://localhost:15672/api/bindings/Client2/e/Client2-Maison2/q/Client2

curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"client1maison1","arguments":{}}' \
    http://localhost:15672/api/bindings/Client1-BackHome/e/Client1-BackHome/q/Client1-
Maison1
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -
d'{"routing_key":"client1maison2","arguments":{}}' \
    http://localhost:15672/api/bindings/Client1-BackHome/e/Client1-BackHome/q/Client1-
Maison2

```

q/Client2

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"routing_key":"client2maison1","arguments":{}}' \
http://localhost:15672/api/bindings/Client2-BackHome/e/Client2-BackHome/q/Client2-Maison1
```

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"routing_key":"client2maison2","arguments":{}}' \
http://localhost:15672/api/bindings/Client2-BackHome/e/Client2-BackHome/q/Client2-Maison2
```

# Send a message on an exchange:

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"properties":{}}' \
http://localhost:15672/api/exchanges/Client1/Client1-Maison1/publish
```

# result: works

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"properties":{}}' \
http://localhost:15672/api/exchanges/Client1-BackHome/Client1-BackHome/publish
```

# result: works

# Get Messages:

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"count":5,"requeue":true,"encoding":"auto","truncate":50000,"ackmode":"ack_requeue_false"}' \
http://localhost:15672/api/queues/Client1/Client1/get
```

```
curl -i -u rabbitmq:rabbitmq -H "content-type:application/json" -XPOST -d'{"count":5,"requeue":true,"encoding":"auto","truncate":50000,"ackmode":"ack_requeue_false"}' \
http://localhost:15672/api/queues/Client1-BackHome/Client1-Maison1/get
```

# Result: both worked fine !

**On a même réussi à afficher les messages dans la console finalement !**

**Dernière version sur mon github**

**[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/launch\\_all.sh](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/launch_all.sh)**

## GÉNÉRATION AUTOMATIQUE DE MESSAGES

Oui, mon hostname est bien celui ci :

```
(base) stephane@DELLXPS-15-Linux:~/media/stephane/DATA/ESILV/AS/Dev Apps et Web services pour l'IOT/TP/Fichiers TP/rabbitmq$ docker exec -it rabbitmq_rabbitmq1_1 /bin/bash
root@rabbitmq1:/# cat /etc/hostname
rabbitmq1
root@rabbitmq1:/#
```

rabbitmq1

Lorsque j'utilise le script ci dessous :

version: '3.3'

services:

moke1:

image: "pcourbin/mock-data-generator:latest"

hostname: "moke1"

environment:

SENZING\_SUBCOMMAND: random-to-rabbitmq

SENZING\_RANDOM\_SEED: 1

SENZING\_RECORD\_MIN: 1

SENZING\_RECORD\_MAX: 100

SENZING\_RECORDS\_PER\_SECOND: 1

SENZING\_RABBITMQ\_HOST: rabbitmq1

SENZING\_RABBITMQ\_PASSWORD: rabbitmq

SENZING\_RABBITMQ\_USERNAME: rabbitmq

SENZING\_RABBITMQ\_QUEUE: Client1

MIN\_VALUE: 500

MAX\_VALUE: 700

SENZING\_DATA\_TEMPLATE: '{"SENSOR":"Temp1","DATE":"date\_now",

"VALUE":"float"}'

tty: true

labels:

NAME: "moke1"

networks:

- iot-labs

networks:

iot-labs:

external: true

Je suis confronté à un problème : il me crée une nouvelle queue dans / car il n'y a pas spécifié dans le docker-compose de vhost en particulier :

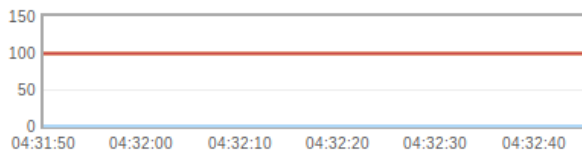
Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	Client1	classic		idle	99	0	99	0.00/s			
Client1	Client1	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
Client1-BackHome	Client1-Maison1	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
Client1-BackHome	Client1-Maison2	classic	D	idle	0	0	0				
Client2	Client2	classic	D	idle	0	0	0				
Client2-BackHome	Client2-Maison1	classic	D	idle	0	0	0				
Client2-BackHome	Client2-Maison2	classic	D	idle	0	0	0				



## Queue Client1 in virtual host /

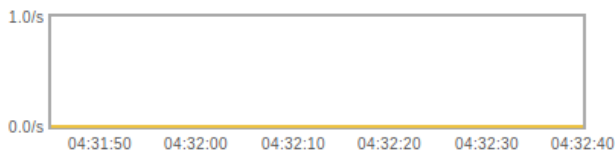
### ▼ Overview

Queued messages **last minute** ?



Ready 99  
Unacked 0  
Total 99

Message rates **last minute** ?



Publish 0.00/s

Details

il aurait fallu modifier peut être la ligne 1174 du fichier **mock-data-generator/mock-data-generator.py**

:

```
connection = pika.BlockingConnection(pika.ConnectionParameters(host=rabbitmq_host,
credentials=credentials))
```

Et ajouter donc le virtual host dans les paramètres de connection

Mais ben, flemme, donc je vais plutôt utiliser à l'avenir un script python que j'aurai concocté.

Et puis, pour juste l'exercice à la limite j'utiliserai cette nouvelle queue dans '/' comme ça je pourrais à la limite gagner du temps.

Le script python que j'ai concocté peut être trouvé ici : [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/rabbitmq/generate\\_data.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/rabbitmq/generate_data.py)

Il génère les données dans la format suivant dans le fichier generated\_data.txt :

```
{
  "Client/Vhost": {
    "Pièce": [
      {
        "generated_data (données de capture)": [
          {
            "DateCapture": "Date au format : 10/01/2020 13:12:39",
            "NomCapteur": "Nom du capteur",
            "ValeurCapture": Valeur enregistrée
          }
        ],
        "sensor_name": "Nom du capteur",
      }
    ]
  }
}
```

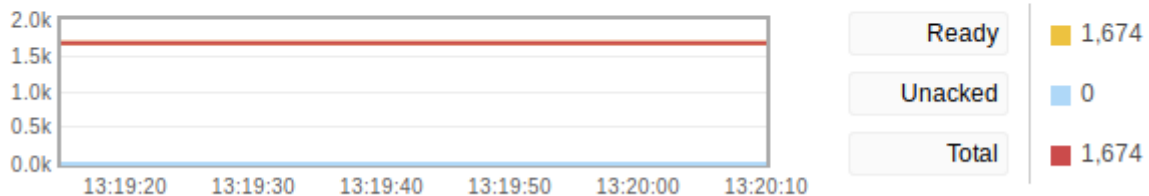
```
    "sensor_type": "Type du capteur"  
  },
```

Et ensuite le script les envoie aux bons Vhosts :

## Queue Client1 in virtual host Client1

▼ Overview

Queued messages last minute ?



## STOCKAGE (TP2)

### DESCRIPTION DU PROBLÈME

Il sera question dans cette section de stocker les données ; que ce soit des metadata, ou les données provenant des différents capteurs

### CHOIX DE LA SOLUTION

#### BASE DE DONNÉES

La solution de mongodb a été sélectionnée, au vu des données diverses pouvant être stockées dedans ainsi que la possibilité de gérer les données de manière plus simple. Nous aurions pu prendre une base de données temporelles, mais mongodb nous permet de tout stocker directement dessus.

#### ARCHITECTURE

1 BDD par client

2 collections par client avec comme architecture pour les documents:

collection des metadata :

```
{  
  "capteurs" : [  
    {  
      "TypeCapteur" : "Temperature",
```

```

        "NomCapteur" : "TempX",
        "NomPiece" : "Chambre",
        "NomMaison" : "Maison1"
    }
]
}

```

Oui, ce n'est pas optimisé comme dans une base de données relationnelles, car il y aura des données dupliquées (noms pièces ou maison par exemple). Mais comme ces données sont peu nombreuses en soi car un client ne va pas avoir des millions de pièces et de maisons, et changent peu ; cela ne pose pas de problème particulier.

Pour obtenir les metadata d'un capteur il n'y aura plus qu'à requêter en filtrant le nom du capteur et c'est tout.

Il faudrait donc un index sur les noms de capteurs, mais aussi sur les types de capteurs, des pièces et maisons ; au cas où on voudrait faire d'autres types de recherches sur la bdd.

Si une personne ne veut pas ses données à l'étranger ou voudrait les gérer directement chez soi, elle n'aura qu'à réutiliser le script écrit plus tard et le réutiliser en lançant une bdd chez elle.

Pour ce qui est de la répartition de données, l'ensemble des clients vont générer un peu près les mêmes quantités de données. Mais il faudrait effectivement les répartir via un sharding.

collection des données de capteurs :

```

{
    "NomCapteur" : [
        {
            "NomCapteur" : "TempX",
            "DateCapture" : "date",
            "ValeurCapture" : "val",
            "ValeurCaptureLongitude" : "val",
            "ValeurCaptureLatitude" : "val"
        }
    ]
}

```

Et ici ce qui nous intéresse, c'est le nom du capteur, qui nous permettra d'ensuite retrouver ses metadonnées dans l'autre collection et filtrer également dans cette collection les données du capteur en particulier. Il faudrait donc également un index là dessus.

La séparation metadonnées et données de capteurs nous permet tout de même d'éviter une certaine redondance de données.

# IMPLÉMENTATION

## DOCKER COMPOSE

Voici le fichier docker-compose de base :

```
version: '3.3'
services:
  mongo:
    image: mongo:4.2.0-bionic
    hostname: "mongo"
    restart: always
    labels:
      NAME: "mongo"
    networks:
      - iot-labs
  mongo-express:
    image: mongo-express:0.49.0
    hostname: "mongo_express"
    restart: always
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_SERVER: mongo
    labels:
      NAME: "mongo_express"
    networks:
      - iot-labs
networks:
  iot-labs:
    external: true
```

la commande pour lancer le container :

```
docker-compose -f docker-compose-mongodb.yml up -d
```

ceci est rajouté bien sur dans le fichier shell launch\_all.sh

ATTENTION UN SOUCI A ETE RENCONTRE CAR PAS DE PORTS DE SPECIFIES PAR  
DEFAULT POUR LE CONTAINER DE MONGODB DANS LE DOCKER-COMPOSE :

Il a fallu ajouter :

```
ports:
  - 27017:27017
```

Voyez par vous même : <https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/docker-compose-mongodb.yml>

## ARCHITECTURE MONGODB VIA MONGO EXPRESS

Pour y accéder : <http://0.0.0.0:8081/>
















Voici donc mes bdd pour chaque client :

### Mongo Express

Databases			Database Name	+ Create Database
	View	Client1		 Del
	View	Client2		 Del

Voici ensuite les collections pour chaque client :

### Viewing Database: Client1

Collections					captures	+ Create collection
	View	 Export	 [JSON]	 Import	captures	 Del
	View	 Export	 [JSON]	 Import	delete_me	 Del
	View	 Export	 [JSON]	 Import	metadata	 Del

Pour ce qui est de l'insertion de données, je vais simplement mettre un peu de metadata car ce sera mieux de le faire en script python.

Donc je vais simplement insérer une donnée exemple :

key :

capteurs

value :

```
[{"TypeCapteur" : "OuverturePorteFenetre", "Nom" : "capteur" : "OuverturePorteFenetre2", "NomPiece" : "Chambre", "NomMaison" : "Maison1"}]
```

## ARCHITECTURE MONGODB VIA SCRIPT PYTHON

Afin d'avoir mon script à ajouter dans le launch\_all.sh, et pouvoir relancer le container quand je veux ;

Voici mon script de chargement des bases de données, collections et metadata dans mongodb :

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb\\_generate\\_config.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb_generate_config.py)

Le script se base sur le json généré par le script utilisé pour rabbitmq, le fichier generated\_data.txt. :

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/generated\\_data.txt](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/generated_data.txt)

C'est ensuite une simple boucle qui se charge de tout charger dans mongodb.

## SCRIPT DE LECTURE SIMPLE DE DONNÉES (PYTHON)

Disponible ici :

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/simple\\_read.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/simple_read.py)

Voici les données affichées en sortie :

```
{ '_id': ObjectId('5e19995c7e60ba99d7553ea4'), 'TypeCapteur': 'OuverturePorteFenetre',  
'NomCapteur': 'OuverturePorteFenetre2', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553ea5'), 'TypeCapteur': 'OuverturePorteFenetre',  
'NomCapteur': 'OuverturePorteFenetre3', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553ea6'), 'TypeCapteur': 'Ampoule', 'NomCapteur':  
'Ampoule3', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553ea7'), 'TypeCapteur': 'Presence', 'NomCapteur':  
'Presencel', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553ea8'), 'TypeCapteur': 'Temperature', 'NomCapteur':  
'Temperaturel', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553ea9'), 'TypeCapteur': 'Chauffage', 'NomCapteur':  
'Chauffage2', 'NomPiece': 'Chambre', 'NomMaison': 'Maison1' }  
{ '_id': ObjectId('5e19995c7e60ba99d7553eaa'), 'TypeCapteur': 'Presence', 'NomCapteur':  
'Presence2', 'NomPiece': 'Cuisine', 'NomMaison': 'Maison1' }
```

## ENVOI DE DONNÉES POUR UN CAPTEUR (PYTHON)

Script disponible ici : [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb\\_send\\_data\\_from\\_json\\_file.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb_send_data_from_json_file.py)

Il supprime toutes les données de la collection de captures et la remplit en utilisant toujours le même fichier json généré pour rabbitmq.

Voici un aperçu du résultat :

## Viewing Collection: captures

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

Simple

Advanced

String

[Find](#)

Delete all 1620 documents retrieved

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

_id	DateCapture	NomCapteur	ValeurCapture
5e19da9247ff7ebd9be6a102	10/01/2020 13:36:01	OuverturePorteFenetre2	1
5e19da9247ff7ebd9be6a103	10/01/2020 13:36:01	OuverturePorteFenetre2	1
5e19da9247ff7ebd9be6a104	10/01/2020 13:36:01	OuverturePorteFenetre2	1
5e19da9247ff7ebd9be6a105	10/01/2020 13:36:01	OuverturePorteFenetre2	1
5e19da9247ff7ebd9be6a106	10/01/2020 13:36:01	OuverturePorteFenetre2	1

## SCRIPTS AVANCÉS DE REQUÊTES (AVEC DONNÉES DU JSON GÉNÉRÉ)

Nouveau script disponible ici : [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get\\_functions.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get_functions.py)

Il regroupe un ensemble de fonctions pour faire les différentes requêtes.

Ayant eu un souci avec le format de date généré de base par le script de rabbitmq ; j'ai rajouté un timestamp (secondes depuis le 1/1/1970) en plus de la date de capture et l'ai nommé "TimestampCapture".

Comme ceci :

```
{ '_id': ObjectId('5e19e516b12e987a57842853'), 'DateCapture': '01/09/2019 12:49:24',  
'NomCapteur': 'Temperature1', 'TimestampCapture': 1567334964.238596, 'ValeurCapture': -  
8.808946471696846 }
```

Pour le script (a), de récupération de dernière valeur enregistrée, je n'aurai plus qu'à trier en fonction du timestamp et prendre la première valeur de la liste.

Pur ce qui est de la suite, je me suis basé sur le tutoriel simpa :

<https://www.compose.com/articles/aggregations-in-mongodb-by-example/>

J'ai utilisé le timestamp au lieu de l'objet pour les dates par défaut de mongodb car c'était plus simple pour moi que tout refaire depuis rabbitmq

# CONNECTION AVEC RABBITMQ

## Mon docker-compose adapté :

```
#https://github.com/marcelmaatkamp/docker-rabbitmq-mongodb
version: '3.3'
services:
  amqp2mongo1:
    image: "marcelmaatkamp/rabbitmq-mongodb"
    hostname: "amqp2mongo1"
    environment:
      AMQPHOST: 'amqp://rabbitmq:rabbitmq@rabbitmq1:5672/Client1'
      MONGODB: 'mongodb://mongo/Client1'
      MONGOCOLLECTION: 'captures'
      TRANSLATECONTENT: 'true'
    command: 'Client1'
    tty: true
    labels:
      NAME: "amqp2mongo1"
    networks:
      - iot-labs
    restart: always
networks:
  iot-labs:
    external: true
```

L'URI pour rabbitmq a du être modifiée pour accéder au bon vhost avec les credentials.

Et la queue en question est bien vidée.

L'avantage d'une telle image docker est qu'elle est assez simple à lancer ; permet de consommer les messages de rabbitmq de manière très légère et nous permet même de convertir le format des messages (on le verra plus tard).

Mais en revanche le format des données n'est pas du tout le même car il est en base64 et accompagné d'informations provenant de rabbitmq :

```
{
  "_id": ObjectID("5e1a0b0ae80f2b0100acd545"),
  "date": ISODate("2020-01-11T17:51:06.982Z"),
  "queue": "Client1",
  "fields": {
    "consumerTag": "amq.ctag-jxbQiQ0H4Zvu8hI_qgnoAg",
    "deliveryTag": 1,
    "redelivered": false,
    "exchange": "Client1-Maison1",
    "routingKey": ""
  },
  "properties": {
    "contentEncoding": "base64",
```



```

    "contentType": "application/octet-stream"
  },
  "content":
"eydOb21DYXB0ZXVYJzogJ091dmVydHVyZVBvcnRlRW50cmVlMCcsICdEYXRlQ2FwdHVyZSc6ICcyMS8wOS8yMDE5
IDlzMjMyOjAyJywgJ1ZhbGV1ckNhchR1cmUnOiAxLCAnVGltZXN0YW1wQ2FwdHVyZSc6IDE1NjIxMDE1MjIuNTA1O
TQzfQ=="
}

```

On retrouve bien le message :

## Decode from Base64 format

Simply use the form below

```

eydOb21DYXB0ZXVYJzogJ091dmVydHVyZVBvcnRlRW50cmVlMCcsICdEYXRlQ2FwdHVyZSc6ICcyMS8wOS8yMDE5
IDlzMjMyOjAyJywgJ1ZhbGV1ckNhchR1cmUnOiAxLCAnVGltZXN0YW1wQ2FwdHVyZSc6IDE1NjIxMDE1MjIuNTA1O
TQzfQ==

```

**i** For encoded binaries (like images, documents, etc.) upload your data via the [file decode form](#) below.

UTF-8

Source charset.

☐ Live mode OFF

Decodes in real-time when you type or paste (supports only unicode charsets).

< DECODE >

Decodes your data into the textarea below.

```

{'NomCapteur': 'OuverturePorteEntree0', 'DateCapture': '21/09/2019 23:32:02', 'ValeurCapture': 1,
'TimestampCapture': 1569101522.505943}

```

## SCRIPTS AVANCÉS DE REQUÊTES (AVEC DONNÉES RETIRÉES DEPUIS LA QUEUE RABBITMQ)

Eh oui, comme rabbitmq m'a imposé un nouveau format de données (base64), il faut que je m'adapte.

J'ai trouvé une solution en lisant cette doc : <https://www.npmjs.com/package/amqp-to-mongo>

Et : <https://www.rabbitmq.com/consumers.html#message-properties>

J'ai rajouté des propriétés à mes messages pour ne pas qu'ils soient encodés en base64 :

The screenshot shows the 'Publish message' section of the RabbitMQ web interface. It includes fields for 'Routing key', 'Headers' (with a dropdown set to 'String'), and 'Properties'. The 'Properties' section is expanded, showing 'content\_encoding' set to 'utf8' and 'content-type' set to 'application/json'. The 'Payload' field contains a JSON object: {"DateCapture": "06/10/2018 04:07:58", "NomCapteur": "Chauffage6", "TimestampCapture": 1538791678.200401, "ValeurCapture": "Confort -1 C"}. A 'Publish message' button is at the bottom left.

Et ensuite dans mon code comme ceci : ([https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/rabbitmq/generate\\_data.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/rabbitmq/generate_data.py))

```
def pub_msg(channel, exchange= "Client1-Maison1", routing_key = "", msg='Hello World!'):
    properties = pika.BasicProperties(content_type="application/json",
    content_encoding="utf8")
    channel.basic_publish(exchange=exchange,
        routing_key=routing_key,
        body=msg,
        properties=properties
    )
```

Et maintenant les données dans mongodb sont correctes :

```
{
  "_id": ObjectId("5e1a1c82d3fa540100ce6d6f"),
  "date": ISODate("2020-01-11T19:05:38.360Z"),
  "queue": "Client1",
  "fields": {
    "consumerTag": "amq.ctag-Z360Q--7DY75TIZb3FNw3Q",
    "deliveryTag": 275,
    "redelivered": false,
```

```

    "exchange": "Client1-Maison1",
    "routingKey": ""
  },
  "properties": {
    "contentType": "application/json",
    "contentEncoding": "utf8"
  },
  "content": {
    "NomCapteur": "OuverturePorteEntree0",
    "DateCapture": "25/05/2018 18:57:17",
    "ValeurCapture": 0,
    "TimestampCapture": 1527267437.8548
  }
}

```

Et ensuite il a fallu dans un nouveau fichier de script qui contient les fonctions de requetes avancées ; modifier mes requêtes pour correspondre à la structure ci-dessus : [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get\\_functions\\_\(data\\_from\\_rabbitmq\).py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get_functions_(data_from_rabbitmq).py)

Voici les résultats :

Last capture for sensor Temperature1:

```

{'_id': ObjectId('5e1a1c82d3fa540100ce6e10'), 'date': datetime.datetime(2020, 1, 11, 19, 5, 38, 419000), 'queue': 'Client1', 'fields': {'consumerTag': 'amq.ctag-Z360Q--7DY75TIzb3FNw3Q', 'deliveryTag': 436, 'redelivered': False, 'exchange': 'Client1-Maison1', 'routingKey': ''}, 'properties': {'contentType': 'application/json', 'contentEncoding': 'utf8'}, 'content': {'NomCapteur': 'Temperature1', 'DateCapture': '26/08/2020 23:43:06', 'ValeurCapture': 13.29853505540521, 'TimestampCapture': 1598478186.142886}}

```

Average for sensor Temperature1 between dates 01/09/2019 00:00:00 and 01/01/2021 00:00:00:

```

[
  {
    "_id": "Temperature1",
    "average": 20.03336792851846
  }
]

```

Minimum for sensor Temperature1 between dates 01/09/2018 00:00:00 and 01/01/2020 00:00:00:

```

[
  {
    "_id": "Temperature1",
    "min": 9.084538404176037
  }
]

```

```
}  
]
```

## INDEXES

Les indexes auxquels on pourrait penser seraient par exemple dans les metadata de le faire sur tous les attributs.

Et de le faire sur le nom de capteur, la date des enregistrements.

Indexes ajoutés dans ces fichiers :

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get\\_functions\\_\(data\\_from\\_rabbitmq\).py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get_functions_(data_from_rabbitmq).py)

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get\\_functions.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get_functions.py)

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb\\_generate\\_config.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb_generate_config.py)

## ETAT DU CONTENU DES COLLECTIONS

Dans la collection de captures pour le client 2, on a l'ancien format de données, provenant du fichier json généré :

```
{  
  "_id": ObjectID("5e1a2ae0cedd43cc2afd43ad"),  
  "DateCapture": "28/08/2020 15:34:49",  
  "NomCapteur": "Ampoule5",  
  "TimestampCapture": 1598621689.304265,  
  "ValeurCapture": 0  
}
```

Et dans la collection de captures pour le client 1, on a le nouveau format de données, provenant de rabbitmq :

```
{  
  "_id": ObjectID("5e1a2ae383f13801006cb22b"),  
  "date": ISODate("2020-01-11T20:06:59.638Z"),  
  "queue": "Client1",  
  "fields": {  
    "consumerTag": "amq.ctag-I6_qqD7Mrhn2tn19k3Qwsg",  
    "deliveryTag": 1,  
    "redelivered": false,  
    "exchange": "Client1-Maison1",  
    "routingKey": ""  
  },  
  "properties": {  
    "contentType": "application/json",  
  }  
}
```

```

        "contentEncoding": "utf8"
    },
    "content": {
        "NomCapteur": "OuverturePorteEntree0",
        "DateCapture": "25/11/2020 02:35:23",
        "ValeurCapture": 1,
        "TimestampCapture": 1606268123.011438
    }
}

```

## CLE DE SHARDING

On verra si on a le temps.

## DATAROUTING (TP3)

---

### DESCRIPTION DU PROBLÈME

On va chercher maintenant à faire bien communiquer l'ensemble de nos outils informatiques ensemble, avec une orchestration en utilisant apache nifi

En effet, nifi permet d'éviter de nombreux scripts ou outils variés pour faire communiquer tout au même endroit.

### CHOIX DE LA SOLUTION

## IMPLÉMENTATION

### DOCKER-COMPOSE

Voici le fichier docker-compose:

```

version: '3.3'
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
```

Nifi est donc disponible ici:

/localhost:8083/nifi/

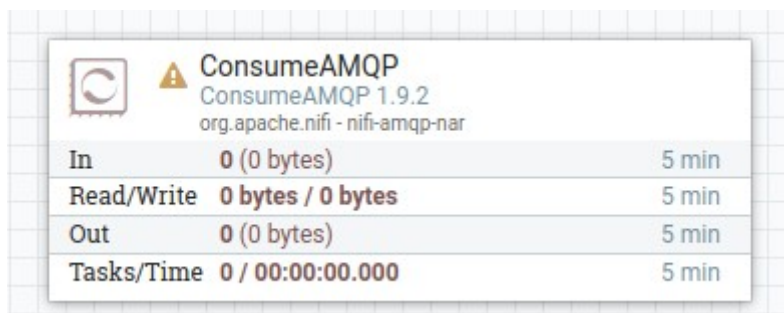
## TEMPLATES NIFI

Vous pourrez trouver tous mes templates nifi ici:

<https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/tree/master/nifi/templates>

## CONSOMMATION RABBITMQ / PUSH DANS LA COLLECTION MONGODB

Processeur de consommation des données depuis Rabbitmq:



The screenshot shows the Nifi console interface for a processor named 'ConsumeAMQP'. The processor is version 1.9.2 and is from the 'org.apache.nifi - nifi-amqp-nar' namespace. It has a warning icon. Below the header, there is a table showing the processor's performance metrics over a 5-minute interval.

ConsumeAMQP ConsumeAMQP 1.9.2 org.apache.nifi - nifi-amqp-nar		
In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Config du processeur de rabbitmq:

## Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Scheduling Strategy ?

Timer driven

Concurrent Tasks ?

1

Run Schedule ?

30 sec

Execution ?

All nodes

CANCEL

APPLY

## Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value
Queue	?	Client1
Auto-Acknowledge messages	?	false
Batch Size	?	10
Host Name	?	rabbitmq1
Port	?	5672
Virtual Host	?	Client1
User Name	?	rabbitmq
Password	?	Sensitive value set
AMQP Version	?	0.9.1
SSL Context Service	?	No value set
Use Certificate Authentication	?	false
Client Auth	?	REQUIRED

On peut noter le Vhost, le nom de la queue et le hostname provenant du docker-

compose de rabbitmq.

L'identifiant de connection est celui par défaut dans rabbitmq.

Processeur d'insertion de données dans mongodb:

### Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

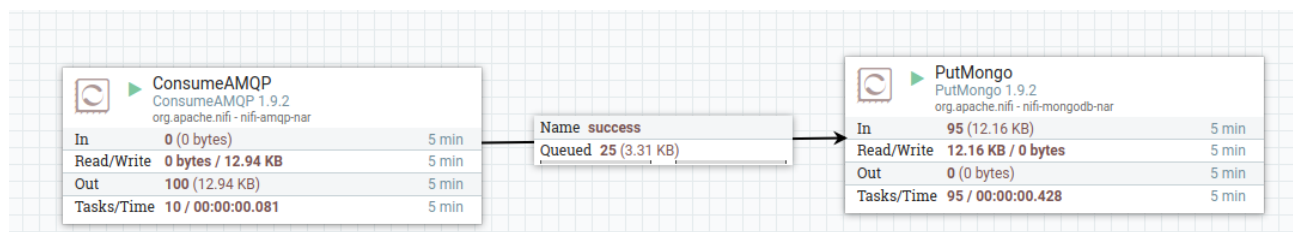
COMMENTS

Required field

+

Property	Value	
Client Service	<div>?</div> No value set	
Mongo URI	<div>?</div> mongodb://mongo:27017/	
Mongo Database Name	<div>?</div> Client1	
Mongo Collection Name	<div>?</div> captures	
SSL Context Service	<div>?</div> No value set	
Client Auth	<div>?</div> REQUIRED	
Mode	<div>?</div> insert	
Upsert	<div>?</div> false	
Update Query Key	<div>?</div> No value set	
Update Query	<div>?</div> No value set	
Update Mode	<div>?</div> With whole document	
Write Concern	<div>?</div> ACKNOWLEDGED	
Character Set	<div>?</div> UTF-8	

Ce qui a pour résultat:



Un très joli flux qui fonctionne.

Et dans mongodb, je retrouve bien les données:



## Viewing Collection: captures

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

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

[Find](#)

Delete all 140 documents retrieved

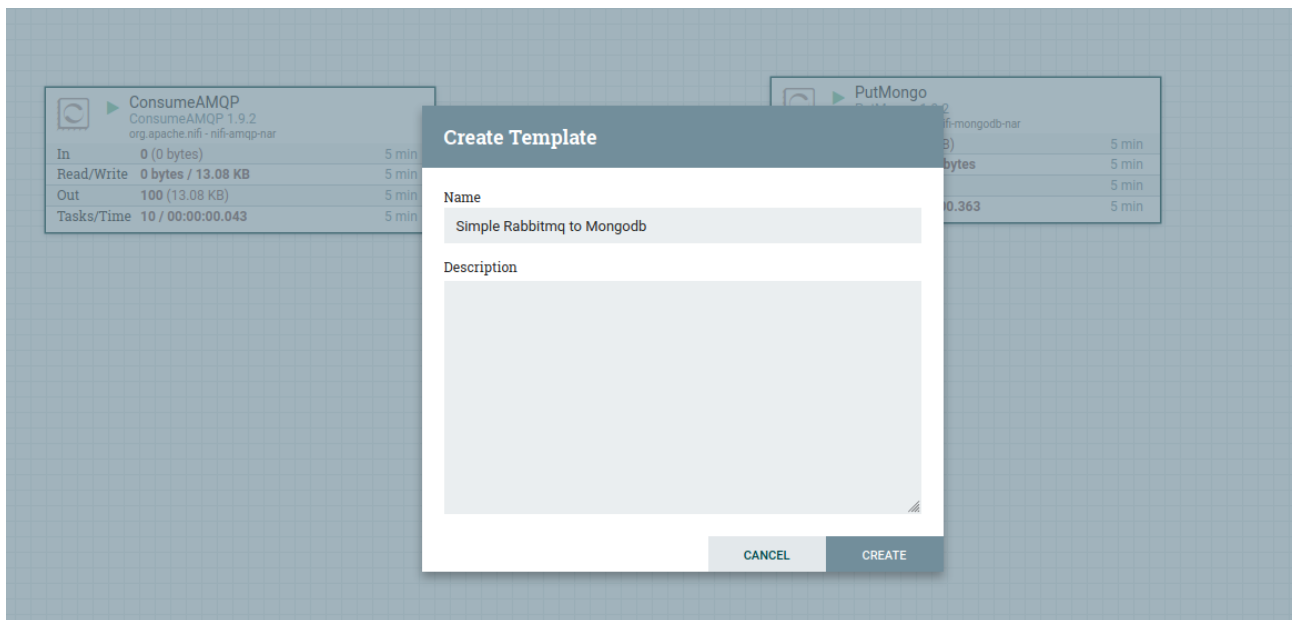
[First](#) [Prev](#) [Next](#) [Last](#)

_id	NomCapteur	DateCapture	ValeurCapture	TimestampCapture
5e1a64368b87bc003977a7df	OuverturePorteEntree0	07/07/2020 16:12:06	0	1594131126.942843
5e1a64548b87bc003977a7e0	OuverturePorteEntree0	27/11/2018 02:18:54	0	1543281534.760466
5e1a64728b87bc003977a7e1	OuverturePorteEntree0	22/08/2020 01:35:01	0	1598052901.399416
5e1a64978b87bc003977a7e3	OuverturePorteEntree0	09/11/2018 08:59:37	0	1541750377.10495
5e1a64988b87bc003977a7e4	OuverturePorteEntree0	11/05/2020 13:12:34	0	1589195554.957014
5e1a64998b87bc003977a7e5	OuverturePorteEntree0	03/05/2019 11:03:56	0	1556874236.409539
5e1a649a8b87bc003977a7e6	OuverturePorteEntree0	25/11/2020 17:25:05	0	1606321505.885646
5e1a649b8b87bc003977a7e7	OuverturePorteEntree0	30/06/2018 13:51:01	0	1530359461.282251

Mais particularité par rapport au container utilisé précédemment pour faire le pont entre rabbitmq et mongodb: les données sont déjà au bon format ! Pas besoin de les modifier:

```
{
  "_id": ObjectID("5e1a64368b87bc003977a7df"),
  "NomCapteur": "OuverturePorteEntree0",
  "DateCapture": "07/07/2020 16:12:06",
  "ValeurCapture": 0,
  "TimestampCapture": 1594131126.942843
}
```

J'ai créé mon template à partir de ce connecteur avec ces deux processeurs:



Il est accessible ici: [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/nifi/templates/Simple\\_Rabbitmq\\_to\\_Mongodb.xml](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/nifi/templates/Simple_Rabbitmq_to_Mongodb.xml)

## JSON PATH EVALUATION

Un type de requête assez particulier pour requeter un json:

Je peux donc utiliser simplement \$ pour avoir l'ensemble du JSON en utilisant le processeur nifi EvaluateJSONPath:

### Inputs

☐ Output paths

#### JSONPath Syntax

\$..ValeurCapture

Example '\$.phoneNumbers[\*].type' See also [JSONPath expressions](#)

#### JSON

```
1 {
2   "NomCapture": "OuverturePorteEntree0",
3   "DateCapture": "07/07/2020 16:12:06",
4   "ValeurCapture": 0,
5   "TimestampCapture": 1594131126.942843
6 }
```

### Evaluation Results

```
1 [
2   0
3 ]
```

Malheureusement cela ne fonctionne pas à cause du champ `_id` provenant de mongodb qui est de type `ObjectID`, ce que nifi n'a pas l'air d'aimer.

Donc j'ai configuré la chose de cette façon:

## Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Destination	?	flowfile-attribute	
Return Type	?	json	
Path Not Found Behavior	?	ignore	
Null Value Representation	?	empty string	
fields.DateCapture	?	\$.DateCapture	
fields.NomCapteur	?	\$.NomCapteur	
fields.TimestampCapture	?	\$.TimestampCapture	
fields.ValeurCapture	?	\$.ValeurCapture	

Et donc je me retrouve avec les données directement dans les attributs des flowfiles comme ceci:

## FlowFile

DETAILS

ATTRIBUTES

### Attribute Values

amqp\$contentType  
application/json

fields.DateCapture  
01/06/2018 22:11:36

fields.NomCapteur  
Ampoule3

fields.TimestampCapture  
1.527883896953595E9

fields.ValeurCapture  
0

filename  
f18126e8-c1cc-4c00-bf2d-098e41e6c453

## NIFI EXPRESSION LANGUAGE

Nifi utilise un type de langage pour interagir avec les données dans le flux de données: Nifi expression language.

Ici on peut trouver le guide: <https://nifi.apache.org/docs/nifi-docs/html/expression-language-guide.html>

De manière synthétique, voici quelques exemples:

```
${filename}
${"my attribute"}
${hostname():toUpper()}
${filename:toUpper():equals('HELLO.TXT')}
${filename:equals( ${uuid} )}
```

C'est comme ça qu'on agit sur les attributs et qu'on utilise des fonctions.

## NIFI RECORDPATH: DOMAIN SPECIFIC LANGUAGE (DSL)

UpdateRecord makes use of the NiFi [RecordPath Domain-Specific Language \(DSL\)](#) to allow the user to indicate which field(s) update

## MODIFIER VALEUR EN UTILISANT UN PROCESSEUR UPDATERECORD

Tuto: <https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.UpdateRecord/>

<https://community.cloudera.com/t5/Community-Articles/Update-the-Contents-of-FlowFile-by-using-UpdateRecord/ta-p/248267>

C'est vraiment les exemples ici, qui m'ont aidé:  
<https://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.6.0/org.apache.nifi.processors.standard.UpdateRecord/additionalDetails.html>

Un processeur de type UpdateRecord a besoin de 4 paramètres minimum:


**Configure Processor**




SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field 

Property	Value
Record Reader	 No value set
Record Writer	 No value set
Replacement Value Strategy	 Literal Value

Notes:

**Record Reader** Specifies the Controller Service to use for reading incoming data

**Record Writer** Specifies the Controller Service to use for writing out the records.

**Replacement Value Strategy** Specifies how to interpret the configured replacement values

### 1.Literal Value

The **value entered** for a Property (after Expression Language has been evaluated) is **the desired value** to update the Record Fields with. Expression Language may reference variables 'field.name', 'field.type', and 'field.value' to access information about the field and the value of the field being evaluated.

### 2.Record Path Value

The value entered for a Property (after Expression Language has been evaluated) is **not the literal value to use** but rather is a **Record Path** that should be evaluated against the Record, and the result of the RecordPath will be used to update the Record. if this option is selected, and the Record Path results in **multiple values for a given Record**, the input **FlowFile** will be **routed** to the **'failure'** Relationship.

UpdateRecord 1.9.2

org.apache.nifi - nifi-standard-nar

Updates the contents of a FlowFile that contains Record-oriented data (i.e., data that can be read via a RecordReader and written by a RecordWriter). This Processor requires that at least one user-defined Property be added. The name of the Property should indicate a RecordPath that determines the field that should be updated. The value of the Property is either a replacement val...

---

Tout d'abord il a fallu que je fasse la configuration d'un processeur EvaluateJSONPath comme plus haut. Cela m'avait permis de mettre mes données dans les attributs de Flowfiles, ce qui les rendait accessibles pour un record.

Voici donc comment j'ai paramétré mon update record:

Un record par flowfile va être créé car mes données sont des objets et non des tableaux de jsons.

## Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Record Reader		JsonPathReader	→
Record Writer		JsonRecordSetWriter	→
Replacement Value Strategy		Literal Value	
//ValeurCapture		\${fields.ValeurCapture:multiply(10000)}	

Le reader et writer sont pour du JSON du coup.

J'utilise une valeur littérale car pas un JSON path pour attribuer une nouvelle valeur à mon attribut.

J'ai défini une propriété avec // car la valeur va être remplacée par la nouvelle, multipliée par 1000.

j'ai utilisé pour ça l'expression language de nifi, et donc la fonction multiply.

Pour ce qui est des deux service controllers:

le reader:

## Controller Service Details

SETTINGS

PROPERTIES

COMMENTS

### Required field

Property		Value
Schema Access Strategy	?	Infer Schema
Schema Registry	?	No value set
Schema Name	?	\${schema.name}
Schema Version	?	No value set
Schema Branch	?	No value set
Schema Text	?	\${avro.schema}
Date Format	?	No value set
Time Format	?	No value set
Timestamp Format	?	No value set
DateCapture	?	\$.DateCapture
NomCapteur	?	\$.NomCapteur
TimestampCapture	?	\$.TimestampCapture
ValeurCapture	?	\$.ValeurCapture

J'ai recréé donc les jsonpaths dont les records auraient besoin. C'est eux qui vont pouvoir être modifiés.

J'ai choisi d'inférer le schéma plutôt que de l'écrire car cela demanderait de recréer un autre service, ce que je trouve un peu lourd et long à faire.

Pour ce qui est du writer:

Il me semble que c'est la config par défaut, où le schéma est hérité mais non écrit.

La seule chose que j'ai modifié, qui posait problème pour l'insertion dans mongodb:

Pretty Print JSON	?	false
Suppress Null Values	?	Never Suppress
Output Grouping	?	One Line Per Object

Le grouping je l'ai mis en line au lieu de Array, car sinon cela faisait une erreur car mongodb attend des objets et non des listes, et je ne fais que travailler avec des objets donc nul besoin de cela. (Je pense que la fonctionnalité de Array est là pour le

cas où pour une entrée, update record créait plusieurs records, mais pour moi ce sera jamais le cas).

La j'ai réussi à modifier une valeur, mais l'idéal serait de mettre une condition pour que la mise à jour se fasse sur un type de capteur en particulier et un nom de capteur en particulier. Mais je pense ne pas avoir le temps pour cela.

## CONNECTION AVEC WEATHERSTACK

Ma clé API: 9e52a1e3368cbaced01bb0989aaee641

mail: [stephane.boucaud@edu.devinci.fr](mailto:stephane.boucaud@edu.devinci.fr)

Exemple de requête:

[http://api.weatherstack.com/current?  
access\\_key=9e52a1e3368cbaced01bb0989aaee641&query=Paris](http://api.weatherstack.com/current?access_key=9e52a1e3368cbaced01bb0989aaee641&query=Paris)

Retour:

```
"request":{
  • "type":"City",
  • "query":"Paris, France",
  • "language":"en",
  • "unit":"m"
},
• "location":{
  • "name":"Paris",
  • "country":"France",
  • "region":"Ile-de-France",
  • "lat":"48.867",
  • "lon":"2.333",
  • "timezone_id":"Europe\Paris",
  • "localtime":"2020-01-12 18:59",
  • "localtime_epoch":1578855540,
  • "utc_offset":"+1.0"
},
• "current":{
  • "observation_time":"05:59 PM",
  • "temperature":8,
  • "weather_code":122,
  • "weather_icons":[
    1. "https://assets.weatherstack.com/images/wsymbols01_png_64/wsymb01_0004_black_low_cloud.png"
  ],
  • "weather_descriptions":[
    1. "Overcast"
  ],
}
```



```

        • "wind_speed":13,
        • "wind_degree":200,
        • "wind_dir":"SSW",
        • "pressure":1026,
        • "precip":0,
        • "humidity":93,
        • "cloudcover":100,
        • "feelslike":6,
        • "uv_index":1,
        • "visibility":10,
        • "is_day":"no"
    }
}
{
  • "request":{
      • "type":"City",
      • "query":"Paris, France",
      • "language":"en",
      • "unit":"m"
    },
  • "location":{
      • "name":"Paris",
      • "country":"France",
      • "region":"Ile-de-France",
      • "lat":"48.867",
      • "lon":"2.333",
      • "timezone_id":"Europe/Paris",
      • "localtime":"2020-01-12 18:59",
      • "localtime_epoch":1578855540,
      • "utc_offset":"+1.0"
    },
  • "current":{
      • "observation_time":"05:59 PM",
      • "temperature":8,
      • "weather_code":122,
      • "weather_icons":[
          1. "https://assets.weatherstack.com/images/wsymbols01_png_64/
             wsymbol_0004_black_low_cloud.png"
        ],
      • "weather_descriptions":[
          1. "Overcast"
        ],
      • "wind_speed":13,
      • "wind_degree":200,
      • "wind_dir":"SSW",
      • "pressure":1026,
      • "precip":0,
      • "humidity":93,

```

```

    • "cloudcover":100,
    • "feelslike":6,
    • "uv_index":1,
    • "visibility":10,
    • "is_day":"no"
  }
}

```

Cela fait bcp de données et donc le but sera de filtrer pour n'avoir que current.

Tout d'abord la récupération de données de l'API:

**Configure Processor**

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

**Required field**

Property	Value
<b>URL</b>	<b>?</b> <a href="http://api.weatherstack.com/current?access_ke...">http://api.weatherstack.com/current?access_ke...</a>
<b>Filename</b>	<b>?</b> /data/weatherdata
SSL Context Service	<b>?</b> No value set
Username	<b>?</b> No value set
Password	<b>?</b> No value set
<b>Connection Timeout</b>	<b>?</b> 30 sec
<b>Data Timeout</b>	<b>?</b> 30 sec
User Agent	<b>?</b> No value set
Accept Content-Type	<b>?</b> No value set
Follow Redirects	<b>?</b> false
Redirect Cookie Policy	<b>?</b> default
Proxy Configuration Service	<b>?</b> No value set

J'utilise le même lien que celui cité précédemment.

Pour ce qui est du filtrage sur l'attribut "current":

Utilisation du processeur EvaluateJSONPath configuré comme ceci:

## Configure Processor

SETTINGS
SCHEDULING
**PROPERTIES**
COMMENTS

Required field

Property		Value
Destination	?	flowfile-content
Return Type	?	json
Path Not Found Behavior	?	ignore
Null Value Representation	?	empty string
fields.DateCapture	?	\$.current

Je me suis trompé pour le nom de la propriété mais c'est pas vraiment un souci car cela ne concerne que les attributs du flowfile.

En revanche cette fois ci j'ai choisi de mettre comme destination le contenu du flowfile, ce qui me permet d'écraser directement les données qui seront envoyées à mongodb. Donc de les filtrer.

Et le processeur pour mongodb a été adapté également:

## Processor Details

SETTINGS
SCHEDULING
**PROPERTIES**
COMMENTS

Required field

Property		Value
Client Service	?	No value set
Mongo URI	?	mongodb://mongo:27017/
Mongo Database Name	?	weather
Mongo Collection Name	?	paris
SSL Context Service	?	No value set
Client Auth	?	REQUIRED
Mode	?	insert
Upsert	?	false
Update Query Key	?	No value set
Update Query	?	No value set
Update Mode	?	With whole document
Write Concern	?	ACKNOWLEDGED
Character Set	?	UTF-8

J'ai simplement mis à jour le nom de la base de données et la collection.

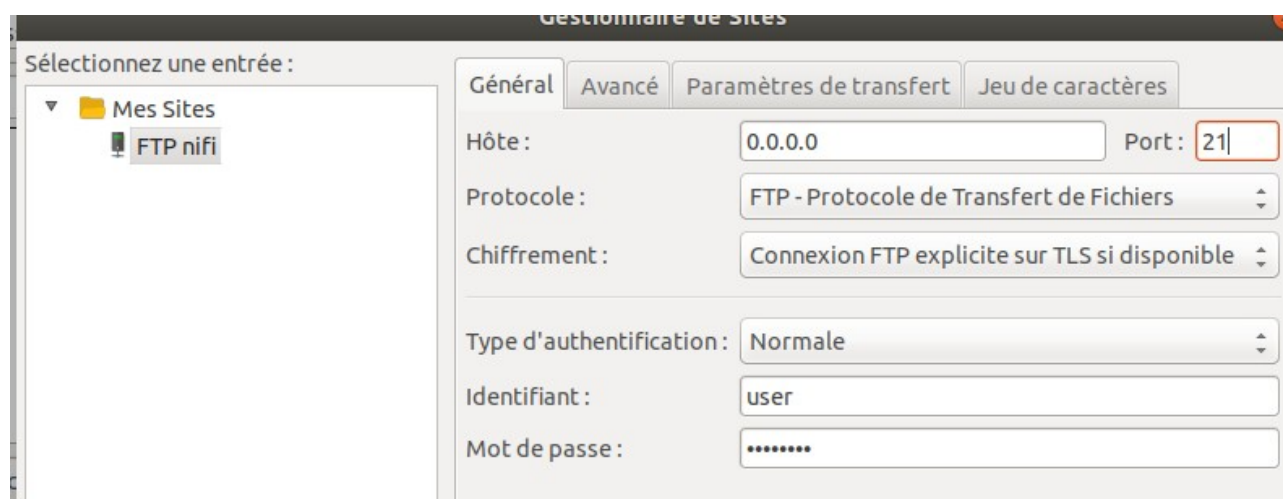
Je les ai créées à la main mais j'ai rajouté dans mon script de génération de config de mongodb ce qu'il faut pour les créer automatiquement aussi:

[https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb\\_generate\\_config.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/mongodb_generate_config.py)

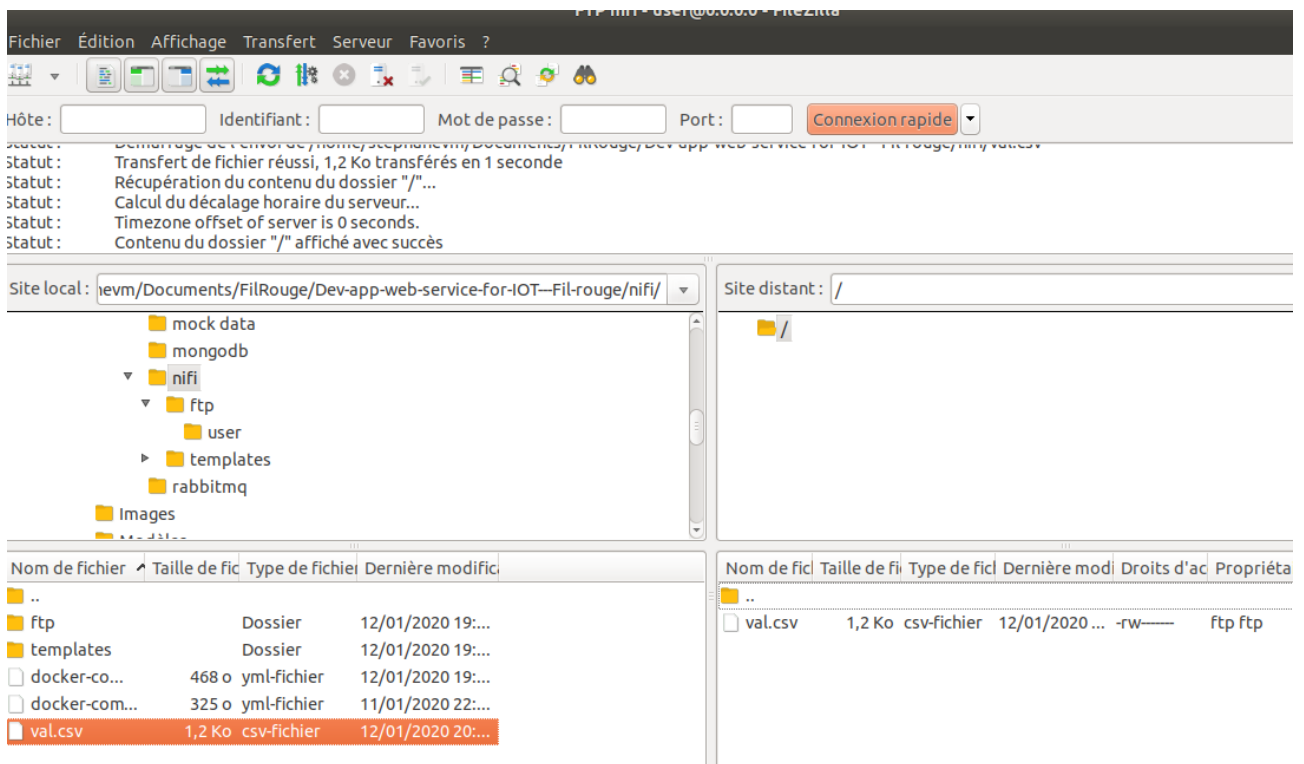
```
# creating weather collection
connection = pymongo.MongoClient("mongodb://localhost:27017/")
weather_db = connection["weather"]
country_col = weather_db["paris"]
```

## RÉCUPÉRATION CSV DEPUIS SERVEUR FTP

Voici ma connexion au serveur FTP:



Et voici mon csv envoyé sur le serveur ftp:



Voici les paramétrages du processeur GetFTP:

## Configure Processor

SETTINGS
SCHEDULING
PROPERTIES
COMMENTS

**Required field**

Property	Value
Hostname	ftp
Port	21
Username	user
Password	Sensitive value set
Connection Mode	Passive
Transfer Mode	Binary
Remote Path	No value set
File Filter Regex	No value set
Path Filter Regex	No value set
Polling Interval	60 sec
Search Recursively	false
Follow symlink	false
Ignore Dotted Files	true
Delete Original	true

Et voici pour SplitRecord:

### Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field

Property		Value	
Record Reader	?	CSVReader	→
Record Writer	?	JsonRecordSetWriter	→
Records Per Split	?	1	

Et voici le service controller pour le reader CSV:

### Controller Service Details

SETTINGS

PROPERTIES

COMMENTS

Required field

Property		Value	
Schema Access Strategy	?	Infer Schema	
Schema Registry	?	No value set	
Schema Name	?	\${schema.name}	
Schema Version	?	No value set	
Schema Branch	?	No value set	
Schema Text	?	\${avro.schema}	
CSV Parser	?	Apache Commons CSV	
Date Format	?	No value set	
Time Format	?	No value set	
Timestamp Format	?	No value set	
CSV Format	?	Custom Format	
Value Separator	?	,	
Treat First Line as Header	?	true	
Treat First Line as Header	?	true	
Ignore CSV Header Column Names	?	false	
Quote Character	?	"	false
Escape Character	?	\	
Comment Marker	?	No value set	
Null String	?	No value set	
Trim Fields	?	true	
Character Set	?	UTF-8	

Et pour le Json writer:

SETTINGS	PROPERTIES	COMMENTS
----------	------------	----------

Required field

Property	Value
<b>Schema Write Strategy</b>	<b>Do Not Write Schema</b>
Schema Cache	No value set
<b>Schema Access Strategy</b>	<b>Inherit Record Schema</b>
Schema Registry	No value set
Schema Name	\${schema.name}
Schema Version	No value set
Schema Branch	No value set
Schema Text	\${avro.schema}
Date Format	No value set
Time Format	No value set
Timestamp Format	No value set
<b>Pretty Print JSON</b>	<b>false</b>
<b>Suppress Null Values</b>	<b>Never Suppress</b>

Avec encore une fois pour Output Grouping : One line per object.

Et finalement pour putMongo:

## Configure Processor

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
----------	------------	------------	----------

Required field

Property	Value
Client Service	No value set
Mongo URI	mongodb://mongo:27017
<b>Mongo Database Name</b>	<b>FTP</b>
<b>Mongo Collection Name</b>	<b>csv</b>
SSL Context Service	No value set
Client Auth	NONE
<b>Mode</b>	<b>insert</b>
<b>Upsert</b>	<b>false</b>
Update Query Key	No value set

insert

On retrouve bien les données dans mongodb:



Mongo Express

Database: FTP ▾

Collection: csv ▾

Document 5e1b6fa48b87bc003977a91c

## Editing Document: 5e1b6fa48b87bc003977a91c

```
1 {  
2   "_id": ObjectId("5e1b6fa48b87bc003977a91c"),  
3   "date": "07/10/19 13:30",  
4   "sensor": "Test",  
5   "value": "693,349247462734"  
6 }
```

## PRÉSENTATION (TP4)

### CHOIX DE SOLUTION

Je pense qu'il serait intéressant d'afficher côte à côte les données de la maison du client à côté de celles de la météo.

## IMPLÉMENTATION

### DOCKER-COMPOSE

version: '3.3'

services:

influxdb:

image: "influxdb"

hostname: "influxdb"

ports:

- "8086:8086"

networks:

- iot-labs

labels:

NAME: "influxdb"

influxdbchrono:



```
image: "chronograf"
hostname: "chronograf"
ports:
  - "8087:8888"
networks:
  - iot-labs
labels:
  NAME: "chronograf"
command: "--influxdb-url=http://influxdb:8086"
```

networks:

iot-labs:

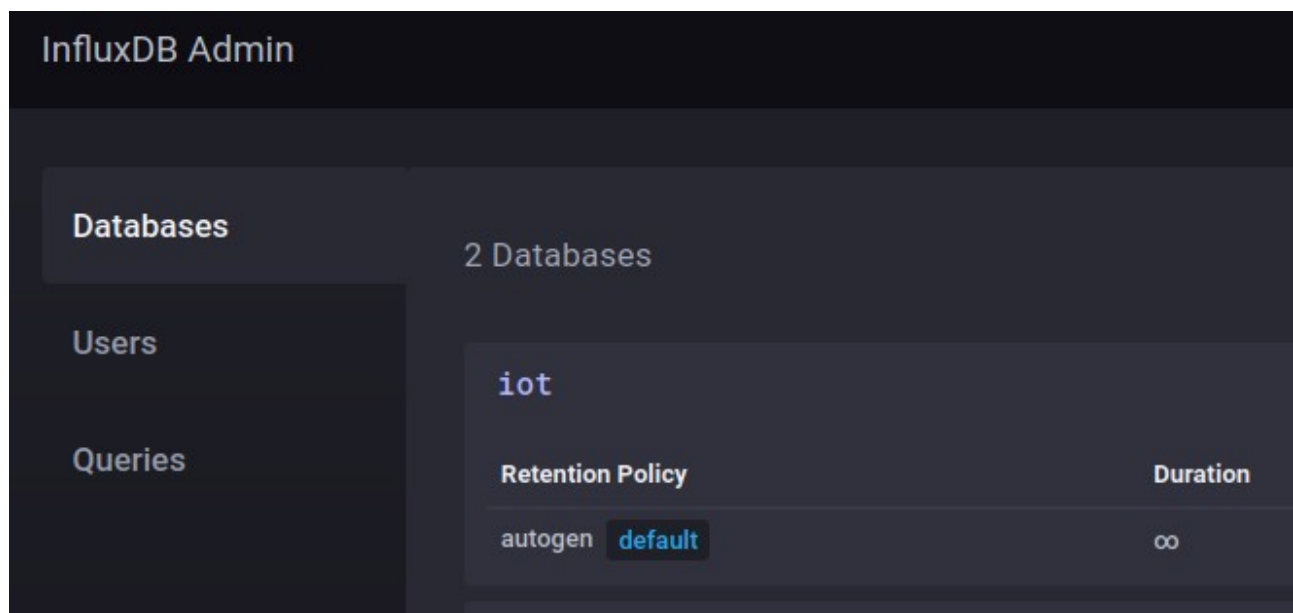
external: true

L'interface web est disponible ici:

<http://0.0.0.0:8087>

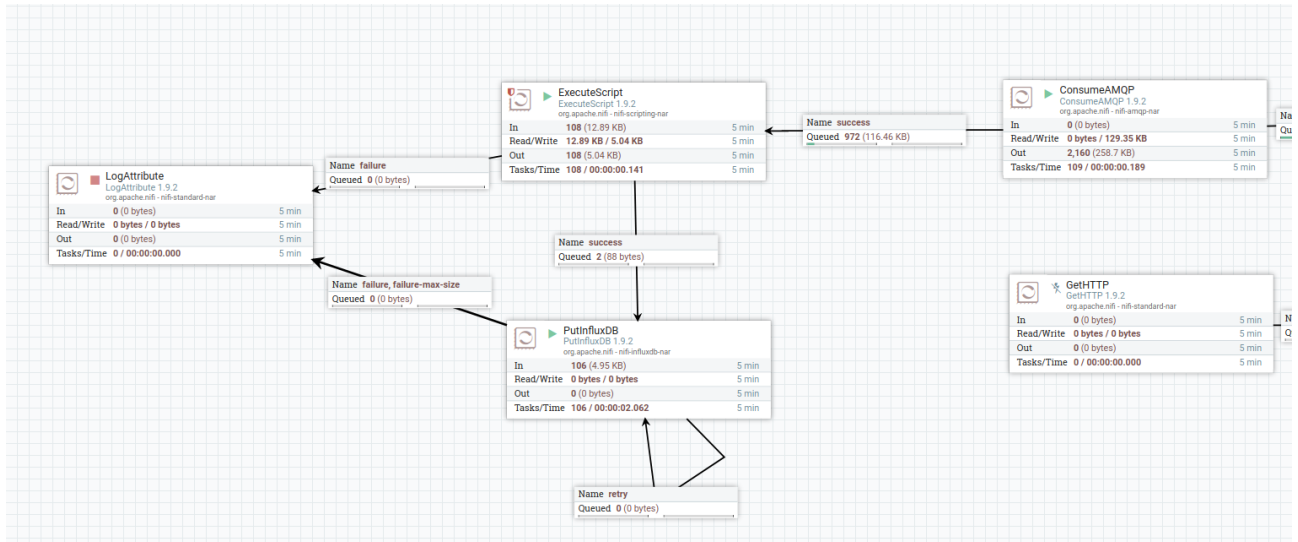
## PREMIÈRE CONFIG DE INFLUXDB

J'ai créé la bdd dont nous nous servons:



# CONFIGURATION NIFI

Voici la config requise:



En revanche j'ai eu un souci avec le processeur ExecuteScript, car il a fallu que j'adapte le code à mes données, vu que je n'utilise pas les données générées par mocker-data.

Voici donc la version qui a fonctionné pour moi:

```
flowFile = session.get();
if (flowFile != null) {
    var StreamCallback = Java.type("org.apache.nifi.processor.io.StreamCallback");
    var IOUtils = Java.type("org.apache.commons.io.IOUtils");
    var StandardCharsets = Java.type("java.nio.charset.StandardCharsets");
    var error = false;
    var measure = "Client1"
    var line = "";
    var sep = "\n"
    // Get attributes
    flowFile = session.write(flowFile, new StreamCallback(function (inputStream,
    outputStream) {
        var content = IOUtils.toString(inputStream, StandardCharsets.UTF_8); // message
    or content
        var message_content = {};
        var sensor = "";
        var date = "";
        var value = "";
        try {
            message_content = JSON.parse(content);
            for (key in message_content){
                if (key == 'NomCapteur') {
```

```

        sensor = message_content[key]
        measure = message_content[key]
    } else if (key == 'TimestampCapture') {
        date = message_content[key] * 1000 * 1000 * 1000
    } else if (key == 'ValeurCapture') {
        if (typeof message_content[key] == "number" &&
message_content[key] == 0){
            value = false;
        }
        else if (typeof message_content[key] == "number" &&
message_content[key] == 1){
            value = true;
        }
        else if (typeof message_content[key] == "string"){
            value = '"' + message_content[key] + '"';
        }
        else{
            value = message_content[key];
        }
    }
}
line = measure + " " + sensor + "=" + value + " " + date + sep
// Write output content
outputStream.write(line.getBytes(StandardCharsets.UTF_8));

} catch (e) {
    error = true;
    log.error('Something went wrong', e)
    outputStream.write(content.getBytes(StandardCharsets.UTF_8));
}
}));
if (error) {
    session.transfer(flowFile, REL_FAILURE)
} else {
    session.transfer(flowFile, REL_SUCCESS)
}
}

```

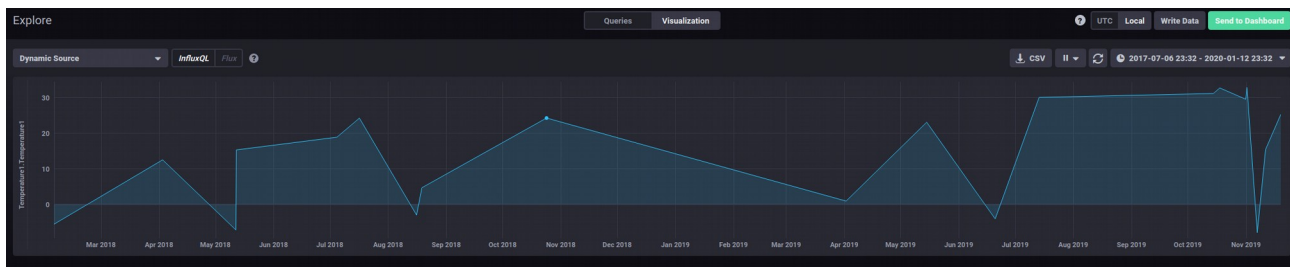
J'ai également du modifier mon fichier python de generation de données pour correspondre au format requis par influxdb.

À savoir ce format: Client1 Temperature0=201.54 1570985503

Comme mean j'utilise donc le nom du capteur car ils n'ont pas le même type de données, ni le même endroit ni la même fonction.

## PREMIER AFFICHAGE DE DONNÉES

Voici une visualisation des données pour le capteur Temperature1:



Pour le faire j'ai utilisé la requête: `SELECT * FROM "iot"."autogen"."Temperature1"`

Où temperature est le measurement mais aussi le nom du capteur.

Ce sont des données générées aléatoirement d'où les pics aussi forts.

## GRAFANA

Docker-compose:

```
version: '3.3'
services:
  grafana:
    image: "grafana/grafana"
    hostname: "grafana"
    environment:
      GF_SECURITY_ADMIN_PASSWORD: "secret"
    ports:
      - "8084:3000"
    networks:
      - iot-labs
    labels:
      NAME: "grafana"
networks:
  iot-labs:
    external: true
```


on peut y acceder comme ça:

<http://0.0.0.0:8084/?orgId=1>

On utilise le compte admin et mot de passe secret

Voici un aperçu des données pour le capteur Temperature0:

Ensuite pour configurer la connection à influxDB:



## Data Sources / InfluxDB

Type: InfluxDB

Settings

Name

InfluxDB

Default

### HTTP

URL

http://influxdb:8086

Access

Server (default)

[Help ▶](#)

Whitelisted Cookies

Add Name

Add

### Auth

Basic auth

With Credentials

TLS Client Auth

With CA Cert

Skip TLS Verify

Forward OAuth Identity

### InfluxDB Details

Database

iot

User

Password

Password

HTTP Method

GET



En haut on a la variation des temperatures, et en bas la valeur moyenne.

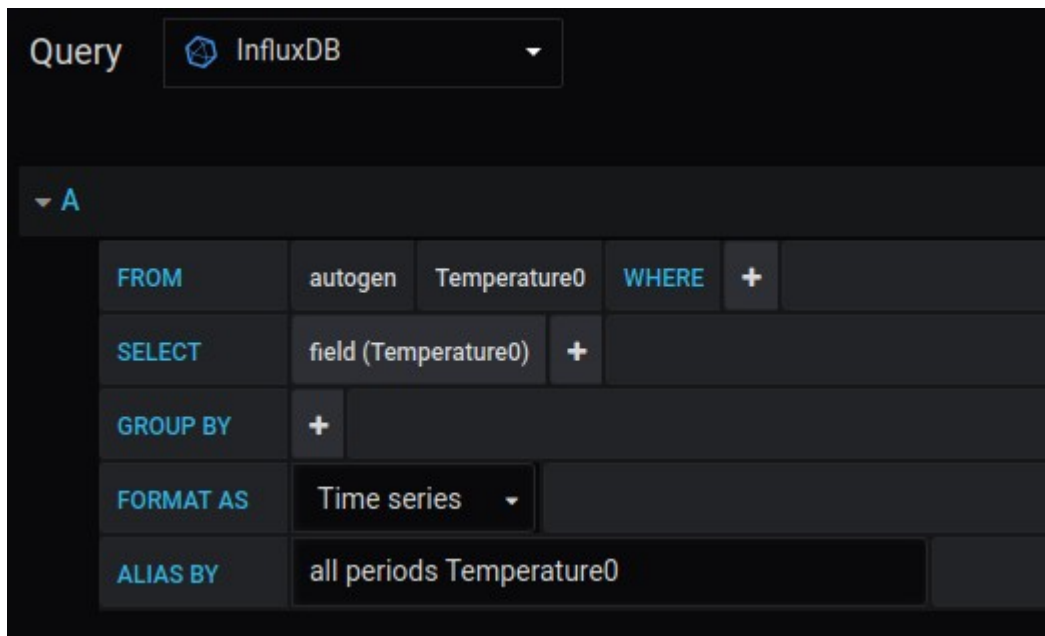
Pour la moyenne, la requete est celle ci:

Query InfluxDB ▾

▾ A

FROM	autogen	Temperature0	WHERE	+	
SELECT	field (Temperature0)	mean ()	+		
GROUP BY	time (\$__interval)	fill (null)	+		
FORMAT AS	Time series ▾				
ALIAS BY	Mean Temperature0				

Et pour les temperature de la longue periode:



C'est une interface vraiment pratique pour avoir une visualisation de données rapidement et efficacement.

## API (TP5)

---

Pour préparer Swagger, il a fallu que j'adapte un peu ma commande docker:

```
docker run --rm -v "${PWD}/API Swagger":/local swaggerapi/swagger-codegen-cli-v3:3.0.14
generate -i /local/timeseries_iot.yaml -l python-flask -DpackageName=TimeSeriesIoT -o
/local/out/python-ts-iot
```

Mais ensuite, j'ai bien trouvé un fichier Dockerfile généré:

```
FROM python:3.6-alpine

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/

RUN pip3 install --no-cache-dir -r requirements.txt

COPY . /usr/src/app

EXPOSE 8080
```

```
ENTRYPOINT ["python3"]

CMD ["-m", "TimeSeriesIoT"]
```

## Je l'ai mis à jour:

```
FROM python:3.6-alpine

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY requirements.txt /usr/src/app/

RUN pip3 install --no-cache-dir -r requirements.txt

RUN pip3 install connexion[swagger-ui]

COPY . /usr/src/app

EXPOSE 8080

ENTRYPOINT ["python3"]

CMD ["-m", "TimeSeriesIoT"]
```

## Et voici mon docker compose que j'ai modifié pour pointer sur le dockerfile:

```
version: '3.3'
services:
  api:
    build: ./out/python-ts-iot
    image: "myapi"
    hostname: "myapi"
    ports:
      - "8080:8080"
    networks:
      - iot-labs
    labels:
      NAME: "myapi"
networks:
  iot-labs:
    external: true
```

Voici l'adresse à laquelle je pourrai faire mes appels API: <http://0.0.0.0:8080/>



Avant de pouvoir coder les différentes requêtes à faire sur mongodb, il me faut d'abord modifier le fichier requirements.txt: de cette manière je pourrai par exemple inclure pymongo.

Voici donc ce que l'on a dans ce fichier:

```
connexion == 2.2.0
python_dateutil == 2.6.0
setuptools >= 21.0.0
pymongo == 3.10.1
```

Mais on va faire de même pour le fichier requirements de tests.

On va ensuite mettre à jour la fonction mean\_sensor\_id\_get du fichier controllers/default\_controller.py:

On va récupérer mon code depuis le fichier [https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get\\_functions.py](https://github.com/Stephane-Bcd/Dev-app-web-service-for-IOT---Fil-rouge/blob/master/mongodb/get_functions.py), plus particulièrement les fonctions;

Puis on modifie la fonction qui était déjà présente pour renvoyer les données lors d'un appel API:

```
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]
    """

    return get_avg_in_period(sensor_name=sensor_id,
start=datetime.fromtimestamp(start_date), end=datetime.fromtimestamp(end_date))
```

Ensuite, je "compile le tout" en faisant un docker build:

```
docker build API\Swagger/out/python-ts-iot
```

Et on refait un docker-compose:

```
docker-compose -f API\Swagger/docker-compose.yml up -d --force-recreate
```

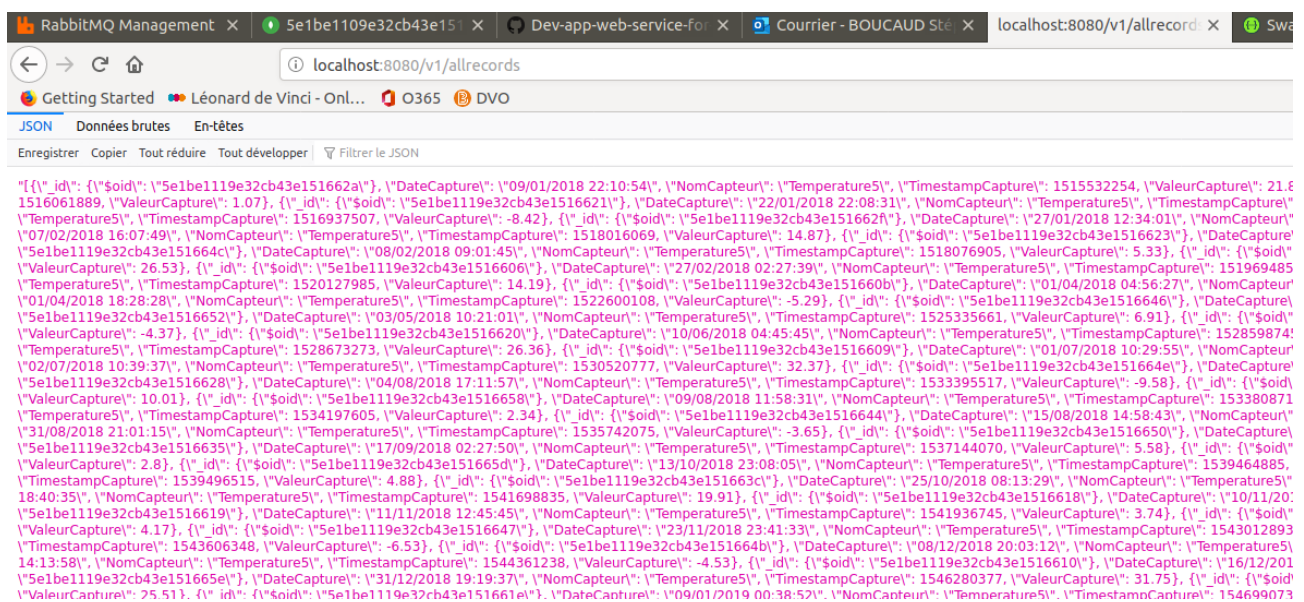
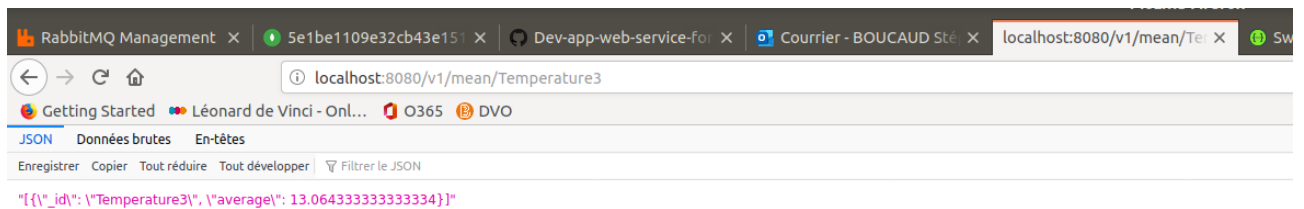
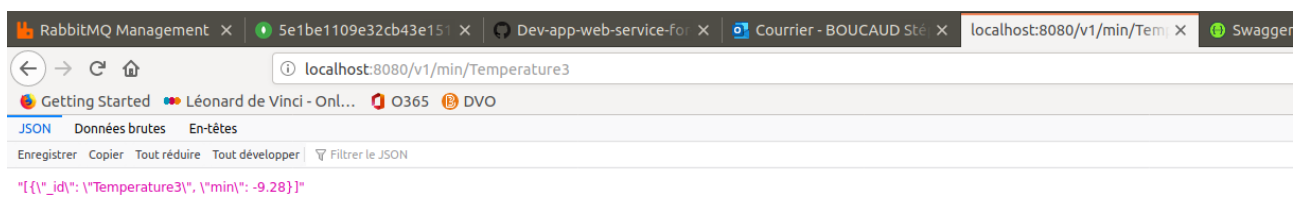
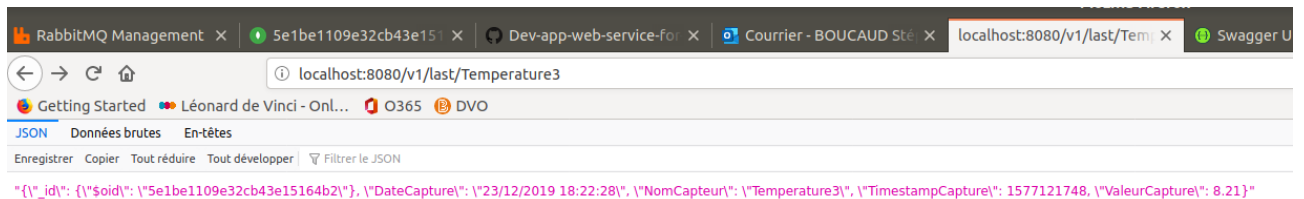
Je l'ai fait mais pour une certaine raison cela ne met pas à jour le résultat sorti dans le web..

Dans mon code j'ai mis à jour l'adresse ip du container de mongodb:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
mongodb_mongo_1
```

172.19.0.3

Et ai modifié le type de retour et ça a marché comme sur des roulettes:



# CONCLUSION GÉNÉRALE

---

Avec tous les outils mis en place il est plus simple de simaginer comment on peut gérer une infrastructure cloud de manière à la rendre modulable, pratique, et répondre aux besoins d'un client.

Avec ce projet on a su partir des objets connectés jusqu'aux dashboards et API web, c'était très enrichissant.

Rabbitmq qui sert de récupérateur de données, nifi, son dispatcher et modificateur, les bases de données , les apis, les interfaces web. Tout pour gère au mieux son projet de manière scalable.

En plus, tout est dockérisé donc cela facilite d'autant plus la tâche et offre des autres possibilités pour les très grandes infrastructures.

## QUESTIONS POUR LE PROFESSEUR

---

### VAGUE 1

Pour ce qui est de l'ingestion de donnée avec RabbitMQ,

Ne faudrait il pas:

1 container par client complètement isolés niveau réseau (networks différents)

1 exchange de type topic par Maison chez le client.

1 Vhost par client dans RabbitMQ pour séparer les clients de manière sécurisée plutôt que des containers

1 exchange par maison

Les topics étant définis dans les noms des capteurs par exemple : « Capteur\_Chambre1\_Temp1 » irait dans la queue Chambre1\_Temp1 et dans la queue Temp en général. Voire même peut être dans la queue Maison 1 ?

Quand un message peut entrer dans plusieurs topics (selon le sélecteur) ; comment le topic est choisi ? (quel ordre de priorités?). Le message est il dupliqué ?

====> Copié,

Pas vraiment l'intérêt de copier dans des queues plus globales (vue sur la france etc) car il n'y a pas besoin d'une forte latence dans notre cas : On pourra le faire par la suite en requêtant simplement la BDD

La séparation en queues : but pas de structurer les données mais juste de les repartir :

Donc intérêt de la modularité de l'infrastructure.

## VAGUE 2

Pour créer des mots de passes pour les échanges ou les queues, au final on est obligés de créer à chaque fois un virtual host puis lui attribuer un utilisateur ?

Si oui : Mon idée serait donc d'avoir pour chaque client 3 Vhosts : 1 pour envoyer ses données par un échange et récupérer les nouvelles données depuis la queue, 1 pour que un consommateur récupère les messages de la queue, 1 pour qu'une autre source renvoie des données au client via un échange.

Est ce juste ?

Serait il possible également d'avoir un retour de votre part concernant mon architecture actuelle ? (Voir ci dessus)

## VAGUE 3 (APRÈS LECTURE)

Alors vous aimez ? Si oui mettez un j'aime:

