

Documentation technique application IoT Python

Equipe 11 – AL MASRI Marwan, NICHELE Angelo, BABELA Guychel et KRILL Maxence

Table des matières

Présentation	3
Ressources	3
Fonctionnalités	3
Structure du code.....	3
Analyse	4
Imports et déclaration.....	4
Abonnement au flux MQTT	4
Gestion des données	4
Ecriture des données	5
Connexion au serveur MQTT	5
Gestion des exceptions	5
Commentaires.....	5
Utilisation et configuration	5
Exemple d'utilisation (étapes à suivre).....	6
Cas de test.....	6
Cas de test nominal.....	6
Fichier de configuration config.yaml	6
Lecture des données	6
Fichier data.json avec les données récoltées	7
Fichier alerte.json vide car aucune alerte	7
Cas de test des alertes.....	7
Fichier de configuration config.yaml (volontairement faussé pour forcer les alertes).....	7
Lecture des données	8
Fichier data.json avec les données récoltées	8
Fichier alerte.json avec les alertes relevées.....	9
Conclusion	9

Présentation

Le code Python présentée ici fait partie d'un système de gestion des données de capteurs déployés à l'IUT Blagnac. Son rôle est d'accéder aux données des capteurs, de les traiter selon des paramètres configurables, et d'écrire ces données dans des fichiers. L'application s'interface avec un serveur MQTT pour récupérer les données des capteurs et utilise un fichier de configuration pour définir son comportement.

A noter que le présent document se concentre uniquement sur la partie Python de l'application IoT. Le système global comprend également une application Java qui permet de configurer le comportement de l'application Python et d'afficher les valeurs captées.

Ressources

Langages utilisés : Python, JSON

Système d'exploitation : Linux, Windows, Mac

IDE : Visual Studio Code

Librairies utilisées : os, yaml, json, signal, threading, sys, time, paho.mqtt.client

Fonctionnalités

Le code Python offre plusieurs fonctionnalités essentielles pour la gestion des données de capteurs. Voici un aperçu de ses principales fonctionnalités :

- Lecture du fichier de configuration : Le code lit un fichier de configuration (`config.yaml`) pour déterminer son comportement. Ce fichier spécifie des paramètres tels que l'hôte (`chirpstack.iut-blagnac.fr`) et le port du serveur MQTT (1883), le topic auquel s'abonner (`AM107/by-room/+data`), les types de données à collecter (température, activité, CO2, pression, humidité), les seuils d'alerte, ...
- Lecture des données MQTT : Le code se connecte au serveur MQTT (`chirpstack.iut-blagnac.fr`) et s'abonne au topic spécifié dans le fichier de configuration. Elle extrait les données publiées sur le bus MQTT, les analyse et les traite salle par salle, évitant tout mélange de données.
- Gestion de la lecture du bus : L'application gère la lecture du bus MQTT à une fréquence définie dans le fichier de configuration. Sur les systèmes Unix, elle utilise des appels système pour assurer la régularité de la lecture.
- Écriture des données dans des fichiers : Les valeurs captées sont écrites dans des fichiers au format JSON au bout d'une intervalle donnée.
- Détection des dépassements de seuils d'alerte : L'application surveille les valeurs captées et détecte les dépassements des seuils d'alerte définis dans le fichier de configuration. En cas de dépassement, elle enregistre les données anormales dans le fichier d'alerte.

Structure du code

Le code Python est structuré en plusieurs fonctions pour faciliter la lisibilité et la maintenance. Voici un aperçu des principales fonctions :

- on_connect : Cette fonction est appelée lors de la connexion au serveur MQTT. Elle gère l'abonnement au topic spécifié dans le fichier de configuration et initialise les intervalles de lecture en fonction du système d'exploitation.

- on_message : Cette fonction est appelée lorsqu'un message est reçu du serveur MQTT. Elle extrait les données, les traite salle par salle, et détecte les dépassements de seuils d'alerte.
- write : Cette fonction gère l'écriture des données dans les fichiers. Elle prend en charge les différences entre les systèmes Unix (appels systèmes) et autres OS.
- unixwrite : Fonction spécifique à Unix pour appeler la fonction write sans paramètres
- dataload et readfile : Ces fonctions gèrent la lecture du fichier de configuration et des fichiers de données. Elles initialisent les variables nécessaires au fonctionnement du code.

Analyse

Le code Python fourni présente une application de gestion des données de capteurs, qui se connecte à un serveur MQTT, récupère les données des capteurs, les traite, détecte les alertes et enregistre le tout dans des fichiers. Voici une analyse détaillée des aspects techniques :

Imports et déclaration

Les bibliothèques `os`, `yaml`, `json`, `signal`, `threading`, `sys`, `time` et `paho.mqtt.client` sont importées pour le bon fonctionnement de l'application : connexion au flux MQTT, gestion des fichiers, signaux Unix, `threading`, ...

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python3

import os, yaml, json, signal, threading, sys, time
import paho.mqtt.client as mqtt
```

Abonnement au flux MQTT

`on_connect` est appelé lors de la connexion au serveur MQTT et s'occupe de l'abonnement au topic spécifié.

```
def on_connect(client, userdata, flags, rc):
    ...
```

`on_message` est appelé lors de la réception d'un message MQTT, decode le message, traite les données salle par salle, et détecte les éventuelles alertes. Le tout est stocké dans des variables tableaux qui seront par la suite écrites dans les fichiers JSON : `config`, `data`, et `alerte`.

```
def on_message(client, userdata, msg):
    ...
```

Gestion des données

`dataload` initialise les variables globales `config`, `data`, et `alerte` en lisant le fichier de configuration et les fichiers de données.

```
def dataload():
    ...
```

`readfile` gère la lecture des fichiers de données, les initialise s'ils n'existent pas, et retourne leur contenu.

```
def readfile(file):
    ...
```

Ecriture des données

write écrit les données dans les fichiers, gère les différences entre les systèmes Unix et les autres OS, et redémarre le timer ou programme le prochain signal suivant l'OS.

```
| def write():  
|     ...
```

unixwrite est une fonction spécifique à Unix pour appeler write à intervalles réguliers sans les paramètres retournées par l'alerte Unix (numero et frame) qui ne nous sont pas utiles.

```
| def unixwrite(numero, frame):  
|     ...
```

Connexion au serveur MQTT

Les variables sont initialisées en appelant dataload. Un client MQTT est créé et connecté au serveur en utilisant les informations du fichier de configuration.

```
| dataload()  
  
| client = mqtt.Client()  
| client.on_connect = on_connect  
| client.on_message = on_message  
| client.connect(config["connection"]["host"],  
| config["connection"]["port"], 60)  
| client.loop_forever()
```

Gestion des exceptions

Le code gère automatiquement les exceptions en retournant des messages d'erreurs.

Commentaires

Le code est accompagné de commentaires clairs expliquant chaque étape, facilitant la compréhension du code.

Utilisation et configuration

Si ce n'est pas déjà fait, assurez-vous d'avoir Python (version 3.11+) sur votre machine, correctement configuré, et d'avoir les dépendances nécessaires. Vous pouvez les installer avec la commande :

```
| pip install pyyaml paho-mqtt
```

Avant d'exécuter le code, assurez-vous d'avoir un fichier de configuration (config.yaml) correctement renseigné (nous recommandons d'utiliser l'application JavaFX fournie à cet effet) dans le même répertoire que l'application Python main.py.

Ce fichier spécifie les paramètres essentiels tels que l'hôte MQTT, le port, le topic, les types de données à collecter, les seuils d'alerte, etc...

Pour lancer le code, placez-vous dans le dossier avec l'ensemble des fichiers et exécutez le script main.py avec la commande :

```
| python3 main.py
```

Vous obtenez un message confirmant le bon démarrage une fois l'application connectée.

Exemple d'utilisation (étapes à suivre)

Voici un exemple d'utilisation du code Python :

1. Assurez-vous que le fichier de configuration (`config.yaml`) est correctement configuré.
2. Exécutez le script `main.py` avec la commande `python3 main.py`.
3. Le code se connectera au serveur MQTT, lira les données des capteurs, les traitera et les écrira dans les fichiers spécifiés (`data.json` et `alerte.json`).
4. Les éventuelles alertes seront détectées et enregistrées dans les fichiers d'alerte (`alerte.json`).
5. Pour arrêter le programme, appuyez sur `Ctrl+C`.


Cas de test




Cas de test nominal

Fichier de configuration `config.yaml`

```
connection:
  host: chirpstack.iut-blagnac.fr
  port: 1883
  topic: AM107/by-room/+/data
ecriture:
  fichiers:
    data: data
    alerte: alerte
  intervalle: 10
collecte:
  - temperature
  - activity
  - co2
  - pressure
  - humidity
alerte:
  co2:
    min: 500.0
    max: 900.0
  humidity:
    min: 20.0
    max: 80.0
  pressure:
    min: 800.0
    max: 1300.0
  temperature:
    min: 15.0
    max: 25.0
```

Lecture des données

 Connecté avec le code 0 à `chirpstack.iut-blagnac.fr:1883` sur le topic `AM107/by-room/+/data`

-  Données reçues de la salle " hall-amphi" (9 données)
-  Données reçues de la salle "E103" (9 données)
-  Données enregistrées

Fichier data.json avec les données récoltées

```
{
  " hall-amphi": [
    {
      "date": "1705256752",
      "temperature": 19.9,
      "humidity": 61,
      "activity": 87,
      "co2": 718,
      "pressure": 996
    }
  ],
  "E103": [
    {
      "date": "1705256754",
      "temperature": 21.4,
      "humidity": 57,
      "activity": 178,
      "co2": 939,
      "pressure": 996.3
    }
  ]
}
```

Fichier alerte.json vide car aucune alerte

```
{}
```


Cas de test des alertes


Fichier de configuration config.yaml (volontairement faussé pour forcer les alertes)


```
connection:
  host: chirpstack.iut-blagnac.fr
  port: 1883
  topic: AM107/by-room/+/data
ecriture:
  fichiers:
    data: data
    alerte: alerte
  intervalle: 10
collecte:
  - temperature
  - activity
  - co2
  - pressure
  - humidity
alerte:
```


```
co2:
  min: 0.0
  max: 10.0
humidity:
  min: 0.0
  max: 10.0
pressure:
  min: 0.0
  max: 10.0
temperature:
  min: 0.0
  max: 10.0
```


Lecture des données


 Connecté avec le code 0 à chirpstack.iut-blagnac.fr:1883 sur le topic AM107/by-room/+/data


 Données reçues de la salle "E207" (9 données)


 Anomalie dans la salle "E207" pour la donnée "temperature" : 17.2


 Anomalie dans la salle "E207" pour la donnée "humidity" : 40.5


 Anomalie dans la salle "E207" pour la donnée "co2" : 459


 Anomalie dans la salle "E207" pour la donnée "pressure" : 988.3


 Données reçues de la salle "E003" (9 données)

 Anomalie dans la salle "E003" pour la donnée "temperature" : 17.6

 Anomalie dans la salle "E003" pour la donnée "humidity" : 43

 Anomalie dans la salle "E003" pour la donnée "co2" : 467

 Anomalie dans la salle "E003" pour la donnée "pressure" : 989.1

 Données enregistrées

Fichier data.json avec les données récoltées

```
{
  "E207": [
    {
      "date": 1705256791,
      "temperature": 17.2,
      "humidity": 40.5,
      "activity": 0,
      "co2": 459,
      "pressure": 988.3
    }
  ],
  "E003": [
    {
      "date": 1705256792,
      "temperature": 17.6,
      "humidity": 43,
      "activity": 0,
      "co2": 467,
      "pressure": 989.1
    }
  ]
}
```



```
}  
]  
}
```

Fichier alerte.json avec les alertes relevées

```
{  
  "E207": [  
    {  
      "date": 1705256791,  
      "temperature": 17.2,  
      "humidity": 40.5,  
      "co2": 459,  
      "pressure": 988.3  
    }  
  ],  
  "E003": [  
    {  
      "date": 1705256792,  
      "temperature": 17.6,  
      "humidity": 43,  
      "co2": 467,  
      "pressure": 989.1  
    }  
  ]  
}
```

Conclusion

Ce document technique a présenté les principaux aspects du code Python pour la gestion des données de capteurs. Le code offre une bonne solution pour collecter, traiter et enregistrer les données, tout en détectant les dépassements de seuils d'alerte. Sa modularité facilite la maintenance et les évolutions futures.