Introduction à MQTT

Prérequis

Pour aborder ce cours, il est essentiel d'avoir des bases solides en programmation, notamment en Python. Une connaissance des structures de données, des fonctions et des boucles est indispensable. De plus, une familiarité avec les concepts de base des réseaux informatiques, tels que les adresses IP et les ports, sera utile.

Ce cours s'inscrit dans le programme de Numérique et Sciences Informatiques (NSI) de Terminale Générale, plus précisément dans le chapitre consacré à l'Internet des Objets (IoT) et aux protocoles de communication. Il vient après l'étude des bases de la programmation et des réseaux, et prépare à des projets plus avancés impliquant la collecte et l'échange de données entre des dispositifs connectés.

Chapitre 1 : Les Fondamentaux de MQTT

1.1 Introduction à l'Internet des Objets (IoT)

L'Internet des Objets (IoT) désigne l'interconnexion de dispositifs physiques (objets) via Internet, leur permettant de collecter et d'échanger des données. Ces objets peuvent être des capteurs, des actionneurs, des appareils électroménagers, des véhicules, etc. MQTT (Message Queuing Telemetry Transport) est un protocole de communication léger, conçu spécifiquement pour l'IoT.

1.2 Qu'est-ce que MQTT ?

MQTT est un protocole de messagerie de type *publish-subscribe* (publication-abonnement). Contrairement aux protocoles de communication directe (comme HTTP), MQTT utilise un *broker* (courtier) central qui reçoit les messages des *publishers* (éditeurs) et les distribue aux *subscribers* (abonnés) intéressés.

- **Publisher (Éditeur) :** Un dispositif qui envoie des messages au broker.
- **Subscriber (Abonné) :** Un dispositif qui s'abonne à un ou plusieurs *topics* (sujets) pour recevoir les messages correspondants.
- **Broker (Courtier) :** Un serveur qui reçoit les messages des publishers et les distribue aux subscribers.
- **Topic (Sujet) :** Une chaîne de caractères hiérarchique utilisée pour catégoriser les messages. Par exemple : “maison/temperature/salon”.

1.3 Le Modèle Publish-Subscribe

Le modèle publish-subscribe offre plusieurs avantages :

- **Découplage :** Les publishers et les subscribers ne connaissent pas l'existence l'un de l'autre. Ils interagissent uniquement avec le broker.
- **Scalabilité :** Le broker peut gérer un grand nombre de publishers et de subscribers.
- **Efficacité :** Les messages sont envoyés uniquement aux subscribers intéressés, ce qui réduit la bande passante utilisée.

1.4 Les Niveaux de Qualité de Service (QoS)

MQTT propose trois niveaux de qualité de service (QoS) pour garantir la fiabilité de la transmission des messages :

- **QoS 0 (At Most Once) :** Le message est envoyé une seule fois, sans confirmation de réception. C'est le niveau le plus rapide, mais le moins fiable.
- **QoS 1 (At Least Once) :** Le message est envoyé au moins une fois, avec une confirmation de réception. Le message peut être envoyé plusieurs fois en cas de problème de réseau.
- **QoS 2 (Exactly Once) :** Le message est envoyé exactement une fois, même en cas de problème de réseau. C'est le niveau le plus fiable, mais le plus lent.

Chapitre 2 : MQTT en Pratique avec Python

2.1 Installation de la Bibliothèque Paho MQTT

Pour utiliser MQTT en Python, nous allons utiliser la bibliothèque Paho MQTT. Vous pouvez l'installer avec la commande suivante :

```bash
pip install paho-mqtt
```

2.2 Publication d'un Message

Voici un exemple de code Python pour publier un message sur un broker MQTT :

```python
import paho.mqtt.client as mqtt
```

Configuration du broker

```
broker_address = "test.mosquitto.org" # Broker public pour les tests
port = 1883
topic = "mon/topic"
```

```
message = "Bonjour le monde !"
```

Création du client MQTT

```
client = mqtt.Client()
```

Connexion au broker

```
client.connect(broker_address, port, 60)
```

Publication du message

```
client.publish(topic, message)
```

Déconnexion du broker

```
client.disconnect()
```
```

### 2.3 Abonnement à un Topic

Voici un exemple de code Python pour s'abonner à un topic et recevoir les messages :

```python

```
import paho.mqtt.client as mqtt
```

Configuration du broker

```
broker_address = "test.mosquitto.org"
port = 1883
topic = "mon/topic"
```

Fonction de callback pour gérer les messages reçus

```
def on_message(client, userdata, msg):
 print(f"Message reçu sur le topic {msg.topic}: {msg.payload.decode()}")
```

Création du client MQTT

```
client = mqtt.Client()
```

Connexion au broker

```
client.connect(broker_address, port, 60)
```

Abonnement au topic

```
client.subscribe(topic)
```

Définition de la fonction de callback

```
client.on_message = on_message
```

Boucle principale pour écouter les messages

```
client.loop_forever()
```
```

### 2.4 Utilisation de JSON pour les Payloads

JSON (JavaScript Object Notation) est un format d'échange de données léger et facile à lire. Il est souvent utilisé pour structurer les payloads (contenus) des messages MQTT.

En Python, vous pouvez utiliser la bibliothèque `json` pour encoder et décoder les données JSON.
```

```python
import json
```

Création d'un dictionnaire Python

```
data = {
 "temperature": 25.5,
 "humidity": 60.2,
 "timestamp": "2023-10-27T10:00:00"
}
```

Conversion du dictionnaire en chaîne JSON

```
json_data = json.dumps(data)
```

Publication du message JSON

```
client.publish(topic, json_data)
```

Dans la fonction on_message, décoder le JSON

```
def on_message(client, userdata, msg):
 json_payload = json.loads(msg.payload.decode())
 print(f"Température: {json_payload['temperature']}")
 print(f"Humidité: {json_payload['humidity']}")
```

Chapitre 3 : Exercices et Applications

3.1 Exercice 1 : Capteur de Température

Créez un programme Python qui simule un capteur de température. Le programme doit publier la température actuelle (une valeur aléatoire entre 20 et 30 degrés Celsius) sur un topic MQTT toutes les 5 secondes, en utilisant un payload JSON contenant la température et un timestamp.

Corrigé Guidé :

1. Importez les bibliothèques `paho.mqtt.client`, `json`, `time` et `random`.
2. Définissez les paramètres du broker, du topic et de la fréquence de publication.
3. Créez un client MQTT et connectez-vous au broker.
4. Dans une boucle infinie, générez une température aléatoire, créez un dictionnaire Python contenant la température et le timestamp, convertissez le dictionnaire en chaîne JSON, publiez le message sur le topic, et attendez 5 secondes.
5. N'oubliez pas de déconnecter le client MQTT lorsque le programme se termine.

3.2 Exercice 2 : Contrôle d'un Actionneur

Créez un programme Python qui s'abonne à un topic MQTT et contrôle un actionneur (par exemple, une LED) en fonction des messages reçus. Si le message reçu est “ON”, allumez l'actionneur. Si le message reçu est “OFF”, éteignez l'actionneur.

Corrigé Guidé :

1. Importez les bibliothèques `paho.mqtt.client` et `json`.
2. Définissez les paramètres du broker et du topic.
3. Créez un client MQTT et connectez-vous au broker.
4. Abonnez-vous au topic.
5. Définissez une fonction de callback `on_message` qui reçoit les messages. Dans cette fonction, décodez le message et, en fonction de sa valeur (“ON” ou “OFF”), allumez ou éteignez l'actionneur (simulé par une simple instruction `print`).
6. Démarrez la boucle principale pour écouter les messages.

Résumé

- **IoT (Internet des Objets)** : Interconnexion de dispositifs physiques via Internet.
- **MQTT (Message Queuing Telemetry Transport)** : Protocole de messagerie de type publish-subscribe, léger et adapté à l'IoT.
- **Publisher (Éditeur)** : Dispositif qui envoie des messages.
- **Subscriber (Abonné)** : Dispositif qui reçoit des messages.
- **Broker (Courtier)** : Serveur central qui distribue les messages.
- **Topic (Sujet)** : Chaîne de caractères hiérarchique pour catégoriser les messages.
- **QoS (Quality of Service)** : Niveau de qualité de service pour garantir la fiabilité de la transmission des messages (QoS 0, QoS 1, QoS 2).
- **JSON (JavaScript Object Notation)** : Format d'échange de données léger et facile à lire, souvent utilisé pour structurer les payloads des messages MQTT.
- La bibliothèque `paho-mqtt` en Python permet d'implémenter facilement le protocole MQTT.
- La fonction `json.dumps()` convertit un dictionnaire Python en chaîne JSON.
- La fonction `json.loads()` convertit une chaîne JSON en dictionnaire Python.