# TP 3 SUITE: Communication MQTT avec Mosquitto

## 🎯 Objectif

À l'issue de ce Travail Pratique, vous serez capables de :

- Comprendre le fonctionnement **réel d'un broker MQTT** au sein d'un réseau local.
- Installer, configurer et utiliser le broker **Eclipse Mosquitto**.
- Connecter un microcontrôleur **ESP32 (réel ou simulé)** à votre propre broker.
- Publier (**Publish**) et recevoir (**Subscribe**) des messages MQTT en temps réel.

## 🧠 Rappel (Lien avec le TP précédent)

> **Contexte :** Dans le TP précédent, vous avez utilisé un broker public hébergé sur le cloud (ex:
>
> `broker.mqttdashboard.com`
>
> ou
>
> `test.mosquitto.org`
>
> ).
>
> ➡️ **Aujourd'hui :** Vous allez franchir une étape cruciale en IoT en **créant votre propre broker local**. Cela garantit une meilleure sécurité, une latence minimale et une indépendance vis-à-vis d'Internet.

## Architecture du système

Le flux de communication pour ce TP est structuré de la manière suivante :

```
ESP32 ➔ Wi-Fi ➔ Mosquitto (PC Local) ➔ MQTT Explorer / Client Web
```

## 🧰 Matériel Requis

- Une carte de développement **ESP32** (et son câble USB).
- Un **Ordinateur** (Windows, macOS ou Linux) connecté au même réseau Wi-Fi que l'ESP32.
- Le logiciel broker **Mosquitto**.
- Le logiciel client **MQTT Explorer** (ou HiveMQ Web Client en alternative).

## 🧪 TP 3 SUITE.1 : Installation de Mosquitto

Mosquitto est un broker MQTT open-source léger et très populaire. Suivez les instructions correspondant à votre système d'exploitation.

### 🔹 Pour Windows

1. **Télécharger Mosquitto :** Rendez-vous sur la page officielle de téléchargement ou utilisez les liens directs ci-dessous. La version 2.1.2 est la plus récente au moment de la rédaction de ce TP.
   **Page de téléchargement Mosquitto**:
   - *Lien direct (64-bit) : mosquitto-2.1.2-install-windows-x64.exe*
   - *Lien direct (32-bit) : mosquitto-2.1.2-install-windows-x86.exe*
2. **Installer le logiciel :** Exécutez le fichier `.exe` téléchargé et suivez l'assistant d'installation (laissez les options par défaut).
3. **Lancer Mosquitto :** Ouvrez une invite de commande (cmd) et tapez : *Note : Si la commande n'est pas reconnue, naviguez jusqu'au dossier d'installation (ex: `C:\Program Files\mosquitto`) avant de lancer la commande.*

```
mosquitto
```

### 🔹 Pour Linux (Ubuntu / Debian)

Pour obtenir la version la plus récente, il est recommandé d'utiliser le PPA (Personal Package Archive) du projet Mosquitto. Ouvrez un terminal et exécutez :

```
# 1. Ajouter le dépôt Mosquitto
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa

# 2. Mettre à jour la liste des paquets et installer
sudo apt-get update
sudo apt install mosquitto mosquitto-clients
```

Pour démarrer le service Mosquitto en arrière-plan et s'assurer qu'il se lance au démarrage :

```
sudo systemctl start mosquitto
sudo systemctl enable mosquitto
```

### 🔹 Pour macOS

La méthode la plus simple pour installer Mosquitto sur macOS est d'utiliser le gestionnaire de paquets Homebrew.

1. **Installer Homebrew (si nécessaire) :** Suivez les instructions sur brew.sh.
2. **Installer Mosquitto :** Ouvrez votre terminal et exécutez la commande :

```
brew install mosquitto
```

3. **Démarrer le broker :**

```
brew services start mosquitto
```

## 🧪 TP 3 SUITE.2 : Test du broker Mosquitto en local

Avant de connecter l'ESP32, nous allons vérifier que le broker fonctionne correctement en utilisant les outils clients en ligne de commande (installés avec Mosquitto).

### 📥 Terminal 1 (Le Subscriber / Abonné)

Ouvrez un premier terminal. Ce terminal va écouter tous les messages arrivant sur le topic `test/topic`.

```
mosquitto_sub -h localhost -t test/topic
```

### 📤 Terminal 2 (Le Publisher / Publieur)

Ouvrez un second terminal. Ce terminal va envoyer un message sur le même topic.

```
mosquitto_pub -h localhost -t test/topic -m "Hello IoT"
```

> 👉 **Résultat attendu :** Dans le**Terminal 1**, vous devriez voir apparaître instantanément le message :
>
> ```
> Hello IoT
> ```

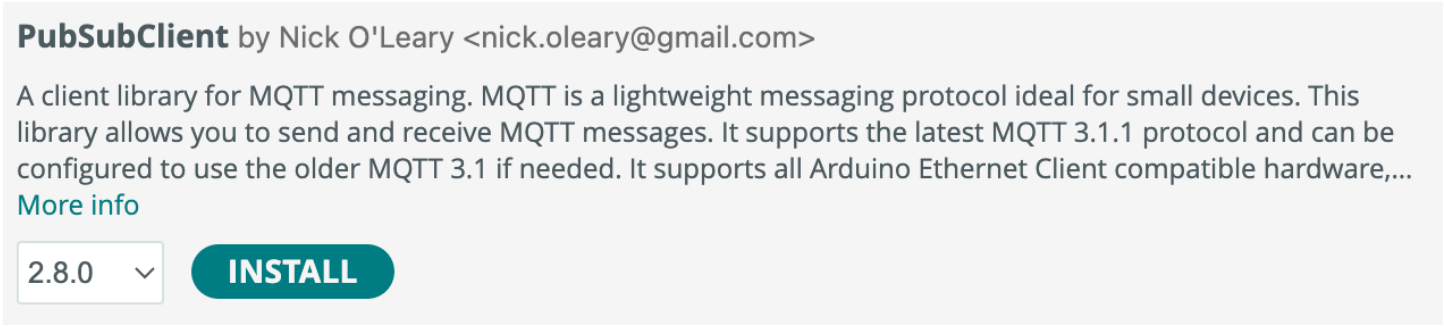
## 🧪 TP 3 SUITE.3 : Communication ESP32 → Mosquitto

> ⚠️ **Important : Changement d'adresse IP**
> Dans votre code précédent, vous pointiez vers un serveur public :
>
> ```
> client.setServer("test.mosquitto.org", 1883);
> ```
>
> Vous devez maintenant pointer vers**l'adresse IP locale de votre ordinateur** (ex: 192.168.1.X) :
>
> ```
> client.setServer("192.168.X.X", 1883);
> ```
>
> *Astuce : Sous Windows, utilisez la commande `ipconfig` dans le cmd pour trouver votre adresse IPv4. Sous Linux/Mac, utilisez `ip a` ou `ifconfig`.*

### 💻 Code ESP32 (Publisher MQTT vers Mosquitto Local)

Téléversez le code suivant dans votre ESP32 via l'IDE Arduino. N'oubliez pas de modifier les identifiants Wi-Fi et l'adresse IP du broker.

Installe : **PubSubClient by Nick O'Leary:**




```
#include <WiFi.h>
#include <PubSubClient.h>

// Configuration Wi-Fi
const char* ssid = "YOUR_WIFI";
const char* password = "YOUR_PASSWORD";

// Configuration MQTT
```

```
const char* mqtt_server = "192.168.1.100"; // Remplacer par l'IP de votre PC

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);

  // Connexion au Wi-Fi
  WiFi.begin(ssid, password);
  Serial.print("Connexion au WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connecté.");

  // Configuration du serveur MQTT
  client.setServer(mqtt_server, 1883);
}

void reconnect() {
  // Boucle jusqu'à ce qu'on soit connecté
  while (!client.connected()) {
    Serial.print("Tentative de connexion MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("connecté");
    } else {
      Serial.print("échec, rc=");
      Serial.print(client.state());
      Serial.println(" nouvelle tentative dans 5 secondes");
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
```

```
    reconnect();
  }
  client.loop();

  // Publication d'une température simulée toutes les 3 secondes
  client.publish("esp32/temp", "25");
  Serial.println("Message publié : 25 sur le topic esp32/temp");
  delay(3000);
}
```

## 🧪 TP 3 SUITE.4 : Visualisation avec MQTT Explorer

MQTT Explorer est un outil graphique indispensable pour déboguer et visualiser l'arborescence de vos topics MQTT.

1. **Installer MQTT Explorer :** Téléchargez-le depuis mqtt-explorer.com.
2. **Configurer la connexion :**

- **Name :** Mon Broker Local
- **Host :** L'adresse IP de votre PC (ex: `192.168.1.100`)
- **Port :**`1883`

3. **S'abonner au topic :** Cliquez sur "Connect". Dans l'interface, observez l'arborescence ou abonnez-vous manuellement au topic :

```
esp32/temp
```

> 👉 **Résultat :** Vous verrez les messages "25" envoyés par votre ESP32 s'afficher en temps réel avec un graphique de l'historique.

## 🧪 TP 3 SUITE.5 : Subscriber ESP32 (Avancé)

### 🎯 Objectif

Rendre l'ESP32 capable de **recevoir des commandes** depuis Mosquitto (par exemple, pour allumer une LED).

Ajoutez la fonction `callback` suivante à votre code, et modifiez le `setup()` pour vous abonner au topic de commande.

```
// Fonction déclenchée à la réception d'un message
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message reçu sur le topic [");
  Serial.print(topic);
  Serial.print("] : ");

  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Exemple d'action : si le message est "ON", allumer une LED...
}

void setup() {
  // ... (Code Wi-Fi précédent) ...

  client.setServer(mqtt_server, 1883);

  // Définition de la fonction de callback
  client.setCallback(callback);
}

void reconnect() {
  while (!client.connected()) {
    if (client.connect("ESP32Client")) {
      // Une fois connecté, on s'abonne au topic de commande
      client.subscribe("esp32/cmd");
    } else {
      delay(5000);
    }
  }
}
```

*Testez cette fonctionnalité en publiant un message sur `esp32/cmd` depuis MQTT Explorer et observez le moniteur série de l'IDE Arduino.*

## TP 3 SUITE.6 : Analyse comparative

### Travail demandé

Analysez le tableau ci-dessous et comprenez les compromis architecturaux entre un broker cloud et un broker local (Edge computing).

| Critère | Broker Public (Cloud) | Mosquitto Local (Edge) |
|---|---|---|
| **Latence** | Moyenne (dépend du routage Internet) | **Faible** (réseau local direct) |
| **Sécurité** | Faible (topics souvent publics) | **Configurable** (TLS, mots de passe) |
| **Dépendance Internet** | Oui (coupure = arrêt du système) | **Non** (fonctionne en réseau local) |
| **Performance** | Variable (partagé avec d'autres utilisateurs) | **Stable** (dédié à votre application) |

## Questions de validation

1. Quelle est la différence fondamentale entre un broker public et un broker local en termes de flux de données ?
2. Pourquoi une entreprise industrielle préférera-t-elle utiliser une instance Mosquitto privée plutôt qu'un broker public ?
3. Quels sont les avantages spécifiques du protocole MQTT mis en évidence dans ce TP (par rapport à du HTTP classique) ?
4. Que se passe-t-il au niveau de l'ESP32 si le service Mosquitto est arrêté sur le PC ? Comment le code gère-t-il cette situation ?

## 🎓 Conclusion

Ce TP vous a permis de :

- Passer d'une **simulation cloud à un système réel et autonome**.
- Comprendre le rôle **central et critique du broker MQTT** dans une architecture IoT.

Maîtriser les bases d'une architecture IoT professionnelle (Edge Computing), préparant le terrain pour l'intégration d'outils de traitement de données comme Node-RED ou des bases de données temporelles (InfluxDB).

# Annexe — Résolution du problème Mosquitto sous macOS (Homebrew)

## 1. Contexte du problème

Mosquitto, dans le monde réel (prod), est censé être installé sous Linux.

Lors de l'installation de Mosquitto via Homebrew sous MacOS, certains étudiants peuvent rencontrer une erreur lors du démarrage du service. La cause principale est l'absence du fichier de configuration attendu par Homebrew : /usr/local/etc/mosquitto/mosquitto.conf.

## 2. Cause du problème

Homebrew utilise un fichier de configuration spécifique pour démarrer Mosquitto en tant que service. Si ce fichier n'existe pas, le service échoue au démarrage. Toutefois, Mosquitto peut fonctionner sans fichier de configuration pour des tests simples.

## 3. Solution proposée

### Étape 1 : Créer le dossier de configuration

mkdir -p /usr/local/etc/mosquitto

### Étape 2 : Créer un fichier de configuration minimal

cat > /usr/local/etc/mosquitto/mosquitto.conf <<'EOF'
listener 1883
allow_anonymous true
EOF

### Étape 3 : Tester Mosquitto manuellement

/usr/local/opt/mosquitto/sbin/mosquitto -c /usr/local/etc/mosquitto/mosquitto.conf -v

Si Mosquitto démarre correctement, cela signifie que la configuration est valide.

### Étape 4 : Tester la communication MQTT

Terminal 1 :

mosquitto_sub -h localhost -p 1883 -t test/topic -d -o /dev/null

Terminal 2 :

mosquitto_pub -h localhost -p 1883 -t test/topic -m "Hello MQTT" -d

L’option -o /dev/null permet d’ignorer un éventuel fichier de configuration utilisateur pouvant provoquer des erreurs.

### Étape 5 : Relancer le service Homebrew

brew services stop mosquitto
rm -f ~/Library/LaunchAgents/homebrew.mxcl.mosquitto.plist
brew services start mosquitto
brew services list

## 4. Conclusion

Le problème provient principalement de l’absence du fichier de configuration attendu par Homebrew. La création de ce fichier et le test manuel du broker permettent de résoudre efficacement l’erreur. Cette procédure est recommandée avant toute utilisation du service en arrière-plan.