ESP8266 – MQTT

08/12/2021 IoT Comments: 0

ESP8266 – MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức gởi dạng publish/subscribe sử dụng cho các thiết bị Internet of Things với băng thông thấp, độ tin cậy cao và khả năng được sử dụng trong mạng lưới không ổn định.
Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng M2M.
MQTT cũng là giao thức sử dụng trong Facebook Messager.
Và MQTT là gì? Để có một cái nhìn toàn diện hoặc định nghĩa chi tiết, chỉ cần google “what is mqtt”, “mqtt slides” … Ở đây chúng ta chỉ nói ngắn gọn thôi, đủ để hiểu giao thức MQTT, bao gồm các định nghĩa **“subscribe”, “publish”, “qos”, “retain”, “last will and testament (lwt)”**

Publish, subscribe

Trong một hệ thống sử dụng giao thức MQTT, nhiều node trạm (gọi là mqtt client – gọi tắt là client) kết nối tới một MQTT Server (gọi là Broker). Mỗi client sẽ đăng ký một vài kênh (topic), ví dụ như “/client1/channel1”, “/client1/channel2”. Quá trình đăng ký này gọi là **“subscribe”**, giống như chúng ta đăng ký nhận tin trên một kênh Youtube vậy. Mỗi Client sẽ nhận được dữ liệu khi bất kỳ trạm nào khác gởi dữ liệu vào kênh đã đăng ký. Khi một Client gởi dữ liệu tới kênh đó, gọi là **“publish”**.

QoS

Ở đây có 3 tuỳ chọn **QoS (Qualities of service)** khi “publish” và “subscribe”:

- **QoS0** Broker/Client sẽ gởi dữ liệu đúng 1 lần, quá trình gởi được xác nhận bởi chỉ giao thức TCP/IP, giống kiểu đem con bỏ chợ.

- **QoS1** Broker/Client sẽ gởi dữ liệu với ít nhất 1 lần xác nhận từ đầu kia, nghĩa là có thể có nhiều hơn 1 lần xác nhận đã nhận được dữ liệu.
- **QoS2** Broker/Client đảm bảm khi gởi dữ liệu thì phía nhận chỉ nhận được đúng 1 lần, quá trình này phải trải qua 4 bước bắt tay.

Xem thêm QoS: code.google.com/p/mqtt4erl/wiki/QualityOfServiceUseCases
Một gói tin có thể được gởi ở bất kỳ QoS nào, và các Client cũng có thể subscribe với bất kỳ yêu cầu QoS nào. Có nghĩa là Client sẽ lựa chọn QoS tối đa mà nó có để nhận tin. Ví dụ, nếu 1 gói dữ liệu được publish với QoS2, và Client subscribe với QoS0, thì gói dữ liệu được nhận về Client này sẽ được broker gởi với QoS0, và 1 Client khác đăng ký cùng kênh này với QoS 2, thì nó sẽ được Broker gởi dữ liệu với QoS2.
Một ví dụ khác, nếu 1 Client subscribe với QoS2 và gói dữ liệu gởi vào kênh đó publish với QoS0 thì Client đó sẽ được Broker gởi dữ liệu với QoS0. QoS càng cao thì càng đáng tin cậy, đồng thời độ trễ và băng thông đòi hỏi cũng cao hơn.

Retain

Nếu RETAIN được set bằng 1, khi gói tin được publish từ Client, Broker **PHẢI** lưu trữ lại gói tin với QoS, và nó sẽ được gởi đến bất kỳ Client nào subscribe cùng kênh trong tương lai. Khi một Client kết nối tới Broker và subscribe, nó sẽ nhận được gói tin cuối cùng có RETAIN = 1 với bất kỳ topic nào mà nó đăng ký trùng. Tuy nhiên, nếu Broker nhận được gói tin mà có QoS = 0 và RETAIN = 1, nó sẽ huỷ tất cả các gói tin có RETAIN = 1 trước đó. Và phải lưu gói tin này lại, nhưng hoàn toàn có thể huỷ bất kỳ lúc nào.
Khi publish một gói dữ liệu đến Client, Broker phải set RETAIN = 1 nếu gói được gởi như là kết quả của việc subscribe mới của Client (giống như tin nhắn ACK báo subscribe thành công). RETAIN phải bằng 0 nếu không quan tâm tới kết quả của việc subscribe.

LWT

Gói tin LWT (last will and testament) không thực sự biết được Client có trực tuyến hay không, cái này do gói tin KeepAlive đảm nhận. Tuy nhiên gói tin LWT như là thông tin điều gì sẽ xảy đến sau khi thiết bị ngoại tuyến.

Một ví dụ

Tôi có 1 cảm biến, nó gởi những dữ liệu quan trọng và rất không thường xuyên. Nó có đăng ký trước với Broker một tin nhắn lwt ở topic **/node/gone-offline** với tin nhắn **id** của nó. Và tôi cũng đăng ký theo dõi topic **/node/gone-offline**, sẽ gởi SMS tới điện thoại thôi mỗi khi nhận được tin nhắn nào ở kênh mà tôi theo dõi.
Trong quá trình hoạt động, cảm biến luôn giữ kết nối với Broker bởi việc luôn gởi gói tin keepAlive. Nhưng nếu vì lý do gì đó, cảm biến này chuyển sang ngoại tuyến, kết nối tới Broker timeout do Broker không còn nhận được gói keepAlive.
Lúc này, do cảm biến của tôi đã đăng ký LWT, do vậy broker sẽ đóng kết nối của Cảm biến, đồng thời sẽ publish một gói tin là Id của cảm biến vào kênh **/node/gone-offline**, dĩ nhiên là tôi cũng sẽ nhận được tin nhắn báo cái cảm biến yêu quý của mình đã ngoại tuyến.

Ngắn gọn

Ngoài việc đóng kết nối của Client đã ngoại tuyến, gói tin LWT có thể được định nghĩa trước và được gởi bởi Broker tới kênh nào đó khi thiết bị đăng ký LWT ngoại tuyến.

MQTT Client

Như chúng ta đã tìm hiểu ở phần trước, 2 thành phần publisher và subscriber là đặc trưng tạo nên giao thức MQTT. Các MQTT Client không kết nối trực tiếp với nhau, mọi gói dữ liệu được gửi đi đều thông qua MQTT Broker. Để có thể triển khai các ứng dụng của MQTT Client, chúng ta cần MQTT Broker (sẽ được trình bày trong phần sau). Ở phần này chúng ta sẽ làm quen với giao thức MQTT bằng các ví dụ sử dụng MQTT Client thông dụng và các dịch vụ MQTT Broker miễn phí và phổ biến, 2 trong số chúng là test.mostquitto.org và cloudmqtt.com

MQTT Lens

Thông tin

MQTT Lens là một tiện ích mở rộng của Chrome (Chrome Extension), nó sử dụng trình duyệt Chrome để kết nối đến MQTT Broker cũng như test các tính năng publish/subcribe của giao thức MQTT. Đây là một công cụ rất hữu ích để kiểm tra kết nối đến MQTT Broker và kiểm tra việc gửi và nhận gói tin.

Một số thông tin của MQTT Lens được trình bày ở bảng dưới.

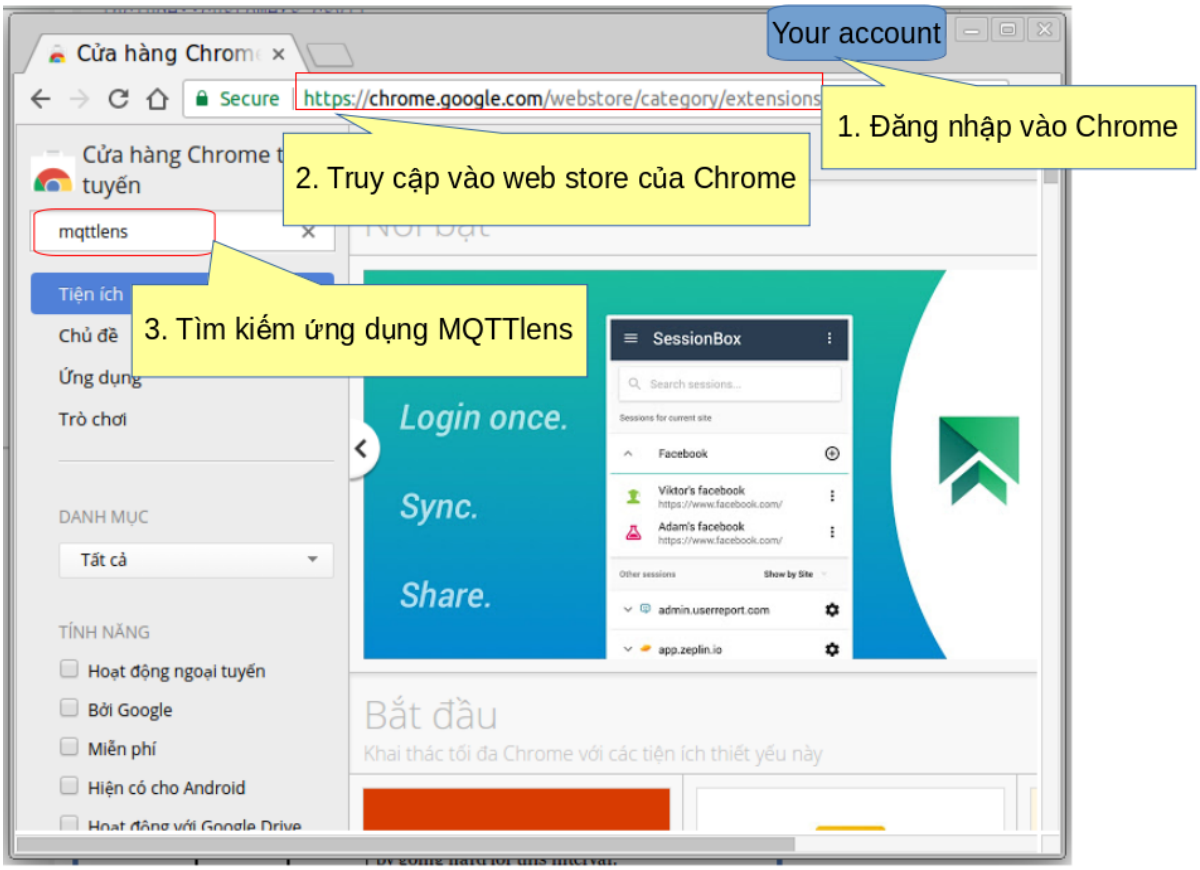
Bảng 1. Short Profile

Type	Chrome App
License	MIT
Operating Systems	Windows, Linux & MacOSX
Website	“https

Kết nối

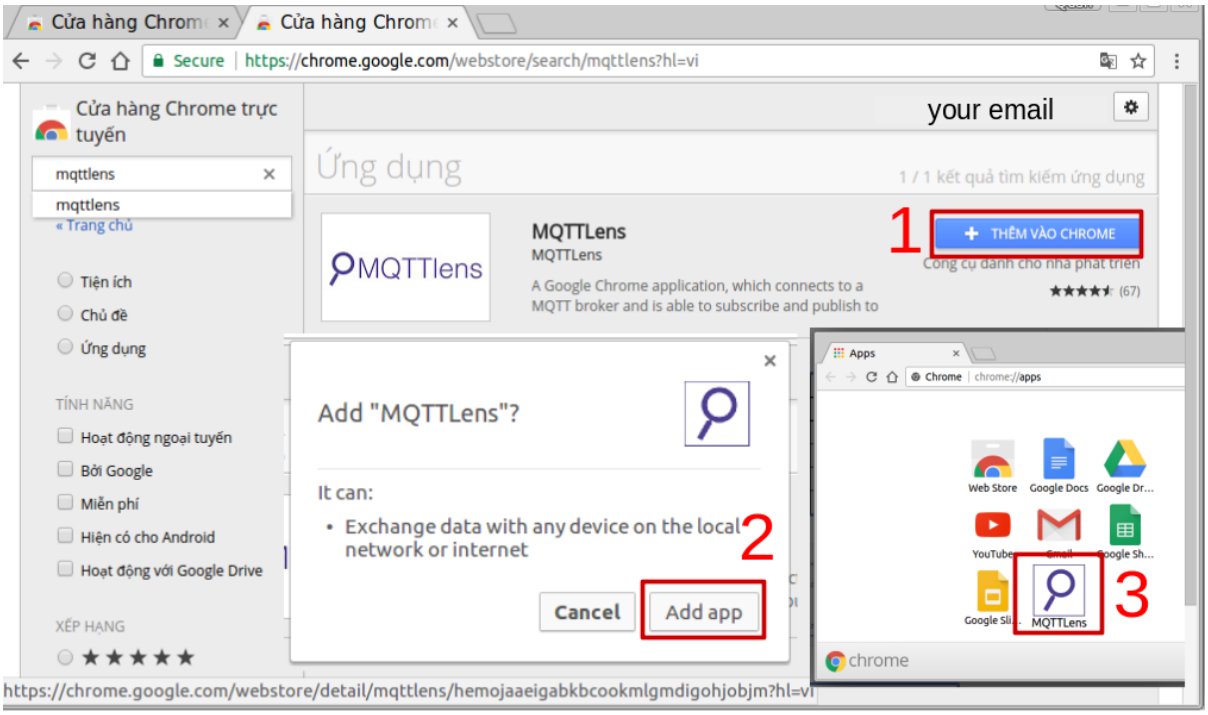
Chúng ta sẽ sử dụng công cụ này với dịch vụ MQTT Broker tại iot.eclipse.org được trình bày như các bước bên dưới:

» Bước 1: Cài đặt trình duyệt Chrome, thực hiện đăng nhập tài khoản của bạn vào chrome, truy cập vào địa chỉ chrome.google.com/webstore/category/extensions và gõ mqttlens vào mục tìm kiếm tiện ích như hình bên dưới.



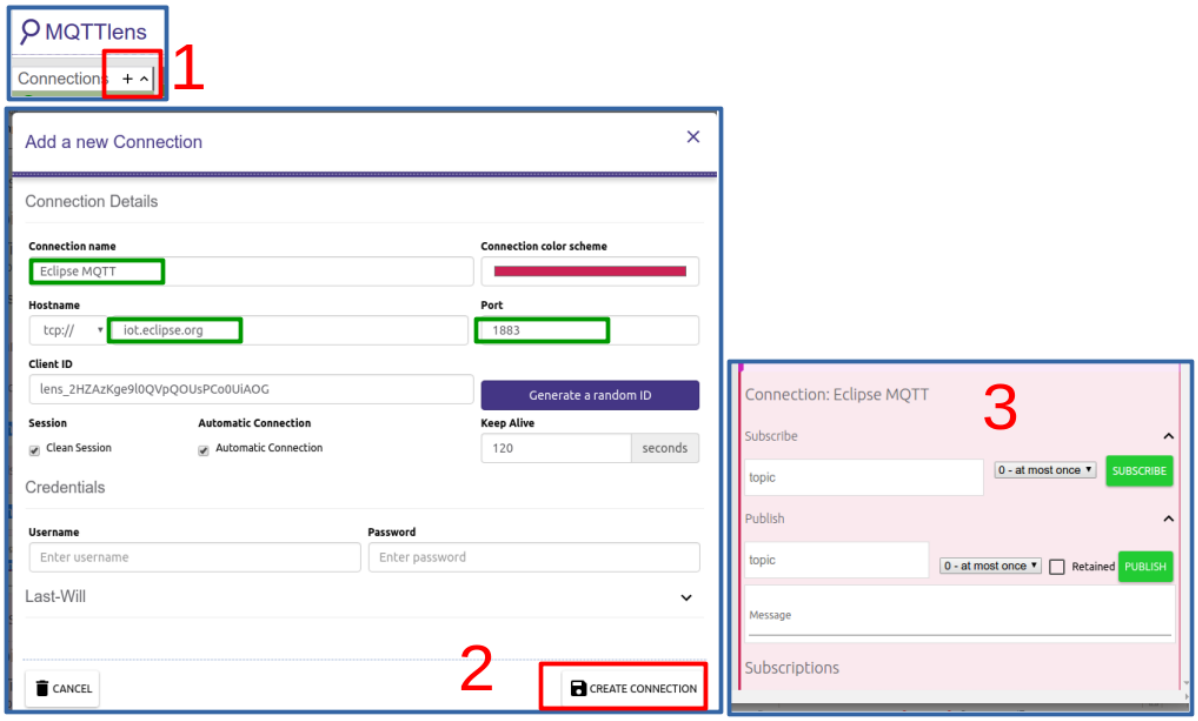
Hình 1. Hình ảnh tìm kiếm tiện ích mqttlens trên chrome store

» Bước 2: Thêm và khởi chạy MQTT lens



Hình 2. Hình ảnh các bước thêm và khởi chạy tiện ích MQTT lens

» Bước 3 : Tạo 1 MQTT Client kết nối đến MQTT Broker eclipse như các bước bên dưới.



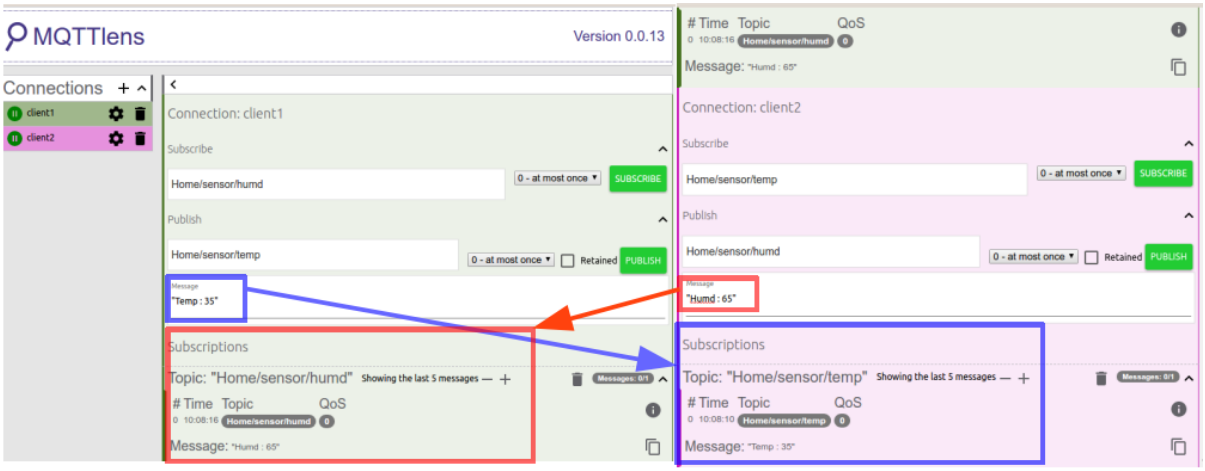
Hình 3. Hình ảnh tạo 1 MQTT client

Giải thích

Chúng ta sẽ tạo 1 connection có tên eclipse MQTT với host name của MQTT Broker là `iot.eclipse.org`, Broker này sẽ giúp trao đổi dữ liệu của các Client với nhau và lắng nghe các Client ở port 1883 (port sử dụng giao thức MQTT và không mã hóa dữ liệu, các port khác tham khảo tại test.mosquitto.org)

Ở connection này sẽ đăng kí nhận gói tin tại topic `Home/garden/sensor/#` (kí tự # cho phép subcribe các topic `Home/garden/sensor/1`, `Home/garden/sensor/2` vv...). Tiếp theo chúng ta sẽ pulish 1 gói dữ liệu với nội dung `"Temp in garden: 27degree Celcius "` tại topic `Home/garden/sensor/1`.

Kết quả: Tại mục subrcriptions, chúng ta sẽ nhận được gói dữ liệu đã publish do đã subcribe topic `Home/garden/sensor/#` như hình bên dưới.



Hình 4. Hình ảnh dữ liệu nhận được sau khi publish gói tin

Mở rộng

Tạo nhiều connection để subcribe và publish các gói tin với MQTT Broker iot.eclipse.org đồng thời test các gói tin với QoS và LWT

MQTT.js

MQTT.js là một thư viện MQTT client, được viết bằng ngôn ngữ JavaScript trên nền tảng Node.js và hỗ trợ MQTT Over Websocket (MOW).
MQTT.js là dự án mã nguồn mở (open source), bạn có thể tải MQTT.js bản cập nhật mới nhất tại github.com/mqttjs/MQTT.js.git

Cài đặt

Trước tiên ta cần kiểm tra hệ điều hành đã hỗ trợ Node.js trước khi cài đặt MQTT.js. Nếu chưa thì có thể tham khảo cách cài đặt tại nodejs.org/en/
Khởi tạo một dự án Node.js. Để dễ quản lý, có thể tạo một thư mục riêng, ví dụ `mqtt-client` và một file javascrip trong đó, ví dụ như `client-a.js`. Đi đến thư mục này và mở terminal (linux OS) hoặc Command Prompt (trên Windowns OS) và dùng lệnh:

```
npm init
```

Khi chạy lệnh này, bạn cũng cần phải khai báo thêm một số thông tin cho dự án như tên, phiên bản, keywords, tác giả,... Sau khi tạo xong, trong thư mục vừa tạo sẽ xuất hiện một file là `package.json` với nội dung là các phần đã khai báo. File này cũng chứa thuộc tính dùng để lưu trữ các package chúng ta đã cài đặt.
Tiếp theo chúng ta sẽ cài `MQTT.js`, sử dụng lệnh:

```
npm install mqtt --save
```

Sau khi cài đặt xong, bạn có thể sử dụng module `mqtt` để thực hiện việc kết nối MQTT Client với Broker, publish message hay subscribe topic. Tất nhiên, toàn bộ các file liên quan đến thư viện sẽ nằm trong thư mục `node_modules`, trong thư mục dự án.
Để hiểu rõ hơn cách hoạt động của MQTT.js, chúng ta sẽ tạo ra thêm 1 số file mã nguồn Javascript (file .js) là `client-a.js` và `client-b.js` thực hiện subcribe và puslish các gói tin.

Nội dung thực hiện

2 Client này sẽ kết nối vào cùng 1 MQTT Broker. Client A sẽ subscribe kênh `/client-a/sub`, nếu nhận bất kỳ dữ liệu nào được publish vào kênh này, client A sẽ public dữ liệu `Hello from client A` vào kênh `/client-b/sub` và đóng kết nối, kết thúc. Client B sẽ subscribe kênh `/client-b/sub`, nếu nhận bất kỳ dữ liệu nào được public vào kênh này, client B sẽ đóng kết nối và kết thúc. Ngay khi kết nối vào Broker, client B sẽ public 1 gói tin `Hello from client B` vào kênh `/client-a/sub`

Mã nguồn của client A

client-a.js

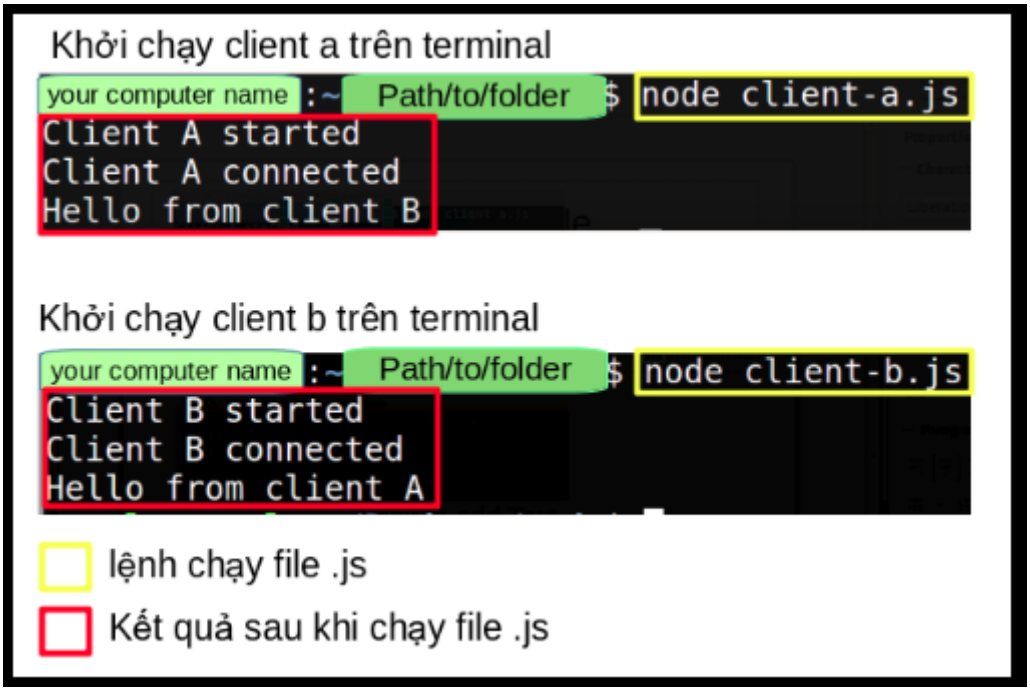
```
// tạo biến mqtt sử dụng các chức năng của module mqtt
var mqtt = require('mqtt')
// tạo biến client sử dụng thuộc tính connect để kết nối đến broket MQTT với
var client = mqtt.connect('mqtt://iot.eclipse.org')
// function có chức năng subscribe 1 topic nếu đã kết nối thành công đến bro
client.on('connect', function() {
    console.log('Client A connected')
    // client subcribe topic /client-a/sub
    client.subscribe('/client-a/sub')
})
```

```
// function có chức năng gửi 1 gói tin đến đến topic đã đăng kí
client.on('message', function(topic, message) {
    // in ra màn hình console 1 message ở định dạng string
    console.log(message.toString())
    // publish gói tin 'Hello from client A' đến topic /client-b/sub
    client.publish('/client-b/sub', 'Hello from client A')
    // đóng kết nối của client
    client.end()
})
console.log('Client A started')
```

client-b.js

```
// tạo biến mqtt sử dụng các chức năng của module mqtt
var mqtt = require('mqtt')
// tạo biến client sử dụng thuộc tính connect để kết nối đến broket MQTT với
var client = mqtt.connect('mqtt://iot.eclipse.org')
// function có chức năng subscribe 1 topic nếu đã kết nối thành công đến bro
client.on('connect', function() {
    console.log('Client B connected')
    // client subcribe topic /client-b/sub
    client.subscribe('/client-b/sub')
    // publish gói tin 'Hello from client B' đến topic /client-a/sub
    client.publish('/client-a/sub', 'Hello from client B')
})

client.on('message', function(topic, message) {
    // in ra màn hình console 1 message ở định dạng string
    console.log(message.toString())
    // đóng kết nối của client
    client.end()
})
console.log('Client B started')
```

Kết quả hiển thị như hình bên dưới:



Hình 5. Hình ảnh kết quả khi khởi chạy các MQTT client

Ngoài ra, MQTT.js còn cung cấp thêm các lệnh để có thể tương tác với Broker thông qua terminal. Để làm được điều này, chúng ta cài đặt MQTT.js như một module toàn cục bằng cách sử dụng lệnh:

```
npm install mqtt -g.
```

Bạn có thể kiểm tra bằng cách mở 2 màn hình terminal, ở màn hình 1 (tạm gọi là subscriber) sẽ subscribe vào topic tên là “topicA” bằng lệnh:

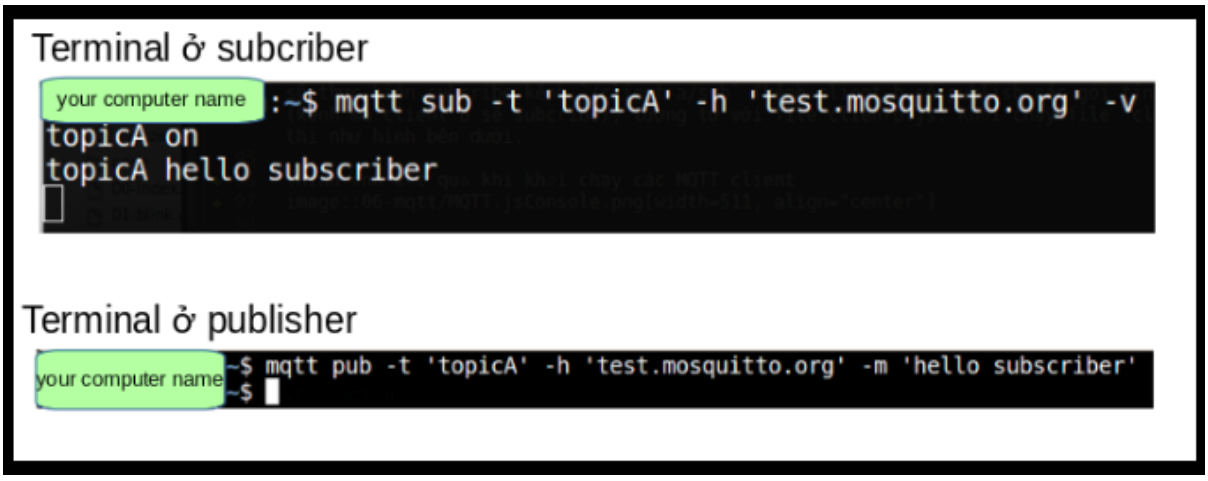
```
mqtt sub -t 'topicA' -h 'test.mosquitto.org' -v
```

Ở terminal thứ 2 (tạm gọi là publisher) thực hiện publish một tin nhắn với nội dung “hello subscriber” tới “topicA”:

```
mqtt pub -t 'topicA' -h 'test.mosquitto.org' -m 'hello subscriber'
```

Các options:

- `-t` : MQTT topic, nơi sẽ thực hiện pushlish 1 message.
- `-h` : Xác định máy chủ sẽ kết nối đến.
- `-m` : Gửi 1 message dùng command line.
- `-v` : verbose, option cho phép ghi lại nhật kí hoạt động của các tập tin trong file cấu hình.



Hình 6. Hình ảnh message được publish dùng command line

◌	Để xem thêm các API khác trong MQTT.js, bạn có thể sử dụng lệnh: `mqtt help [command]`.

ESP8266 MQTT Client

Thực tế có khá nhiều thư viện MQTT cho ESP8266 trên Arudino, ở đây chúng ta chỉ đề cập đến 2 thư viện phổ biến là `PubSubClient` và `ESP8266MQTTClient`

PubSubClient

Trong phần này chúng ta sẽ thực hiện kết nối board NodeMCU đến 1 broker sử dụng thư viện PubSubClient.

- **Bước 1 :** Download thư viện PubSubClient tại đường dẫn github.com/knolleary/pubsubclient và add vào chương trình Arduino. Ngoài ra có thể import thư viện này trong Arduino bằng cách tìm kiếm thư viện với từ khóa `PubSubClient`, chọn thư viện PubSubClient của tác giả Nick O'Leary và nhấn install.
- **Bước 2 :** Viết và nạp chương trình cho ESP8266. Mã nguồn được trình bày ở phía dưới

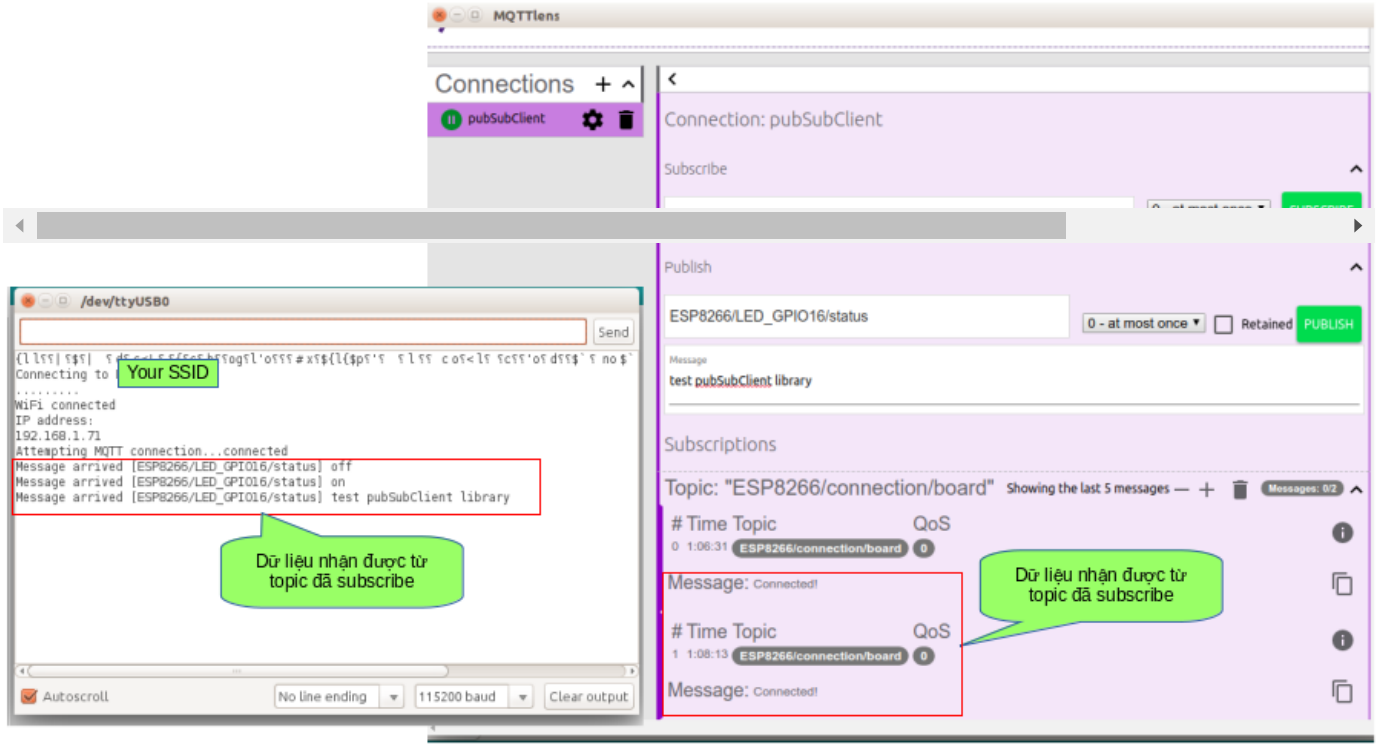
```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = ".......";
const char* password = ".........";
const char* mqtt_server = "broker.mqtt-dashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  pinMode(16, OUTPUT);
  Serial.begin(115200);
  // hàm thực hiện chức năng kết nối Wifi và in ra địa chỉ IP của ESP8266
  setup_wifi();
  // cài đặt server là broker.mqtt-dashboard.com và lắng nghe client ở port
  client.setServer(mqtt_server, 1883);
  // gọi hàm callback để thực hiện các chức năng publish/subcribe
  client.setCallback(callback);
  // gọi hàm reconnect() để thực hiện kết nối lại với server khi bị mất kết
  reconnect();
}

void setup_wifi() {
  delay(10);
```

```
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  // kết nối đến mạng Wifi
  WiFi.begin(ssid, password);
  // in ra dấu . nếu chưa kết nối được đến mạng Wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // in ra thông báo đã kết nối và địa chỉ IP của ESP8266
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  //in ra tên của topic và nội dung nhận được từ kênh MQTT lens đã publish
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  // kiểm tra nếu dữ liệu nhận được từ topic ESP8266/LED_GPIO16/status là ch
  // sẽ bậtled GPIO16, nếu là chuỗi "off" sẽ tắt led GPIO16
  if ((char)payload[0] == 'o' && (char)payload[1] == 'n') //on
    digitalWrite(16, LOW);
  else if ((char)payload[0] == 'o' && (char)payload[1] == 'f' && (char)paylo
    digitalWrite(16, HIGH);
  Serial.println();
}

void reconnect() {
  // lặp cho đến khi được kết nối trở lại
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266")) {
      Serial.println("connected");
      // publish gói tin "Connected!" đến topic ESP8266/connection/board
      client.publish("ESP8266/connection/board", "Connected!");
      // đăng kí nhận gói tin tại topic ESP8266/LED_GPIO16/status
```

```
      client.subscribe("ESP8266/LED_GPIO16/status");

    } else {
      // in ra màn hình trạng thái của client khi không kết nối được với MQT
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // delay 5s trước khi thử lại
      delay(5000);
    }
  }
}

void loop() {
  // kiểm tra nếu ESP8266 chưa kết nối được thì sẽ thực hiện kết nối lại
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Chúng ta sẽ tạo một biến espClient thuộc lớp WiFiClient, biến này được khai báo là MQTT Client và sử dụng các thuộc tính của thư viện `PubSubClient`. Tại hàm `setup()` sẽ thiết lập ESP8266 ở chế độ station, kết nối đến mạng wifi. Bên cạnh đó hàm `setup()` cũng sẽ thực hiện chức năng tự động kết nối lại với MQTT Broker khi xảy ra mất kết nối đồng thời thực hiện các chức năng publish, subscribe của 1 MQTT Client thông qua hàm `reconnect()`. Hàm `callback()` có nhiệm vụ lấy dữ liệu của các puslisher khi publish 1 message sau đó gửi đến các client đã subscribe topic đó và kiểm tra nội dung của message để điều khiển led ở GPIO16. Hàm `loop()` có chức năng kết nối Client là ESP8266 với Broker, thực hiện chức năng publish 1 message và subscribe topic. `client.loop()` sẽ kiểm tra thời gian kết nối của Client với gói `KEEP_ALIVE` để đưa ra các thông tin về trạng thái kết nối của ESP8266 đồng thời lấy dữ liệu của message từ buffer để gửi đến các Client đã subcribe topic.

- **Bước 3 :** Mở MQTT lens trên trình duyệt Chrome, tạo 1 connection với host name `broker.mqtt-dashboard.com`, sử dụng port 1883. Thực hiện subscribe topic `ESP8266/connection/board`. Sau khi nhấn nút subscribe trên MQTT lens sẽ xuất hiện 1 message gửi từ esp8288 với nội dung `connnected`. Thực hiện pushlish các message vào topic `ESP8266/LED_GPIO16/status`. Nếu pushlish message với nội dung on, led GPIO16 trên board sẽ sáng, pushlish message `off` led GPIO16 trên board sẽ tắt. Các message với nội dung khác thì vẫn sẽ hiển thị dữ liệu nhận được trên serial terminal của Arduino nhưng sẽ không có tác dụng điều khiển led GPIO16. Kết quả hiển thị như hình bên dưới:



Hình 7. Kết quả hiển thị trên serial terminal và MQTT lens khi sử dụng thư viện pubsubClient

ESP8266MQTTClient

Tiếp theo, chúng ta sẽ tìm hiểu cách sử dụng thư viện ESP8266MQTTClient, thư viện được cộng đồng developer đánh giá là ổn định dễ sử dụng hơn so với thư viện PubSubClient thông qua 1 ứng dụng điều khiển led với board NodeMCU bằng 1 ứng dụng trên điện thoại smartphone.

- **Bước 1 :** Download thư viện `ESP8266MQTTClient` tại đường dẫn github.com/tuanpmt/ESP8266MQTTClient và add vào chương trình Arduino. Ngoài ra có thể import thư viện này trong Arduino bằng cách tìm kiếm thư viện với từ khóa `ESP8266MQTT`, chọn thư viện của tác giả Tuan PM, version 1.3 và nhấn install.
- **Bước 2 :** Viết và nạp chương trình cho ESP8266. Mã nguồn được trình bày ở phía dưới.

```
#include <ESP8266MQTTClient.h>
#include <ESP8266WiFi.h>

#define ledPin 16   //Led on board ESP8266 WiFi Uno

MQTTClient mqtt;

const char* ssid = "Your SSID";
const char* password = "Your password";

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
```

```
// OFF led GPIO16 khi bắt đầu chương trình
digitalWrite(ledPin, HIGH);

// Thiết lập ESP8266 ở chế độ STA và kết nối Wifi
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
Serial.print("\nConnecting to ");
Serial.println(ssid);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

// Kết nối đến server MQTT, in ra id của topic "esp8266/GPIO16" đồng thời
// và subcribe các topic "hello/esp8266"
mqtt.onConnect([]() {
  Serial.printf("MQTT: Connected\r\n");
  Serial.printf("Subscribe id: %d\r\n", mqtt.subscribe("esp8266/GPIO16", 0
  mqtt.subscribe("esp8266/GPIO16", 0);
});

// Thực hiện chức năng subscribe topic và publish các message
mqtt.onSubscribe([](int sub_id) {
  //in ra id của các topic đã subscribe là "hello/esp8266" và "MQTTlens/te
  Serial.printf("Subscribe topic id: %d ok\r\n", sub_id);

  //publish message có nội dung hello app đến topic Broker/app với gói QoS
  mqtt.publish("Broker/app", "hello app", 0, 0);
});

// Xử lí dữ liệu nhận được của các topic đã subscribe
mqtt.onData([](String topic, String data, bool cont) {
  Serial.printf("Data received, topic: %s, data: %s\r\n", topic.c_str(), d

  // Nếu chuỗi nhận được là 'on' sẽ ON led trên board ESP8266 WiFi Uno, ch
  if (topic == "esp8266/GPIO16" && data[0] == 'o' && data[1] == 'n' && dat
    digitalWrite(ledPin, LOW);
    Serial.println("Turn on the led on board");
  } else if (topic == "esp8266/GPIO16" && data[0] == 'o' && data[1] =
```

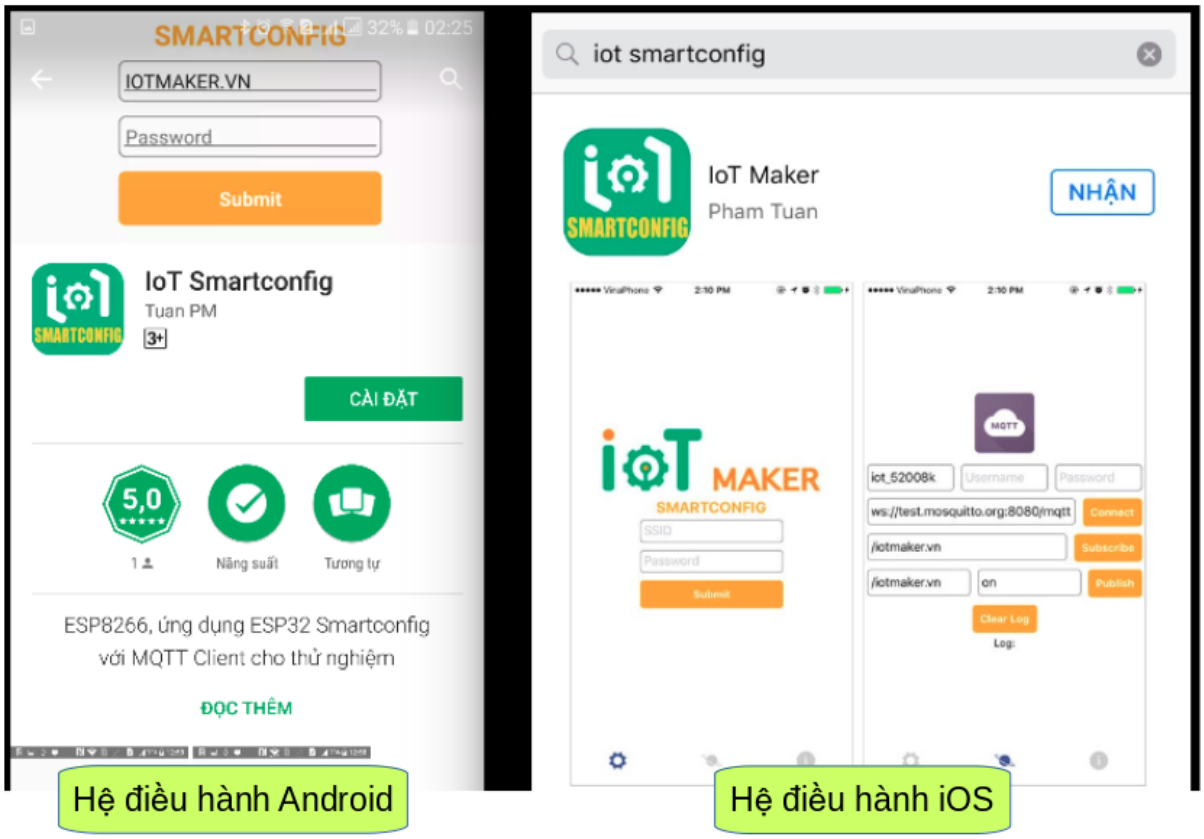
```
      digitalWrite(ledPin, HIGH);
      Serial.println("Turn off the led on board");
    }
  });
  // khởi tạo broker MQTT là iot.eclipse.org sử dụng phương thức websocket v
  mqtt.begin("ws://iot.eclipse.org:80/ws");
}

void loop() {
  // Hàm khởi tạo MQTT, kiểm tra và xử lí các dữ liệu từ các topic, kiểm tra
  // thức như gói keep-a-live, gói tin QoS, id của topic...,
  mqtt.handle();
}
```

Tương tự như mã nguồn của chương trình sử dụng thư viện pubsubClient, chúng ta cũng sẽ khởi tạo ESP8266 là MQTT Client trong class MQTT của thư viện ESP8266MQTTClient. Cài đặt ESP8266 ở chế độ Station và kết nối đến network wifi. Chức năng của các hàm trong thư viện đã được giải thích ở file mã nguồn, ở hàm `mqtt.onConnect()` chúng ta sẽ subscribe topic là `esp8266/GPIO16`. Hàm `mqtt.onSubscribe()` sẽ thực hiện puslish các message ở topic đã chỉ định là `Broker/app`. Hàm `mqtt.onData()` sẽ nhận, kiểm tra và xử lí dữ liệu nhận được từ topic đã subscribe. Ở đây ta sẽ dùng 1 public MQTT Broker là iot.eclipse.org, sử dụng phương thức Websocket là lắng nghe các MQTT Client ở port 80, đây là port mặc định khi sử dụng Websocket. Việc gửi nhận dữ liệu bằng phương thức Websocket sẽ giúp giảm băng thông và độ trễ khi truyền nhận dữ liệu thông qua giao thức MQTT. Chi tiết về Websocket chúng ta sẽ được học ở các bài học sau. Ở `loop()` chúng ta chỉ cần gọi hàm `handle()` để khởi tạo và kiểm tra các thuộc tính của giao thức cũng như xử lí, truyền và nhận dữ liệu từ các topic đã subscribe và public.

◌	Để tìm hiểu chi tiết file cấu hình của thư viện, có thể xem tại github.com/tuanpmt/ESP8266MQTTClient/tree/master/src

» **Bước 3 :** Cài đặt và sử dụng ứng dụng trên điện thoại để điều khiển led GPIO16.

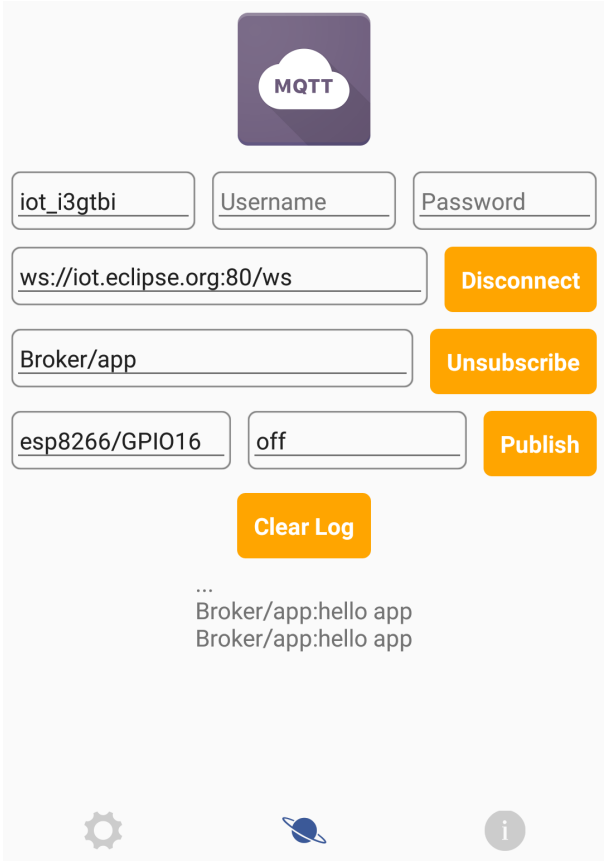
Truy cập vào `App Store` trên hệ điều hành iOS hoặc `CH Play` trên hệ điều hành Android. nhập từ khóa `IoT Smartconfig` và cài đặt ứng dụng `IoT Smartconfig` của developer Tuan PM. Hình ảnh ứng dụng hiển thị như bên dưới:



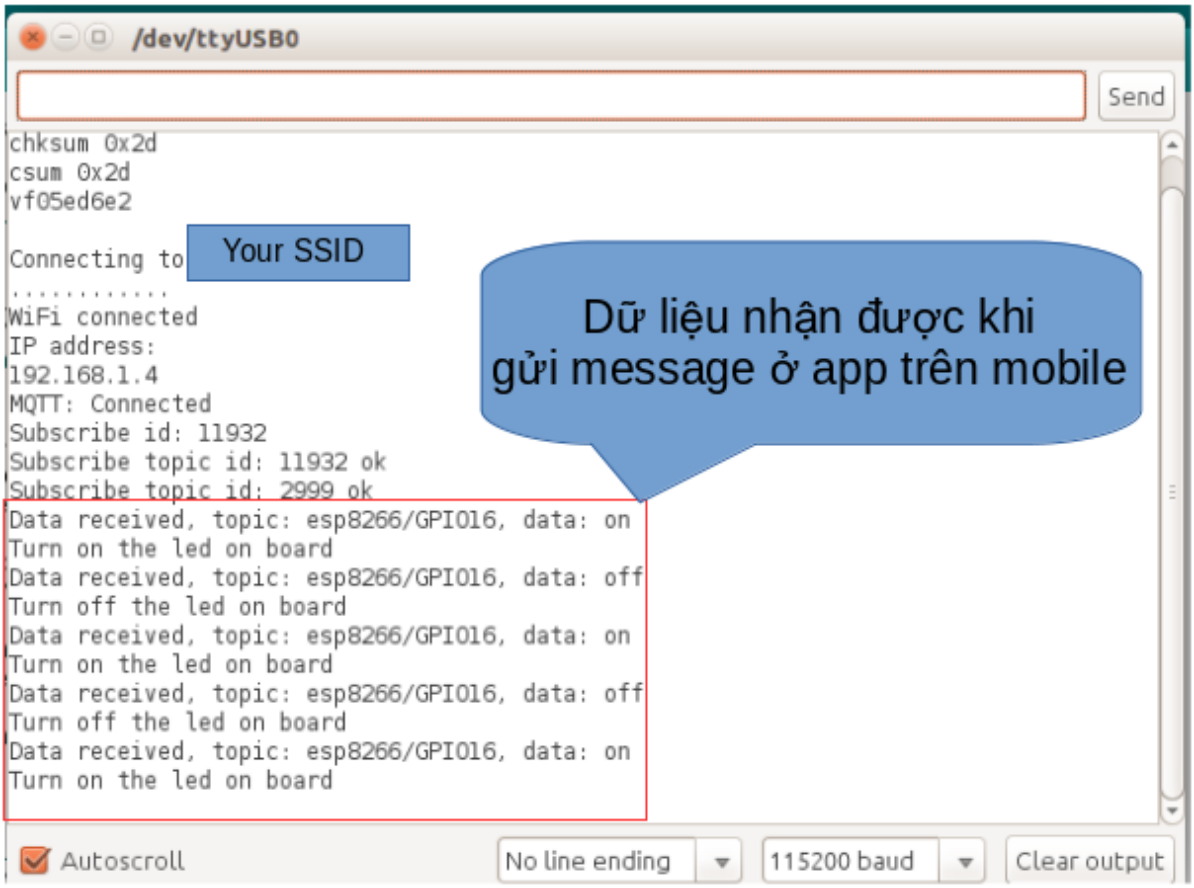
Hình 8. Hình ảnh ứng dụng IoT Smartconfig trên hệ điều hành iOS và Android

Ứng dụng này sử dụng với ESP8266 và ESP32, ngoài chức năng cơ bản là publish, subscribe của giao thức MQTT, ứng dụng còn có chức năng smartconfig để ESP8266 và ESP32 có thể dễ dàng thiết lập kết nối với các network wifi khác nhau một cách thuận tiện và nhanh chóng mà không phải nạp lại mã nguồn.

Tiếp theo, trượt ứng dụng qua phần MQTT, nhấn vào nút `connect` để kết nối đến server MQTT Broker `ws://iot.eclipse.org:80/ws`. Thực hiện subscribe topic `Broker/app` và publish message vào topic `esp8266/GPIO16`. Nếu publish message on vào `esp8266/GPIO16` thì board NodeMCU sẽ bật led, gửi `off` sẽ tắt led, đồng thời khi ESP8266 publish các message ở topic `Broker/app` thì nội dung các message sẽ được hiển thị trên ứng dụng. Kết quả hiển thị như hình bên dưới:



Hình 9. Hình ảnh subcribe topic và publish các message trên ứng dụng



Hình 10. Hình ảnh trên Serial terminal của Arduino

MQTT Broker

Ở phần trước chúng ta sử dụng các dịch vụ MQTT Broker miễn phí để thử nghiệm, tuy nhiên ta có thể sẽ phải trả phí dịch vụ với những ứng dụng lớn cần băng thông rộng và tốc độ đáp ứng nhanh, cộng với việc dữ liệu có thể bị tấn công do độ bảo mật thông tin chưa cao. Do đó, ở phần này, chúng ta sẽ tự mình xây dựng 1 MQTT Broker. Việc tự thiết lập 1 MQTT broker giúp chúng ta có thể sử dụng giao thức MQTT trên máy local mà không cần kết nối đến các dịch vụ MQTT Broker ở mạng internet. Quá trình truyền, nhận và xử lí dữ liệu diễn ra 1 cách nhanh chóng cũng như bảo mật thông tin của người dùng. Tuy nhiên, để tạo được 1 MQTT Broker với đầy đủ tính năng của giao thức MQTT đòi hỏi chúng ta phải có kiến thức tốt về giao thức MQTT cũng như các ngôn ngữ lập trình bổ trợ cho việc xây dựng nó. Để bắt đầu, ta sẽ tạo ra 1 MQTT Broker đơn giản bằng cách dùng 1 module hỗ trợ sẵn có đó là `Mosca`.

MOSCA

Mosca là 1 trong số rất nhiều server MQTT Broker của giao thức MQTT. Có thể kể đến các server khác như HiveMQ, Apache Apollo, Mosquitto, Mongoose. Mosca có 1 số đặc điểm như sau:

- Nó là 1 Node.js Broker, được viết bằng ngôn ngữ JavaScript vì vậy để có thể xây dựng MQTT Broker, chúng ta cần Node.js để chạy. Mosca có thể nhúng vào ứng dụng của bạn nếu ứng dụng này được viết bằng Node.js
- Mosca là 1 multi-transport MQTT Broker, có nghĩa là nó hỗ trợ tất cả các chức năng publish, subscribe của các broker khác. Danh sách các publish/subscribe broker được hỗ trợ bao gồm RabbitMQ, Redis, Mosquitto, ZeroMQ. Ở phần này chúng ta sẽ tạo ta 1 MQTT Broker đơn giản dùng Mosca với sự hỗ trợ của cơ sở dữ liệu `Mongodb`

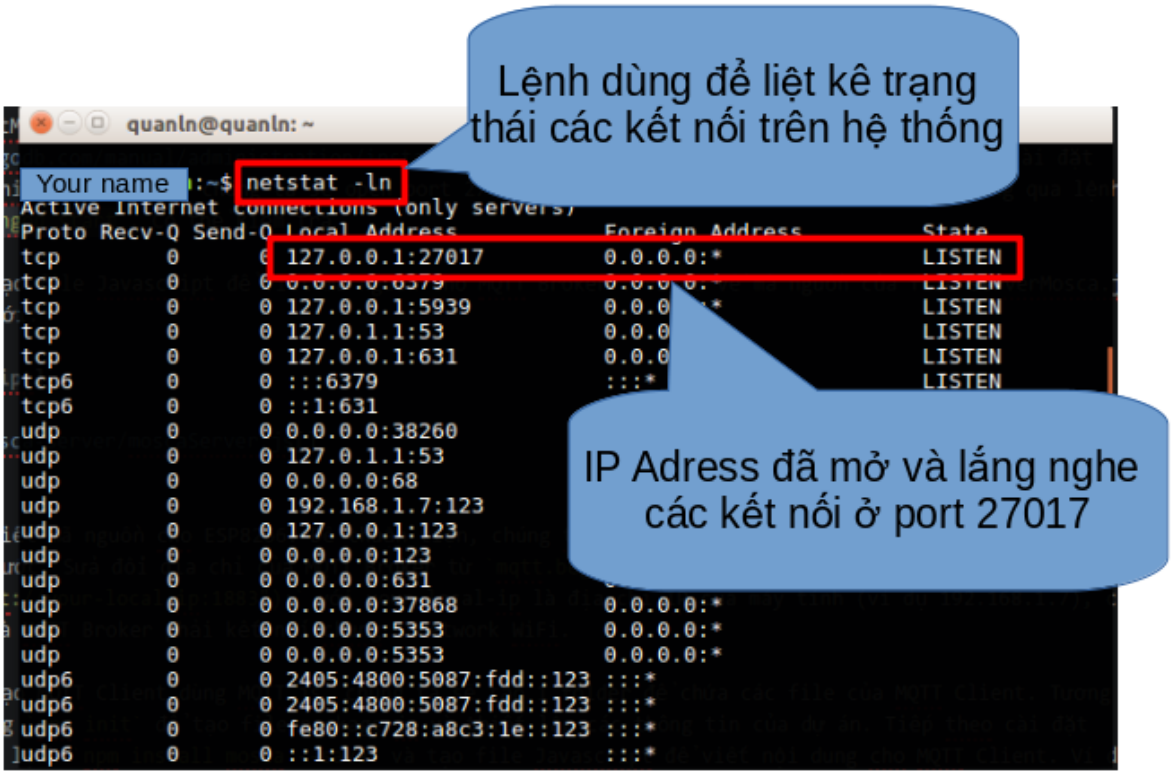
Mục tiêu

- Chúng ta sẽ tạo 1 MQTT Client là ESP8266 và 1 MQTT Client trên máy tính sử dụng MQTT.js nhằm kết nối đến MQTT Broker , subscribe topic và publish các message.
- Dùng Mosca tạo 1 MQTT Broker trên máy tính cá nhân nhằm broadcast messages (truyền bá các gói tin) đến các MQTT Client.

Các bước thực hiện

- **Bước 1 :** Trước tiên, chúng ta nên tạo 1 folder để thiết lập 1 MQTT Broker trên máy local. Đi đến folder này, tạo file package.js bằng lệnh `npm init` và thiết lập các thông tin của dự án. Tiếp theo, cài đặt module mosca bằng lệnh `npm install mosca --save`. Để lắng nghe các kết nối đến từ client cũng như lưu trữ dữ liệu về thông tin kết nối và nội dung các message ta cần công cụ hỗ trợ đó là `MongoDB`, bạn cũng có thể chọn Redis, Mosquitto, RabbitMQ… và tìm hiểu thêm về điểm mạnh, yếu của các cơ sở dữ liệu này. Để cài đặt MongoDB, chúng ta sẽ truy cập vào địa chỉ docs.mongodb.com/manual/administration/install-community/, tùy theo hệ điều hành để chọn gói cài đặt thích hợp. Sau khi cài đặt xong, chúng ta sẽ mở port 27017 (port mặc định khi dùng mongodb, chúng ta có thể điều chỉnh port ở file cấu hình của mongo[illegible] lắng nghe các kết nối từ client thông qua lệnh

`sudo service mongod start`. Trên hệ điều hành Linux, có thể kiểm tra các kết nối trên hệ thống bằng lệnh `Netstat` như hình dưới:



Hình 11. Hình ảnh port 27017 đã mở thành công và lắng nghe các kết nối

» **Bước 2 :** Tạo file Javascript để viết mã nguồn cho MQTT Broker. Ví dụ về mã nguồn của file serverMosca.js được viết bên dưới:

Mã nguồn file serverMosca.js

```
var mosca = require('mosca');   // Khai báo biến mosca sử dụng các thuộc tín
// Sử dụng thư viện ascoltatore nhằm hỗ trợ publish message, subscribe topic
var ascoltatore = {

  type: 'mongo',
  url: 'mongodb://localhost:27017/mqtt', // url: địa chỉ url của mongodb, se
                                         // chỉ localhost:27017
  pubsubCollection: 'ascoltatori',       // pubsubCollection: Nơi để lưu trữ
  mongo: {}                              // mongo: Cài đặt dành cho kết nối
};
var settings = {
  port: 1883,          // port kết nối đến server
  backend: ascoltatore // ascoltatore sẽ được gọi và thực thi khi tạo server
};

// Lệnh tạo server sử dụng mosca
var server = new mosca.Server(settings);
```

```
// Thực hiện hàm setup, in ra màn hình console nếu có sự kiện ready của serv
server.on('ready', setup);
function setup() {
  console.log('Mosca server is up and running');
}

// In ra dòng chữ client connected và id của client khi có sự kiện client kế
server.on('clientConnected', function(client) {
    console.log('client connected', client.id);
});

// In ra dòng chữ client disconnected và id của client khi có sự kiện client
server.on('clientDisconnected', function(client) {
    console.log('client disconnected', client.id);
});

// In ra message của client gửi ở dạng string khi có sự kiện client publish
server.on('published', function(packet, client) {
  console.log('Published', packet.payload.toString());
});
```

- **Bước 3 :** Viết mã nguồn cho ESP8266. Để nhanh chóng, chúng ta sẽ dùng mã nguồn của thư viện ESP8266MQTTClient đã viết ở mục trước. Sửa đổi địa chỉ của MQTT Broker từ `mqtt.begin("mqtt://iot.eclipse.org:1883");` thành `mqtt.begin("mqtt://your-local-ip:1883");` với your-local-ip là địa chỉ IP của máy tính (ví dụ 192.168.1.7), chú ý rằng ESP8266 và MQTT Broker phải kết nối chung 1 network WiFi.
- **Bước 4 :** Tạo MQTT Client dùng MQTT.js. Chúng ta sẽ tạo 1 folder để chứa các file của MQTT Client. Tương tự như bước 1 , dùng `npm init` để tạo file package.js và thiết lập các thông tin của dự án. Tiếp theo cài đặt module mqtt bằng lệnh `npm install mqtt --save` và tạo file Javascript để viết nội dung cho MQTT Client. Ví dụ về mã nguồn file `moscaClient.js` được trình bày bên dưới:

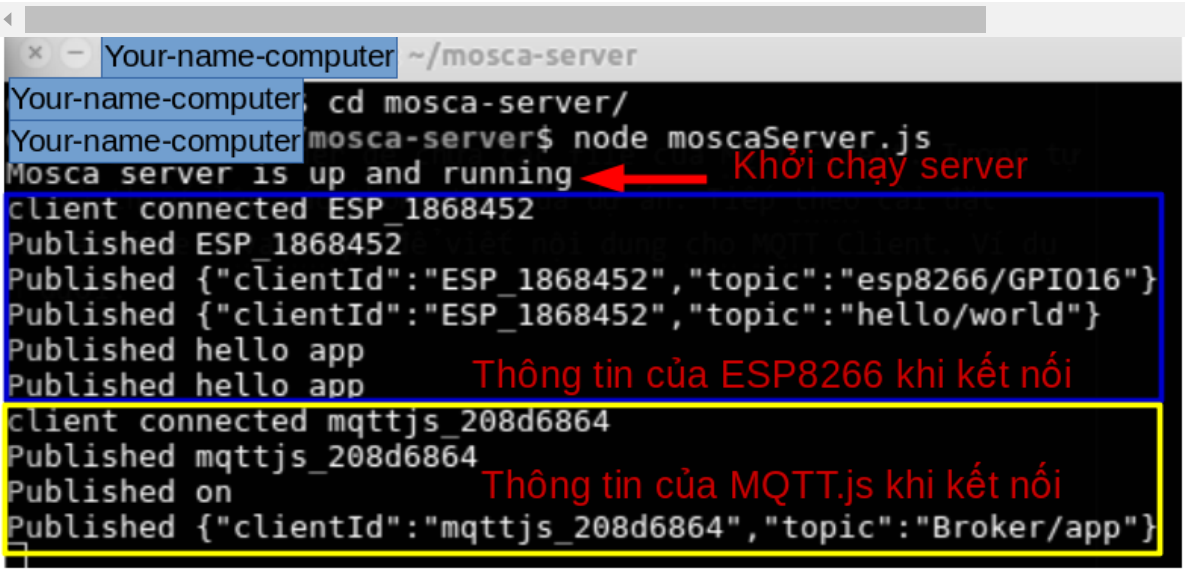
Mã nguồn file moscaClient.js

```
// Khai báo biến mqtt để sử dụng các thuộc tính thuộc module mqtt
var mqtt = require('mqtt');
// Tạo 1 kết nối đến địa chỉ 192.168.1.7 port 1883 thông qua giao thức MQTT
var client  = mqtt.connect('mqtt://192.168.1.7:1883');
// Khi có sự kiện connect đến server, client sẽ subscribe topic MQTTlens/tes
// publish 1 message "on" vào topic hello/world để ON led ở board ESP82
client.on('connect', function () {
```

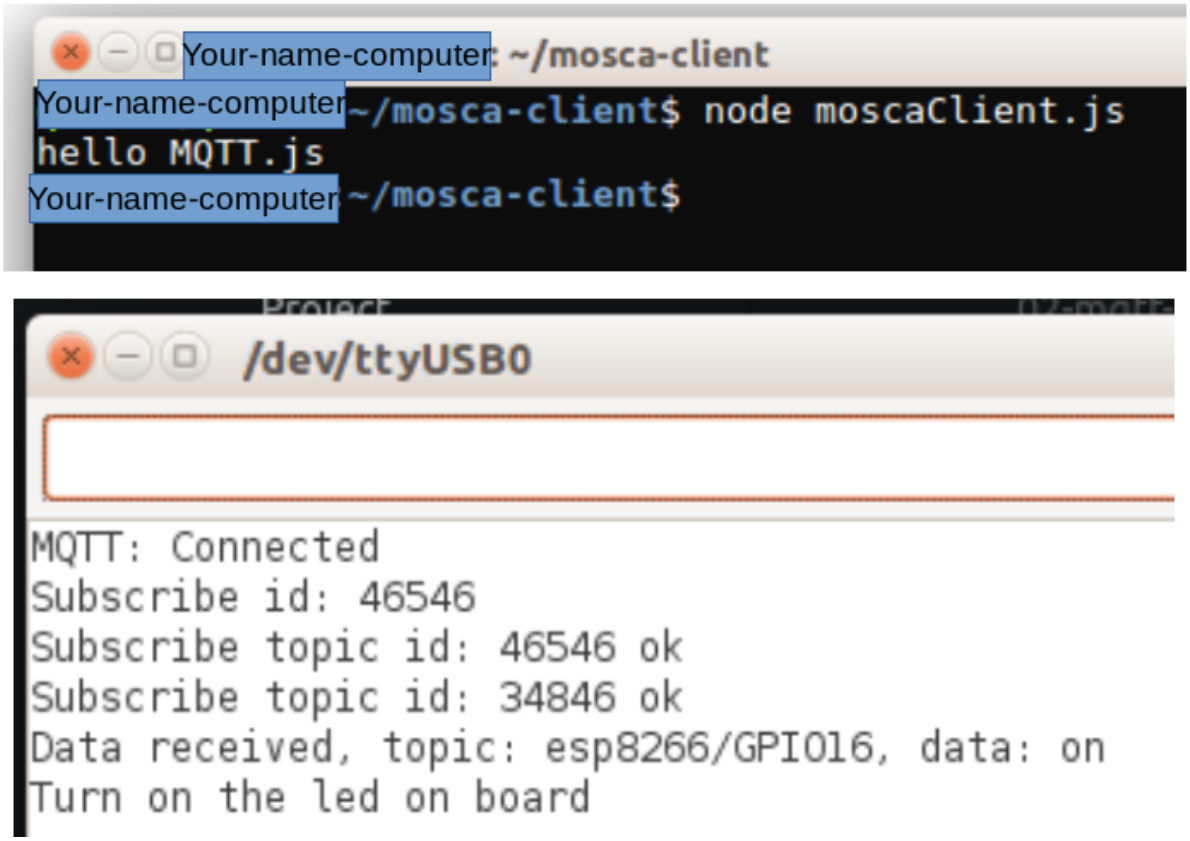
```
  client.subscribe('Broker/app');
  client.publish('esp8266/GPIO16', 'on');

})
// Khi có message gửi đến client, client sẽ chuyển đổi dữ liệu từ Buffer san
// hình console dữ liệu nhận được và đóng kết nối.
client.on('message', function (topic, message) {
  // message is Buffer
  console.log(message.toString());
  //client.end();
})
```

Kết quả

Trên terminal, đi đến thư mục chứa file `moscaServer.js` và khởi chạy server bằng lệnh node `moscaServer.js`. Server sẽ khởi động và lắng nghe các kết nối đến từ các MQTT Client. Tiếp theo, nạp chương trình trên Arduino cho ESP8266, sau đó khởi chạy MQTT Client trên máy tính bằng lệnh node `moscaClient.js`. Khi có các sự kiện kết nối , ngắt kết nối, pushlish 1 message hay subcrible 1 topic đến từ các client thì bên phía server đều sẽ hiển thị nội dung và thông tin. Các terminal hiển thị kết quả như hình bên dưới:



Hình 12. Hình ảnh thông tin nhận được từ các client ở server mosca



Hình 13. Hình ảnh thông tin nhận được ở client mqtt.js và client ESP8266

Một số MQTT Broker sử dụng cho sản phẩm thực tế

Mosquitto

Mosquitto là 1 MQTT Broker viết bằng ngôn ngữ lập trình C. Một số đặc điểm nổi bật của mosquitto là tốc độ truyền nhận và xử lí dữ liệu nhanh, độ ổn định cao, được sử dụng rộng rãi và phù hợp với những ứng dụng embedded. Thích hợp cho các hệ thống nhỏ chạy trên máy local như Raspberry Pi, bên cạnh đó Mosquitto cũng được hỗ trợ các giao thức TLS/SSL (các giao thức nhằm xác thực server và client, mã hóa các message để bảo mật dữ liệu).
Một số nhược điểm của mosquitto là khó thiết kế khi làm những ứng dụng lớn và ít phương thức xác thực thiết bị nên khả năng bảo mật vẫn chưa tối ưu.

EMQ

EMQ (Erlang MQTT Broker) là một MQTT Broker được viêt bằng ngôn ngữ lập trình Erlang. Ưu điểm của EMQ là tính ổn định cao, thích hợp để thiết kế các hệ thống lớn do khả năng mở rộng ứng dụng dễ dàng cũng như khá dễ để cài đặt. Ngoài ra EMQ còn hỗ trợ nhiều phương thức xác thực người dùng, phát triển và cập nhật tính năng liên tục bởi cộng đồng developer. Tuy nhiên điểm yếu của MQTT broker này là khó đối với những người mới bắt đầu. Thông tin về EMQ có thể xem tại trang emqttd-docs.readthedocs.io/en/latest/#

Tổng kết

Từ những nội dung đã trình bày ở trên, chúng ta phần nào hiểu rõ về cách thức hoạt động của giao thức MQTT cũng như vai trò của nó trong các ứng dụng IoT. Những nội dung được trình bày ở phần này chỉ là phần cơ bản của giao thức, vì đây là giao thức quan trọng và thường sử dụng trong IoT nên chúng ta hãy dành nhiều thời gian hơn để nghiên cứu thêm về hoạt động của các gói QoS, Keep alive, cũng như các vấn đề chứng thực tài khoản, vấn đề bảo mật dữ liệu khi sử dụng MQTT.

Thẻ:iot

Trả lời

Email của bạn sẽ không được hiển thị công khai. Các trường bắt buộc được đánh dấu *

Message

Your Name

Your E-mail

Your Website

Save my name, email, and website in this browser for the next time I comment.

Phản hồi