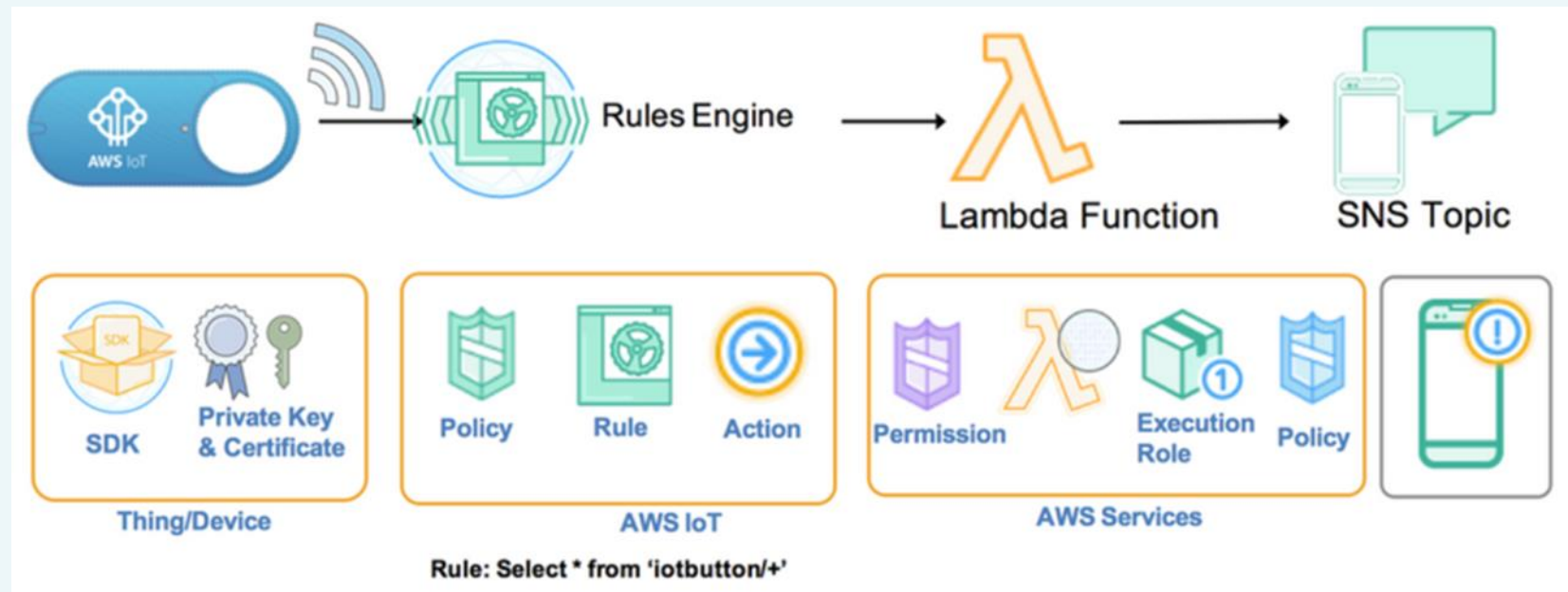


# **Internet of Things class 9**

**Connecting to AWS-IoT**

# Example: AWS-IoT Button

- A programmable button based on the [Amazon Dash Button](#) hardware
- Wi-Fi device designed to get started with [AWS IoT Core](#), [AWS Lambda](#), [Amazon DynamoDB](#), [Amazon SNS](#), and many other Amazon Web Services
- Applications: to count or track items, call or alert someone, start or stop something, order services, or even provide feedback. For example, to unlock or start a car, open your garage door, call a cab, call your spouse or a customer service representative, track the use of common household chores, medications or products, or remotely control your home appliances



# Install Libraries for AWS-IoT

- Installing Library files:

<https://github.com/ExploreEmbedded/Hornbill-Examples>

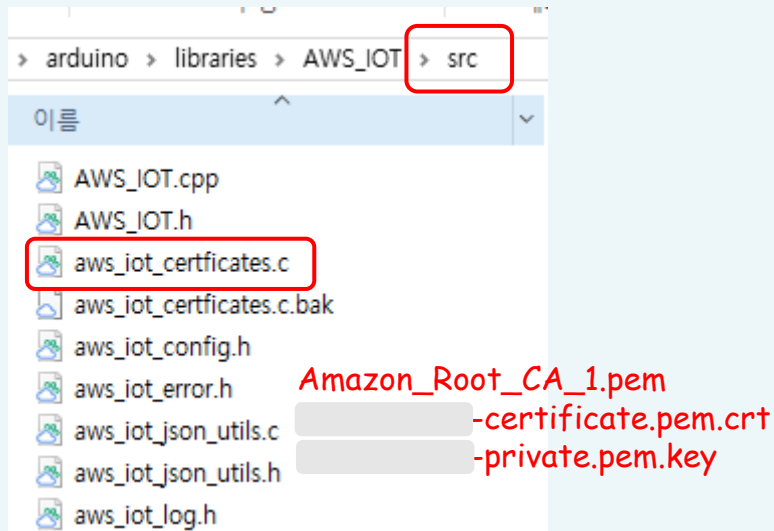
- Download zip: Hornbill-Examples-master.zip
- 압축을 풀고, arduino-esp32\AWS\_IOT 폴더를 ZIP으로 만듦 (AWS\_IOT.zip)

- Arduino > 스케치 > 라이브러리 포함하기 > .ZIP 라이브러리 추가

- Arduino IDE 재실행

- arduino\libraries\AWS\_IOT\src 폴더의 Certificates File 수정:

- 저장해 두었던 CA, Certificate, Private key File 사용



BEGIN부터.. END까지

```
const char aws_root_ca_pem[] = {"-----BEGIN CERTIFICATE-----\n\nMIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB41kPm1jZbyjANBgkqhkiG9w0BAQsF\nAQA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbW6\nb24gUm9vdCBDOQA5MB4XDTE1MDIvNiAwMDAwMEoXDTE1MDIvNiAwMDAwMEowOTEl\njMS1+ymRQ+HDKXJ10a1dXgJ0KR042M40WtBV800ZXJNDUZZHwLnOQdexeGADKpy\nr\nKRfboQnoZsG4q5WTP468SQvvG5\n-----END CERTIFICATE-----\n"};
```

매 라인마다 '\n' 추가

```
const char certificate_pem.crt[] = {"-----BEGIN CERTIFICATE-----\n\nMIIDWTTCAkGgAwIBAgIUTBDEcuMbTaRjwLAWZDimdhc/1XIwDQYJKoZIhvcNAQE\nBQAwTTFLEkGA1UECwwCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFTYXpvcvi5jb20g\nSW51LjRMPVNIYXR0bGUuU1Q9V2EzaGluZ3RvbiBDPVVMTB4XDTE1MDIvNiAwMDAw\n-----\n\nconst char private_pem_key[] = {"-----BEGIN RSA PRIVATE KEY-----\n\nMIIEpAIBAAKCAQEAzPH74vY5cNXIrCy2WJ9cy2JfxPX1laalLpxZQkkWc885GgT9\n0FkLuuH3UiVAAWSPd/xipvovoYk31E0syjHzH7Upi2dsSjDbBVFf0cKbTnHVY6sx\n-----\n"};
```

# Make thing: ESP32\_testButton

<Task09-B> Make thing (ESP32 button)

Connect ESP32Button to AWS-IoT

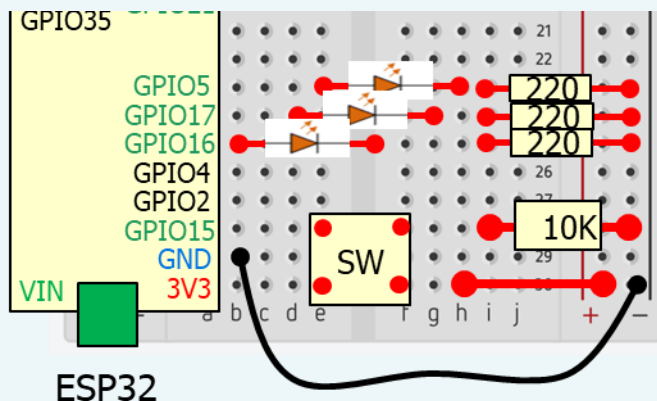
(pub: esp32/button, sub: esp32/buttAck)

Test to Publish and Subscribe Topics

Configure Rule-Engine as follows:

<if Button Pressed>

- send Email to yourself with Button ID
- reply ACK to ESP32 {"ack":{"message": "ACK for button"}}



```
Connecting to WiFi..  
Connected to wifi  
Connected to AWS  
Subscribe Successfull  
Publish failed  
Publish Message:ESP32-Button:001 Pressed!!  
Received Message:{"ack":{"message":"ACK for button"}}  
☒ 자동 스크롤 ☐ 타임스탬프 표시
```

mail to gen1223 3 받은편지함 AWS Notification Message - {"ack":{"message":"ACK for button"}} - If you wi

# Make thing: ESP32\_testButton

- Setting Thing-endpoint, Topics for Pub / Sub

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

AWS_IOT testButton;

const char* ssid = "KAU-Guest";
const char* password = "";
char HOST_ADDRESS[]="xxxxxxxx-ats.iot.ap-northeast-2.amazonaws.com";
char CLIENT_ID[]="ChoiESP32";
char sTOPIC_NAME[]="esp32/buttAck"; // subscribe topic name
char pTOPIC_NAME[]="esp32/button"; // publish topic name

int status = WL_IDLE_STATUS;
int msgCount=0,msgReceived = 0;
char payload[512];
char rcvdPayload[512];
const int buttonPin = 15; // pushbutton pin
unsigned long preMil = 0;
const long intMil = 5000;
```

AWS IoT > 사물 > ESP32\_testButton

사물

ESP32\_testButton

유형 없음

세부 정보

이 사물은 이미 연결되어 있는 것 같습니다.

보안

HTTPS

사물 그룹

결제 그룹

이 Rest API 엔드포인트를 사용하여 사물 채도우를 업데이트합니다. 자세히 알아보기

채도우

-ats.iot.ap-northeast-2.amazonaws.com

상호 작용

## Make thing: ESP32\_testButton

- **Setting Thing-endpoint, Topics for Pub / Sub**

[AWS IoT](#) > [연결](#) > 디바이스 한 개 연결

Step 1  
디바이스 준비

Step 2  
디바이스 등록 및 보안

Step 3  
플랫폼 및 SDK 선택

Step 4  
연결 키트 다운로드

Step 5  
연결 키트 실행

## 디바이스 준비 정보

### 작동 방식

이 마법사에서는 사물 리소스를 AWS IoT에 생성합니다. 사물 리소스는 물리적 디바이스 또는 논리적 엔터티를 디지털로 표현한 것입니다.

사물 리소스는 디바이스 할당, 소프트웨어, 이 마법사와 연결된 정보입니다.

### 디바이스 준비

- 디바이스의 전원을 켜고 인터넷에 연결되어 있는지 확인합니다.
- 디바이스에 파일을 로드하는 방법을 선택합니다.
  - 디바이스가 브라우저를 지원하는 경우 디바이스에서 AWS IoT 콘솔을 열고 이 마법사를 실행합니다. 브라우저에서 디바이스로 파일을 직접 다운로드할 수 있습니다.
  - 디바이스가 브라우저를 지원하지 않는 경우 브라우저를 사용하여 컴퓨터에서 디바이스로 파일을 전송하는 가장 좋은 방법을 선택합니다. 파일 전송 프로토콜(FTP)을 사용하거나 USB 메모리 스틱을 사용할 수 있습니다.
- 디바이스에서 명령줄 인터페이스에 액세스할 수 있는지 확인합니다.
  - IoT 디바이스에서 이 마법사를 실행하는 경우, 디바이스에서 터미널 창을 열어 명령줄 인터페이스에 액세스합니다.
  - IoT 디바이스에서 이 마법사를 실행하지 않는 경우, 이 디바이스에서 SSH 터미널 창을 열고 IoT 디바이스에 연결합니다.
- 터미널 창에서 다음 명령을 입력합니다.
 

```
ping a111a2b3c4d5e6f7g8h9i0j-k1l2m3n4o5p6q7r8s9t0-ats.iot.ap-northeast-2.amazonaws.com
```

복사

이러한 단계를 완료하고 성공적인 ping 응답을 받으면 계속해서 디바이스를 AWS IoT에 연결할 준비가 된 것입니다.

취소

다음

# Make thing: ESP32\_testButton

---

- Build Subscribe callback handler
  - Called when Subscribe Topic arrives with Payload

```
void mySubCallbackHandler (char *topicName, int payloadLen, char *payLoad)
{
    strncpy(rcvdPayload,payLoad,payloadLen);
    rcvdPayload[payloadLen] = 0;
    msgReceived = 1;
}
```

# Make thing: ESP32\_testButton

---

- Setup WiFi

```
void setup() {  
  Serial.begin(115200);  
  //++choi This is here to force the ESP32 to reset the WiFi and initialize correctly.  
  Serial.print("WIFI status = ");  
  Serial.println(WiFi.getMode());  
  WiFi.disconnect(true);  
  delay(1000);  
  WiFi.mode(WIFI_STA);  
  delay(1000);  
  Serial.print("WIFI status = ");  
  Serial.println(WiFi.getMode());          //++choi  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL_CONNECTED) {  
    delay(1000);  
    Serial.println("Connecting to WiFi..");  
  }  
  Serial.println("Connected to wifi");  
}
```



# Make thing: ESP32\_testButton

- Setup Connecting to AWS
  - Register Subscribe Callback Handler with Topic Name
- Initialize Test Button

```
if(testButton.connect(HOST_ADDRESS,CLIENT_ID)== 0) {  
    Serial.println("Connected to AWS");  
    delay(1000);  
    if(0==testButton.subscribe(sTOPIC_NAME,mySubCallBackHandler)) {  
        Serial.println("Subscribe Successfull");  
    }  
    else {  
        Serial.println("Subscribe Failed, Check the Thing Name and Certificates");  
        while(1);  
    }  
}  
else {  
    Serial.println("AWS connection failed, Check the HOST Address");  
    while(1);  
}  
// initialize the pushbutton pin as an input  
pinMode(buttonPin, INPUT);  
delay(2000);  
}
```

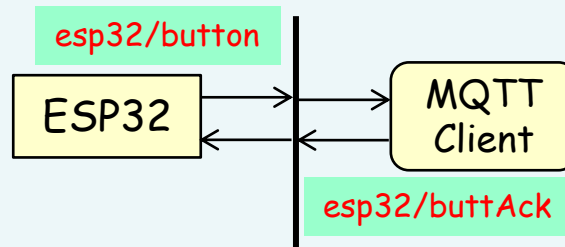
# Make thing: ESP32\_testButton

- Loop
  - Check and Print Subscribe Topic with Payload
  - Check the Button status every 5 Sec at least
    - Publish Topic with Button ID if Button Pressed

```
void loop() {
  if(msgReceived == 1) {
    msgReceived = 0;
    Serial.print("Received Message:");
    Serial.println(rcvdPayload);
  }
  if((millis()-preMil) > intMil) {
    // read the state of the pushbutton value
    if (digitalRead(buttonPin)) {
      preMil = millis();
      sprintf(payload,"ESP32-Button:001 Pressed!!");
      if(testButton.publish(pTOPIC_NAME,payload) == 0) {
        Serial.print("Publish Message:");
        Serial.println(payload);
      }
    }
    else
      Serial.println("Publish failed");
  }
}
```

# Test thing: ESP32\_testButton

- MQTT Client:
  - Subscribe.. “esp32/button”
  - Publish.. “esp32/buttAck”



게시

QoS of 0으로 게시할 주제와 메시지를 지정합니다.

esp32/buttAck

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

esp32/button

11월 02, 2020, 18:42:07 (UTC+0900)

JSON으로 메시지를 표시할 수 없어 UTF-8 문자열로 대신 표시합니다.

ESP32-Button:001 Pressed!!

```
Connecting to WiFi..
Connected to wifi
Connected to AWS
Subscribe Successfull
Received Message:{
  "message": "Hello from AWS IoT console"
}
```

☒ 자동 스크롤 ☐ 타임스탬프 표시

# Action for ESP32-Button

## ■ Configure Rule-Engine

AWS IoT > 규칙 > esp32\_testButton\_Action

규칙

### esp32\_testButton\_Action

활성

개요

Tags

설명

if ESP32 button is pressed, send Email with Button ID

규칙 쿼리 설명문

이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.

```
SELECT {"message":"ACK for button"} as ack FROM 'esp32/button'
```

SQL 버전 사용 2016-03-23

작업

작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기



SNS 푸시 알림 메시지 전송  
emailToMe



메시지를 AWS IoT 주제에 재게시  
esp32/buttAck

AWS IoT > 규칙 > esp32\_testButton\_Action

### 작업 구성



SNS 푸시 알림 메시지 전송  
SNS

\*SNS 대상

emailToMe

메시지 형식

RAW

AWS IoT > 규칙 > esp32\_testButton\_Action

### 작업 구성



메시지를 AWS IoT 주제에 재게시  
AWS IoT 재게시

이 작업은 메시지를 다른 AWS IoT 주제에 재게시

\*주제 ?

esp32/buttAck

서비스 품질 ?

- ☒ 0 - 메시지가 0번 이상 전달됩니다.
- ☐ 1 - 메시지가 1번 이상 전달됩니다.

# Results for ESP32-Button

- Configure Rule-Engine
- Press Button
- Check Email
- Check Serial Monitor

```
Connecting to WiFi..  
Connected to wifi  
Connected to AWS  
Subscribe Successfull  
Publish failed  
Publish Message:ESP32-Button:001 Pressed!!  
Received Message:{"ack":{"message":"ACK for button"}}
```

☒ 자동 스크롤 ☐ 타임스탬프 표시

mail to gen1223 3

받은편지함 AWS Notification Message - {"ack":{"message":"ACK for button"}} – If you wi

# Test Thing by MQTT-fx

## <Task09-A>

- Install MQTT-fx.. from Internet..
- Thing을 만들기전에 Topic/Payload 동작 확인

– Broker addr:

Thing Rest-API endpoint

– Broker Port: 8883

– CA, Certificate,  
Private-key file 지정  
(for my-thing)

Profile Name: myIoTButton

Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: ats.iot.ap-northeast-2.amazonaws.com

Broker Port: 8883

Client ID: MQTT\_FX\_Client

Generate

General User Credentials **SSL/TLS** Proxy LWT

Enable SSL/TLS ☒ Protocol: TLSv1.2

☐ CA signed server certificate

☐ CA certificate file

☐ CA certificate keystore

☒ Self signed certificates

CA File: G:\내 드라이브\@KAU\교육과정\2020-2\IoT\AWS-IoT\Amazon\_Roc

Client Certificate File: G:\내 드라이브\@KAU\교육과정\2020-2\IoT\AWS-IoT\ESP32\_testE

Client Key File: G:\내 드라이브\@KAU\교육과정\2020-2\IoT\AWS-IoT\ESP32\_testE

Client Key Password:

PEM Formatted ☒

# Test Thing by MQTT-fx

---

- Rule-Engine Setup:

1. 온도가 40 이상이면 자신의 email로 통보하고
2. 온도가 30 이상이고 습도가 40 이상일때 LED를 OFF
3. 온도가 20 이하이고 습도가 10 이하일때 LED를 ON

- Topic Setup:

Publish: esp32/bme280 {"temp":n, "humid":n, "press":n}

Subscribe: esp32/led {"state": {"led": "ON" | "OFF"}}

# Test Thing by MQTT-fx

## ■ Rule-Engine Setup

규칙

esp32\_bmeled\_control1

활성

작업

개요

설명

Tags

판

1. emailToGen1223 if temp >= 40 2. led off if temp >= 30 and humid >= 40 3. led on if temp <=20 and humid <= 10

규칙 쿼리 설명문

판


이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.

```
SELECT * FROM 'esp32/bme280' WHERE temp >= 40
```

SQL 버전 사용 2016-03-23

작업

작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기

 SNS 푸시 알림 메시지 전송  
emailToGen1223

제거 편집 ▶



# Test Thing by MQTT-fx

## ■ Rule-Engine Setup

**esp32\_bmeled\_control2**  
활성

개요

설명

Tags

1. emailToGen1223 if temp >= 40 2. led off if temp >= 30 and humid >= 40 3. led on if temp <=20 and humid <= 10

규칙 쿼리 설명문

이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.

```
SELECT {"led":"OFF"} as state FROM 'esp32/bme280' WHERE temp >= 30 AND humid >= 40
```

SQL 버전 사용 2016-03-23

작업

작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기

 메시지를 AWS IoT 주제에 재게시  
esp32/led

제거 편집 ▶

# Test Thing by MQTT-fx

## ■ Rule-Engine Setup

**esp32\_bmeled\_control3**  
활성

개요

설명

Tags

1. emailToGen1223 if temp >= 40 2. led off if temp >= 30 and humid >= 40 3. led on if temp <=20 and humid <= 10

규칙 쿼리 설명문

이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.

```
SELECT {"led":"ON"} as state FROM 'esp32/bme280' WHERE temp <= 20 AND humid <= 10
```

SQL 버전 사용 2016-03-23

작업

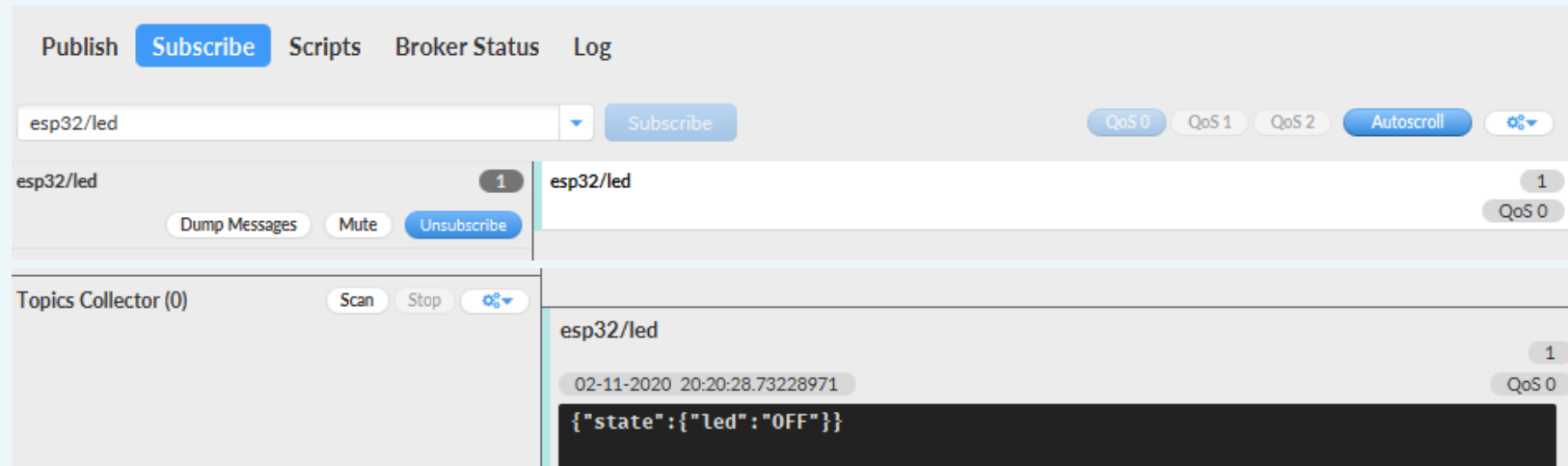
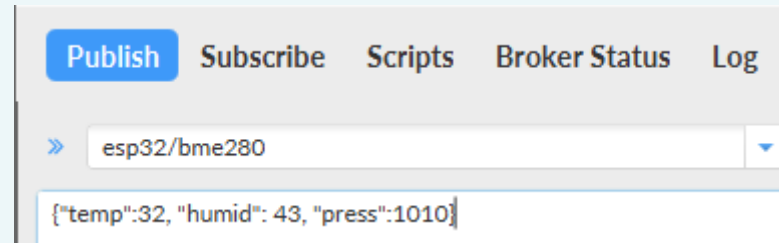
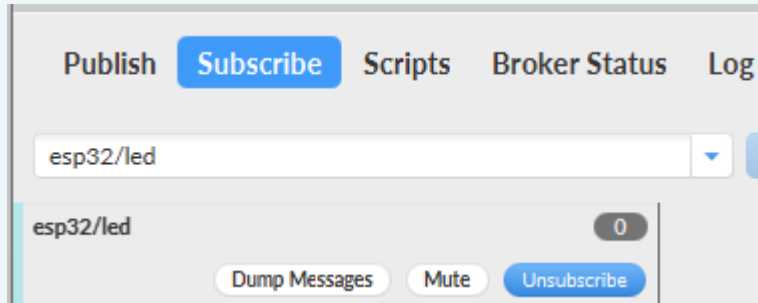
작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기

 메시지를 AWS IoT 주제에 재게시  
esp32/led

제거 편집 ▶

# Test Thing by MQTT-fx

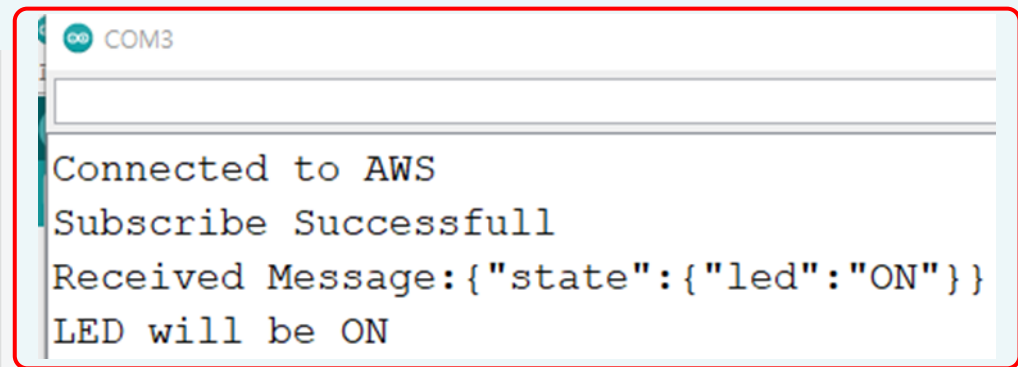
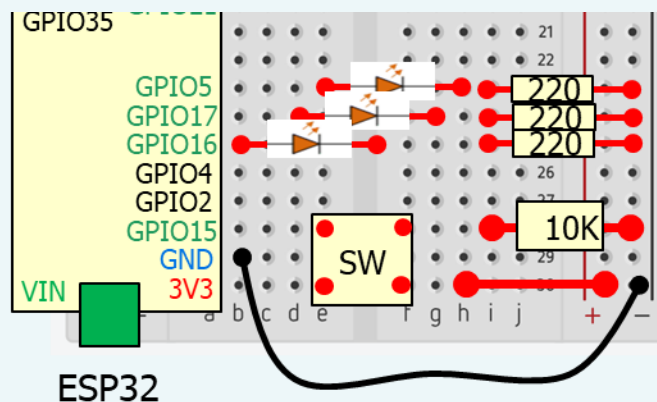
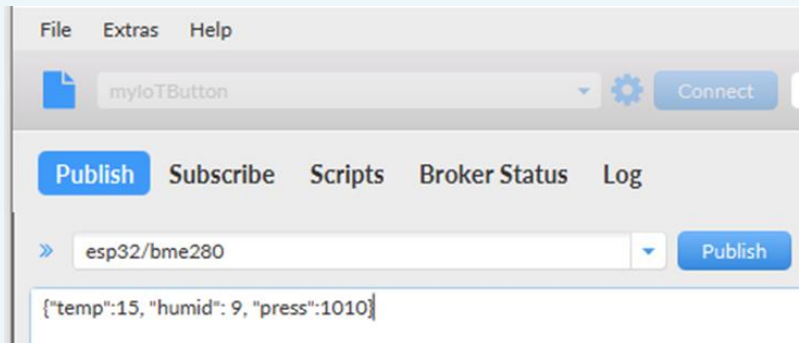
- Test by using MQTT-fx



# Make thing: ESP32\_led

## <Task09-A> Make thing (ESP32\_led)

- Publish 'esp32/bme280' by using MQTT-fx
- Rule-Engine will publish 'esp32/led'
- Your ESP32 should receive 'esp32/led' topic and control led



# Make thing: ESP32\_led

<Task09-A> use task09-b and “ArduinoJson” example

```
#include <AWS_IOT.h>
#include <WiFi.h>
#include <Arduino_JSON.h> // refer JSONObject example for more information!!!

...
char sTOPIC_NAME[] = "esp32/led"; // subscribe topic name
// char pTOPIC_NAME[] = "esp32/bme280"; // publish topic name.. Not yet

...
const int ledPin = 16; // led pin

...
void loop() {

    if(msgReceived == 1)
    {
        msgReceived = 0;
        Serial.print("Received Message:");
        Serial.println(rcvdPayload);
        // Parse JSON
        JSONVar myObj = JSON.parse(rcvdPayload);
        JSONVar state = myObj["state"];
        String led = (const char*) state["led"];
        Serial.print("LED will be ");
        Serial.println(led);
        if (led == "ON")
            digitalWrite(ledPin, HIGH);
        else if (led == "OFF")
            digitalWrite(ledPin, LOW);
    }

    ...
}
```

# Make thing: ESP32\_bme280\_led

## <Task09-C> Make thing (ESP32\_bme280\_led)

- Connect BME280 onto your ESP32 with I2C

- Publish 'esp32/bme280'

{“temp”:n, “humid”:n, “press”:n}

\*\* Publish period : 10 ~ 20 sec

or every time button pressed

- Rule-Engine will send email,  
publish 'esp32/led' on condition of Step-A

- Your ESP32 should change LED or  
send mail according to temp, humid..  
( Modify condition of your Rule-Engine  
for test )

