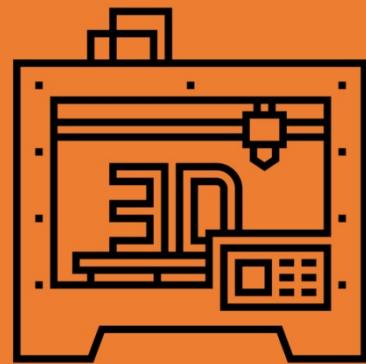
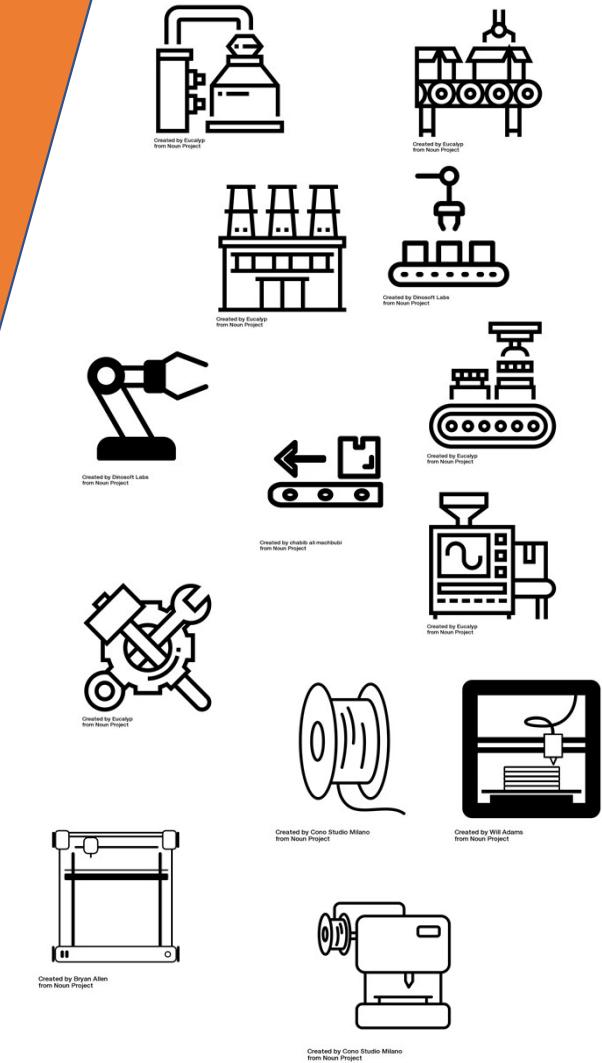


IOT 클라우드 플랫폼

1595016김현우
1594023유병찬



Created by Jemis mali
from Noun Project



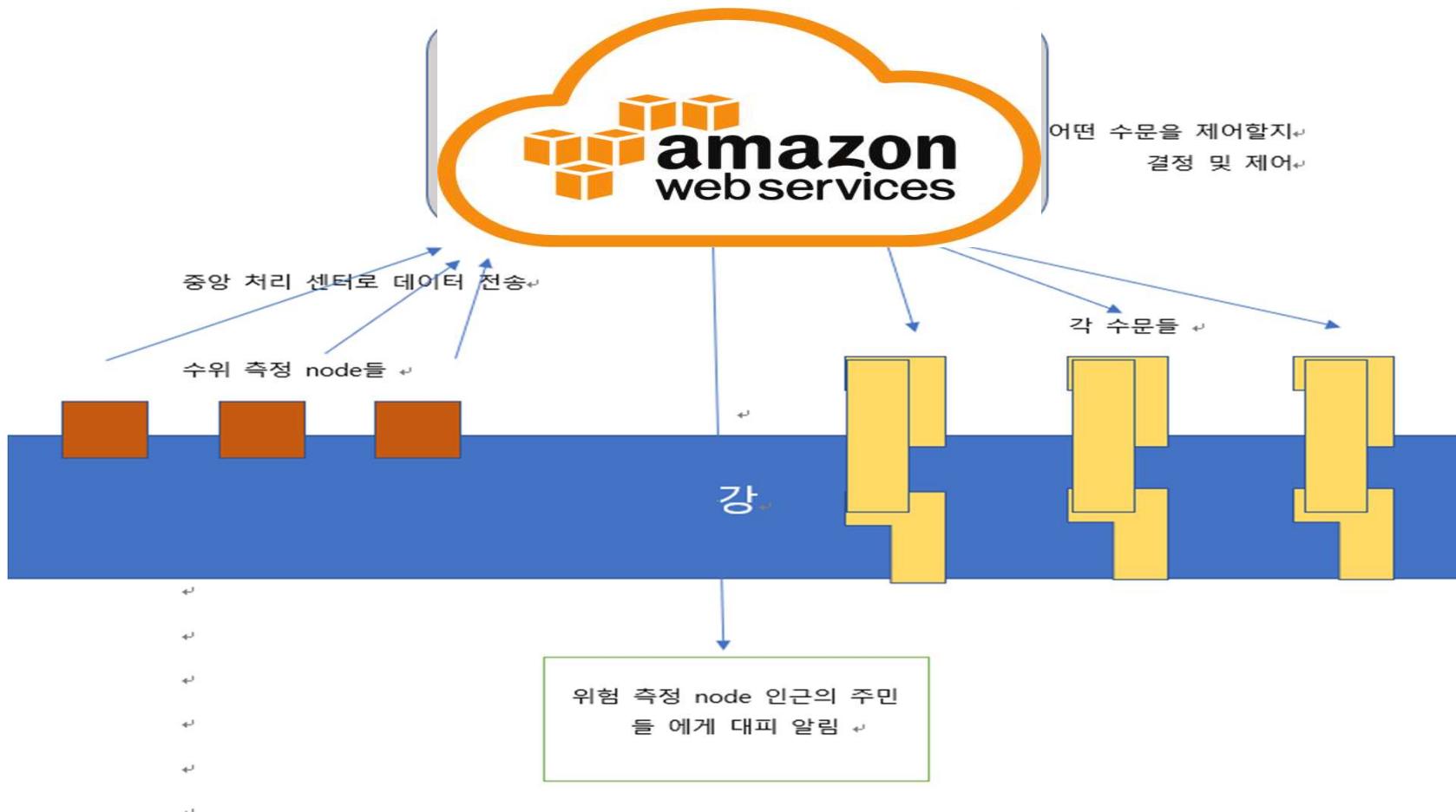
목차

- 1. 서비스 개요
- 2. 제품 동작 설명
- 3. 과정 및 코드 설명
- 3. 제품 사진
- 4. 시연 영상

서비스 개요



서비스 구조도



서비스 구현 과정-디바이스 생성(수위측정 노드)

AWS_IoT_WaterLevel | 아두이노 1.8.9 (Windows Store 1.8.21.0)

파일 편집 스케치 를 도움말



```
#include <ArduinoBearSSL.h>
#include <ArduinoECCX08.h>
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h> // change to #include <WiFi101.h> for MKR1000

#include "arduino_secrets.h"

#define LED_1_PIN 5 //RED
#define LED_2_PIN 4 //YELLOW
#define LED_3_PIN 3 //GREEN

#include <ArduinoJson.h>
#include "Led.h"

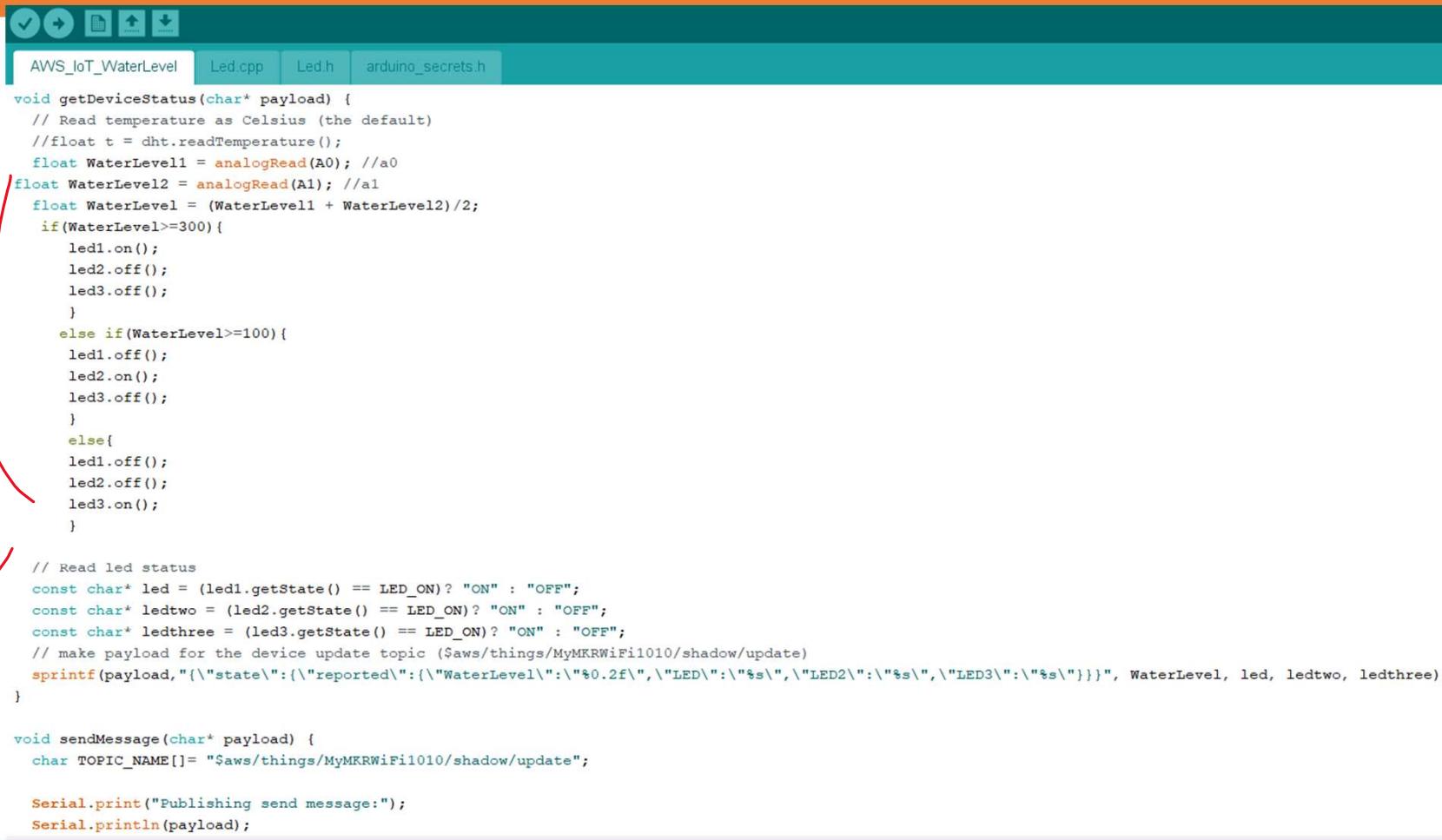
////// Enter your sensitive data in arduino_secrets.h
const char ssid[] = SECRET_SSID;
const char pass[] = SECRET_PASS;
const char broker[] = SECRET_BROKER;
const char* certificate = SECRET_CERTIFICATE;

WiFiClient wifiClient; // Used for the TCP socket connection
BearSSLClient sslClient(wifiClient); // Used for SSL/TLS connection, integrates with ECC508
MqttClient mqttClient(sslClient);

unsigned long lastMillis = 0;

Led led1(LED_1_PIN);
Led led2(LED_2_PIN);
Led led3(LED_3_PIN);
```

서비스 구현 과정 -디바이스 생성(수위측정 노드)



The screenshot shows the Arduino IDE interface with the following details:

- File Menu:** File, Open, Save, Print, Upload, Download, Preferences
- Project Tab:** AWS_IoT_WaterLevel (selected), Led.cpp, Led.h, arduino_secrets.h
- Code Area:** The main code block contains C++ code for an Arduino project. It includes functions for reading sensor values, controlling LEDs based on water level, and generating a payload for AWS IoT. A red bracket on the left side groups the first two sections of the code.

```
void getDeviceStatus(char* payload) {
    // Read temperature as Celsius (the default)
    //float t = dht.readTemperature();
    float WaterLevel1 = analogRead(A0); //a0
    float WaterLevel2 = analogRead(A1); //a1
    float WaterLevel = (WaterLevel1 + WaterLevel2)/2;
    if(WaterLevel>=300){
        led1.on();
        led2.off();
        led3.off();
    }
    else if(WaterLevel>=100){
        led1.off();
        led2.on();
        led3.off();
    }
    else{
        led1.off();
        led2.off();
        led3.on();
    }

    // Read led status
    const char* led = (led1.getState() == LED_ON)? "ON" : "OFF";
    const char* ledtwo = (led2.getState() == LED_ON)? "ON" : "OFF";
    const char* ledthree = (led3.getState() == LED_ON)? "ON" : "OFF";
    // make payload for the device update topic ($aws/things/MyMKRWiFi1010/shadow/update)
    sprintf(payload, "{\"state\":{\"reported\":{\"WaterLevel\":\"%0.2f\", \"LED\":\"%s\", \"LED2\":\"%s\", \"LED3\":\"%s\"}}}", WaterLevel, led, ledtwo, ledthree);
}

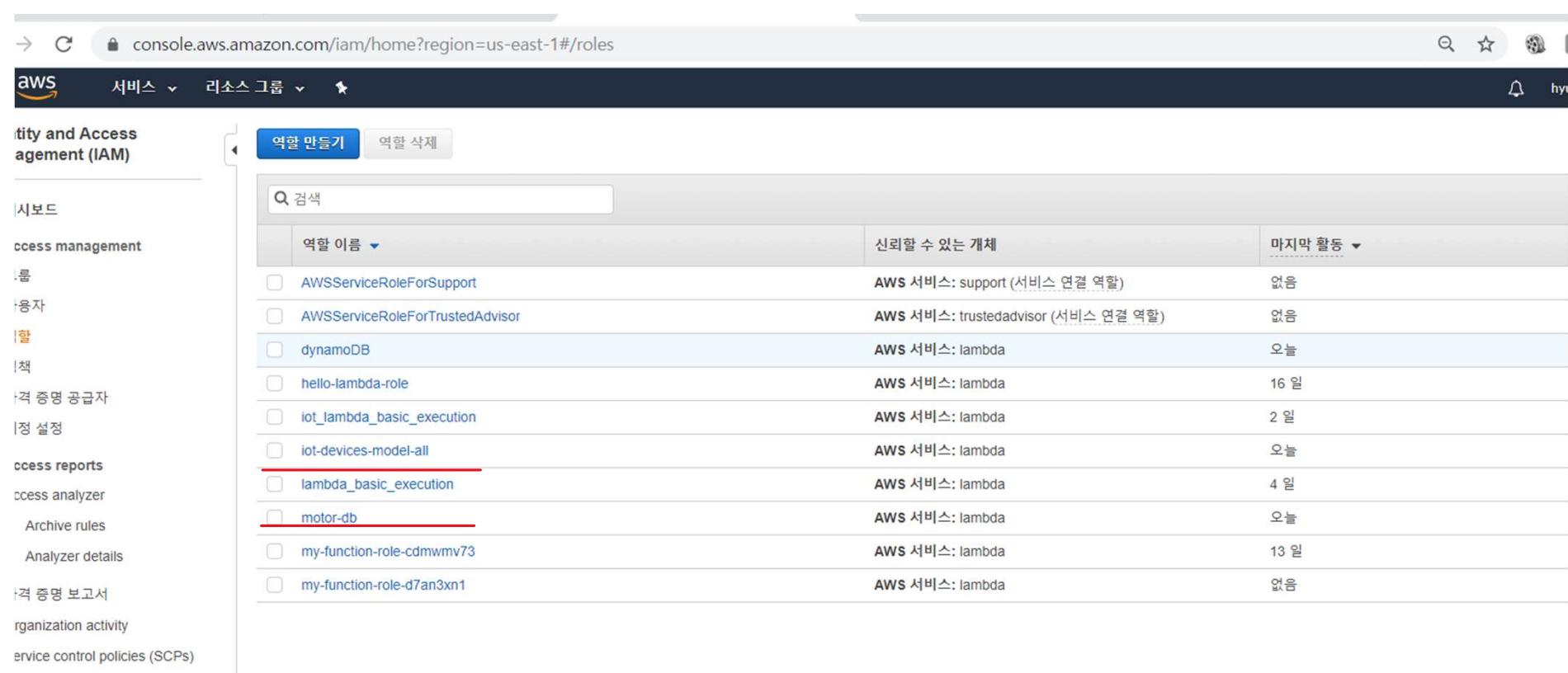
void sendMessage(char* payload) {
    char TOPIC_NAME[] = "$aws/things/MyMKRWiFi1010/shadow/update";

    Serial.print("Publishing send message:");
    Serial.println(payload);
```

서비스 구현 과정-디바이스 생성(수위측정 노드)

The screenshot shows the AWS IoT Device Management console. The top navigation bar includes the AWS logo, service dropdown, resource group dropdown, and user information (hyunwoonaver, 베지니아 북부, 지원). On the left, a sidebar menu lists: 모니터링, 온보딩, 관리 (selected), 사물, 유형, 사물 그룹, 결제 그룹, 작업, and 터널. The main content area has a title '사물' and a search bar with placeholder '사물 검색'. A green notification box at the top right says '선택한 리소스를 삭제했습니다.' (The selected resources have been deleted.). Below the search bar is a button labeled '플릿 인덱싱' with a count of 0. A card view displays a single device entry: 'MyMKRWiFi1010' with a status of '유형 없음' and three vertical dots for more options.

서비스 구현 과정 - IAM 역할 만들기



The screenshot shows the AWS IAM Roles page. The left sidebar lists various IAM-related services and reports. The main area displays a table of existing roles, with two specific roles highlighted with red underlines: "iot-devices-model-all" and "motor-db".

역할 이름	신뢰할 수 있는 개체	마지막 활동
AWSServiceRoleForSupport	AWS 서비스: support (서비스 연결 역할)	없음
AWSServiceRoleForTrustedAdvisor	AWS 서비스: trustedadvisor (서비스 연결 역할)	없음
dynamoDB	AWS 서비스: lambda	오늘
hello-lambda-role	AWS 서비스: lambda	16 일
iot_lambda_basic_execution	AWS 서비스: lambda	2 일
iot-devices-model-all	AWS 서비스: lambda	오늘
lambda_basic_execution	AWS 서비스: lambda	4 일
motor-db	AWS 서비스: lambda	오늘
my-function-role-cdmwmv73	AWS 서비스: lambda	13 일
my-function-role-d7an3xn1	AWS 서비스: lambda	없음

서비스 구현 과정 - DB 테이블 만들기

The screenshot shows the Alibaba Cloud Table Service management interface. At the top, there are navigation links for '서비스' (Service), '리소스 그룹' (Resource Group), and a user dropdown for 'hyunwoonaver'. On the right, there are icons for notifications, user profile, region (Beijing North), and account (Ziyuan). Below the header, there are two tabs: '테이블 만들기' (Create Table) which is selected, and '테이블 삭제' (Delete Table). The main area is a table listing seven existing tables:

이름	상태	파티션 키	정렬 키	인덱스	총 읽기 용량	총 쓰기 용량	Auto Scaling	암호화
DeviceData	활성	time (번호)	-	0	5	5	-	기본값
motordb	활성	deviceID (문자열)	time (번호)	0	5	5	-	기본값
People	활성	id (번호)	-	0	5	5	-	기본값
testV2	활성	time (번호)	-	0	5	5	-	기본값
waterlevel	활성	time (번호)	-	0	5	5	-	기본값
waterleveldb	활성	time (번호)	-	0	5	5	-	기본값
waterleveldb2	활성	deviceID (문자열)	time (번호)	0	5	5	-	기본값

서비스 구현 과정-DB에 올리기 위한 Lambda함수생성

n/java/com/example/lambda/recording/RecordingDeviceInfoHandler2.java - Eclipse IDE

Run Window Help

RecordingDe... RecordingDe... UpdateDevic... C:\Users\kh... MotorUpdate... GetDeviceHan... C:\Users\kh... MotorGetDev... »20

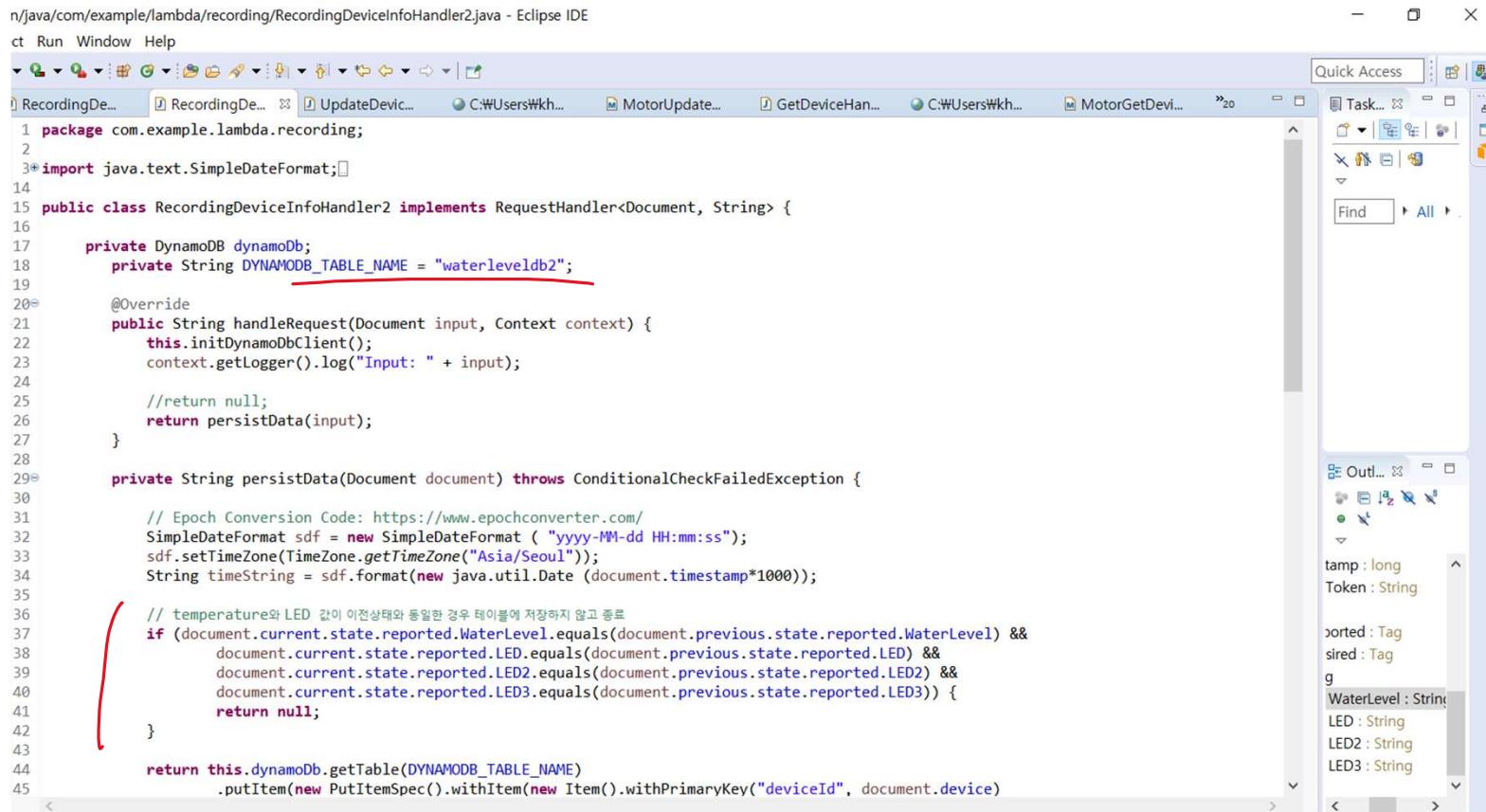
```
1 package com.example.lambda.recording;
2
3 import java.text.SimpleDateFormat;
4
5 public class RecordingDeviceInfoHandler2 implements RequestHandler<Document, String> {
6
7     private DynamoDB dynamoDb;
8     private String DYNAMODB_TABLE_NAME = "waterleveldb2";
9
10    @Override
11    public String handleRequest(Document input, Context context) {
12        this.initDynamoDbClient();
13        context.getLogger().log("Input: " + input);
14
15        //return null;
16        return persistData(input);
17    }
18
19    private String persistData(Document document) throws ConditionalCheckFailedException {
20
21        // Epoch Conversion Code: https://www.epochconverter.com/
22        SimpleDateFormat sdf = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
23        sdf.setTimeZone(TimeZone.getTimeZone("Asia/Seoul"));
24        String timeString = sdf.format(new java.util.Date (document.timestamp*1000));
25
26        // temperature와 LED 같이 이전상태와 동일한 경우 테이블에 저장하지 않고 종료
27        if (document.current.state.reported.WaterLevel.equals(document.previous.state.reported.WaterLevel) &&
28            document.current.state.reported.LED.equals(document.previous.state.reported.LED) &&
29            document.current.state.reported.LED2.equals(document.previous.state.reported.LED2) &&
30            document.current.state.reported.LED3.equals(document.previous.state.reported.LED3)) {
31            return null;
32        }
33
34        return this.dynamoDb.getTable(DYNAMODB_TABLE_NAME)
35            .putItem(new PutItemSpec().withItem(new Item().withPrimaryKey("deviceId", document.device)
36
37
38
39
40
41
42
43
44
45
```

Quick Access

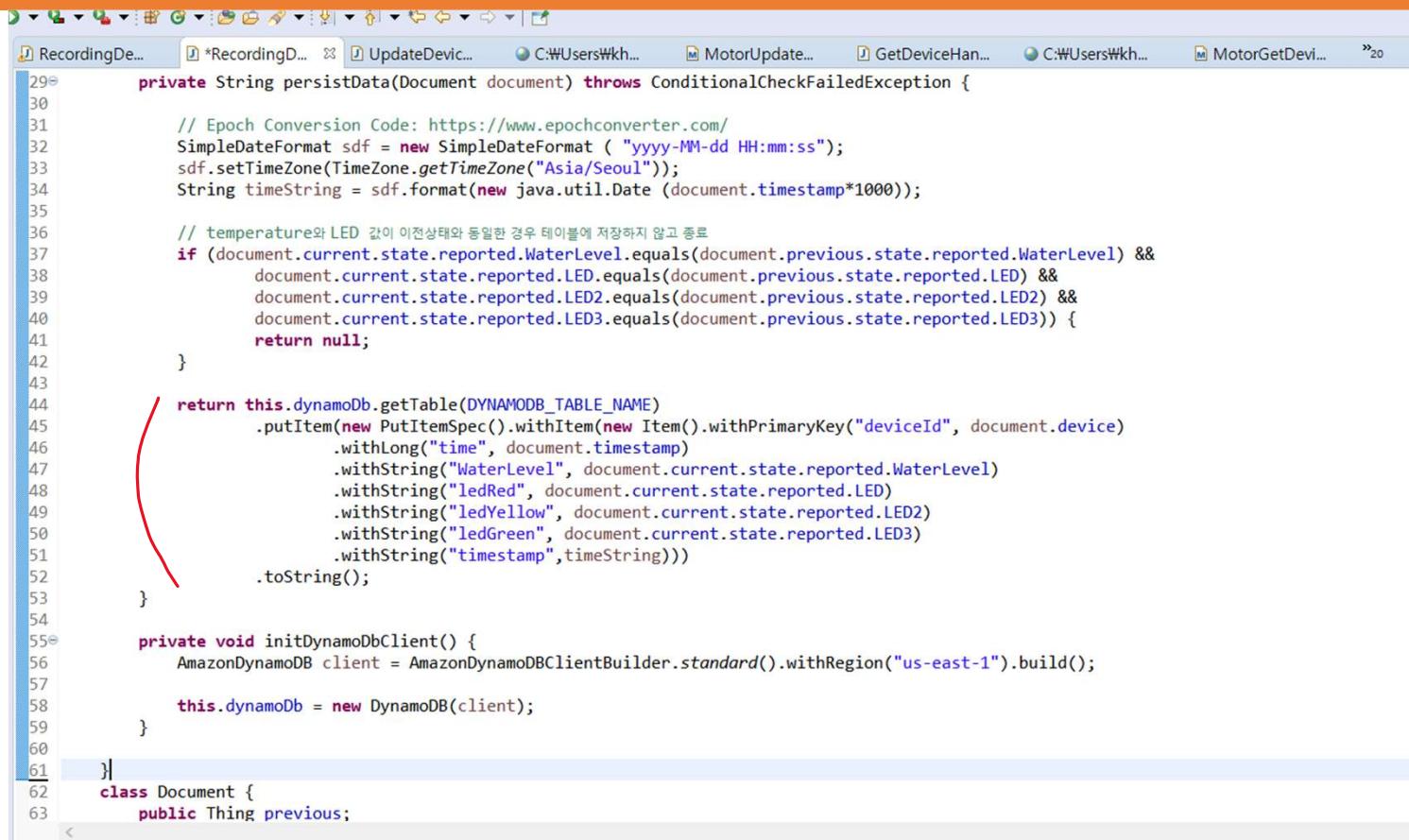
Task... Find All

Out... tamp : long Token : String

ported : Tag sired : Tag g WaterLevel : String LED : String LED2 : String LED3 : String



서비스 구현 과정 -DB에 올리기 위한 Lambda함수생성

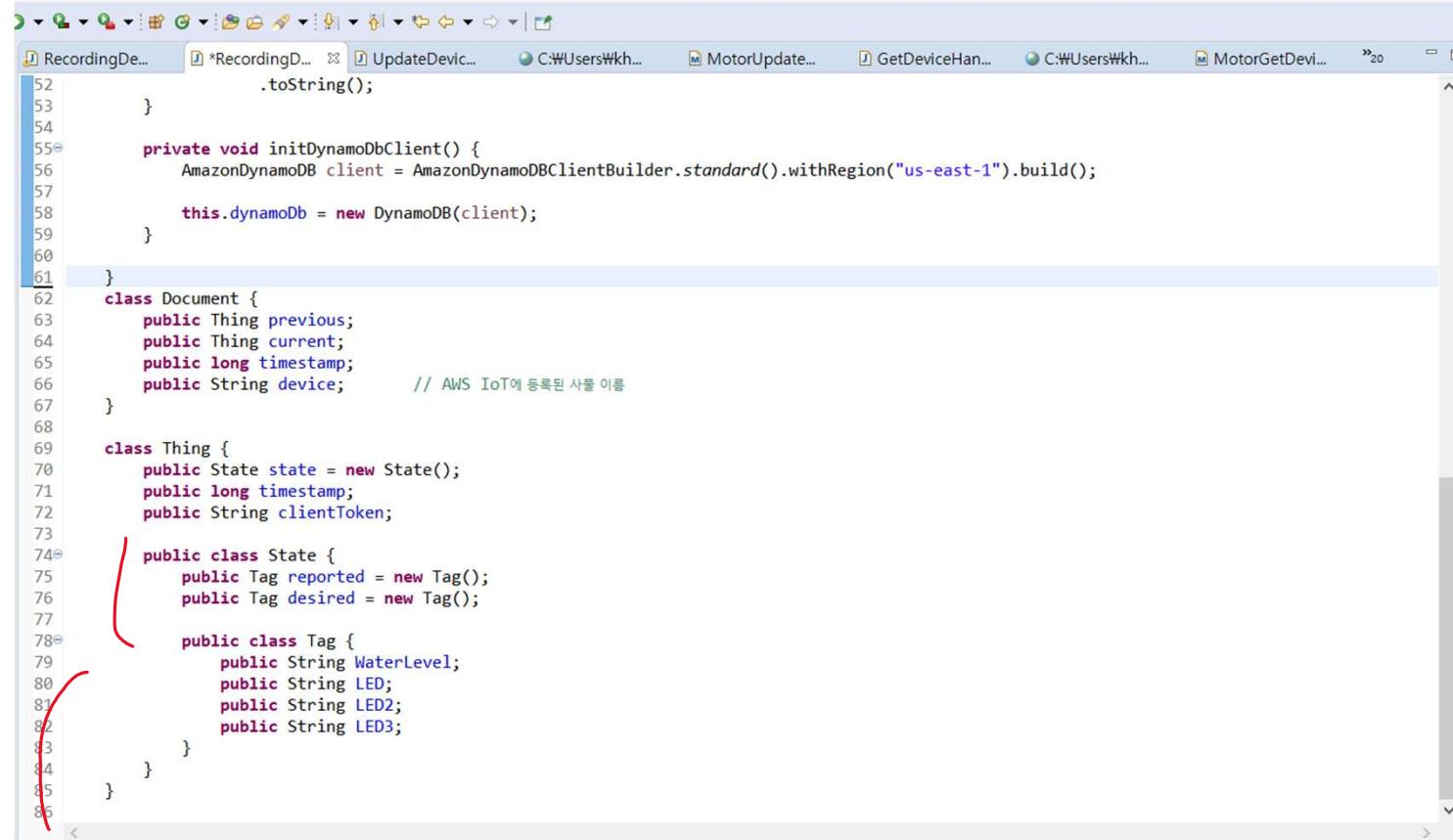


```
29     private String persistData(Document document) throws ConditionalCheckFailedException {
30
31         // Epoch Conversion Code: https://www.epochconverter.com/
32         SimpleDateFormat sdf = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
33         sdf.setTimeZone(TimeZone.getTimeZone("Asia/Seoul"));
34         String timeString = sdf.format(new java.util.Date (document.timestamp*1000));
35
36         // temperature와 LED 값이 이전상태와 동일한 경우 테이블에 저장하지 않고 종료
37         if (document.current.state.reported.WaterLevel.equals(document.previous.state.reported.WaterLevel) &&
38             document.current.state.reported.LED.equals(document.previous.state.reported.LED) &&
39             document.current.state.reported.LED2.equals(document.previous.state.reported.LED2) &&
40             document.current.state.reported.LED3.equals(document.previous.state.reported.LED3)) {
41             return null;
42         }
43
44         return this.dynamoDb.getTable(DYNAMODB_TABLE_NAME)
45             .putItem(new PutItemSpec().withItem(new Item().withPrimaryKey("deviceId", document.device)
46                 .withLong("time", document.timestamp)
47                 .withString("WaterLevel", document.current.state.reported.WaterLevel)
48                 .withString("ledRed", document.current.state.reported.LED)
49                 .withString("ledYellow", document.current.state.reported.LED2)
50                 .withString("ledGreen", document.current.state.reported.LED3)
51                 .withString("timestamp",timeString)))
52             .toString();
53     }
54
55     private void initDynamoDbClient() {
56         AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().withRegion("us-east-1").build();
57
58         this.dynamoDb = new DynamoDB(client);
59     }
60
61 }
62 class Document {
63     public Thing previous;
```

서비스 구현 과정 -DB에 올리기위한 Lambda함수

main/java/com/example/lambda/recording/RecordingDeviceInfoHandler2.java - Eclipse IDE

Object Run Window Help



```
RecordingDe... *RecordingD... UpdateDevic... C:\Users\kh... MotorUpdate... GetDeviceHan... C:\Users\kh... MotorGetDev... »20
52     }
53 }
54
55 private void initDynamoDbClient() {
56     AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().withRegion("us-east-1").build();
57
58     this.dynamoDb = new DynamoDB(client);
59 }
60
61 class Document {
62     public Thing previous;
63     public Thing current;
64     public long timestamp;
65     public String device;      // AWS IoT에 등록된 사물 이름
66 }
67
68 class Thing {
69     public State state = new State();
70     public long timestamp;
71     public String clientToken;
72
73     public class State {
74         public Tag reported = new Tag();
75         public Tag desired = new Tag();
76
77         public class Tag {
78             public String WaterLevel;
79             public String LED;
80             public String LED2;
81             public String LED3;
82         }
83     }
84 }
85
86 }
```

서비스 구현 과정-Act(규칙)생성

The screenshot shows the AWS IoT Rule Hub interface. At the top, there are three tabs: '한성 e-class' (selected), 'dynamodb', and 'AWS Open API Sar'. Below the tabs is a navigation bar with icons for back, forward, and refresh, followed by a URL: 'console.aws.amazon.com/iot/home?region=us-east-1#/rulehub'. The main content area has a sidebar on the left with categories: 모니터링, 온보딩, 관리, Greengrass, 보안, 보호, and 액트 (selected). The main panel is titled '규칙' (Rules) and contains a search bar. Two rules are listed: 'waterleveldbact' (활성) and 'waterleveldb2' (활성). Each rule card has three dots for more options.

서비스 구현 과정 -Act(규칙)생성

규칙
waterleveledb2
활성

작업 ▾

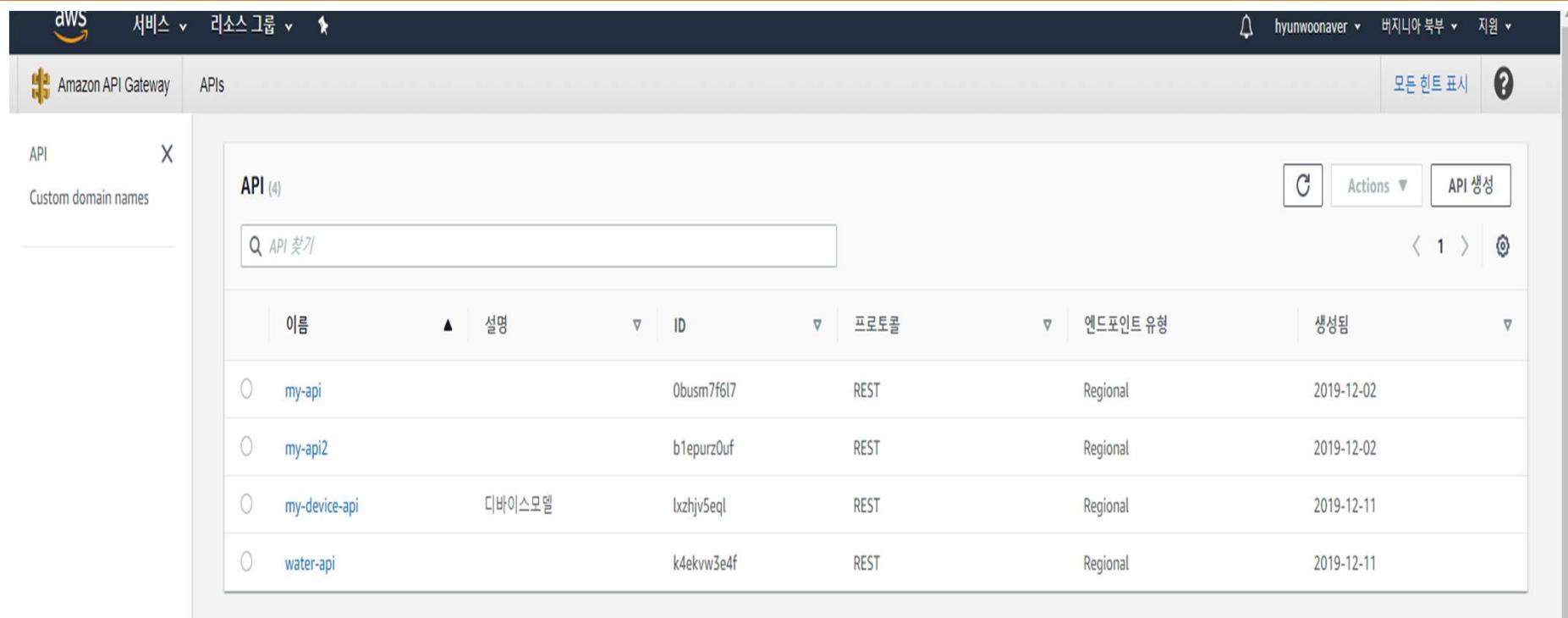
개요	설명	편집
Tags	설명이 없습니다	
규칙 쿼리 설명문	편집	
이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.		
<pre>SELECT *, 'MyMKRWiFi1010' as device FROM '\$aws/things/MyMKRWiFi1010/shadow/update/documents'</pre>		
SQL 버전 사용 2016-03-23		
작업		
작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기		
 메시지 데이터를 전달하는 Lambda 함수 호출 waterleveledb2		
제거 편집 ▾		
작업 추가		
오류 작업		

서비스 구현 과정-DB확인

The screenshot shows a database management interface with a toolbar at the top containing various tabs like '개요' (Overview), '항목' (Items), '측정치' (Measurements), '알람' (Alarms), '용량' (Capacity), '인덱스' (Indexes), '글로벌 테이블' (Global Table), '백업' (Backup), 'Contributor Insights', '트리거' (Trigger), '액세스 제어' (Access Control), and '태그' (Tags). Below the toolbar, there are two buttons: '항목 만들기' (Create Item) and '작업' (Operations). A search bar displays the query '스캔: [표] waterleveldb2: deviceld, time ^'. The main area is a table titled '[표] waterleveldb2: deviceld, time' with columns: deviceld, time, WaterLevel, ledGreen, ledRed, ledYellow, and timestamp. The table lists 12 rows of data for the device 'MyMKRWIFI1010' with timestamps from 2019-12-13 20:12:20 to 2019-12-13 20:17:17.

	deviceld	time	WaterLevel	ledGreen	ledRed	ledYellow	timestamp
	MyMKRWIFI1010	1576235540	51	ON	OFF	OFF	2019-12-13 20:12:20
	MyMKRWIFI1010	1576235737	70.50	ON	OFF	OFF	2019-12-13 20:15:37
	MyMKRWIFI1010	1576235747	71.50	ON	OFF	OFF	2019-12-13 20:15:47
	MyMKRWIFI1010	1576235757	145.00	ON	OFF	OFF	2019-12-13 20:15:57
	MyMKRWIFI1010	1576235767	81.50	ON	OFF	OFF	2019-12-13 20:16:07
	MyMKRWIFI1010	1576235777	112.50	ON	OFF	OFF	2019-12-13 20:16:17
	MyMKRWIFI1010	1576235787	152.00	ON	OFF	OFF	2019-12-13 20:16:27
	MyMKRWIFI1010	1576235797	180.00	ON	OFF	OFF	2019-12-13 20:16:37
	MyMKRWIFI1010	1576235807	63.00	ON	OFF	OFF	2019-12-13 20:16:47
	MyMKRWIFI1010	1576235817	122.00	ON	OFF	OFF	2019-12-13 20:16:57
	MyMKRWIFI1010	1576235827	111.50	ON	OFF	OFF	2019-12-13 20:17:07
	MyMKRWIFI1010	1576235837	117.00	ON	OFF	OFF	2019-12-13 20:17:17

서비스 구현 과정-API Gateway



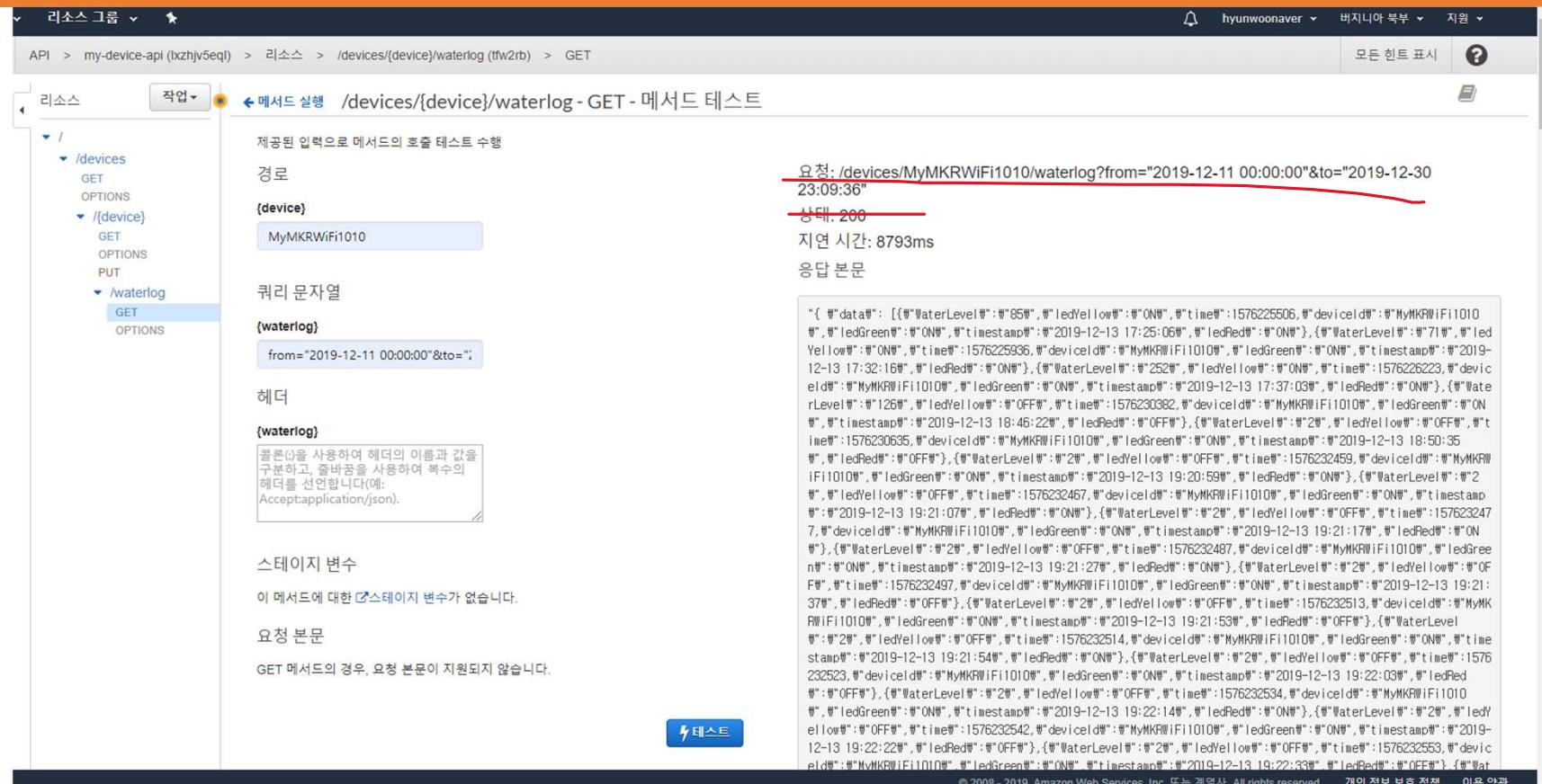
The screenshot shows the AWS API Gateway console interface. At the top, there's a navigation bar with the AWS logo, service dropdown, resource group dropdown, and user information (hyunwoonaver, 베지니아 북부, 지원). Below the navigation is a header with the AWS Lambda logo, 'Amazon API Gateway', and 'APIs' tabs, along with '모든 힌트 표시' and a help icon.

The main area is titled 'API (4)' and contains a search bar labeled 'API 찾기'. Below the search bar is a table with the following columns: 이름 (Name), 설명 (Description), ID, 프로토콜 (Protocol), 엔드포인트 유형 (Endpoint Type), and 생성됨 (Created). There are four rows in the table:

이름	설명	ID	프로토콜	엔드포인트 유형	생성됨
my-api		Obusm7f6l7	REST	Regional	2019-12-02
my-api2		b1epurz0uf	REST	Regional	2019-12-02
my-device-api	디바이스모델	lxzhjv5eql	REST	Regional	2019-12-11
water-api		k4ekvw3e4f	REST	Regional	2019-12-11

At the bottom right of the table, there are buttons for 'Actions' (with a dropdown arrow) and 'API 생성' (Create API).

서비스 구현 과정-API Gateway



서비스 구현 과정-디바이스 생성(수문제어 노드)

```
AWS_IoT_WaterDoor Motor.h arduino_secrets.h

#include "Motor.h"
Servo servo;

Motor::Motor(int pin) {
    // Use 'this->' to make the difference between the
    // 'pin' attribute of the class and the
    // local variable 'pin' created from the parameter.
    this->pin = pin;
    init();
}

void Motor::init() {
    servo.attach(pin);
    // Always try to avoid duplicate code.
    // Instead of writing digitalWrite(pin, LOW) here,
    // call the function off() which already does that
    down();
    state = Motor_DOWN;
}

void Motor::up() {
    servo.write(170);
    state = Motor_UP;
}

void Motor::down() {
    servo.write(10);
    state = Motor_DOWN;
}

byte Motor::getState() {
    return state;
}
```

```
AWS_IoT_WaterDoor Motor.cpp Motor.h arduino_secrets.h

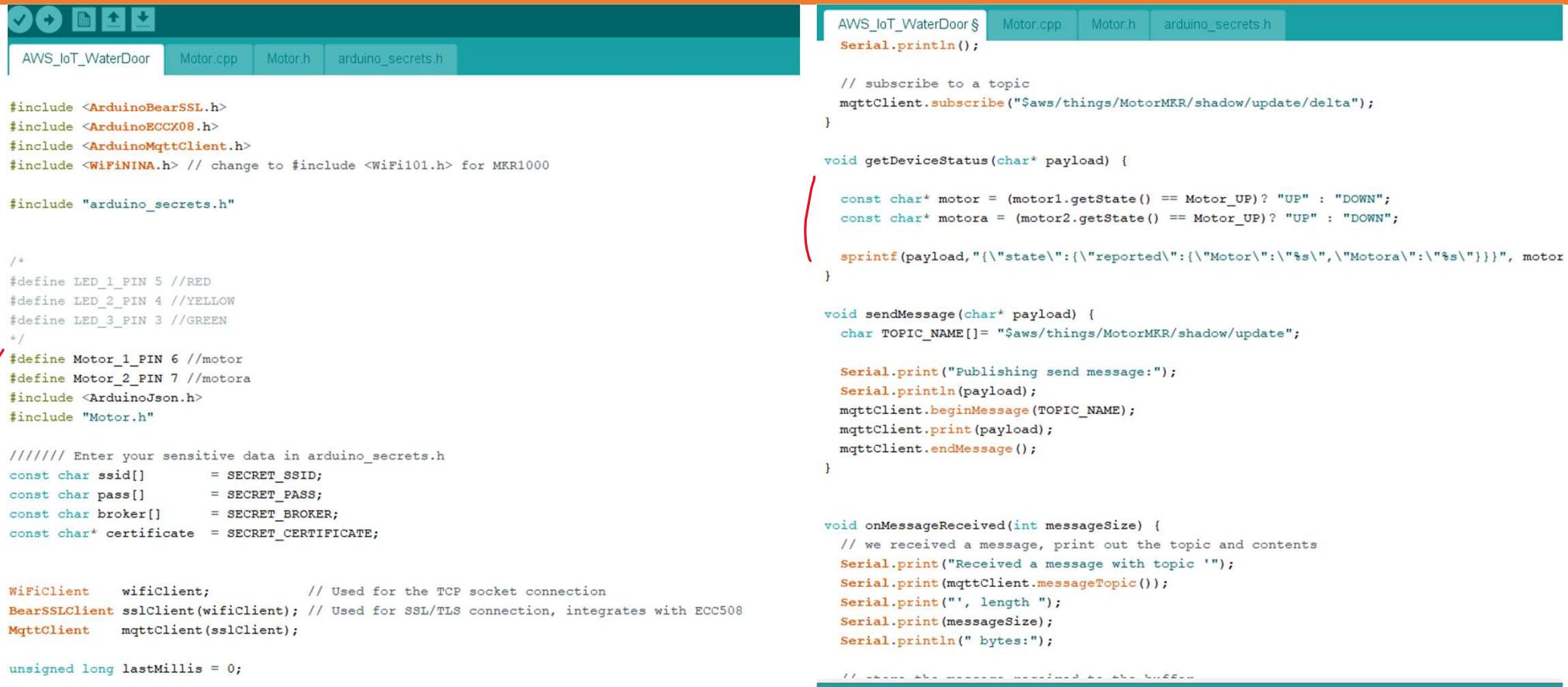
#include <Arduino.h>
#include <Servo.h>

#define Motor_DOWN 0
#define Motor_UP 1

class Motor {
private:
    int pin;
    byte state;

public:
    Motor(int pin);
    void init();
    void up();
    void down();
    byte getState();
};
```

서비스 구현 과정-디바이스 생성(수문제어 노드)



```
AWS_IoT_WaterDoor.cpp
AWS_IoT_WaterDoor.h
Motor.cpp
Motor.h
arduino_secrets.h
```

```
#include <ArduinoBearSSL.h>
#include <ArduinoECCX08.h>
#include <ArduinoMqttClient.h>
#include <WiFiNINA.h> // change to #include <WiFi101.h> for MKR1000

#include "arduino_secrets.h"

/*
#define LED_1_PIN 5 //RED
#define LED_2_PIN 4 //YELLOW
#define LED_3_PIN 3 //GREEN
*/
#define Motor_1_PIN 6 //motor
#define Motor_2_PIN 7 //motora
#include <ArduinoJson.h>
#include "Motor.h"

// Enter your sensitive data in arduino_secrets.h
const char ssid[] = SECRET_SSID;
const char pass[] = SECRET_PASS;
const char broker[] = SECRET_BROKER;
const char* certificate = SECRET_CERTIFICATE;

WiFiClient wifiClient; // Used for the TCP socket connection
BearSSLClient sslClient(wifiClient); // Used for SSL/TLS connection, integrates with ECC508
MqttClient mqttClient(sslClient);

unsigned long lastMillis = 0;
```

```
AWS_IoT_WaterDoor.h
Motor.cpp
Motor.h
arduino_secrets.h
```

```
Serial.println();

// subscribe to a topic
mqttClient.subscribe("$aws/things/MotorMKR/shadow/update/delta");
}

void getDeviceStatus(char* payload) {
    const char* motor = (motor1.getState() == Motor_UP) ? "UP" : "DOWN";
    const char* motora = (motor2.getState() == Motor_UP) ? "UP" : "DOWN";

    sprintf(payload, "{\"state\":{\"reported\":{\"Motor\":\"%s\", \"Motora\":\"%s\"}}}", motor
}

void sendMessage(char* payload) {
    char TOPIC_NAME[] = "$aws/things/MotorMKR/shadow/update";
    Serial.print("Publishing send message:");
    Serial.println(payload);
    mqttClient.beginMessage(TOPIC_NAME);
    mqttClient.print(payload);
    mqttClient.endMessage();
}

void onMessageReceived(int messageSize) {
    // we received a message, print out the topic and contents
    Serial.print("Received a message with topic ");
    Serial.print(mqttClient.messageTopic());
    Serial.print(", length ");
    Serial.print(messageSize);
    Serial.print(" bytes:");
}
```

서비스 구현 과정-디바이스 생성(수문제어 노드)

AWS_IoT_WaterDoor § Motor.cpp Motor.h arduino_secrets.h

```
DynamicJsonDocument doc(1024);
deserializeJson(doc, buffer);
JsonObject root = doc.as<JsonObject>();
JsonObject state = root["state"];
const char* motor = state["Motor"];
const char* motora = state["Motora"];
Serial.print(motor);
Serial.println(motora);
char payload[512];

if (strcmp(motor,"UP")==0) {
    motor1.up();
    sprintf(payload,"{\"state\":{\"reported\":{\"Motor\":\"%s\"}}}", "UP");
    sendMessage(payload);
}

else if (strcmp(motor,"DOWN")==0) {
    motor1.down();
    sprintf(payload,"{\"state\":{\"reported\":{\"Motor\":\"%s\"}}}", "DOWN");
    sendMessage(payload);
}

if (strcmp(motora,"UP")==0) {
    motor2.up();
    sprintf(payload,"{\"state\":{\"reported\":{\"Motora\":\"%s\"}}}", "UP");
    sendMessage(payload);
}

else if (strcmp(motora,"DOWN")==0) {
    motor2.down();
    sprintf(payload,"{\"state\":{\"reported\":{\"Motora\":\"%s\"}}}", "DOWN");
    sendMessage(payload);
}

}
```

서비스 구현 과정-디바이스 생성(수위측정 노드)

The screenshot shows the AWS IoT Device Management console. The top navigation bar includes the AWS logo, service and resource group dropdowns, and user information (hyunwoonaver, Seoul, Korean). On the left, a sidebar menu lists 'AWS IoT' (selected), '모니터링', '온보딩', '관리' (selected), '사물', '유형', '사물 그룹', and '결제 그룹'. The main content area is titled '사물' and features a search bar ('사물 검색'), a '플릿 인덱싱' button, and a '카드' dropdown. A card is displayed for the device 'MotorMKR', which has no types assigned. There are also '생성' (Create) and help icons.

서비스 구현 과정 - IAM 역할 만들기

역할 만들기		
역할 이름 ▾	신뢰할 수 있는 개체	마지막 활동 ▾
<input type="checkbox"/> AWSServiceRoleForSupport	AWS 서비스: support (서비스 연결 역할)	없음
<input type="checkbox"/> AWSServiceRoleForTrustedAdvisor	AWS 서비스: trustedadvisor (서비스 연결 역할)	없음
<input type="checkbox"/> dynamoDB	AWS 서비스: lambda	오늘
<input type="checkbox"/> hello-lambda-role	AWS 서비스: lambda	16 일
<input type="checkbox"/> iot_lambda_basic_execution	AWS 서비스: lambda	2 일
<input type="checkbox"/> iot-devices-model-all	AWS 서비스: lambda	오늘
<input type="checkbox"/> lambda_basic_execution	AWS 서비스: lambda	4 일
<input type="checkbox"/> motor-db	AWS 서비스: lambda	오늘
<input type="checkbox"/> my-function-role-cdmwmv73	AWS 서비스: lambda	13 일
<input type="checkbox"/> my-function-role-d7an3xn1	AWS 서비스: lambda	없음

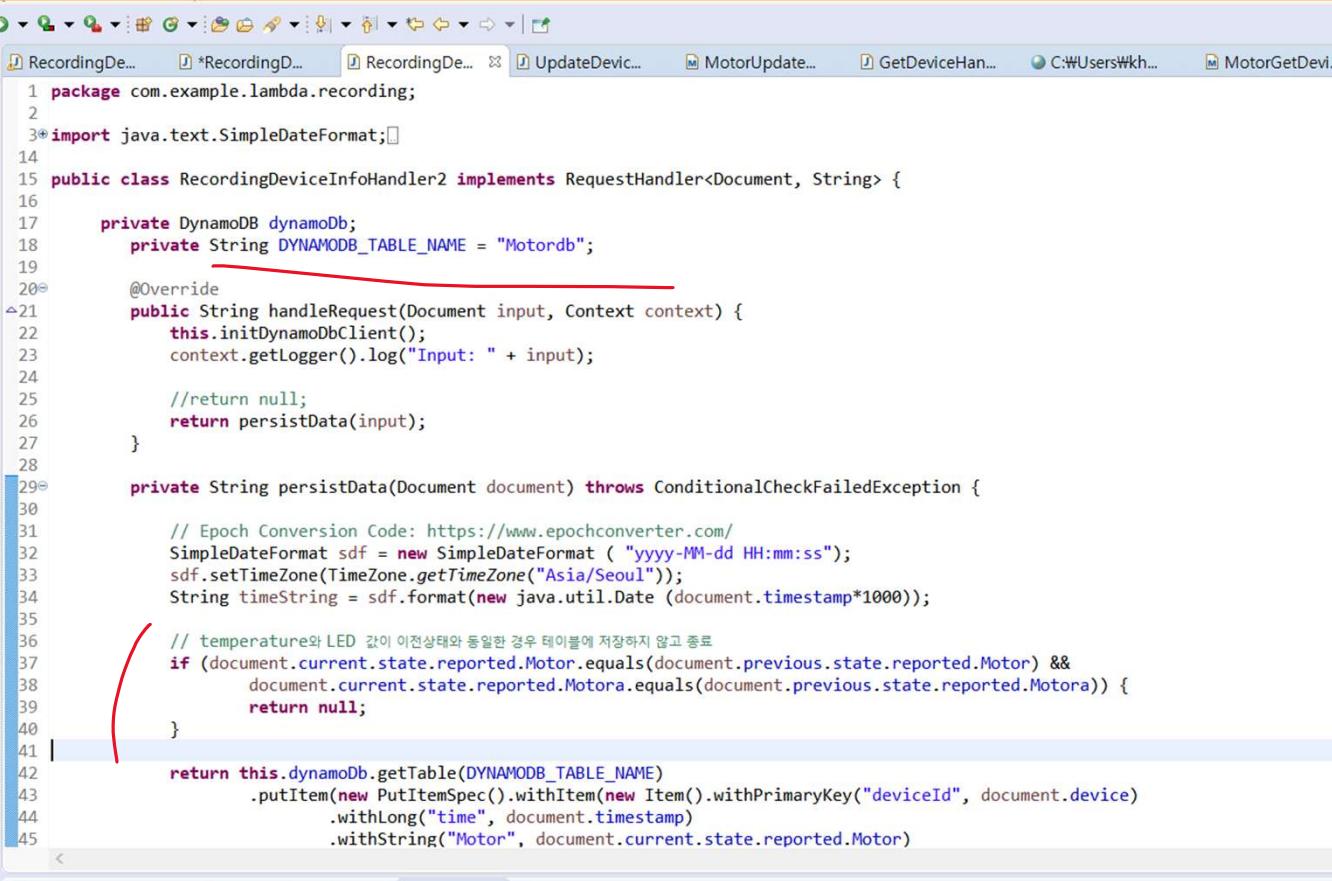
서비스 구현 과정 -DB 테이블 역할 만들기

The screenshot shows a user interface for managing database tables. At the top, there is a navigation bar with links for '서비스' (Service), '리소스 그룹' (Resource Group), and a user icon 'hyunwoonaver'. On the right side of the bar are dropdown menus for '서울' (Seoul) and '지원' (Support). Below the navigation bar is a toolbar with buttons for '테이블 만들기' (Create Table), '테이블 삭제' (Delete Table), and three other icons for refresh, settings, and help.

The main area displays a table with the following columns: '이름' (Name), '상태' (Status), '파티션 키' (Partition Key), '정렬 키' (Sort Key), '인덱스' (Index), '총 읽기 용량' (Total Read Capacity), and '총 쓰기 용량' (Total Write Capacity). There is also a search bar labeled '테이블 이름으로 필터링' (Filter by table name) and a dropdown for '테이블 그룹 선택' (Select Table Group).

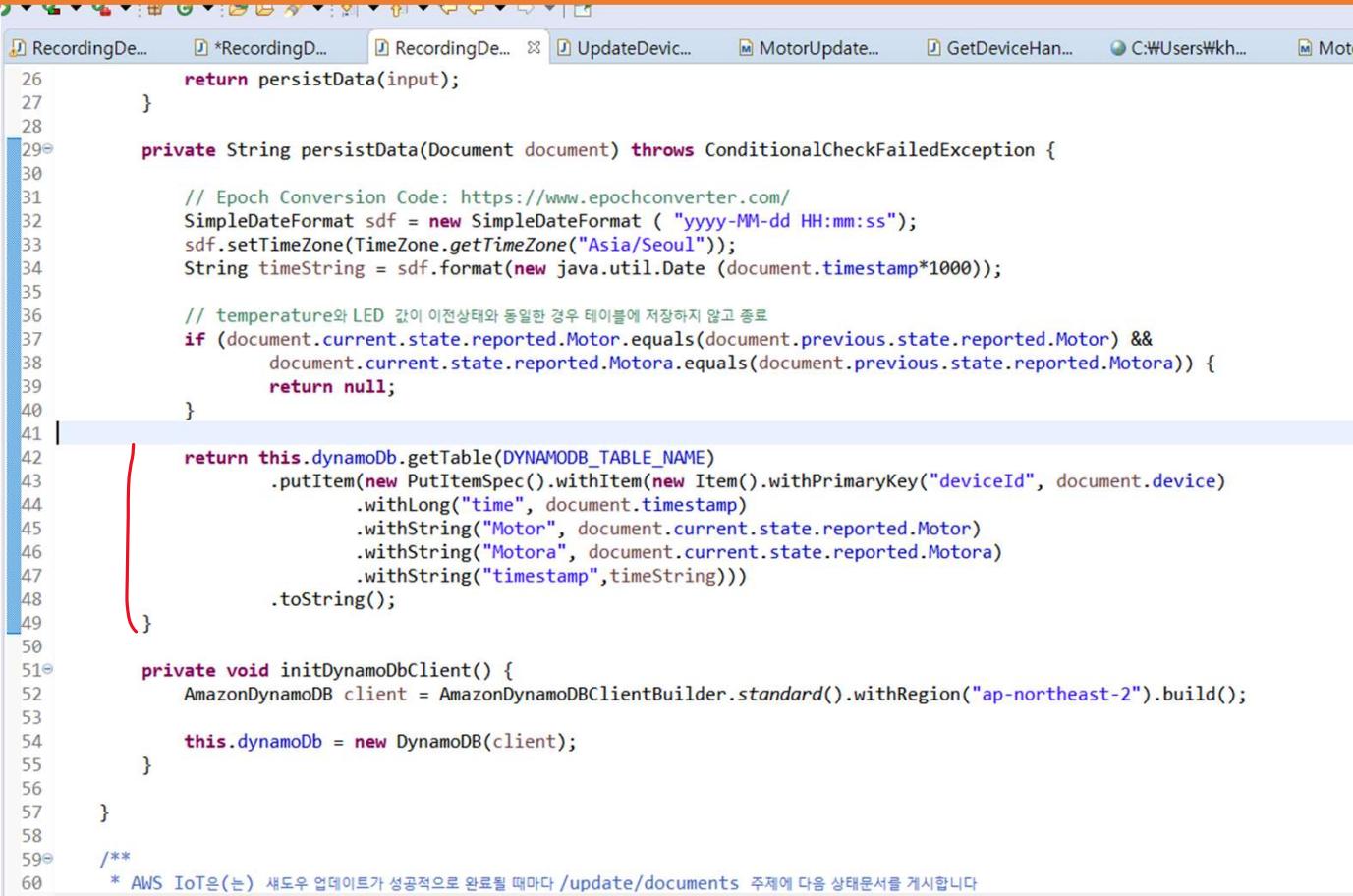
이름	상태	파티션 키	정렬 키	인덱스	총 읽기 용량	총 쓰기 용량
Motordb	활성	deviceId (문자열)	time (번호)	0	5	5

서비스 구현 과정-DB에 올리기 위한 Lambda 함수



```
1 package com.example.lambda.recording;
2
3 import java.text.SimpleDateFormat;
4
5 public class RecordingDeviceInfoHandler2 implements RequestHandler<Document, String> {
6
7     private DynamoDB dynamoDb;
8     private String DYNAMODB_TABLE_NAME = "Motordb";
9
10    @Override
11    public String handleRequest(Document input, Context context) {
12        this.initDynamoDbClient();
13        context.getLogger().log("Input: " + input);
14
15        //return null;
16        return persistData(input);
17    }
18
19    private String persistData(Document document) throws ConditionalCheckFailedException {
20
21        // Epoch Conversion Code: https://www.epochconverter.com/
22        SimpleDateFormat sdf = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
23        sdf.setTimeZone(TimeZone.getTimeZone("Asia/Seoul"));
24        String timeString = sdf.format(new java.util.Date (document.timestamp*1000));
25
26        // temperature와 LED 값이 이전상태와 동일한 경우 테이블에 저장하지 않고 종료
27        if (document.current.state.reported.Motor.equals(document.previous.state.reported.Motor) &&
28            document.current.state.reported.Motora.equals(document.previous.state.reported.Motora)) {
29            return null;
30        }
31
32        return this.dynamoDb.getTable(DYNAMODB_TABLE_NAME)
33            .putItem(new PutItemSpec().withItem(new Item().withPrimaryKey("deviceId", document.device)
34                .withLong("time", document.timestamp)
35                .withString("Motor", document.current.state.reported.Motor))
36
37
38
39
40
41
42
43
44
45
```

서비스 구현 과정-DB에 올리기 위한 Lambda 함수



```
26     return persistData(input);
27 }
28
29     private String persistData(Document document) throws ConditionalCheckFailedException {
30
31         // Epoch Conversion Code: https://www.epochconverter.com/
32         SimpleDateFormat sdf = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
33         sdf.setTimeZone(TimeZone.getTimeZone("Asia/Seoul"));
34         String timeString = sdf.format(new java.util.Date (document.timestamp*1000));
35
36         // temperature와 LED 값이 이전상태와 동일한 경우 테이블에 저장하지 않고 종료
37         if (document.current.state.reported.Motor.equals(document.previous.state.reported.Motor) &&
38             document.current.state.reported.Motora.equals(document.previous.state.reported.Motora)) {
39             return null;
40         }
41
42         return this.dynamoDb.getTable(DYNAMODB_TABLE_NAME)
43             .putItem(new PutItemSpec().withItem(new Item().withPrimaryKey("deviceId", document.device)
44                 .withLong("time", document.timestamp)
45                 .withString("Motor", document.current.state.reported.Motor)
46                 .withString("Motora", document.current.state.reported.Motora)
47                 .withString("timestamp",timeString)))
48             .toString();
49     }
50
51     private void initDynamoDbClient() {
52         AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().withRegion("ap-northeast-2").build();
53
54         this.dynamoDb = new DynamoDB(client);
55     }
56
57 }
58
59 /**
60 * AWS IoT은(는) 새도우 업데이트가 성공적으로 완료될 때마다 /update/documents 주제에 다음 상태문서를 게시합니다
61 }
```

서비스 구현 과정-DB에 올리기 위한 Lambda 함수



```
57     }
58
59     /**
60      * AWS IoT은(는) 새도우 업데이트가 성공적으로 완료될 때마다 /update/documents 주제에 다음 상태문서를 게시합니다
61      * JSON 형식의 상태문서는 2개의 기본 노드를 포함합니다. previous 및 current.
62      * previous 노드에는 업데이트가 수행되기 전의 전체 새도우 문서의 내용이 포함되고,
63      * current에는 업데이트가 성공적으로 적용된 후의 전체 새도우 문서가 포함됩니다.
64      * 새도우가 처음 업데이트(생성)되면 previous 노드에는 null이 포함됩니다.
65      *
66      * timestamp는 상태문서가 생성된 시간 정보이고,
67      * device는 상태문서에 포함된 값은 아니고, IoT규칙을 통해서 Lambda함수로 전달된 값이다.
68      * 이 값을 해당 규칙과 관련된 사물이름을 나타낸다.
69      */
70     class Document {
71         public Thing previous;
72         public Thing current;
73         public long timestamp;
74         public String device;           // AWS IoT에 등록된 사물 이름
75     }
76
77     class Thing {
78         public State state = new State();
79         public long timestamp;
80         public String clientToken;
81
82         public class State {
83             public Tag reported = new Tag();
84             public Tag desired = new Tag();
85
86             public class Tag {
87                 public String Motor;
88                 public String Motora;
89             }
90         }
91     }

```

서비스 구현 과정 -Act(규칙)생성

The screenshot shows the AWS IoT Rules interface. On the left, there is a sidebar with the AWS IoT logo and links to Monitoring, Billing, Management, Greengrass, and Security. The main area is titled "규칙" (Rules). It features a search bar labeled "검색 범위" with a magnifying glass icon. Below the search bar is a list containing one item: "motordbact" with the status "활성" (Active) and three vertical dots indicating more options.

서비스 구현 과정 -Act(규칙)생성

규칙
motordbact
활성

작업 ▾

개요	설명	편집
Tags	설명이 없습니다.	
규칙 쿼리 설명문	이 규칙을 사용하여 처리하고자 하는 메시지의 소스입니다.	편집
<pre>SELECT *, 'MyMKRWIFI1010' as device FROM '\$aws/things/MotorMKR/shadow/update/documents'</pre>		
SQL 버전 사용 2016-03-23		
작업		
작업은 규칙이 트리거되면 이루어지는 것입니다. 자세히 알아보기		
 motordb	메시지 데이터를 전달하는 Lambda 함수 호출	제거 편집 ▾

서비스 구현 과정-DB확인

Motordb 닫기

개요 항목 측정치 알람 용량 인덱스 글로벌 테이블 백업 트리거 액세스 제어 태그

항목 만들기 작업

스캔: [표] Motordb: deviceld, time ^ 항목 1~88개를 보는 중

스캔 [표] Motordb: deviceld, time + 필터 추가 검색 시작

	deviceld ⓘ	time	Motor	Motora	timestamp
	MyMKRWiFi1010	1576306723	DOWN	DOWN	2019-12-14 15:58:43
	MyMKRWiFi1010	1576307676	UP	DOWN	2019-12-14 16:14:36
	MyMKRWiFi1010	1576307806	DOWN	UP	2019-12-14 16:16:46
	MyMKRWiFi1010	1576309797	UP	DOWN	2019-12-14 16:49:57
	MyMKRWiFi1010	1576313033	UP	UP	2019-12-14 17:43:53
	MyMKRWiFi1010	1576314959	DOWN	DOWN	2019-12-14 18:15:59
	MyMKRWiFi1010	1576314960	UP	UP	2019-12-14 18:16:00

서비스 구현 과정-API Gateway

리소스 그룹 ★

API > motor-api (eblw5xtiu4) > 리소스 > /devices/{device} (6oqbr8) > GET

작업 모든 힌트 표시 ?

리소스 작업 ◀ 메서드 실행 /devices/{device} - GET - 메서드 테스트

제공된 입력으로 메서드의 호출 테스트 수행

경로

{device} 요청: /devices/MotorMKR?from="2019-12-11 00:00:00"&to="2019-12-30 23:09:36"

상태: 200

지연 시간: 7877ms

응답 본문

```
{"state": {"desired": {"Motor": "UP"}, "Motor": "DOWN"}, "reported": {"Motor": "UP", "Motor": "DOWN"}, "metadata": {"desired": {"Motor": {"timestamp": 1576408094}, "Motor": {"timestamp": 1576408094}}, "reported": {"Motor": {"timestamp": 1576408149}, "Motor": {"timestamp": 1576408149}}, "version": 4568, "timestamp": 1576419188}}
```

쿼리 문자열

{device} 응답 헤더

from="2019-12-11 00:00:00"&to="

헤더

{device}

콜론()을 사용하여 헤더의 이름과 값을
구분하고, 줄바꿈을 사용하여 복수의
헤더를 선언합니다(예:
Accept:application/json).

응답 헤더

```
{"Access-Control-Allow-Origin": "*", "X-Amzn-Trace-Id": "Root=1-5df63f6c-c204fad68e1c03016da7a4ab;Sampled=0", "Content-Type": "application/json"}
```

로그

Execution log for request ed9339cf-f562-43d3-935d-5748f314a84b
Sun Dec 15 14:13:00 UTC 2019 : Starting execution for request: ed9339cf-f562-43d3-935d-5748f314a84b

스테이지 변수

이 메서드에 대한 스테이지 변수가 없습니다.

© 2008 - 2019, Amazon Web Services, Inc. 또는 계열사. All rights reserved. 개인 정보 보호 정책 이용 약관

서비스 구현 과정-Motor 상태변경을 위한 Lambda함수

The screenshot shows a Java project structure on the left and a code editor on the right. The project structure includes a package named com.amazonaws.lambda.demo containing an UpdateDeviceHandler.java file and a resources folder. Below this are test and resources folders, Maven Dependencies, JRE System Library, and source code folders. The code editor displays the pom.xml file with several dependency blocks highlighted in blue. The dependencies listed are for the aws-java-sdk-iot and aws-lambda-java-events libraries.

```
<dependency>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-iot</artifactId>
</dependency>

<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-events</artifactId>
    <version>1.3.0</version>
</dependency>
```

서비스 구현 과정-Motor 상태변경을 위한 Lambda함수

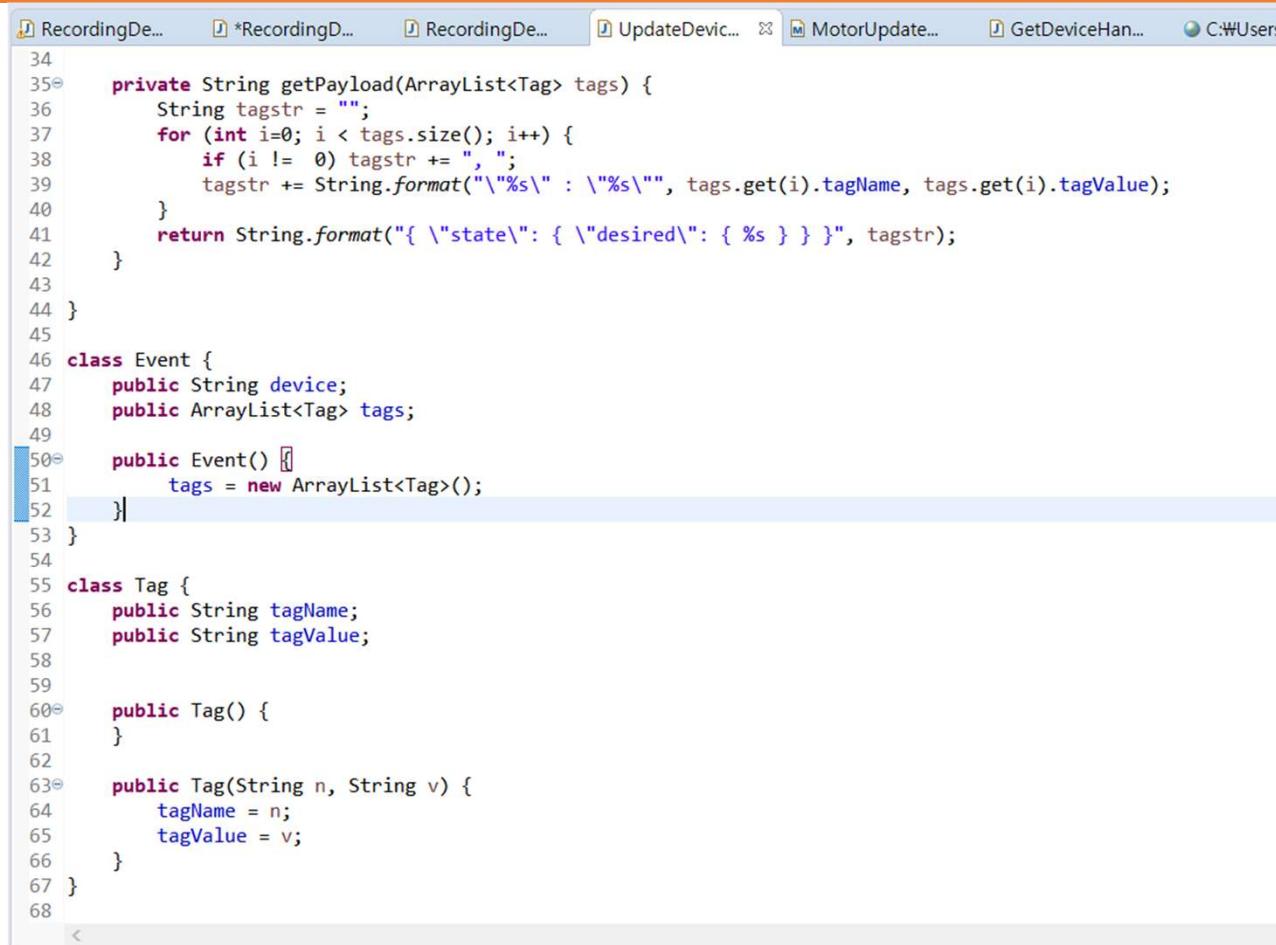
n/java/com/amazonaws/lambda/demo/UpdateDeviceHandler.java - Eclipse IDE

File Run Window Help

RecordingDe... *RecordingD... RecordingDe... UpdateDevice... MotorUpdate... GetDeviceHan... C:\Users\kh...

```
12
13 public class UpdateDeviceHandler implements RequestHandler<Event, String> {
14
15@    @Override
16    public String handleRequest(Event event, Context context) {
17        context.getLogger().log("Input: " + event);
18
19        AWSIoTData iotData = AWSIoTDataClientBuilder.standard().build();
20
21        String payload = getPayload(event.tags);
22
23        UpdateThingShadowRequest updateThingShadowRequest =
24            new UpdateThingShadowRequest()
25                .withThingName(event.device)
26                .withPayload(ByteBuffer.wrap(payload.getBytes()));
27
28        UpdateThingShadowResult result = iotData.updateThingShadow(updateThingShadowRequest);
29        byte[] bytes = new byte[result.getPayload().remaining()];
30        result.getPayload().get(bytes);
31        String resultString = new String(bytes);
32        return resultString;
33    }
34
35@    private String getPayload(ArrayList<Tag> tags) {
36        String tagstr = "";
37        for (int i=0; i < tags.size(); i++) {
38            if (i != 0) tagstr += ",";
39            tagstr += String.format("\'%s\' : \'%s\'", tags.get(i).tagName, tags.get(i).tagValue);
40        }
41        return String.format("{ \"state\": { \"desired\": { %s } } }", tagstr);
42    }
43
44}
45
46 class Event {
```

서비스 구현 과정-Motor 상태변경을 위한 Lambda함수



The screenshot shows a Java code editor with several tabs at the top: RecordingDe..., *RecordingD..., RecordingDe..., UpdateDevic..., MotorUpdate..., GetDeviceHan..., and C:\Users\... . The code itself is as follows:

```
34
35     private String getPayload(ArrayList<Tag> tags) {
36         String tagstr = "";
37         for (int i=0; i < tags.size(); i++) {
38             if (i != 0) tagstr += ", ";
39             tagstr += String.format("\'%s\' : \'%s\'", tags.get(i).tagName, tags.get(i).tagValue);
40         }
41         return String.format("{ \"state\": { \"desired\": { %s } } }", tagstr);
42     }
43
44 }
45
46 class Event {
47     public String device;
48     public ArrayList<Tag> tags;
49
50     public Event() {
51         tags = new ArrayList<Tag>();
52     }
53 }
54
55 class Tag {
56     public String tagName;
57     public String tagValue;
58
59     public Tag() {
60     }
61
62     public Tag(String n, String v) {
63         tagName = n;
64         tagValue = v;
65     }
66 }
67 }
68 <
```

서비스 구현 과정-API Gateway

API > motor-api (eblw5xtlu4) > 리소스 > /devices/{device} (6oqbr8) > PUT

모든 힌트 표시 ?

리소스 작업 ▾

◀ 메서드 실행 /devices/{device} - PUT - 메서드 테스트

제공된 입력으로 메서드의 출력 테스트 수행

경로

{device}

MotorMKR

쿼리 문자열

{device}

param1=value1¶m2=value2

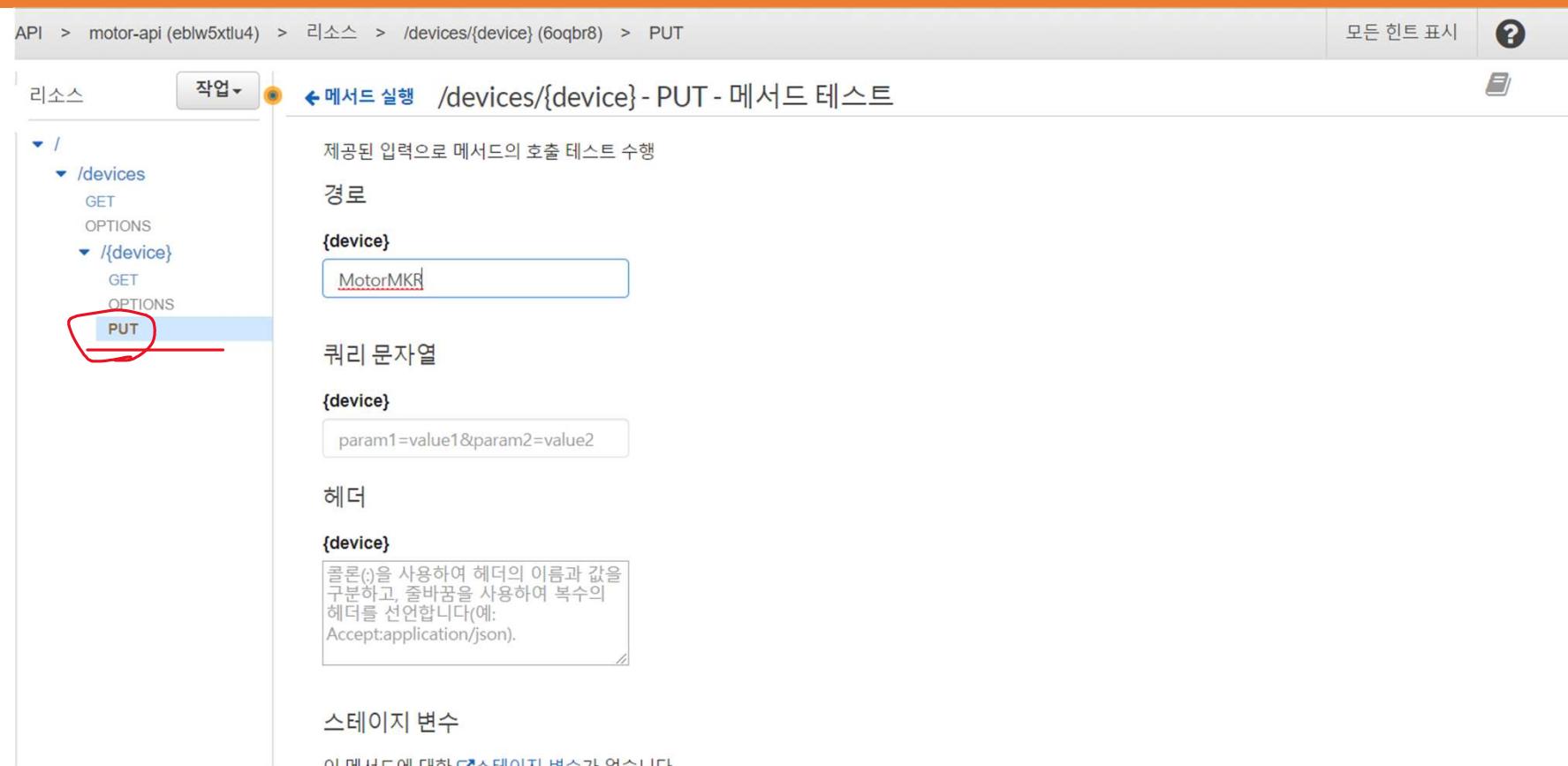
헤더

{device}

콜론()을 사용하여 헤더의 이름과 값을
구분하고, 줄바꿈을 사용하여 복수의
헤더를 선언합니다(예:
Accept:application/json).

스테이지 변수

이 메서드에 대해 스케일링 범스가 없습니다.



서비스 구현 과정-API Gateway

스테이지 변수

이 메서드에 대한 스테이지 변수가 없습니다.

요청 본문

```
1 {  
2   "tags" : []  
3   {  
4     "tagName": "Motor",  
5     "tagValue": "UP"  
6   },  
7   {  
8     "tagName": "Motora",  
9     "tagValue": "DOWN"  
10    }  
11  ]  
12 }
```

⚡ 테스트

서비스 구현 과정-API Gateway

The screenshot shows the AWS API Gateway test interface for a PUT request to the path `/devices/{device}`. The request body contains the value `MotorMKR`. The response status is 200, and the execution time is 6872ms. The response body is a JSON object representing a state transition with a timestamp of 1576419736.

리소스 그룹 > API > motor-api (eblw5xtlu4) > 리소스 > /devices/{device} (6oqbr8) > PUT

작업 > 리소스 < 메서드 실행 /devices/{device} - PUT - 메서드 테스트

제공된 입력으로 메서드의 호출 테스트 수행

경로

{device}

요청: /devices/MotorMKR

상태: 200

지연 시간: 6872ms

응답 본문

```
{"state": {"desired": {"Motor": "UP", "MotorA": "DOWN"}, "metadata": {"desired": {"Motor": {"timestamp": 1576419736}, "MotorA": {"timestamp": 1576419736}}}, "version": 4569, "timestamp": 1576419736}
```

응답 헤더

```
{"Access-Control-Allow-Origin": "*", "X-Amzn-Trace-Id": "Root=1-5df64191-aa4da09a56a732ac8a8def58;Sampled=0", "Content-Type": "application/json"}
```

로그

```
Execution log for request a9b71c08-cd51-4272-a4c1-7c37639c342d
Sun Dec 15 14:22:09 UTC 2019 : Starting execution for request: a9b71c08-cd51-4272-a4c1-7c37639c342d
Sun Dec 15 14:22:09 UTC 2019 : HTTP Method: PUT, Resource Path: /devices/MotorMKR
```

서비스 구현 과정-디바이스 상태 확인

COM7

```
|  
Received a message with topic '$aws/things/MotorMKR/shadow/update/delta', length 110 bytes:  
{"version":5316,"timestamp":1576470039,"state":{"Motor":"DOWN"},"metadata":{"Motor":{"timestamp":1576470038}}}  
  
DOWN  
Publishing send message: {"state": {"reported": {"Motor": "DOWN"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}  
Publishing send message: {"state": {"reported": {"Motor": "DOWN", "Motora": "UP"}}}
```



서비스 구현 과정- 웹구축

```
index.html      list_devices.html      controller_devices.js      list_devices.js      motor_devices.html      project.html
1 // API 시작
2 function UP() {
3
4     invokeAPI2();
5     invokeAPImotor();
6 }
7
8 function DOWN() {
9
10    invokeAPI3();
11    invokeAPImotor();
12 }
13
14 var invokeAPI2 = function(){
15
16 // 디바이스 조회 URI
17 // prod 스테이지 편집기의 맨 위에 있는 "호출 URL/devices"로 대체해야 함
18 var API_URI = 'https://eblw5xtlu4.execute-api.ap-northeast-2.amazonaws.com/prod/devices/MotorMKR';
19 var param = {
20
21     "tags" : [
22         {
23             "tagName": "Motor",
24             "tagValue": "DOWN"
25         },
26         {
27             "tagName": "Motora",
28             "tagValue": "UP"
29         }
30     ]
31 };
32 };
33 $.ajax(API_URI, {
34     method: 'PUT',
35     contentType: "application/json",
36
37     data : JSON.stringify(param),
38
39     success: function (data, status, xhr) {

```

서비스 구현 과정- 웹구축

```
16 // 디바이스 조회 URI
17 // prod 스테이지 편집기의 맨 위에 있는 "호출 URL/devices"로 대체해야 함
18 var API_URI = 'https://eblw5xtlu4.execute-api.ap-northeast-2.amazonaws.com/prod/devices/MotorMKR';
19 var param = {
20
21     "tags" : [
22         {
23             "tagName": "Motor",
24             "tagValue": "DOWN"
25         },
26         {
27             "tagName": "Motora",
28             "tagValue": "UP"
29         }
30     ]
31 };
32 $.ajax(API_URI, {
33     method: 'PUT',
34     contentType: "application/json",
35
36     data : JSON.stringify(param),
37
38     success: function (data, status, xhr) {
39         console.log(data);
40     },
41     error: function(xhr,status,e){
42         // document.getElementById("result").innerHTML="Error";
43         alert("error");
44     }
45 });
46 });
47 };
48
49 var invokeAPI3 = function(){
50
51 // 디바이스 조회 URI
52 // prod 스테이지 편집기의 맨 위에 있는 "호출 URL/devices"로 대체해야 함
53 var API_URI = 'https://eblw5xtlu4.execute-api.ap-northeast-2.amazonaws.com/prod/devices/MotorMKR';
```

서비스 구현 과정-웹구축

```
43         // document.getElementById("result").innerHTML="Error";
44         alert("error");
45     });
46 };
47 };
48
49 var invokeAPI3 = function(){
50
51 // 디바이스 조회 URI
52 // prod 스테이지 편집기의 맨 위에 있는 "호출 URL/devices"로 대체해야 함
53 var API_URI = 'https://eblw5xtlu4.execute-api.ap-northeast-2.amazonaws.com/prod/devices/MotorMKR';
54 var param = {
55
56     "tags" : [
57         {
58             "tagName": "Motor",
59             "tagValue": "UP"
60         },
61         {
62             "tagName": "Motora",
63             "tagValue": "DOWN"
64         }
65     ]
66 };
67 $.ajax(API_URI, {
68     method: 'PUT',
69     contentType: "application/json",
70
71     data : JSON.stringify(param),
72
73     success: function (data, status, xhr) {
74         console.log(data);
75     },
76     error: function(xhr,status,e){
77         // document.getElementById("result").innerHTML="Error";
78         alert("error");
79     }
80 });
81 };
82 };
```

서비스 구현 과정-API Gateway

한성 e-class | api-gateway-3.2 | DynamoDB · AWS Console | AWS Open API Sample

파일 | C:/Users/khw11/Desktop/project/list_devices.html

My AWS API

나의 디바이스 데이터 조회 [조회]

수	위	RED	YELLOW	GREEN	발생 시간
85	ON	ON	ON	ON	2019-12-13 17:25:06
71	ON	ON	ON	ON	2019-12-13 17:32:16
252	ON	ON	ON	ON	2019-12-13 17:37:03
126	OFF	OFF	ON	ON	2019-12-13 18:46:22
2	OFF	OFF	ON	ON	2019-12-13 18:50:35
2	ON	OFF	ON	ON	2019-12-13 19:20:59
2	ON	OFF	ON	ON	2019-12-13 19:21:07
2	ON	OFF	ON	ON	2019-12-13 19:21:17
2	ON	OFF	ON	ON	2019-12-13 19:21:27
2	OFF	OFF	ON	ON	2019-12-13 19:21:37
2	OFF	OFF	ON	ON	2019-12-13 19:21:53
2	ON	OFF	ON	ON	2019-12-13 19:21:54
2	OFF	OFF	ON	ON	2019-12-13 19:22:03
2	ON	OFF	ON	ON	2019-12-13 19:22:14
2	OFF	OFF	ON	ON	2019-12-13 19:22:22
2	OFF	OFF	ON	ON	2019-12-13 19:22:33
2	ON	OFF	ON	ON	2019-12-13 19:22:34
2	ON	OFF	ON	ON	2019-12-13 19:22:43
2	OFF	OFF	ON	ON	2019-12-13 19:23:15
2	ON	OFF	ON	ON	2019-12-13 19:23:16
2	OFF	OFF	ON	ON	2019-12-13 19:23:25
2	ON	OFF	ON	ON	2019-12-13 19:23:26
2	ON	OFF	ON	ON	2019-12-13 19:23:35

MY AWS Device Control

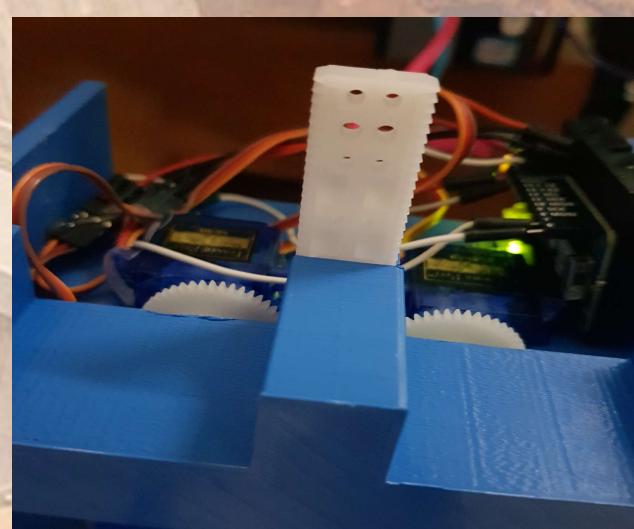
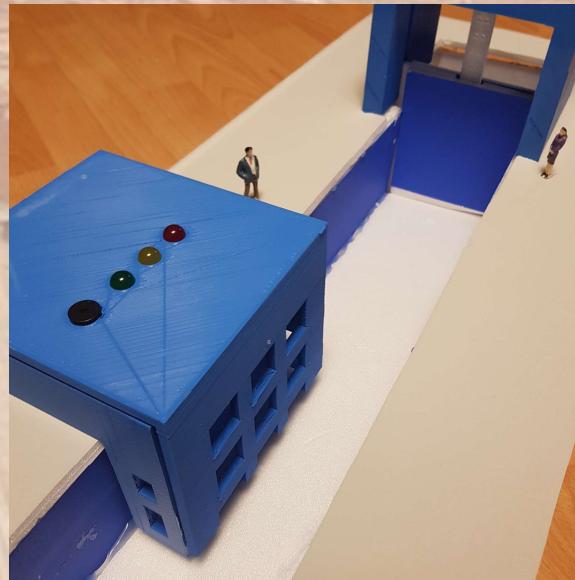
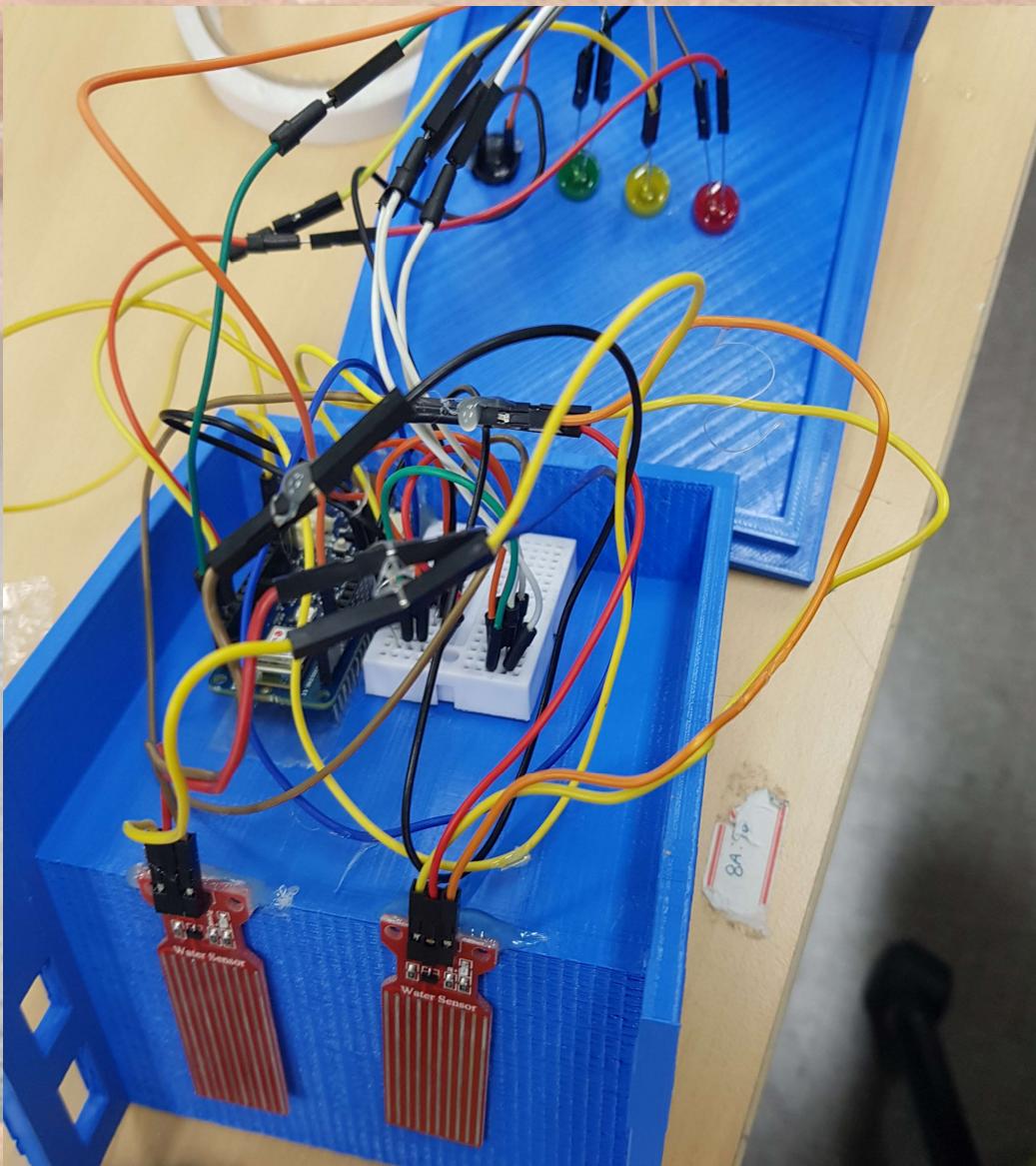
나의 디바이스 제어 UP DOWN

No Data

강 수심

전체 보기

오전 12:56
2019-12-16



시연 영상

https://www.youtube.com/watch?v=pWMmUJvj6_E

<https://github.com/Yubyungchan/byung>