

2024년 ICT이노베이션스퀘어 확산사업

충북 인공지능 교육

디지털전환을 위한 AI 기반 제조·공정 업무자동화 [센서와 IoT 기술]

- 1 AWS IoT Core 연결
- 2 데이터 수집 및 저장
- 3 Device Shadow

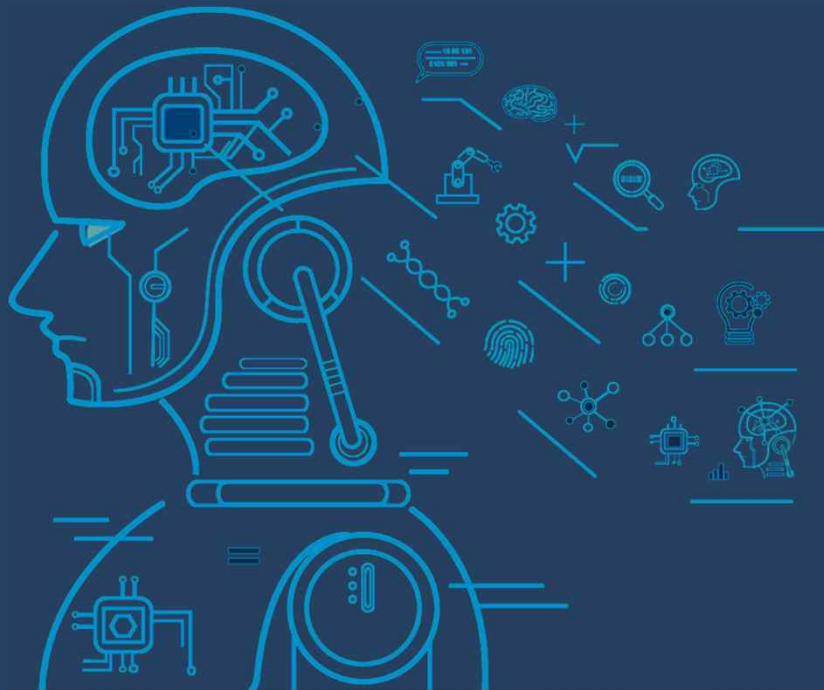
CONTENTS

디지털전환을 위한 AI 기반 제조·공정 업무자동화

1 AWS IoT Core 연결

2 데이터 수집 및 저장

3 Device Shadow





AWS IoT Core

▣ AWS IoT Core란?

AWS IoT Core는 연결된 디바이스가 쉽고 안전하게 클라우드 애플리케이션 및 다른 디바이스와 상호 작용할 수 있게 해주는 관리형 클라우드 플랫폼이다.

AWS IoT Core는 수십억 개의 디바이스와 수조 건의 메시지를 지원하고, 안전하고 안정적으로 이러한 메시지를 처리하여 AWS 엔드포인트 및 다른 디바이스로 라우팅할 수 있다.

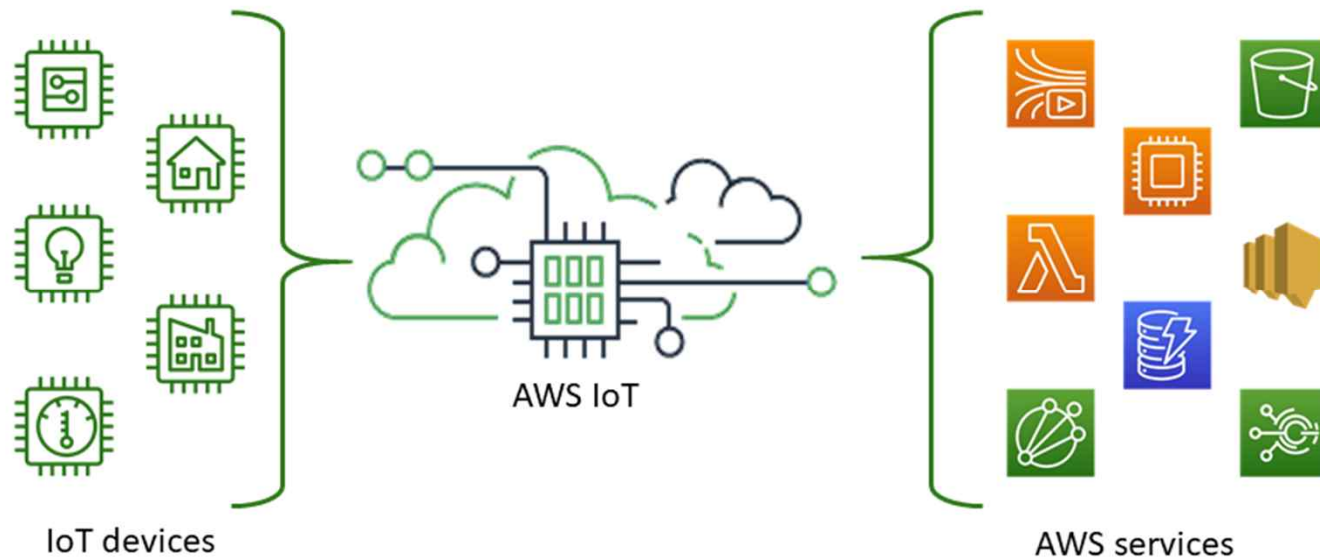
AWS IoT Core의 경우, 디바이스가 연결되어 있지 않더라도 언제든지 애플리케이션에서 모든 디바이스를 추적하고 디바이스와 통신할 수 있다.



AWS IoT Core

▣ IoT Core에서 지원하는 프로토콜

- MQTT(메시지 큐 및 원격 분석 전송)
- MQTT over WSS(웹 소켓 보안)
- HTTPS(하이퍼텍스트 전송 프로토콜 - 보안)
- LoRaWAN(장거리 광역 네트워크)

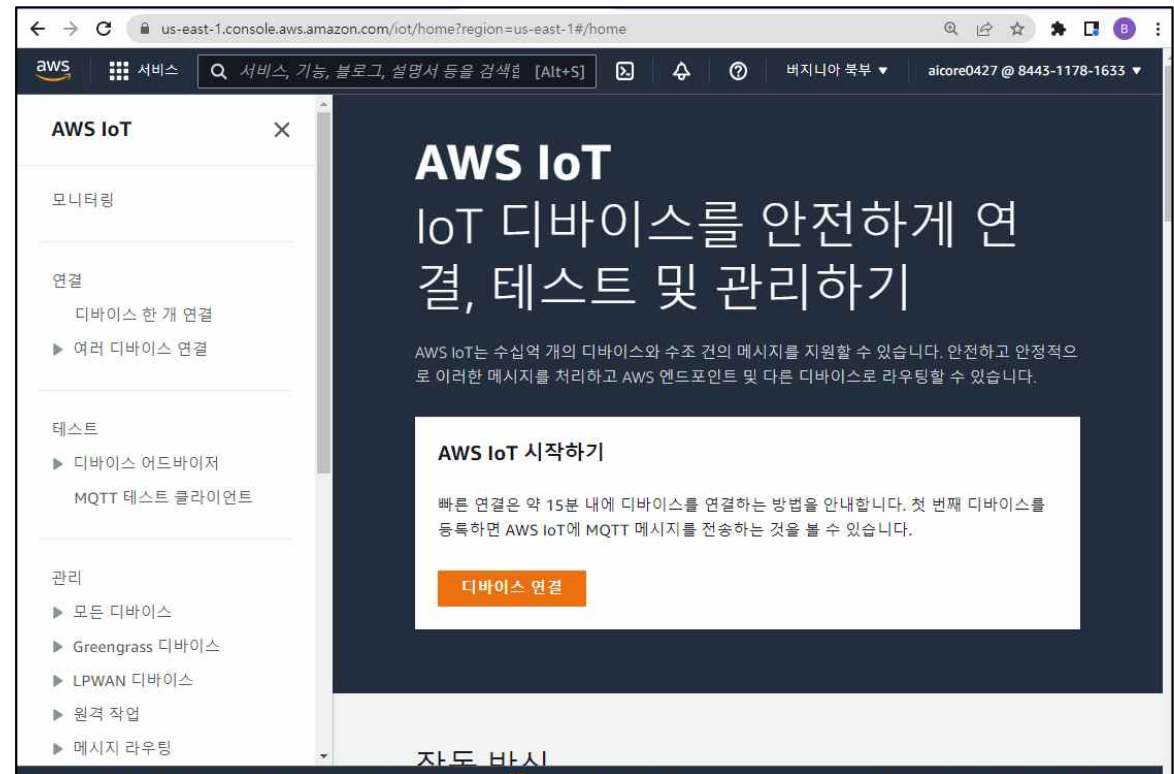
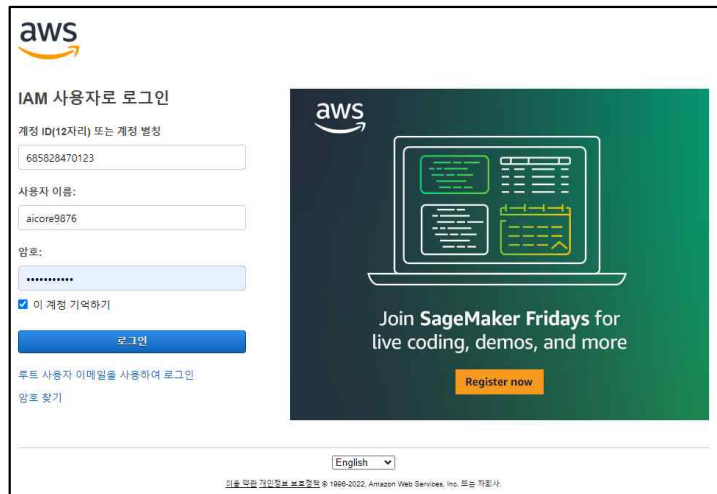




AWS IoT Core

▣ AWS IoT Core 연결

AWS 계정에 로그인하고 AWS IoT 콘솔을 연다
<https://aws.amazon.com/ko/>





AWS IoT Core

▣ AWS IoT Core 연결

AWS IoT Core 콘솔에서 [디바이스 연결]을 클릭한다

AWS IoT IoT 디바이스를 안전하게 연결, 테스트 및 관리하기

AWS IoT는 수십억 개의 디바이스와 수조 건의 메시지를 지원할 수 있습니다. 안전하고 안정적으로 이러한 메시지를 처리하고 AWS 엔드포인트 및 다른 디바이스로 라우팅할 수 있습니다.

AWS IoT 시작하기

빠른 연결은 약 15분 내에 디바이스를 연결하는 방법을 안내합니다. 첫 번째 디바이스를 등록하면 AWS IoT에 MQTT 메시지를 전송하는 것을 볼 수 있습니다.

디바이스 연결



AWS IoT Core

▣ AWS IoT Core 연결

아래와 같이 디바이스 준비 화면이 나온다

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 인증서를 사용하여 디바이스와 AWS IoT 간의 통신을 보호합니다. AWS IoT 정책은 AWS IoT 리소스에 대한 액세스를 제어합니다. 이 마법사는 디바이스에 대한 인증서와 정책을 생성합니다.

디바이스가 AWS IoT에 연결되면 정책을 통해 AWS IoT 메시지 브로커를 사용하여 MQTT 메시지를 구독 및 게시할 수 있습니다.



AWS IoT Core

▣ AWS IoT Core 연결 : 디바이스 준비 (Pi3)

1. 디바이스의 전원을 켜고 인터넷에 연결되어 있는지 확인합니다.
2. 디바이스에 파일을 로드하는 방법을 선택합니다.
파일 전송 프로토콜(FTP)을 사용하거나 USB 메모리 스틱을 사용할 수 있다.
3. 디바이스에서 명령줄 인터페이스에 액세스할 수 있는지 확인합니다.
4. 터미널 창에서 다음 명령을 입력한다(자신의 엔드 포인트 주소입력)

ping *a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com*

성공적인 ping 응답을 받으면 디바이스를 AWS IoT에 연결할 준비가 된 것이다.



AWS IoT Core

▣ AWS IoT Core 연결

[복사] 버튼을 눌러 디바이스의 터미널창에 붙여 넣어 실행시키고
ping 응답이 성공하면 [다음] 버튼을 누른다

3. 디바이스에서 명령줄 인터페이스에 액세스할 수 있는지 확인합니다.

1. IoT 디바이스에서 이 마법사를 실행하는 경우, 디바이스에서 터미널 창을 열어 명령줄 인터페이스에 액세스합니다.

2. IoT 디바이스에서 이 마법사를 실행하지 않는 경우, 이 디바이스에서 SSH 터미널 창을 열고 IoT 디바이스에 연결합니다.

4. 터미널 창에서 다음 명령을 입력합니다.

```
ping a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com
```

 복사

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

취소

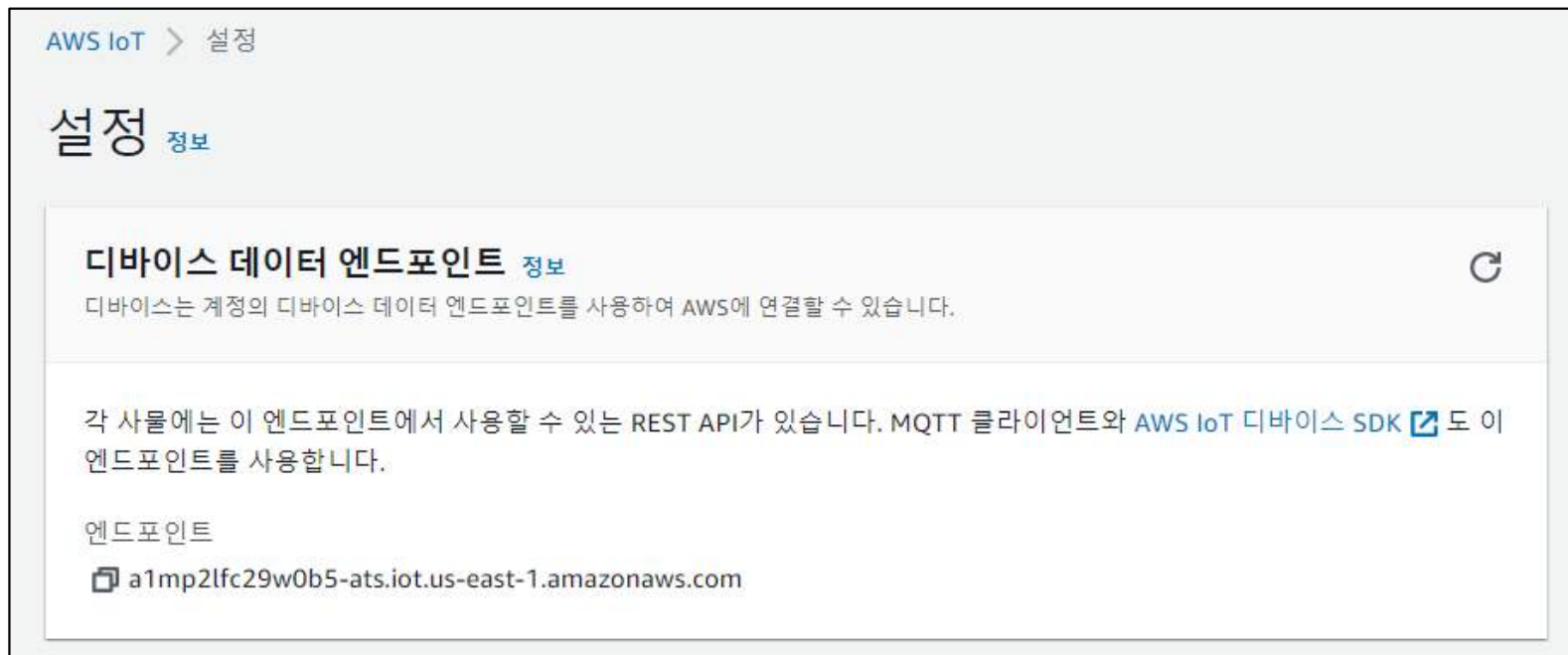
다음



AWS IoT Core

▣ AWS IoT Core 연결

디바이스 엔드포인트 주소는 AWS IoT 콘솔의 “설정”에서 확인해 볼 수 있다





AWS IoT Core

▣ AWS IoT Core 연결

“새 사물 생성”을 선택하고 사물 이름으로 “RaspberryPi”를 입력해주고 하단의 [다음]을 누른다

사물 속성

☒ 새 사물 생성

☐ 기존 사물 선택

사물 이름

RaspberryPi

문자, 숫자, 하이픈, 콜론 또는 밑줄만 포함하는 고유한 이름을 입력합니다. 사물 이름에는 공백을 포함할 수 없습니다.

취소

이전

다음



AWS IoT Core

▣ AWS IoT Core 연결

디바이스 플랫폼 운영 체제

AWS에 연결할 디바이스에 설치된 운영 체제입니다.

☒ Linux/macOS

Linux 버전: any

macOS 버전: 10.13+

☐ Windows

버전 10

AWS IoT 디바이스 SDK

디바이스가 지원하는 언어로 된 디바이스 SDK를 선택합니다.

☐ Node.js

버전 10+

Node.js 및 npm 설치 필요

☒ Python

버전 3.6+

Python 및 Git 설치 필요

☐ Java

버전 8

Java JDK, Maven 및 Git 설치 필요

디바이스 플랫폼 운영체제는
“Linux/macOS” 를 선택하고

AWS IoT 디바이스 SDK는
“Python”으로 선택해준다

하단의 [다음] 버튼을 누른다

취소

이전

다음



AWS IoT Core

▣ AWS IoT Core 연결

[연결 키트 다운로드] 버튼을 눌러 파일을 다운로드 받는다
“connect_device_package.zip” 압축파일이 다운 받아진다


연결 키트

인증서 RaspberryPi.cert.pem	프라이빗 키 RaspberryPi.private.key	AWS IoT 디바이스 SDK Python
메시지 전송 및 수신 스크립트 start.sh	정책 RaspberryPi-Policy 정책 보기	

다운로드

디바이스의 브라우저에서 이를 실행하는 경우 연결 키트를 다운로드하면 브라우저의 다운로드 폴더에 다운로드됩니다.

디바이스의 브라우저에서 이를 실행하지 않는 경우 1단계에서 디바이스를 준비할 때 테스트한 방법을 사용하여 브라우저의 다운로드 폴더에서 디바이스로 연결 키트를 전송해야 합니다.

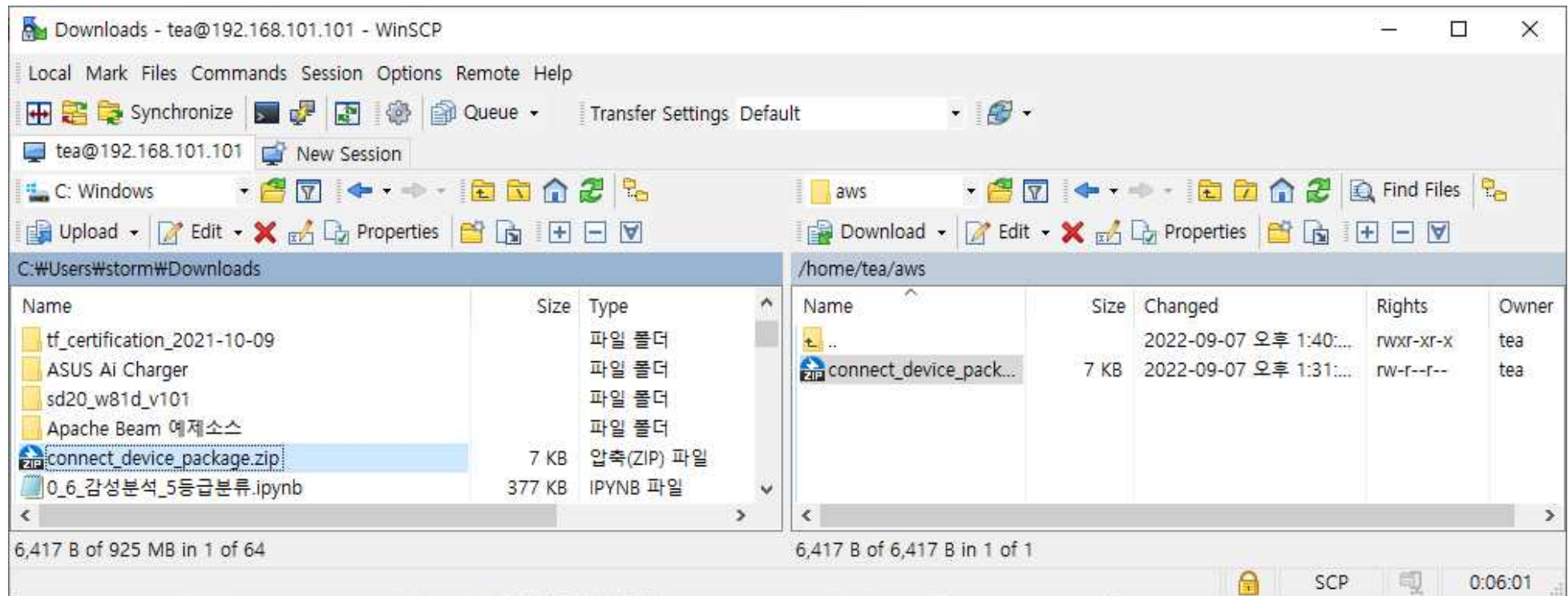
 [연결 키트 다운로드](#)



AWS IoT Core

▣ AWS IoT Core 연결

다운로드한 압축파일을 WinSCP를 사용하여 pi3보드로 전송한다
(/home/tea/아래에 'aws' 디렉토리를 하나 만들고 여기에 복사한다)



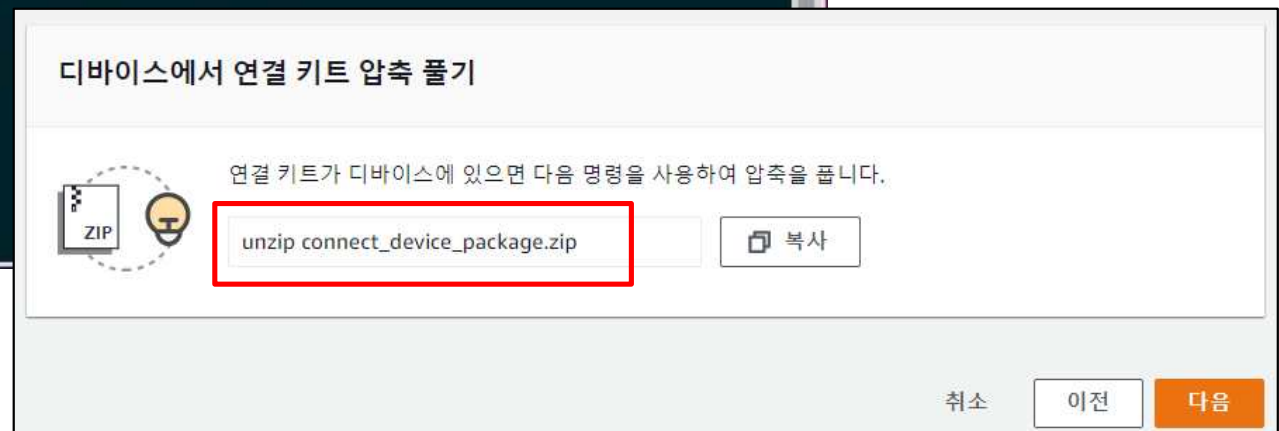


AWS IoT Core

▣ AWS IoT Core 연결

[복사] 버튼을 눌러 압축 해제 명령을 복사하여 디바이스의 터미널에서 aws경로로 이동한 뒤 붙여 넣어 실행시키고 [다음] 버튼을 누른다

```
tea@planx:~/aws
tea@planx ~$ ls
Desktop  Downloads  Pictures  Templates  aws      works
Documents Music      Public    Videos    projects
tea@planx ~$ cd aws
tea@planx ~/aws$ unzip connect_device_package.zip
Archive: connect_device_package.zip
  extracting: RaspberryPi.cert.pem
  extracting: RaspberryPi.public.key
  extracting: RaspberryPi.private.key
  extracting: RaspberryPi-Policy
  extracting: start.sh
tea@planx ~/aws$
```





AWS IoT Core

▣ AWS IoT Core 연결

“연결키트 실행”에서 아래 두 개의 명령을 차례로 복사하여 디바이스의 터미널창에 붙여 넣어 실행시킨다(약 25분 소요)

연결 키트 실행 정보

디바이스의 메시지를 표시하는 방법

1단계: 실행 권한 추가
디바이스에서 터미널 창을 시작하여 명령을 복사하고 붙여넣어 실행 권한을 추가합니다.

`chmod +x start.sh` 복사

2단계: 시작 스크립트 실행
디바이스에서 명령을 복사하여 터미널 창에 붙여넣고 시작 스크립트를 실행합니다.

`./start.sh` 복사

주의!

마지막 명령에서 'sudo'를 빠뜨리지 말고 반드시 사용한다

```
chmod +x start.sh
```

```
sudo ./start.sh
```



AWS IoT Core

▣ AWS IoT Core 연결

약 25분 정도 기다리면 설치가 완료된다

```
192.168.101.101 (planx) - VNC Viewer
sudo ./start.sh

tea@planx ~/aws$ chmod +x start.sh
tea@planx ~/aws$ sudo ./start.sh

Downloading AWS IoT Root CA certificate from AWS...
% Total    % Received % Xferd  Average Speed   Time    Time     Current
           %             Dload  Upload    Total   Spent    Left     Speed
100 1188  100 1188    0     0 12444      0  --:--:-- --:--:-- --:--:-- 12638

Cloning the AWS SDK...
Cloning into 'aws-iot-device-sdk-python-v2'...
remote: Enumerating objects: 1489, done.
remote: Counting objects: 100% (585/585), done.
remote: Compressing objects: 100% (278/278), done.
remote: Total 1489 (delta 434), reused 395 (delta 306), pack-reused 904
Receiving objects: 100% (1489/1489), 1.82 MiB | 2.89 MiB/s, done.
Resolving deltas: 100% (892/892), done.
Checking connectivity... done.

Installing AWS SDK...
Processing ./aws-iot-device-sdk-python-v2
Collecting awscrt==0.14.5 (from awsiotsdk==1.0.0.dev0)
  Downloading https://files.pythonhosted.org/packages/46/d4/dd6f5803036249f66172fec4a
t-0.14.5.tar.gz (21.5MB)
    100% |#####| 21.5MB 13kB/s
Installing collected packages: awscrt, awsiotsdk
Running setup.py install for awscrt ... -
```




AWS IoT Core

▣ AWS IoT Core 연결

디바이스에서 설치 완료 후 자동으로 테스트 메시지를 보낸다

```
Running pub/sub sample application...
Connecting to a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'Hello World! [1]'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'Hello World! [2]'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'Hello World! [3]'
Publishing message to topic 'sdk/test/Python': Hello World! [4]
Received message from topic 'sdk/test/Python': b'Hello World! [4]'
Publishing message to topic 'sdk/test/Python': Hello World! [5]
Received message from topic 'sdk/test/Python': b'Hello World! [5]'
Publishing message to topic 'sdk/test/Python': Hello World! [6]
Received message from topic 'sdk/test/Python': b'Hello World! [6]'
Publishing message to topic 'sdk/test/Python': Hello World! [7]
Received message from topic 'sdk/test/Python': b'Hello World! [7]'
```

```
python3 aws-iot-device-sdk-python-v2/samples/pubsub.py --endpoint
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt
--cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id
basicPubSub --topic sdk/test/Python --count 0
```



AWS IoT Core

▣ AWS IoT Core 연결

AWS IoT 콘솔에서 아래와 같이 메시지가 수신 되는 것을 볼 수 있다

3단계: 이 화면으로 돌아와 디바이스의 메시지 확인
시작 스크립트를 실행한 후 이 화면으로 돌아와 디바이스와 AWS IoT 간의 메시지를 확인합니다. 디바이스의 메시지가 다음 목록에 나타납니다.

구독	sdk/test/Python	일시 중지	지우기
sdk/test/Pyt...	<div>▼ sdk/test/Python</div> <div>September 07, 2022, 15:57:26 (UTC+0900)</div> <div>"Hello World! [3]"</div>		
	<div>▼ sdk/test/Python</div> <div>September 07, 2022, 15:57:25 (UTC+0900)</div> <div>"Hello World! [2]"</div>		
	<div>▼ sdk/test/Python</div> <div>September 07, 2022, 15:57:25 (UTC+0900)</div> <div>"Hello World! [1]"</div>		

취소 이전 계속

[계속]을 누르고 하단의
[사물 보기]를 누르면
사물 세부 정보를 볼 수 있다

모든 사물 보기

사물 보기



AWS IoT Core

▣ AWS IoT Core 연결

“테스트”에서 “MQTT 테스트 클라이언트”를 클릭하고 주제 필터에 “sdk/test/Python”을 입력하고 [구독] 버튼을 클릭한다

AWS IoT > MQTT test client

MQTT 테스트 클라이언트 정보

MQTT 테스트 클라이언트를 사용하여 AWS 계정에서 전달되는 MQTT 메시지를 모니터링할 수 있습니다. 디바이스는 주제를 게시하여 디바이스와 앱에 변경 사항 및 이벤트를 알립니다. MQTT 테스트 클라이언트를 사용하여 MQTT 메시지 주제를

주제 구독

주제 게시

주제 필터 정보
주제 필터는 구독할 주제를 설명합니다. 주제 필터에는 MQTT 와일드카드 문자가 포함될 수 있습니다.

▶ 추가 구성

구독



AWS IoT Core

▣ AWS IoT Core 연결

연결키트 다운로드 파일 `connect_device_package.zip` 을 압축 풀어서 `start.sh`파일의 마지막 줄의 명령을 복사하여 디바이스의 터미널에서 실행시킨다(CTRL-C를 눌러 중단 시킬 수 있다)

```
python3 aws-iot-device-sdk-python-v2/samples/pubsub.py --endpoint a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt --cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id basicPubSub --topic sdk/test/Python --count 0
```

```
tea@planx ~/aws python3 aws-iot-device-sdk-python-v2/samples/pubsub.py --endpoint a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt --cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id basicPubSub --topic sdk/test/Python --count 0
Connecting to a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'"Hello World! [1]"'
```



AWS IoT Core

▣ AWS IoT Core 연결

아래와 같이 메시지가 수신 되는 것을 볼 수 있다

구독

sdk/test/Python

일시 중지

지우기

내보내기

편집

▼ sdk/test/Python September 07, 2022, 16:13:39 (UTC+0900)

"Hello World! [3]"

▼ sdk/test/Python September 07, 2022, 16:13:38 (UTC+0900)

"Hello World! [2]"

▼ sdk/test/Python September 07, 2022, 16:13:37 (UTC+0900)

"Hello World! [1]"

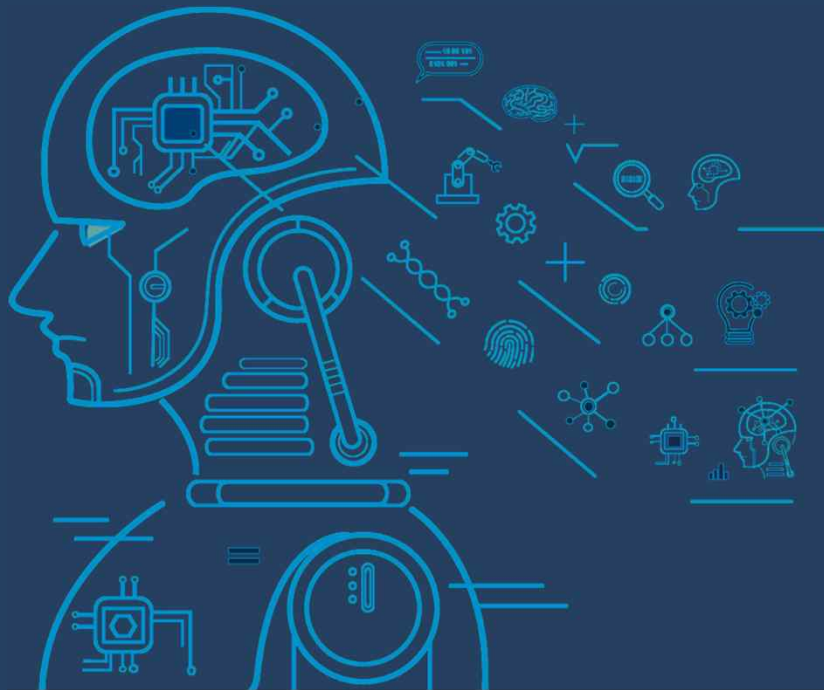
CONTENTS

디지털전환을 위한 AI 기반 제조·공정 업무자동화

1 AWS IoT Core 연결

2 데이터 수집 및 저장

3 Device Shadow

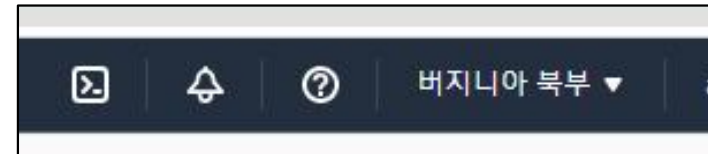




데이터 수집 및 저장

▣ DynamoDB 생성

AWS 콘솔의 우측상단의 CloudShell 아이콘을 눌러 AWS CloudShell을 실행시킨다



CloudShell에서 아래 DynamoDB 테이블 생성 명령을 실행시킨다

```
aws dynamodb create-table ₩
```

```
--table-name lotDB ₩
```

```
--attribute-definitions AttributeName=SensorId,AttributeType=N
```

```
AttributeName=TimeStamp,AttributeType=N ₩
```

```
--key-schema AttributeName=SensorId,KeyType=HASH
```

```
AttributeName=TimeStamp,KeyType=RANGE ₩
```

```
--provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
```



데이터 수집 및 저장

▣ DynamoDB 생성

```
us-east-1
aws help or aws <command> help or aws <command> --cli-auto-prompt
[cloudshell-user@ip-10-0-50-243 ~]$ aws dynamodb create-table \
> --table-name IotDB \
> --attribute-definitions AttributeName=SensorId,AttributeType=N AttributeName=TimeStamp,AttributeType=N \
> --key-schema AttributeName=SensorId,KeyType=HASH AttributeName=TimeStamp,KeyType=RANGE \
> --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1
{
  "TableDescription": {
    "AttributeDefinitions": [
      {
        "AttributeName": "SensorId",
        "AttributeType": "N"
      },
      {
        "AttributeName": "TimeStamp",
        "AttributeType": "N"
      }
    ],
    "TableName": "IotDB",
    "KeySchema": [
      {
        "AttributeName": "SensorId",
        "KeyType": "HASH"
      },
      {
        "AttributeName": "TimeStamp",
        "KeyType": "RANGE"
      }
    ],
    "TableStatus": "CREATING",
    "CreationDateTime": "2022-09-09T08:28:12.370000+00:00",
    "ProvisionedThroughput": {
      "NumberOfDecreasesToday": 0,
      "ReadCapacityUnits": 1,
      "WriteCapacityUnits": 1
    },
    "TableSizeBytes": 0,
    "ItemCount": 0,
    "TableArn": "arn:aws:dynamodb:us-east-1:844311781633:table/IotDB",
    "TableId": "f47dd11b-14ef-457d-a711-44f2330d7e3b"
  }
}
```




데이터 수집 및 저장

▣ DynamoDB 생성

AWS 콘솔의 DynamoDB로 가서 왼쪽 에서 “테이블 ”을 클릭하면
아래 와 같이 테이블 목록이 나타난다

DynamoDB > 테이블

테이블 (1) 정보

<input type="checkbox"/>	이름 ▲	상태	파티션 키	정렬 키	인덱스	읽기 용량 모드	쓰기 용량 모드	크기	테이블 클래스
<input type="checkbox"/>	lotDB	✓ 활성	SensorId (N)	TimeStamp (N)	0	프로비저닝됨 (1)	프로비저닝됨 (1)	0바이트	DynamoDB Standard



데이터 수집 및 저장

Rule 생성

AWS IoT Core 콘솔의 [메시지 라우팅]→[규칙]을 클릭하고 [규칙생성]을 클릭한다





데이터 수집 및 저장

▣ Rule 생성

규칙 이름을 “iotddb”로 입력하고 [다음] 버튼을 클릭한다

규칙 속성 지정 정보

규칙 리소스에는 MQTT 주제 스트림을 기반으로 하는 작업 목록이 포함되어 있습니다.

규칙 속성

규칙 이름

영숫자 문자열을 입력합니다. 밑줄(_) 문자는 포함할 수 있지만 공백은 포함할 수 없습니다.

규칙 설명 - 선택 사항

설명을 입력하여 다른 사용자에게 규칙에 대한 추가 세부 정보를 제공합니다.

▼ 태그 - 선택 사항

리소스와 연결된 태그가 없습니다.

태그를 1개 더 추가할 수 있습니다.



데이터 수집 및 저장

Rule 생성

SQL문에 “SELECT * FROM 'iot/sensor'”를 입력하고 [다음]을 클릭한다

SQL 문 구성 정보

단순화된 SQL 구문을 추가하여 MQTT 주제에서 수신된 메시지를 필터링하고 데이터를 다른 곳에 푸시합니다.

SQL 문

SQL 버전

규칙을 평가할 때 사용할 SQL 규칙 엔진의 버전입니다.

2016-03-23 ▼

SQL 문

SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>을(를) 사용하여 SQL 문을 입력합니다. 예: SELECT temperature FROM 'iot/topic' WHERE temperature > 50. 자세한 내용은 AWS IoT SQL 참조를 참조하세요.

1

SELECT * FROM 'iot/sensor'

취소

이전

다음



데이터 수집 및 저장

▣ Rule 생성

[규칙 작업]의 [작업 1]에서 “작업 선택”에서 DynamoDB를 선택해주고
아래와 같이 설정해준다

테이블 이름 : iotddb
파티션 키 : SensorId
파티션 키 유형 : NUMBER
파티션 키 값 : \${SensorId}
정렬 키 : TimeStamp
범위 키 유형: NUMBER
범위 키 값 : \${TimeStamp}
이 열에 메시지 데이터 쓰기 :
Payload



데이터 수집 및 저장

Rule 생성

작업 1

DynamoDB

DynamoDB 테이블에 메시지 삽입

테이블 이름 정보

lotDB

↺

보기

DynamoDB 테이블 생성

파티션 키

파티션 키(해시 키라고도 함)는 사용자가 생성한 DynamoDB 테이블의 파티션 키와 일치해야 합니다.

SensorId

파티션 키 유형

파티션 키(해시 키라고도 함) 유형은 STRING 또는 NUMBER일 수 있습니다. 기본값은 STRING입니다.

NUMBER

파티션 키 값

파티션 키(해시 키라고도 함) 값은 런타임에 데이터를 제공하는 대체 템플릿을 지원합니다.

\${SensorId}

정렬 키 - 선택 사항

정렬 키(범위 키라고도 함)는 사용자가 생성한 DynamoDB 테이블의 정렬 키와 일치해야 합니다.

TimeStamp

범위 키 유형

정렬 키(범위 키라고도 함) 유형은 STRING 또는 NUMBER일 수 있습니다. 기본값은 STRING입니다.

NUMBER

범위 키 값

정렬 키(범위 키라고도 함) 값은 런타임에 데이터를 제공하는 대체 템플릿을 지원합니다.

\${TimeStamp}

이 열에 메시지 데이터 쓰기 - 선택 사항

Payload



데이터 수집 및 저장

▣ Rule 생성

[IAM 역할]에서 [새 역할 생성]을 클릭한다

IAM 역할
엔드포인트에 대한 액세스 권한을 AWS IoT에 부여하기 위해 역할을 선택합니다.

IAM 역할 선택 ▼  보기  새 역할 생성

AWS IoT는 선택한 IAM 역할 아래에 접두사가 "aws-iot-rule"인 정책을 자동으로 생성합니다.

[역할 생성] 창에서 [역할 이름]에 "iotddb_role"을 입력하고 [생성]을 클릭한다

[다음]을 클릭하여 다음 단계로 넘어간다

역할 생성 ×

역할 이름

iotddb_role

영숫자, 하이픈 및 밑줄이 포함된 고유한 역할 이름을 입력합니다. 역할 이름에는 공백을 포함할 수 없습니다.

취소 생성



데이터 수집 및 저장

▣ Rule 생성

[검토 및 생성] 에서 내용을 검토하고 하단 [생성]을 눌러 완료 한다

검토 및 생성 정보

1단계: 규칙 속성 편집

규칙 속성

이름
iotddb

설명
-

이전 생성

규칙 (1) 정보 ↺ 환

규칙은 사물이 다른 서비스와 상호 작용할 수 있게 합니다. 규칙을 분석하고 디바이스에서 게시한 메시지를 기준으로 특정 작업을 수행합니다.

🔍 규칙 찾기

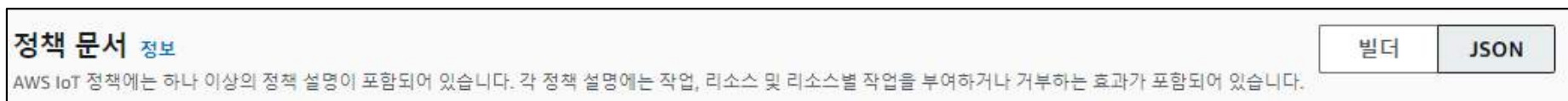
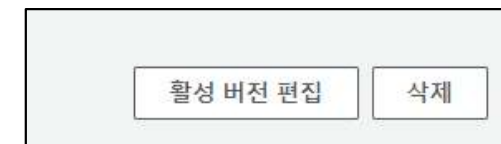
<input type="checkbox"/>	이름	▲ 상태 ▼	규칙 주제 ▼	생성 날짜
<input type="checkbox"/>	iotddb	🟢 활성	iot/sensor	September 08, 2022, 00:01:39 (UTC+0900)



데이터 수집 및 저장

▣ 정책 설정

IoT Core 콘솔의 [보안]에 있는 [정책]을 클릭하고 정책이름 목록에서 “RaspberryPi-Policy”를 클릭하고 우측 상단의 [활성 버전 편집]을 클릭한 다음 [정책 문서] 우측의 [JSON]을 클릭해준다





데이터 수집 및 저장

▣ 정책 설정

'Publish', 'Receive', 'Subscribe' Action에 대한 Resource를 설정한다.
13 Line과 23 Line 부근에서 topic_1를 **“iot/sensor”**로 변경한다. 두 군데를 변경해 준다

'Connect' Action에 대한 Resource를 설정한다. 31 Line 부근에서 sdk-java를 **“\${iot:ClientId}”**으로 변경한다. 그리고, 32라인과 33라인은 삭제하고, 31라인 제일 뒤 콤마(,)를 삭제한다

하단의 [정책 버전 상태]에 있는 [활성 정책]에서 **“편집한 버전을 이 정책의 활성 버전으로 설정”**을 체크해주고 [새 버전으로 저장]을 클릭한

활성 정책

☒ 편집한 버전을 이 정책의 활성 버전으로 설정

나중에 정책 세부 정보 페이지에서 이 설정을 변경할 수 있습니다.

새 버전으로 저장



데이터 수집 및 저장

정책 설정

정책 문서

```
9      ],
10     "Resource": [
11       "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/java",
12       "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/Python",
13       "arn:aws:iot:us-east-1:844311781633:topic/iot/sensor",
14       "arn:aws:iot:us-east-1:844311781633:topic/topic_2"
15     ]
16   },
17   {
18     "Effect": "Allow",
19     "Action": "iot:Subscribe",
20     "Resource": [
21       "arn:aws:iot:us-east-1:844311781633:topicfilter/sdk/test/java",
22       "arn:aws:iot:us-east-1:844311781633:topicfilter/sdk/test/Python",
23       "arn:aws:iot:us-east-1:844311781633:topicfilter/iot/sensor",
24       "arn:aws:iot:us-east-1:844311781633:topicfilter/topic_2"
25     ]
26   },
27   {
28     "Effect": "Allow",
29     "Action": "iot:Connect",
30     "Resource": "arn:aws:iot:us-east-1:844311781633:client/${iot:ClientId}"
31   }
32 ]
```



데이터 수집 및 저장

▣ 테스트 클라이언트

MQTT 테스트 클라이언트 정보

MQTT 테스트 클라이언트를 사용하여 AWS 계정에서 전달되는 MQTT 테스트 클라이언트를 사용하여 MQTT 메시지 주제를 구독

주제 구독 | 주제 게시

주제 필터 정보
주제 필터는 구독할 주제를 설명합니다. 주제 필터에는 MQTT 와일드

iot/sensor

▶ 추가 구성

구독

구독	iot/sensor
iot/sensor	No messages h

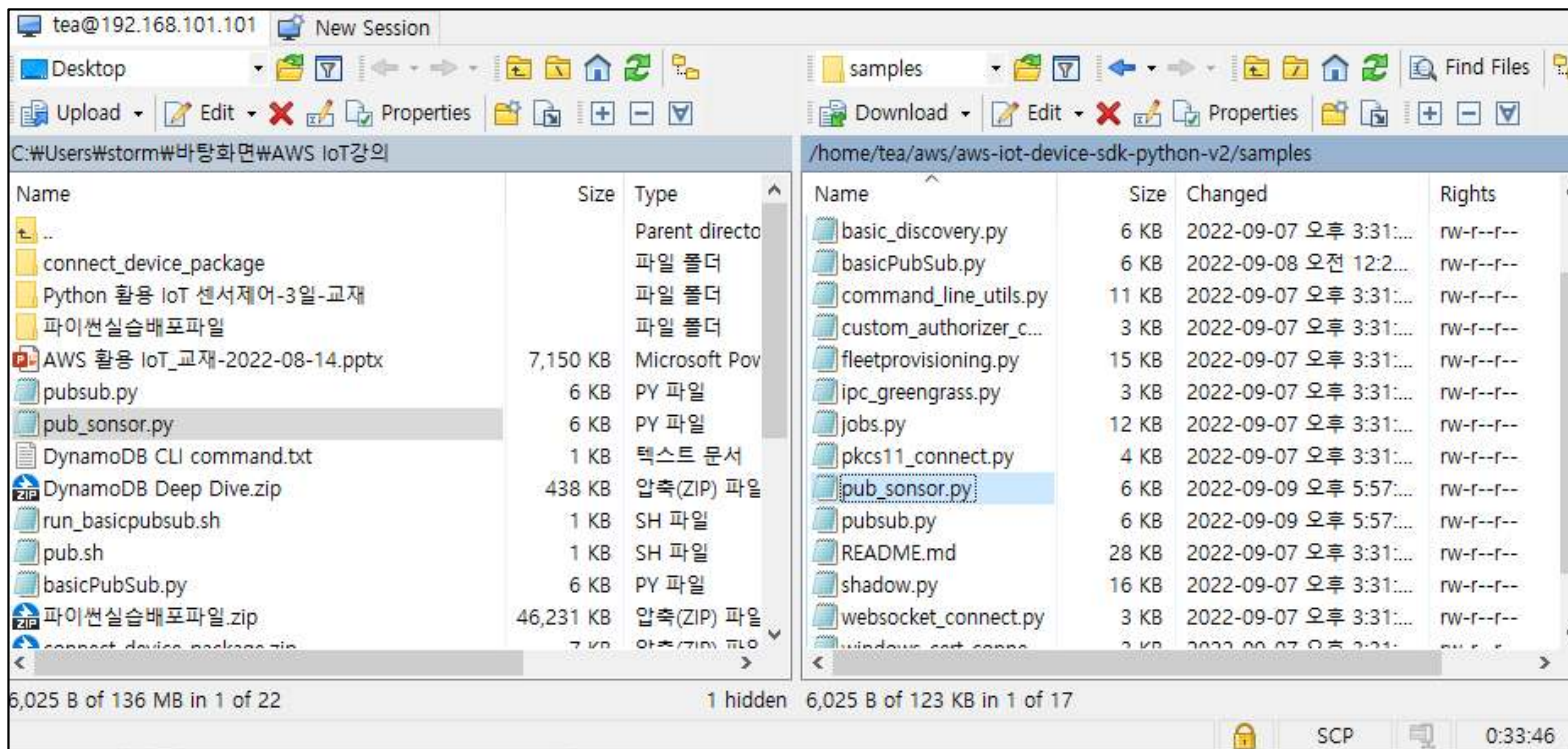
AWS IoT 콘솔의 [테스트]안에 있는
[MQTT 테스트 클라이언트]를 선택하고
[주제 구독] 탭의 [주제 필터]에 “iot/sensor”를
입력하고 [구독]버튼을 누른다



데이터 수집 및 저장

▣ 디바이스 소스 준비

WinSCP를 사용하여 배포된 pub_sensor.py 파일을 Pi3보드의 /home/tea/aws/aws-iot-device-sdk-python-v2/samples/아래에 복사한다





데이터 수집 및 저장

▣ Shell Script 준비

Pi3보드의 터미널에서 아래 명령을 실행 시킨다

```
tail -1 start.sh > pub_sensor.sh
```

```
chmod +x pub_sensor.sh
```

생성된 스크립트 파일을 확인해본다(endpoint 주소 값은 각자 다르다
AWS IoT 콘솔의 설정에서 “디바이스 데이터 엔드포인트” 확인 가능)

```
cat pub_sensor.sh
```

```
python3 aws-iot-device-sdk-python-v2/samples/pub_sensor.py --endpoint  
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt --  
cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id  
basicPubSub --topic iot/sensor --count 0% (다음 페이지의 변경 후 결과임)
```



데이터 수집 및 저장

▣ Shell Script 작성

생성된 pub_sensor.sh 파일을 편집기로 아래 두 곳을 수정해준다

[원본]

```
python3 aws-iot-device-sdk-python-v2/samples/pubsub.py --endpoint  
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt -  
-cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id  
basicPubSub --topic sdk/test/Python --count 0
```

[변경]

```
python3 aws-iot-device-sdk-python-v2/samples/pub_sensor.py --  
endpoint a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file  
root-CA.crt --cert RaspberryPi.cert.pem --key RaspberryPi.private.key --  
client_id basicPubSub --topic iot/sensor --count 0
```



데이터 수집 및 저장

▣ Publishing 하기

Pi3보드의 터미널 창에서 아래 명령을 실행하면 메시지가 publish된다
`./pub_sensor.sh`

```
tea@planx: ~/aws
}'
publish_count: 4 {"Value": 8.5, "SensorId": 0, "TimeStamp": 1662713874, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 8.5, "SensorId": 0, "TimeStamp": 1662713874, "SensorName": "UltraSonic"}'
}'
publish_count: 5 {"Value": 3.39, "SensorId": 0, "TimeStamp": 1662713875, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 3.39, "SensorId": 0, "TimeStamp": 1662713875, "SensorName": "UltraSonic"}'
}'
publish_count: 6 {"Value": 6.23, "SensorId": 0, "TimeStamp": 1662713876, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 6.23, "SensorId": 0, "TimeStamp": 1662713876, "SensorName": "UltraSonic"}'
}'
publish_count: 7 {"Value": 9.08, "SensorId": 0, "TimeStamp": 1662713877, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 9.08, "SensorId": 0, "TimeStamp": 1662713877, "SensorName": "UltraSonic"}'
}'
publish_count: 8 {"Value": 5.77, "SensorId": 0, "TimeStamp": 1662713878, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 5.77, "SensorId": 0, "TimeStamp": 1662713878, "SensorName": "UltraSonic"}'
}'
publish_count: 9 {"Value": 8.55, "SensorId": 0, "TimeStamp": 1662713879, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 8.55, "SensorId": 0, "TimeStamp": 1662713879, "SensorName": "UltraSonic"}'
}'
publish_count: 10 {"Value": 5.16, "SensorId": 0, "TimeStamp": 1662713880, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 5.16, "SensorId": 0, "TimeStamp": 1662713880, "SensorName": "UltraSonic"}'
}'
publish_count: 11 {"Value": 1.63, "SensorId": 0, "TimeStamp": 1662713881, "SensorName": "UltraSonic"}
Received message from topic 'iot/sensor': b'{"Value": 1.63, "SensorId": 0, "TimeStamp": 1662713881, "SensorName": "UltraSonic"}'
}'
```



데이터 수집 및 저장

▣ Subscribing 하기

AWS IoT 콘솔의
[MQTT 테스트 클라
이언트]에서 다음과
같은 구독된 메시지를
볼 수 있다

구독	iot/sensor	일시 중지	지우기	내보내기	편집
iot/sensor ♥ X	<div>▼ iot/sensor September 09, 2022, 17:49:10 (UTC+0900)</div> <pre>{ "Value": 9.74, "SensorId": 0, "TimeStamp": 1662713350, "SensorName": "UltraSonic" }</pre>				
	<div>▼ iot/sensor September 09, 2022, 17:49:10 (UTC+0900)</div> <pre>{ "Value": 6.83, "SensorId": 0, "TimeStamp": 1662713349, "SensorName": "UltraSonic" }</pre>				



데이터 수집 및 저장

▣ DynamoDB 확인

AWS 콘솔의 DynamoDB로 가서 IoTDB테이블로 들어가서 우측 상단 [표 항목 탐색]을 누르면 아래 와 같이 수집된 항목들이 나타난다

<input type="checkbox"/>	SensorId ▾	TimeStamp ▾	Payload
<input type="checkbox"/>	0	1662713339	{ "Value" : { "N" : "8.43" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713339" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713340	{ "Value" : { "N" : "7.88" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713340" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713341	{ "Value" : { "N" : "8.31" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713341" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713342	{ "Value" : { "N" : "6.68" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713342" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713343	{ "Value" : { "N" : "4.86" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713343" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713344	{ "Value" : { "N" : "6.05" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713344" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713345	{ "Value" : { "N" : "7.23" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713345" }, "SensorName" : { "S" : "UltraSonic" } }
<input type="checkbox"/>	0	1662713346	{ "Value" : { "N" : "9.37" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713346" }, "SensorName" : { "S" : "UltraSonic" } }



데이터 수집 및 저장

로깅 설정

AWS IoT 콘솔의 [설정]에서 로그수준을 “오류(최소 상세 수준)”이나 “디버그(최대 상세 정보 표시)”로 설정하여 로깅 된 정보를 AWS CloudWatch로 확인한다

로그 수준 정보

로그에 대해 원하는 상세 수준을 선택합니다. 오류(최소 상세 정보 표시)를 선택하면 오류만 기록되며 최소한의 정보만 표시됩니다. 디버그(최대 상세 정보 표시)를 선택하면 가장 세부적인 로그가 생성됩니다. 더 세부적인 로그를 수집하면 로깅 비용이 증가할 수 있습니다.

로그 수준

오류(최소 상세 수준) ▼

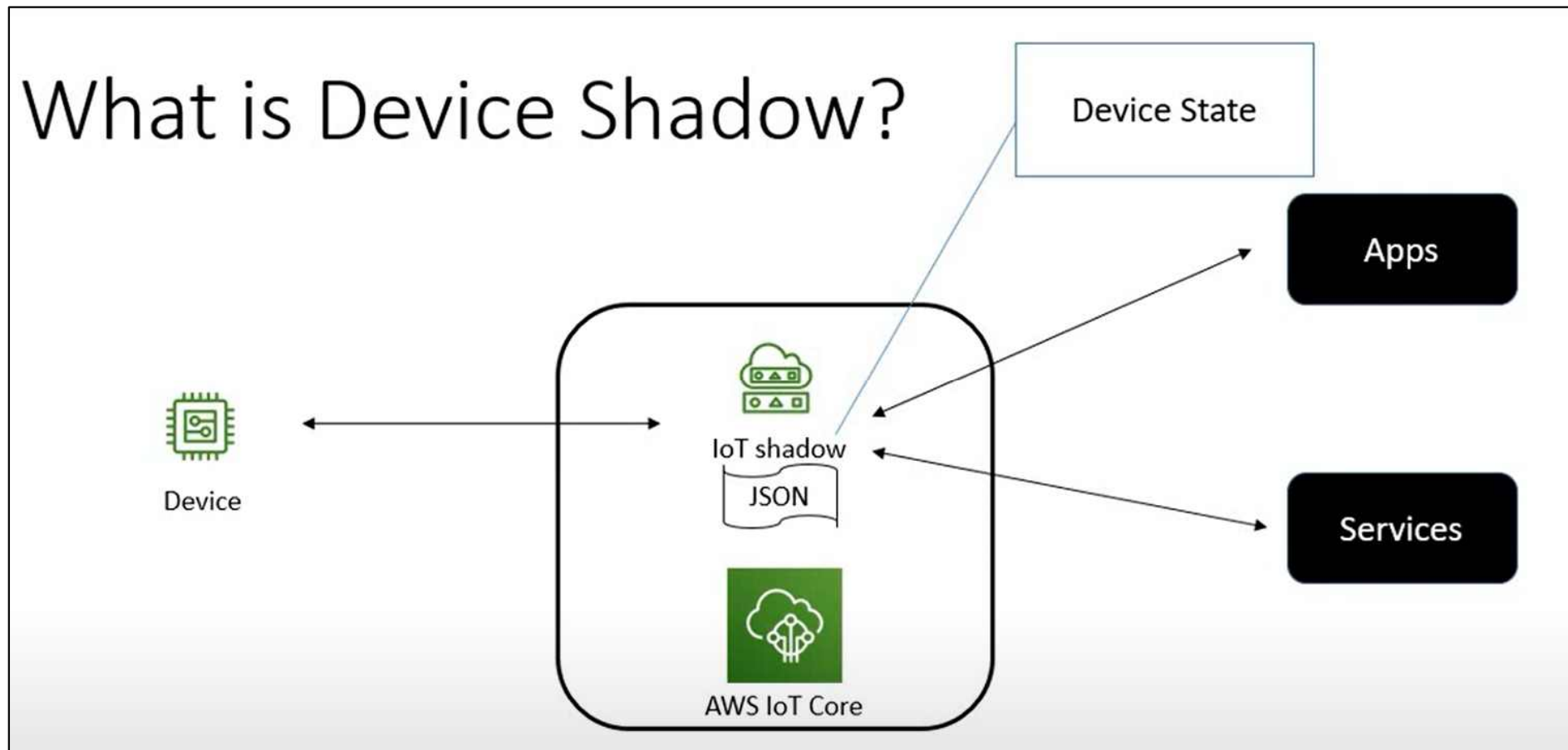
취소 업데이트

- 1 AWS IoT Core 연결
- 2 데이터 수집 및 저장
- 3 **Device Shadow**



Device Shadow 기능

▣ Device Shadow 기능





Device Shadow

▣ Shadow를 위한 정책 설정

IoT Core 콘솔의 [보안]에 있는 [정책]을 클릭하고 정책이름 목록에서 “RaspberryPi-Policy”를 클릭하고 우측 상단의 [활성 버전 편집]을 클릭한다음 [정책 문서] 우측의 [JSON]을 클릭해준다

'Publish', 'Receive', 'Subscribe' Action에 대한 Resource를 설정한다.
14 Line과 24 Line에서 topic_2를 “**\$aws/things/RaspberryPi/shadow/***”으로 변경한다. 두 군데를 변경해 준다

하단의 [정책 버전 상태]에 있는 [활성 정책]에서 “**편집한 버전을 이 정책의 활성 버전으로 설정**”을 체크해주고 [새 버전으로 저장]을 클릭한다



Device Shadow

Shadow를 위한 정책 설정

정책 문서

```
10 ▼    "Resource": [  
11        "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/java",  
12        "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/Python",  
13        "arn:aws:iot:us-east-1:844311781633:topic/iot/sensor",  
14        "arn:aws:iot:us-east-1:844311781633:topic/$aws/things/RaspberryPi/shadow/*"  
15    ],  
16    },  
17 ▼    {  
18        "Effect": "Allow",  
19        "Action": "iot:Subscribe",  
20 ▼    "Resource": [  
21        "arn:aws:iot:us-east-1:844311781633:topicfilter/sdk/test/java",  
22        "arn:aws:iot:us-east-1:844311781633:topicfilter/sdk/test/Python",  
23        "arn:aws:iot:us-east-1:844311781633:topicfilter/iot/sensor",  
24        "arn:aws:iot:us-east-1:844311781633:topicfilter/$aws/things/RaspberryPi/shadow/*"  
25    ],  
26    },  
27 ▼    {  
28        "Effect": "Allow",  
29        "Action": "iot:Connect",  
30        "Resource": "arn:aws:iot:us-east-1:844311781633:client/${iot:ClientId}"  
31    }  
32  ]  
33 }
```



Device Shadow

▣ Shell Script 준비

Pi3보드의 터미널에서 아래 명령을 실행 시킨다

```
tail -1 start.sh > run_shadow.sh
```

```
chmod +x run_shadow.sh
```

생성된 스크립트 파일을 확인해본다(endpoint 주소 값은 각자 다르다
AWS IoT 콘솔의 설정에서 “디바이스 데이터 엔드포인트” 확인 가능)

```
cat run_shadow.sh
```

```
python3 aws-iot-device-sdk-python-v2/samples/shadow.py --endpoint  
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt --  
cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id  
Raspi_Shadow --thing_name RaspberryPi (다음 페이지의 변경 후 결과임)
```



Device Shadow

▣ Shell Script 준비

생성된 run_shadow.sh 파일을 편집기로 아래 부분을 수정해준다

[원본]

```
python3 aws-iot-device-sdk-python-v2/samples/pubsub.py --endpoint  
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt  
--cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id  
basicPubSub --topic sdk/test/Python --count 0
```

[변경]

```
python3 aws-iot-device-sdk-python-v2/samples/shadow.py --endpoint  
a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com --ca_file root-CA.crt  
--cert RaspberryPi.cert.pem --key RaspberryPi.private.key --client_id  
Raspi_Shadow --thing_name RaspberryPi
```



Device Shadow

Shadow 소스 실행(디바이스)

Pi3보드에서 run_shadow.sh 을 실행한다

./run_shadow.sh

```
192.168.101.101 (planx) - VNC Viewer
./run_shadow.sh
tea@planx ~/aws$ ./run_shadow.sh
Connecting to a1mp2lfc29w0b5-ats.iot.us-east-1.amazonaws.com with client ID 'Raspi_Shadow'...
Connected!
Subscribing to Update responses...
shadow_thing_name RaspberryPi
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Launching thread to read user input...
Thing has no shadow document. Creating with defaults...
Changed local shadow value to 'off'.
Updating reported shadow value to 'off'...
Update request published.
on_update_shadow_accepted: awsiot.iotshadow.UpdateShadowResponse(client_token='e9f10a10-6698-49be-8984-94849515bb50', metadata=awsiot.iotshadow.ShadowMetadata(desired={'color': {'timestamp': 1662787263}}, reported={'color': {'timestamp': 1662787263}}), state=awsiot.iotshadow.ShadowState(desired={'color': 'off'}, desired_is_nullable=False, reported={'color': 'off'}, reported_is_nullable=False), timestamp=datetime.datetime(2022, 9, 10, 14, 21, 3), version=119)
Finished updating reported shadow value to 'off'.
Enter desired value:
```




Device Shadow

▣ 클래식 새도우 생성

AWS IoT 콘솔의 [관리] → [모든 디바이스] → [사물]로 가서 목록에 있는 “RaspberryPi”를 클릭하고 들어가서 하단에서 [디바이스 새도우] 탭을 선택하고 [새도우 생성]을 클릭한다

디바이스 새도우 생성

☐ 명명된 새도우
속성에 대한 액세스를 관리하고 디바이스 속성을 논리적으로 그룹화하기 위해 이름이 서로 다른 여러 디바이스 새도우를 생성합니다.

☒ 이름 없는 (클래식) 새도우
사물에는 이름 없는 (클래식) 새도우가 하나만 있을 수 있습니다.

취소 생성

[디바이스 새도우 생성] 창에서 **[이름없는 (클래식) 새도우]**를 선택하고 [생성] 버튼을 클릭한다



Device Shadow

클래식 새도우 확인

생성된 “클래식 새도우 ” 를 클릭하면 하단의 [디바이스 새도우 문서]에서 오른쪽과 같은 디바이스 새도우 상태와 새도우 메타데이터를 볼 수 있게 된다

디바이스 새도우 (1) 정보

디바이스 새도우를 사용하면 연결된 디바이스가 AWS와 상태를 동기화할 수 있습니다. HTTPS 및 MQTT 주제를 사용하여 이 사물의 디바이스 새도우에 대한 상태 정보를 가져오거나 업데이트하거나 삭제할 수도 있습니다.

디바이스 새도우 필터링

이름

MQTT 주제 접두사

클래식 새도우

\$aws/things/RaspberryPi/shadow

디바이스 새도우 상태

```
{
  "state": {
    "desired": {
      "welcome": "aws-iot"
    },
    "reported": {
      "welcome": "aws-iot"
    }
  }
}
```

디바이스 새도우 메타데이터

```
{
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1662788416
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1662788416
      }
    }
  }
}
```

충청 ICT IS
Chungcheong ICT Innovation Square

54

KSA 한국표준협회



Device Shadow

▣ 새도우 상태 변경

Pi3보드에서 “red”라고 입력하고 enter를 치면
AWS 콘솔의 디바이스 새도우 상태가 변경 되는 것을
볼 수 있을 것이다

“blue”, “yellow”, “white”등으로 차례로 변경해본다

```
red
Changed local shadow value to 'red'.
Updating reported shadow value to 'red'...
Update request published.
on_update_shadow_accepted: awsiot.iotshadow.UpdateShadowResponse
=awsiot.iotshadow.ShadowMetadata(desired={'color': {'timestamp': 1662789108,
state=awsiot.iotshadow.ShadowState(desired={'color': 'red'},
s_nullable=False), timestamp=datetime.datetime(2022, 9, 10, 1, 10, 8)),
Finished updating reported shadow value to 'red'.
Enter desired value:
_
```

디바이스 새도우 상태

```
{
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "red"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "red"
    }
  }
}
```

디바이스 새도우 메타데이터

```
{
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1662789077
      },
      "color": {
        "timestamp": 1662789108
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1662789077
      },
      "color": {
        "timestamp": 1662789108
      }
    }
  }
}
```



Device Shadow

▣ 새도우 상태 변경

AWS 콘솔의 디바이스 새도우 문서 옆의 [MQTT 주제] 탭으로 가서 이름 /get 에서 오른쪽 끝의 링크 아이콘을 누르면 이 토픽를 주제로 하는 [MQTT 테스트 클라이언트]가 표시된다

디바이스 새도우 문서		MQTT 주제
MQTT 주제 정보 이 디바이스 새도우에 대한 MQTT 주제를 사용하면 애플리케이션이 이 사물의 디바이스 새도우와 상호 작용하는 MQTT 메시지를 게시 및 구		
이름	작업	MQTT 주제
/get	게시	\$aws/things/RaspberryPi/shadow/get
/get/accepted	구독	\$aws/things/RaspberryPi/shadow/get/accepted
/get/rejected	구독	\$aws/things/RaspberryPi/shadow/get/rejected
/update	게시	\$aws/things/RaspberryPi/shadow/update



Device Shadow

▣ 새도우 상태 변경

[게시] 버튼을 누르면
현재 새도우 상태를
얻어 창에 출력 된다

Pi3보드는 “off”를 입력하고
다시 “quit”을 입력하여
프로그램을 종료해 놓는다

```
Enter desired value:
quit
Exiting sample: User has quit
Disconnecting...
Disconnected.
tea@planx ~/aws
```

주제 구독
주제 게시

주제 이름
주제 이름은 메시지를 식별합니다. 메시지 페이로드는 0의 서비스 품질(QoS)을 사용하여 이 주제에 게시됩니다.

메시지 페이로드

▶ 추가 구성

게시

구독	\$aws/things/RaspberryPi/shadow/get/accepted
\$aws/things/RaspberryPi/shadow/get/rejected ♡ ✕	
\$aws/things/RaspberryPi/shadow/get/accepted ♡ ✕	<div>▼ \$aws/things/RaspberryPi/shadow/get/accepted</div> <pre>{ "state": { "desired": { "welcome": "aws-iot", "color": "off" }, "reported": { "welcome": "aws-iot", "color": "off" } } }</pre>



Device Shadow

▣ 새도우 상태 변경

다시 [MQTT 주제]탭으로 가서
이름 /update 의 링크를 눌러
[MQTT 테스트 클라이언트]
화면에서 [메시지 페이로드]에
다음을 입력하고 [게시]버튼을
누른다

```
{
  "state": {
    "desired": {
      "color": "purple"
    }
  }
}
```

주제 구독
주제 게시

주제 이름
주제 이름은 메시지를 식별합니다. 메시지 페이로드는 0의 서비스 품질(QoS)을 사용

메시지 페이로드

```
{
  "state": {
    "desired": {
      "color": "purple"
    }
  }
}
```

▶ 추가 구성

게시



Device Shadow

▣ 새도우 상태 변경

다시 MQTT 테스트 클라이언트

\$aws/things/RaspberryPi/shadow/get 으로 가서 [게시]를 누르면 다음과 같이 delta를 볼 수 있을 것이다

desired의 color값과 reported의 color값이 다르므로 생성된 것이다

\$aws/things/RaspberryPi/shadow/get/accepted

▼ \$aws/things/RaspberryPi/shadow/get/accepted

```
{
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "purple"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "off"
    },
    "delta": {
      "color": "purple"
    }
  },
}
```




Device Shadow

▣ 새도우 상태 변경

pi3보드에서 다시 ./run_shadow를 실행 시키고 get으로 얻어오면
delta값에 있던 color값 purple 로
reported의 color값이 변경되고
delta 가 없어지는 것을 알 수 있다

```
▼ $aws/things/RaspberryPi/shadow/get/accepted  
  
{  
  "state": {  
    "desired": {  
      "welcome": "aws-iot",  
      "color": "purple"  
    },  
    "reported": {  
      "welcome": "aws-iot",  
      "color": "purple"  
    }  
  },  
}
```



Device Shadow

▣ 새도우 상태 변경(실습)

run_shadow_led.sh파일을 새로 만들고 끝부분에
"--shadow_property led" 를 추가하고 led 값을
'on' 과 'off'로 변경하면서 실습 보드의 led를 제어해본다
shadow_led.py 소스 파일을 사용한다

shadow_led 프로그램을 종료하고 새도우
update후 다시 보드에서 프로그램을 다시
실행시켜 LED 동작을 확인해본다

```
$aws/things/RaspberryPi/shadow/update

메시지 페이로드

{
  "state": {
    "desired": {
      "led": "on"
    }
  }
}
```

감사합니다

