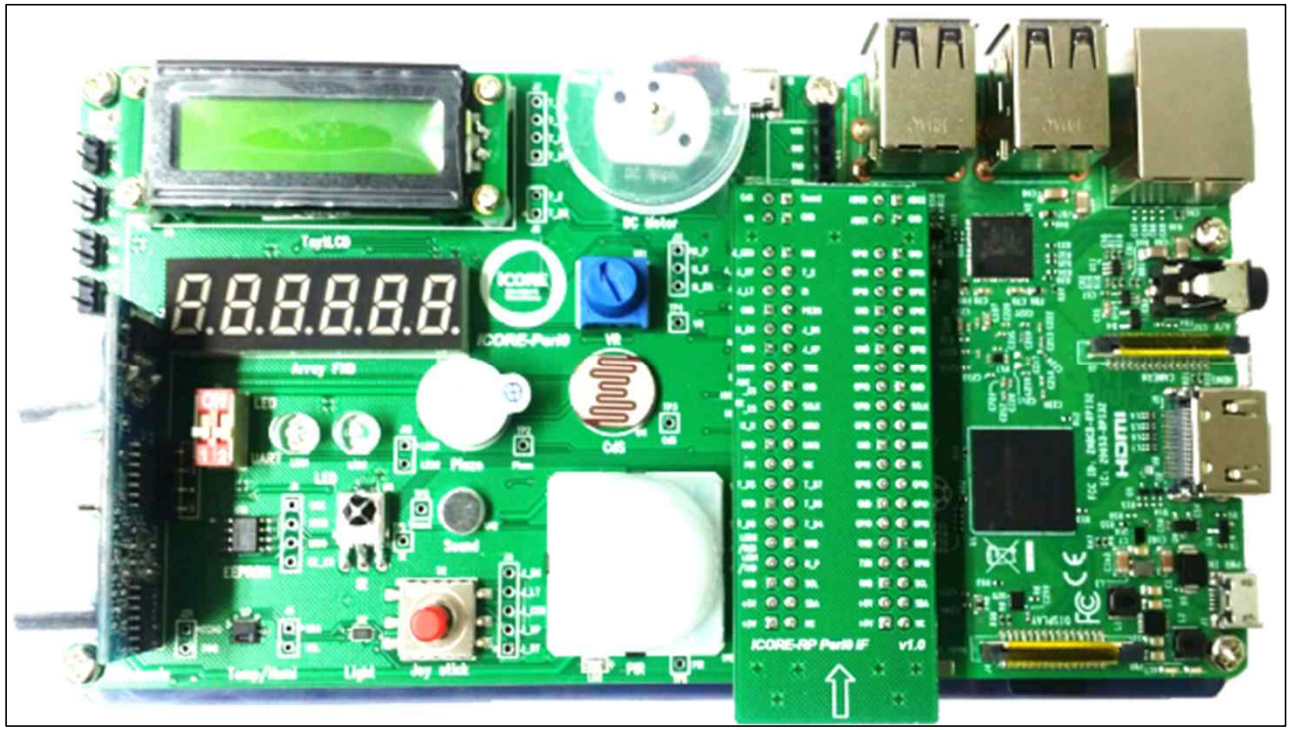
AWS 활용 IoT

[1] AWS IoT Device Core

강사 : 고병화

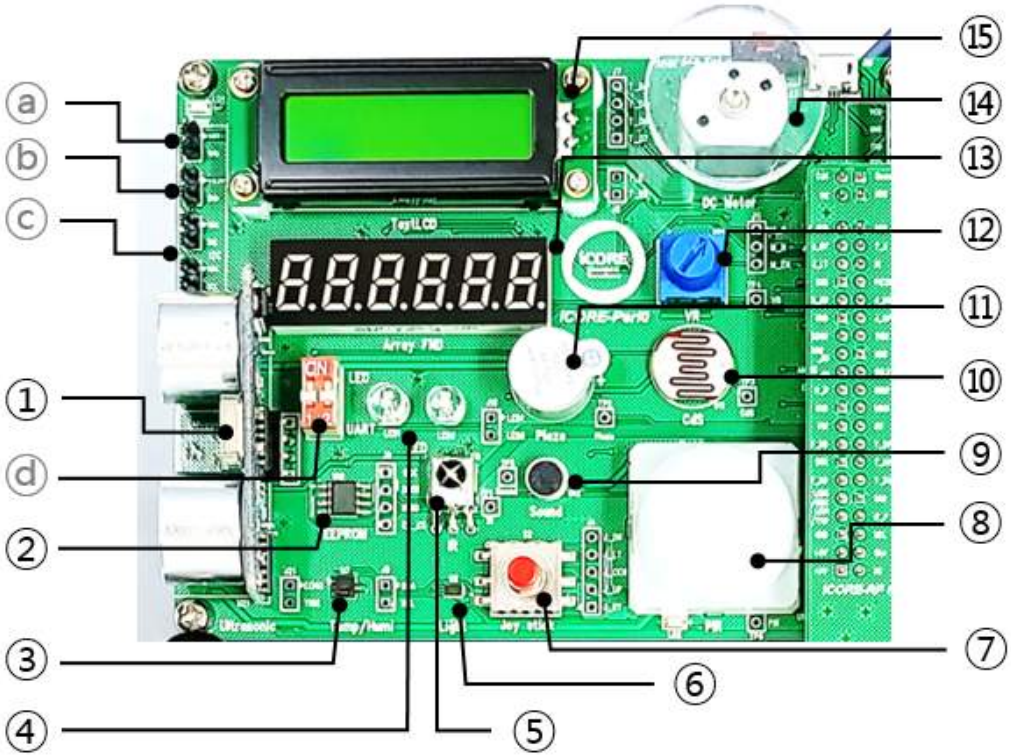
실습 보드 환경 구성
ICORE-SDP (Raspberry PI 3)

실습 보드 : ICORE-SDP (Raspberry PI 3)



iCORE-SDP는
Raspberry PI 3
main 모듈과
센서 모듈로
구성되어 있다

실습 보드 : ICORE-SDP (Raspberry PI 3)



① Ultrasonic Sensor: 초음파 송수신기 모듈. V2.0은 ECHO 핀이 LOW일 때 거리 측정. V2.1은 HIGH일 때 거리 측정

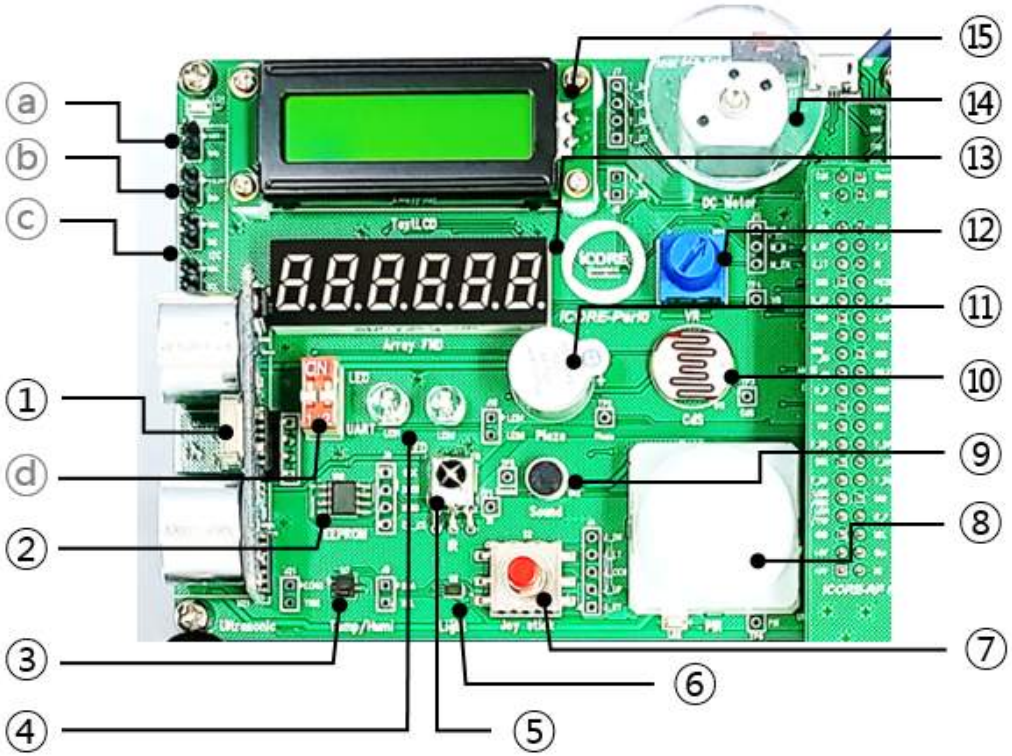
② EEPROM: 2Kbyte 크기의 EEPROM

③ Temperature & Humidity Sensor: I2C로 제어하는 온습도 센서

④ LED x 2: UART 핀과 공유하는 LED. LED 선택 스위치가 LED면 UART기능에 의해 사용하지 않아도 켜져 있음

⑤ IR Receiver: 일반 리모컨 신호 수신 모듈

실습 보드 : ICORE-SDP (Raspberry PI 3)



⑥ Light Sensor: 럭스 단위로 빛의 밝기를 측정하는 디지털 센서

⑦ Joystick: 위, 아래, 오른쪽, 왼쪽, 확인(중앙)으로 구성된 선택 스위치

⑧ PIR Sensor: 적외선 기반 움직임 검출 센서

⑨ Sound Sensor: ADC에 연결된 주변 소리 감지 센서

⑩ CdS Sensor: ADC에 연결된 빛의 밝기를 측정하는 아날로그 센서

⑪ Piezo Buzzer: 주파수에 따른 소리를 출력하는 부저

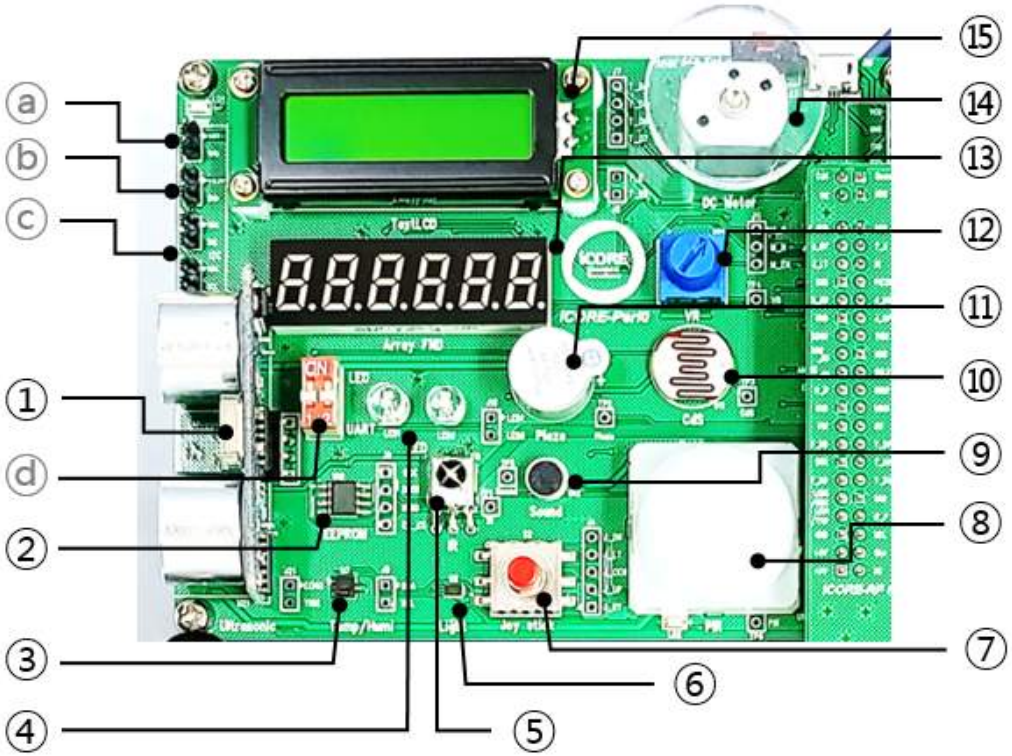
⑫ Potentiometer: ADC에 연결된 가변 저항

⑬ FND Array: 6개의 FND를 하나로 묶은 LED 기반 출력 전용 모듈

⑭ DC Motor: 모터 드라이버에 연결된 DC 모터

⑮ Character LCD: 4bit 데이터 라인으로 제어하는 텍스트 출력 전용 모듈

실습 보드 : ICORE-SDP (Raspberry PI 3)



ⓐ 전원1: 5V, GND 전원 핀

ⓓ 전원2: 3.3V, GND 전원 핀

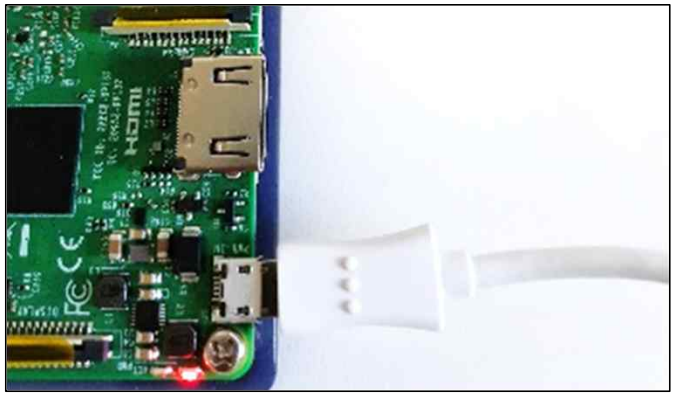
ⓒ I2C X 2: I2C 채널0, 1

ⓓ LED 선택 스위치: LED일 때 LED/UART 공용 신호를 LED 소자에 연결

실습 보드 : 시작 하기(전원 연결)

iCORE-SDP에 부팅 디바이스가 연결되었는지 확인한다. iCORE-SDP는 T-Flash(Micro SD) 또는 USB 저장소가 탑재된 상태로 제공되며 부팅 디바이스에는 라즈비안 픽셀 OS의 수정 버전이 탑재되어 있다.

1. iCORE-SDP에 전원 어댑터를 연결한다. iCORE-SD 전원 어댑터는 5V 2A 사용을 권장한다.



✓ 전원 스위치가 없으므로 전원 어댑터를 연결하는 즉시 부팅을 시작한다.

전원 커넥터 아래쪽 빨간색 전원 LED가 켜짐

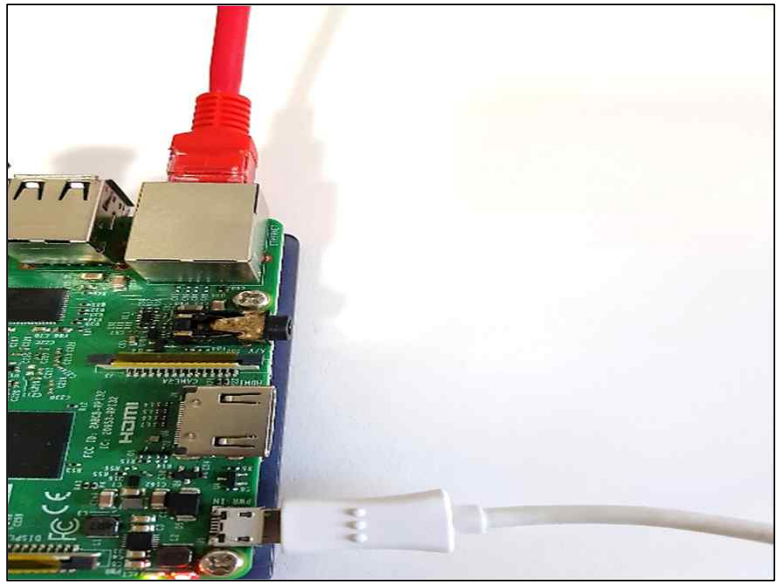
USB 저장소를 사용할 때 부팅 시간은 일반적으로 20초 이내이며, 그래픽 데스크톱 실행까지는 45초 정도가 걸린다. T-FLASH는 이보다 더 빠르다.

실습 보드 : 네트워크 설정

1. Wi-Fi는 AP를 통해 인터넷에 연결되어 있다고 가정한다.

2. 이더넷은 PC와 연결되어 있으며 PC의 이더넷 주소는 192.168.101.120으로 가정한다.

실습 보드 : 네트워크 설정



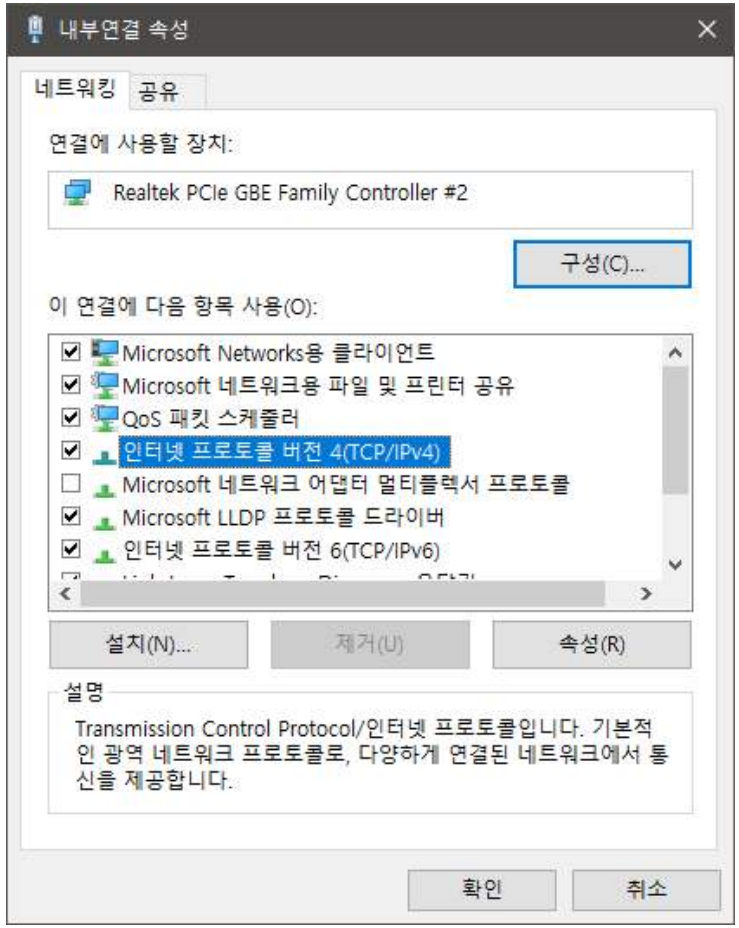
1. 제공하는 이더넷 케이블로 PC와 iCORE SDP를 연결한다. 만약 이더넷이 없는 호스트 환경에서는 시중에서 판매하는 Ethernet to USB 모듈 사용하면 된다

2. iCORE-SDP는 이더넷의 IP 주소로 192.168.101.101 이 설정되어 있다.

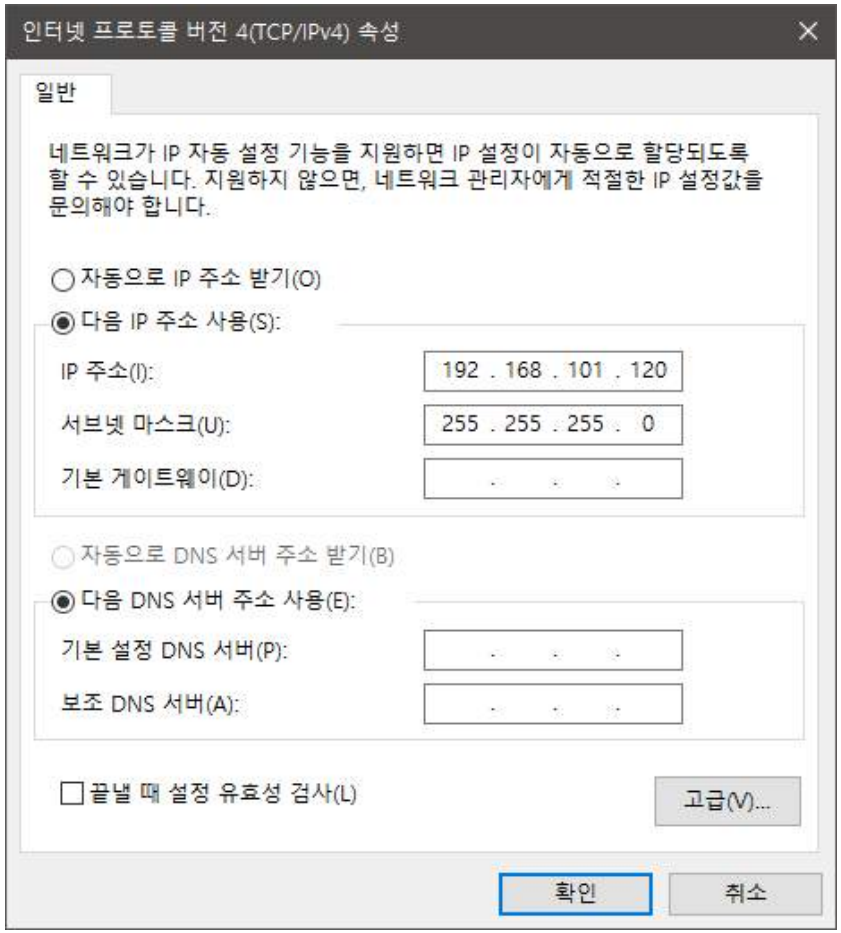
3. PC의 제어판에서 '네트워크 및 인터넷 --> 네트워크 상태 및 작업 보기 --> 어댑터 설정 변경'을 선택한다.

실습 보드 : 네트워크 설정

iCORE-SDP와 이더넷 케이블로 연결한 이더넷 포트의 속성 창을 실행한 후 ‘인터넷 프로토콜 버전 4(TCP/IPv4)’의 속성을 선택한다.



실습 보드 : 네트워크 설정



IP 주소는 192.168.101.120을 입력하고 서브넷 마스크는 255.255.255.0를 입력한다. 기본 게이트웨이와 DNS는 비워 둔다.

- ✓ 실제 IP 주소 중 그룹을 나타내는 앞 부분 3개 필드(192.168.101)는 변경할 수 없지만 호스트 주소인 마지막 필드는 1 ~ 254 중101을 제외하면 어떤 값을 사용해도 된다.

  게이트웨이는 반드시 비워 두어야 한다.

실습 보드 : VNC 서비스와 원격 데스크톱

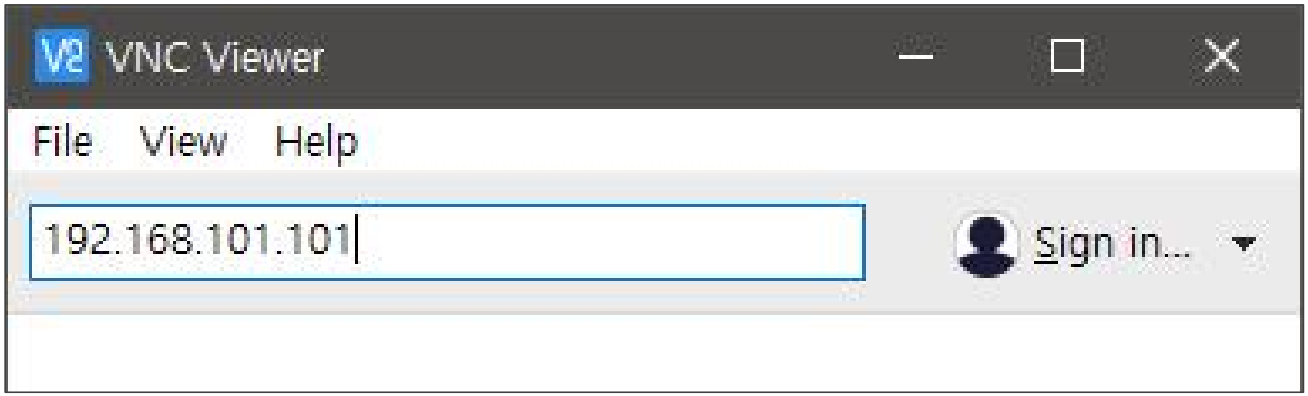
VNC Virtual Network Computing 서비스는 TCP/IP 네트워크 환경에서 RFB Remote Frame Buffer 프로토콜을 이용해 그래픽 데스크톱 화면을 원격지에서 공유하는 서비스로 키보드와 마우스 이벤트를 원격지로 전송해 네트워크를 거쳐 그래픽 화면을 갱신하는 방식을 사용한다. iCORE-SDP에는 VNC 서버가 실행 중이므로 PC에 클라이언트인 VNC Viewer를 설치한 후 iCORE-SDP에 접속해 원격 데스크톱을 시작할 수 있다.

실습 보드 : VNC 서비스와 원격 데스크톱

1. 배포 사이트에서 VNC Viewer를 다운로드해 설치한다.(배포파일 사용)

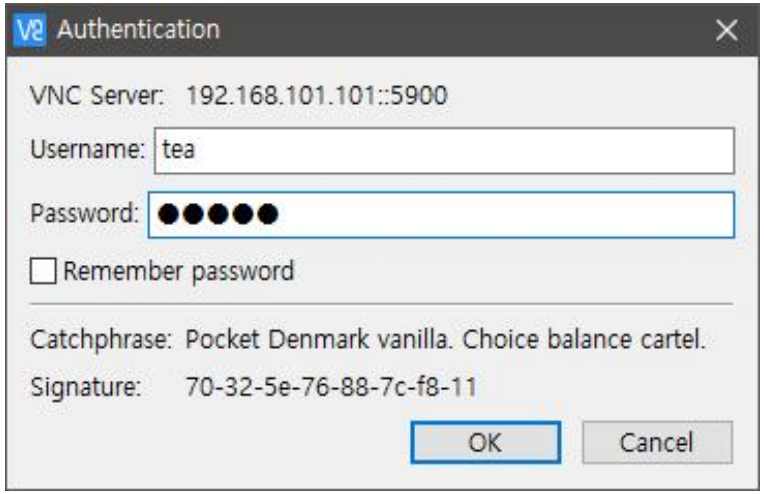
- https://www.realvnc.com/download/viewer

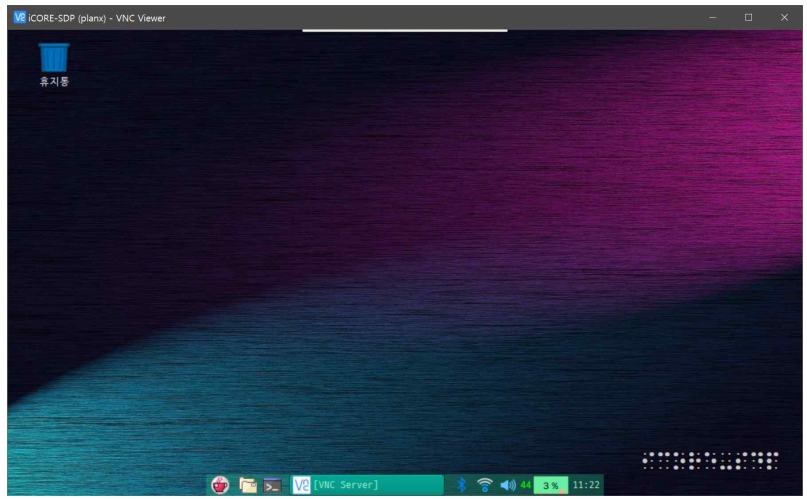
2. VNC Viewer를 실행한 후 주소 창에 192.168.101.101을 입력해 iCORE-SDP에 접속한다.



실습 보드 : VNC 서비스와 원격 데스크톱

3. 인증 창이 표시되면 Username에는 tea를 입력하고, Passwd에는 planx를 입력한 후 OK 버튼을 누른다.





4. 정상적으로 연결되면 iCORE-SDP의 데스크톱 화면이 VNC Viewer에 표시된다. VNC Viewer 화면 위에서 발생하는 마우스나 키보드 이벤트는 이더넷을 통해 iCORE-SDP에 전달된다.

실습 보드 : SSH 서비스와 가상 터미널

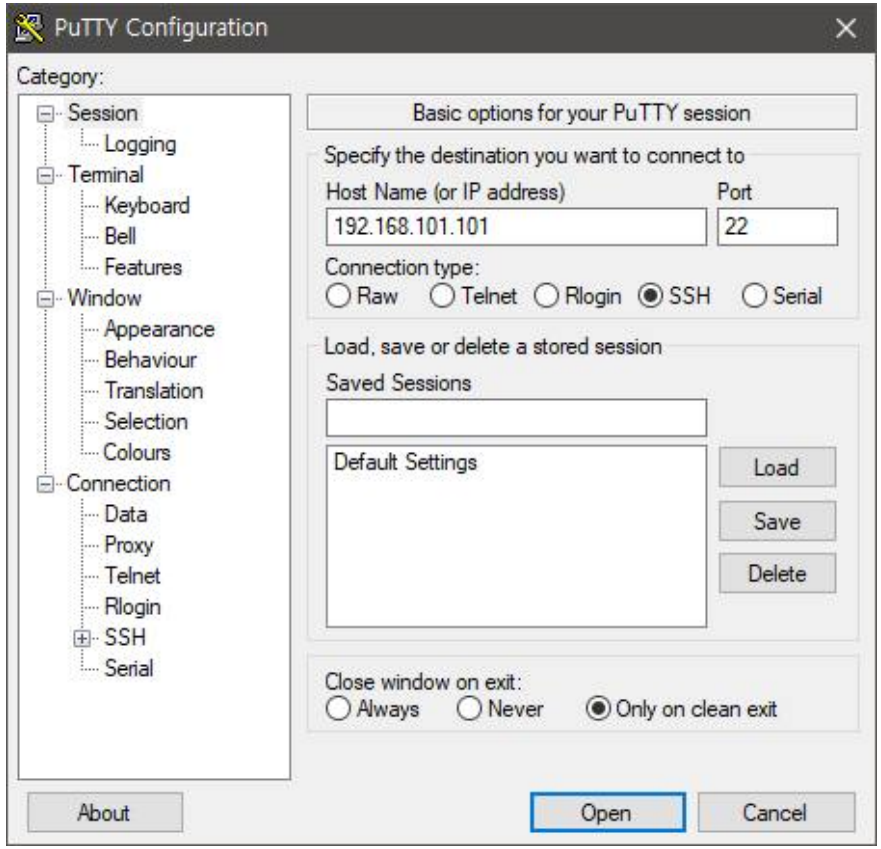
SSH Secure Shell은 TCP/IP 네트워크를 기반으로 다른 컴퓨터에 로그인하거나 원격 시스템에서 명령을 실행하고 다른 시스템으로 파일을 복사할 수 있도록 해주는 응용프로그램 또는 프로토콜을 가리킨다. 기존의 rsh, rlogin, telnet 등을 대체하기 위해 개발되었으며 암호화된 통신을 지원하므로 안전하지 못한 네트워크에서 안전하게 통신할 수 있는 기능을 제공한다. iCORE-SDP에는 SSH 서버가 실행 중이므로 PC에 클라이언트인 PuTTY를 설치한 후 iCORE-SDP에 접속해 가상 터미널을 시작할 수 있다.

실습 보드 : SSH 서비스와 가상 터미널

1. 배포 사이트에서 PuTTY를 다운로드해 설치한다.

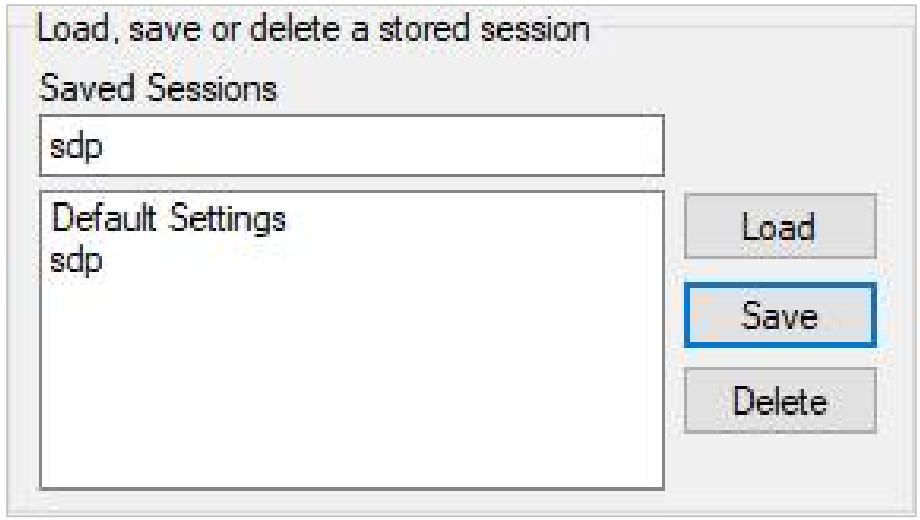
- http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

2. PuTTY를 실행한 후 Connection types는 SSH를 선택하고 Host Name에는 192.168.101.101을 입력하고 Port에는 22를 입력한다.



실습 보드 : SSH 서비스와 가상 터미널

3. Saved Sessions에 sdp를 입력한 후 Save 버튼을 누른다. 다음에 PuTTY를 실행할 때는 목록에서 sdp를 선택한 후 Load 버튼을 누르면 저장된 설정을 재 사용할 수 있다.

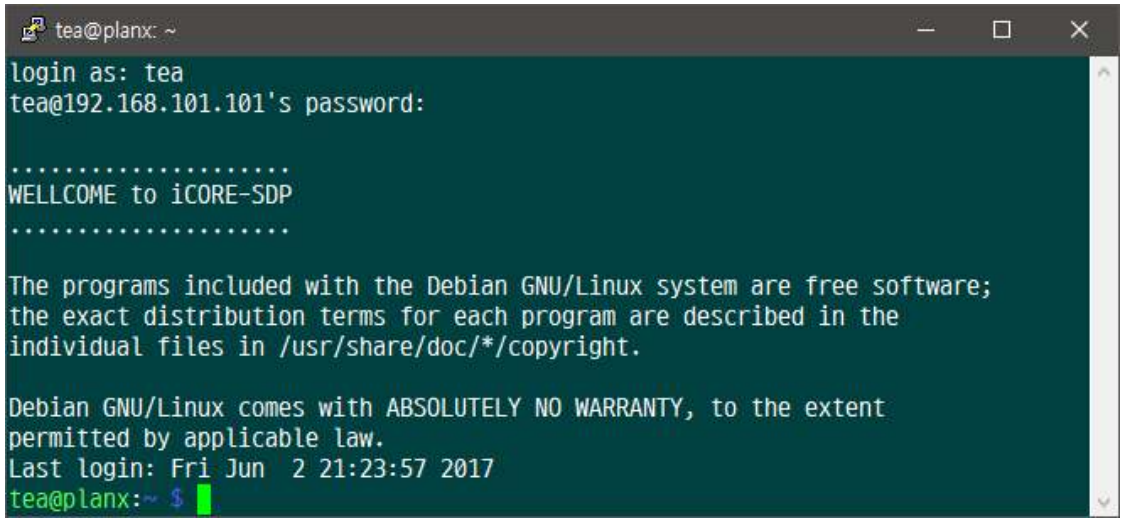


실습 보드 : SSH 서비스와 가상 터미널

4. 설정이 완료되면 Open 버튼을 눌러 iCORE-SDP에 접속하는데, 보안 경고 창이 표시되면 iCORE-SDP의 보안 키를 PC에 저장하도록 yes 버튼을 누른다.

실습 보드 : SSH 서비스와 가상 터미널

5. 로그인 메시지가 표시되면 ID와 비밀번호는 각각 **tea**와 **planx**를 입력한다. 비밀번호를 화면에 표시되지 않으며 틀리면 ID부터 다시 해야 한다.



실습 보드 : 네트워크 연결(WinSCP)

WinSCP 는 PC와 Pi3 간에 네트워크로 파일 전송을 할 수 있는 툴이다

File protocol : SCP

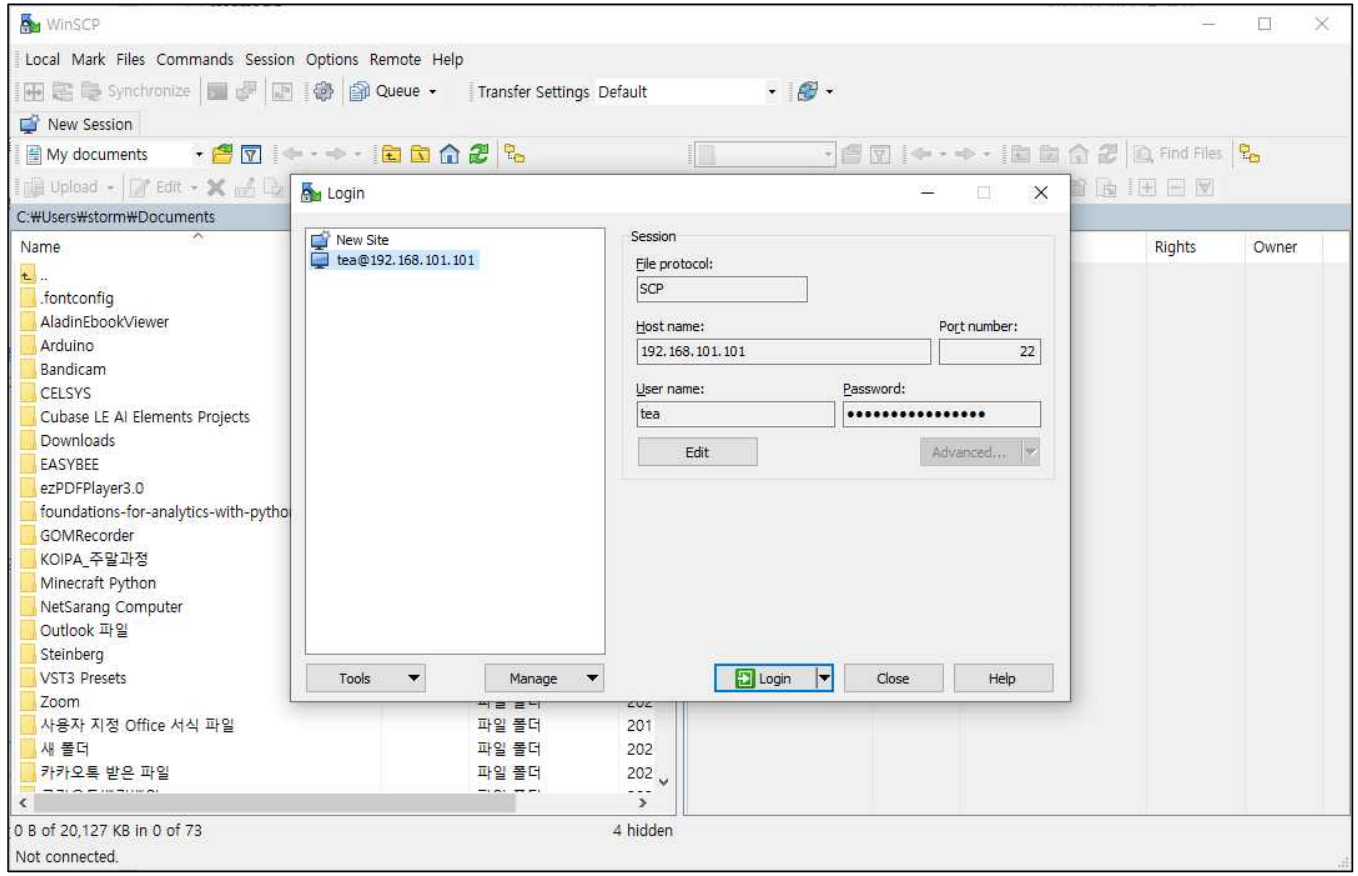
Host name:192.168.101.101

Port number: 22

User name : tea

Password: planx

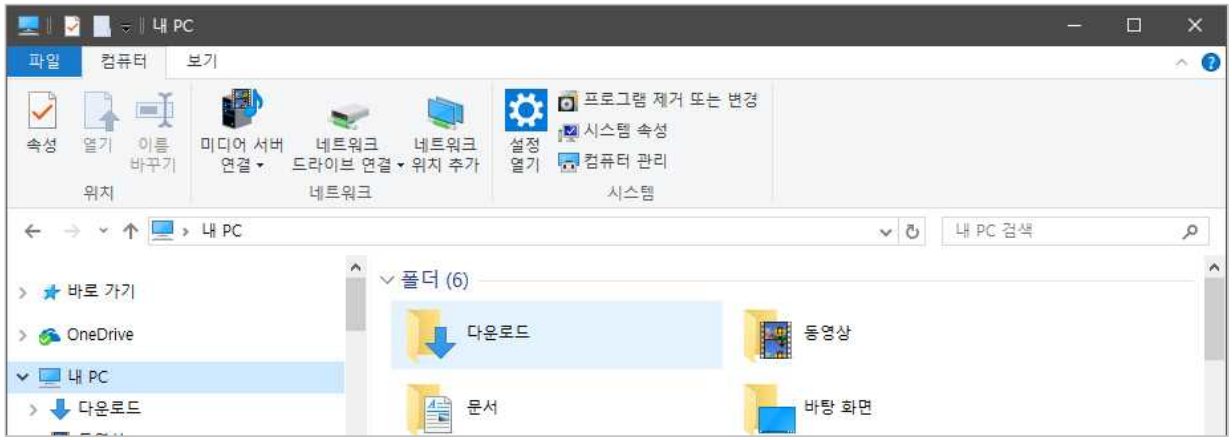
으로 입력하여 로그인 한다



실습 보드 : 네트워크 드라이브 연결(Samba)

네트워크 드라이브는 윈도우 운영체제에서 원격지의 저장소를 로컬 저장소처럼 접근하는 기술로 iCORE-SDP에는 Samba 서버가 실행 중이므로 iCORE-SDP의 저장소를 PC에서 네트워크 드라이브로 연결해 접근하는 것이 가능하다.

1. 윈도우 탐색기를 실행한다.

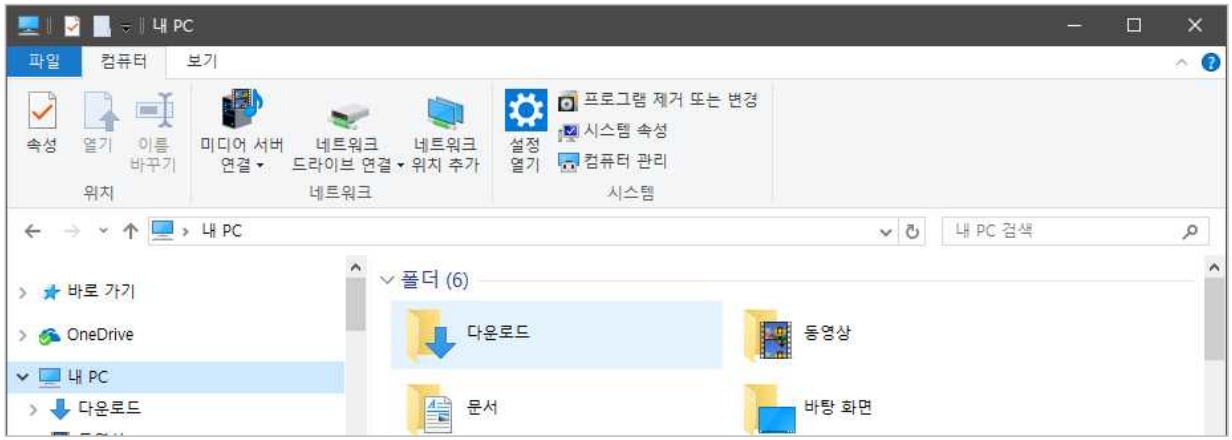


2. 내 PC를 선택한 후 도구 모음에서 네트워크 드라이브 연결을 선택한다.

실습 보드 : 네트워크 드라이브 연결(Samba)

네트워크 드라이브는 윈도우 운영체제에서 원격지의 저장소를 로컬 저장소처럼 접근하는 기술로 iCORE-SDP에는 Samba 서버가 실행 중이므로 iCORE-SDP의 저장소를 PC에서 네트워크 드라이브로 연결해 접근하는 것이 가능하다.

1. 윈도우 탐색기를 실행한다.



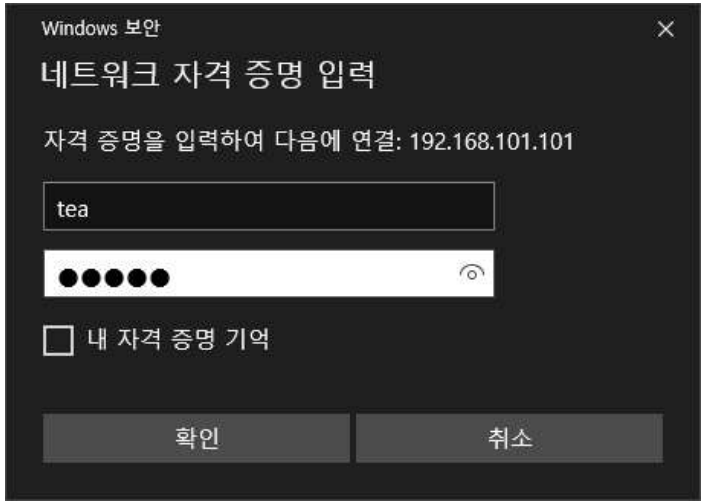
2. 내 PC를 선택한 후 도구 모음에서 네트워크 드라이브 연결을 선택한다.

실습 보드 : 네트워크 드라이브 연결(Samba)

3. 네트워크 드라이브 연결 창이 실행되면 폴더 입력란에 \\192.168.101.101\tea를 입력한다. “로그인할 때 다시 연결”은 체크하지 않는다. 드라이브 문자는 변경 가능하나 z:로 가정한다.

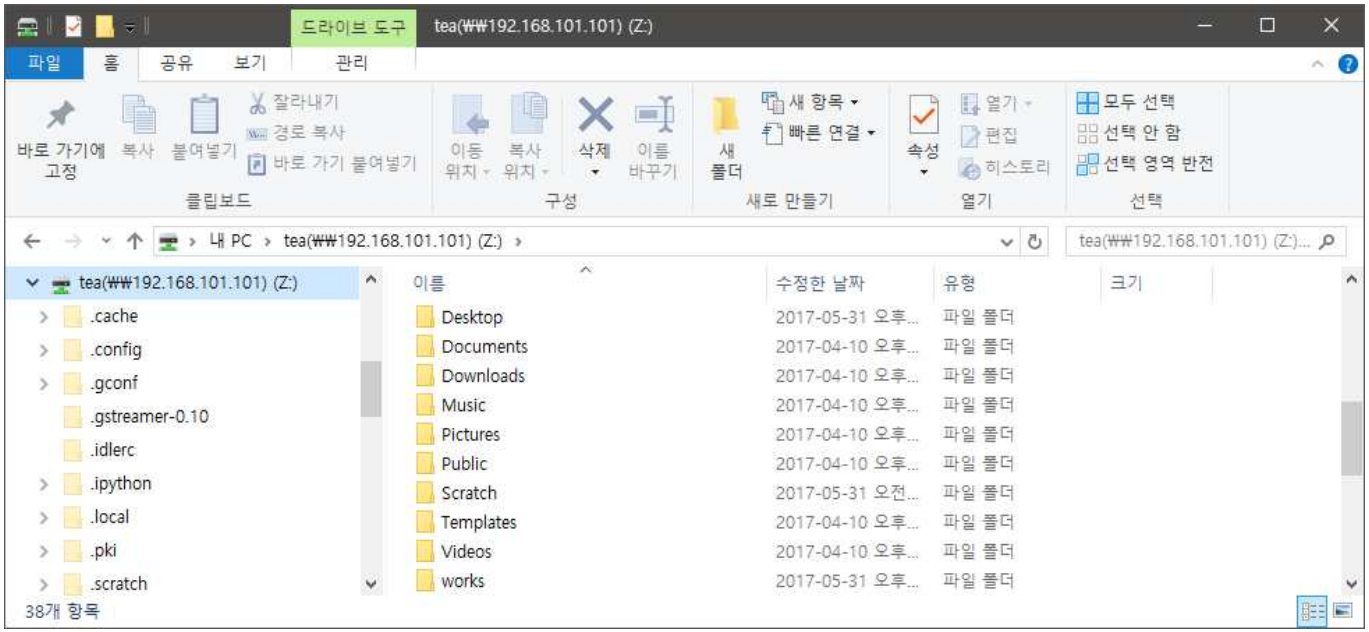
실습 보드 : 네트워크 드라이브 연결(Samba)

4. Windows 보안 창이 실행되면 ID와 패스워드에 tea, planx를 차례로 입력한다. ‘내 자격 증명 기억’은 체크하지 않는다.



실습 보드 : 네트워크 드라이브 연결(Samba)

5. 이제 Z: 드라이버를 통해 iCORE-SDP에 파일을 추가하거나 복사하기, 삭제, 편집 등이 가능하다.



✓ TEA 계정의 홈 디렉터리에서만 작업이 가능하다.

✓ 네트워크 드라이브는 PC와 ICORE-SDP 사이에 파일을 복사하는데 사용한다.

실습 보드 : 안전한 종료

iCORE-SDP에는 별도의 전원 스위치가 없다. 오직 USB 전원 포트에 전원 어댑터를 연결하면 부팅을 시작하고 전원 어댑터를 분리하면 꺼지지만 가급적 다음 소개하는 안전한 종료를 권장한다.

1. 안전하게 종료할 때는 VNC Viewer로 iCORE-SDP에 접속한 후 터미널에서 shutdown명령을 실행한다.

```
tea@planx:~ $ sudo shutdown now
[ 3858.506906] reboot: Power down
```

2. 일정 시간(약 5초 이내)이 지나면 네트워크 상태를 나타내는 녹색 LED가 여러 번 깜빡이다가 꺼지는데 이 후 전원을 분리한다.

실습 보드 : 안전한 종료

3. 전원 차단 없이 다시 시작할 필요가 있을 때는 reboot 명령을 실행한다.

```
tea@planx:~ $ sudo reboot
[14754.863772] reboot: Restarting system
```

4. 참고로 데스크톱 환경에서는 '패널 > 시작 메뉴 > shutdown...'을 눌러 Shutdown options 창이 표시되면 Shutdown 또는 Reboot 버튼을 선택하면 된다.

AWS IoT Core 연결

AWS IoT Core란?

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

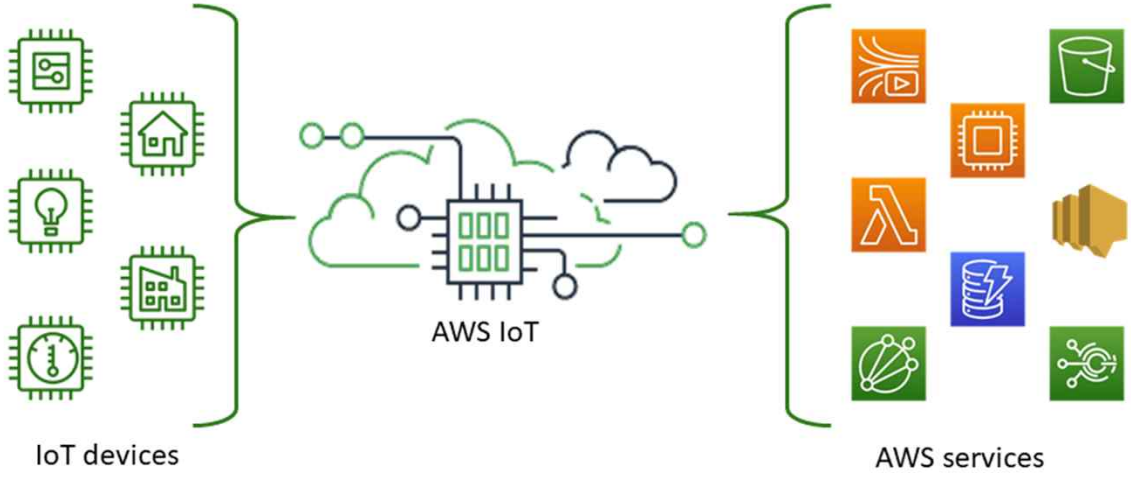
AWS IoT Core는 수십억 개의 디바이스와 수조 건의 메시지를 지원하고, 안전하고 안정적으로 이러한 메시지를 처리하여 AWS 엔드포인트 및 다른 디바이스로 라우팅할 수 있다.

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

AWS IoT Core란?

IoT Core에서 지원하는 프로토콜

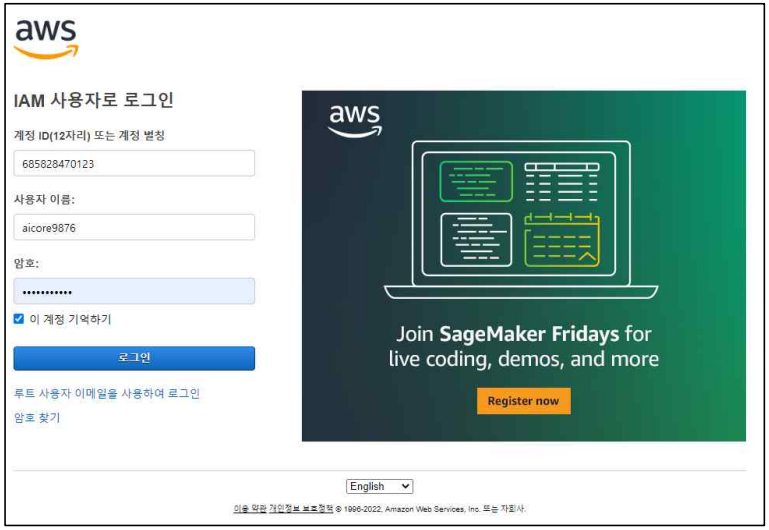
- MQTT(메시지 큐 및 원격 분석 전송)
- MQTT over WSS(웹 소켓 보안)
- HTTPS(하이퍼텍스트 전송 프로토콜 - 보안)
- LoRaWAN(장거리 광역 네트워크)

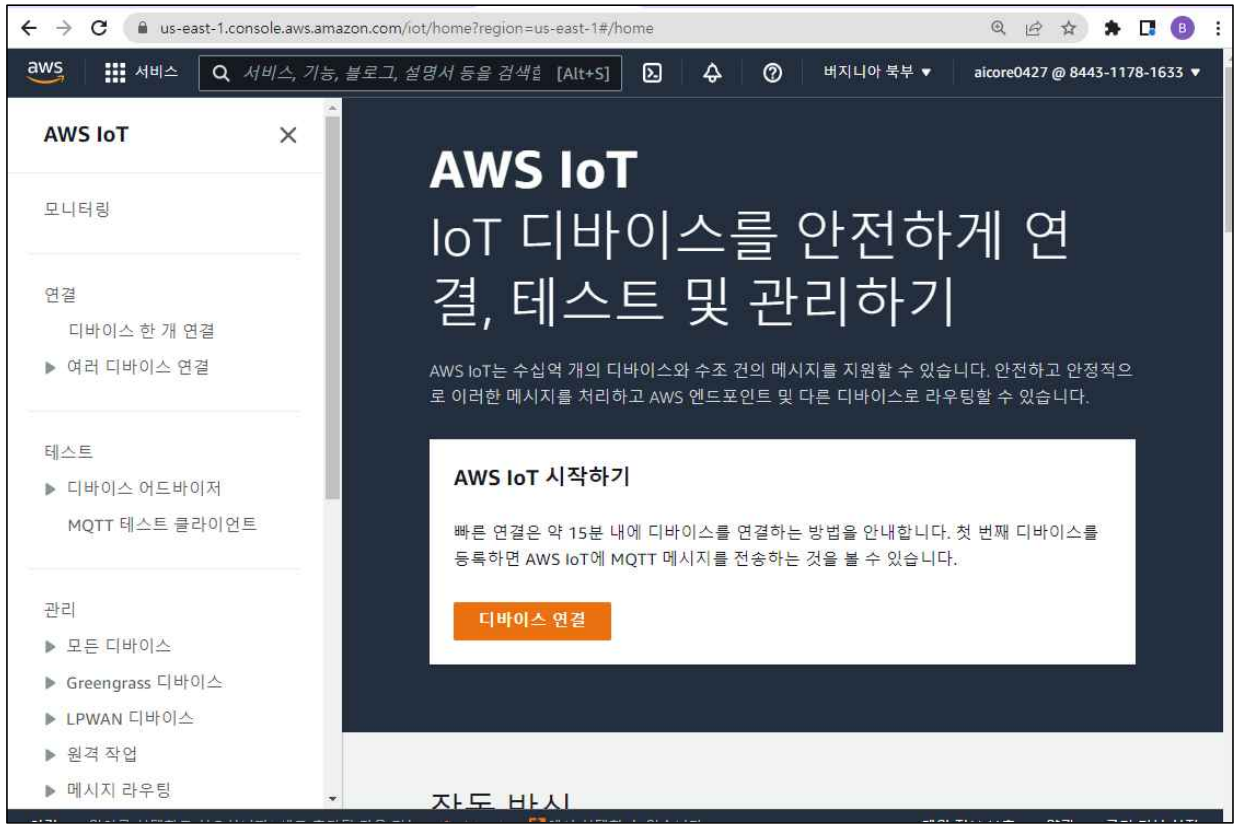


AWS IoT Core 연결

AWS 계정에 로그인하고 AWS IoT 콘솔을 연다

https://aws.amazon.com/ko/





AWS IoT Core 연결

AWS IoT Core 콘솔에서 [디바이스 연결]을 클릭한다

AWS IoT Core 연결

아래와 같이 디바이스 준비 화면이 나온다

AWS IoT Core 연결 : 디바이스(Pi3) 준비

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*

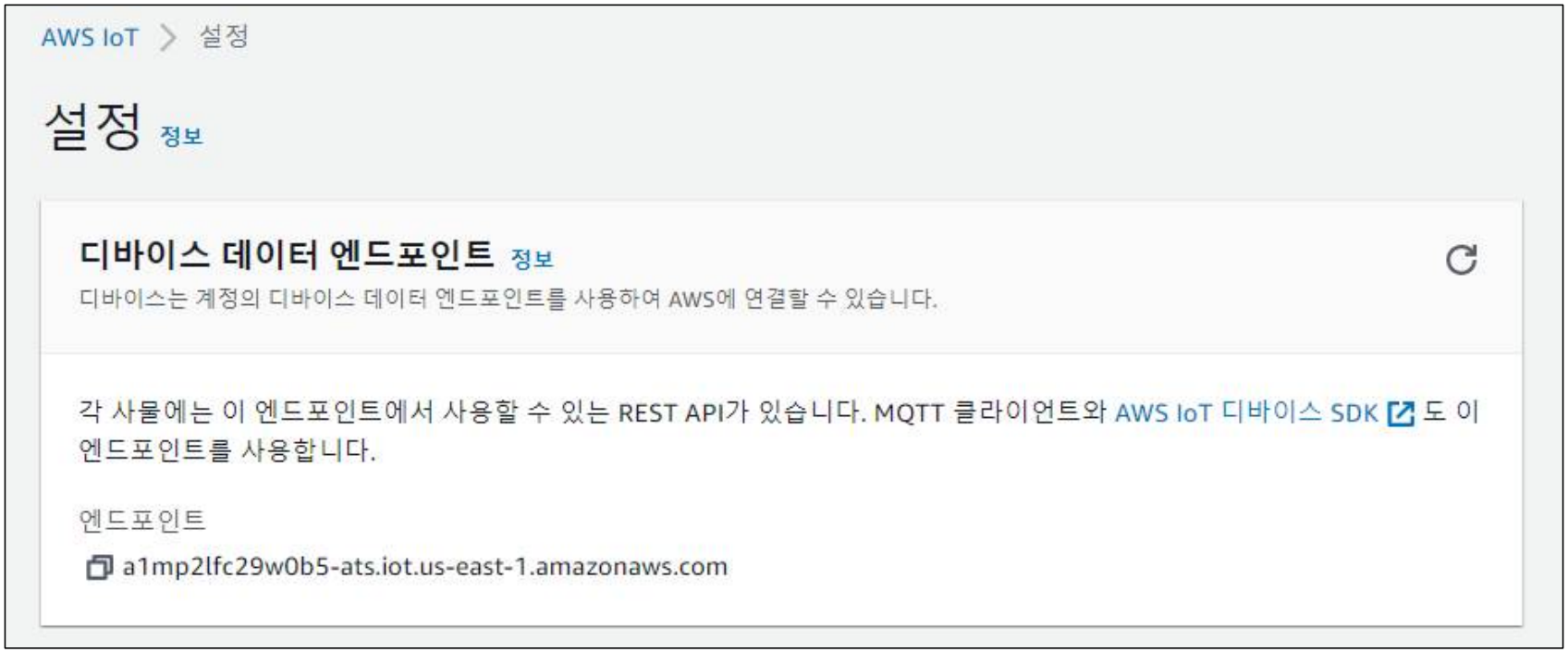
성공적인 ping 응답을 받으면 디바이스를 AWS IoT에 연결할 준비가 된 것이다.

AWS IoT Core 연결

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

AWS IoT Core 연결

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



AWS IoT Core 연결

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

AWS IoT Core 연결

디바이스 플랫폼 운영체제는
“Llinux/macOS” 를 선택하고

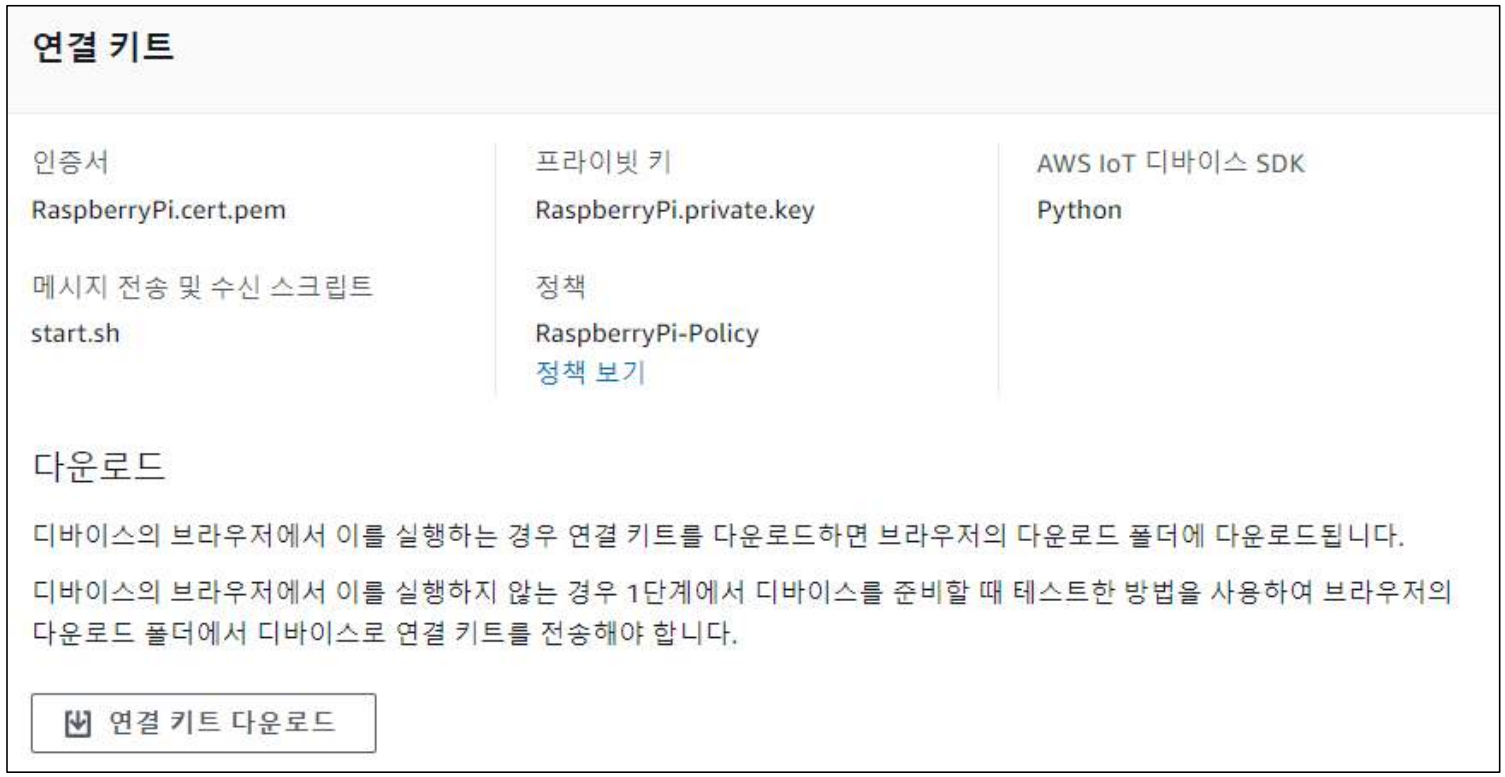
AWS IoT 디바이스 SDK는
“Python”으로 선택해준다

하단의 [다음] 버튼을 누른다

AWS IoT Core 연결

[연결 키트 다운로드복사] 버튼을 눌러 파일을 다운로드 받는다

“connect_device_package.zip” 압축파일이 다운 받아진다

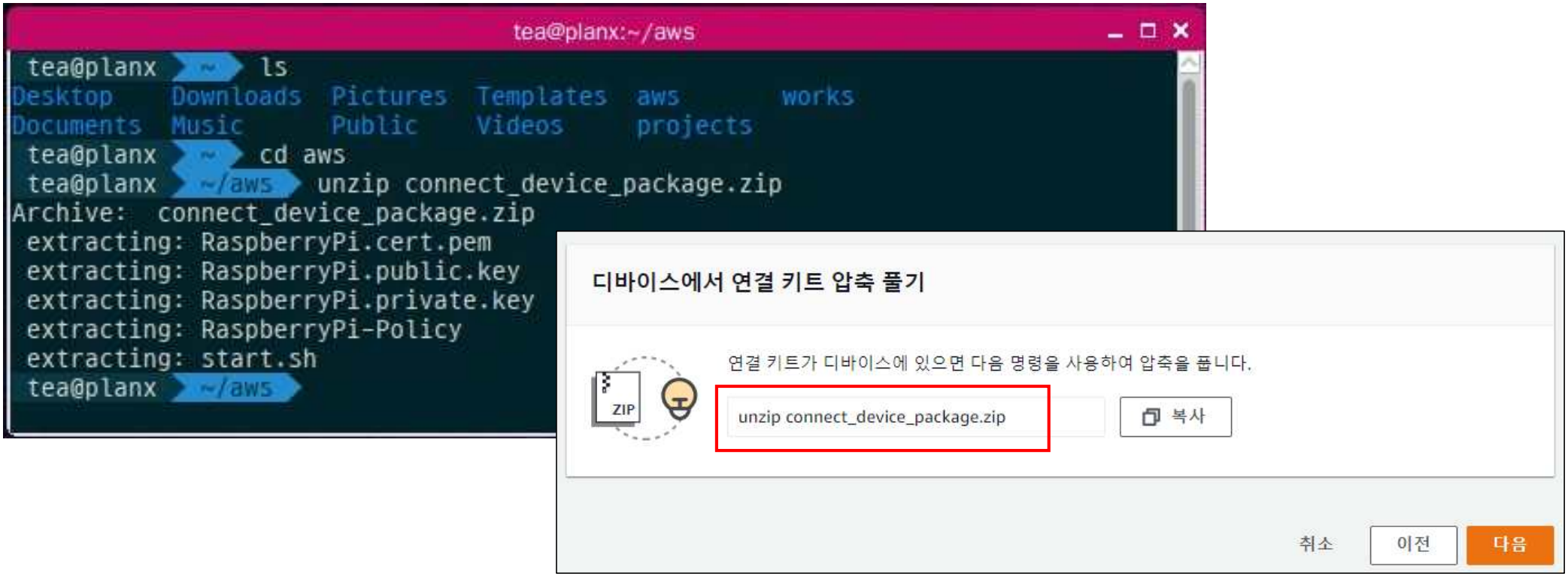


AWS IoT Core 연결

다운로드한 압축파일을 WinSCP를 사용하여 Pi3보드로 전송한다
(/home/tea/아래에 'aws'디렉토리를 하나 만들고 여기에 복사한다)

AWS IoT Core 연결

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



AWS IoT Core 연결

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

주의!
마지막 명령에서 'sudo’를 빠뜨리지 말고 반드시 사용한다

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

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

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     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% |[redacted]| 21.5MB 13kB/s
Installing collected packages: awscrt, awsiotsdk
  Running setup.py install for awscrt ... -
```

AWS IoT Core 연결

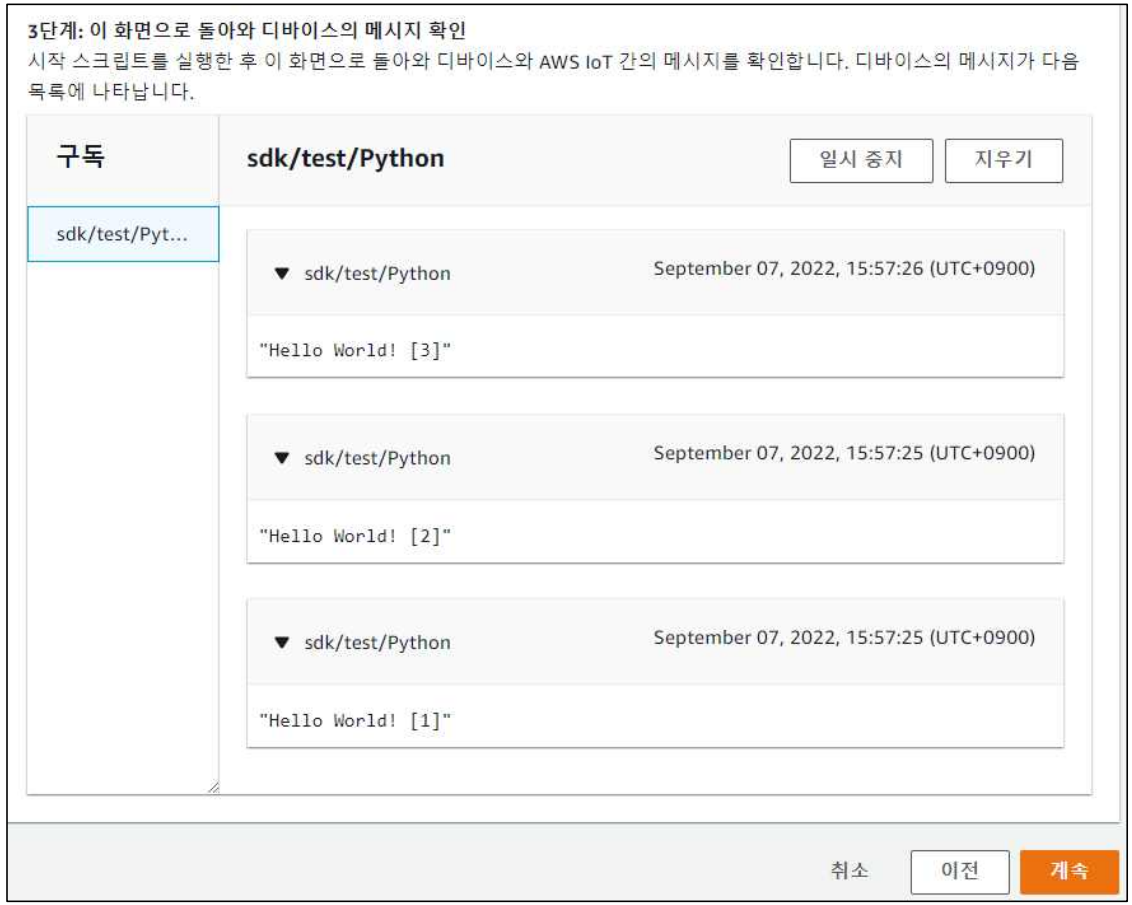
디바이스에서 설치 완료 후 자동으로 테스트 메시지를 보낸다

```
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 연결

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



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

AWS IoT Core 연결

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

AWS IoT Core 연결

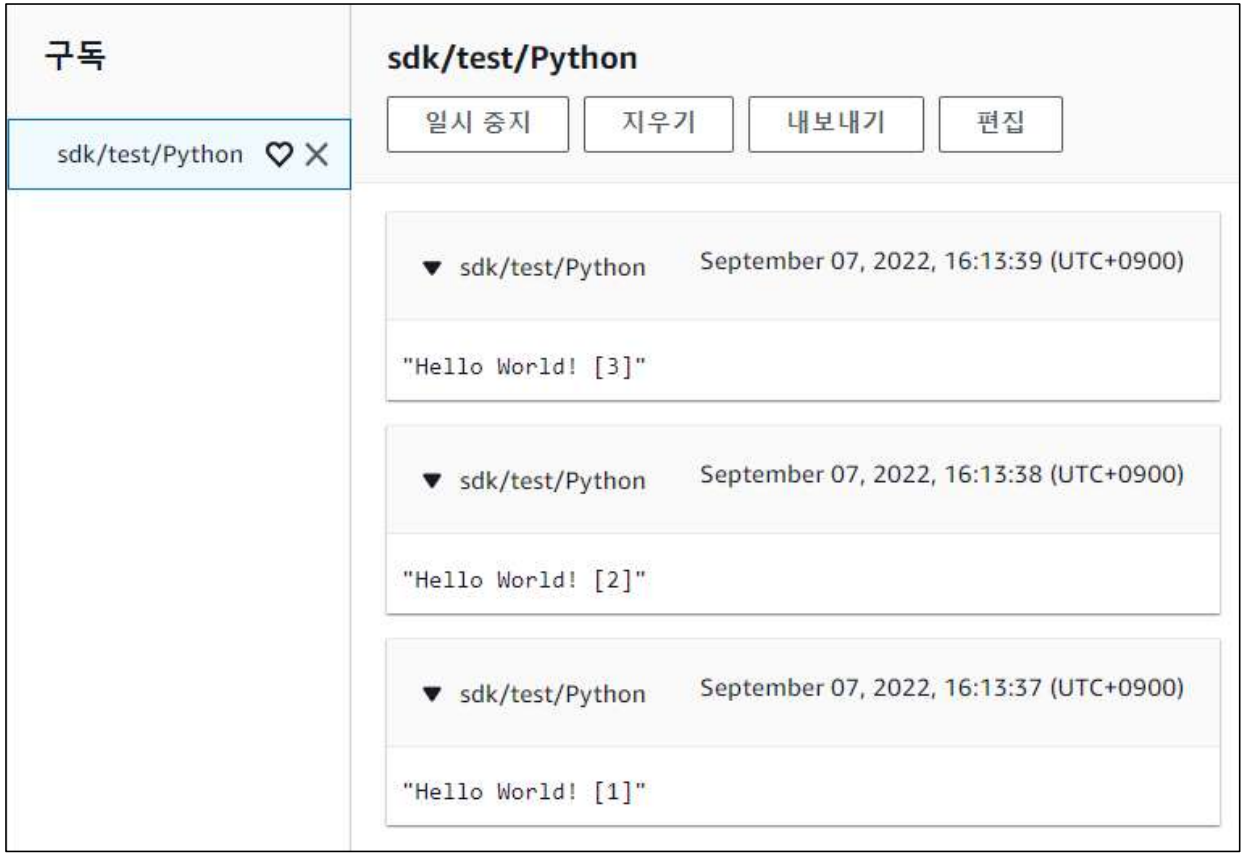
연결키트 다운로드 파일 connect_device_package.zip 을 압축 풀어서 start.sh파일의 마지막 줄의 명령을 복사하여 디바이스의 터미널에서 실행시킨다(CTRL-C를 눌러 중단 시킬 수 있다)

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.ama
onaws.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 연결

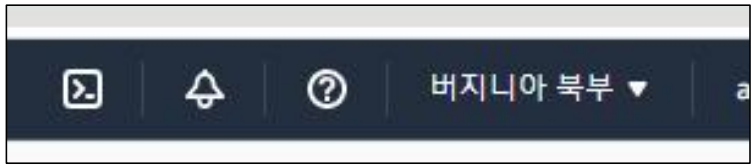
아래와 같이 메시지가 수신 되는 것을 볼 수 있다



데이터 수집 및 저장
(Rules engine / DynamoDB)

데이터 수집 및 저장 : DynamoDB 생성

AWS 콘솔의 우측상단의 CloudShell 아이콘을 눌러 AWS CloudShell을 실행시킨다



CloudShell에서 아래 DynamoDB 테이블 생성 명령을 실행시키다

```
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
```

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

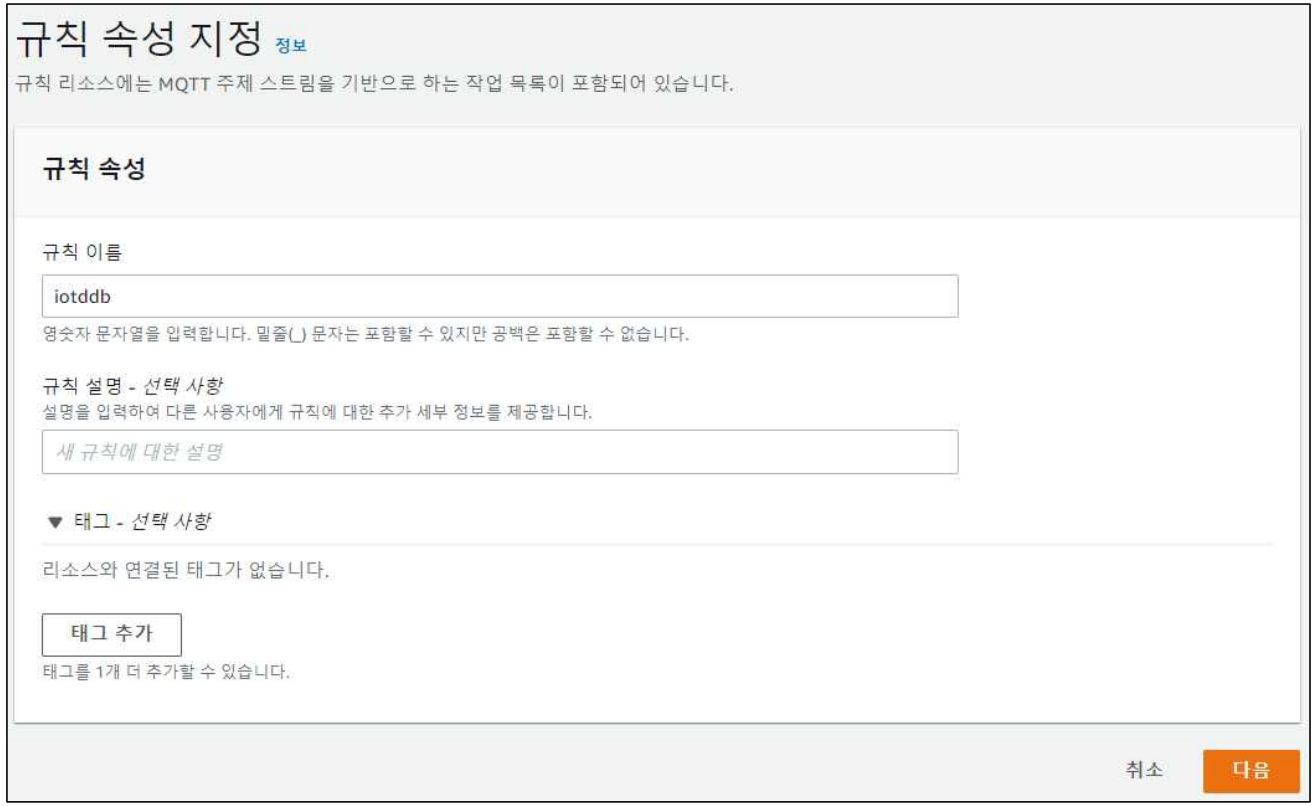
AWS 콘솔의 DynamoDB로 가서 왼쪽 에서 “테이블 ”을 클릭하면 아래 와 같이 테이블 목록이 나타난다

데이터 수집 및 저장 : Rule 생성

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

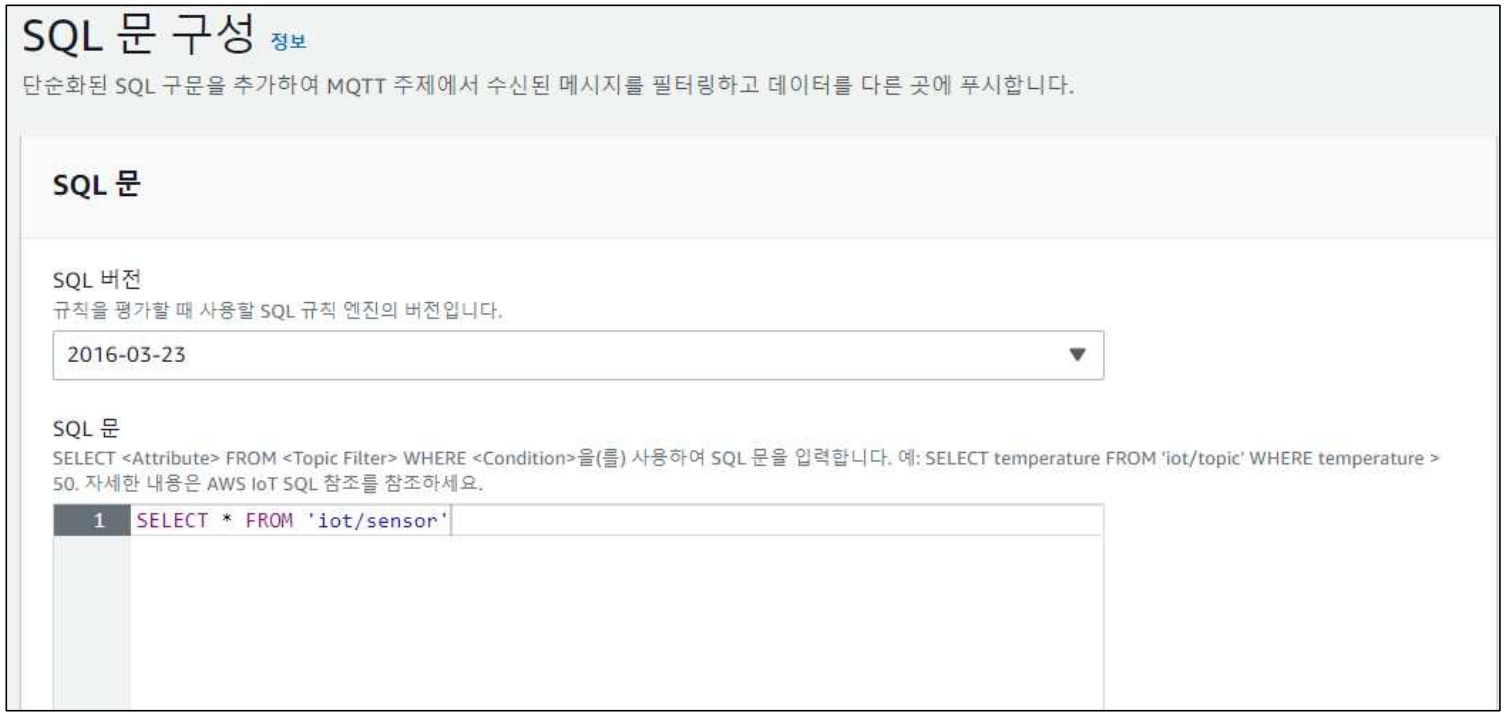
데이터 수집 및 저장 : Rule 생성

규칙 이름을 “iotddb”로 입력하고 [다음] 버튼을 클릭한다



데이터 수집 및 저장 : Rule 생성

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



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

IotDB | 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

[역할 생성] 창에서 [역할 이름]에 “iotddb_role”을 입력하고 [생성]을 클릭한다

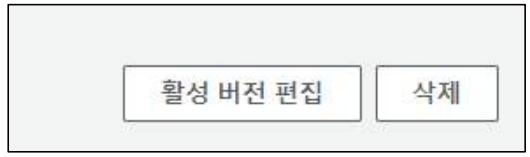
[다음]을 클릭하여 다음 단계로 넘어간다

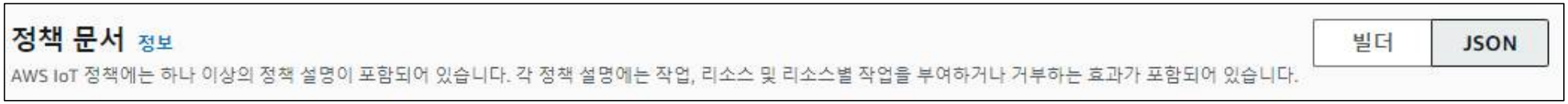
데이터 수집 및 저장 : Rule 생성

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

데이터 수집 및 저장 : 정책 설정

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



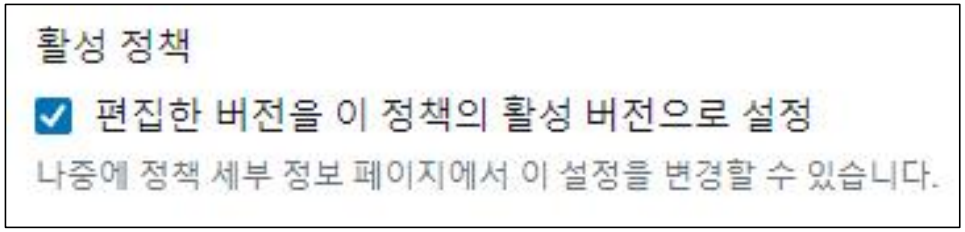


데이터 수집 및 저장 : 정책 설정

'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 클릭한다



데이터 수집 및 저장 : 정책 설정

정책 문서

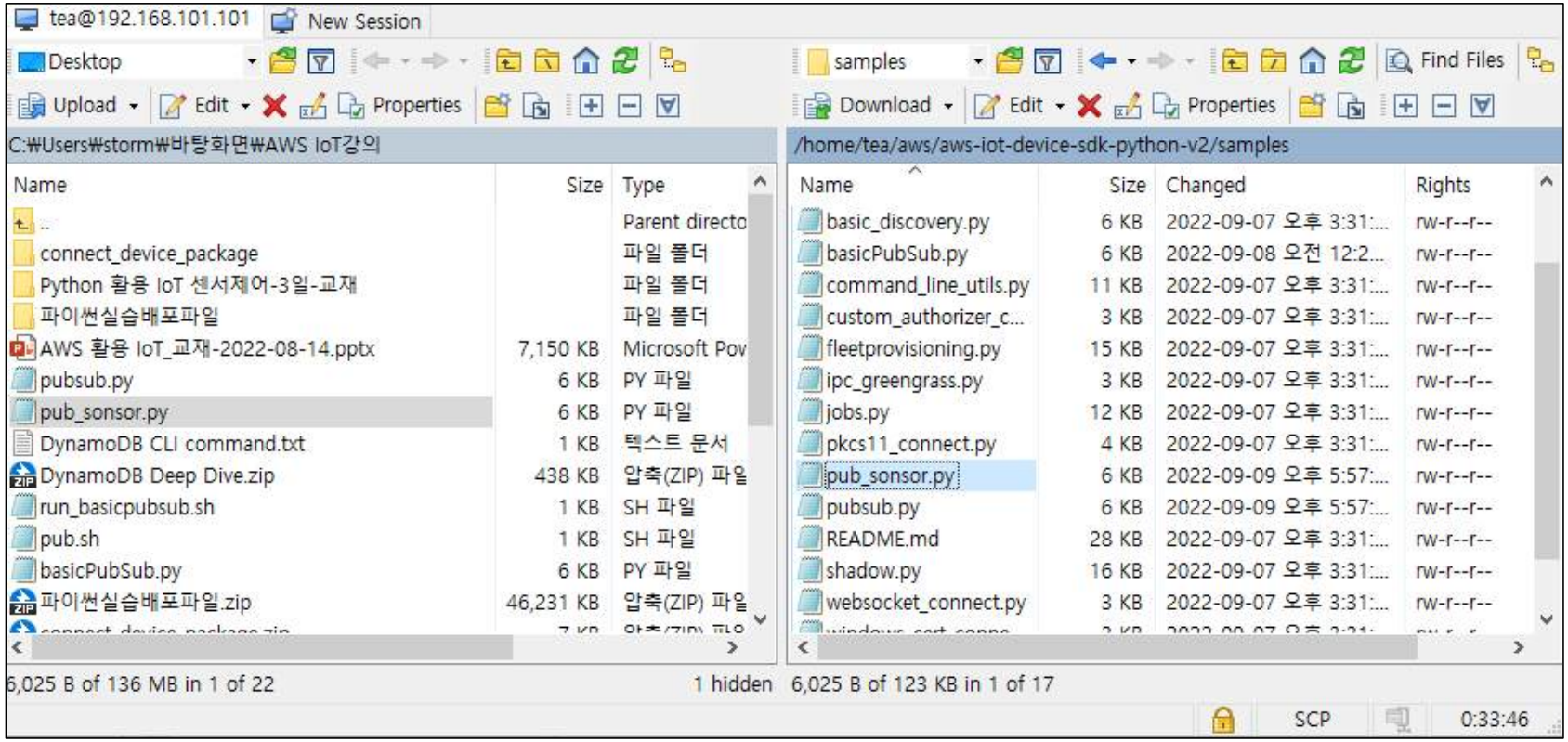
```
9          ],
10         "Resource": [
11           "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/java",
12           "arn:aws:iot:us-east-1:844311781633:topic/sdk/test/Python",
13           "arn:aws:iot:us-east-1:844311781633:topic/iot/sensor",
14           "arn:aws:iot:us-east-1:844311781633:topic/topic_2"
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

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

데이터 수집 및 저장 : 디바이스 소스 준비

WinSCP를 사용하여 배포된 pub_sonsor.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_sonsor.sh

chmod +x pub_sonsor.sh

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

cat pub_sonsor.sh

python3 aws-iot-device-sdk-python-v2/samples/pub_sonsor.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_sonsor.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_sonsor.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

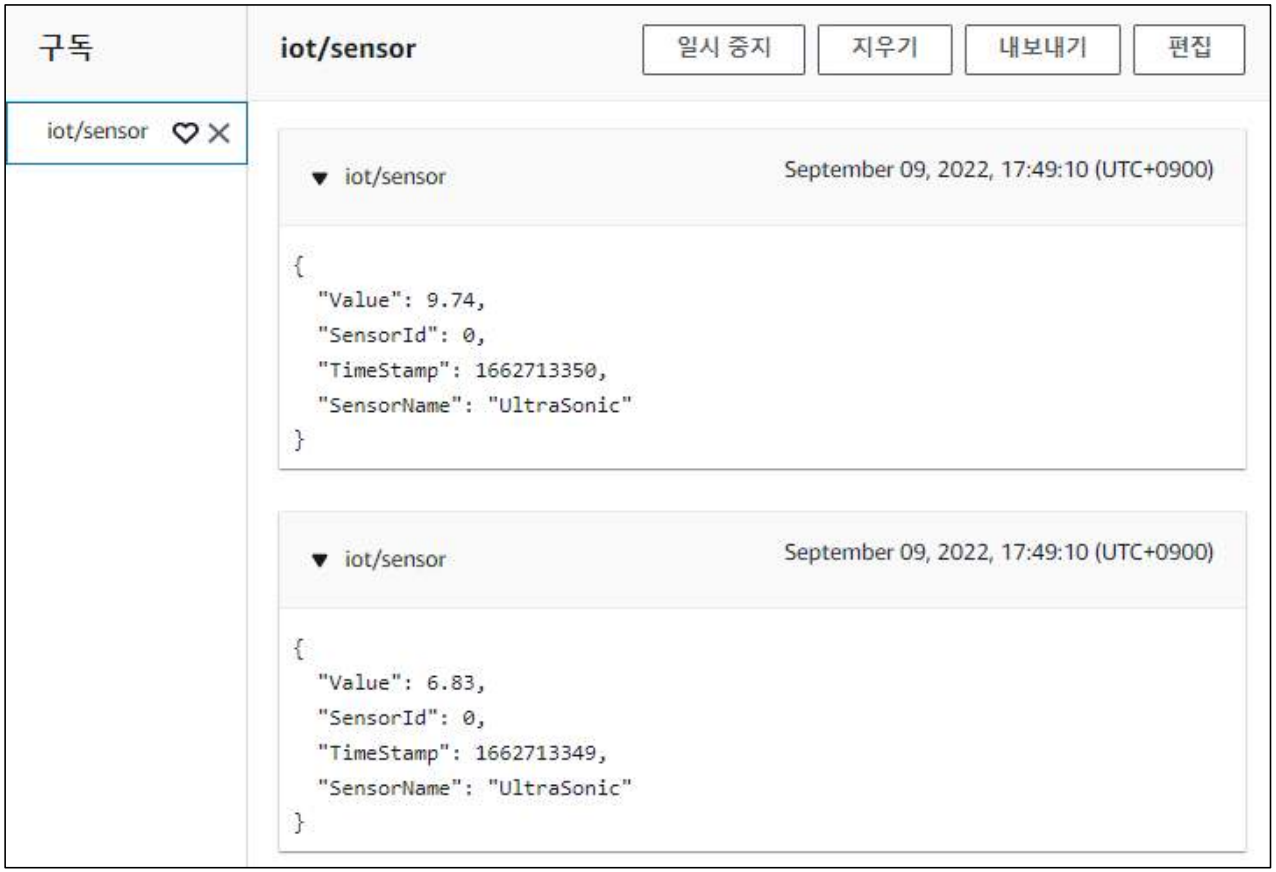
Pi3보드의 터미널 창에서 아래 명령을 실행하면 메시지가 publish된다

./pub_sonsor.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 : Subscribing하기

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



데이터 수집 및 저장 : DynamoDB 확인

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

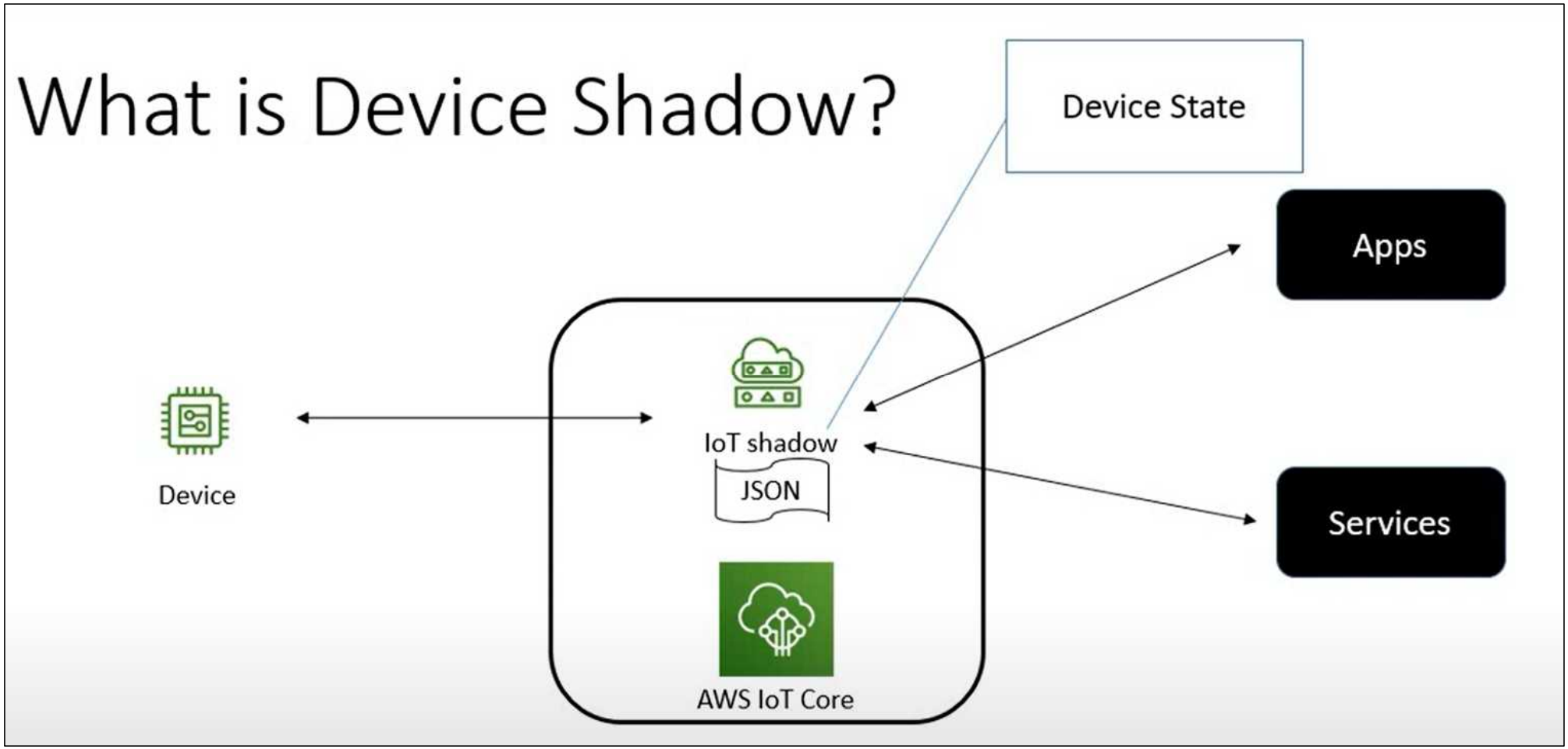
SensorId	TimeStamp	Payload
0	1662713339	{ "Value" : { "N" : "8.43" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713339" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713340	{ "Value" : { "N" : "7.88" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713340" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713341	{ "Value" : { "N" : "8.31" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713341" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713342	{ "Value" : { "N" : "6.68" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713342" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713343	{ "Value" : { "N" : "4.86" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713343" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713344	{ "Value" : { "N" : "6.05" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713344" }, "SensorName" : { "S" : "UltraSonic" } }
0	1662713345	{ "Value" : { "N" : "7.23" }, "SensorId" : { "N" : "0" }, "TimeStamp" : { "N" : "1662713345" }, "SensorName" : { "S" : "UltraSonic" } }
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

Device Shadow

Device Shadow : 기능



Device Shadow : Shadow를 위한 정책 설정

IoT Core 콘솔의 [보안]에 있는 [정책]을 클릭하고 정책이름 목록에서 “RapberryPi-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    }
```

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 : 섀도우 소스실행(디바이스)

Pi3보드에서 run_shadow.sh 을 실행한다

./run_shadow.sh

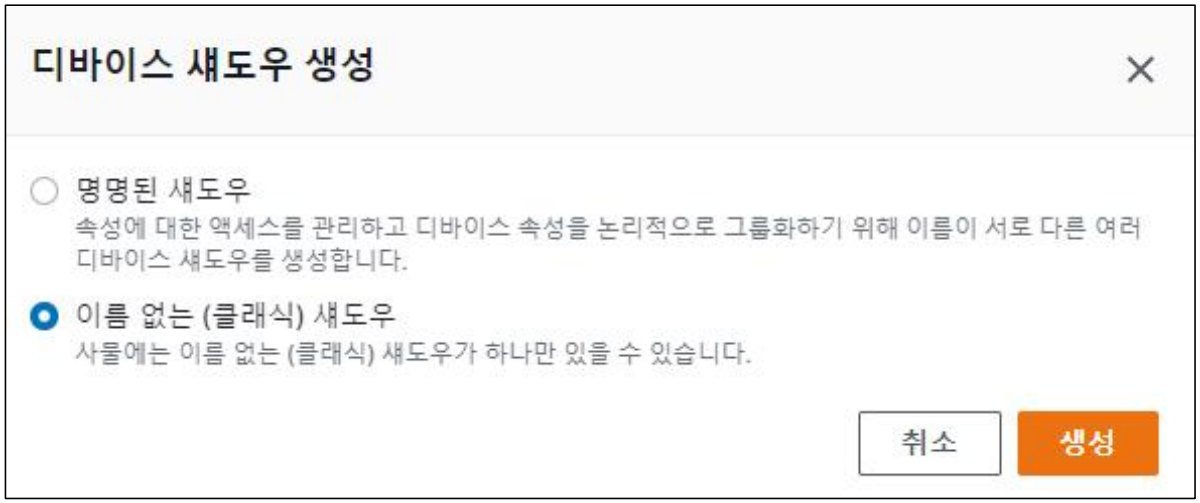
192.168.101.101 (planx) - VNC Viewer

./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_i
s_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 섀도우 생성

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



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

Device Shadow : 클래식 섀도우

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

디바이스 섀도우 (1) 정보

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

디바이스 섀도우 필터링

이름	MQTT 주제 접두사
클래식 섀도우	$aws/things/RaspberryPi/shadow

디바이스 섀도우 상태


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


디바이스 섀도우 메타데이터


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

Device Shadow : 섀도우 상태 변경

Pi3보드에서 “red”라고 입력하고 enter를 치면
AWS 콘솔의 디바이스 섀도우 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.UpdateShadowRespo
=awsiot.iotshadow.ShadowMetadata(desired={'color': {'timestam
 state=awsiot.iotshadow.ShadowState(desired={'color': 'red'},
s_nullable=False), timestamp=datetime.datetime(2022, 9, 10, 1
Finished updating reported shadow value to 'red'.
Enter desired value:
```

./run_shadow.sh

디바이스 섀도우 상태


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


디바이스 섀도우 메타데이터


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

Device Shadow : 섀도우 상태 변경

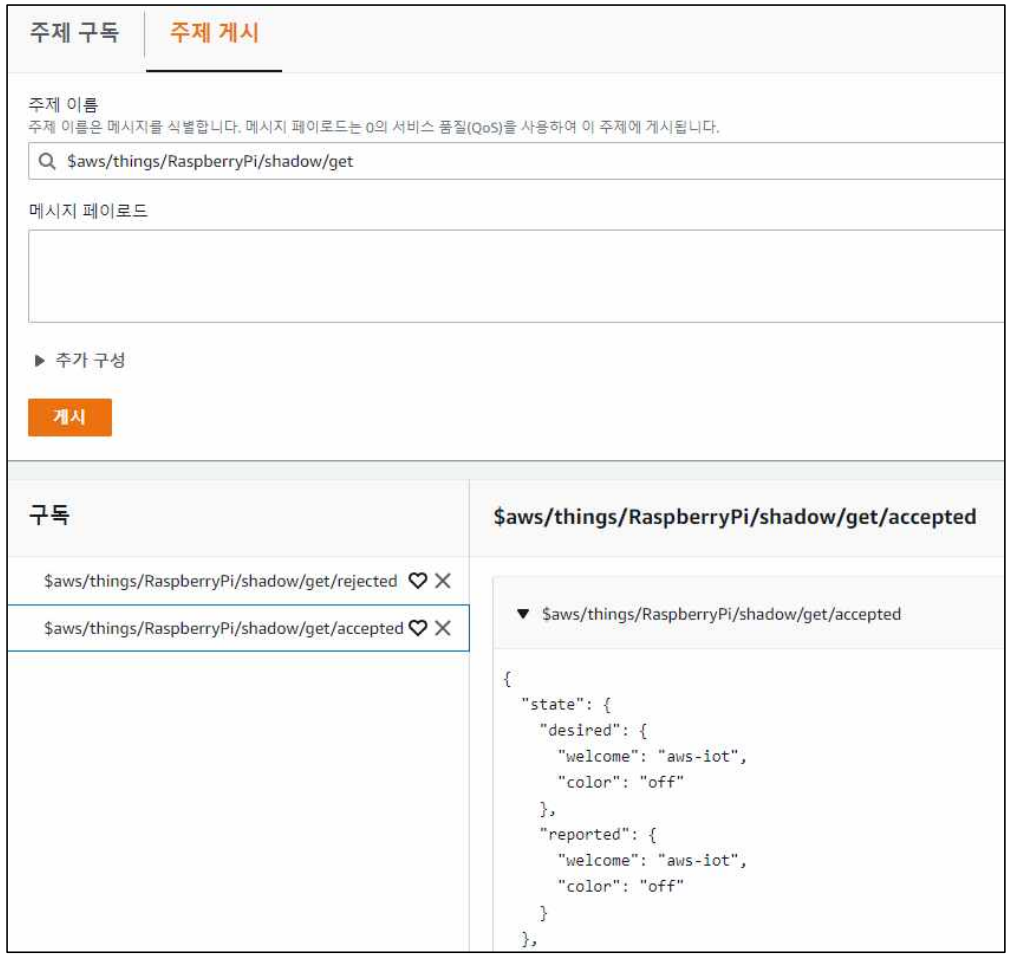
AWS 콘솔의 디바이스 섀도우 문서 옆의 [MQTT 주제]탭으로 가서 이름 /get 에서 오른쪽 끝의 링크 아이콘을 누르면 이 토픽를 주제로 하는 [MQTT 테스트 클라이언트]가 표시된다

Device Shadow : 섀도우 상태 변경

[게시] 버튼을 누르면
현재 섀도우 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
```



Device Shadow : 섀도우 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"
  }  }  }
```

Device Shadow : 섀도우 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 : 섀도우 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 : 섀도우 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 섀도우 update후 다시 보드에서 프로그램을 다시 실행시켜 LED 동작을 확인해본다

$aws/things/RaspberryPi/shadow/update

메시지 페이로드

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

The End