스마트 센서와 클라우드 연동2

한백전자 기술연구소

HANBACK ELECTRONICS CO.,LTD

학습6

스마트 센서와 클라우드 연동2

- Cloud 연동하여 LED 제어하기
- Cloud 연동하여 인체감지센서 상태 확인하기
- Cloud 연동하여 소리감지센서 상태 확인하기
- Cloud 연동하여 Buzzer 제어하기
- Cloud 연동하여 DC모터 제어하기
- Cloud 연동하여 Step모터 제어하기

1. 프로젝트 생성

- Cloud 와 연동하여 LED를 제어하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client led
pi@raspberrypi:~ $ cd led
pi@raspberrypi:~/led $ ls
build  CMakeCache.txt  cmake_install.cmake  install   README.md  src
cmake  CMakeFiles      CMakeLists.txt       Makefile  run.sh     thinger
pi@raspberrypi:~/led $
```

1. LED 제어 프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
//GPIO Information
#define RED_LED_PIN 7
#define GREEN_LED_PIN 21
#define BLUE_LED_PIN 22
```

```
int main(int argc, char *argv[]){
   thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
   wiringPiSetup();
   pinMode(RED_LED_PIN,OUTPUT);
   pinMode(BLUE_LED_PIN,OUTPUT);
   pinMode(GREEN_LED_PIN,OUTPUT);
   thing["led_red"] << [](pson& in){
      if(in.is_empty()){
         in = (bool)digitalRead(RED_LED_PIN);
      }else{
         if(in){
            digitalWrite(RED_LED_PIN,HIGH);
         }else{
            digitalWrite(RED_LED_PIN,LOW);
         }
      }
```

1. LED 제어 프로그램 작성

main.cpp

```
};
  thing["led_green"] << [](pson& in){
    if(in.is_empty()){
      in = (bool)digitalRead(GREEN_LED_PIN);
    }else{
      if(in){
        digitalWrite(GREEN_LED_PIN,HIGH);
      }else{
        digitalWrite(GREEN_LED_PIN,LOW);
      }
    }
  };
  thing["led_blue"] << [](pson& in){
    if(in.is_empty()){
      in = (bool)digitalRead(BLUE_LED_PIN);
    }else{
      if(in){
        digitalWrite(BLUE_LED_PIN,HIGH);
      }else{
        digitalWrite(BLUE_LED_PIN,LOW);
      }
    }
  };
  thing.start();
  return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 각각 LED를 제어할 핀 번호를 변수로 선언한다.

```
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
#define RED_LED_PIN 7
#define GREEN_LED_PIN 21
#define BLUE_LED_PIN 22
```

3) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. LED 는 출력을 제어해야 함으로 GPIO의 설정을 OUTPUT 으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
wiringPiSetup();
pinMode(RED_LED_PIN,OUTPUT);
pinMode(BLUE_LED_PIN,OUTPUT);
pinMode(GREEN_LED_PIN,OUTPUT);
```

4) LED 등록 및 제어

- 제어할 LED를 등록한다. thing["led_red"] 라고 선언을 하게 되면 Cloud 에 led_red 라는 액츄에이터가 등록되게 된다. Cloud 로부터 전달된 데이터는 in 이라는 변수에 저장되고 이 변수에 전달된 데이터에 따라 LED를 켜거나 끄는 작업을 수행한다.

```
thing["led_red"] << [](pson& in){
  if(in.is_empty()){
    in = (bool)digitalRead(RED_LED_PIN);
  }else{
    if(in){
      digitalWrite(RED_LED_PIN,HIGH);
    }else{
      digitalWrite(RED_LED_PIN,LOW);
    }
  }
  };
thing["led_green"] << [](pson& in){
  if(in.is_empty()){
    in = (bool)digitalRead(GREEN_LED_PIN);
  }else{
    if(in){
      digitalWrite(GREEN_LED_PIN,HIGH);
    }else{
      digitalWrite(GREEN_LED_PIN,LOW);
    }
  }
};
thing["led_blue"] << [](pson& in){
  if(in.is_empty()){
    in = (bool)digitalRead(BLUE_LED_PIN);
  }else{
    if(in){
      digitalWrite(BLUE_LED_PIN,HIGH);
    }else{
      digitalWrite(BLUE_LED_PIN,LOW);
    }
  }
};
```

컴파일을 위한 수정

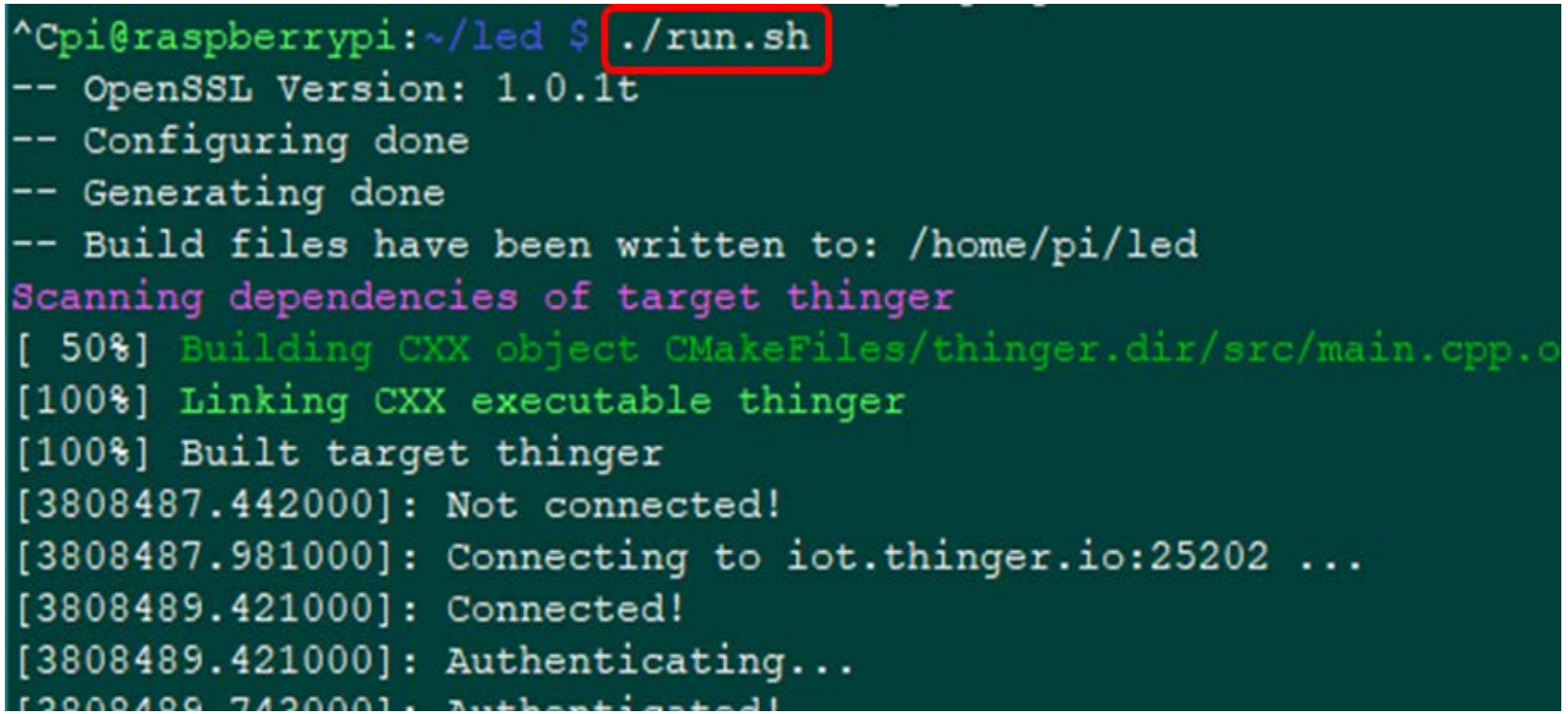
1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

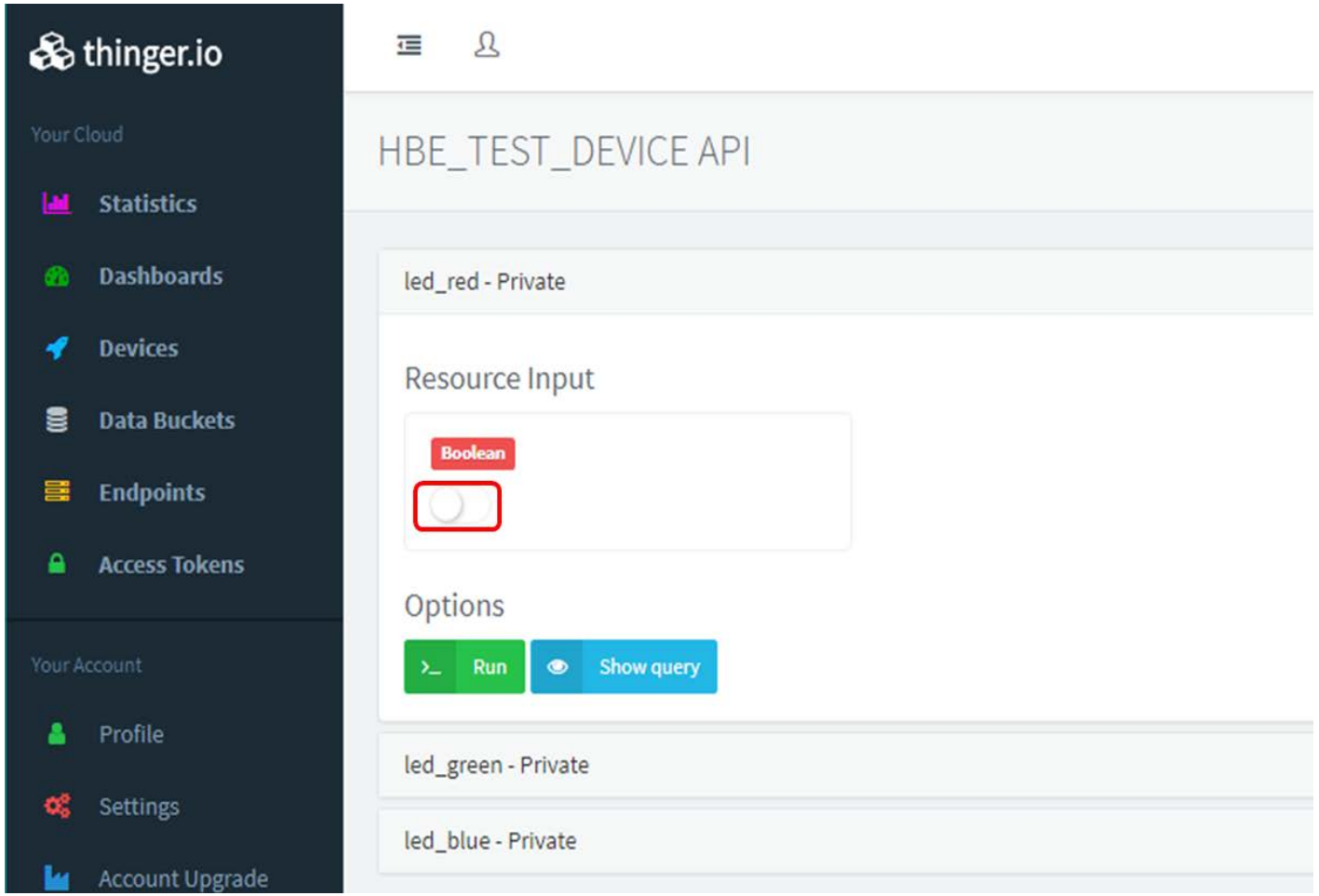
1. 프로그램 컴파일 (in Raspberry Pi)

- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.



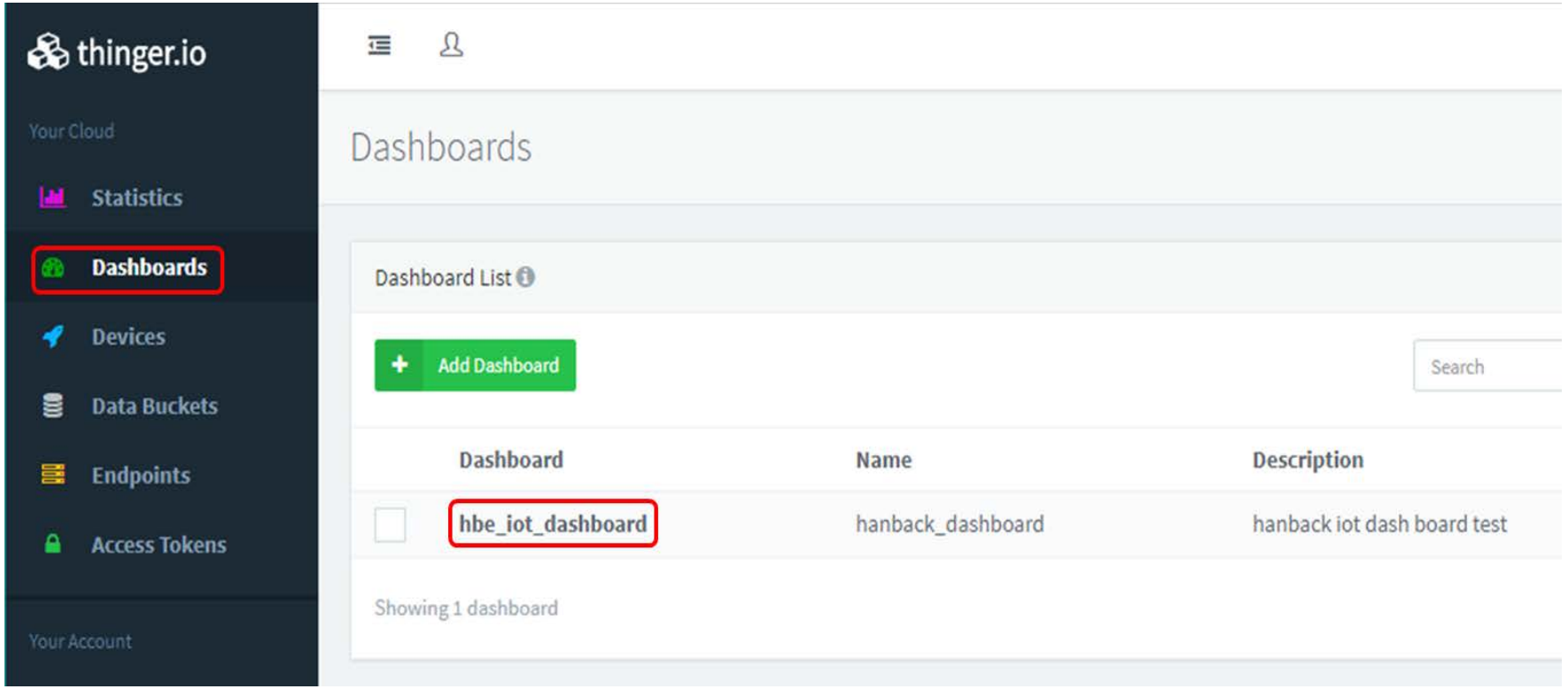
결과 확인

- 웹페이지에서 Device 정보 내에 Device API 부분을 확인해 보면 코드에 작성한 led 관련 내용들이 등록되어 있는 것을 확인 할 수 있다. 각 led 의 이름을 누르면 아래 그림과 같이 버튼이 나타나는 것을 확인해 볼 수 있다.



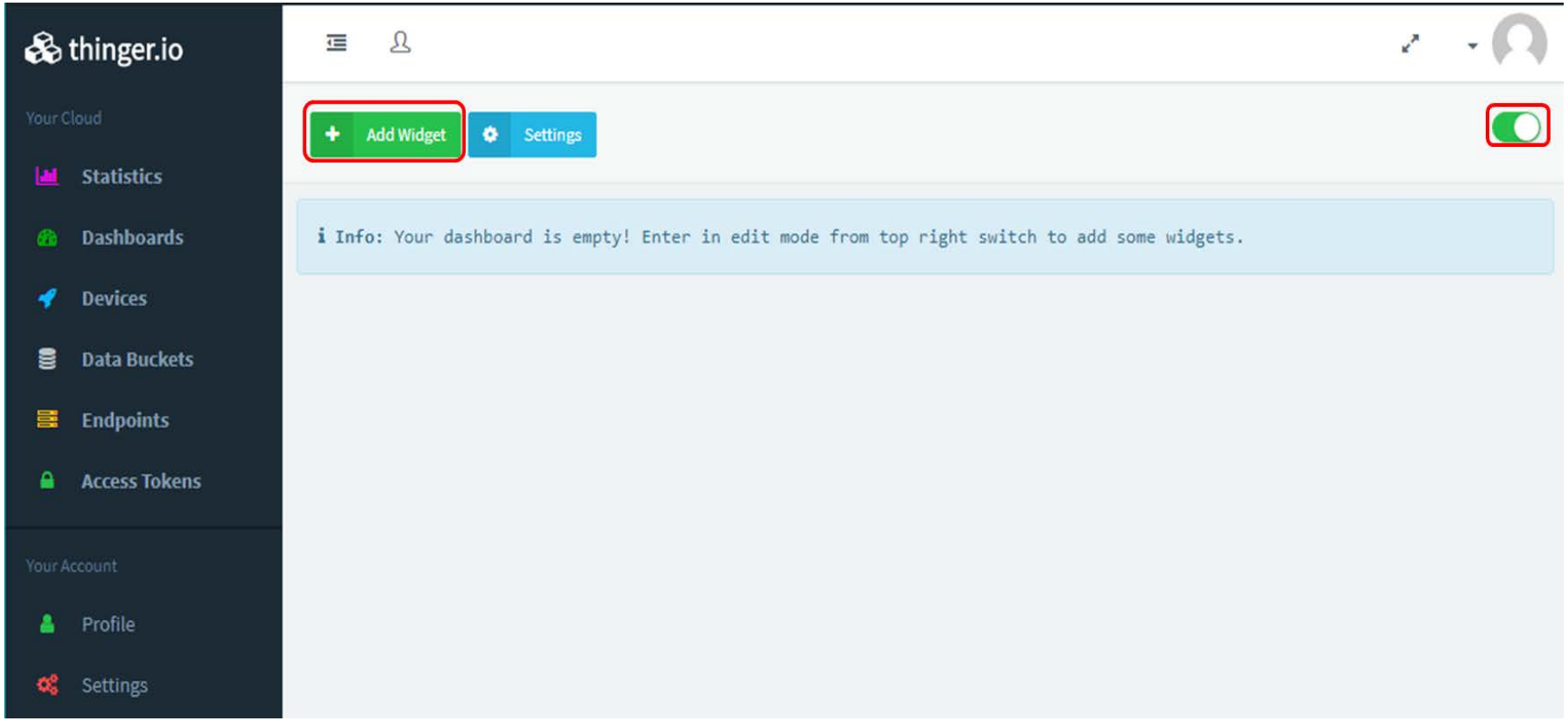
결과 확인

- 웹페이지 좌측 메뉴에서 Dashboards 메뉴를 선택하면 앞서 생성한 대시보드를 확인할 수 있다. 여기서 대시보드의 이름을 누르게 되면 대시보드에 설정된 화면을 볼 수 있다.



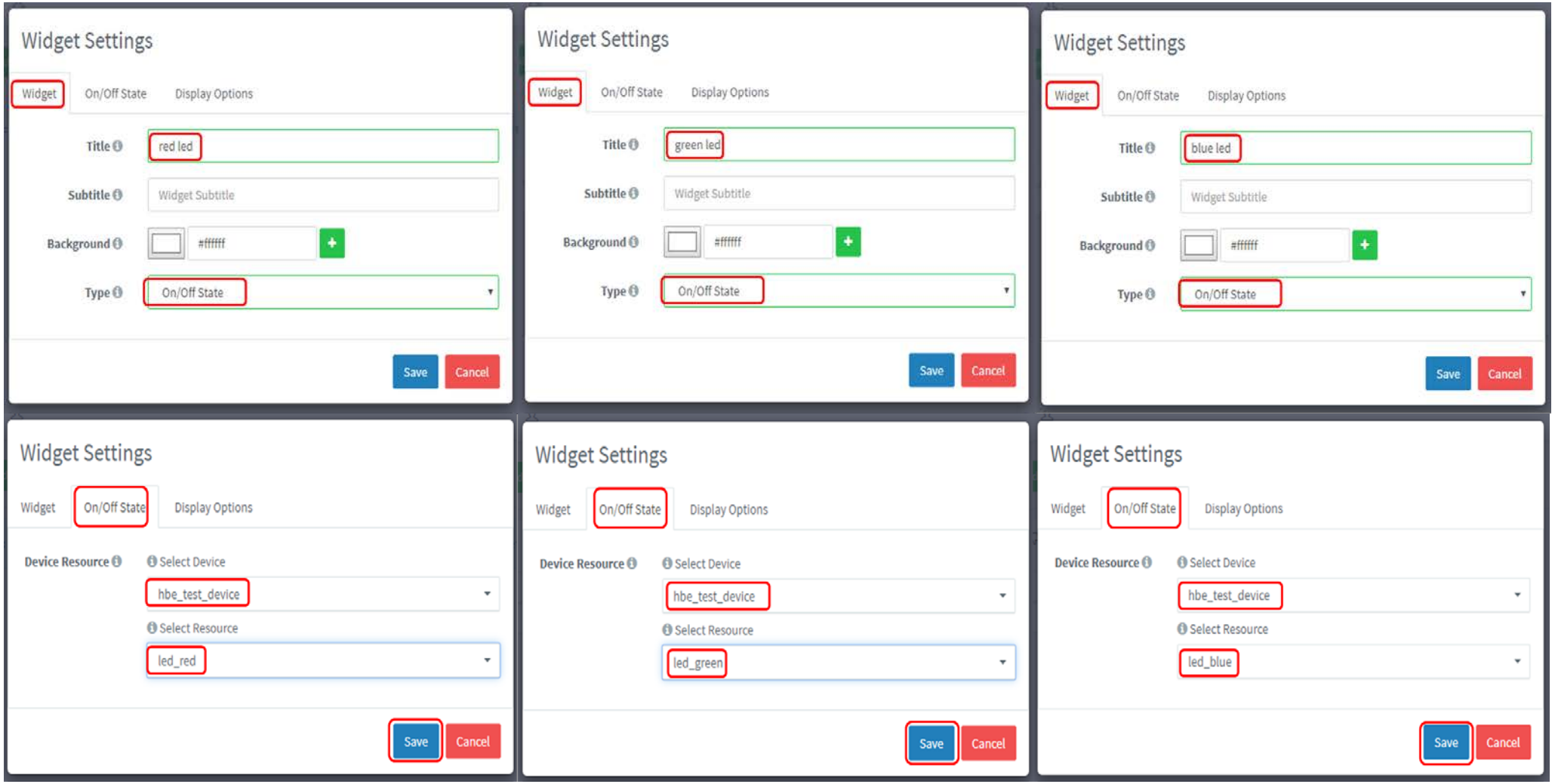
결과 확인

- 우측 상단에 버튼을 활성화 하고 좌측에 있는 Add Widget 버튼을 눌러 LED 제어에 사용할 Widget을 추가한다.



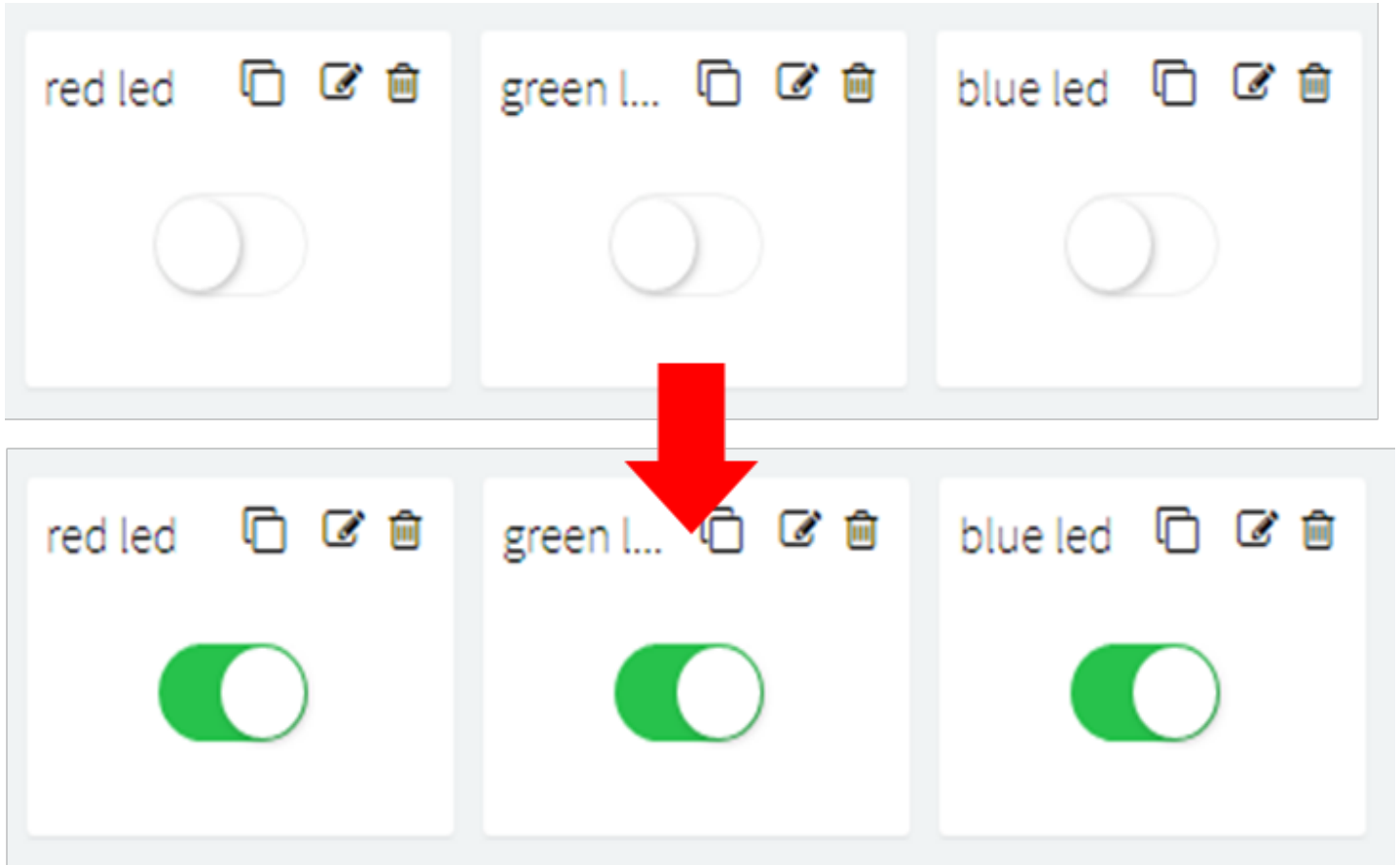
결과 확인

HANBACK ELECTRONICS.,LTD



결과 확인

- Widget을 설정하고 나면 대시보드에 Widget 이 추가 된다.
  Widget 추가 시 배경 색상들은 원하는 대로 지정하여 사용하면 된다.
  생성된 Widget의 버튼을 누르면 각각의 LED 가 제어되는 것을 확인할 수 있다.



1. 프로젝트 생성

- Cloud 와 연동하여 인체감지센서의 상태를 확인하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client pir
pi@raspberrypi:~ $ cd pir/
pi@raspberrypi:~/pir $
```

1. 인체감지 프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"

//GPIO Information
#define PIR_PIN 2
```

```
int main(int argc, char *argv[])
{               thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
                wiringPiSetup();
                pinMode(PIR_PIN,INPUT);
//Sensor & Actuator Caontrol
                thing["pir"] >> [](pson& out){
                                out = digitalRead(PIR_PIN);
                };
                thing.start();
                return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 사용자 정보와 디바이스 정보 그리고 PIR 센서의 상태를 확인하기 위한 핀 번호를 변수로 선언한다.

```
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"


#define PIR_PIN 2
```

3) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. 인체 감지 센서는 센서의 데이터를 입력을 받아야 함으로 GPIO의 설정을 INPUT 으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
wiringPiSetup();
pinMode(PIR_PIN,INPUT);
```

4) 인체감지센서 등록 및 데이터 송신

- 인체감지 센서를 등록한다. thing["pir"] 라고 선언을 하게 되면 Cloud 에 pir 라는 센서가 등록되게 된다. Cloud 로 전송할 데이터는 out 이라는 변수에 전달하면 대시보드에 설정된 주기에 따라 데이터를 Cloud 로 전송하게 된다.

```
thing["pir"] >> [](pson& out){
        out = digitalRead(PIR_PIN);
};
```

컴파일을 위한 수정

1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

1. 프로그램 컴파일 (in Raspberry Pi)

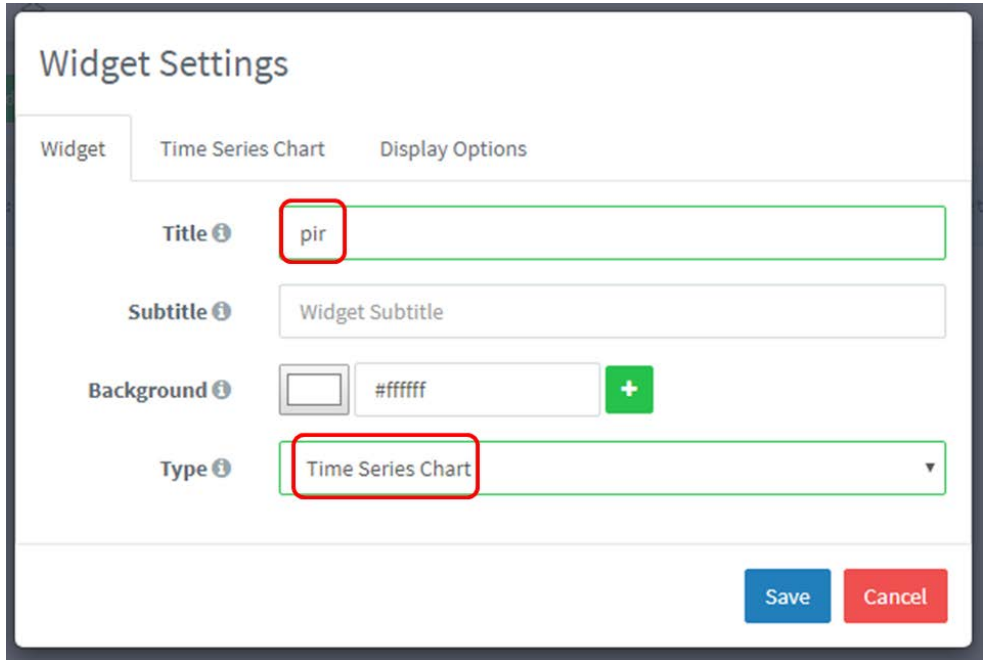
- “./run.sh” 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

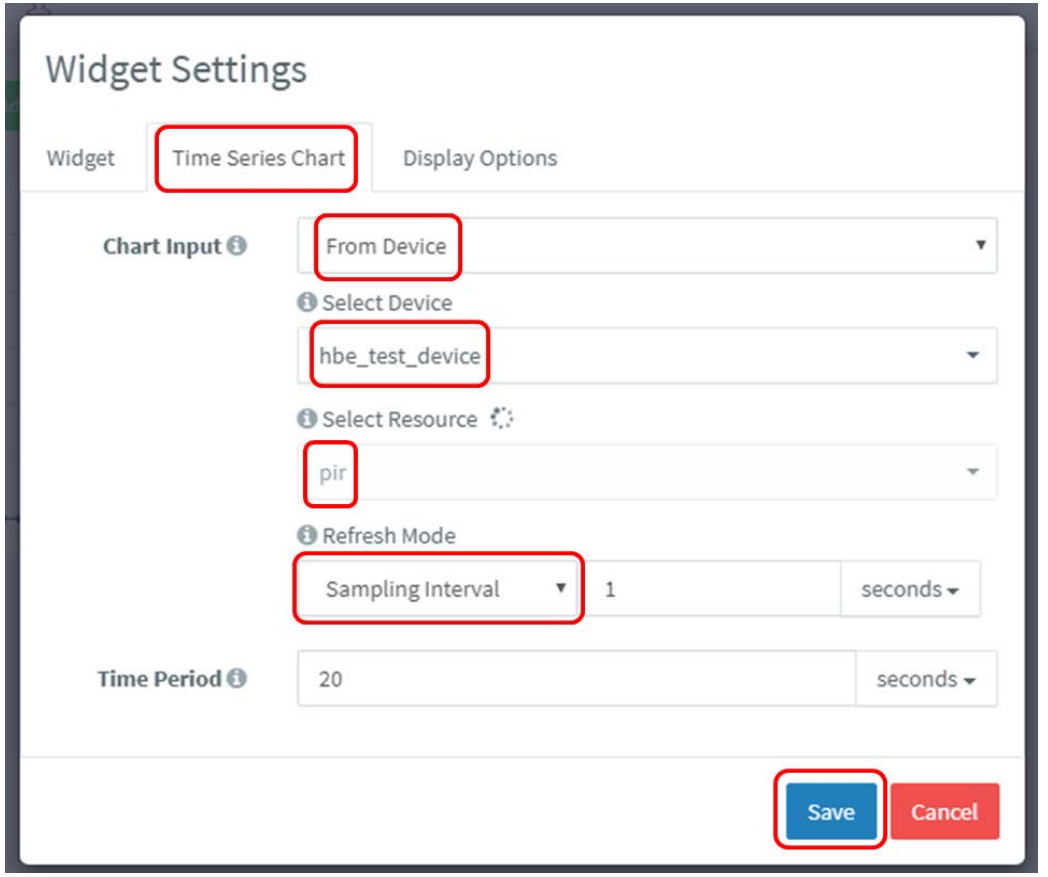
```
pi@raspberrypi:~/pir $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/pir
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3811746.262000]: Not connected!
[3811746.771000]: Connecting to iot.thinger.io:25202 ...
[3811748.211000]: Connected!
[3811748.212000]: Authenticating...
[3811748.538000]: Authenticated!
```

결과 확인

HANBACK ELECTRONICS.,LTD

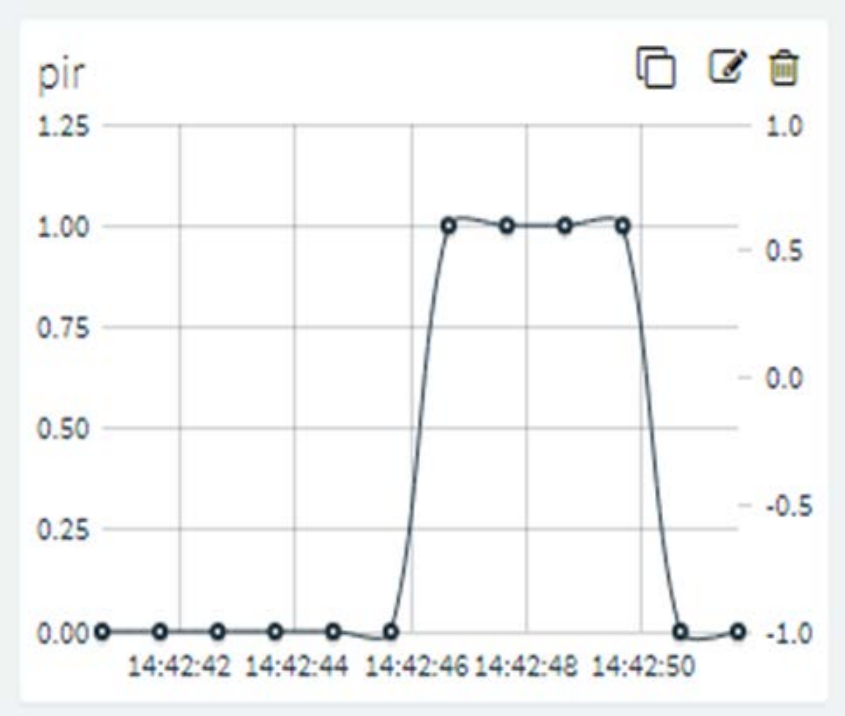
- Dashboard에서 Add Widget을 눌러 인체감지센서를 모니터링 할 Widget을 추가한다.





결과 확인

- 여기서 Widget 설정 시 Time Series Chart를 선택하였는데 이 메뉴는 시간대 별로 전달된 데이터를 그래프 형태로 표기해주는 메뉴이다. 센서 수집주기는 Refresh Mode에서 설정이 가능한대 시간을 초 단위부터 년 단위까지 자유롭게 설정하여 확인해 볼 수 있다.



1. 프로젝트 생성

- Cloud 와 연동하여 소리 감지 센서의 상태를 확인하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client sound
pi@raspberrypi:~ $ cd sound/
pi@raspberrypi:~/sound $
```

1. 소리감지프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"

//SPI
#define SPI_CH 0
#define SPI_SPEED 500000
```

```
//ADC Control PIN
#define ADC_CS          8
#define ADC_MISO        13
#define ADC_MOSI        12
#define ADC_SCLK        14

//ADC Channel
#define SOUND_CH        2
```

1. 소리감지 프로그램 작성

main.cpp

```
//ADC control
int mcp3208(int channel){
        int adc_value=0;
        unsigned char buf[3];
                buf[0] = 0x06 | ((channel & 0x07)>>2);
        buf[1] = ((channel & 0x07)<<6);
        buf[2] = 0x00;
        digitalWrite(ADC_CS,LOW);
        wiringPiSPIDataRW(SPI_CH,buf,3);
        buf[1] = 0x0F & buf[1];
        adc_value = (buf[1] << 8) | buf[2];
        digitalWrite(ADC_CS,HIGH);
        return adc_value;
}
```

```
int main(int argc, char *argv[])
{       thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
        wiringPiSetup();
        pinMode(ADC_CS,OUTPUT);
        if(wiringPiSPISetup(SPI_CH,SPI_SPEED) == -1){
                printf("wiringPi SPI Setup Failed!₩n");
                exit(0);
        }
//Sensor & Actuator Caontrol
   thing["sound"] >> [](pson& out){
                out = mcp3208(SOUND_CH);
        };
        thing.start();
        return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 사용자 정보와 디바이스 정보 그리고 소리감지센서의 상태를 확인하기 위한 핀 번호를 변수로 선언한다. 또한 SPI 통신을 위한 변수를 선언한다.

```
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"

#define SPI_CH 0
#define SPI_SPEED 500000

#define ADC_CS               8
#define ADC_MISO             13
#define ADC_MOSI             12
#define ADC_SCLK             14

#define SOUND_CH             2
```

3) 아날로그 데이터 수신 함수 선언

- SPI 통신을 통해 아날로그 데이터를 수신한다. 데이터 송신 및 수신시 CS 핀을 HIGH -> LOW 로 변경하고 데이터를 전달하고 수신이 완료되면 CS 핀을 LOW -> HIGH 로 변경한다.

```
int mcp3208(int channel){
        int adc_value=0;
        unsigned char buf[3];
                buf[0] = 0x06 | ((channel & 0x07)>>2);
        buf[1] = ((channel & 0x07)<<6);
        buf[2] = 0x00;
        digitalWrite(ADC_CS,LOW);
        wiringPiSPIDataRW(SPI_CH,buf,3);
        buf[1] = 0x0F & buf[1];
        adc_value = (buf[1] << 8) | buf[2];
        digitalWrite(ADC_CS,HIGH);
        return adc_value;
}
```

4) LED 등록 및 제어

- Cloud 와 연결을 위한 초기화를 수행한다. 그리고 wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. SPI 통신을 사용할 때 CS 핀은 HIGH & LOW를 지속적으로 변경해야 되기 때문에 GPIO 설정을 OUTPUT으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);

wiringPiSetup();

pinMode(ADC_CS,OUTPUT);

if(wiringPiSPISetup(SPI_CH,SPI_SPEED) == -1){

        printf("wiringPi SPI Setup Failed!₩n");

        exit(0);

}
```

5) 센서 등록 및 데이터 송신

- 센서를 등록한다. thing["sound"] 라고 선언을 하게 되면 Cloud 에 sound 라는 센서가 등록되게 된다. Cloud 로 전송할 데이터는 out 이라는 변수에 전달하면 대시보드에 설정된 주기에 따라 데이터를 Cloud 로 전송하게 된다.

```
thing["sound"] >> [](pson& out){

        out = mcp3208(SOUND_CH);

};
```

컴파일을 위한 수정

1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

1. 프로그램 컴파일 (in Raspberry Pi)

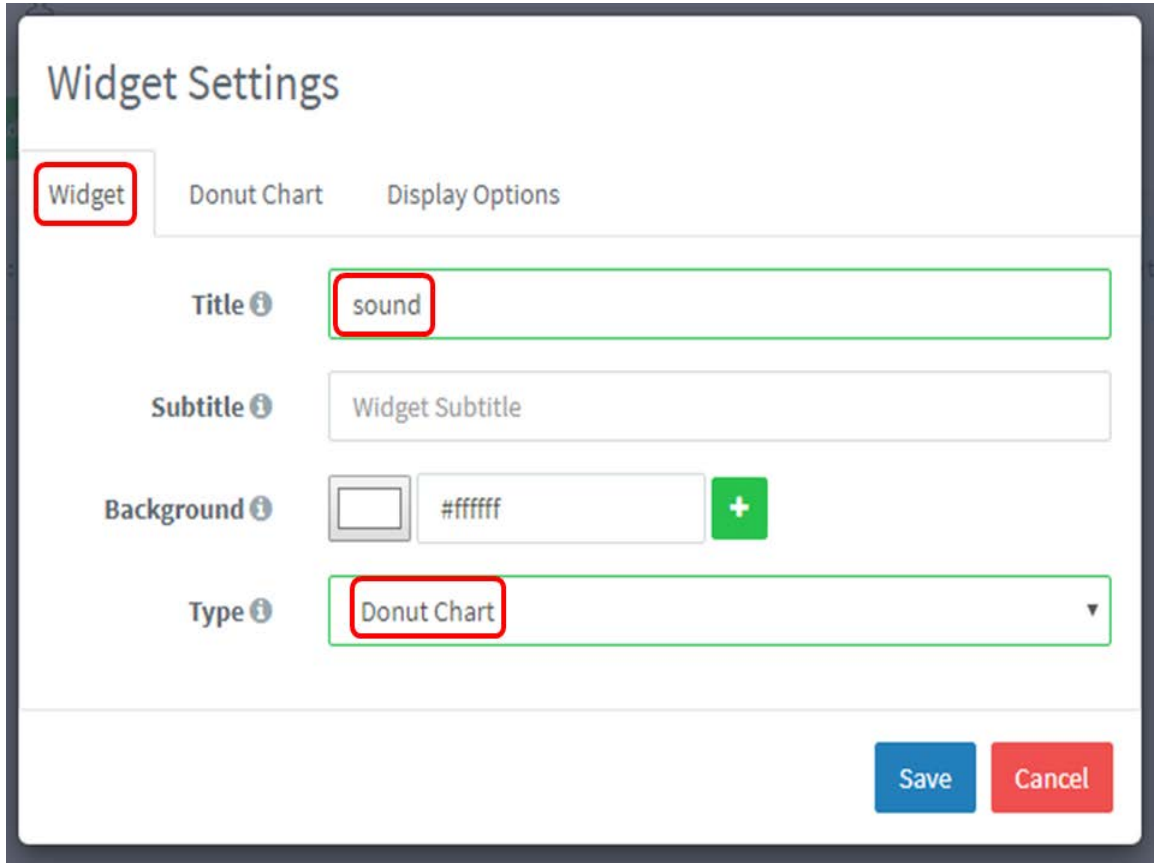
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

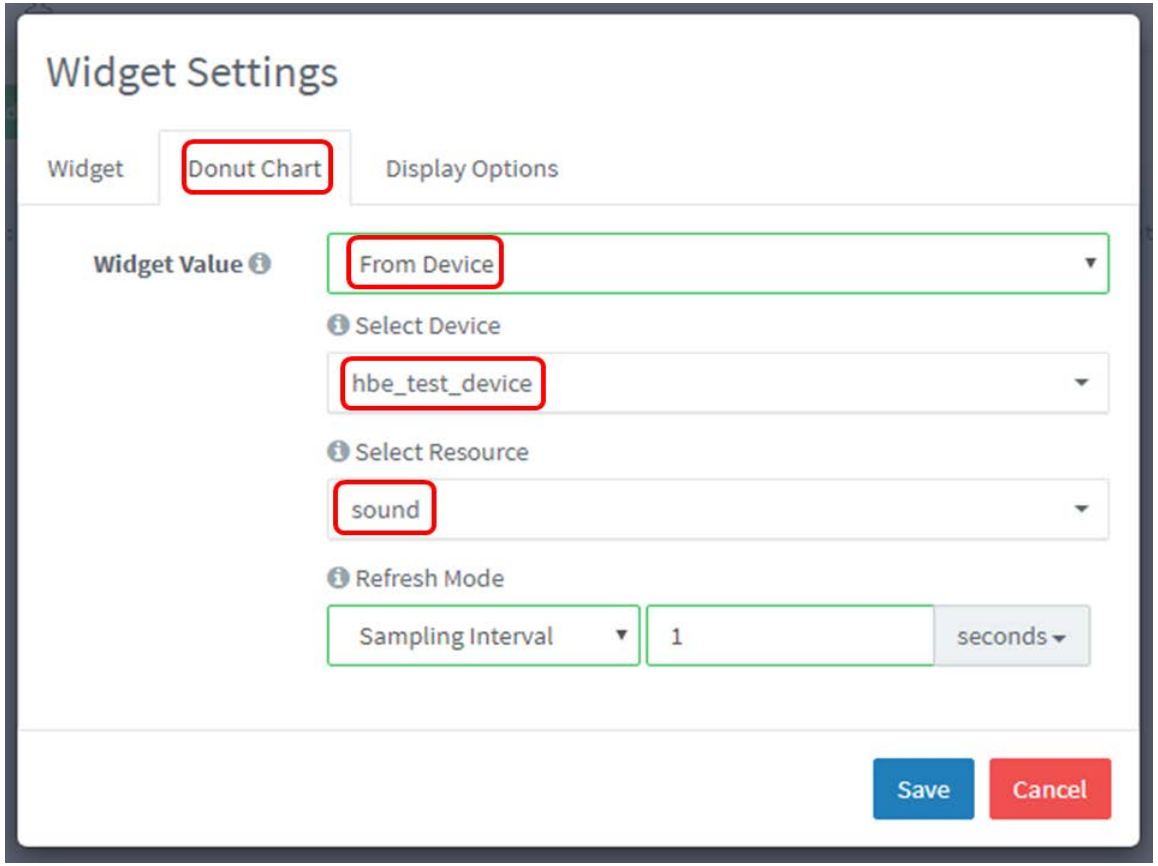
```
^Cpi@raspberrypi:~/sound $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/sound
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3812453.062000]: Not connected!
[3812453.424000]: Connecting to iot.thinger.io:25202 ...
[3812454.883000]: Connected!
[3812454.883000]: Authenticating...
```

결과 확인

HANBACK ELECTRONICS.,LTD

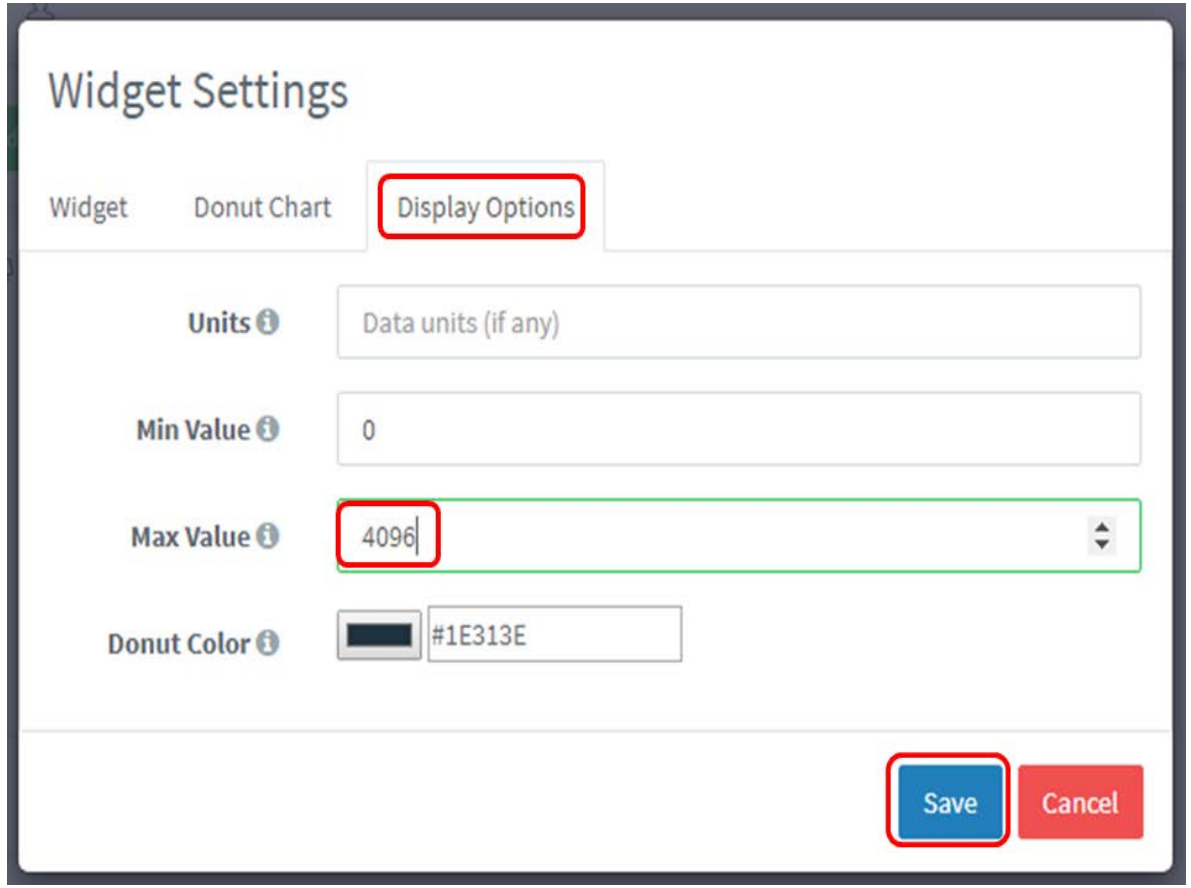
- 라즈베리파이에서 프로그램이 실행되고 나면 Cloud 에서 소리감지센서 상태 확인을 위해 대시보드 설정을 한다. 아래 그림과 같이 Add Widget을 눌러 소리감지 센서를 모니터링 할 Widget을 추가한다.

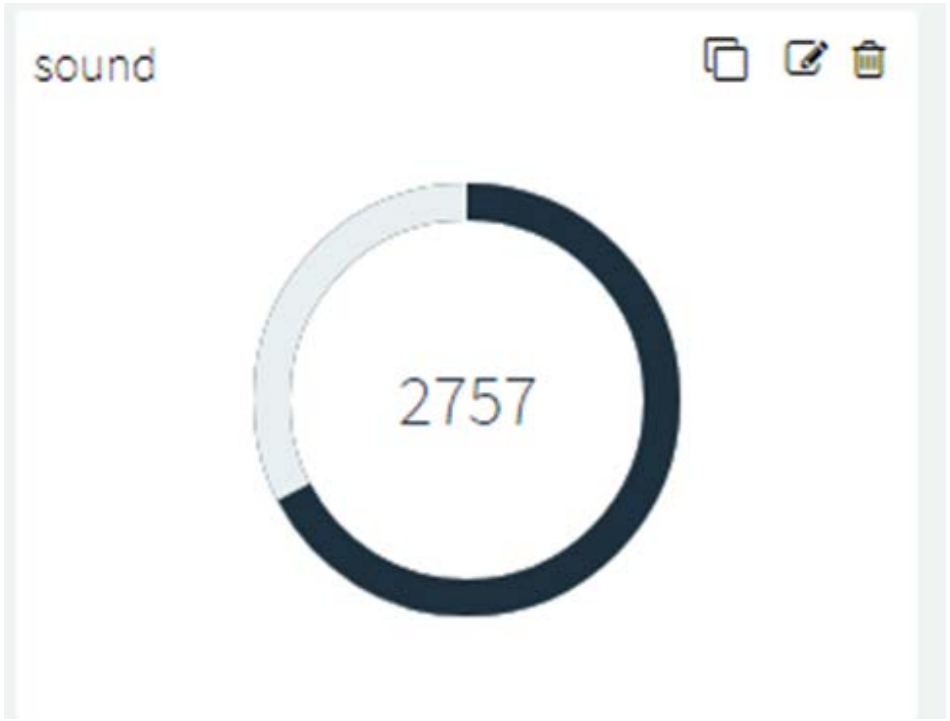




결과 확인

- Widget 설정 시 Display Option에 Max Value를 4096 으로 설정한 이유는
  ADC 칩이 12비트 ADC의 결과를 나타내기 때문에 최대로 나타날 수 있는 값이 4096 이다.
  따라서 설정을 위와 같이 수행하고 저장하면 센서 데이터가 변화하는 것을 확인할 수 있다.





1. 프로젝트 생성

- Cloud 와 연동하여 Buzzer를 제어하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client buzzer
pi@raspberrypi:~ $ cd buzzer/
pi@raspberrypi:~/buzzer $
```

1. Buzzer 제어 프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"

//GPIO Information
#define BUZZER_PIN 15
```

Buzzer 제어 프로그램 작성


```
int main(int argc, char *argv[])
{               thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
                wiringPiSetup();
                pinMode(BUZZER_PIN,OUTPUT);
                //Sensor & Actuator Caontrol
                thing["buzzer"] << [](pson& in){
                                if(in.is_empty()){
                                                in = (bool)digitalRead(BUZZER_PIN);
                                }else{
                                                if(in){
                                                                digitalWrite(BUZZER_PIN,HIGH);
                                                }else{
                                                                digitalWrite(BUZZER_PIN,LOW);
                                                }
                                }
                };
                thing.start();
                return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 사용자 정보와 디바이스 정보 그리고 각 Buzzer 를 제어할 핀 번호를 변수로 선언한다.

```
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"


#define BUZZER_PIN 15
```

3) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. 그리고 wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. Buzzer는 출력을 제어해야 하므로 GPIO의 설정을OUTPUT 으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
wiringPiSetup();
pinMode(RED_LED_PIN,OUTPUT);
pinMode(BLUE_LED_PIN,OUTPUT);
pinMode(GREEN_LED_PIN,OUTPUT);
```

4) 등록 및 제어

- 제어할 Buzzer를 등록한다. thing["buzzer"] 라고 선언을 하게 되면 Cloud 에 buzzer 라는 액츄에이터가 등록되게 된다. Cloud 로부터 전달된 데이터는 in 이라는 변수에 저장되고 이 변수에 전달된 데이터에 따라 Buzzer 를 켜거나 끄는 작업을 수행한다. 이때 digitalWrite 함수를 이용하여 GPIO 설정은 HIGH 또는 LOW 로 제어한다.

```
thing["buzzer"] << [](pson& in){
        if(in.is_empty()){
                in = (bool)digitalRead(BUZZER_PIN);
        }else{
                if(in){
                        digitalWrite(BUZZER_PIN,HIGH);
                }else{
                        digitalWrite(BUZZER_PIN,LOW);
                }
        }
};
```

컴파일을 위한 수정

1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

1. 프로그램 컴파일 (in Raspberry Pi)

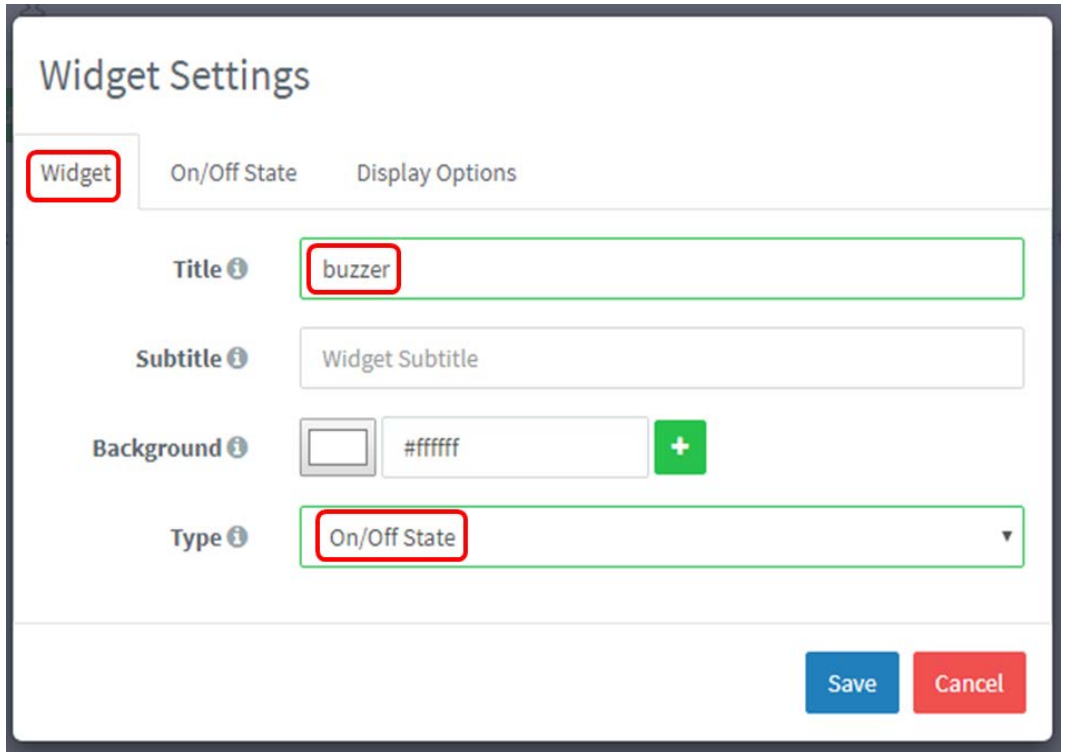
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

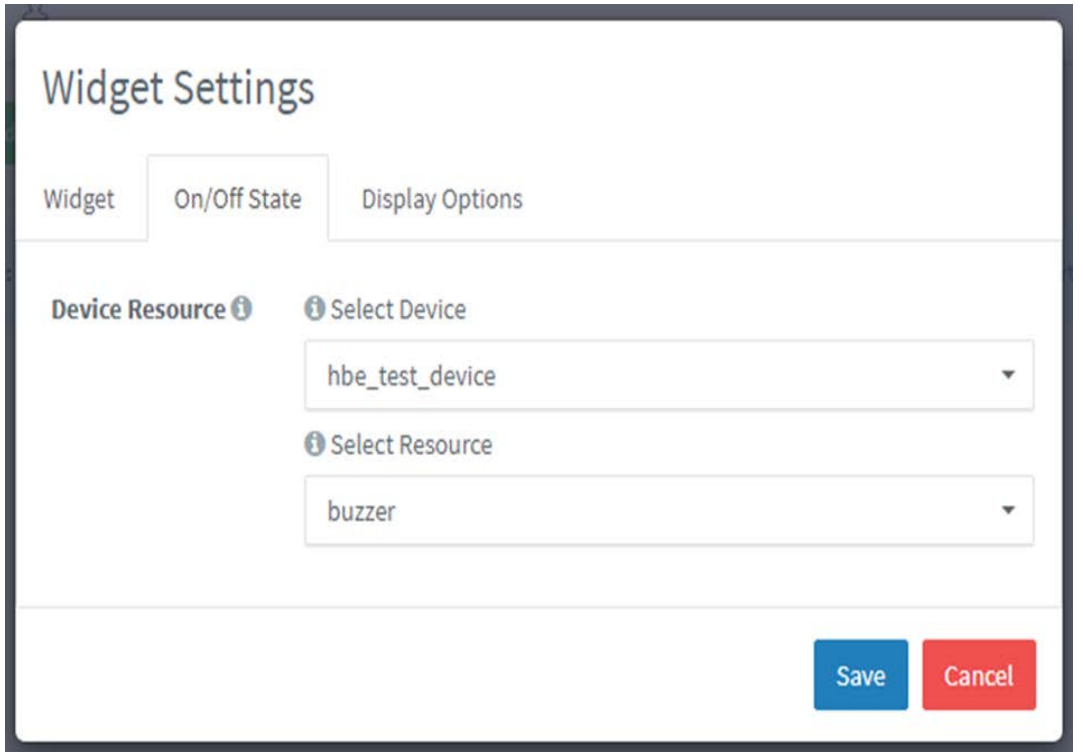
```
pi@raspberrypi:~/buzzer $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/buzzer
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3813543.906000]: Not connected!
[3813545.214000]: Connecting to iot.thinger.io:25202 ...
[3813546.655000]: Connected!
[3813546.655000]: Authenticating...
```

결과 확인

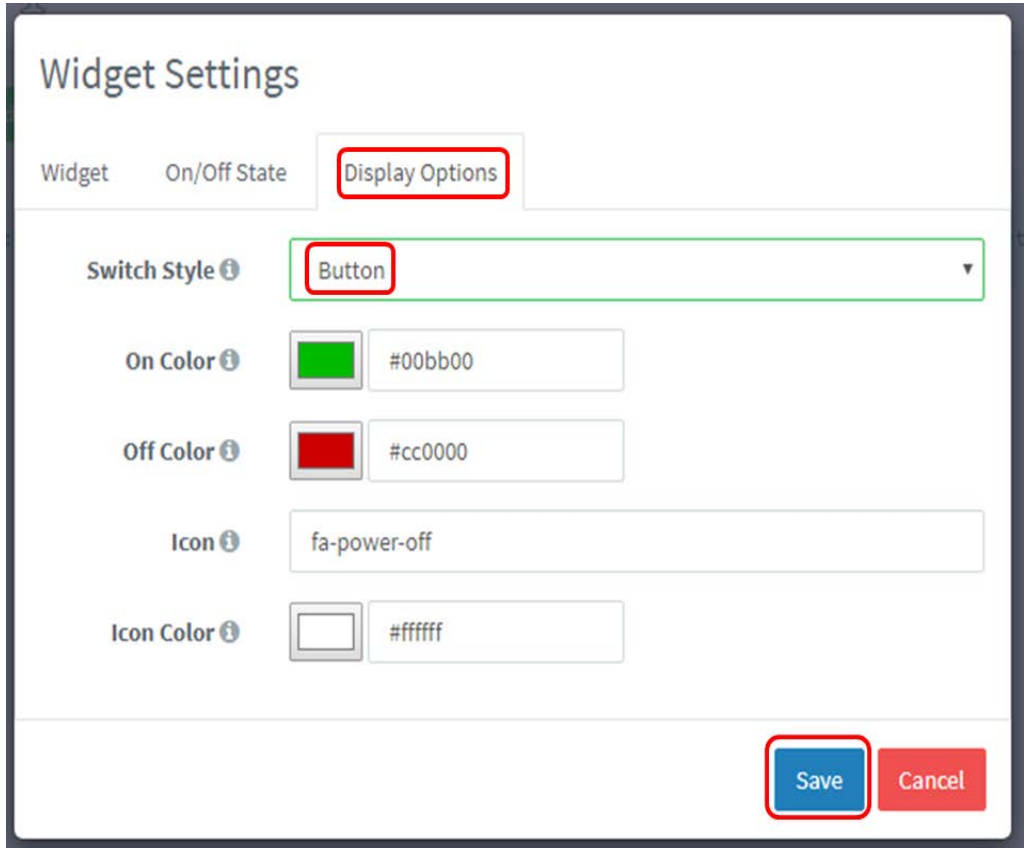
HANBACK ELECTRONICS.,LTD

- 대시보드에서 Buzzer 메뉴를 생성하여 제어 하는 방법은 다음과 같다.
  Dashboard에서 Add Widget을 눌러 Buzzer 를 제어 할 Widget을 추가한다.



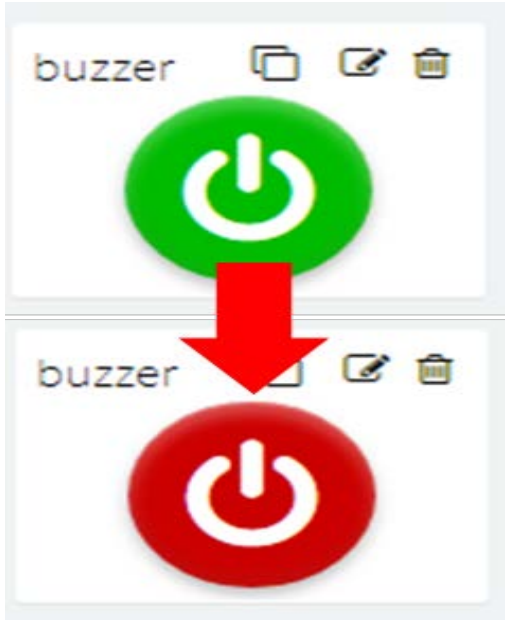


결과 확인


Widget Settings
Widget
On/Off State
Display Options
Switch Style
Button
On Color
#00bb00
Off Color
#cc0000
Icon
fa-power-off
Icon Color
#ffffff
Save
Cancel

결과 확인

- 보드에서 Buzzer 메뉴를 생성하여 제어 하는 방법은 다음과 같다.



1. 프로젝트 생성

- Cloud 와 연동하여 DC 모터를 제어하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client dcmotor
pi@raspberrypi:~ $ cd dcmotor/
pi@raspberrypi:~/dcmotor $
```

1. DC 모터 제어 프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"

//GPIO Information
#define DC_INA_PIN 26
#define DC_INB_PIN 23
```

```
int main(int argc, char *argv[])
{               thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
                wiringPiSetup();
                pinMode(DC_INA_PIN,OUTPUT);
                pinMode(DC_INB_PIN,OUTPUT);
//Sensor & Actuator Caontrol
                thing["dc_forward"] << [](pson& in){
                                digitalWrite(DC_INB_PIN,LOW);
                                if(in.is_empty()){
                                                in = (bool)digitalRead(DC_INA_PIN);
                                }else{
                                                if(in){

                digitalWrite(DC_INA_PIN,HIGH);
```

DC 모터 제어 프로그램 작성

HANBACK ELECTRONICS.,LTD

```
                                        }else{
                                                digitalWrite(DC_INA_PIN,LO
W);
                                        }
                        }
        };
        thing["dc_backward"] << [](pson& in){
                        digitalWrite(DC_INA_PIN,LOW);
                        if(in.is_empty()){
                                in = (bool)digitalRead(DC_INB_PIN);
                        }else{
```

```
                                        if(in){
        digitalWrite(DC_INB_PIN,HIGH);
                                        }else{
        digitalWrite(DC_INB_PIN,LOW);
                                        }
                        }
        };
        thing.start();
        return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 사용자 정보와 디바이스 정보 그리고 DC모터를 제어하기위한 핀 번호를 변수로 선언한다.

```
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"


#define DC_INA_PIN 26
#define DC_INB_PIN 23
```

3) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. DC 모터는 는 출력을 제어해야 함으로 GPIO의 설정을 OUTPUT 으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);

wiringPiSetup();

pinMode(DC_INA_PIN,OUTPUT);

pinMode(DC_INB_PIN,OUTPUT);
```

4) LED 등록 및 제어

- 제어할 DC모터를 등록한다. thing["dc_forward“] 라고 선언을 하게 되면 Cloud 에 dc_for ward 라는 액츄에이터가 등록되게 된다. Cloud 로부터 전달된 데이터는 in 이라는 변수에 저장되고 이 변수에 전달된 데이터에 따라 DC 모터를 정회전 하거나 역회전, 또는 멈추게 하는 작업을 수행한다. 이때 digitalWrite 함수를 이용하여 GPIO 설정은 HIGH 또는 LOW 로 제어한다.

```
thing["dc_forward"] << [](pson& in){
        digitalWrite(DC_INB_PIN,LOW);
        if(in.is_empty()){
                in = (bool)digitalRead(DC_INA_PIN);
        }else{
                if(in){
                        digitalWrite(DC_INA_PIN,HIGH);
                }else{
                        digitalWrite(DC_INA_PIN,LOW);
                }
        }
};
```

```
thing["dc_backward"] << [](pson& in){
        digitalWrite(DC_INA_PIN,LOW);
        if(in.is_empty()){
                in = (bool)digitalRead(DC_INB_PIN);
        }else{
                if(in){
                        digitalWrite(DC_INB_PIN,HIGH);
                }else{
                        digitalWrite(DC_INB_PIN,LOW);
                }
        }
};
```

컴파일을 위한 수정

1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

1. 프로그램 컴파일 (in Raspberry Pi)

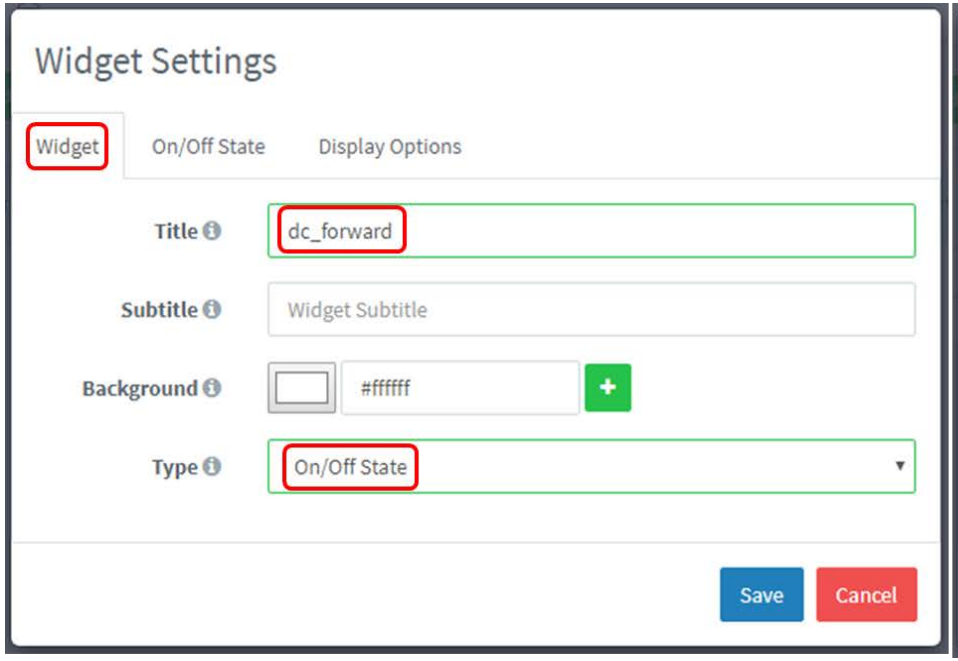
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

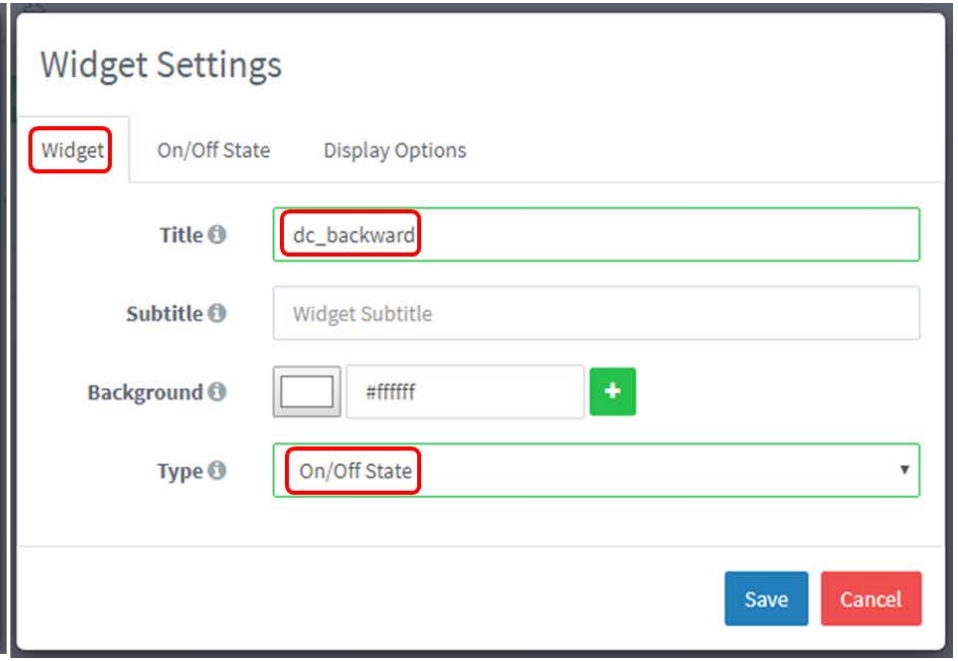
```
pi@raspberrypi:~/dcmotor $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/dcmotor
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3814074.943000]: Not connected!
[3814075.160000]: Connecting to iot.thinger.io:25202 ...
[3814076.636000]: Connected!
[3814076.636000]: Authenticating...
```

결과 확인

HANBACK ELECTRONICS.,LTD

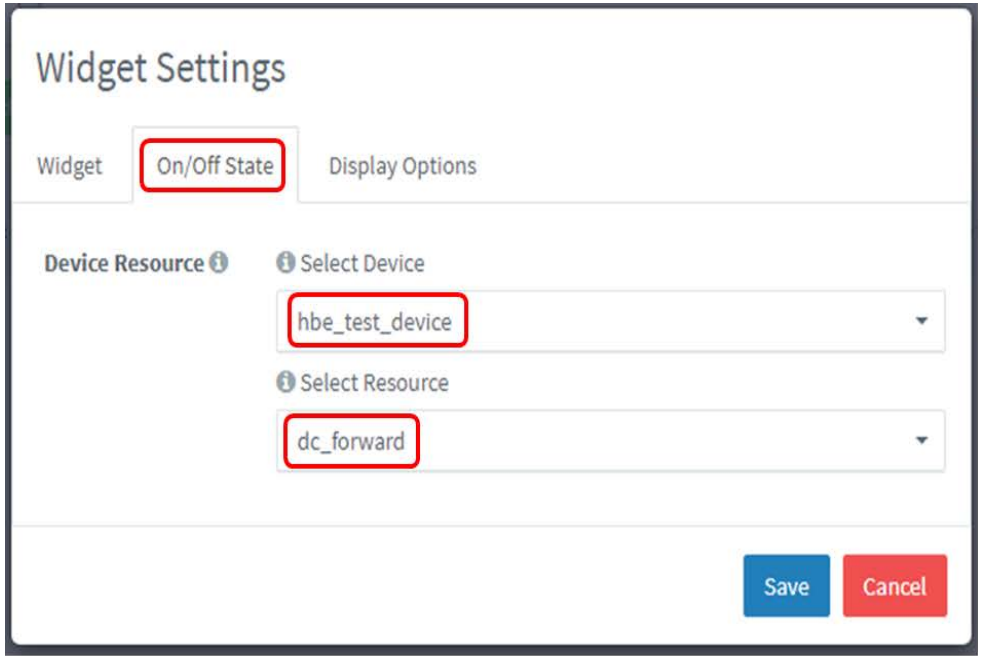
- 라즈베리파이에서 프로그램이 실행되고 나면 Cloud 에서 DC모터 제어를 위해 대시보드 설정을 한다. 대시보드에서 Add Widget을 눌러 Buzzer 를 제어 할 Widget을 추가한다.

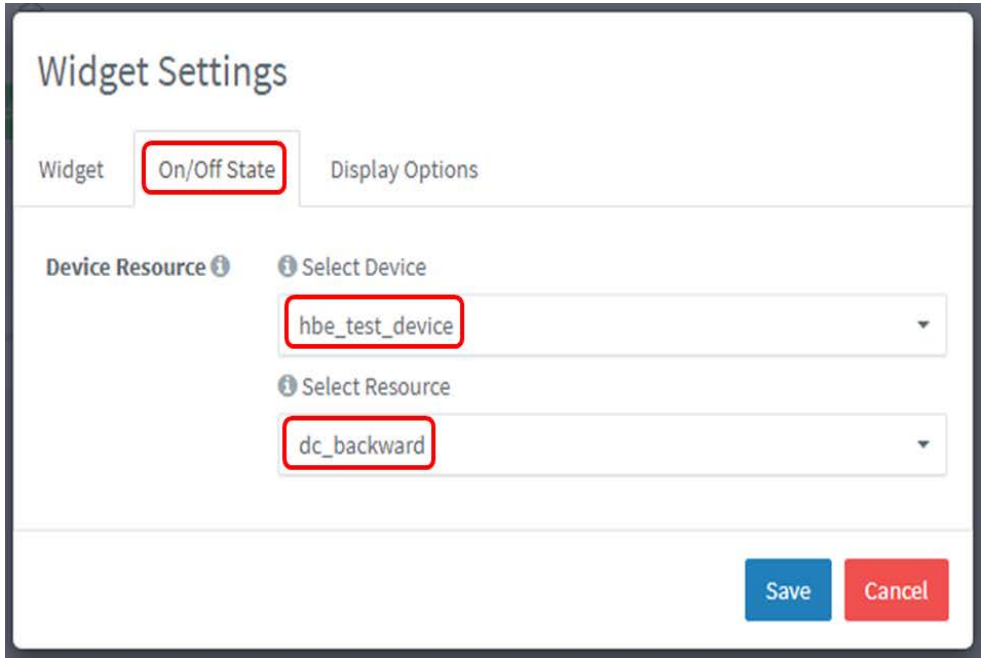




결과 확인

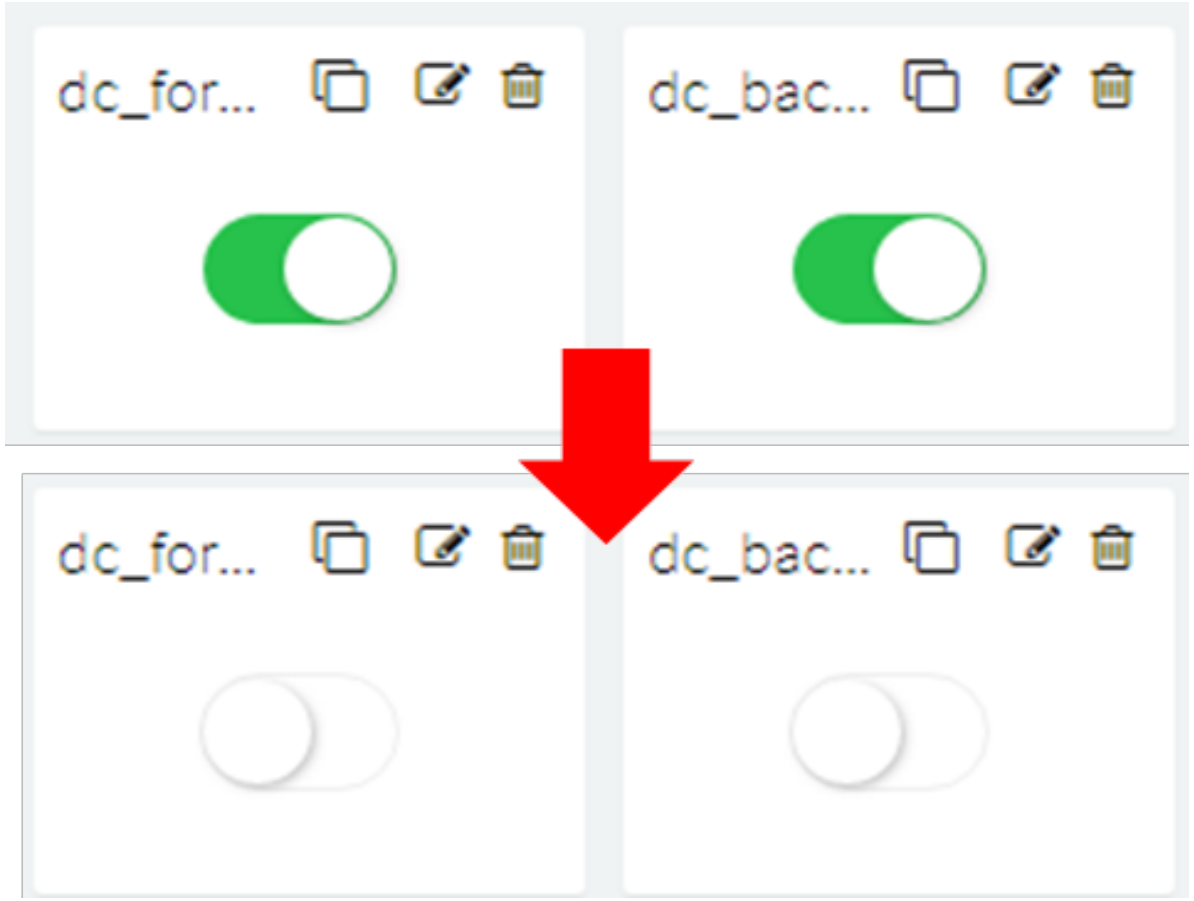
HANBACK ELECTRONICS.,LTD





결과 확인

- Widget 설정이 완료 되면 아래 그림과 같이 2개의 버튼이 생성 된다.
  forward 버튼을 활성화 하면 정회전, backward 버튼을 활성화 하면 역회전하게 된다.



1. 프로젝트 생성

- Cloud 와 연동하여 Step모터를 제어하기 위해 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 및 액츄에이터를 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client stepmotor
pi@raspberrypi:~ $ cd stepmotor/
pi@raspberrypi:~/stepmotor $
```

1. Step 모터 제어 프로그램 작성

main.cpp

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//Thinger.io Connection Information
#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
```

```
//GPIO Information
#define STEP_1A_PIN        27
#define STEP_1B_PIN        0
#define STEP_2A_PIN        1
#define STEP_2B_PIN        24
pthread_t pth_step;
int step_flag=0;
```

Step 모터 제어 프로그램 작성


```
void * step_thread(void *arg){
        while(1){
                if(step_flag){
                        digitalWrite(STEP_1A_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_1A_PIN,LOW);
                        digitalWrite(STEP_1B_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_1B_PIN,LOW);
                        digitalWrite(STEP_2A_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_2A_PIN,LOW);
                        digitalWrite(STEP_2B_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_2B_PIN,LOW);
                }
                usleep(2000);
        }
```

Step 모터 제어 프로그램 작성

HANBACK ELECTRONICS.,LTD

```
int main(int argc, char *argv[])
{               thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
                wiringPiSetup();
                pinMode(STEP_1A_PIN,OUTPUT);
                pinMode(STEP_1B_PIN,OUTPUT);
                pinMode(STEP_2A_PIN,OUTPUT);
                pinMode(STEP_2B_PIN,OUTPUT);
                pthread_create(&pth_step,NULL,step_thread, NULL);
                thing["step"] << [](pson& in){
                                if(in.is_empty()){
                                        in = (bool)step_flag;
                                }else{
                                        if(in){
                                                step_flag=1;
                                        }else{
                                                step_flag=0;
                                        }
                                }
                        };
                        thing.start();
                        return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 입출력 함수를 사용하기 위해 헤더파일을 선언한다.

```
#include "thinger/thinger.h"
#include <wiringPi.h>
#include <wiringPiSPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
```

2) 변수 선언

- 사용자 정보와 디바이스 정보 그리고 Step 모터 제어에 사용할 핀 번호를 변수로 선언한다. 또한 Step 모터 제어 시 생성하여 사용할 Thread 관련 변수도 선언한다.

```
#define USER_ID                "USER_ID"
#define DEVICE_ID              "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"


#define STEP_1A_PIN        27
#define STEP_1B_PIN        0
#define STEP_2A_PIN        1
#define STEP_2B_PIN        24
pthread_t pth_step;
int step_flag=0;
```

3) Step 모터 제어 Thread 선언

- Step 모터를 제어하기 위한 Thread를 생성한다. step_flag 변수의 값에 따라 Step 모터를 회전 시키거나 정지 하는 역할을 한다. 이 때 사용되는 제어 방식은 1상 여자 방식이다.

```
void * step_thread(void *arg){
        while(1){
                if(step_flag){
                        digitalWrite(STEP_1A_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_1A_PIN,LOW);
                        digitalWrite(STEP_1B_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_1B_PIN,LOW);
                        digitalWrite(STEP_2A_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_2A_PIN,LOW);
                        digitalWrite(STEP_2B_PIN,HIGH);
                        usleep(8000);
                        digitalWrite(STEP_2B_PIN,LOW);
                }
                usleep(2000);
        }
}
```

4) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. 그리고 wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. Step 모터는 출력을 제어해야 함으로 GPIO의 설정을 OUTPUT 으로 설정한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
wiringPiSetup();
pinMode(STEP_1A_PIN,OUTPUT);
pinMode(STEP_1B_PIN,OUTPUT);
pinMode(STEP_2A_PIN,OUTPUT);
pinMode(STEP_2B_PIN,OUTPUT);
```

5) Thread 생성

- 앞서 선언한 Step 모터 제어용 Thread를 생성하여 동작 시킨다.

```
pthread_create(&pth_step,NULL,step_thread, NULL);
```

6) 등록 및 제어

- 제어할 Step 모터를 등록한다. thing["step"] 라고 선언을 하게 되면 Cloud 에 step 이라는 액츄에이터가 등록되게 된다. Cloud 로부터 전달된 데이터는 in 이라는 변수에 저장되고 이 변수에 전달된 데이터에 따라 step_flag 의 값을 변경하여 별도로 동작하고 있는 Thread를 통해 Step 모터를 제어한다.

```
thing["step"] << [](pson& in){
        if(in.is_empty()){
                in = (bool)step_flag;
        }else{
                if(in){
                        step_flag=1;
                }else{
                        step_flag=0;
                }
        }
};
```

컴파일을 위한 수정

1. 프로그램 컴파일을 위한 파일 수정(in Raspberry Pi)

- 컴파일을 수행하기 전에 컴파일을 위한 옵션을 추가한다. 수정할 파일의 명칭은 CMakeLists.txt 이다.

```
---------------------------------------- 중 략----------------------------------------
include(CheckCXXCompilerFlag)
CHECK_CXX_COMPILER_FLAG("-std=c++11 -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX1 1)
CHECK_CXX_COMPILER_FLAG("-std=c++0x -lwiringPi -lpthread" COMPILER_SUPPORTS_CXX0 X)
if(COMPILER_SUPPORTS_CXX11)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 -lwiringPi -lpthread")
elseif(COMPILER_SUPPORTS_CXX0X)
                set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x -lwiringPi -lpthread")
else()
    message(STATUS "The compiler ${CMAKE_CXX_COMPILER} has no C++11 support. Please use a different C++ compiler.")
endif()
---------------------------------------- 중 략----------------------------------------
```

1. 프로그램 컴파일 (in Raspberry Pi)

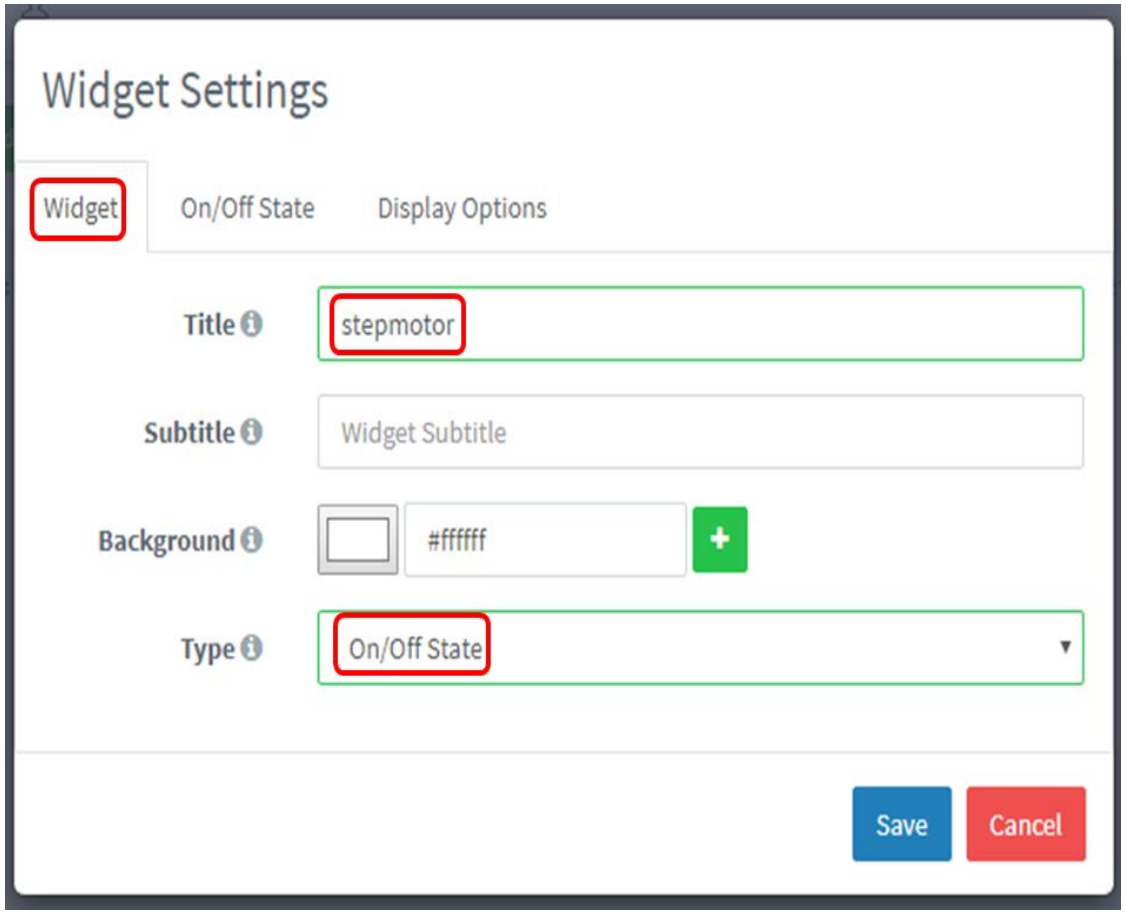
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다. 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다. 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

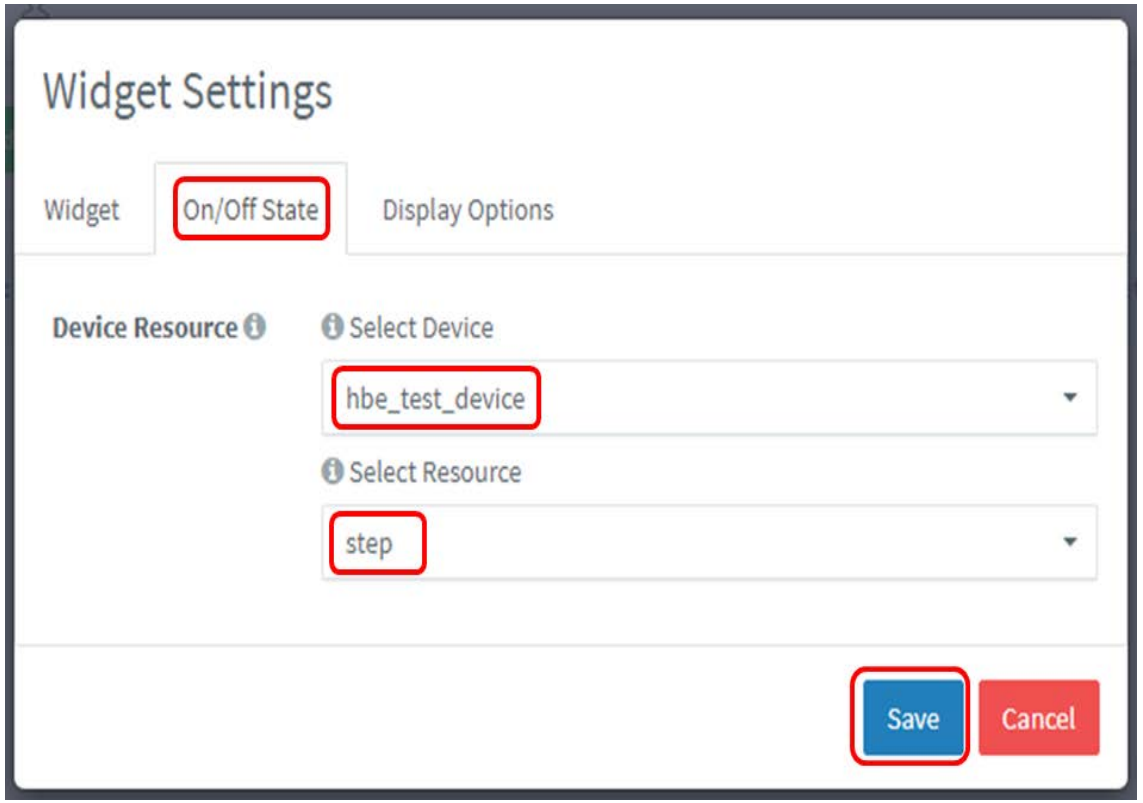
```
pi@raspberrypi:~/stepmotor $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/stepmot
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3814733.747000]: Not connected!
[3814734.374000]: Connecting to iot.thinger.io:25202
[3814735.852000]: Connected!
[3814735.852000]: Authenticating...
```

결과 확인

HANBACK ELECTRONICS.,LTD

- 라즈베리파이에서 프로그램이 실행되고 나면 Cloud 에서 Step모터 제어를 위해 대시보드 설정을 한다.





결과 확인

HANBACK ELECTRONICS.,LTD

- Widget 생성이 완료되면 아래와 같이 Step 모터를 제어할 수 있는 버튼이 생성 된다.
  이 버튼을 누르게 되면 Step 모터가 회전 하거나 멈추는 것을 확인해 볼 수 있다.