스마트 센서와 클라우드 연동3

한백전자 기술연구소

HANBACK ELECTRONICS CO.,LTD

학습6

스마트 센서와 클라우드 연동3

- Cloud 연동하여 Input 모듈 모니터링 하기
- Cloud 연동하여 Light 센서 모니터링하기
- Cloud 연동하여 Ultrasonic 센서 모니터링하기
- Cloud 연동하여 온/습도 센서 모니터링하기
- Cloud 연동하여 가변저항 모니터링 하기

1. 프로젝트 생성

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

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

1. Input 모듈 프로그램 작성

main.cpp

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

#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
#define KEY_UP_PIN          3
#define KEY_DOWN_PIN    4
#define KEY_LEFT_PIN        5
#define KEY_RIGHT_PIN     6
```

```
int main(int argc, char *argv[]){
    thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
    wiringPiSetup();
    pinMode(KEY_1_PIN,INPUT);
    pinMode(KEY_2_PIN,INPUT);
    pinMode(KEY_3_PIN,INPUT);
    pinMode(KEY_4_PIN,INPUT);
    thing["key"] >> [](pson& out){
        out["key1"] = digitalRead(KEY_1_PIN);
        out["key2"] = digitalRead(KEY_2_PIN);
        out["key3"] = digitalRead(KEY_3_PIN);
        out["key4"] = digitalRead(KEY_4_PIN);
    };
    thing.start();
    return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 각종 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

- 키 입력에 사용할 핀 번호와 SPI 통신에 사용할 변수를 선언한다.

```
#define USER_ID             "USER_ID"
#define DEVICE_ID           "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL   "USER_DEVICE_CREDENTIAL"
#define KEY_1_PIN           3
#define KEY_2_PIN           4
#define KEY_3_PIN           5
#define KEY_4_PIN           6
```

3) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다.

```
thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
wiringPiSetup();
pinMode(KEY_1_PIN,INPUT);
pinMode(KEY_2_PIN,INPUT);
pinMode(KEY_3_PIN,INPUT);
pinMode(KEY_4_PIN,INPUT);
```

4) 등록 및 제어

- 모니터링할 센서를 등록한다. thing["key"] 라고 선언을 하게 되면 Cloud 에 key 이라는 센서가 등록되게 되고 out["key1"] 로 선언을 하면 key 센서 내에 key1 이라는 항목이 생성 된다. 이 프로그램에서는 key 라는 센서를 등록하고 key 센서에는 key1, key2, key3, key4 4가지 항목을 등록한다.

```
thing["key"] >> [](pson& out){
        out["key1"] = digitalRead(KEY_1_PIN);
        out["key2"] = digitalRead(KEY_2_PIN);
        out["key3"] = digitalRead(KEY_3_PIN);
        out["key4"] = digitalRead(KEY_4_PIN);
};
```

5) Cloud 연결

- 앞서 설정한 연결 정보를 기반으로 Cloud 와 연결을 시도한다. 설정한 정보가 틀린 내용이 있다면 정상적으로 연결이 되지 않는다.

```
thing.start();
```

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)

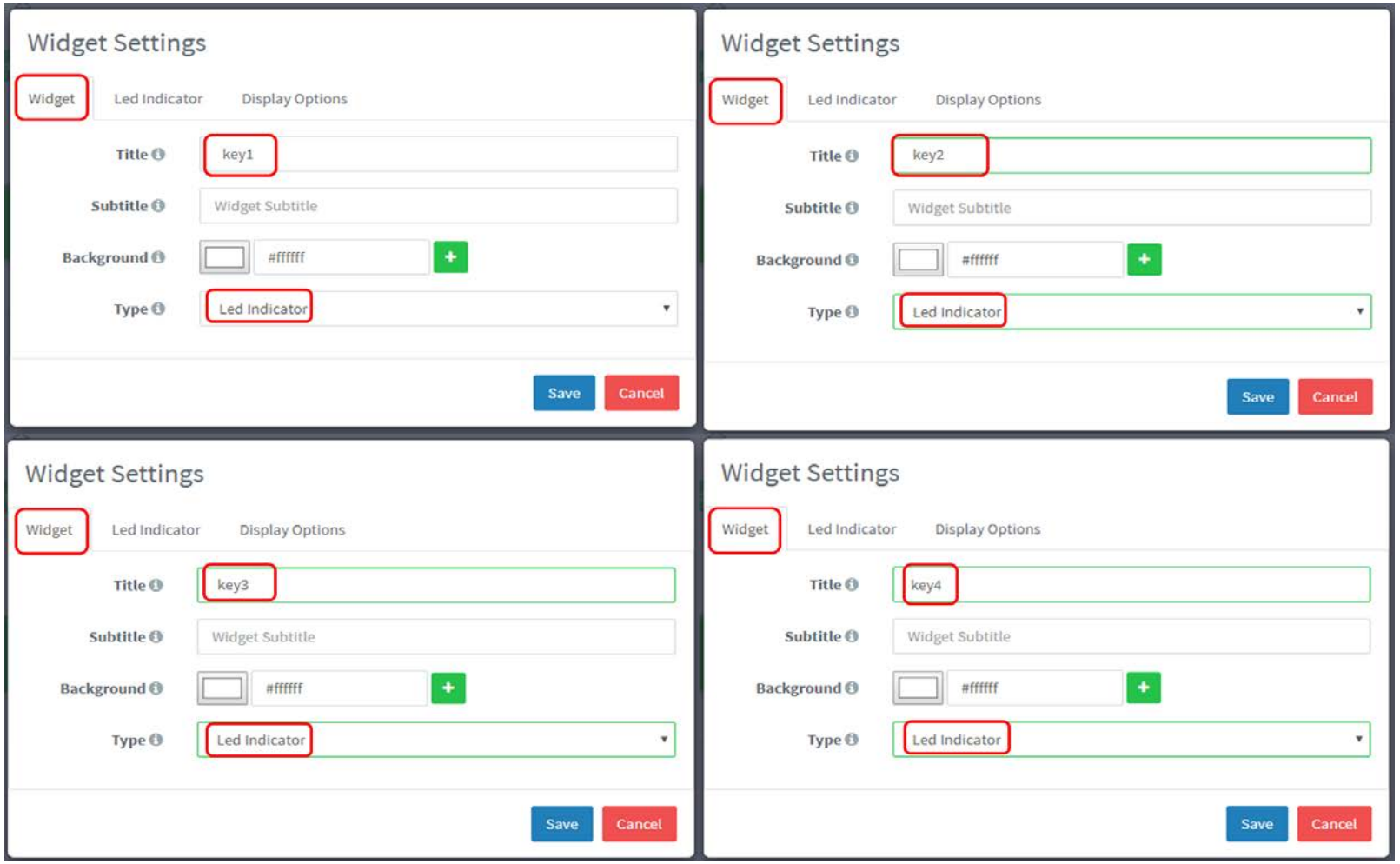
- “./run.sh” 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다.
- 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다.
- 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

```
pi@raspberrypi:~/Input $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/Input
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
[3815384.487000]: Not connected!
[3815385.791000]: Connecting to iot.thinger.io:25202 ...
[3815387.095000]: Connected!
[3815387.095000]: Authenticating...
```

결과 확인

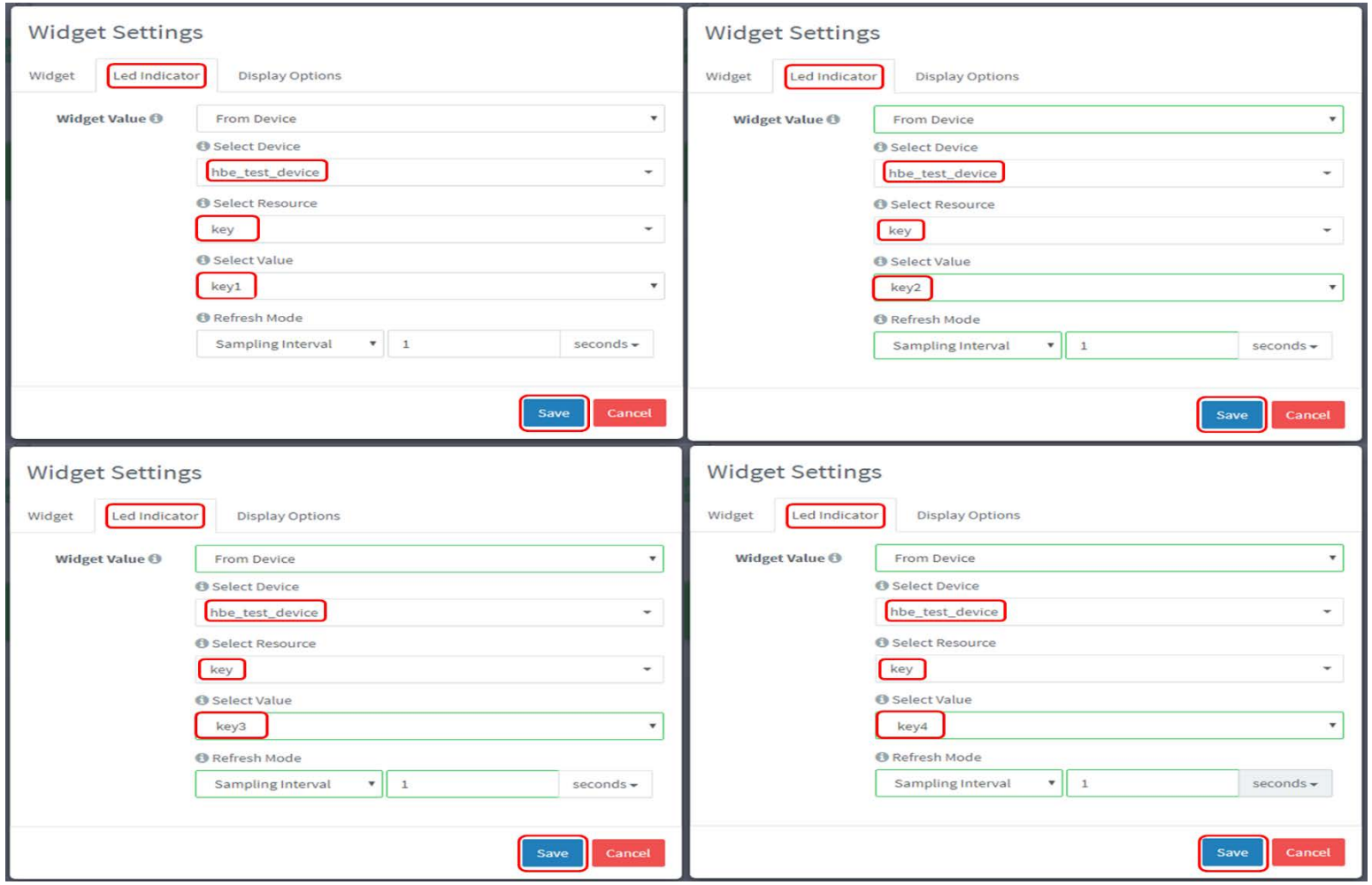
HANBACK ELECTRONICS.,LTD

- Cloud 에서 Input 모듈의 상태 모니터링을 위해 대시보드 설정을 한다.
- 각 key 에 대한 메뉴들을 생성한다.



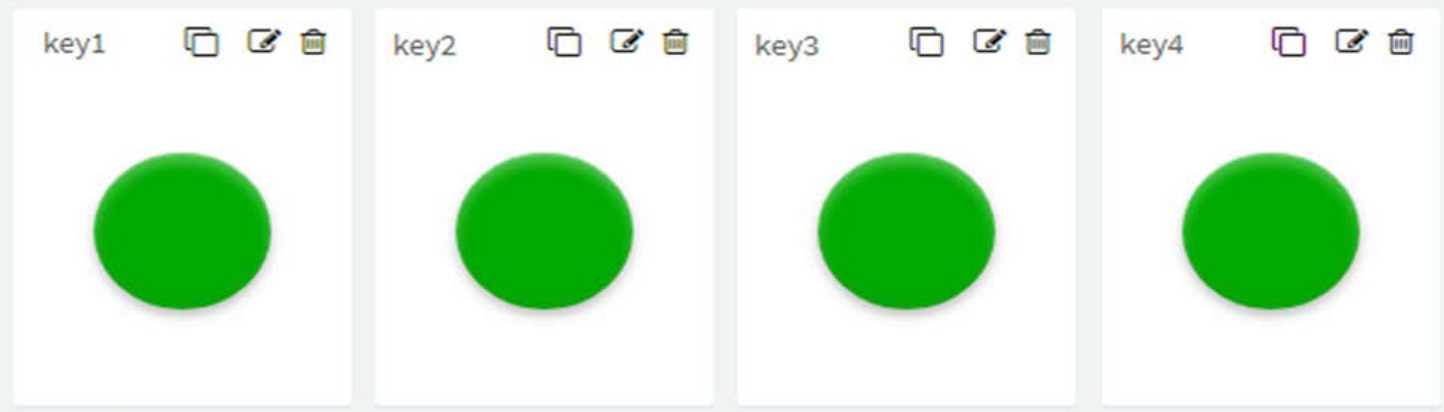
결과 확인

HANBACK ELECTRONICS.,LTD



결과 확인

- 생성된 UI 를 통해 입력된 키의 모니터링을 할 수 있다.



1. 프로젝트 생성

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

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

1. Light 센서 프로그램 작성

main.cpp

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

#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 CDS_CH               0


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;
}
```

Light 센서 프로그램 작성

```
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);
                }
                thing["cds"] >> [](pson& out){
                                out = mcp3208(CDS_CH);
                };
                thing.start();
                return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 각종 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

- 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 CDS_CH               0
```

3) 아날로그 데이터 수신 함수 선언

- SPI 통신을 통해 아날로그 데이터를 수신한다. 데이터 송신 및 수신 시 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) Cloud 연결 초기화 및 제어핀 설정

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

```
thing["cds"] >> [](pson& out){
        out = mcp3208(CDS_CH);
};
```

6) Cloud 연결

- 앞서 설정한 연결 정보를 기반으로 Cloud 와 연결을 시도한다. 설정한 정보가 틀린 내용이 있다면 정상적으로 연결이 되지 않는다.

```
thing.start();
```

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)

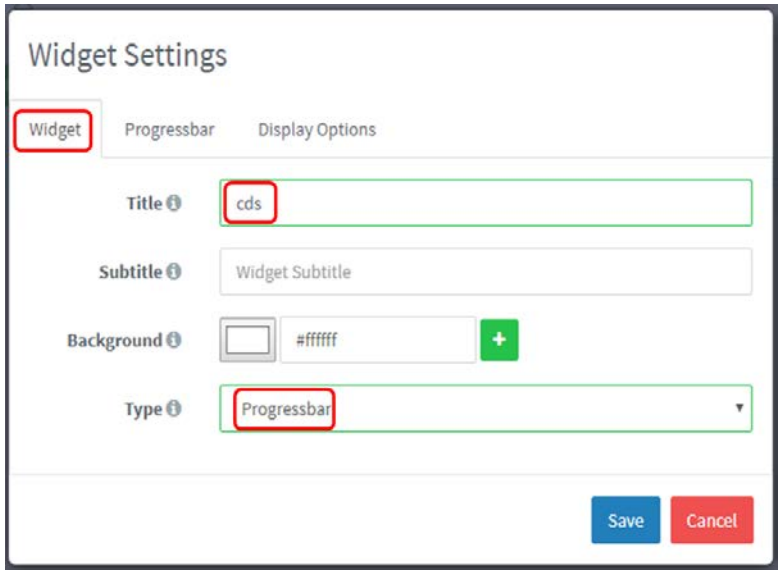
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다.
- 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다.
- 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

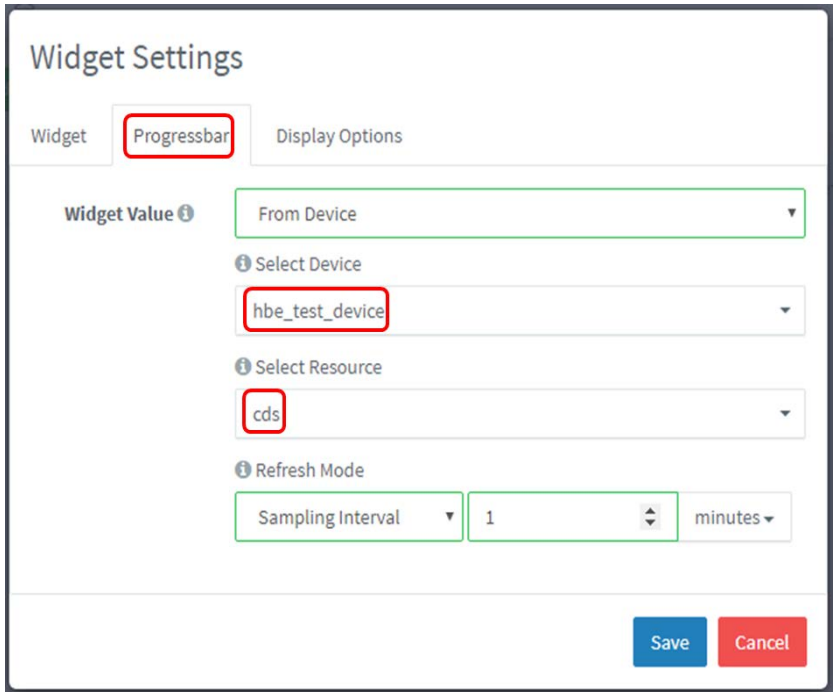
```
pi@raspberrypi:~/cds $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/cds
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3818315.517000]: Not connected!
[3818315.526000]: Connecting to iot.thinger.io:25202 ...
[3818316.734000]: Connected!
[3818316.734000]: Authenticating...
[3818317.027000]: Authenticated!
```

결과 확인

HANBACK ELECTRONICS.,LTD

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

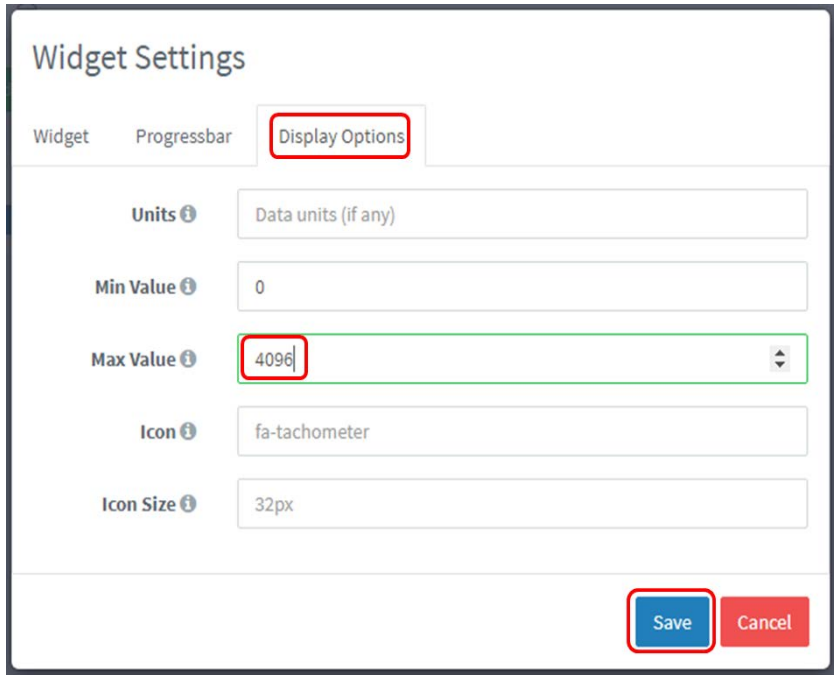




결과 확인

HANBACK ELECTRONICS.,LTD

- 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.



1.프로젝트 생성

- Cloud 와 연동하여 Ultrasonic 센서를 모니터링하기 위한 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 제어 하는 내용을 기술하면 된다.

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

1. Ultrasonic 센서 프로그램 작성

main.cpp

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

#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
#define ULTRA_TRIG_PIN  28
#define ULTRA_OUT_PIN   29

int ultra(void){
                int distance=0;
                long startTime, travelTime;
                digitalWrite(ULTRA_TRIG_PIN,LOW);
                usleep(2);
                digitalWrite(ULTRA_TRIG_PIN,HIGH);
                usleep(20);
                digitalWrite(ULTRA_TRIG_PIN,LOW);
                while(digitalRead(ULTRA_OUT_PIN) == LOW);
                startTime = micros();
                while(digitalRead(ULTRA_OUT_PIN) == HIGH);
                travelTime = micros() - startTime;
                distance = travelTime / 58;
                return distance;
}

int main(int argc, char *argv[])
{
```

```
int main(int argc, char *argv[])
{
                thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
                 wiringPiSetup();
                 pinMode(ULTRA_TRIG_PIN,OUTPUT);
                 pinMode(ULTRA_OUT_PIN,INPUT);
                 thing["ultra"] >> [](pson& out){
                                out = ultra();
                 };
                 thing.start();
                 return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 각종 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 및 Ultrasonic 센서를 제어하기 위한 핀 번호를 변수로 선언한다.

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


#define ULTRA_TRIG_PIN            28
#define ULTRA_OUT_PIN             29
```

3) Ultrasonic 제어

- UltraSonic 의 데이터를 읽기 위하여 Trig 핀을 LOW -> HIGH -> LOW 순으로 제어한다. 이후 Echo 핀에 HIGH 데이터가 수신 되면 해당 시간부터 Echo 핀의 정보가 변경 될 때 까지의 시간을 계산하여 거리를 산출하여 반환 한다.

```
int ultra(void){
                int distance=0;
                long startTime, travelTime;
                digitalWrite(ULTRA_TRIG_PIN,LOW);
                usleep(2);
                digitalWrite(ULTRA_TRIG_PIN,HIGH);
                usleep(20);
                digitalWrite(ULTRA_TRIG_PIN,LOW);
                while(digitalRead(ULTRA_OUT_PIN) == LOW);
                startTime = micros();
                while(digitalRead(ULTRA_OUT_PIN) == HIGH);
                travelTime = micros() - startTime;
                distance = travelTime / 58;
                return distance;
}
```

4) Ultrasonic 핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. 그리고 wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다. TRIG 핀은 LOW -> HIGH, HIGH -> LOW 제어해야 되기 때문에 OUTPUT으로 OUT 핀은 입력된 신호를 읽어야 함으로 INPUT 으로 설정한다.

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

wiringPiSetup();

pinMode(ULTRA_TRIG_PIN,OUTPUT);

pinMode(ULTRA_OUT_PIN,INPUT);
```

5) 센서 등록 및 데이터 송신

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

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

        out = ultra();

};
```

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----------------------------------------
```

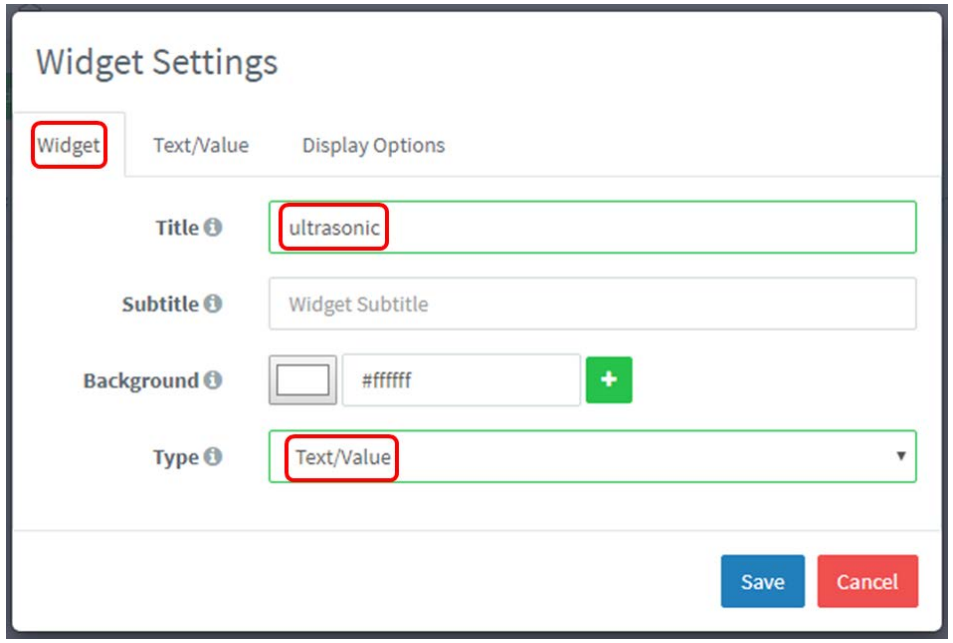
1. 프로그램 컴파일 (in Raspberry Pi)

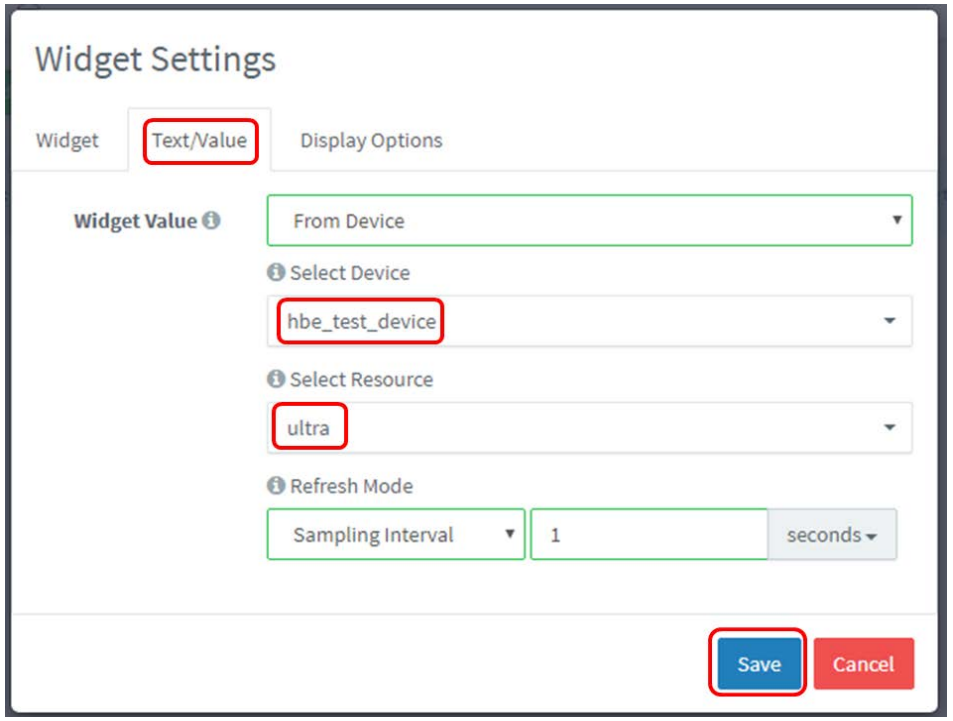
- “./run.sh” 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다.
- 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다.
- 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

```
pi@raspberrypi:~/ultrasonic $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/ultrasonic
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
[3822777.718000]: Not connected!
[3822777.856000]: Connecting to iot.thinger.io:25202 ...
[3822779.063000]: Connected!
[3822779.063000]: Authenticating...
[3822779.347000]: Authenticated!
```

결과 확인

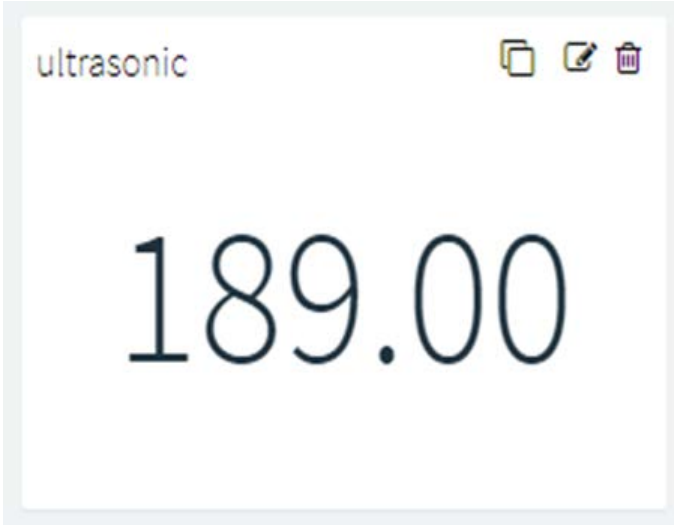
- 라즈베리파이에서 프로그램이 실행되고 나면 Cloud 에서 Ultrasonic센서 상태 확인을 위해 대시보드 설정을 한다. 아래 그림과 같이 Add Widget을 눌러 Ultrasonic 센서를 모니터링 할 Widget을 추가한다.





결과 확인

- 설정을 위와 같이 수행하고 저장하면 Ultrasonic 센서 전방에 있는 물체를 측정하여 거리 데이터를 반환하는 것을 확인할 수 있다.



1.프로젝트 생성

- Cloud 와 연동하여 온/습도 센서를 모니터링하기 위한 프로젝트를 생성한다.
  생성하는 프로젝트는 앞서 디바이스 등록 때 사용한 Linux-Client를 복사하여 사용한다.
  복사한 프로젝트에는 Cloud 와 연동 하는 내용 및 센서 제어 하는 내용을 기술하면 된다.

```
pi@raspberrypi:~ $ cp -rf Linux-Client dht11
pi@raspberrypi:~ $ cd dht11
pi@raspberrypi:~/dht11 $
```

1. 온/습도 센서 프로그램 작성

main.cpp

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

#define USER_ID              "USER_ID"
#define DEVICE_ID            "USER_DEVICE_ID"
#define DEVICE_CREDENTIAL    "USER_DEVICE_CREDENTIAL"
#define DHT11_PIN            25
#define MAX_TIME 100

int dht11_val[5]={0,0,0,0,0};
```

```
int dht11(void){
        int i,j=0,cnt=0,laststate=1;
        float temp;
        for(i=0;i<5;i++){
                dht11_val[i] = 0;
        }
        pinMode(DHT11_PIN,OUTPUT);
        digitalWrite(DHT11_PIN,LOW);
        delay(18);
        digitalWrite(DHT11_PIN,HIGH);
        delayMicroseconds(40);
        pinMode(DHT11_PIN,INPUT);
        for(i=0;i<MAX_TIME;i++){
                cnt=0;
                while(digitalRead(DHT11_PIN) == laststate){
```

온/습도 센서 프로그램 작성

```
                cnt++;
                delayMicroseconds(1);
                if(cnt == 255)
                        break;
        }
        laststate = digitalRead(DHT11_PIN);
        if(cnt == 255)
                break;
        if((i>=4) && (i%2==0)){
                dht11_val[j/8]<<=1;
                if(cnt>16)
                   dht11_val[j/8]|=1;
                j++;
        }
}
```

```
        if((j>=40) && (dht11_val[4] == ((dht11_val[0]+dht11_val[1]+dht1
1_val[2]+dht11_val[3])&0xFF))){
                return 0;
        }else{
                return -1;
        }
}

int main(int argc, char *argv[]){
        thinger_device thing(USER_ID, DEVICE_ID, DEVICE_CREDENTIAL);
        wiringPiSetup();
        thing["dht11"] >> [](pson& out){
                int result = dht11();
                out["temp"] = dht11_val[2];
                out["humi"] = dht11_val[0];
        };
        thing.start();
        return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 각종 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 및 온/습도 센서를 제어하기 위한 핀 번호를 변수로 선언한다. 수신된 데이터를 저장할 변수도 함께 선언한다.

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


#define DHT11_PIN     25
#define MAX_TIME 100
int dht11_val[5]={0,0,0,0,0};
```

3) 온/습도 데이터 수신

- 온/습도 데이터를 수신한다. 먼저 센서에 연결된 핀을 OUTPUT으로 설정하고 이 핀의 상태를 LOW에서 HIGH 로 변경한다. 이후 핀의 설정을 INPUT 으로 변경하고 수신된 데이터들을 저장한다.

```
int dht11(void){
        int i,j=0,cnt=0,laststate=1;
        float temp;
        for(i=0;i<5;i++){
                dht11_val[i] = 0;
        }
        pinMode(DHT11_PIN,OUTPUT);
        digitalWrite(DHT11_PIN,LOW);
        delay(18);
        digitalWrite(DHT11_PIN,HIGH);
        delayMicroseconds(40);
        pinMode(DHT11_PIN,INPUT);
        for(i=0;i<MAX_TIME;i++){
                cnt=0;
                while(digitalRead(DHT11_PIN) == laststate){
                        cnt++;
                        delayMicroseconds(1);
                        if(cnt == 255)
                                break;
                }
                laststate = digitalRead(DHT11_PIN);
                if(cnt == 255)
                        break;
                if((i>=4) && (i%2==0)){
                        dht11_val[j/8]<<=1;
                        if(cnt>16)
                                dht11_val[j/8]|=1;
                        j++;
                }
        }
        if((j>=40) && (dht11_val[4] == ((dht11_val[0]+dht11_val[1]+dht11_val[2]+dht11_val[3])&0xFF))){
                return 0;
        }else{
                return -1;
        }
}
```

4) Cloud 연결 초기화 및 제어핀 설정

- Cloud 와 연결을 위한 초기화를 수행한다. 그리고 wiringPisetup() 함수를 이용하여 GPIO 사용을 위한 초기화를 진행한다.

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

wiringPiSetup();
```

5) 센서 등록 및 데이터 송신

- 센서를 등록한다. thing["dht11"] 라고 선언을 하게 되면 Cloud 에 dht11 라는 센서가 등록되게 된다. Cloud 로 전송할 데이터는 out 이라는 변수에 전달하면 대시보드에 설정된 주기에 따라 데이터를 Cloud 로 전송하게 된다.

```
thing["dht11"] >> [](pson& out){
        int result = dht11();
        out["temp"] = dht11_val[2];
        out["humi"] = dht11_val[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----------------------------------------
```

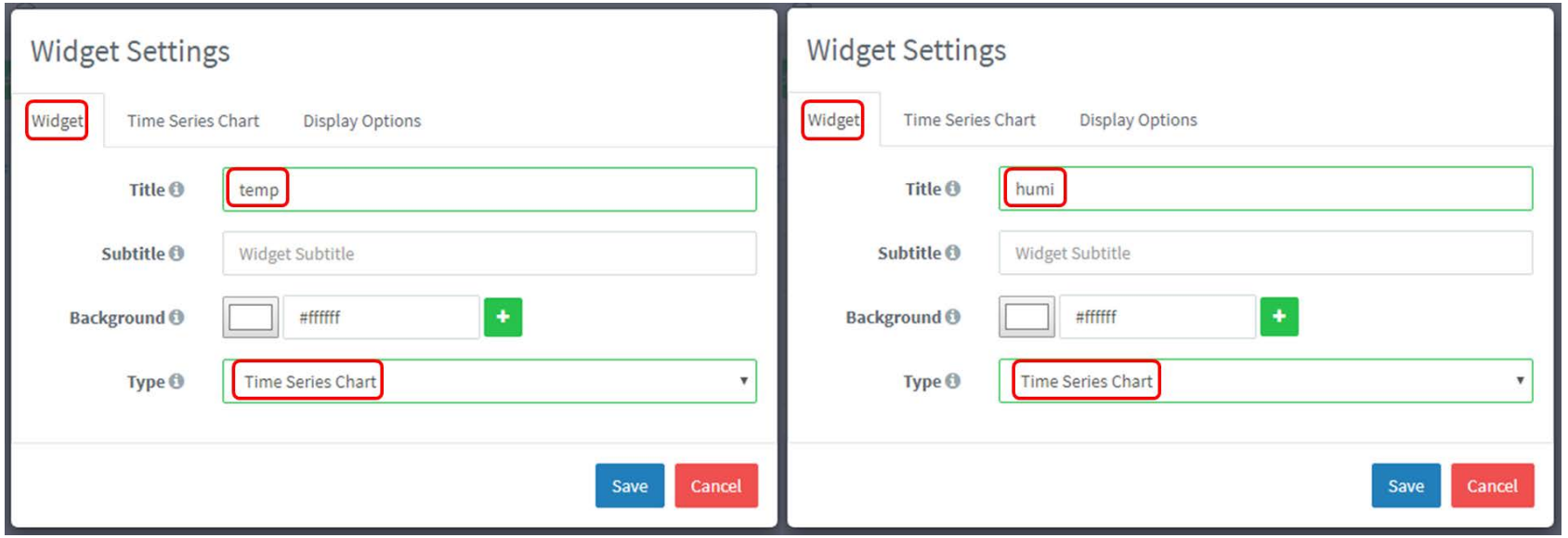
1. 프로그램 컴파일 (in Raspberry Pi)

- “./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:~/dht11 $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/dht11
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
[3823083.711000]: Not connected!
[3823083.721000]: Connecting to iot.thinger.io:25202 ...
[3823084.921000]: Connected!
[3823084.922000]: Authenticating...
[3823085.205000]: Authenticated!
```

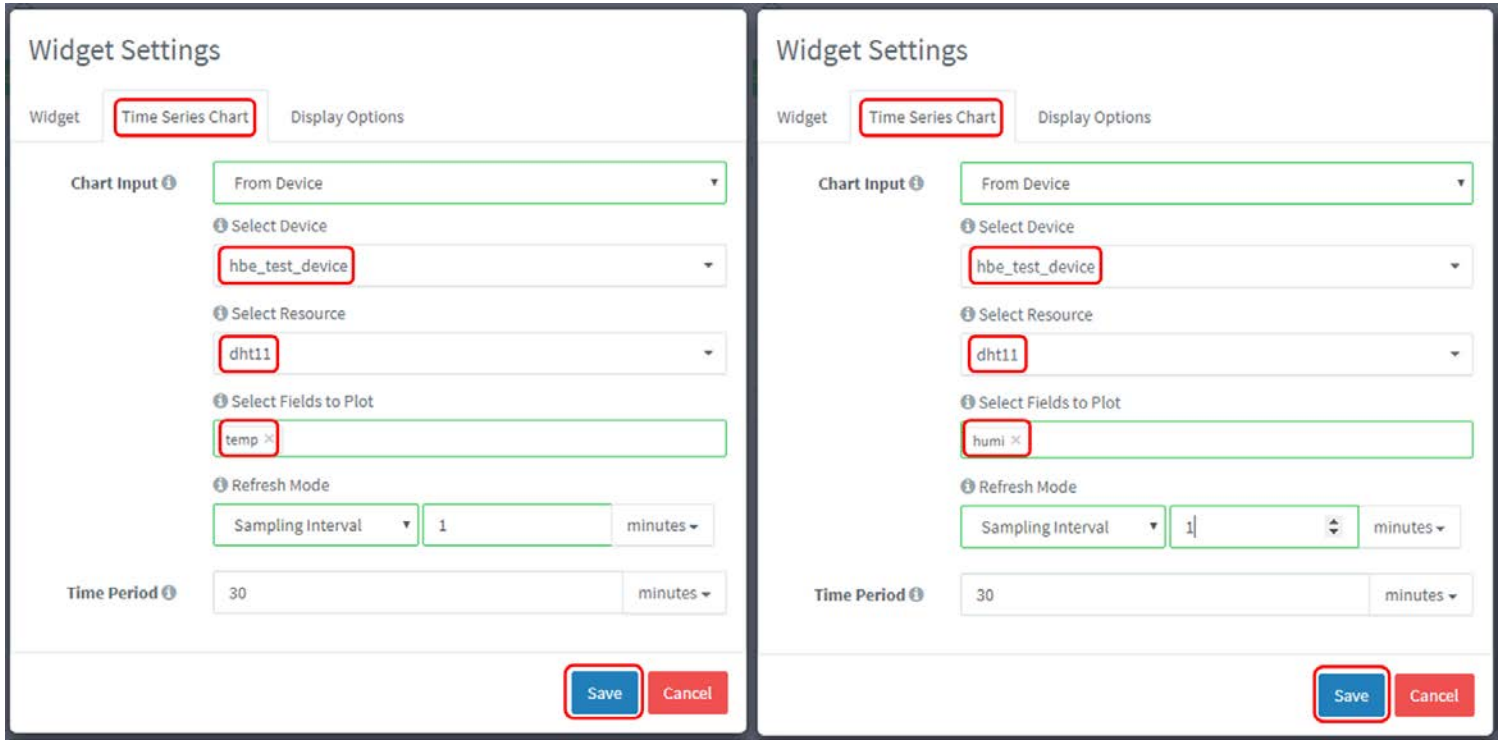
결과 확인

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



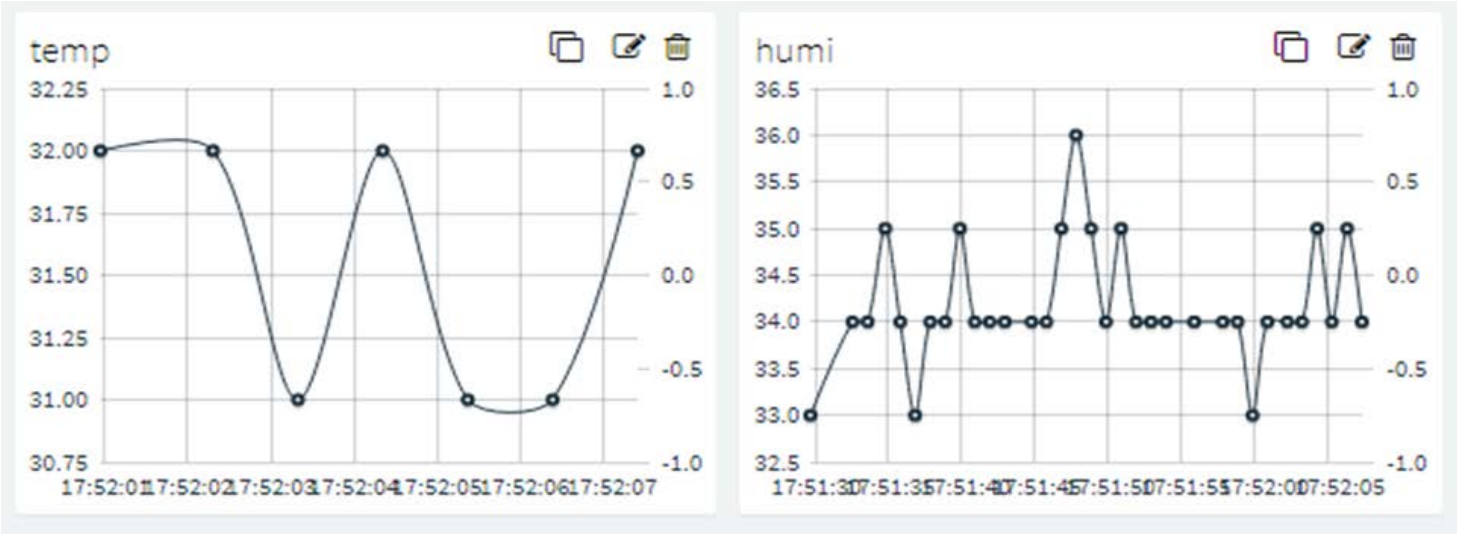
결과 확인

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



결과 확인

- 설정을 위와 같이 수행하고 센서 주변에서 바람을 불거나 센서주변의 온도를 변화 시키면 데이터가 변화하는 것을 확인할 수 있다.



1.프로젝트 생성

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

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

1. 가변저항 프로그램 작성

main.cpp

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

#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 VR_CH                1

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;
}
```

가변저항 프로그램 작성


```
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);
        }
        thing["vr"] >> [](pson& out){
                out = mcp3208(VR_CH);
        };
        thing.start();
        return 0;
}
```

1. main.cpp

1) 사용 헤더파일 선언

- 각종 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 및 가변저항의 상태를 확인하기 위한 핀 번호를 변수로 선언한다. SPI 통신을 위한 변수도 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 VR_CH        1
```

3) 아날로그 데이터 수신 함수 선언

- SPI 통신을 통해 아날로그 데이터를 수신한다. 데이터 송신 및 수신 시 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) Cloud 연결 초기화 및 제어핀 설정

- 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);

}
```

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----------------------------------------
```

결과 확인

1. 프로그램 컴파일 (in Raspberry Pi)

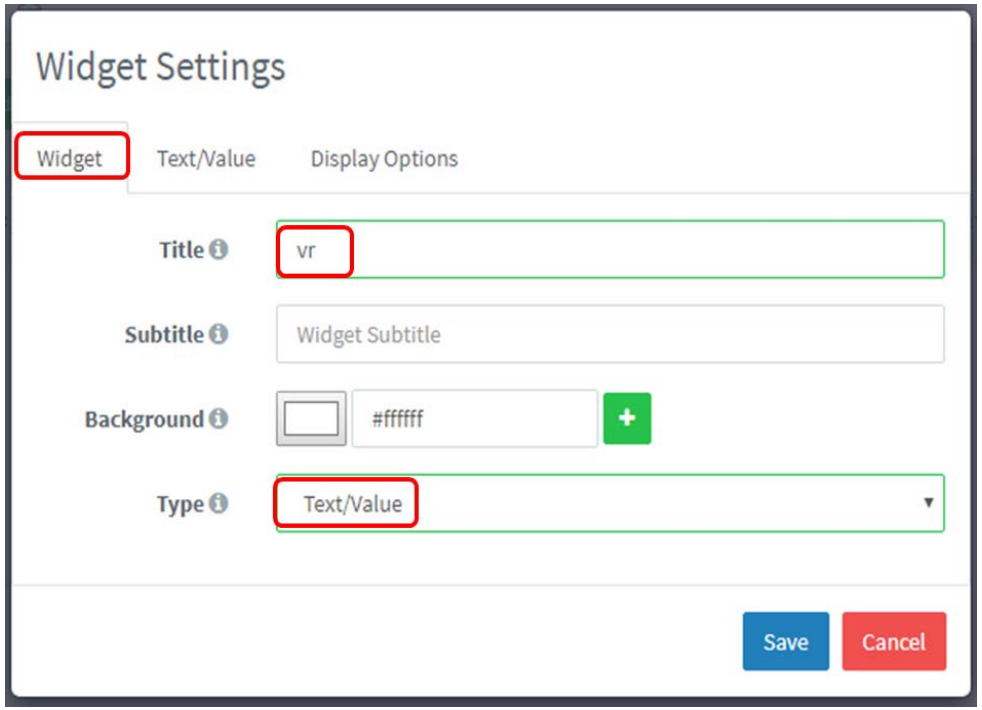
- "./run.sh" 명령을 통해 run.sh 파일을 실행하면 Cloud에 연결하기에 앞서 컴파일을 진행한다.
- 컴파일이 에러 없이 진행되고 나면 자동으로 프로그램을 실행하여 Cloud 에 연동을 시작한다.
- 이 때 Cloud 연동 정보에 오타가 없어야 연결이 정상적으로 수행된다.

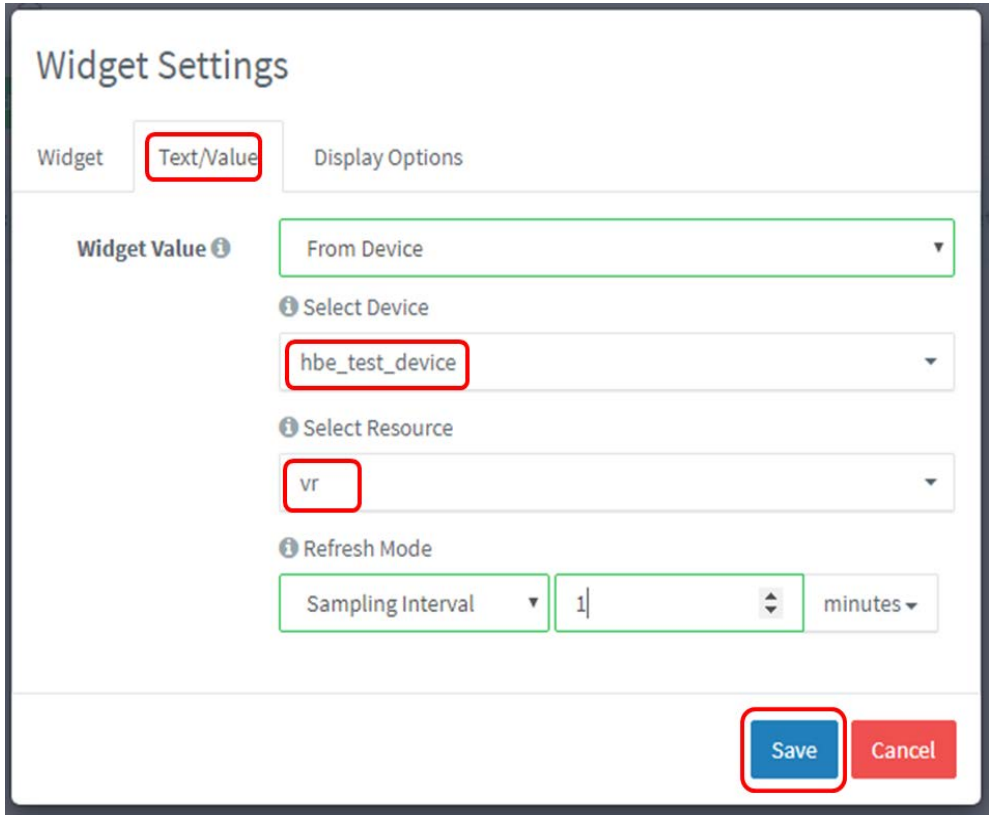
```
pi@raspberrypi:~/vr $ ./run.sh
-- OpenSSL Version: 1.0.1t
-- Configuring done
-- Generating done
-- Build files have been written to: /home/pi/vr
Scanning dependencies of target thinger
[ 50%] Building CXX object CMakeFiles/thinger.dir/src/main.cpp.o
[100%] Linking CXX executable thinger
[100%] Built target thinger
[3823897.892000]: Not connected!
[3823897.901000]: Connecting to iot.thinger.io:25202 ...
[3823899.106000]: Connected!
[3823899.106000]: Authenticating...
[3823899.392000]: Authenticated!
```

결과 확인

HANBACK ELECTRONICS.,LTD

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





결과 확인

- 설정을 위와 같이 수행하고 저장하면 센서 데이터가 변화하는 것을 확인할 수 있다.