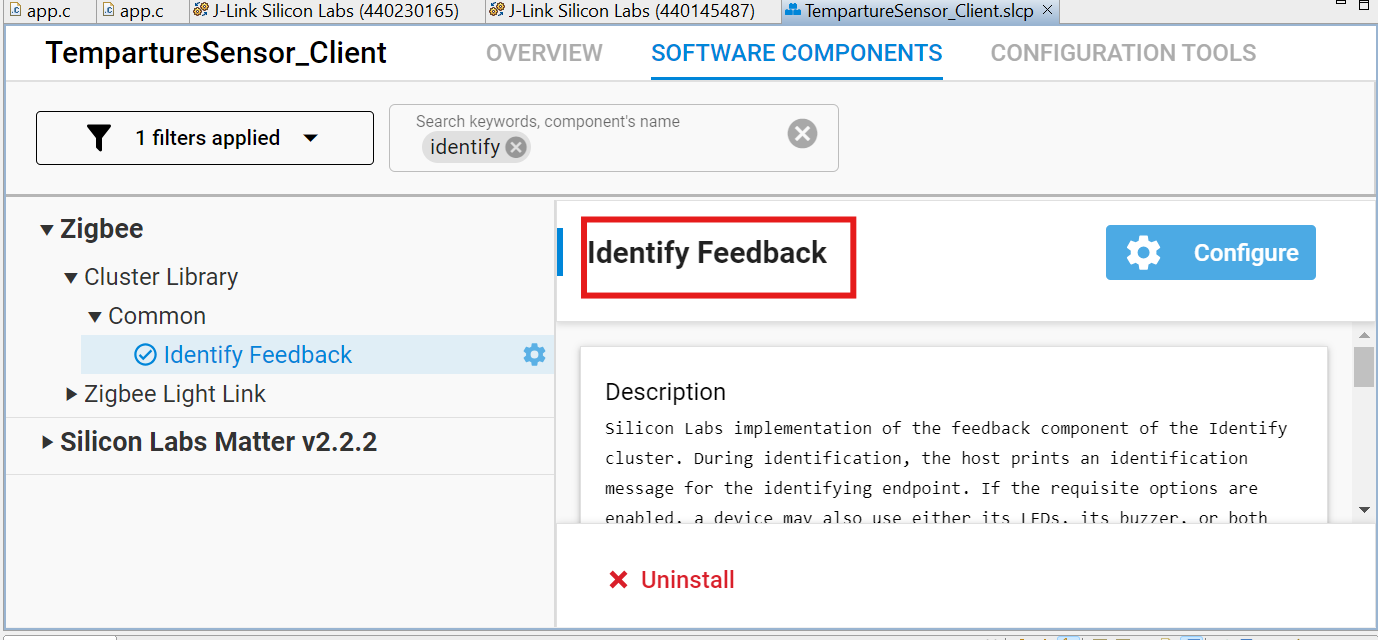
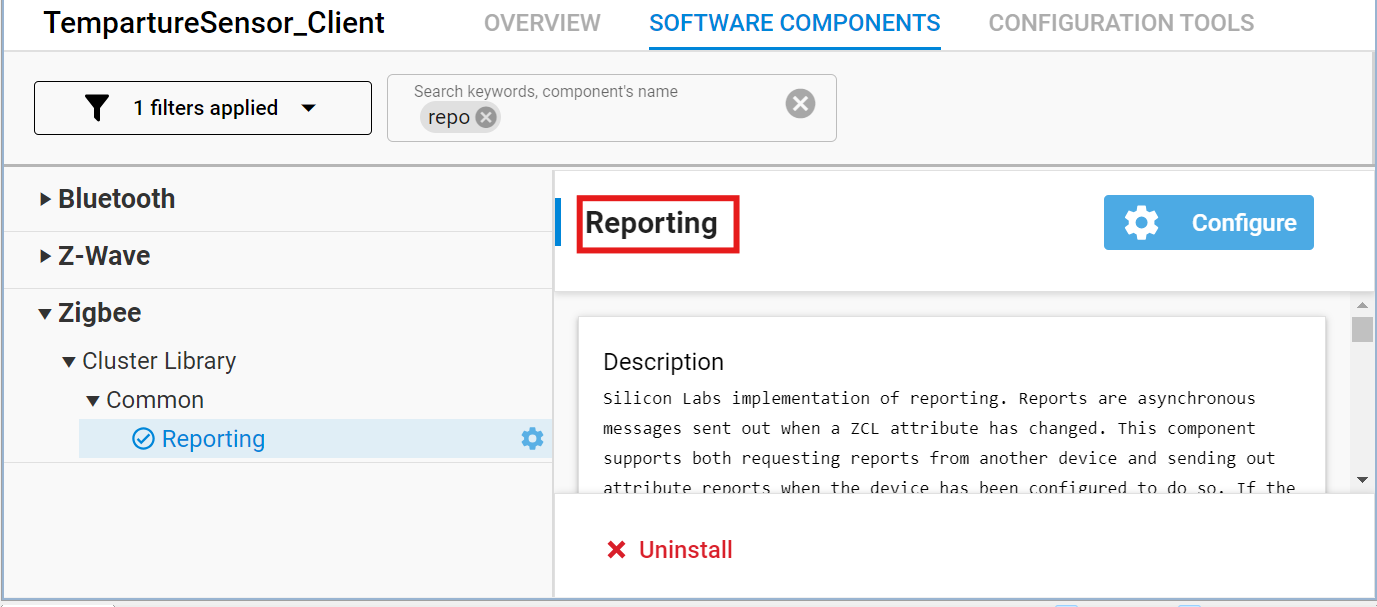
**Project Progress Update**

So far, we have successfully formed and joined the network and enabled the transmission of ON/OFF button commands between the Light and Switch devices. Now, we will extend the project further by integrating the RHT sensor available on the board. The goal is to transmit sensor data from the router to the coordinator.

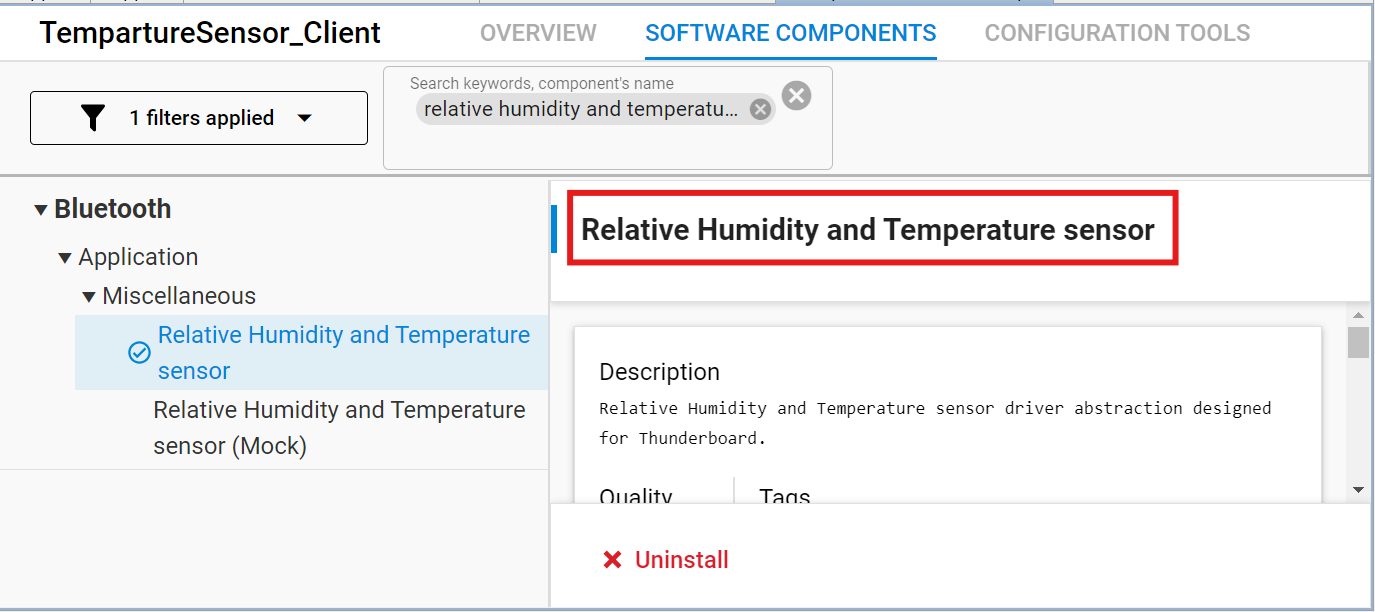
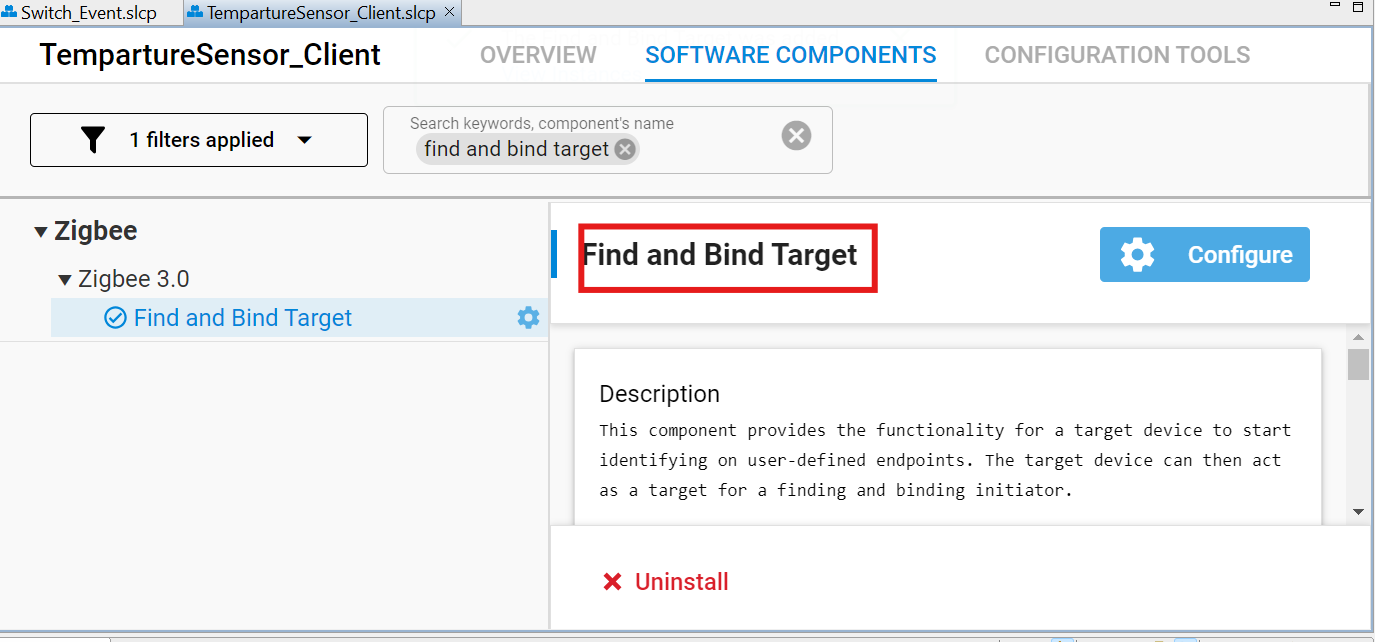
**Note:** This document does not cover the steps for network creation/forming or sending ON/OFF commands. It begins directly with sensor integration. Please refer to earlier documents for those initial steps.

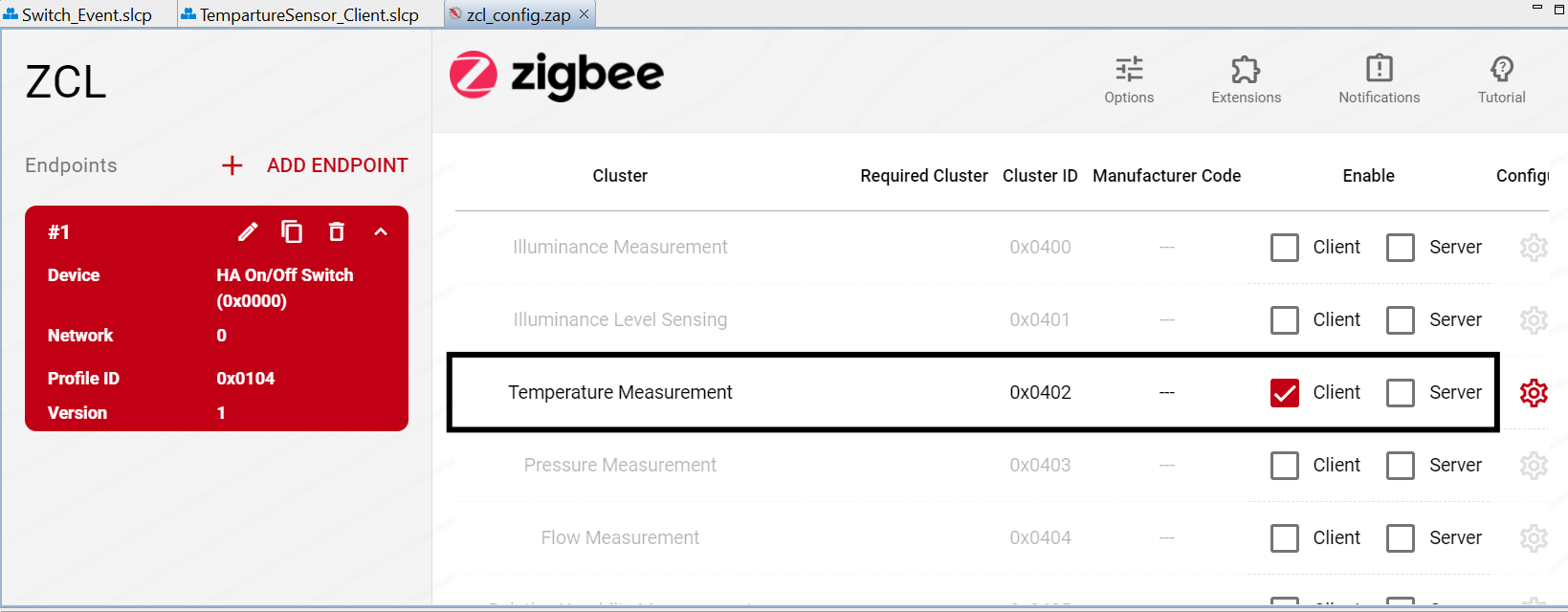
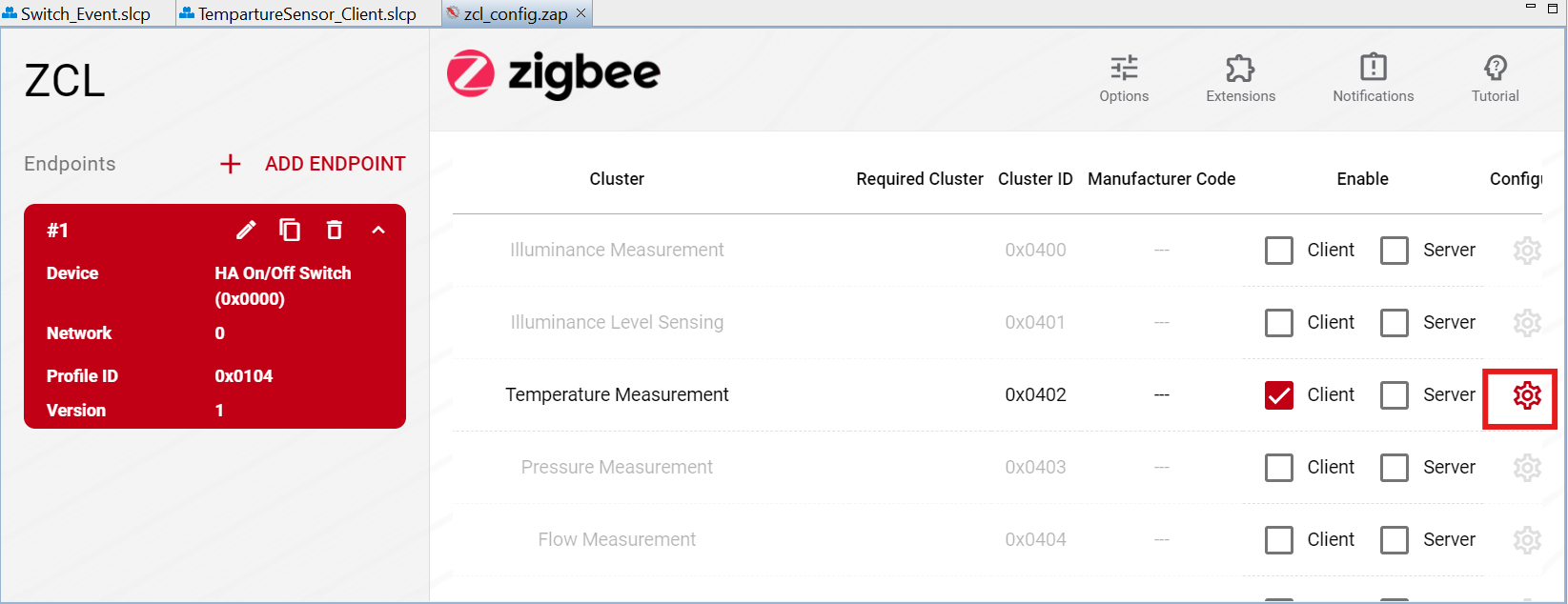
**Switch Side :**   
  
Now we try to install the below **software components** on Switch side   
  
  
1) Identify Feedback  
  


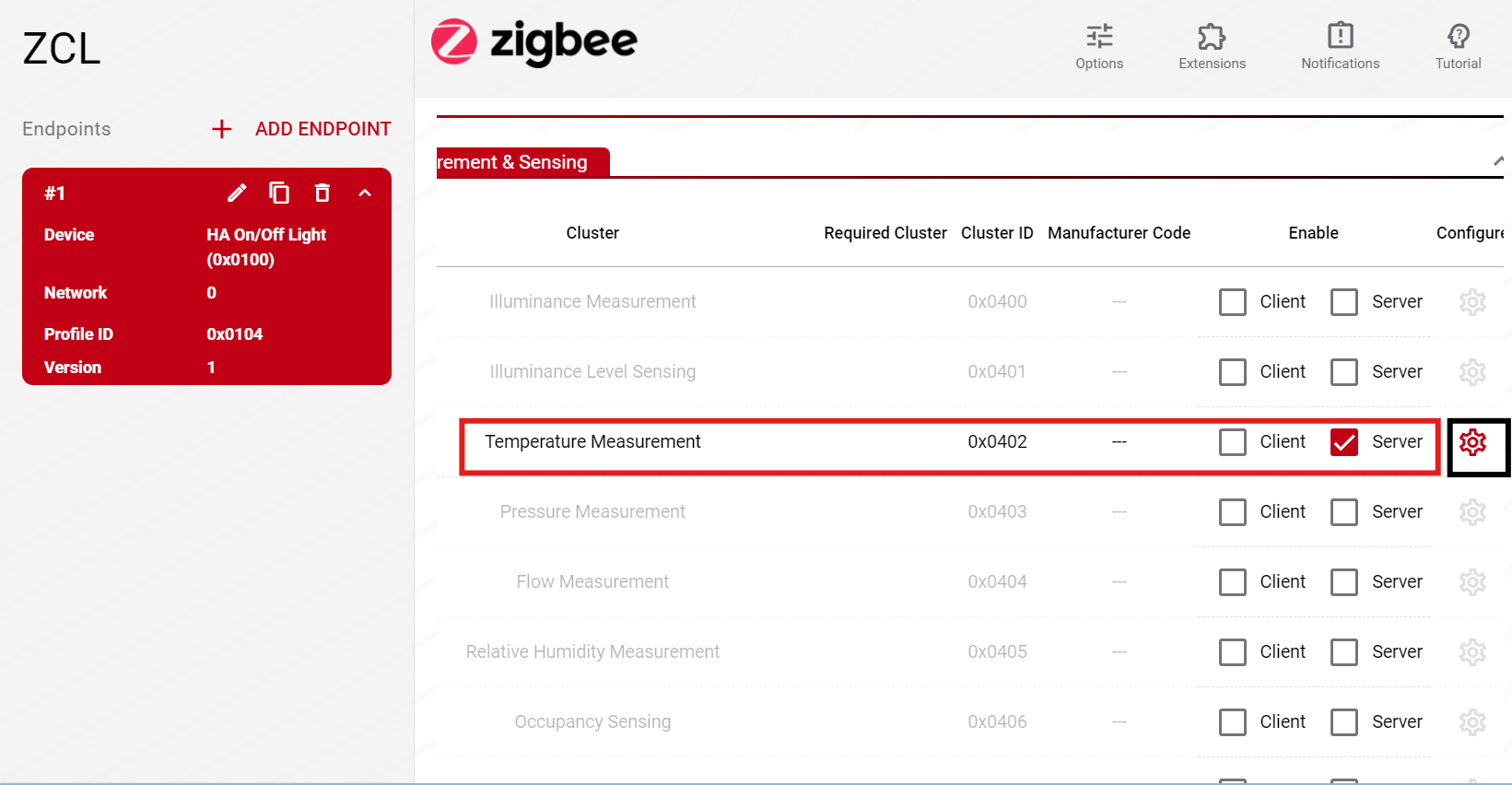
2) Reporting

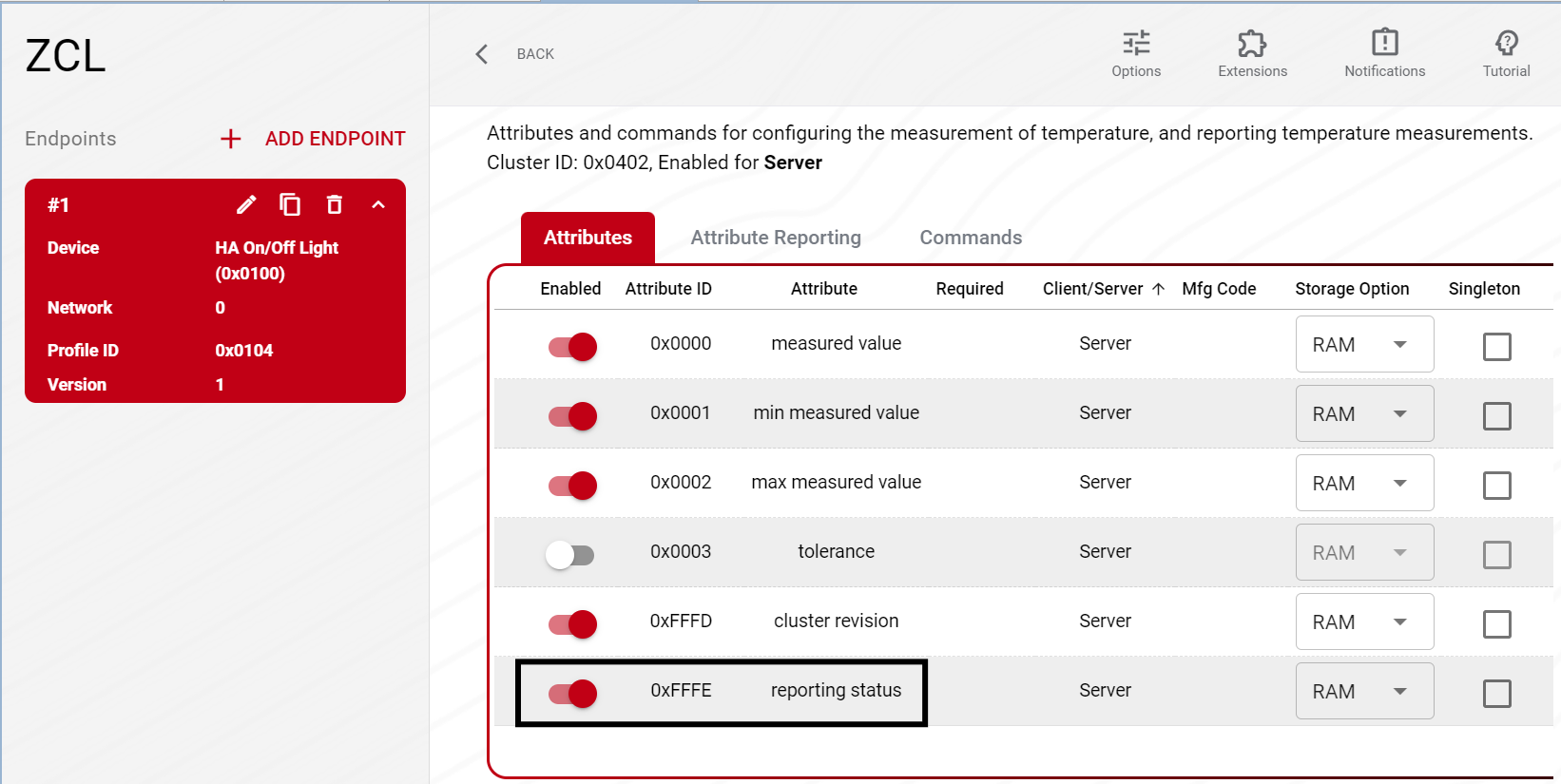


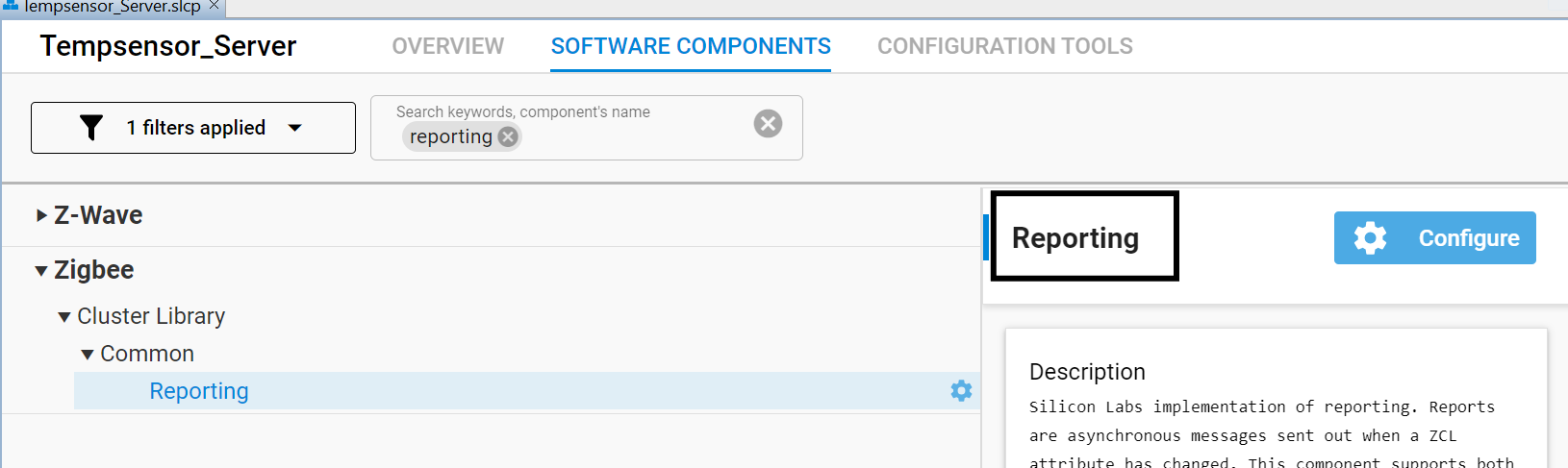
3) Relative Humidity and Temperature Sensor

  
  
  
4) Find and Bind Target  
  
  


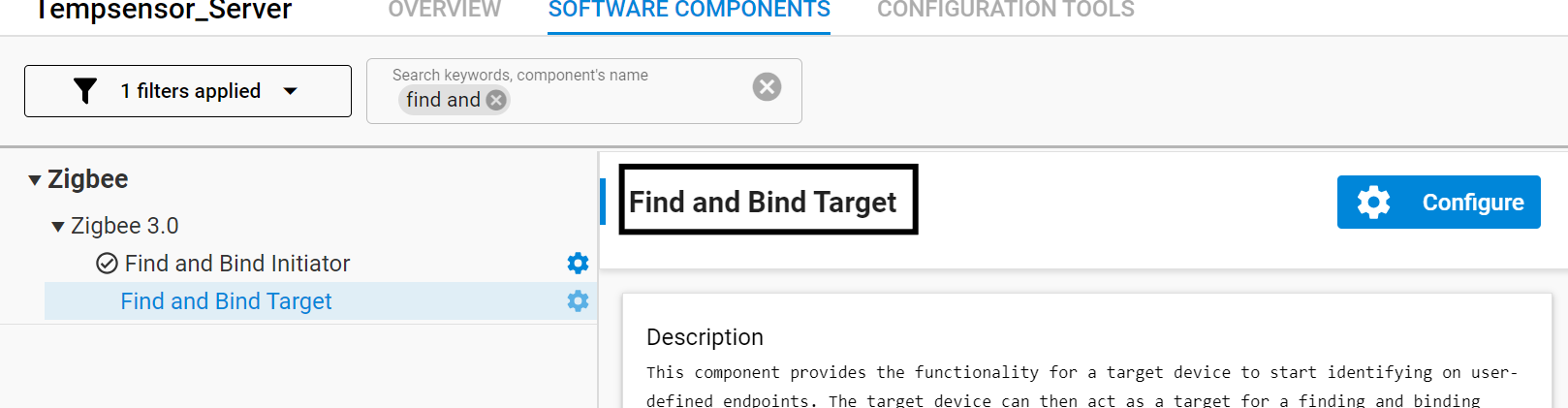
Now Lets configure the **ZCL Advanced Platform (ZAP)** on switch side   
  
  
1) In the existing endpoint, in the **measurement and sensing** section we can find the **Temperature Measurement** and set it as **client** as shown below   
  
  
2) Now go to the setting button and **enable reporting status** as shown below  


**Light:**  
  
Now Lets configure the **ZCL Advanced Platform (ZAP)** on Light side   
  
1) In the existing endpoint, in the **measurement and sensing** section we can find the **Temperature Measurement** and set it as **server** as shown below   
  


And within the settings section **enable the reporting status** :   
  
  
  
  
Now we try to install the below **software components** on Light side  
  
1) Reporting



2) Find and Bind Target

  
  
  
  
  
Switch Code :   
  
//SWITCH//

#include "app/framework/include/af.h"

#include "sl\_simple\_button.h"

#include "sl\_simple\_button\_instances.h"

#include "app/framework/util/af-main.h"

#include "app/util/common/common.h"

#include "app/framework/plugin/network-steering/network-steering.h"

#include "app/framework/plugin/find-and-bind-initiator/find-and-bind-initiator.h"

#include "sl\_sensor\_rht.h"

#define BUTTON0 0

#define BUTTON1 1

#define FIND\_AND\_BIND\_DELAY\_MS 3000

#define REPORT\_PERIOD\_MS (4000)

volatile bool button0Pressed; // Used for joining the network

volatile bool button1Pressed; // enabling the reporting sensor data

static bool commissioning = false;

static bool binding = false;

static EmberStatus reportAttribute(EmberAfClusterId cluster,

EmberAfAttributeId attributeID,

uint8\_t attribute\_type,

uint8\_t buff[2]);

void emberAfRadioNeedsCalibratingCallback(void)

{

sl\_mac\_calibrate\_current\_channel();

}

static sl\_zigbee\_event\_t sendCommandEventOn;

static sl\_zigbee\_event\_t sendCommandEventOff;

static sl\_zigbee\_event\_t network\_steering\_event\_control;

static sl\_zigbee\_event\_t finding\_and\_binding\_event\_control;

static sl\_zigbee\_event\_t attribute\_report\_event\_control;

static void sendCommandEventOnHandler(sl\_zigbee\_event\_t \*event);

static void sendCommandEventOffHandler(sl\_zigbee\_event\_t \*event);

static void network\_steering\_event\_handler(sl\_zigbee\_event\_t \*event);

static void finding\_and\_binding\_event\_handler(sl\_zigbee\_event\_t \*event);

static void attribute\_report\_event\_handler(sl\_zigbee\_event\_t \*event);

// Check if device joined network, if not joined enables the finding and binding event

void emberAfPluginNetworkSteeringCompleteCallback(EmberStatus status,

uint8\_t totalBeacons,

uint8\_t joinAttempts,

uint8\_t finalState)

{

sl\_zigbee\_app\_debug\_print("%s network %s: 0x%02X\n", "Join", "complete", status);

if (status != EMBER\_SUCCESS) {

commissioning = false;

} else {

// On successful join, do find and bind after a short delay

sl\_zigbee\_event\_set\_delay\_ms(&finding\_and\_binding\_event\_control,

FIND\_AND\_BIND\_DELAY\_MS);

}

}

// Used for finding and binding the end point called when Button 1 is pressed.

void emberAfPluginFindAndBindInitiatorCompleteCallback(EmberStatus status)

{

sl\_zigbee\_app\_debug\_print("Find and bind initiator %s: 0x%X\n", "complete", status);

if (status == EMBER\_SUCCESS) {

sl\_zigbee\_event\_set\_delay\_ms(&attribute\_report\_event\_control,

REPORT\_PERIOD\_MS);

} else {

sl\_zigbee\_app\_debug\_print("Ensure a valid binding target!\n");

sl\_zigbee\_event\_set\_inactive(&attribute\_report\_event\_control);

binding = false;

}

}

void sl\_button\_on\_change(const sl\_button\_t \*handle)

{

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON0) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button0Pressed = true;

}

}

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON1) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button1Pressed= true;

}

}

}

void sendMessage(){

if(button0Pressed){

sl\_zigbee\_event\_set\_active(&sendCommandEventOn);

sl\_zigbee\_event\_set\_active(&network\_steering\_event\_control);

}

if(button1Pressed){

sl\_zigbee\_event\_set\_active(&sendCommandEventOff);

sl\_zigbee\_event\_set\_delay\_ms(&attribute\_report\_event\_control,

REPORT\_PERIOD\_MS);

}

}

static void sendCommandEventOnHandler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status;

emberAfFillCommandOnOffClusterOn();

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

emberAfCorePrintln("Sending Zigbee On Command...");

button0Pressed =false;

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Command Sent Successfully");

} else {

emberAfCorePrintln("Command Failed, Status: 0x%x", status);

}

}

static void sendCommandEventOffHandler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status;

emberAfFillCommandOnOffClusterOff();

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

emberAfCorePrintln("Sending Zigbee On Command...");

button1Pressed=false;

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Command Sent Successfully");

} else {

emberAfCorePrintln("Command Failed, Status: 0x%x", status);

}

}

void emberAfMainTickCallback(void)

{

sendMessage();

}

static void network\_steering\_event\_handler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status;

if (emberAfNetworkState() == EMBER\_JOINED\_NETWORK) {

// Check if the device has successfully established bindings, if not do so

if (!binding) {

sl\_zigbee\_event\_set\_active(&finding\_and\_binding\_event\_control);

}

} else {

// If not in a network, attempt to join one

status = emberAfPluginNetworkSteeringStart();

sl\_zigbee\_app\_debug\_print("%s network %s: 0x%X\n",

"Join",

"start",

status);

commissioning = true;

}

}

static void finding\_and\_binding\_event\_handler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status = emberAfPluginFindAndBindInitiatorStart(1);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Bind and Find sucessfull");

} else {

emberAfCorePrintln("Bind and find failed, Status: 0x%x", status);

}

sl\_zigbee\_app\_debug\_print("Find and bind initiator %s: 0x%X\n", "start", status);

binding = true;

}

static void attribute\_report\_event\_handler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status = EMBER\_SUCCESS;

uint32\_t rh;

int32\_t t;

union {

int16\_t t;

uint16\_t rh;

}attribute;

sl\_zigbee\_event\_set\_delay\_ms(&attribute\_report\_event\_control,

REPORT\_PERIOD\_MS);

if (emberAfNetworkState() != EMBER\_JOINED\_NETWORK) {

return;

}

if (SL\_STATUS\_OK == sl\_sensor\_rht\_get(&rh, &t)) {

sl\_zigbee\_app\_debug\_print("RH: %d, Temperature: %d\n", rh, t);

// Attribute MeasuredValue = 100 x temperature in degrees Celsius.

attribute.t = t / 10;

status = reportAttribute( ZCL\_TEMP\_MEASUREMENT\_CLUSTER\_ID,

ZCL\_TEMP\_MEASURED\_VALUE\_ATTRIBUTE\_ID,

ZCL\_INT16S\_ATTRIBUTE\_TYPE,

(uint8\_t \*)&attribute);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Temperature Report bind sucessfull");

} else {

emberAfCorePrintln("Temperature report bind failed, Status: 0x%x", status);

}

sl\_zigbee\_app\_debug\_print("%s reported: 0x%X\n", "Temp - MeasuredValue", status);

} else {

sl\_zigbee\_app\_debug\_print("Failed to read RHT sensor\n");

}

}

static EmberStatus reportAttribute(EmberAfClusterId cluster,

EmberAfAttributeId attributeID,

uint8\_t attribute\_type,

uint8\_t buff[2])

{

EmberStatus status = EMBER\_SUCCESS;

// Fill attribute record - See af-structs.h for details of ReportAttributeRecord

// contents.

uint8\_t attribute\_record[] = {

LOW\_BYTE(attributeID), //uint16\_t attributeId

HIGH\_BYTE(attributeID),

attribute\_type, //uint8\_t attributeType;

buff[0], buff[1], //uint8\_t\* attributeLocation;

};

//Fill a ZCL global report attributes command buffer

emberAfFillCommandGlobalServerToClientReportAttributes(

cluster,

attribute\_record,

sizeof(attribute\_record)/sizeof(uint8\_t));

//Specify endpoints for command sending

emberAfSetCommandEndpoints(1, 1);

//Use binding table to send unicast command

status = emberAfSendCommandUnicastToBindings();

return status;

}

void emberAfMainInitCallback(void){

sl\_status\_t sc;

sc = sl\_sensor\_rht\_init();

if (sc != SL\_STATUS\_OK) {

sl\_zigbee\_app\_debug\_print("Relative Humidity and Temperature sensor initialization failed.\n");

}

sl\_zigbee\_event\_init(&sendCommandEventOn, sendCommandEventOnHandler);

sl\_zigbee\_event\_init(&sendCommandEventOff,sendCommandEventOffHandler);

sl\_zigbee\_event\_init(&network\_steering\_event\_control, network\_steering\_event\_handler);

sl\_zigbee\_event\_init(&finding\_and\_binding\_event\_control, finding\_and\_binding\_event\_handler);

sl\_zigbee\_event\_init(&attribute\_report\_event\_control, attribute\_report\_event\_handler);

}  
  
  
Light code :

////LIGHT////

#include "app/framework/include/af.h"

#include "sl\_simple\_led\_instances.h"

#include "sl\_simple\_led.h"

void emberAfPluginNetworkSteeringCompleteCallback(EmberStatus status,

uint8\_t totalBeacons,

uint8\_t joinAttempts,

uint8\_t finalState)

{

sl\_zigbee\_app\_debug\_println("%s network %s: 0x%02X", "Join", "complete", status);

}

void emberAfRadioNeedsCalibratingCallback(void)

{

sl\_mac\_calibrate\_current\_channel();

}

// Sending-OnOff-Commands: Step 1

void emberAfPostAttributeChangeCallback(uint8\_t endpoint,

EmberAfClusterId clusterId,

EmberAfAttributeId attributeId,

uint8\_t mask,

uint16\_t manufacturerCode,

uint8\_t type,

uint8\_t size,

uint8\_t\* value)

{

if (clusterId == ZCL\_ON\_OFF\_CLUSTER\_ID

&& attributeId == ZCL\_ON\_OFF\_ATTRIBUTE\_ID

&& mask == CLUSTER\_MASK\_SERVER) {

bool onOff;

if (emberAfReadServerAttribute(endpoint,

ZCL\_ON\_OFF\_CLUSTER\_ID,

ZCL\_ON\_OFF\_ATTRIBUTE\_ID,

(uint8\_t \*)&onOff,

sizeof(onOff)) == EMBER\_ZCL\_STATUS\_SUCCESS) {

if (onOff) {

sl\_led\_turn\_on(&sl\_led\_led0);

} else {

sl\_led\_turn\_off(&sl\_led\_led0);

}

}

}

}

bool emberAfReportAttributesCallback(EmberAfClusterId clusterId,

int8u \*buffer,

int16u bufLen)

{

int16\_t attribute;

// Custom processing done only for cluster 0x405 and 0x402 (RH and Temperature

// measurement clusters)

if (clusterId == ZCL\_RELATIVE\_HUMIDITY\_MEASUREMENT\_CLUSTER\_ID ||

clusterId == ZCL\_TEMP\_MEASUREMENT\_CLUSTER\_ID) {

attribute = (buffer[bufLen-1] << 8) | (buffer[bufLen-2]);

if (clusterId == ZCL\_RELATIVE\_HUMIDITY\_MEASUREMENT\_CLUSTER\_ID) {

emberAfCorePrintln("RH: %2d.%2d %%",

attribute/100,

(attribute % 100));

} else {

emberAfCorePrintln("Temperature: %2d.%2d C",

attribute/100,

(attribute % 100));

}

return true;

}

return false;

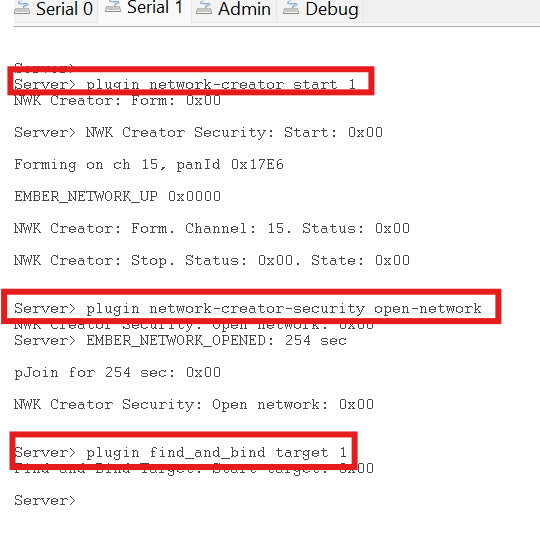
}

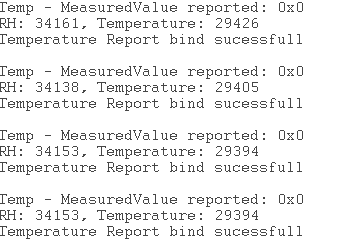
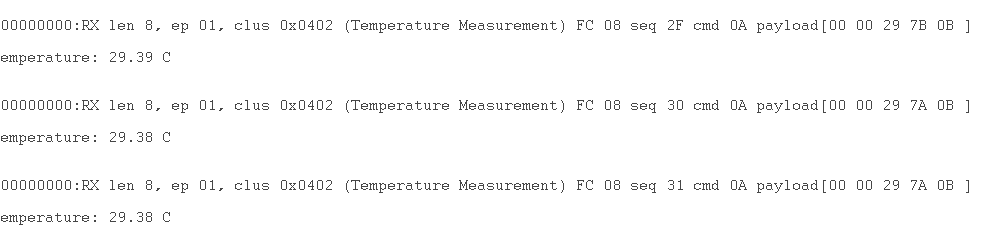
**Workflow:   
On Light side :**

1)Network Formation   
  
 plugin network-creator start 1

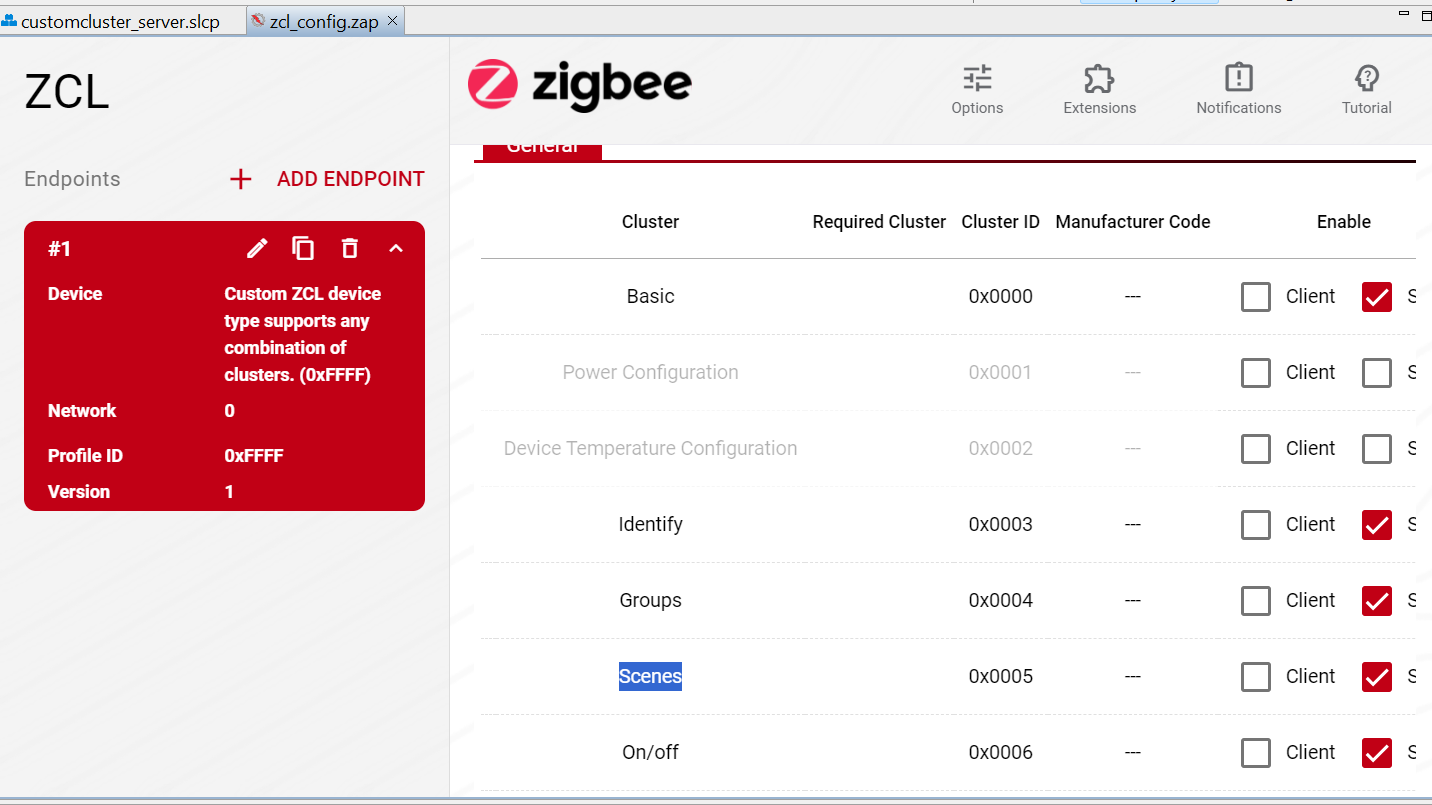
plugin network-creator-security open-network

2) Find and binding the target endpoint:  
  
 plugin find\_and\_bind target 1

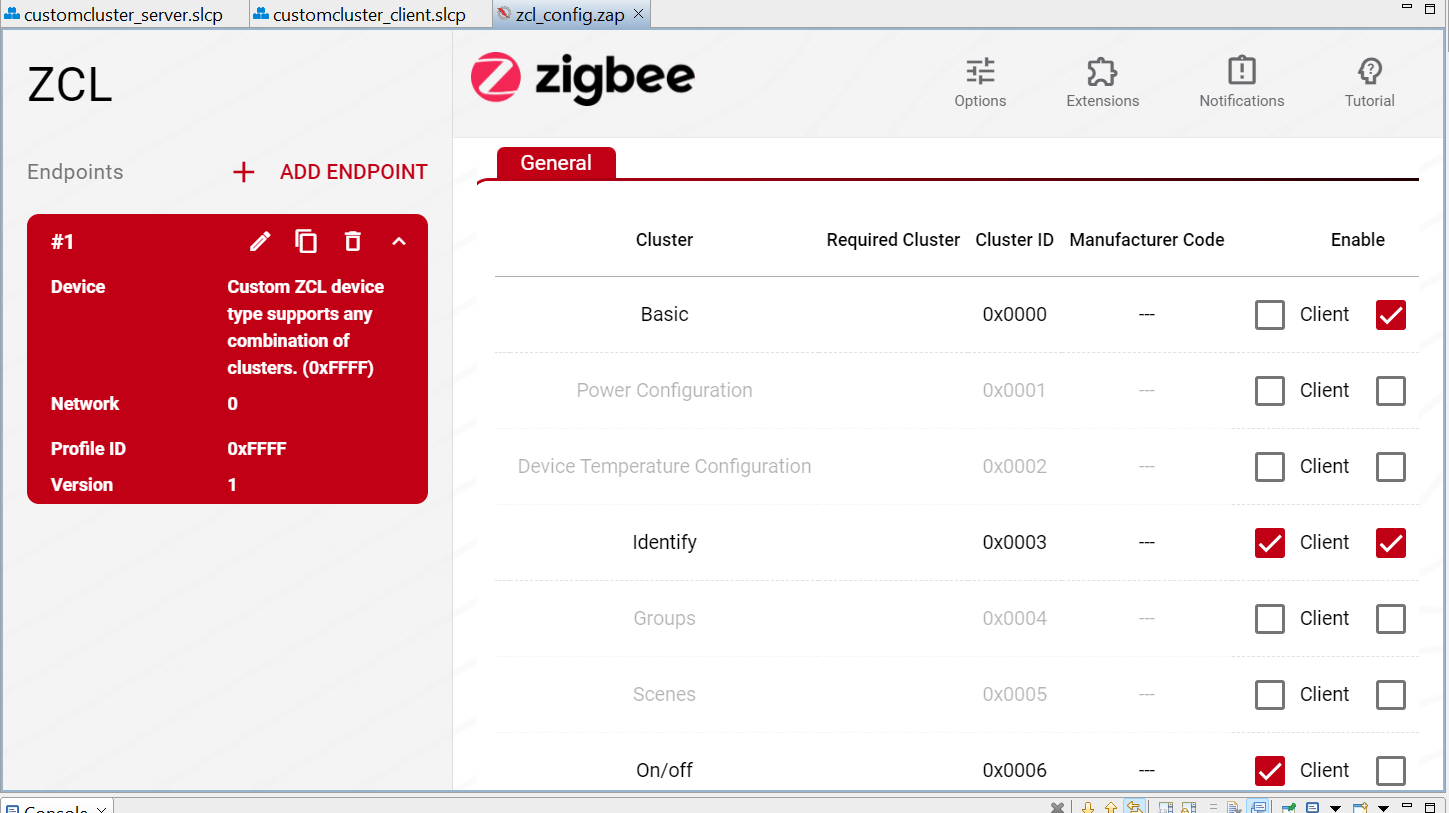
****

**On Switch side :**We have written code such that joining the network is done by pressing the button 0 on the board and enabling the reporting sensor data is done by pressing button 1 on the board.  
  
  
  
  
  
On Light side we get:  
  


(OPTIONAL)

Instead of using an existing endpoint we can create a custom endpoint, here we can configure the endpoints according to our needs.  
 **On Light Side:**  
  
As It is responsible for formation of the network, identifying the devices which are joining, we have to enable **Basic**, **Identify**, **Groups** and **Scenes** as a server(as it is acting as a Coordinator).  
But in our project we are also dealing with ON/OFF configuration, so we have to enable ON/OFF as a server as shown below.  
  
   
  
Now the general configuration is done, we have to enable the temperature measurement as a server and enable reporting in settings as well.

On Switch Side:  
  
Here we only need **basic(client)**, **identify(client and server)** and **ON/OFF(client)** in general configuration and in measurement and sensing we need **temperature measurement** as a client and also need to enable the reporting .



Now by building the project and flashing it into the board by running the commands mentioned in the workflow we are able to send the temperature from client to the server.  
  
  
  
**Joining Additional Router to the Zigbee Network.**

Initially we used 1 Coordinator and 1 Router. Now we will use an additional router in the network which makes the coordinator react to the commands from both the devices.

* Connect the new device and create the project with Zigbee Minimal Configuration.
* Repeat the steps of the previous documents which we followed for the router ( i.e: Endpoints, Software Components).
* Add the router code, build the project and flash the**.s37** file into the new device.
* Check whether install code is derived or not. If not derived, create a batch file and run it with the code given in Forming and Joining Document.
* Now we create a network on Coordinator and from the routers joining the network is done by pressing the button 0 on the board and enabling the reporting sensor data is done by pressing button 1 on the board.