X-CUBE-AWS

rev1.0 09/06/2020

GOAL

Connect STM32 microcontrollers to AWS

PREREQUISITES

Software needed:

- STM32IDE
- X-CUBE-AWS
- TeraTerm

Hardware used in this example:

- 32F769IDISCOVERY

P.S. Be sure to have fully understood the basics of AWS such as devices, certificates and JSON messages.

X-CUBE-AWS

The X-CUBE-AWS Expansion Package consists of a set of libraries and application examples for STM32L4 Series, STM32F4 Series, and STM32F7 Series microcontrollers acting as end devices.

For our example start with opening the example for the 32F769IDISCOVERY board, you can find it in:

..\STM32CubeExpansion_Cloud_AWS_Vx.y.z\Projects\STM32F769I -Discovery\Applications\Cloud\AWS

FreeRTOS

The X-CUBE-AWS libraries works on FreeRTOS, a Real-time operating system for microcontrollers. Using a OS allows the board to perform multiple tasks at almost the same time, indipendently so it’s important to have a basic understanding in how it works and the importance of threads.

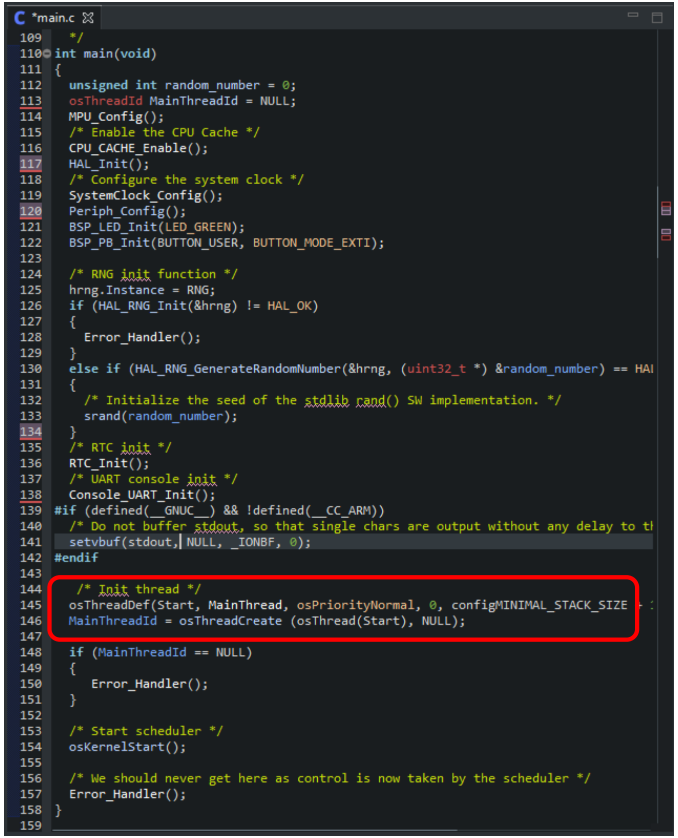
You can find all the infos you may need here.

X-CUBE-AWS Example

Open the *main.c* file and go to the main function: after the initalization of the board, the ***MainThread*** will start: notice that the infinite loop is missing.

Scrolling down (or by searching it on the right tab) find the MainThread function: the aim of this thread is to create all the objects we need for the OS such as threads, tasks, timers and so.

After all of them have been created, the MainThread terminates, while all the other threads will continue to run indipendently.



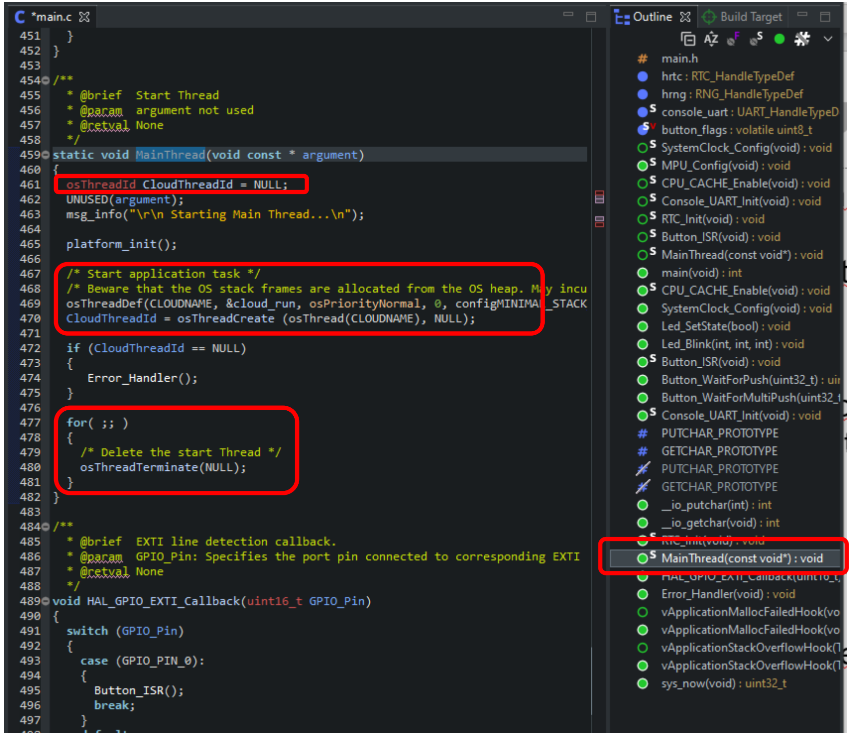
```c
  */
int main(void)
{
  unsigned int random_number = 0;
  osThreadId MainThreadId = NULL;
  MPU_Config();
  /* Enable the CPU Cache */
  CPU_CACHE_Enable();
  HAL_Init();
  /* Configure the system clock */
  SystemClock_Config();
  Periph_Config();
  BSP_LED_Init(LED_GREEN);
  BSP_PB_Init(BUTTON_USER, BUTTON_MODE_EXTI);

  /* RNG init function */
  hrng.Instance = RNG;
  if (HAL_RNG_Init(&hrng) != HAL_OK)
  {
    Error_Handler();
  }
  else if (HAL_RNG_GenerateRandomNumber(&hrng, (uint32_t *) &random_number) == HAI
  {
    /* Initialize the seed of the stdlib rand() SW implementation. */
    srand(random_number);
  }
  /* RTC init */
  RTC_Init();
  /* UART console init */
  Console_UART_Init();
#if (defined(__GNUC__) && !defined(__CC_ARM))
  /* Do not buffer stdout, so that single chars are output without any delay to th
  setvbuf(stdout, NULL, _IONBF, 0);
#endif

   /* Init thread */
  osThreadDef(Start, MainThread, osPriorityNormal, 0, configMINIMAL_STACK_SIZE
  MainThreadId = osThreadCreate (osThread(Start), NULL);

  if (MainThreadId == NULL)
  {
     Error_Handler();
  }

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */
  Error_Handler();
}

```

X-CUBE-AWS Example

Open the *main.c* file and go to the main function: after the initalization of the board, the ***MainThread*** will start: notice that the infinite loop is missing.

Scrolling down (or by searching it on the right tab) find the MainThread function: the aim of this thread is to create all the objects we need for the OS such as threads, tasks, timers and so.

After all of them have been created, the MainThread terminates, while all the other threads will continue to run indipendently.



```
  }
}

/**
  * @brief  Start Thread
  * @param  argument not used
  * @retval None
  */
static void MainThread(void const * argument)
{
  osThreadId CloudThreadId = NULL;
  UNUSED(argument);
  msg_info("\r\n Starting Main Thread...\n");

  platform_init();

  /* Start application task */
  /* Beware that the OS stack frames are allocated from the OS heap. May incu
  osThreadDef(CLOUDNAME, &cloud_run, osPriorityNormal, 0, configMINIMAL_STACK
  CloudThreadId = osThreadCreate (osThread(CLOUDNAME), NULL);

  if (CloudThreadId == NULL)
  {
    Error_Handler();
  }

  for( ;; )
  {
    /* Delete the start Thread */
    osThreadTerminate(NULL);
  }
}

/**
  * @brief  EXTI line detection callback.
  * @param  GPIO_Pin: Specifies the port pin connected to corresponding EXTI
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  switch (GPIO_Pin)
  {
    case (GPIO_PIN_0):
    {
      Button_ISR();
      break;
    }
```

```
main.h
hrtc : RTC_HandleTypeDef
hrng : RNG_HandleTypeDef
console_uart : UART_HandleTypeD
button_flags : volatile uint8_t
SystemClock_Config(void) : void
MPU_Config(void) : void
CPU_CACHE_Enable(void) : void
Console_UART_Init(void) : void
RTC_Init(void) : void
Button_ISR(void) : void
MainThread(const void*) : void
main(void) : int
CPU_CACHE_Enable(void) : void
SystemClock_Config(void) : void
Led_SetState(bool) : void
Led_Blink(int, int, int) : void
Button_ISR(void) : void
Button_WaitForPush(uint32_t) : uir
Button_WaitForMultiPush(uint32_t
Console_UART_Init(void) : void
PUTCHAR_PROTOTYPE
GETCHAR_PROTOTYPE
PUTCHAR_PROTOTYPE
GETCHAR_PROTOTYPE
__io_putchar(int) : int
__io_getchar(void) : int
RTC_Init(void) : void
MainThread(const void*) : void
HAL_GPIO_EXTI_Callback(uint16_t)
Error_Handler(void) : void
vApplicationMallocFailedHook(vo
vApplicationMallocFailedHook(vo
vApplicationStackOverflowHook(T
vApplicationStackOverflowHook(T
sys_now(void) : uint32_t
```

cloud_run()

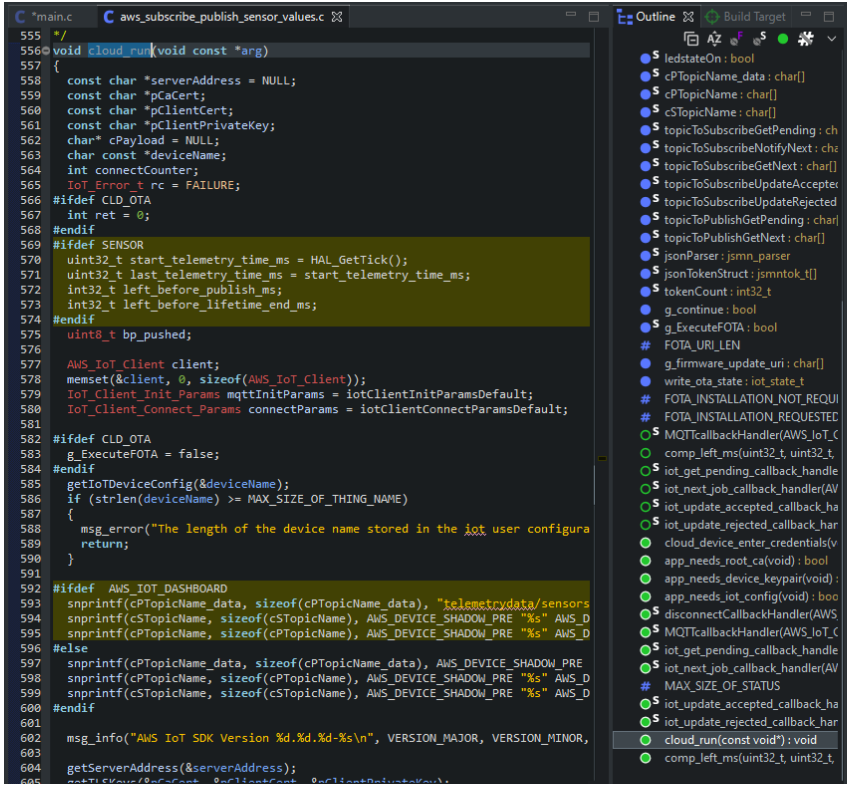
The line:

osThreadDef(CLOUDNAME, &cloud_run,..)

Defines a thread called *CLOUDNAME* that will run the cloud_run function.

Select cloud_run, righ click and go to definition: the file *aws_subscribe_publish_sensor_values.c* will open.

Here are written the main functions that allows our system to communicate with AWS.



```c
*/
void cloud_run(void const *arg)
{
  const char *serverAddress = NULL;
  const char *pCaCert;
  const char *pClientCert;
  const char *pClientPrivateKey;
  char* cPayload = NULL;
  char const *deviceName;
  int connectCounter;
  IoT_Error_t rc = FAILURE;
#ifdef CLD_OTA
  int ret = 0;
#endif
#ifdef SENSOR
  uint32_t start_telemetry_time_ms = HAL_GetTick();
  uint32_t last_telemetry_time_ms = start_telemetry_time_ms;
  int32_t left_before_publish_ms;
  int32_t left_before_lifetime_end_ms;
#endif
  uint8_t bp_pushed;

  AWS_IoT_Client client;
  memset(&client, 0, sizeof(AWS_IoT_Client));
  IoT_Client_Init_Params mqttInitParams = iotClientInitParamsDefault;
  IoT_Client_Connect_Params connectParams = iotClientConnectParamsDefault;

#ifdef CLD_OTA
  g_ExecuteFOTA = false;
#endif
  getIoTDeviceConfig(&deviceName);
  if (strlen(deviceName) >= MAX_SIZE_OF_THING_NAME)
  {
    msg_error("The length of the device name stored in the iot user configura
    return;
  }

#ifdef  AWS_IOT_DASHBOARD
  snprintf(cPTopicName_data, sizeof(cPTopicName_data), "telemetrydata/sensors
  snprintf(cSTopicName, sizeof(cSTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_D
  snprintf(cPTopicName, sizeof(cPTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_D
#else
  snprintf(cPTopicName_data, sizeof(cPTopicName_data), AWS_DEVICE_SHADOW_PRE
  snprintf(cPTopicName, sizeof(cPTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_D
  snprintf(cSTopicName, sizeof(cSTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_D
#endif

  msg_info("AWS IoT SDK Version %d.%d.%d-%s\n", VERSION_MAJOR, VERSION_MINOR,

  getServerAddress(&serverAddress);
  getTLSKeys(&pCaCert, &pClientCert, &pClientPrivateKey);
```

```
ledstateOn : bool
cPTopicName_data : char[]
cPTopicName : char[]
cSTopicName : char[]
topicToSubscribeGetPending : ch
topicToSubscribeNotifyNext : cha
topicToSubscribeGetNext : char[]
topicToSubscribeUpdateAcceptec
topicToSubscribeUpdateRejected
topicToPublishGetPending : char|
topicToPublishGetNext : char[]
jsonParser : jsmn_parser
jsonTokenStruct : jsmntok_t[]
tokenCount : int32_t
g_continue : bool
g_ExecuteFOTA : bool
FOTA_URI_LEN
g_firmware_update_uri : char[]
write_ota_state : iot_state_t
FOTA_INSTALLATION_NOT_REQUI
FOTA_INSTALLATION_REQUESTED
MQTTcallbackHandler(AWS_IoT_C
comp_left_ms(uint32_t, uint32_t,
iot_get_pending_callback_handle
iot_next_job_callback_handler(AV
iot_update_accepted_callback_ha
iot_update_rejected_callback_har
cloud_device_enter_credentials(v
app_needs_root_ca(void) : bool
app_needs_device_keypair(void) :
app_needs_iot_config(void) : boo
disconnectCallbackHandler(AWS_
MQTTcallbackHandler(AWS_IoT_C
iot_get_pending_callback_handle
iot_next_job_callback_handler(AV
MAX_SIZE_OF_STATUS
iot_update_accepted_callback_ha
iot_update_rejected_callback_har
cloud_run(const void*) : void
comp_left_ms(uint32_t, uint32_t,
```

ATTENTION

There are 2 errors on the example given by ST.

Using the find tool in the IDE, substitute the following line (the error is repeated twice) :

```
paramsQOS1.payloadLen = strlen(cPayload) + 1;
```

With:

```
paramsQOS1.payloadLen = strlen(cPayload);
```

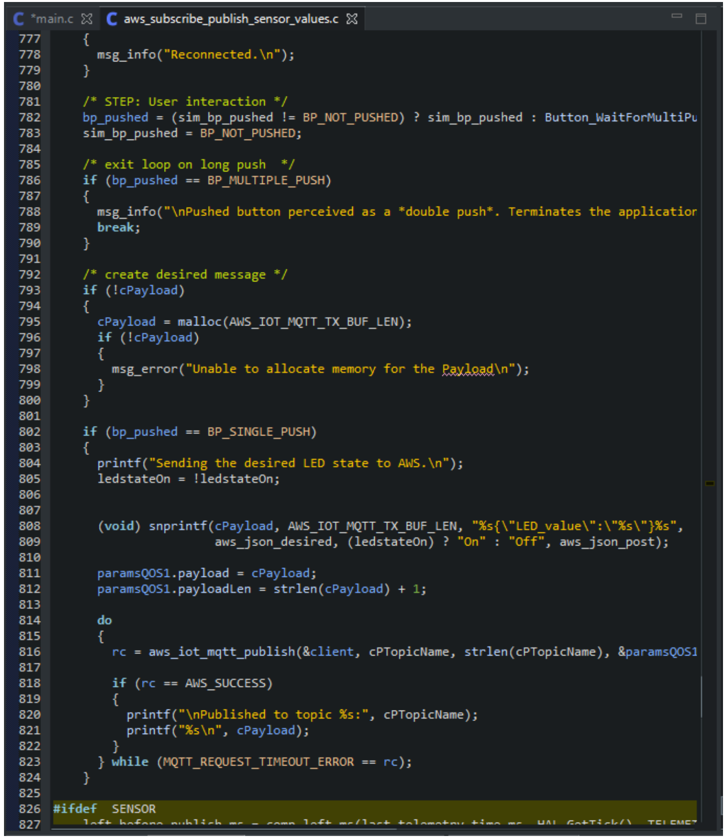
cloud_run

cloud_run() function provides us connection with AWS cloud, subscribing and publishing to topics, checking for errors all the time.

You should never touch this function for the most part.

As you can read on the ST’s papers (you can find the official guide here), the example provided will publish a message to the specific topic in order to change the built-in LED state when the blue button has been pushed.

This functionality is written between lines 802 and 824.



```c
777     {
778       msg_info("Reconnected.\n");
779     }
780 
781     /* STEP: User interaction */
782     bp_pushed = (sim_bp_pushed != BP_NOT_PUSHED) ? sim_bp_pushed : Button_WaitForMultiPu
783     sim_bp_pushed = BP_NOT_PUSHED;
784 
785     /* exit loop on long push  */
786     if (bp_pushed == BP_MULTIPLE_PUSH)
787     {
788       msg_info("\nPushed button perceived as a *double push*. Terminates the applicatior
789       break;
790     }
791 
792     /* create desired message */
793     if (!cPayload)
794     {
795       cPayload = malloc(AWS_IOT_MQTT_TX_BUF_LEN);
796       if (!cPayload)
797       {
798         msg_error("Unable to allocate memory for the Payload\n");
799       }
800     }
801 
802     if (bp_pushed == BP_SINGLE_PUSH)
803     {
804       printf("Sending the desired LED state to AWS.\n");
805       ledstateOn = !ledstateOn;
806 
807 
808       (void) snprintf(cPayload, AWS_IOT_MQTT_TX_BUF_LEN, "%s{\"LED_value\":\"%s\"}%s",
809                       aws_json_desired, (ledstateOn) ? "On" : "Off", aws_json_post);
810 
811       paramsQOS1.payload = cPayload;
812       paramsQOS1.payloadLen = strlen(cPayload) + 1;
813 
814       do
815       {
816         rc = aws_iot_mqtt_publish(&client, cPTopicName, strlen(cPTopicName), &paramsQOS1
817 
818         if (rc == AWS_SUCCESS)
819         {
820           printf("\nPublished to topic %s:", cPTopicName);
821           printf("%s\n", cPayload);
822         }
823       } while (MQTT_REQUEST_TIMEOUT_ERROR == rc);
824     }
825 
826 #ifdef  SENSOR
827     left_before_publish_ms = comp_left_ms(last_telemetry_time_ms, HAL_GetTick(), TELEMET
```

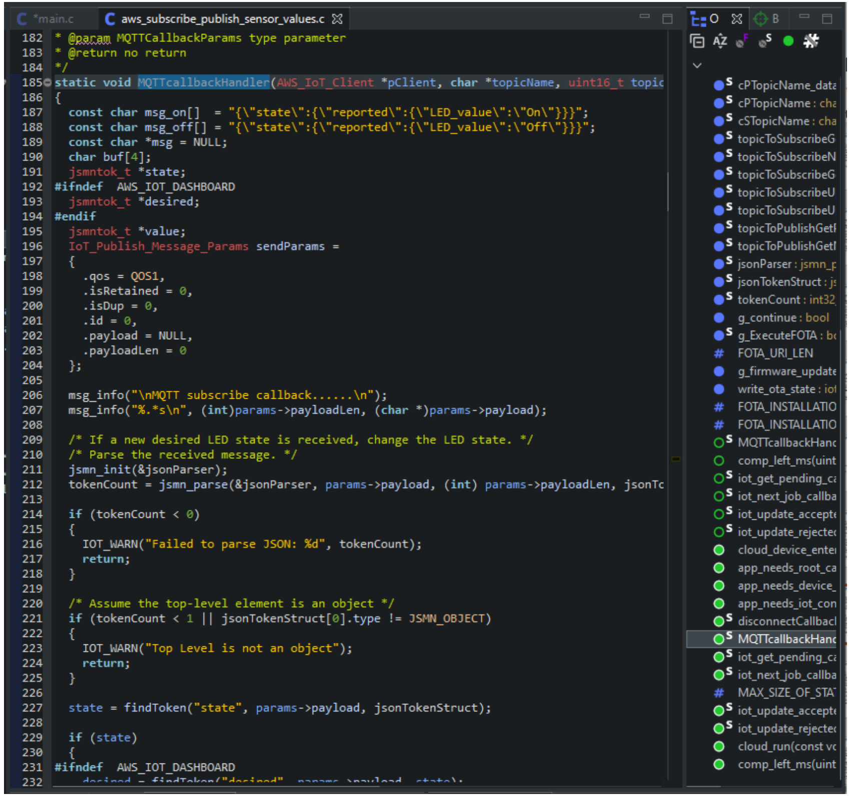
MQTTcallbackHandler

When any message has been published on the topic wich the board subscribed, the MQTTcallbackHandler triggers itself.

This function is written in the same file, at line 185 (try to find it with the browser tab at right) .

Any message that arrives in the topic is stored as payload: the jsmn parser interprets it and in this example, if the message is valid, the board will publish a message with the result of the led toggle, and set the led state.

But, where are defined the topics ?



```c
182 * @param MQTTCallbackParams type parameter
183 * @return no return
184 */
185 static void MQTTcallbackHandler(AWS_IoT_Client *pClient, char *topicName, uint16_t topic
186 {
187   const char msg_on[]  = "{\"state\":{\"reported\":{\"LED_value\":\"On\"}}}";
188   const char msg_off[] = "{\"state\":{\"reported\":{\"LED_value\":\"Off\"}}}";
189   const char *msg = NULL;
190   char buf[4];
191   jsmntok_t *state;
192 #ifndef  AWS_IOT_DASHBOARD
193   jsmntok_t *desired;
194 #endif
195   jsmntok_t *value;
196   IoT_Publish_Message_Params sendParams =
197   {
198     .qos = QOS1,
199     .isRetained = 0,
200     .isDup = 0,
201     .id = 0,
202     .payload = NULL,
203     .payloadLen = 0
204   };
205 
206   msg_info("\nMQTT subscribe callback......\n");
207   msg_info("%.*s\n", (int)params->payloadLen, (char *)params->payload);
208 
209   /* If a new desired LED state is received, change the LED state. */
210   /* Parse the received message. */
211   jsmn_init(&jsonParser);
212   tokenCount = jsmn_parse(&jsonParser, params->payload, (int) params->payloadLen, jsonTc
213 
214   if (tokenCount < 0)
215   {
216     IOT_WARN("Failed to parse JSON: %d", tokenCount);
217     return;
218   }
219 
220   /* Assume the top-level element is an object */
221   if (tokenCount < 1 || jsonTokenStruct[0].type != JSMN_OBJECT)
222   {
223     IOT_WARN("Top Level is not an object");
224     return;
225   }
226 
227   state = findToken("state", params->payload, jsonTokenStruct);
228 
229   if (state)
230   {
231 #ifndef  AWS_IOT_DASHBOARD
232     desired = findToken("desired", params->payload, state);
```

Topics

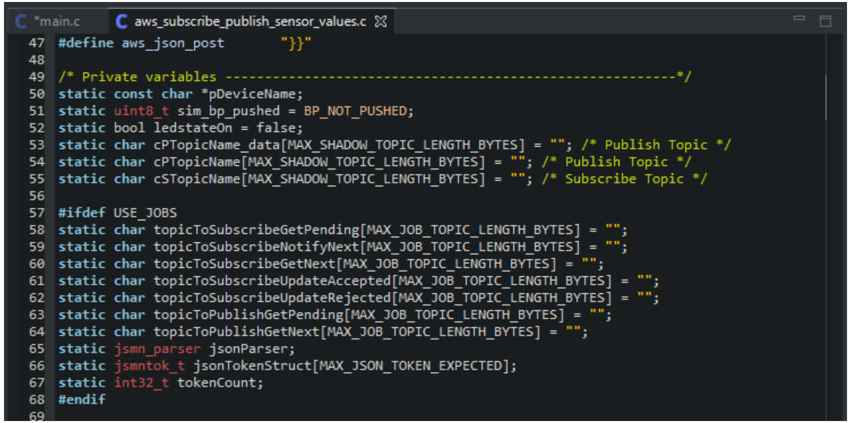
The topics for subscribing and publishing are defined at the beginning of this file at lines 54 and 55 but they are empty. Right click on the topic name and click on call hierarchy: you will find where the topics strings are filled (Lines 592-600).

They consists on a sum of different strings, containing the device name too.

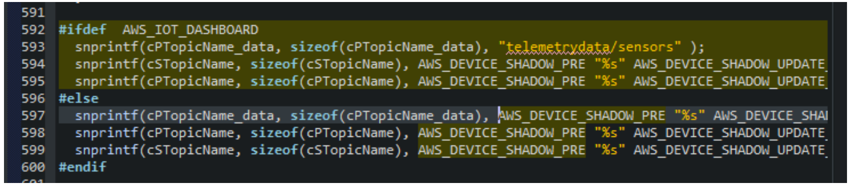
An example is:

$aws/things/afr_test/shadow/update/accepted

As a *cPTopicName* for a device named *afr_test .*



```
47  #define aws_json_post       "}}"
48
49  /* Private variables -------------------------------------------------------*/
50  static const char *pDeviceName;
51  static uint8_t sim_bp_pushed = BP_NOT_PUSHED;
52  static bool ledstateOn = false;
53  static char cPTopicName_data[MAX_SHADOW_TOPIC_LENGTH_BYTES] = ""; /* Publish Topic */
54  static char cPTopicName[MAX_SHADOW_TOPIC_LENGTH_BYTES] = ""; /* Publish Topic */
55  static char cSTopicName[MAX_SHADOW_TOPIC_LENGTH_BYTES] = ""; /* Subscribe Topic */
56
57  #ifdef USE_JOBS
58  static char topicToSubscribeGetPending[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
59  static char topicToSubscribeNotifyNext[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
60  static char topicToSubscribeGetNext[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
61  static char topicToSubscribeUpdateAccepted[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
62  static char topicToSubscribeUpdateRejected[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
63  static char topicToPublishGetPending[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
64  static char topicToPublishGetNext[MAX_JOB_TOPIC_LENGTH_BYTES] = "";
65  static jsmn_parser jsonParser;
66  static jsmntok_t jsonTokenStruct[MAX_JSON_TOKEN_EXPECTED];
67  static int32_t tokenCount;
68  #endif
69
```



```
591
592 #ifdef  AWS_IOT_DASHBOARD
593   snprintf(cPTopicName_data, sizeof(cPTopicName_data), "telemetrydata/sensors" );
594   snprintf(cSTopicName, sizeof(cSTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_DEVICE_SHADOW_UPDATE_
595   snprintf(cPTopicName, sizeof(cPTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_DEVICE_SHADOW_UPDATE_
596 #else
597   snprintf(cPTopicName_data, sizeof(cPTopicName_data), AWS_DEVICE_SHADOW_PRE "%s" AWS_DEVICE_SHA
598   snprintf(cPTopicName, sizeof(cPTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_DEVICE_SHADOW_UPDATE_
599   snprintf(cSTopicName, sizeof(cSTopicName), AWS_DEVICE_SHADOW_PRE "%s" AWS_DEVICE_SHADOW_UPDATE_
600 #endif
```

Let’s Start !

This was an overview of how the X-CUBE-AWS example works, anyway I suggest you to explore better the functionality of FreeRTOS and of the entire X-CUBE-AWS libraries.

Start with creating a device in the AWS Console (for this example it is *afr_test*), create and attach policies to the device and download the certificates (See the AWS basics slides as reference).

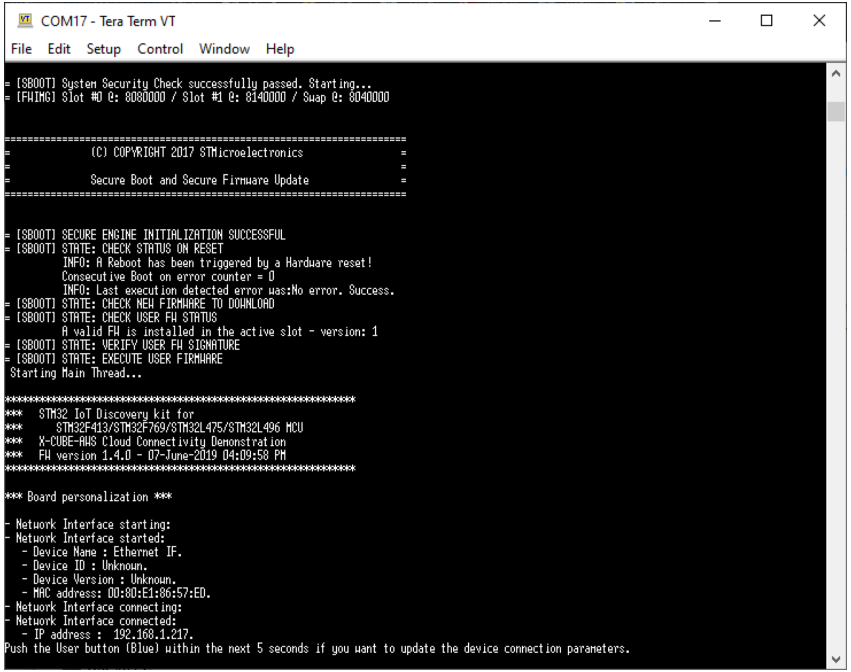
Once done, download the firmware on the board but keep in mind that debugging firmware based on FreeRTOS is not the same as previous examples: it will not work proprierly. See here to more infos.

Console Logging

Open *TeraTerm* software select *Setup→Restore* setup and choose the file provided with the X-CUBE-AWS library in the utilities folder.

Select *Setup→Serial* Port and start the communication, then **reset** the board.

The board will begin to log on the console, asking for credential updates in the first 5 seconds after the network has been configured. Press the blue button within 5 seconds for updating credentials such as broker endpoint, certificates and so.



```
= [SBOOT] System Security Check successfully passed. Starting...
= [FWIMG] Slot #0 @: 8080000 / Slot #1 @: 8140000 / Swap @: 8040000


======================================================================
=              (C) COPYRIGHT 2017 STMicroelectronics                 =
=                                                                    =
=              Secure Boot and Secure Firmware Update                =
======================================================================


= [SBOOT] SECURE ENGINE INITIALIZATION SUCCESSFUL
= [SBOOT] STATE: CHECK STATUS ON RESET
          INFO: A Reboot has been triggered by a Hardware reset!
          Consecutive Boot on error counter = 0
          INFO: Last execution detected error was:No error. Success.
= [SBOOT] STATE: CHECK NEW FIRMWARE TO DOWNLOAD
= [SBOOT] STATE: CHECK USER FW STATUS
          A valid FW is installed in the active slot - version: 1
= [SBOOT] STATE: VERIFY USER FW SIGNATURE
= [SBOOT] STATE: EXECUTE USER FIRMWARE
 Starting Main Thread...

*************************************************************
***   STM32 IoT Discovery kit for
***      STM32F413/STM32F769/STM32L475/STM32L496 MCU
***   X-CUBE-AWS Cloud Connectivity Demonstration
***   FW version 1.4.0 - 07-June-2019 04:09:58 PM
*************************************************************

*** Board personalization ***

- Network Interface starting:
- Network Interface started:
   - Device Name : Ethernet IF.
   - Device ID : Unknown.
   - Device Version : Unknown.
   - MAC address: 00:80:E1:86:57:ED.
- Network Interface connecting:
- Network Interface connected:
   - IP address :  192.168.1.217.
Push the User button (Blue) within the next 5 seconds if you want to update the device connection parameters.
```

Enter Credential

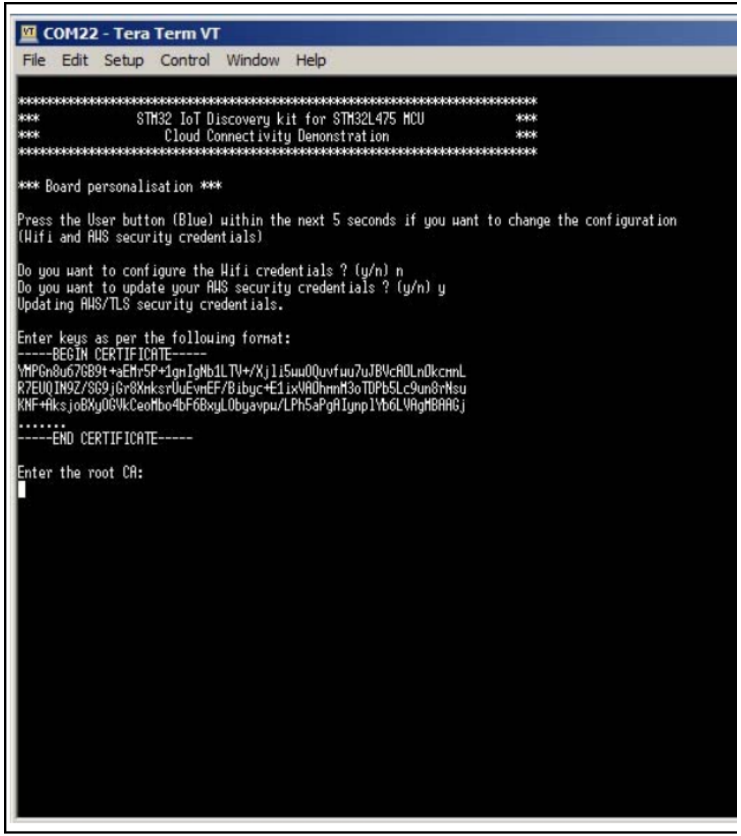
You will be prompted to paste the certicates on the console: open the certificate file with MS Word (notepad will not work) select the text and paste it as in the picture.

If you have any troubles in pasting the text, go to *edit→paste.*

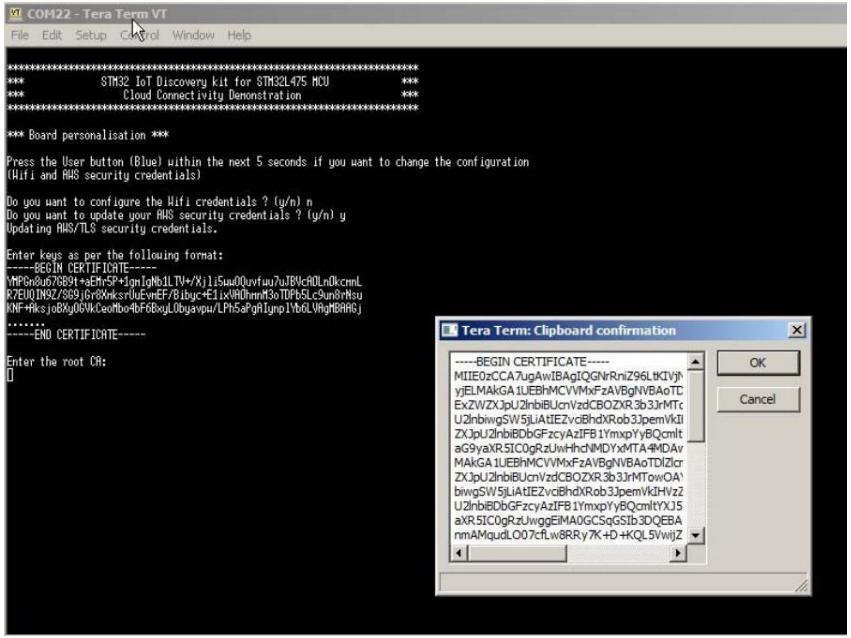
Press enter.

If the text has been inserted correctly, it will be echoed in the console, as in the picture.

Continue with the other certificates.

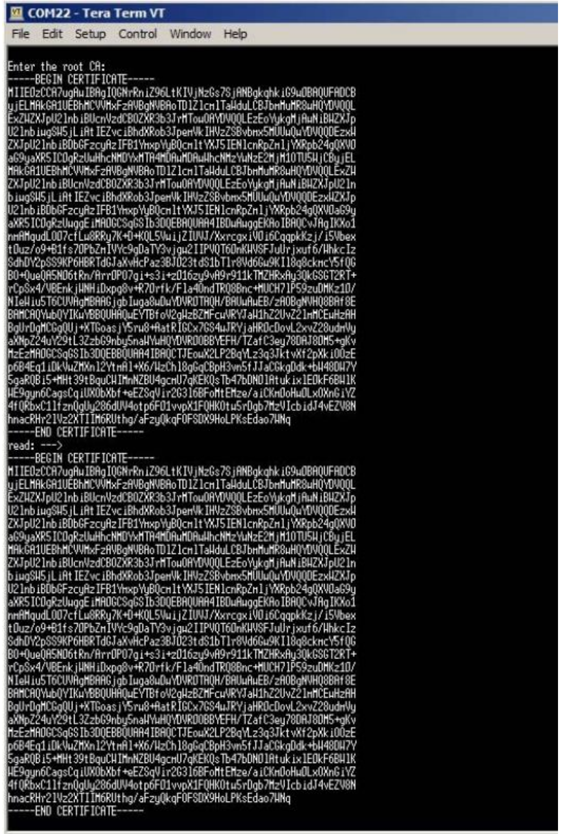


```
**************************************************************************
***            STM32 IoT Discovery kit for STM32L475 MCU               ***
***                Cloud Connectivity Demonstration                    ***
**************************************************************************

*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to change the configuration
(Wifi and AWS security credentials)

Do you want to configure the Wifi credentials ? (y/n) n
Do you want to update your AWS security credentials ? (y/n) y
Updating AWS/TLS security credentials.

Enter keys as per the following format:
-----BEGIN CERTIFICATE-----
YMPGn8u67GB9t+aEMr5P+1gmIgNb1LTV+/Xjli5wwOQuvfwu7uJBVcAOLnOkcmnL
R7EUQIN9Z/SG9jGr8XmksrUuEvmEF/Bibyc+E1ixVAOhmmM3oTDPb5Lc9un8rNsu
KNF+AksjoBXyOGVkCeoMbo4bF6BxyLObyavpw/LPh5aPgAIynplYb6LVAgMBAAGj
.......
-----END CERTIFICATE-----

Enter the root CA:
█
```



```
*****************************************************************************
***              STM32 IoT Discovery kit for STM32L475 MCU                ***
***                  Cloud Connectivity Demonstration                     ***
*****************************************************************************

*** Board personalisation ***

Press the User button (Blue) within the next 5 seconds if you want to change the configuration
(Wifi and AWS security credentials)

Do you want to configure the Wifi credentials ? (y/n) n
Do you want to update your AWS security credentials ? (y/n) y
Updating AWS/TLS security credentials.

Enter keys as per the following format:
-----BEGIN CERTIFICATE-----
VMPGn8u67GB9t+aEMr5P+1gmIgNb1LTV+/Xjli5wwOQuvfwu7uJBVcAOLnOkcmnL
R7EUQIN9Z/SG9jGr8XmksrUuEvmEF/Bibyc+E1ixVAOhmnM3oTDPb5Lc9un8rNsu
KNF+AksjoBXyOGVkCeoMbo4bF6BxyLObyavpw/LPh5aPgAIynplYb6LVAgMBAAGj
.......
-----END CERTIFICATE-----

Enter the root CA:
```

```
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjN
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTD
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTc
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkII
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmlt
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAv
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcr
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAY
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZ
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBA
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZ
```



```
Enter the root CA:
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
read: --->
-----BEGIN CERTIFICATE-----
MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB
yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0
aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL
MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW
ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln
biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp
U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y
aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1
nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex
t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz
SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG
BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+
rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/
NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E
BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH
BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy
aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv
MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE
p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y
5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK
WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N
hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
-----END CERTIFICATE-----
```

Start Sending Data

After all the credential have been written in the flash memory (you have to insert credential only once), after checking for fimware updates, it will connect to the broker endpoint.

Push the blue buttom to publish the message in the topic.

To check if all is working properly, log on AWS IoT core console, subscribe and publish to the same topics.