

Fall 2022 - CSCE 438/838: IoT - Lab 4 - Connecting Things to the Internet

Introduction

Last week we implemented M2M type network with our IoT nodes. In this lab, we will evolve our systems to make it an IoT system. The first three labs have already help you to understand the basics of embedded system to help you develop the Things, but IoT is not just about embedded systems. It's all about **connectivity**. The goal of this lab is to walk through that whole process and get a “Hello World!” message from a remote device, into a gateway, and onto the internet. First we will create the gateway, then we will fire up a device to send the data, and finally we will create an internet application to look for the data.

Takeaways from last week

- Modular design of the system for easy upgrade of system components
- Wireless connectivity with radio

IoT vs M2M

IoT systems may incorporate some M2M nodes (such as a Bluetooth mesh using non-IP communication), but aggregates data at an edge router or gateway. An edge appliance like a gateway or router serves as the entry point onto the internet. Alternatively, some sensors with more substantial computing power can push the internet networking layers onto the sensor itself. Regardless of where the internet *on-ramp* exists, the fact that it has a method of tying into the internet fabric is what defines IoT.

By moving data onto the internet for sensors, edge processors, and smart devices, the legacy world of cloud services can be applied to the simplest of devices. Before cloud technology and mobile communication became mainstream and cost-effective, simple sensors and embedded computing devices in the field had no good means of communicating data globally in seconds, storing information for perpetuity, and analyzing data to find trends and patterns. As cloud technologies advanced, wireless communication systems became pervasive, new energy devices like lithium-ion became cost-effective, and machine learning models evolved to produce actionable value. This greatly improved the IoT value proposition. Without these technologies coming together when they did, we would still be in an M2M world.

LoRaWAN Overview

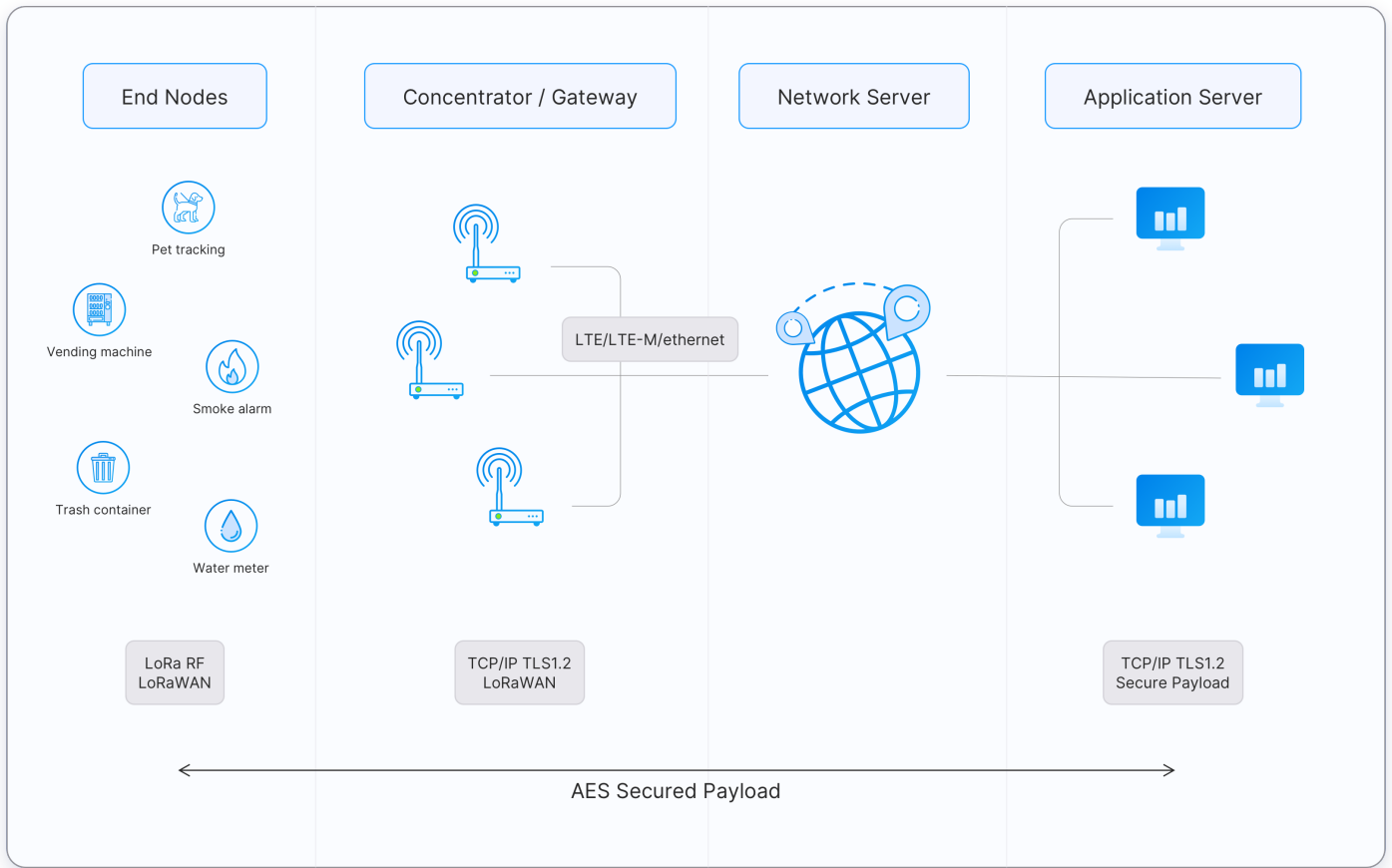
LoRaWAN is a media access control (MAC) protocol for wide area networks. It is designed to allow low-powered devices to communicate with Internet-connected applications over long range wireless connections. LoRaWAN can be mapped to the second and third layer of the OSI model. It is implemented on top of LoRa or FSK modulation in industrial, scientific and medical (ISM) radio bands. The LoRaWAN protocols are defined by the [LoRa Alliance](#) and formalized in the LoRaWAN Specification which can be [downloaded](#) on the LoRa Alliance website.

IoT is the idea that we can add inter-connectivity to a large portion of the things we interact with on a day-to-day basis. For example, if your refrigerator kept track of what was inside and could talk to your cell phone, then when you were at the store you wouldn't be left wondering if you need to buy milk or not. So the Internet of Things is about **connectivity**.

Connectivity is well-solved in the home with WiFi and Bluetooth, but what if your refrigerator was in the middle of a field without access to the internet? This is where **LoRa** comes in. LoRa is "Long Range" radio which was designed for low power consumption and long range transmissions at the expense of bandwidth. This means you can send a little bit of data a long way.

Then you need something that is dedicated to bridge from LoRa messages to internet traffic - called a "**gateway**." As long as you have something that can speak both LoRa and "Internet" then you could make your own solution, but there is a easier and better option. This is where LoRaWAN comes in.

LoRaWAN is a public specification for the system that would be at Starbucks listening for messages from the fridge. One important benefits of LoRaWAN is that you can send encrypted data during transmission and that your fridge could get up and walk to the next state over (again - just a metaphor!) and the messages could still be picked up by a gateway that someone else had built. Since the messages are secure that person won't know about your stinky cheese but the message will still get back to you over the internet. Groups like [The Things Network](#), [Azure IoT Hub](#), [AWS IoT](#) organize everyone's efforts to make this possible.



Terminology

- **End Device, Node, Mote** - an object with an embedded low-power communication device.
- **Gateway** - antennas that receive broadcasts from End Devices and send data back to End Devices.
- **Network Server** - servers that route messages from End Devices to the right Application, and back.
- **Application** - a piece of software, running on a server.
- **Uplink Message** - a message from a Device to an Application.
- **Downlink Message** - a message from an Application to a Device.

Things in your hand

Your Device - Sparkfun Pro RF

A “device” is the remote system that is sending (and in some cases receiving) data. We will set up a device to actually send the “Hello World!” message in the section “Turning a Gateway into A Device.”

Your Gateway - Sparkfun ESP32 Things 1-channel Gateway

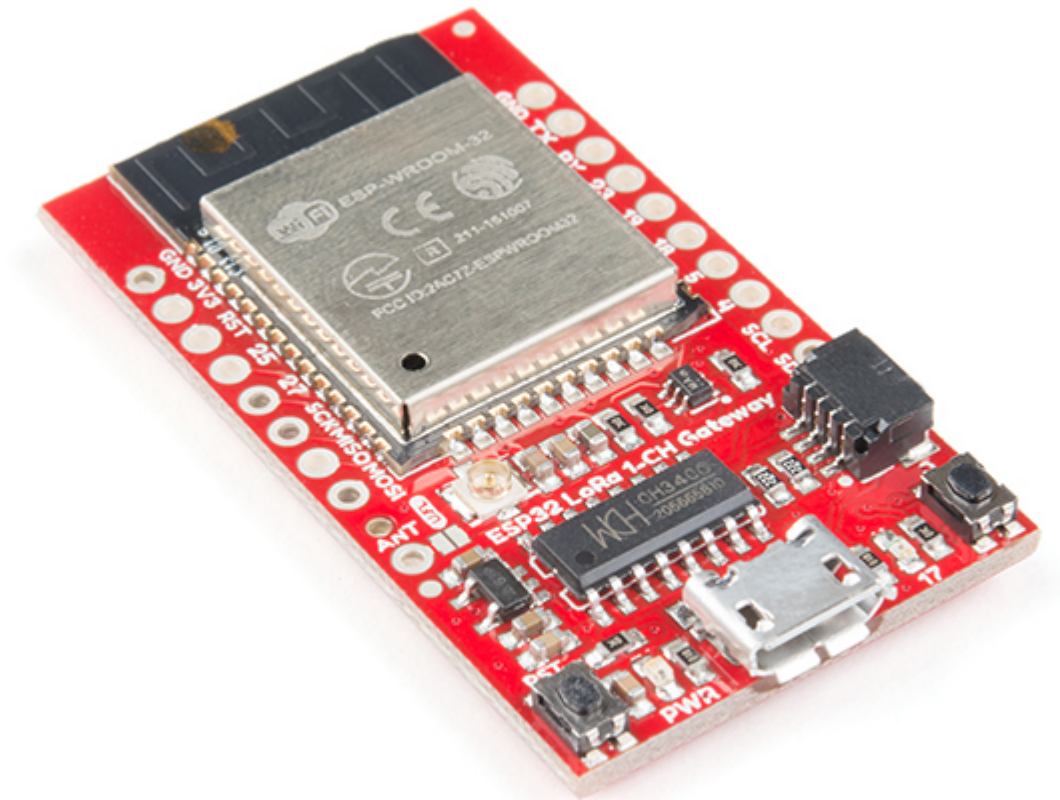
The name of the ESP LoRa Gateway 1-Channel is a dead giveaway. With all its glorious wireless connectivity it acts as the

bridge that speaks both WiFi and LoRa. The section “Single-Channel LoRaWAN Gateway” will cover all the steps you need to make this part.

With these, we can start off the lab and build an IoT!

Hardware

SparkFun LoRa Gateway - 1-Channel (ESP32)



- ESP32-WROOM-32 module

- WiFi, BT+BLE microcontroller
- Integrated PCB antenna
- **Hope RFM95W LoRa modem**
 - Frequency range: 868/915 MHz
 - Spread factor: 6-12
 - SPI control interface
- U.FL antenna connector for LoRa radio
- Reset and ESP32 pin0 buttons
- 14 GPIO ESP32 pin-breakouts
- Power and user LEDs

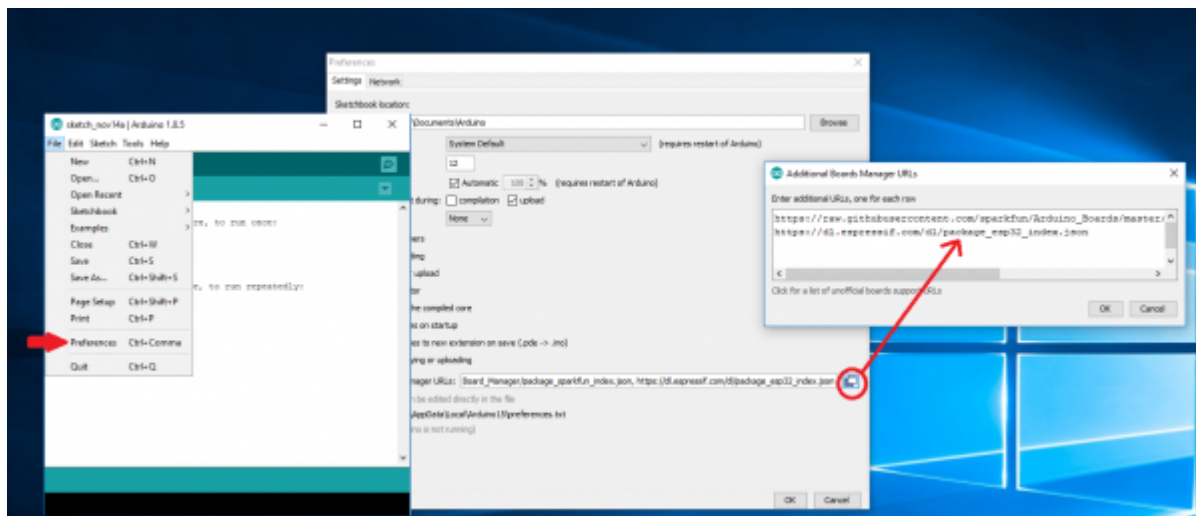
1. Setting Up 1-channel Gateway

Installing ESP32 Arduino Core

The ESP32's relationship with Arduino is growing and now it is very easy to install the core - the Arduino IDE can handle it nearly on its own. All you need to do is make sure you have Arduino IDE version 1.8 or later, then paste

```
https://dl.espressif.com/dl/package_esp32_index.json
```

into the *Additional Board Manager URLs* field of the preferences window, if you have existing URLs, separate them with a comma.



Now accept the changes and restart the Arduino IDE. Next open the **Board Manager** from the top of **Tools > Board** and search for ESP32. Click “Install” on the search result, after a little while the text besides the name should change to “Installed.” Re-start the IDE for good measure.

Upload Blink

To make sure everything is ready, let’s blink the LED. Make sure the correct board is selected (SparkFun ESP32 Things if you installed the variant like we did above) and select the proper programming port. Then open the “Blink” example and change the led pin definition to “LED_BUILTIN” (or pin 17 if you don’t have the variant installed). If you hit upload the code should compile and be transferred to the board, and the pin 17 led should begin to blink.

Now that you are in control of the ESP32 we can move on to exciting things! The remaining portions of this guide will focus on sending a “Hello world!” message from a LoRa device to the internet.

Setting up the Single-Channel LoRaWAN Gateway

Do not press the reset button on your gateway board!

Making a LoRa Gateway

Thanks to 915 MHz LoRa AND WiFi connectivity, the [LoRa Gateway 1-Channel](#) really shines as an inexpensive gateway in a LoRaWAN network. This section will show you how to make your own gateway and access it on the internet.

If you’ve followed along with this hookup guide then you should already be able to program the LoRa Gateway 1-Channel.

The next step is to download a library to run LoRaWAN and modify it to suit our board and needs.

Download the Library

The ESP 1-ch Gateway code is hosted on [GitHub by things4u](#). Although v6 is the current version, the hookup guide by sparkfun still uses v5. Therefore for this lab we are going to use the archived v5 copy hosted:

[ARCHIVED ESP-1CH-GATEWAY-V5.0 \(ZIP\)](#)

This repository includes both the Arduino sketch and the libraries it depends on. Before compiling the sketch you’ll need to extract all libraries from the repository’s “library” folder into your Arduino sketchbook’s “libraries” folder.

To open the example code, open the **ESP-sc-gway.ino** file. When the IDE loads, it should include another dozen-or-so tabs – it’s a hefty, but well-segmented sketch!

Configure the Gateway Sketch

Before uploading the ESP-1ch-Gateway sketch to your board, you’ll need to make a handful of modifications to a couple of files. (Use Ctrl-F to search the file for the setting you want to modify) Here’s a quick overview:

ESP-sc-gway.h

This file is the main source of configuration for the gateway sketch. The definitions you’ll probably have to modify are:

- **Radio**

- ****_LFREQ**** – This sets the frequency range your radio will communicate on. Set this to either **433** (Asia), **868** (EU), or **915** (US)
- ****_SPREADING**** – This sets the LoRa spread factor. **SF7**, **SF8**, **SF9**, **SF10**, **SF11**, or **SF12** can be used. Note that this will affect which devices your gateway can communicate with.
- ****_CAD**** – Channel Activity Detection. If enabled (set to 1) CAD will allow the gateway to monitor messages sent at any spread factor. The tradeoff if enabled: very weak signals may not be picked up by the radio.

- **Hardware**

- ****OLED**** – This board does not include an OLED, **set this to 0**.
- ****_PIN_OUT**** – This configures the SPI and other hardware settings. **Set this to 6**, we'll add a custom hardware definition later.
- ****CFG_sx1276_radio**** – Ensure this is defined and **CFG_sx1272_radio** is *not*. This configures the LoRa radio connected to the ESP32.

- **The Things Network (TTN)**

- ****_TTNSERVER**** – Comment out or delete this.
- ****_TTNPORT**** – Comment out or delete this.
- ****_DESCRIPTION**** – Customize the name of your gateway
- ****_EMAIL**** – Your email address, or that of the owner of the gateway
- ****_LAT**** and ****_LON**** – GPS coordinates of your gateway

- **WiFi**

- Add at least one WiFi network to the **wpas wpa[]** array, but leave the first entry blank. For example:

```
wpas wpa[] = {  
  { "", "" }, // Reserved for WiFi Manager  
  { "NU-IoT", "<password you get after registering>" }  
};
```

There are a lot of other values which can all optionally be configured. For a complete rundown, check out the [Editing the ESP-sc-gway.h](#) part of the README.

loramodem.h

This file defines how the LoRa modem is configured, including which frequency channels it can use and which pins the ESP32 uses to communicate with it. Be careful modifying most of the definitions in here, but one section you will have to modify is the **_PIN_OUT** declarations.

First, find the line that says `#error "Pin Definitions _PIN_OUT must be 1(HALLARD) or 2 (COMRESULT)"` and **delete it**. Then copy and paste these lines in its place (between the `#else` and `#endif`):

```
struct pins {  
    uint8_t dio0 = 26;  
    uint8_t dio1 = 33;  
    uint8_t dio2 = 32;  
    uint8_t ss = 16;  
    uint8_t rst = 27; // Reset not used  
} pins;  
  
#define SCK  14  
#define MISO 12  
#define MOSI 13  
#define SS   16  
#define DI00 26
```

The `int freqs[]` array can be adjusted, if you want to use different subbands, but, beyond that, there's not much else in here we recommend modifying.

Gateway to the Cloud Code

Now we are going to update codes for gateway to cloud communication. By default things4u implementation works for the things network. However, the things network has stopped supports for all single channel gateway devices. Therefore we will use Microsoft Azure IoT hub for the cloud service instead of the things network. For that we need to make a few changes in the source code.

ESP-sc-gway.ino

At the bebegining of ESP-sc-gway.ino include the following header files:

```
#include "Esp32MQTTClient.h"  
#include <HTTPClient.h>  
#include <Arduino.h>
```

Arduino IDE needs AzureIoT and HTTPClnet libraries to include them correctly. Then declare a global variable as:


```
static const char* connectionString = "HostName=PrashantSubediIoTHub.azure-devic  
s.net;DeviceId=iot-hub-1;SharedAccessKey=removed";
```

We will update this variable after enabling our Azure IoT hub device in Azure portal.

Delete or comment out the following function:

```
void pullData()
```

We no longer need this. Now include the following code block at the end of the setup() function:

```
if (!Esp32MQTTClient_Init((const uint8_t*)connectionString)){  
    Serial.println("Initializing IoT hub failed.");  
    return;}  
else{  
    Serial.println("Initializing IoT hub success.");}
```

Inside the loop() delete everything after the function call. That means now it will look like:

```
void loop (){  
    uint32_t uSeconds;                                // micro seconds  
    int packetSize;  
    uint32_t nowSeconds = now();  
    // check for event value, which means that an interrupt has arrived.  
    // In this case we handle the interrupt ( e.g. message received)  
    // in userspace in loop().  
    //  
    stateMachine();                                  // do the state machine  
} //loop
```

_txRx.ino

Include the headers at the beginning:

```
#include "Esp32MQTTClient.h"
#include <HTTPClient.h>
#include <Arduino.h>
```

Now inside the function `int buildPacket(uint32_t tmst, uint8_t *buff_up, struct LoraUp LoraUp, bool internal)` rewrite the conditional preprocessor for `STAT_LOG == 1` as:

```
#if STAT_LOG == 1
    // Do statistics logging. In first version we might only
    // write part of the record to files, later more
    addLog( (unsigned char *)(buff_up), buff_index );
    Serial.println((char *)message);
    for(int idx=0;idx<messageLength;idx++)
    {
        Serial.print(message[idx],HEX);
        Serial.print(" ");
    }
    Serial.println("");
    Serial.println("start sending events.");
    char buff[256];
    //Send the in JSON format
    String res = "{\"Message\": \"";
    res.concat((char *)message);
    res.concat("\"}");
    // Replace the following line with your data sent to Azure IoT Hub
    snprintf(buff, 256, res.c_str());
    Serial.println(res);
    if (Esp32MQTTClient_SendEvent(buff))
    {
        Serial.println("Sending data succeed");
    }
    else
    {
        Serial.println("Failure...");
    }
#endif
```

```
}  
#endif
```

Upload the Code

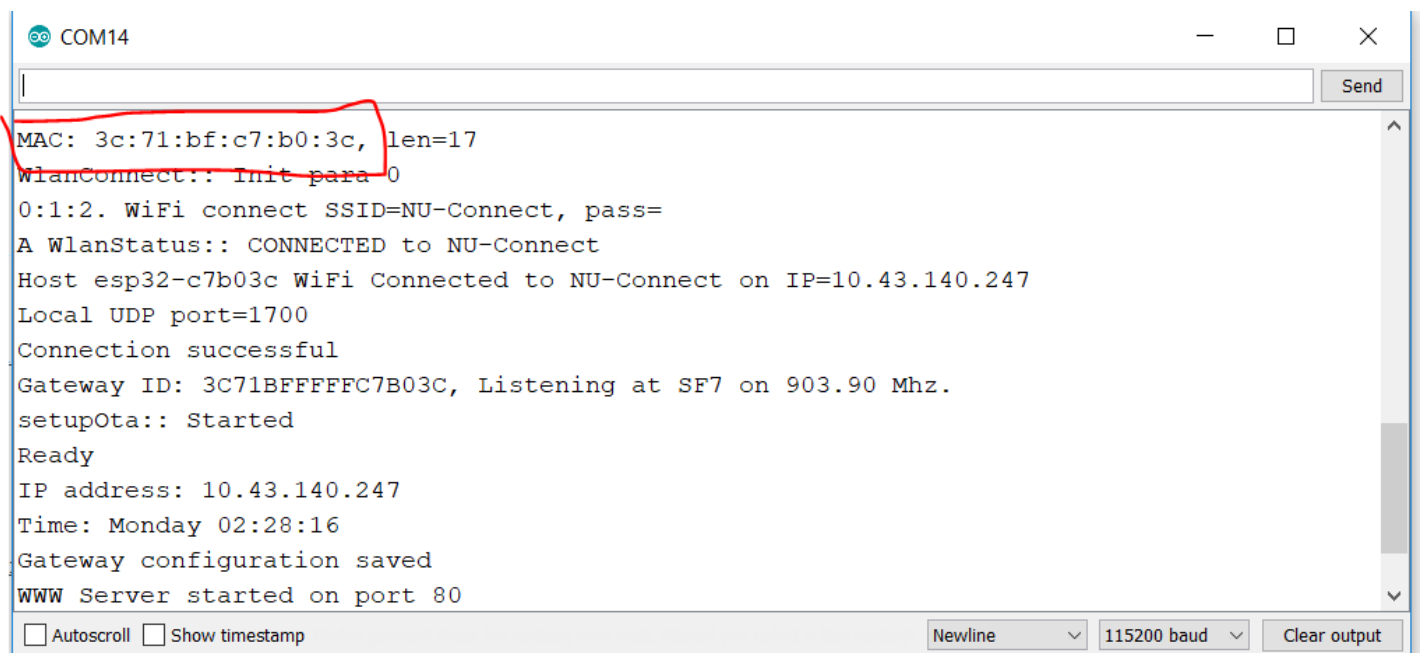
After configuring the gateway project, upload it to the board. Try compiling and uploading the sketch to your ESP32 with program upload BAUD **115200**. Also after it's uploaded, open up your [serial monitor](#) and set the baud rate to **115200**. The sketch may take a long time to set up the first time through – it will format your SPIFFS file system and create a non-volatile configuration file. Once that's complete, you should see the ESP32 attempt to connect to your WiFi network, then initialize the radio.

2. Connecting Gateway to NU-IoT

Without the Wi-Fi network, our device won't have a link to connect to the Internet. UNL Wi-Fi is now open for IoT device enrollment and gives up a big advantage.

Acquire the MAC address

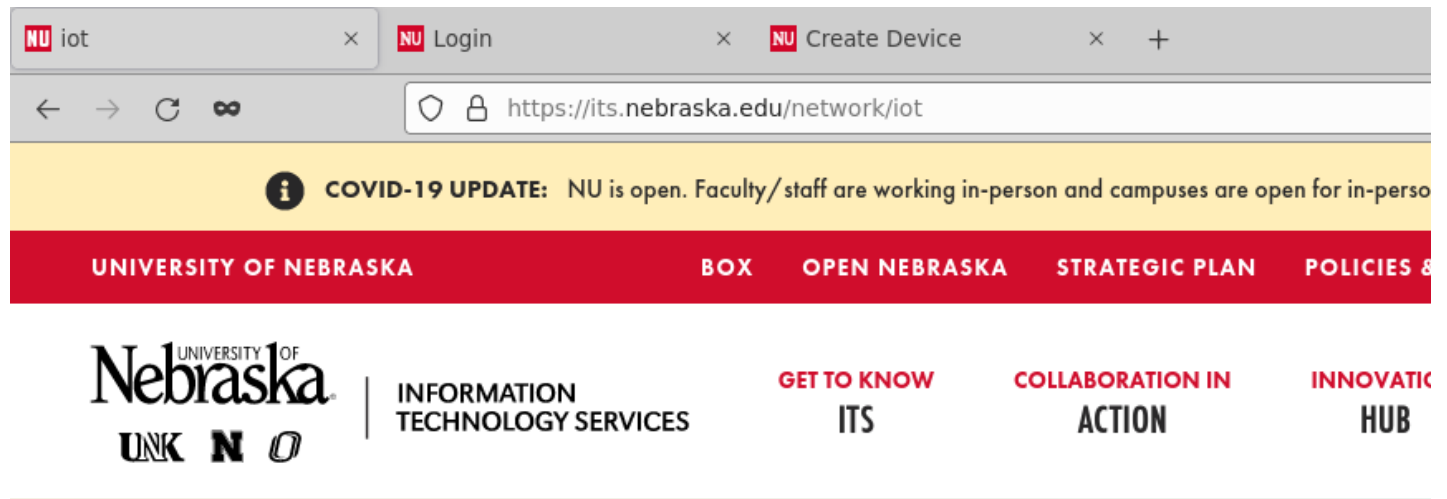
After uploading your gateway program, you would see the below message when the system is starting up. Find a line that includes the MAC address of your device, and save it



```
COM14  
MAC: 3c:71:bf:c7:b0:3c, len=17  
WlanConnect:: Init para 0  
0:1:2. WiFi connect SSID=NU-Connect, pass=  
A WlanStatus:: CONNECTED to NU-Connect  
Host esp32-c7b03c WiFi Connected to NU-Connect on IP=10.43.140.247  
Local UDP port=1700  
Connection successful  
Gateway ID: 3C71BFFFFFFC7B03C, Listening at SF7 on 903.90 Mhz.  
setupOta:: Started  
Ready  
IP address: 10.43.140.247  
Time: Monday 02:28:16  
Gateway configuration saved  
WWW Server started on port 80  
☐ Autoscroll ☐ Show timestamp Newline 115200 baud Clear output
```

Connect it to NU-IoT

Connect your computer wifi to NU-IoT instead of eduroam to register your LoRa gateway. Each group will have one gateway so it should only be done once. You have to register your device from campus network. After visiting to <https://its.nebraska.edu/network/iot> you will see this page. Select “IoT Registration Portal” and continue.



Register IoT Device

IoT devices such as gaming consoles, printers, and wireless streaming hardware can receive internet access by completing the IoT Registration process.


Procedure

1. While on a campus network, open the **IoT Registration portal** in your default browser.
2. On the IoT Registration portal, use your Campus Identity to authenticate. Your username must include the campus domain. Example: jdoe2@nebraska.edu (@unk.edu, @unl.edu, @unomaha.edu).
3. In the IoT Registration portal you will be able to add or manage existing IoT devices. Each device will receive a unique password to authenticate on the NU-IoT wireless SSID

Don't know your Campus Identity?

Use your smartphone or another computer with internet access and visit <https://trueyou.nebraska.edu> to manage your identity.

After logging in with your UNL account, you will see the following page. Put the MAC address on the bar and enroll this gateway.

Register Device	
* MAC Address:	<input type="text"/> MAC address of the device.
Device Name:	<input type="text" value="IoT-class-gw"/> Name of the device.
* Email:	<input type="text" value="psubedi3@huskers.unl.edu"/> Email to send detailed receipt.
Where would you like your credentials sent?	<input checked="" type="radio"/> Email <input type="radio"/> Text Message <input type="radio"/> Both
AirGroup:	<input type="checkbox"/> Enable AirGroup AirGroup uses device ownership and location information to limit the printers and Apple TVs available to network users.
Account Expiration:	1 year from now
* Account Role:	Clients IoT Device Role
* Registered By:	47181417
 CREATE DEVICE	

Test it!

Modify the password field in the ESP-sc-gway.h. You can test it by looking at the messages print out by the serial monitor. If it says connection is successful then we can continue to the next step.

3. Setting up The Azure IoT Hub

Enable Student Credit


Go to [Azure for students](#) and click start free. Login with your UNL email and password. Then activate your student benefit credit in that account.

IoT Hub

Click to create a resource in azure portal. In the search box search ‘IoT Hub’ and select it. Then click ‘create’.


[Home](#) > [Create a resource](#) > [Marketplace](#) >

IoT Hub




...

Microsoft



IoT Hub

 [Add to Favorites](#)

Microsoft

★ 4.2 (1050 Marketplace ratings) | ★ 4.2 (700 external ratings)

Plan

IoT Hub

▼

Create

Overview

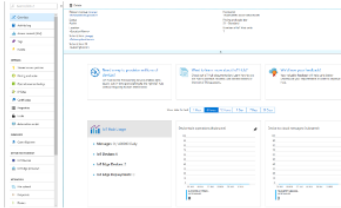
Plans

Usage Information + Support

Reviews

Simultaneously support millions of connected devices—whether they run Windows, Linux, or real-time operating systems. Then monitor performance and send commands to accelerate your digital transformation.

Media



[More products from Microsoft](#) [See All](#)

Select ‘Azure for Students’ as the subscription. Add it in a resource group. If you do not have a resource group create a new. Put a name for your IoT hub and select region as ‘Central US’.

[Home](#) > [Create a resource](#) > [IoT Hub](#) >

IoT hub

Microsoft

Basics Networking Management Tags Review + create

Create an IoT hub to help you connect, monitor, and manage billions of your IoT assets. [Learn more](#)

Project details

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription *	<input type="text" value="Azure for Students"/>
Resource group *	<input type="text" value="(New) csce838"/>

[Create new](#)

Instance details

IoT hub name *	<input type="text" value="PrashantSubediIoTHub"/>
Region *	<input type="text" value="Central US"/>

Review + create

< Previous

Next: Networking >

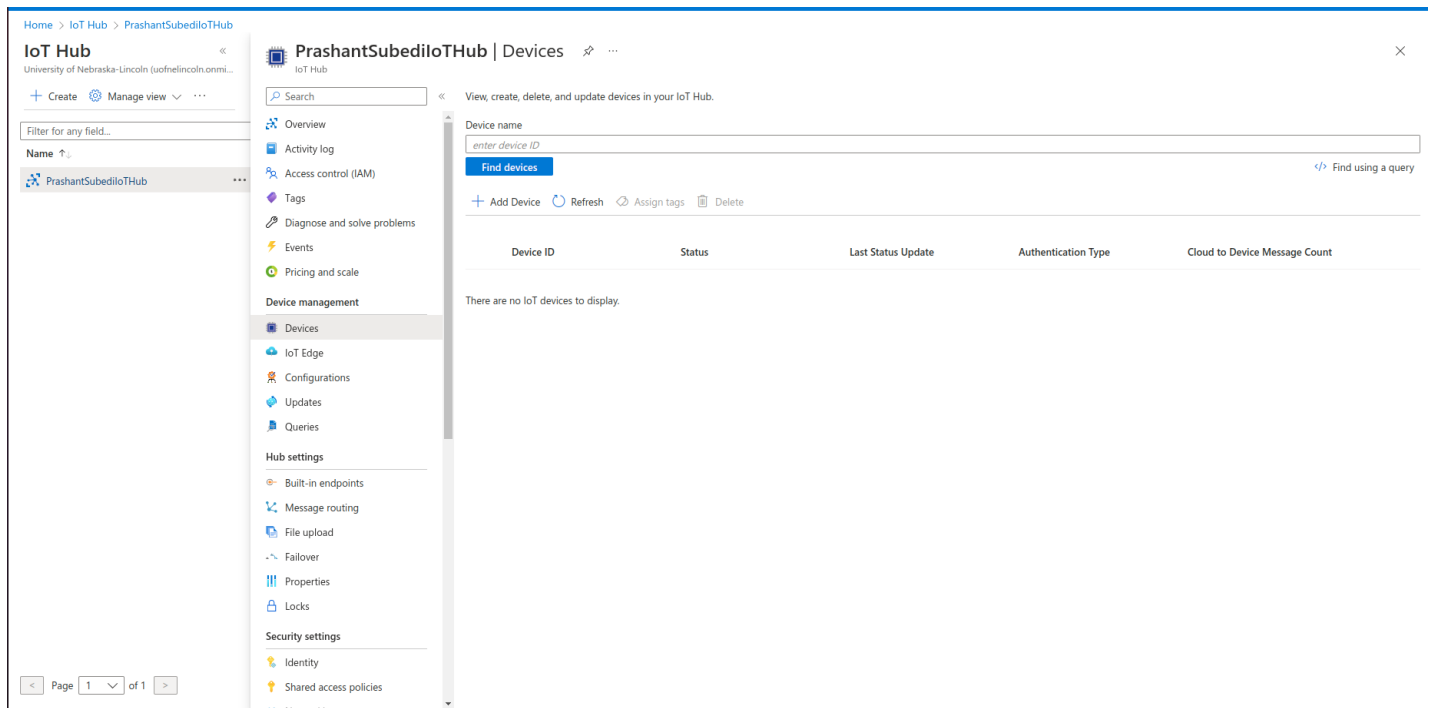
Now click on the 'Management' tab. Select 'F1: Free tier' as your pricing and scale tier. Keep others as the default settings.

Now Click 'Review+create' button from the bottom. After a while you are getting your new azure IoT hub.

For this free tier hub, you can exchange a total of 8000 messages per day. Therefore, program your device accordingly so that it does not flood your cloud with messages and finish the quota.

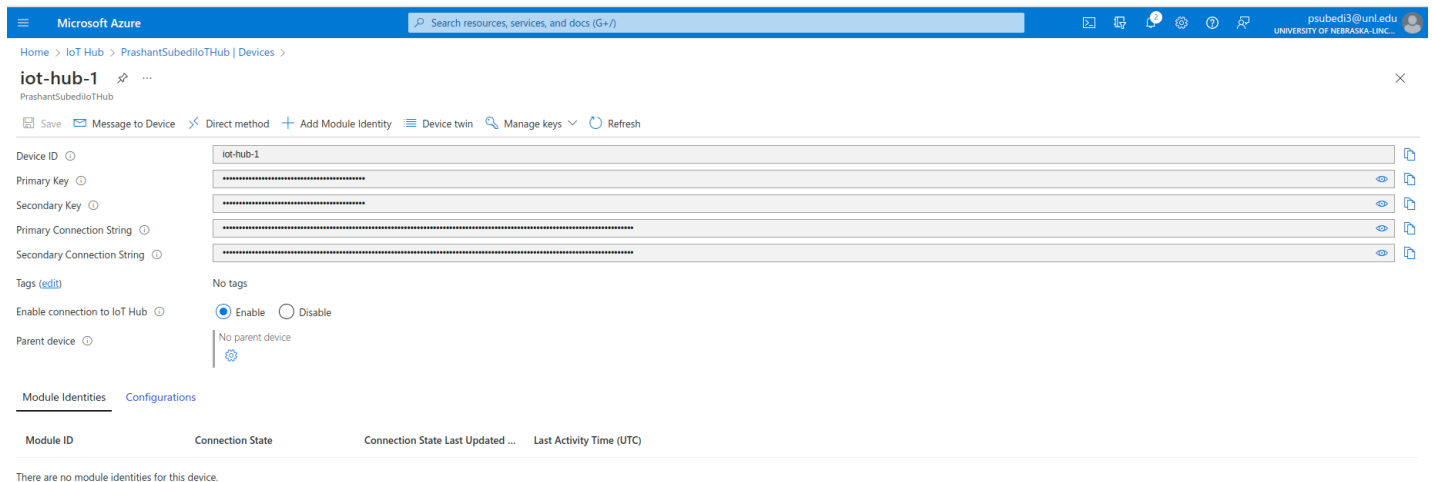
IoT Device for the Hub

Once you have the hub, now add an IoT device in it. Click to the newly created hub. Then click on 'Devices' from the left side column.



Now click on Add Device button. Put a 'Device ID' on it. Select 'Authentication type' as symmetric key and check 'Auto-generate keys' key option. Make 'Connect this device to an IoT hub' enable. Click 'Save' on the bottom. It will create an IoT device for your IoT hub.

Now click on that device. It will show all information of that device. Copy the 'Primary Connection String' of it. This connection string needs to be added into static const char* connectionString variable inside ESP32 gateway's code.



We'll need these keys to program your ESP32 so leave this page up on your browser. After finishing the gateway code, let's open up the Arduino IDE again. It's time to program the node!

4. Setting up your device/application

LMIC

- Why not Radiohead?
 - Radiohead is an amateur library for accessing the radio chip hardware. It is not designed for standardized work.
- What is LMIC (LoraMAC-in-C)
 - LoRa MAC implementation in C
 - An real-time OS supports it
 - LoRa communications are supported
 - Highly professional and integrated library

Installing LMIC library

We'll be setting up the SAMD21 Pro RF as a *node* using a library written by **Matthijs Kooijman** which is a modified version of "IBM's LMIC (LoraMAC-in-C)" library. The latest repo is maintained by a company named MCCI. You can download and manually install it from the [GitHub Repository](#).

Configurations

The LoRa settings should match

The spreading factor, frequency, bandwidth should match your gateway configuration.

Configure LMIC

1. This modified example takes directly from the example code provided by the library with a two changes: the function calls to " **Serial** " will need to be replaced with " **SerialUSB** " and changes to the pin mapping that is consistent with the SAMD21 Pro RF. Before we look at the code you'll first need to modify the *lmic_project_config.h* file that came with the LMIC Arduino Library.
2. Find your Arduino *libraries* folder and navigate to **...libraries/arduino-LMIC/project_config/**. You should find a file called *lmic_project_config.h*. Open it in any text editor and find the lines where **CFG_us915** is defined. It should look like this:

```
///#define CFG_eu868 1  
  
#define CFG_us915 1
```

```
// This is the SX1272/SX1273 radio, which is also used on the HopeRF
// RFM92 boards.

// #define CFG_sx1272_radio 1

// This is the SX1276/SX1277/SX1278/SX1279 radio, which is also used on
// the HopeRF RFM95 boards.

#define CFG_sx1276_radio 1
```

Configuring Serial for SAMD21

Remember to change your serial library lines of your SAMD21 Pro RF to `SerialUSB`

Example Code for Sparkfun pro RF LoRaWAN implementation

```

/*****
 * Copyright (c) 2015 Matthijs Kooijman
 * Copyright (c) 2018 Terry Moore, MCCI Corporation
 *
 * Permission is hereby granted, free of charge, to anyone
 * obtaining a copy of this document and accompanying files,
 * to do whatever they want with them without any restriction,
 * including, but not limited to, copying, modification and redistribution.
 * NO WARRANTY OF ANY KIND IS PROVIDED.
 *
 * This example transmits data on hardcoded channel and receives data
 * when not transmitting. Running this sketch on two nodes should allow
 * them to communicate.
 *****/

#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

```

```

// we formerly would check this configuration; but now there is a flag,
// in the LMIC, LMIC.noRXIQinversion;
// if we set that during init, we get the same effect. If
// DISABLE_INVERT_IQ_ON_RX is defined, it means that LMIC.noRXIQinversion is
// treated as always set.
//
// #if !defined(DISABLE_INVERT_IQ_ON_RX)
// #error This example requires DISABLE_INVERT_IQ_ON_RX to be set. Update \
//      lmic_project_config.h in arduino-lmic/project_config to set it.
// #endif

// How often to send a packet. Note that this sketch bypasses the normal
// LMIC duty cycle limiting, so when you change anything in this sketch
// (payload length, frequency, spreading factor), be sure to check if
// this interval should not also be increased.
// See this spreadsheet for an easy airtime and duty cycle calculator:
// https://docs.google.com/spreadsheets/d/1voGAtQAjClqBmaVuP1ApNKs1ekgUjavHuVQIXyYSvNc
#define TX_INTERVAL 60000 //Delay between each message in millidecond.

// Pin mapping for SAMD21
const lmic_pinmap lmic_pins = {
    .nss = 12, //RFM Chip Select
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 7, //RFM Reset
    .dio = {6, 10, 11}, //RFM Interrupt, RFM LoRa pin, RFM LoRa pin
};

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmloc/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }

```

```

void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

void onEvent (ev_t ev) {
}

osjob_t txjob;
osjob_t timeoutjob;
static void tx_func (osjob_t* job);

// Transmit the given string and call the given function afterwards
void tx(const char *str, osjobcb_t func) {
    os_radio(RADIO_RST); // Stop RX first
    delay(1); // Wait a bit, without this os_radio below asserts, apparently because the state hasn't changed yet
    LMIC.dataLen = 0;
    while (*str)
        LMIC.frame[LMIC.dataLen++] = *str++;
    LMIC.osjob.func = func;
    os_radio(RADIO_TX);
    SerialUSB.println("TX");
}

// Enable rx mode and call func when a packet is received
void rx(osjobcb_t func) {
    LMIC.osjob.func = func;
    LMIC.rxtime = os_getTime(); // RX _now_
    // Enable "continuous" RX (e.g. without a timeout, still stops after receiving a packet)
    os_radio(RADIO_RXON);
    SerialUSB.println("RX");
}

static void rxtimeout_func(osjob_t *job) {
    digitalWrite(LED_BUILTIN, LOW); // off

```

```

}

static void rx_func (osjob_t* job) {
    // Blink once to confirm reception and then keep the led on
    digitalWrite(LED_BUILTIN, LOW); // off
    delay(10);
    digitalWrite(LED_BUILTIN, HIGH); // on

    // Timeout RX (i.e. update led status) after 3 periods without RX
    os_setTimedCallback(&timeoutjob, os_getTime() + ms2osticks(3*TX_INTERVAL), rxtimeout_func);

    // Reschedule TX so that it should not collide with the other side's
    // next TX
    os_setTimedCallback(&txjob, os_getTime() + ms2osticks(TX_INTERVAL/2), tx_func);

    SerialUSB.print("Got ");
    SerialUSB.print(LMIC.dataLen);
    SerialUSB.println(" bytes");
    SerialUSB.write(LMIC.frame, LMIC.dataLen);
    SerialUSB.println();

    // Restart RX
    rx(rx_func);
}

static void txdone_func (osjob_t* job) {
    //rx(rx_func);
}

// log text to USART and toggle LED
static void tx_func (osjob_t* job) {
    // say hello
    tx("Hello, world!", txdone_func);
    // reschedule job every TX_INTERVAL (plus a bit of random to prevent

```

```

    // systematic collisions), unless packets are received, then rx_func
    // will reschedule at half this time.
    os_setTimedCallback(job, os_getTime() + ms2osticks(TX_INTERVAL + random(500)),
tx_func);
}

// application entry point
void setup() {
    SerialUSB.begin(115200);
    while(!SerialUSB);
    SerialUSB.println("Starting");
    // #ifdef VCC_ENABLE
    // // For Pinoccio Scout boards
    // pinMode(VCC_ENABLE, OUTPUT);
    // digitalWrite(VCC_ENABLE, HIGH);
    // delay(1000);
    // #endif

    pinMode(LED_BUILTIN, OUTPUT);

    // initialize runtime env
    os_init();

    // this is automatically set to the proper bandwidth in kHz,
    // based on the selected channel.
    uint32_t uBandwidth;
    LMIC.freq = 903900000;
    uBandwidth = 125;
    LMIC.datarate = US915_DR_SF7;           // DR4
    LMIC.txpow = 21;

    // disable RX IQ inversion

```

```

LMIC.noRXIQinversion = true;

// This sets CR 4/5, BW125 (except for EU/AS923 DR_SF7B, which uses BW250)
LMIC.rps = updr2rps(LMIC.datarate);

SerialUSB.print("Frequency: "); SerialUSB.print(LMIC.freq / 1000000);
    SerialUSB.print("."); SerialUSB.print((LMIC.freq / 100000) % 10);
    SerialUSB.print("MHz");
SerialUSB.print("  LMIC.datarate: "); SerialUSB.print(LMIC.datarate);
SerialUSB.print("  LMIC.txpow: "); SerialUSB.println(LMIC.txpow);

// This sets CR 4/5, BW125 (except for DR_SF7B, which uses BW250)
LMIC.rps = updr2rps(LMIC.datarate);

// disable RX IQ inversion
LMIC.noRXIQinversion = true;

SerialUSB.println("Started");
SerialUSB.flush();

// setup initial job
os_setCallback(&txjob, tx_func);
}

void loop() {
    // execute scheduled jobs and events
    os_runloop_once();
}

```

5. Receiving Data on Azure from your device


Stream Your Data

Now we have both devices ready and cable to transmit and receive LoRa packets. In addition, in Azure we have an IoT device inside the azure IoT hub. To access data from this device we need a “Stream Analytics job” or any other equivalent data access resource. In this lab we will utilize Stream Analytics job.

Similar to all other resources creation we now need to click the create button and search for Stream Analytics job. After clicking on create button we have fill the required information and then click on the ‘Review+create’ button at the bottom. Then click ‘Create’ button

[Home](#) > [Stream Analytics jobs](#) >

New Stream Analytics job

 Changes on this step may reset later selections you have made. Review all options prior to deployment.

Basics Storage Tags Review + create

Azure Stream Analytics is a fully managed, SQL-based stream processing engine designed to help you tackle scenarios like streaming ETL to Azure Data Lake Storage, real-time dashboarding with Power BI, event driven applications with Azure SQL DB & Cosmos DB, remote monitoring, predictive maintenance, and more. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure for Students ▼

Resource group * ⓘ csce838 ▼
[Create new](#)

Instance details

Name * csce_838 ✓

Region * ⓘ Central US ▼

Hosting environment ⓘ
☒ Cloud
☐ Edge

Streaming unit details

Streaming units (SUs) represents the computing resources that are allocated to execute a Stream Analytics job. The higher the number of SUs, the more CPU and memory resources are allocated for your job. The number of SUs can be modified once you create the job. You will be charged for the job's Streaming Units only when the job runs. [Learn more](#)

Streaming units * 1 ▼

[Review + create](#) [< Previous](#) [Next : Storage >](#)

Once the job is created click on ‘Go to resource’ button.

Microsoft Azure

Search resources, services, and docs (G+/)

psubedi3@unl.edu
UNIVERSITY OF NEBRASKA-LINCOLN

Home > CreateForm-20220913224423 | Overview >

csce_838

Stream Analytics job

Search

Start ☐ Stop Delete Move Refresh Share feedback

Created

JSON View

Essentials

Resource group (move) : csce838
Location : Central US
Status : Created
Subscription (move) : Azure for Students
Subscription ID : 8d24456d-42a8-4022-8a7e-b761cc98a64f

Created : Tuesday, September 13, 2022 10:47 PM
Started :
Output watermark :
Cluster : Shared
Hosting environment : Cloud

Tags (edit) : [Click here to add tags](#)

Get started Properties Monitoring Tutorials

Build an end-to-end serverless streaming pipeline with just a few clicks

Azure Stream Analytics is a fully managed, real-time stream processing service designed to help you tackle scenarios such as streaming ETL to ADLS Gen2 or Synapse SQL, real time apps with Cosmos DB or SQL DB, live dashboarding with Power BI, or real-time alerting with Azure Functions. [Learn more](#)

Ingest data
Stream Analytics jobs connect to one or more data inputs. Each input defines a connection to an existing data source.
[Add input](#)

Analyze data
Stream Analytics jobs uses Stream Analytics Query Language (SAQL) to transform or analyze your real time data.
[Write query](#)

Output data
Stream Analytics jobs connects to one or more data outputs. There are several output types to which you can send transformed data.
[Add output](#)

Enable logging
Turning on diagnostic settings to Log Analytics will allow you to easily troubleshoot any errors your job may encounter.
[Configure](#)

Here we can see that we need to select inputs and outputs for this job. First click on 'Inputs'. Then click on 'Add stream input' and select 'IoT Hub'. Give an alias name and select your IoT Hub from the drop-down menu. Click 'Save'. Now our IoT hub is connected as an input stream.

IoT Hub



New input

Input alias *

csce-838-stream



- ☐ Provide IoT Hub settings manually
- ☒ Select IoT Hub from your subscriptions

Subscription

Azure for Students



IoT Hub * ⓘ

PrashantSubediloTHub



Consumer group * ⓘ

\$Default



Shared access policy name * ⓘ

iothubowner



Shared access policy key ⓘ

.....

Endpoint ⓘ

Messaging



Partition key ⓘ

Event serialization format * ⓘ

JSON



Encoding ⓘ

UTF-8



Event compression type ⓘ

None



Save

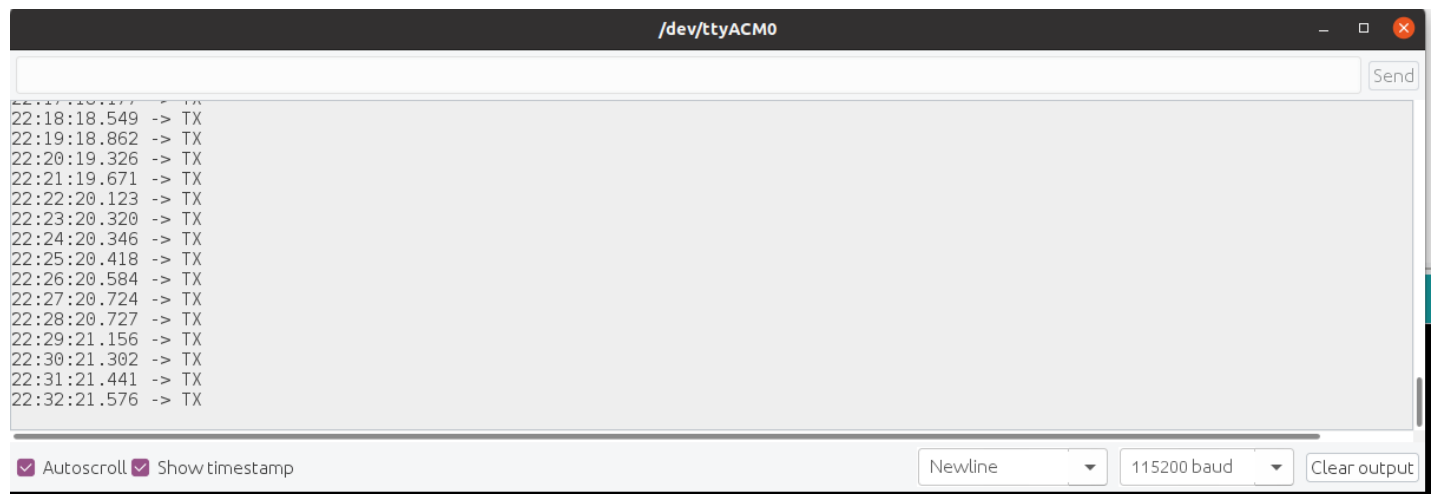
For this lab, we won't need any output source, rather we will query the input stream directly to see the data being sent to IoT Hub. For this, go to Query tab in the stream analytics job and add the following query

```
SELECT
    *
FROM
    [csce-838-stream]
```

Replace csce-838-stream with your input alias. You should see the messages sent to IoT hub in the result section as shown in the screenshot below.

Expected Results

Serial monitor from the Sparkfun pro RF. From there you should see the messages showing transmission or run-time errors defined in the program.



If the gateway successfully received the packet, its serial monitor will show the messages:

```
22:24:06.019 -> MAC: cc:50:e3:8d:cd:48, len=17
22:24:06.019 -> WlanConnect:: Init para 0
22:24:06.119 -> 0:1:3: WiFi connect SSID=QuickAccess, pass=cxph3iciq@f8rv7
22:24:15.125 -> A WlanStatus:: CONNECTED to QuickAccess
22:24:15.257 -> Host esp32-8dc48 WiFi Connected to QuickAccess on IP=192.168.1.113
22:24:15.457 -> Local UDP port=1700
22:24:15.457 -> Connection successful
22:24:16.155 -> Gateway ID: CC50E3FFFF8DCD48, Listening at SF9 on 903.90 Mhz.
22:24:16.421 -> setupOTA:: Started
22:24:16.421 -> Ready
22:24:16.421 -> IP address: 192.168.1.113
22:24:17.020 -> Time: Friday 05:24:16
22:24:17.020 -> Gateway configuration saved
22:24:17.020 -> WWW Server started on port 80
22:24:17.152 -> -----
22:24:17.152 -> Info: Initializing SNMP
22:24:18.150 -> Info: SNMP initialization complete
22:24:18.183 -> Info: IoT Hub SDK for C, version 1.1.23
22:24:20.911 -> Info: >>>Connection status: connected
22:24:20.911 -> Initializing IoT hub success.
22:25:20.551 -> G addLog:: fileno=0, rec=1: 1 50 31 0 CC 50 E3 FF 8D CD 48 {"rxpk":[{"tmst":75821255,"chan":0,"rfch":0,"freq":903.900024,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":9,"
22:25:20.584 -> Hello, world!
22:25:20.584 -> 48 65 6C 6F 2C 20 77 6F 72 6C 64 21
22:25:20.584 -> start sending events.
22:25:20.584 -> {"Message": "Hello, world!"}
22:25:20.584 -> Info: >>>IoTHubClient_LL_SendEventAsync accepted message for transmission to IoT Hub.
22:25:21.182 -> Info: >>>Confirmation[0] received for message tracking_id = 0 with result = IOTHUB_CLIENT_CONFIRMATION_OK
22:25:21.182 -> Sending data succeed
```

The data panel from Azure stream analytics query tab will look like this if all the links are successfully established. This means you now have your end to end IoT application.

Microsoft Azure

Search resources, services, and docs (G+)

psuhed3@unl.edu
UNIVERSITY OF NEBRASKA-LINCOLN

Home > CreateForm-20220913224423 | Overview > csce_838

csce_838 | Query

Stream Analytics job

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Properties

Locks

Job topology

Inputs

Functions

Query

Outputs

Configure

Environment

Storage account settings

Scale

Locale

Event ordering

Error policy

Compatibility level

Managed Identity

Developer tools

Query language docs

Open in VS Code

Share feedback

Refresh

Test query

Save query

Discard changes

1 SELECT

2 *

3 FROM

4 [csce-838-stream]

Input preview

Test results

Showing sample events from 'csce-838-stream'.

Table

Raw

Refresh

Select time range

Upload sample input

Download sample data

Message string	EventProcessedUtcTime datetime	PartitionId bigint	EventEnqueuedUtcTime datetime	IoTHub string
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:50.7080000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:49.2230000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:47.8630000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:46.5820000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:45.7840000Z"	("MessageId":null,"CorrelationId":null,"C...

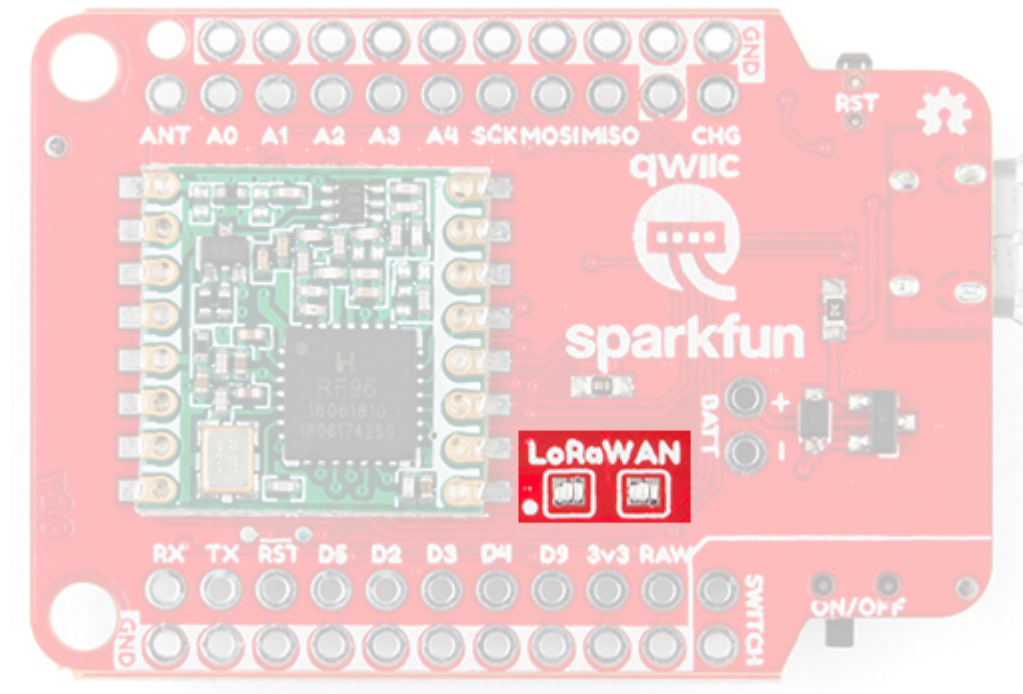
Lab Assignment

In this lab, you will work with your teammates and get familiar with our IoT system.

A lab report is required from everyone in the group. You will need to work together as you need to share the gateway. In the group report, each member needs to provide individual screenshots of Azure account and serial monitor outputs.

In the class: Close the LoRaWAN Jumpers

One last thing. On the underside of the SAMD21 Pro RF there are two jumpers labeled **LoRaWAN**. Closing these jumpers will tell the module that we're broadcasting in the modulation scheme unique to LoRaWAN.



Assignment

Requirements

1. Finish the helloworld example
 1. Record the procedure of setting up the link from your device to IoT Hub with screenshots
2. Keep the code from the previous lab (lab 3), and merge it with the LMIC example code for packet transmissions
 1. Temporarily remove the radio operations in the code and use the LMIC code
 2. Use your packet construction modules and average temperature reading module, etc.
3. Maintain your packet structure from lab 3 and make necessary changes in the gateway to prepare a json data packet. Then send packets with temperature sensor data to the Azure cloud every **60** seconds. Instead of using timers, use LMIC's TX_INTERVAL.
4. Download the json file from Azure and share the contents of it in the report.

Results

5. Code that fulfills each requirement in this lab

1. Each function in this system should be separately presented with explanation, entire code snippet will not be accept

6. Serial message from

1. Sparkfun pro RF device
2. LoRa gateway

7. Screenshots from Azure and json for the data reporting results.

Report format

- Report:
 - Development Process
 - Record your development process
 - **Acknowledge any resources that you found and helped you with your development (open-source projects/forum threads/books)**
 - Record the software/hardware bugs/pitfalls you had and your troubleshooting procedure.
 - Results
 - Required results from the section above
 - The entire program (Arduino sketch) in the appendix (No screenshots will be accepted)

Submission Instructions:

1. Submit your lab on Canvas on or before the deadline (Sep 23rd, 8:29 am)
2. Your submission should include one single pdf explaining everything that was asked in the tasks and screenshots if any
3. Your submission should also include all the code that you have worked on with proper documentation
4. Failing to follow the instructions will make you lose points

Reference

1. <https://learn.sparkfun.com/tutorials/sparkfun-lora-gateway-1-channel-hookup-guide/all>
2. <https://learn.sparkfun.com/tutorials/lorawan-with-prorf-and-the-things-network/all#example-ifttt-integration>

3. https://www.youtube.com/playlist?list=PLlrXD0HtieHh5_pOv-6xsMxS3URD6XD52
4. <https://learn.sparkfun.com/tutorials/sparkfun-samd21-pro-rf-hookup-guide/lorawan-arduino-library-and-example>
5. Lea, Perry. *Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security*. Packt Publishing Ltd, 2018.