

# Fall 2023 - CSCE 438/838: IoT - Lab 4 - Connecting Things to the Internet

## Introduction

Last week, we implemented an 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 helped you to understand the basics of embedded systems 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 fire up a device to send the data, and finally 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 aggregate 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 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 reasonable 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 significantly improved the IoT value proposition.

## LoRaWAN Overview

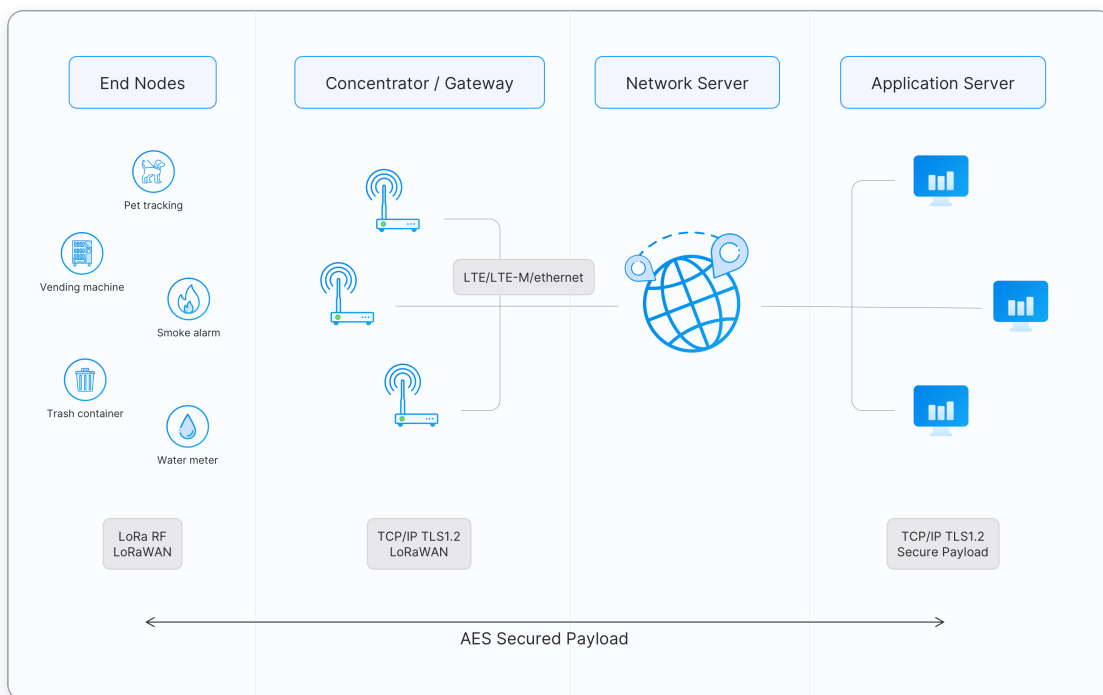
LoRaWAN is a media access control (MAC) protocol for wide-area networks. It allows low-powered devices to communicate with Internet-connected applications over long-range wireless connections. LoRaWAN can be mapped to the second and third layers of the OSI model. It is implemented on top of LoRa or FSK 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 interconnectivity to many things we interact with daily. For example, if your refrigerator kept track of what was inside and could talk

to your cell phone, you wouldn't be left wondering if you needed to buy milk at the store. 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 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 dedicated to bridge from LoRa messages to internet traffic - called a "**gateway**." If you have something that can speak both LoRa and "Internet," then you could make your solution, but there is a more accessible 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. A critical benefit of LoRaWAN is that you can send encrypted data during transmission, and your fridge could get up and walk to the next state over (again - just a metaphor!). The messages could still be picked up by a gateway 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 End-Device - Sparkfun Pro RF** A "end-device" is the remote system sending (and sometimes receiving) data. We will set up a device to send the "Hello World!" message in the section "Turning a Gateway into A Device."

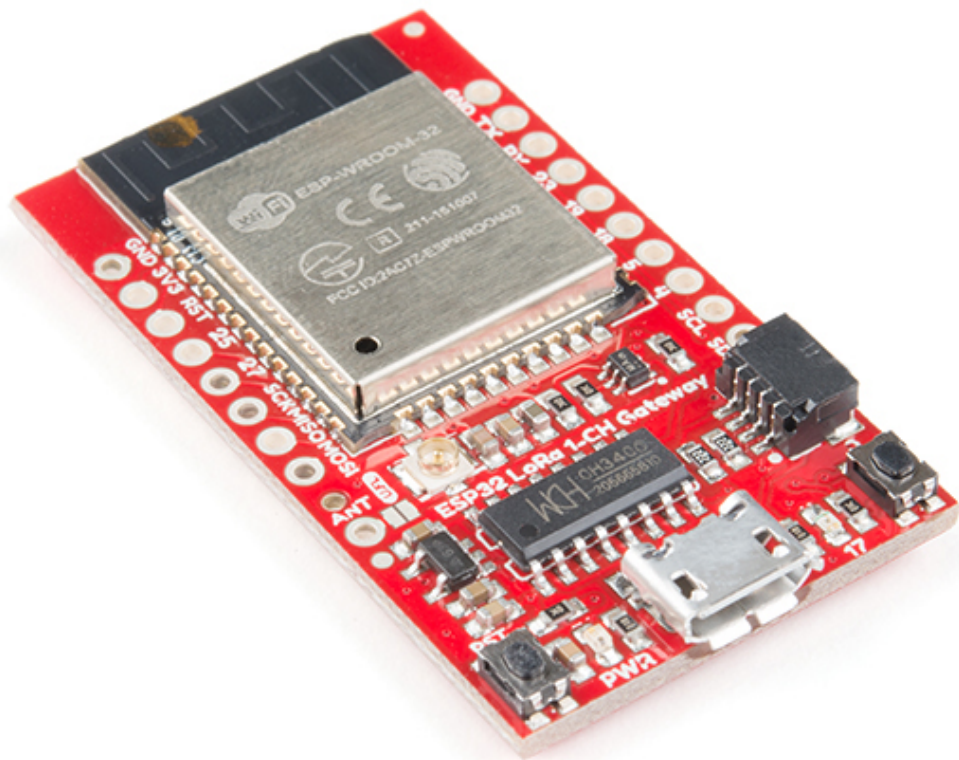
**Your Gateway - Sparkfun ESP32 Things 1-channel Gateway** The name of the ESP LoRa Gateway 1-Channel acts as the bridge that speaks both WiFi and LoRa. The "Single-Channel LoRaWAN Gateway" section will cover all the steps needed to make this happen.

With these, we can start 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
-

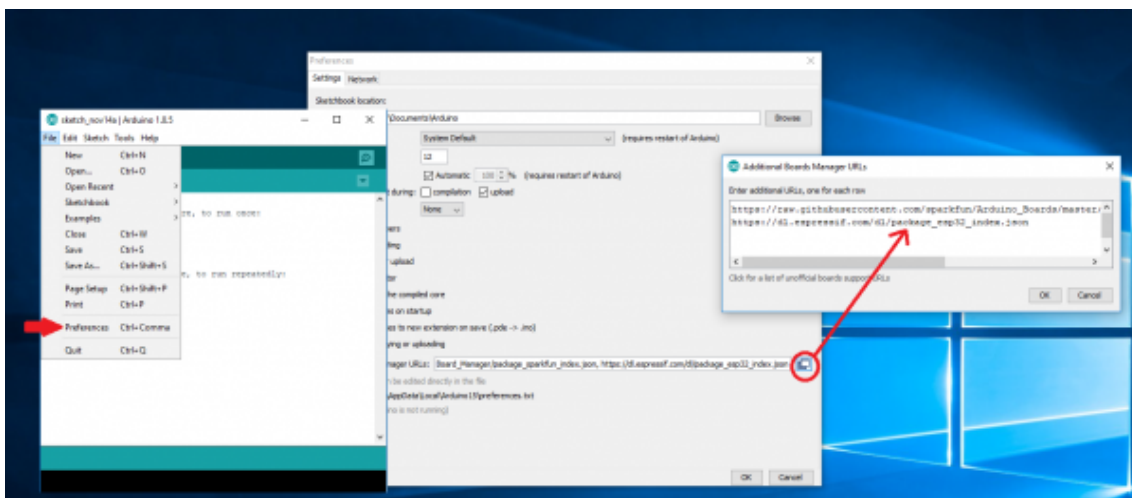
# 1. Setting Up 1-channel Gateway

## Installing ESP32 Arduino Core

The ESP32's relationship with Arduino is growing, and now it is straightforward to install the core - the Arduino IDE can handle it nearly independently. 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 beside 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. Ensure the correct board is selected (SparkFun ESP32 Things if you installed the variant as 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 be compiled and transferred to the board, and the pin 17 LED should begin blinking.

Now that you control 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.

**MacOS python3 problem !**

Since Apple upgraded MacOS Monterey to 12.3, Arduino is not able to find python - apparently there is no more python 2.7 in MacOS. Our SparkFun package uses an outdated version of esp32 libraries that **are** looking for python 2.7. The same problem may occur

on Linux as well. This is what you could get when you try to compile the simple Blink code:

```
exec: "python": executable file not found in $PATH
Compilation error: exec: "python": executable file not found in $PATH
```

Here is a solution.

### 1. Check python

First, check that you have python. Open a Terminal and type `python -V`. Chances are on a new MacOS you will get `zsh: command not found: python`. Then, type `python3 -V` and you'll get `Python 3.9.6`. For future reference, also find its path by `where python3` to which you'll get something like `/usr/bin/python3`. This last path is important, save it.

### 2. Locate the problem path and update with the python3 path

To locate the problem, first go to Arduino IDE Preferences, find the line **Show verbose output during** and check both **compile** and **output** boxes. This will provide you more info during compiling. Then, compile your code again and check the last line before the errors.

```
python
[HOME]/Library/Arduino15/packages/SparkFun/hardware/esp32/1.0.0/tools/gen_esp32part.py
-q
[HOME]/Library/Arduino15/packages/SparkFun/hardware/esp32/1.0.0/tools/partitions/default
...
```

Now you know the path to your SparkFun folder. Then, in the terminal go to the following folder

```
$HOME/Library/Arduino15/packages/SparkFun/hardware/esp32/1.0.0/
```

and edit the **platform.txt** file with an editor such as `vi`. Change the following line

```
tools.gen_esp32part.cmd=python "{runtime.platform.path}/tools/gen_esp32part.py"
```

to the python3 path you found above. For this example, to

```
tools.gen_esp32part.cmd=/usr/bin/python3 "
{runtime.platform.path}/tools/gen_esp32part.py"
```

Make sure you add 3 at the end of python, i.e., python3. This solves the first compilation error, but then you face with another error that is related to the `esptool` binary file that is hardcoded to look for python 2.7.

### 3. Update the esptool binary file

If you look at the second problem's error log, you'll see that it is related to a binary called `esptool` that is located somewhere like

```
$HOME/Library/Arduino15/packages/esp32/tools/esptool_py/2.6.1
```

The SparkFun library we downloaded uses an outdated esp library and we need to update it. Here is a (pretty dirty) hack to do that real quick.

Go to Arduino IDE Preferences and add the following url to **Additional Board Manager URLs**

```
https://espressif.github.io/arduino-esp32/package_esp32_index.json
```

then, go to Tools->Board->Board Manager and search for **esp**. Install **esp32 by Espressif Systems**. This will create a new folder 4.5.1 under the esptool\_py folder above. Now, we will copy the new **esptool** into the 2.6.1 folder (I know!).

```
cp $HOME/Library/Arduino15/packages/esp32/tools/esptool_py/4.5.1/esptool
$HOME/Library/Arduino15/packages/esp32/tools/esptool_py/2.6.1
```

Now it should compile with a blinking light! Credits go to [here](#), [here](#).

## 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](#) is a useful gateway in a LoRaWAN network. This section will show you how to make your own gateway and access it on the internet.

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. **1.**

#### Download and Install the Libraries

Download the modified ESP 1-ch Gateway code from Canvas (**ESP-1ch-Gateway-v5.0-Azure.zip**) and unzip.

[This is a modified version of the ESP 1-ch Gateway code hosted on [GitHub by things4u](#). Although v6 is the current version, the hookup guide by Sparkfun still uses v5. Therefore, for this lab, the modified version is based on the archived v5 copy hosted: [ARCHIVED ESP-1CH-GATEWAY-V5.0 \(ZIP\)](#). You do not need to download these versions. Use the updated one on Canvas.]

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. Go into the **ESP-1ch-Gateway-v5.0-Azure/libraries/** folder and copy all the libraries (i.e., folders) into the Arduino library folder located in **Documents/Arduino/libraries/** (All common in OSX, WINDOWS, LINUX). [Do not copy the **libraries** folder directly as this may delete the existing Arduino libraries that you will need.]

**\*\*Do NOT Update Libraries !**

After copying over the libraries, and opening the Arduino IDE, you may see a little bell at the bottom asking you to update the libraries. Do NOT.

**\*\*Do NOT Update Libraries !**

### 2. Configure the Gateway Sketch

Go to **ESP-sc-gway/** folder and open the **ESP-sc-gway.ino** with Arduino IDE. This will open multiple tabs including all .ino and .h files in the folder. Before uploading the

ESP-1ch-Gateway sketch to your board, you'll need to modify a few 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 primary source of configuration for the gateway sketch. The definitions you'll have to modify are:

- **Radio**
  - **[Required!]** `_LFREQ` -- This sets the frequency range your radio will communicate on. Make sure this is set to `915` (US). [Note that this is **not** your group's frequency. Just use `915` here.]
  - `_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. If enabled, the tradeoff is that the radio may not pick up very weak signals.
- **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)**
  - There are parameters related to the TTN in the code because this code was originally developed for TTN. However, TTN no longer supports our gateway. So the sketches are updated for Azure. Keep the TTN related code as they won't be used but commenting them out or deleting them may cause issues.
- **WiFi**
  - **[Required!]** Add at least one WiFi network to the `wpa_supplicant.conf` array, but leave the first entry blank. For example: `wpa_supplicant.conf = { { "", "" }, // Reserved for WiFi Manager { "NU-IoT", "" } };`
  - You can also add other WiFi networks by creating additional entries (e.g., your home WiFi network.)

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.

### ESP-sc-gway.ino

Locate the following line:

```
static const char* connectionString = "<YOUR AZURE CONNECTION STRING>";
```

We will update this variable after enabling our Azure IoT hub device in the Azure portal. For now, leave it unchanged.

### 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 here, except the following.

Locate `int freqs[]` under `#elif _LFREQ==915` and modify the frequencies with the



frequencies assigned to your group in the following fashion. The current code is defined based on a center frequency of 915. If your assigned frequency is 906 MHz, then modify 914xxxxxx, 915xxxxxx, 916xxxxxx to 905xxxxxx, 906xxxxxx, 907xxxxxx, respectively. More specifically:

```
914900000 -> 905900000,  
915100000 -> 906900000,  
...  
916600000 -> 907600000,
```

Beyond this, there's not much else in here we recommend modifying.

### 3. Upload the Code and Check NU-IoT WiFi Connection

After configuring the gateway project, upload it to the board. The code is not yet ready to connect to Azure but we will first if we can get connected to the NU-IoT WiFi network. Without the Wi-Fi network, our device will not have a link to connect to the Internet. UNL Wi-Fi is now open for IoT device enrollment, through the NU-IoT network. Your gateway is already registered for the NU-IoT network and your group should have received the password from us.

First, make sure you modified the password field in the `ESP-sc-gway.h`. Try compiling and uploading the sketch to your ESP32 with the program. Make sure the following are selected:

- Tools->Board->SparkFun ESP32 Boards->SparkFun LoRa Gateway 1-Channel
- Tools->Port->[Correct Port Selected]
- Tools->Upload Speed->**115200**

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 complete, you should see the ESP32 attempt to connect to your WiFi network, then initialize the radio.

Among the serial monitor message you should see the following:

```
...  
Connection successful  
...  
Initializing IoT hub failed.
```

The first message shows that you are where you want to be, connected to Internet. The second message tells you we still have work to do to connect to Azure.

---

## 2. Setting up The Azure IoT Hub

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

### Enable Student Credit

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

## IoT Hub


Click to create a resource in the 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](#)

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

Plan

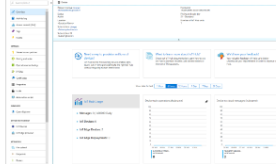
[Create](#)

---

[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 to a resource group. If you do not have a resource group, create a new one. Put a name for your IoT hub and select the 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 \* 

Resource group \*   [Create new](#)

**Instance details**

IoT hub name \*   

Region \* 

---

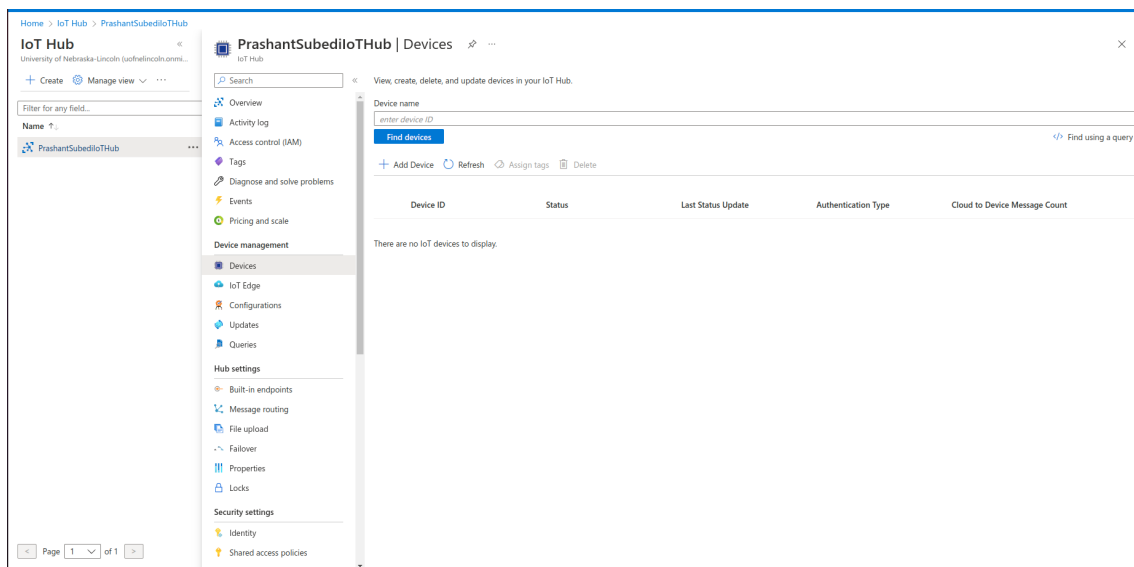
[Review + create](#) [Previous](#) [Next: Networking >](#)

Now click the 'Management' tab. Select 'F1: Free tier' as your pricing and scale tier. Keep others as the default settings. Now click the 'Review+create' button from the

bottom. Once you see the 'Review+create' tab, click the 'Create' button. After a while, you are getting your new Azure IoT hub. For this free tier hub, you can exchange a total of 8,000 messages per day. Therefore, program your device accordingly so that it does not flood your cloud with messages and finish the quota.

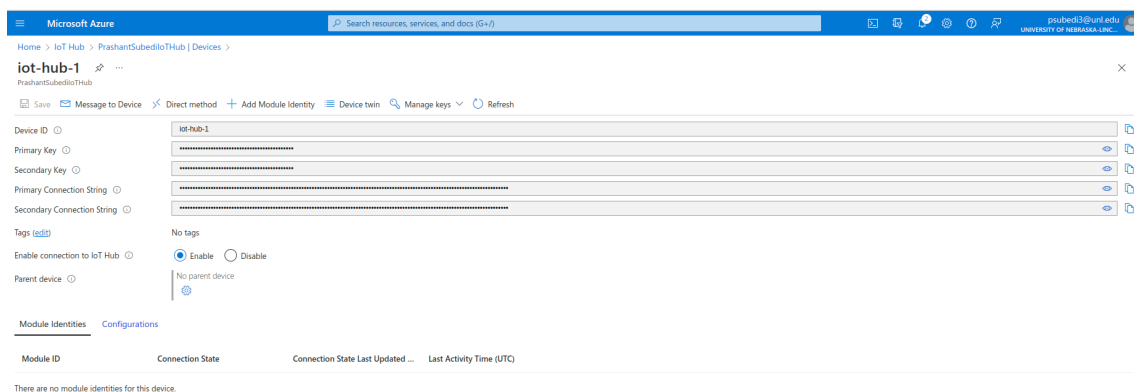
## IoT Device for the Hub

Once you have the hub, add an IoT device to it. Click the newly created hub. Then click 'Devices' from the left side column (or 'Add and configure IoT Devices' in the middle).



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

Now click on that device. It will show all the information about that device. Copy the **Primary Connection String** of it. This connection string needs to be added to `static const char* connectionString` variable inside the `ESP-src-gway.ino` 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 reopen the Arduino IDE. It's time to program the node!

## Test Gateway - Cloud Connection

Compile and upload your updated gateway code. You should see the following in the serial monitor (Yay!)

```
...
Info: >>>Connection status: connected
Initializing IoT hub success.
```

---

## 3. Setting up Your End-Device

Now, we will connect one of the SparkFun Pro RF boards (remember them?) to the Gateway and eventually to the cloud. We will start by installing the LMIC library.

### LMIC

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

### Installing LMIC library

We'll set up the SAMD21 Pro RF as a *node* using a library written by **Matthijs Kooijman**, a modified version of the "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](#). Once you download the zip file go to Arduino IDE Sketch->Include Library->Add .ZIP Library and select the zip file. You'll see a `Library installed` message.

### Configurations

**The LoRa settings should match** The spreading factor, frequency, and bandwidth should match your gateway configuration.

#### Configure LMIC

1. This modified example takes directly from the example code provided by the library with 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 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/` (or `...libraries/MCCI_LoRaWAN_LMIC_library/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:

```
// project-specific definitions
// #define CFG_eu868 1
#define CFG_us915 1
// #define CFG_au915 1
// #define CFG_as923 1
// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP      /* for as923-JP; also
define CFG_as923 */
// #define CFG_kr920 1
// #define CFG_in866 1
#define CFG_sx1276_radio 1
// #define LMIC_USE_INTERRUPTS
```

## Example Code for Sparkfun pro RF LoRaWAN implementation

In `Lab04_LMICClient.ino`, modify `LMIC.freq` to the assigned team frequency. Make sure this is one of the frequencies that you have used in `loramodem.h` assignment of frequencies. E.g. if the assigned frequency is 915Mhz, and you modified `loramodem.h` `int freqs[]` as above, modify `LMIC.freq` to 914900000 (or any other frequency included in `int freqs[]` list).

### Configuring Serial for SAMD21

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

Comment out the line `while(!SerialUSB);` in `setup()` and add a `delay(1000);` instead below. There are two reasons: 1. Waiting for a serial connection is not the best approach in case you'd like to power your end device with any other means (e.g., powerbank). 2. If your gateway and node are connected to the same computer; in most cases, you cannot monitor both serial ports through Arduino (e.g., MacOS does not allow multiple instances of Arduino IDE) - there is another way, see below. In such cases, if you choose to monitor the gateway, the node may never send a package.

Try compiling and uploading the sketch to your SAMD21 Pro RF with the program. Make sure the following are selected:

- Tools->Board->SparkFun SAMD Boards->SparkFun SAMD21 Pro RF
- Tools->Port->[Correct Port Selected]

---

## 4. Receiving Data on Azure from your device

Now let's close the loop.

### 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. We need a "Stream

Analytics job" or any other equivalent data access resource to access data from this device. In this lab, we will utilize the Stream Analytics job.

Like all other resource creation, we need to click the 'Create a resource' button and search for the Stream Analytics job. After clicking the create button, we have to fill in the required information and then click on the 'Review+create' button at the bottom. Then click the '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 \*

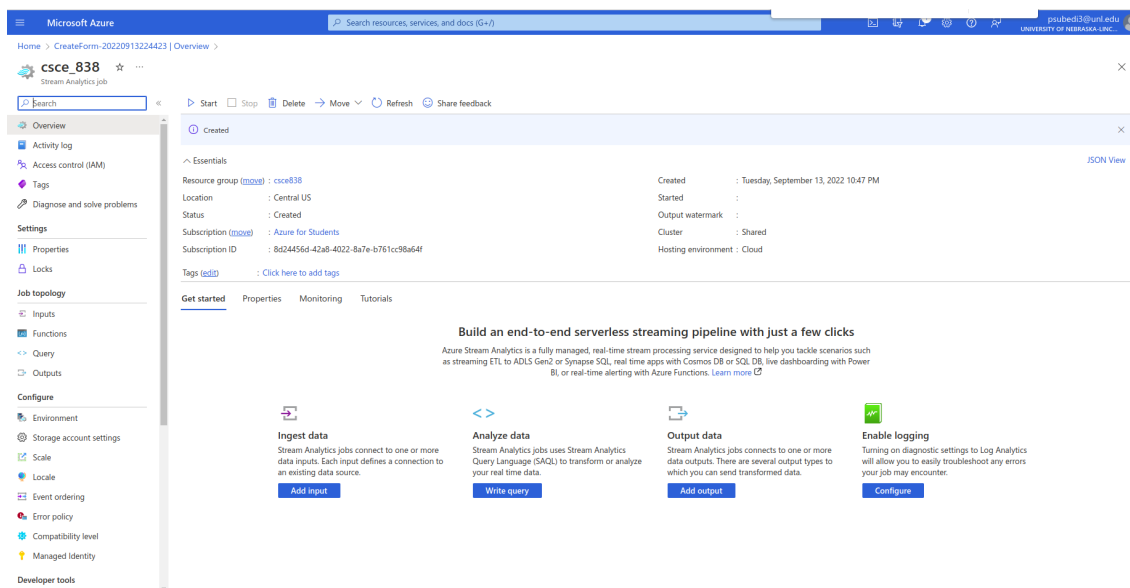
1

Review + create

< Previous

Next : Storage >

Once the job is created, click the 'Go to resource' button.



Here, we need to select inputs and outputs for this job. First click on 'Inputs'. Then click 'Add input' and select 'IoT Hub'. Give an alias name and choose 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 ⓘ

Partition key ⓘ

Event serialization format \* ⓘ

Encoding ⓘ

Event compression type ⓘ

Save

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

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

Replace csce-838-stream with your input alias. **Make sure** you have [ and ] around your alias. [You may want to click two Refresh buttons on the Azure page. One 'Refresh' button is below in the 'Input Preview' section, and the second 'Refresh' button is above the query window. ]

**IMPORTANT!** Before connecting the end-device and gateway, make sure to **connect their antennas**, so that they can communicate.

Once everything is connected, you should see the messages sent to the IoT hub in the result section, as shown in the screenshot below.

## Expected Results

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



```

/dev/ttyACM0
Send

22:18:18.549 -> TX
22:19:18.862 -> TX
22:20:19.326 -> TX
22:21:19.671 -> TX
22:22:20.123 -> TX
22:23:20.320 -> TX
22:24:20.346 -> TX
22:25:20.418 -> TX
22:26:20.584 -> TX
22:27:20.724 -> TX
22:28:20.727 -> TX
22:29:21.156 -> TX
22:30:21.302 -> TX
22:31:21.441 -> TX
22:32:21.576 -> TX

Autoscroll Show timestamp Newline 115200 baud Clear output
```

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

```

Send
22:24:06.019 -> MAC: cc:58:e3:8d:cd:48, len=17
22:24:06.019 -> WlanConnect:: Init para 0
22:24:06.119 -> B1:3, WiFi connect SSID=QuickAccess, pass=cxph3iciq@f8rv7
22:24:15.125 -> A WlanStatus:: CONNECTED to QuickAccess
22:24:15.257 -> Host esp32-8bd4b8 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: CC58E3FF8BD4B8, 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 -> MQTT Server started on port 80
22:24:17.152 -> -----
22:24:17.152 -> Info: Initializing SNMP
22:24:18.158 -> 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: filename=0, rec=1: 1 50 31 0 CC 50 E3 FF 8D CD 48 [{"rxpk":{"t":75821255,"chan":0,"rfch":0,"freq":903.900024,"stat":1,"modu":"LORA","data":{"SF7BW125","codr":"4/5","lenr":9,"
22:25:20.551 -> Hello, world!
22:25:20.584 -> 48 05 6C 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: >>>IoT HubClient_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 Azure stream analytics query tab data panel will look like this if all the links are successfully established. This means you may have implemented your first end-to-end IoT application (double Yay!).

Microsoft Azure

Home > CreateForm-20220913224423 | Overview > csce\_838

csce\_838 | Query

Stream Analytics job

Search

Query language docs Open in VS Code Share feedback Refresh

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 Event ordering Error policy Compatibility level Managed identity Developer tools

Inputs (1) csce-838-stream

Outputs (1) output

Functions (0)

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'.

Message	EventProcessedUtcTime	PartitionId	EventEnqueuedUtcTime	IoT Hub
string	datetime	bigint	datetime	string
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:50.708000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:49.223000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:47.863000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:46.582000Z"	("MessageId":null,"CorrelationId":null,"C...
"Hello, world!"	"2022-09-14T03:51:55.260286Z"	1	"2022-09-14T03:51:45.784000Z"	("MessageId":null,"CorrelationId":null,"C...

Monitoring both boards from a single computer

We recommend using multiple computers in your team for nodes and gateways. Still, if you'd like to connect them both to the same computer, as described above, you may not be able to monitor both serial monitors simultaneously *through Arduino IDE*. Here is a solution for MacOS, which may also work in other OSs.

You can monitor one of the boards through the terminal using the `screen` command. First, find the address(es) of your USB ports:

```
ls /dev/tty.*
```

which would list your USB (and potentially Bluetooth) connections:

```
...  
/dev/tty.usbserial-1110  
/dev/tty.usbmodem1101
```

You can check which port is connected to which board through the Arduino IDE. Let's say the gateway is connected to `...usbserial-1110`. Then, you can monitor this board by

```
screen /dev/tty.usbserial-1110 115200
```

The last part is your baud rate. You can then simultaneously monitor the node through Arduino IDE (useful for checking LoRa connection.)

---

## Lab Assignment

In this lab, you will work with your teammates and improve our IoT system. A lab report is required from each group, one per group. You will need to work together as you need to share the gateway. Each member must provide the individual screenshots of their Azure account and serial monitor outputs in the group report. Make sure you have Azure account results **from each team member** in your report.

## Assignment

### Requirements

1. Finish the helloworld example **for each team member**
  1. Record the procedure of setting up the link from your device to the IoT Hub with screenshots
2. Use the code from the previous lab (Lab 3) and merge it with the LMIC example code for packet transmissions
  1. Remove the radio operations in the Lab 3 code and instead, use the LMIC code.
  2. Use your packet construction modules, average temperature reading modules, etc. from Lab 3.
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

1. Code that fulfills each requirement in this lab
  1. Each function in this system should be separately presented with an explanation. The entire code snippet will not be accepted
2. Serial message from
  1. Sparkfun pro RF device
  2. LoRa gateway
3. Screenshots from Azure and JSON for the data reporting results.

## Report format

- 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
- Appendix
  - The entire program (Arduino sketch) in the Appendix (No screenshots will be accepted). Include only those sketches that you modified.

### Submission Instructions:

1. Submit your lab on Canvas on or before the deadline (Sep 22rd, 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 (Do not attach your code separately as an .ino file. Instead, copy and paste your code in the Appendix. Do not use screenshots in the Appendix.)
4. Failing to follow the instructions will make you lose points.

## References

1. <https://learn.sparkfun.com/tutorials/sparkfun-lora-gateway-1-channel-hookup-guide/all>
2. <https://learn.sparkfun.com/tutorials/lorawan-with-prorrf-and-the-things-network/all#example-ifttt-integration>
3. [https://www.youtube.com/playlist?list=PLlrxD0HtieHh5\\_p0v-6xSMxS3URD6XD52](https://www.youtube.com/playlist?list=PLlrxD0HtieHh5_p0v-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.