Lab 11
WiFi connected Adafruit Feather Bluefruit Sense
using MQTT with Adafruit IO

Part 1 - Prepare your AdafruitIO account then connect with MQTT

Change the name of your Lab 10 dashboard to Lab 11. On the dashboard you should have three blocks. The first block is a Stream block to show the random_value feed. The second block is a Line Chart that also visualizes the random_value feed. The third block is a toggle associated with the output feed.

In Lab 10, you first used basic HTTP Post and Get commands to communicate with AIO before using the adafruit_io library of commands for HTTP communication with AIO. In Lab 11 you will jump right into adafruit_io MQTT commands to communicate with AIO.

Copy the following code into a new Mu editor tab and save it to the Feather Sense as code.py.

```
import time
import board
import busio
import neopixel
from random import randint
from digitalio import DigitalInOut
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
import adafruit_minimqtt.adafruit_minimqtt as MQTT
from adafruit_io.adafruit_io import IO_MQTT

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# AirLift Featherwing
esp32_cs = DigitalInOut(board.D13)
esp32_ready = DigitalInOut(board.D11)
esp32_reset = DigitalInOut(board.D12)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

# set up neopixel as wifi status_light
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)

# initialize wifi manager
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets,
status_light)

# Connect to WiFi
print("*"*70)
print("\tConnecting to WiFi...", end=" ")
wifi.connect()
```

```
print("Connected!")
print("*"*70)

# Define callback functions which will be called when certain events happen.
def connected(client):
    # Connected function will be called when the client is connected to AIO.
    # This is a good place to subscribe to feed changes.  The client parameter
    # passed to this function is the Adafruit IO MQTT client so you can make
    # calls against it easily.

    print("Connected to Adafruit IO!")
    print("*"*70)
    # Subscribe to changes on random-value feed
    client.subscribe("random-value")

def subscribed(client, userdata, topic, granted_qos):
    # This method is called when the client subscribes to a new feed.
    print("Subscribed to {0} with QOS level {1}".format(topic, granted_qos))

def unsubscribed(client, userdata, topic, pid):
    # This method is called when the client unsubscribes from a feed.
    print("Unsubscribed from {0} with PID {1}".format(topic, pid))

def disconnected(client):
    # Disconnected function will be called when the client disconnects.
    print("Disconnected from Adafruit IO!")

def message(client, feed_id, payload):
    # Message function will be called when a subscribed feed has a new value.
    # The feed_id parameter identifies the feed, and the payload parameter has
    # the new value.
    print("Feed {0} received new value: {1}".format(feed_id, payload))

# Initialize MQTT interface with the esp interface
MQTT.set_socket(socket, esp)

# Initialize a new MQTT Client object
mqtt_client = MQTT.MQTT(
    broker="io.adafruit.com",
    username=secrets["aio_username"],
    password=secrets["aio_key"],
)

# Initialize an Adafruit IO MQTT Client
io = IO_MQTT(mqtt_client)

# Connect the callback functions defined above to Adafruit IO MQTT client
io.on_connect = connected
io.on_disconnect = disconnected
io.on_subscribe = subscribed
io.on_unsubscribe = unsubscribed
io.on_message = message

# Connect to Adafruit IO
print("\tConnecting to Adafruit IO...", end=" ")
io.connect()
print("*"*70)

# initialize variable for delay loop
last = 0
# this is a blocking loop, any code below this while loop will never run
print("Publishing a new random_value every 10 seconds...")
while True:
    try:
```

```
        # Explicitly pump the message loop.
        io.loop()

    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        io.reconnect()
        continue
    # Send a new message every 10 seconds.
    if (time.monotonic() - last) >= 10:
            value = randint(25, 50)
            print("Publishing {0} to random_value.".format(value))
            io.publish("random-value", value)
            last = time.monotonic()
```

The code should connect to wifi and then begin “publishing” random values to the random_value feed on AIO. Check your Lab 11 dashboard to confirm it is working.

Take some time to study the code. REALLY study the code this time! This program does some things just as you did in previous labs but using MQTT means you are also doing some things very differently.

Starting at the beginning of the program, the code up to line 44 or so should seem pretty familiar but after that things get interesting. The AIO MQTT object uses callback functions which are then assigned to the different possible AIO client event notifications from the broker.

Here are how the callback functions are tied to the AIO MQTT client called “io” in the program.

```
# Connect the callback functions defined above to Adafruit IO MQTT client
io.on_connect = connected
io.on_disconnect = disconnected
io.on_subscribe = subscribed
io.on_unsubscribe = unsubscribed
io.on_message = message
```

For example:

When the AIO client, “io” receives the `.on_connect` notification the function defined as `connected` will execute.

When the AIO client, “io” receives the `.on_message` notification the function defined as `message` will execute.

The program checks for these notifications using `io.loop()` which is in the while loop of the program. Rather than polling for changes on feeds, the `io.loop()` will receive notifications of changes on subscribed feeds and will then execute the appropriate callback function.

When `io.connect()` is called in line 96, what happens?

Show your instructor your data being collected on AIO and your answer above.

Save your code as lastname_lab11_part1.py. Add header information and inline comments of your code.

Part 2 - Add an "output" feed subscription

Modify your code from part 1 and subscribe to your AIO "output" feed. This should not take more than one line of well placed code. Where does lab11_part1 subscribe to feeds?

Monitor the program's output in the serial console. Does it receive data from both the "random_value" feed and the "output"?

Describe how using MQTT to get updated "output" feed values is different than using HTTP to get updated "output" feed values.

Show your instructor your feeds being received from AIO and your answer above.

Save your code as lastname_lab11_part2.py. Modify header information and inline comments of your code.

Part 3 - Duplicate Lab 8 Part 4 aka Lab 10 Part 3 aka Lab 10 Part 5

In Lab 8 Part 4 and Lab 10 Part 3 and Lab 10 Part 5 the Feather Sense was programmed to recognize a “PAUSE” command from the AIO output feed. The “PAUSE” output was triggered by an Action in AIO. This Action should still be working.

In Lab 11 so far, you have used one callback function to respond to feed updates.

```
io.on_message = message
```

Everytime the AIO client received a notification of a new message, the `“message”` function was executed.

The adafruit_io library also allows us to define specific callback functions for specific feeds. For example:

```
# Set up a message handler for the random_value feed
io.add_feed_callback("random-value", on_random_value_msg)
```

The `.add_feed_callback()` method will associate a new callback function called `on_random_value_msg` with the feed `"random-value"`.

Use this new information to help you to recreate the functionality of the previous labs. Remember, the Feather Sense should be publishing random data values between 25 and 50 to AIO every 10 seconds. If the Feather receives a new “PAUSE” command on the output feed subscription, it should stop sending data points and instead blink the neopixel for 1 minute.

Show your instructor your working code. Use the toggle on the Lab 11 dashboard to force a “PAUSE” ☐

Save your code as lastname_lab11_part3.py. Modify header information and inline comments of your code.

Part 4 - Create a Weather Station

To begin your Weather Station, create a new “Group” in the AIO “Feeds” and call it Weather. Add three new feeds to the Weather Group. Create the feeds: “Temperature”, “Humidity”, and “Pressure”.

Since all three feeds are in the same “group” MQTT allows you to easily subscribe to them all at once with the command:

```
client.subscribe(group_key = group_name)
```

where group_name is a string defined in the program.

Start with your code from Lab 11 Part 3. You can remove the previous subscriptions and callback functions. Keep the basic structure and all the code to connect to wifi and code to set up an AIO client.

Add the group subscription command to the proper callback function. Make sure to define `group_name = "weather"` before calling `io.connect()`.

Test the program by sending random values to each of the new feeds every ten seconds with the following conditions:

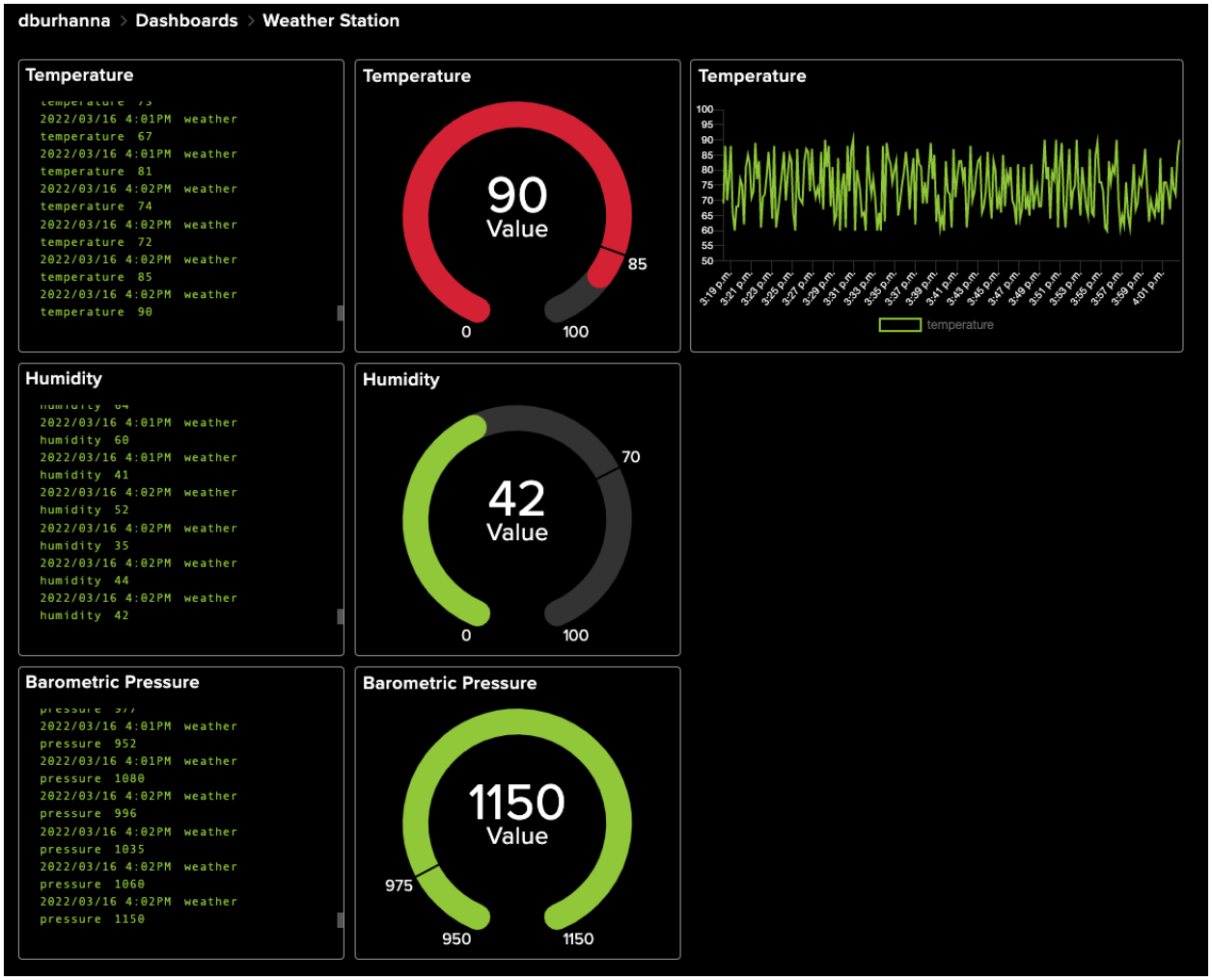
`"weather.temperature"` receives a random integer between 60 and 90

`"weather.humidity"` receives a random integer between 30 and 75

`"weather.pressure"` receives a random integer between 950 and 1150

Save your code to the Feather Sense as code.py. Check the feeds in your "weather" group on AIO. Each of the feeds should be receiving the random values every ten seconds.

Create a new Dashboard called "Weather Station" to help see what is happening. Add blocks to display each of the new feed values. You can choose the blocks that suit you. Here is one possibility:



Each of the blocks has settings which you can use to help visualize your data. The previous dashboard has been configured to have High or Low Warning Values on each of the Gauge blocks.

Show your instructor your working code and the Weather Station dashboard.

Save your code as lastname_lab11_part4.py. Modify header information and inline comments of your code.

Part 5 - Send real sensor values to your Weather Station Dashboard

Modify your code from Part 4. Start sending the true Feather Sense sensor values of temperature, humidity and barometric pressure to your AIO dashboard.

Show your instructor your working code and the Weather Station dashboard.

Save your code as lastname_lab11_part5.py. Modify header information and inline comments of your code.

Part 6 - Add "metadata" to your sensor values to your Weather Station Dashboard

The AIO MQTT .publish() method allows you to send "metadata" along with your data values to any of your data feeds. The metadata AIO accepts is a string of location data. The format for the string is `"longitude, latitude, elevation"`.

All you need to do is define a location string with the proper format for the location of your sensor and include it as the third argument in the .publish() command.

e.g.
```
location_data = "41.82254,-80.98346, 626.6"

io.publish(temperature_feed,temp_value,location_data)
```

Modify your code from Part 5 to include the location metadata. You can use Google Maps to find the longitude and latitude of a location. Open Google Maps and search up a location. Right click anywhere on the map and a pop up will appear with longitude and latitude. Or you can use https://www.freemaptools.com/elevation-finder.htm which will give you latitude, longitude and elevation values.

Check the data feeds in AIO. Where does the metadata appear?

Why is this important? You could have multiple devices in multiple locations publishing to the same AIO feed. By sending the location metadata you will be able to distinguish between sensor locations.

Here is an example:

Show your instructor your working code sending location data.

Save your code as lastname_lab11_part6.py. Modify header information and inline comments of your code.