

## Adafruit IO Environmental Monitor for Feather or Raspberry Pi

Created by Brent Rubell



Last updated on 2018-11-07 09:42:15 PM UTC

## Guide Contents

Guide Contents	2
Overview	3
Parts	4
Adafruit Feather HUZZAH with ESP8266 - Loose Headers	4
Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor	4
Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2	5
Adafruit VEML6070 UV Index Sensor Breakout	5
FeatherWing Proto - Prototyping Add-on For All Feather Boards	5
Lithium Ion Polymer Battery - 3.7v 1200mAh	5
Monitor Enclosure	6
Small Plastic Project Enclosure - Weatherproof with Clear Top	6
Adafruit IO Setup	7
Create Feeds	7
Create a Dashboard	8
Adding Icon Blocks	9
Adding Text Blocks	11
Arduino Wiring and Assembly	15
Wiring	15
Assembly	15
Arduino Setup	21
Arduino Network Config	24
WiFi Config	24
FONA Config	24
Ethernet Config	25
Arduino Code	27
Python Wiring and Assembly	31
Raspberry Pi Zero W	31
Adafruit Perma Proto Bonnet Mini Kit	32
Wiring	32
Assembly	33
Small Plastic Project Enclosure - Weatherproof with Clear Top	34
Making it Portable	35
Python Setup	37
Enable I2C	37
Installing required CircuitPython Libraries	37
Python Code	38
Code	40

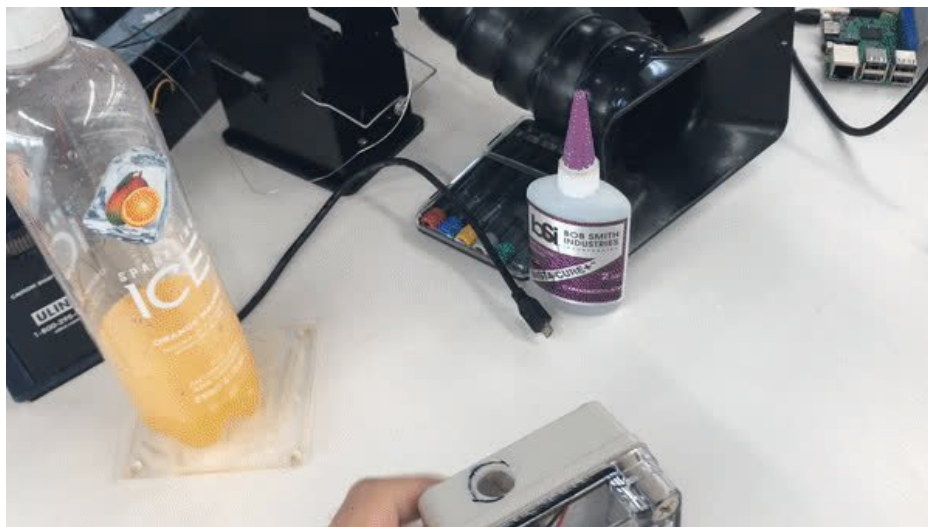
## Overview



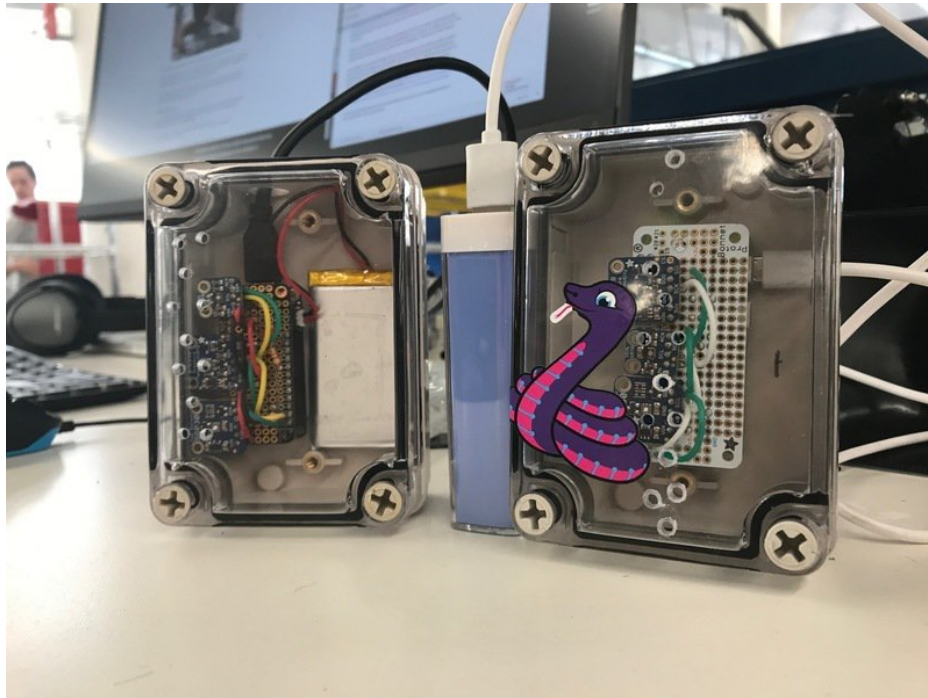
Knowing what's in the air you breathe is important - and building an environmental monitor is a way to visualize the invisible properties of the air you inhale.

This guide covers building a small, internet-enabled environmental monitor which can track a range of data such from temperature to UV-level to the amount of total-volatile-organic-compounds present in the air.

Build it, place it in a location you'd like to log, and then monitor it from anywhere in the world using our Internet-of-Things platform - [Adafruit IO \(https://adafruit.io/fsU\)](https://adafruit.io/fsU).

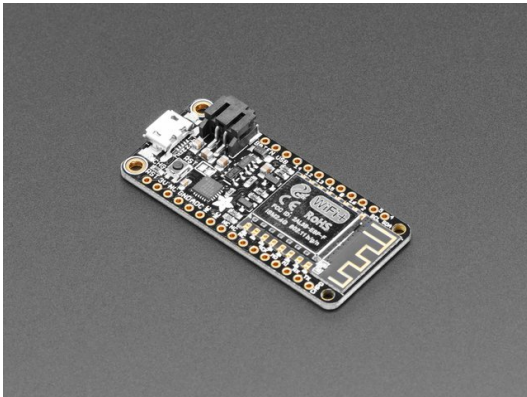


This project can be done with either Arduino or CircuitPython, too!



## Parts

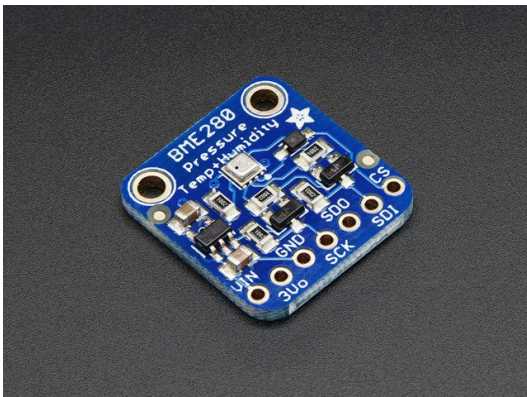
We'll need a Feather HUZZAH and a handful of sensors to build our own environmental monitor.



Adafruit Feather HUZZAH with ESP8266 - Loose Headers

\$16.95  
IN STOCK

[ADD TO CART](#)

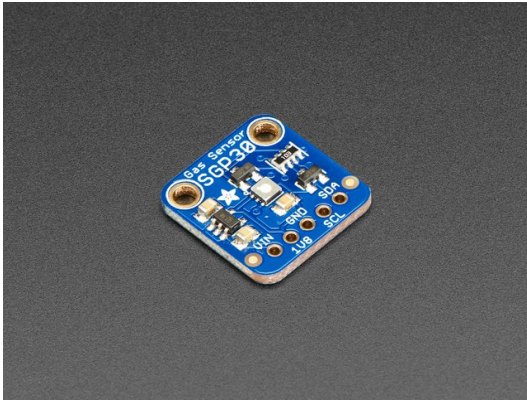


Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor

\$19.95  
IN STOCK

[ADD TO CART](#)

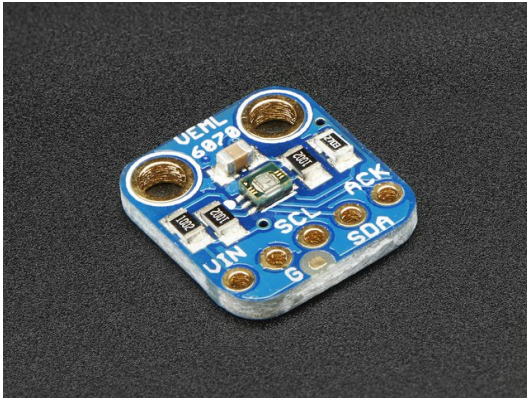




Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2

\$19.95  
IN STOCK

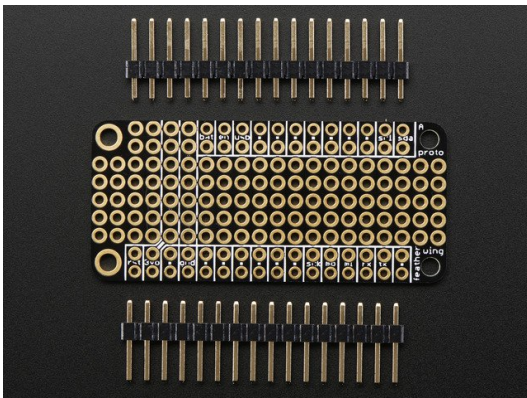
[ADD TO CART](#)



Adafruit VEML6070 UV Index Sensor Breakout

\$5.95  
IN STOCK

[ADD TO CART](#)



FeatherWing Proto - Prototyping Add-on For All Feather Boards

\$4.95  
IN STOCK

[ADD TO CART](#)



Lithium Ion Polymer Battery - 3.7v 1200mAh

\$9.95  
IN STOCK

[ADD TO CART](#)

## Monitor Enclosure

If you'd like to enclosure your environmental monitor, consider picking up the following parts:



Small Plastic Project Enclosure - Weatherproof with Clear Top

\$9.95  
IN STOCK

ADD TO CART

---

### 1 x [Cable Gland](#)

Cable Gland PG-7 size - 0.118" to 0.169" Cable Diameter

ADD TO CART

---

### 4 x [Stranded-Core Wire](#)

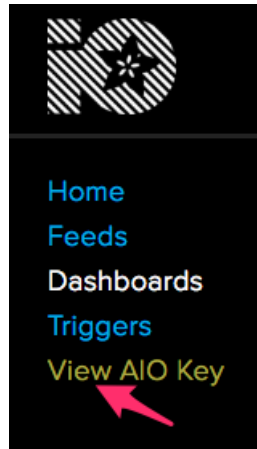
Silicone Cover Stranded-Core Wire - 2m 26AWG

ADD TO CART

---

## Adafruit IO Setup

After logging into Adafruit IO, click the **VIEW AIO KEY** button on the left-sidebar.



A window will pop up with your Adafruit IO key and username. Keep a copy of them in a safe place, we'll need it later.

### YOUR AIO KEY



Your Adafruit IO key should be kept in a safe place and treated with the same care as your Adafruit username and password. People who have access to your AIO key can view all of your data, create new feeds for your account, and manipulate your active feeds.

If you need to regenerate a new AIO key, all of your existing programs and scripts will need to be manually changed to the new key.



**Your AIO Username**

Username

**Your AIO Key**

Active Key

REGENERATE AIO KEY

[Show Code Samples](#)

## Create Feeds

We're going to have a **lot** of sources of data. Our three sensors generate **seven** different types of data.

Luckily for us, Adafruit IO is excellent at tracking data. It keeps data in elements called Feeds, with one feed per each unique source of data.

If you do not know how to create feeds, [head over to the Adafruit IO Basics: Feeds for a quick overview \(https://adafru.it/ioA\)](https://adafru.it/ioA) of this process.

We're going to **create seven unique feeds** to hold and display data generated by the BME280, SGP30, and VEML6070 sensors:

- temperature
- humidity
- pressure
- altitude

- uv
- tvoc
- ecO2

## Create a new Feed

Name

Description

Add to groups

Cancel

Create

IO+ Only

If you have [Adafruit IO Plus \(https://adafru.it/CZp\)](https://adafru.it/CZp) (unlimited feeds), we'll set up seven additional feeds which will be used to specify what will be displayed using icon blocks:

- temperature-block
- humidity-block
- pressure-block
- altitude-block
- uv-block
- tvoc-block
- ecO2-block

## Create a Dashboard

Now that we have feeds to hold data and display the type of data, we need a way to consolidate these feeds in one place. Dashboards are a feature of Adafruit IO which allow us to display and control feeds using widgets called *Blocks*.

- If you haven't used an Adafruit IO Dashboard before, [check out the \*Adafruit IO Basics: Dashboards learn guide\* \(https://adafru.it/f5m\)](https://adafru.it/f5m) for a quick primer.

We're going to navigate to the **Dashboards** page and click **Create a new Dashboard**. You can name it anything you'd like but we'll name it **Environmental Monitor**



## Create a new Dashboard



Name

Environmental Monitor

Description

Cancel

Create

## Adding Icon Blocks

Blocks are widgets which you can add to your dashboard. We're going to first add some *icon blocks*, a *new* type of output block which displays an icon specified by a feed.

On the dashboard, click the **green plus icon** to add a new block. Click on the Icon Block to configure it:

### Create a new block



Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.



We'll select the *temperature-block* feed we created earlier:

## Choose feed



**Icon:** A visible indicator of feed state using custom icons. [Click here](#) to search for valid icon types. Send the icon name to the selected feed to display the named icon inside the block.

If you have lot of feeds, you may want to use the search field. You can also create a feed quickly below.

Group / Feed	Last value	Recorded
My Feeds		
<input checked="" type="checkbox"/> temperature-block		a few seconds ago

Next, we're going to configure the color of the block, and pick an icon type. We're using `w:thermometer` for the temperature block, but you can pick any icon on the [Icons FAQ page \(https://adafru.it/C8S\)](https://adafru.it/C8S). You can also configure the color of the icon using the color-picker from the *Block Settings*.

## Block settings



In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)

Color

Block Preview

Icon A visible indicator of feed state using custom icons. [Click here](#) to search for valid icon types. Send the icon name to the selected feed to display the named icon inside the block.

Test Value

After adding the block to the dashboard, you'll notice there's no icon on the dashboard. That's because the feed is currently empty - there's no icon to display.

Let's fix that by specifying an icon for the feed. Navigate to the **Feeds page** and click on the **temperature-block** feed. From the feed, we're going to click **Actions -> Add Data** and add a the value `w:thermometer` to set the icon block on the dashboard.

## Add Data



VALUE

Cancel

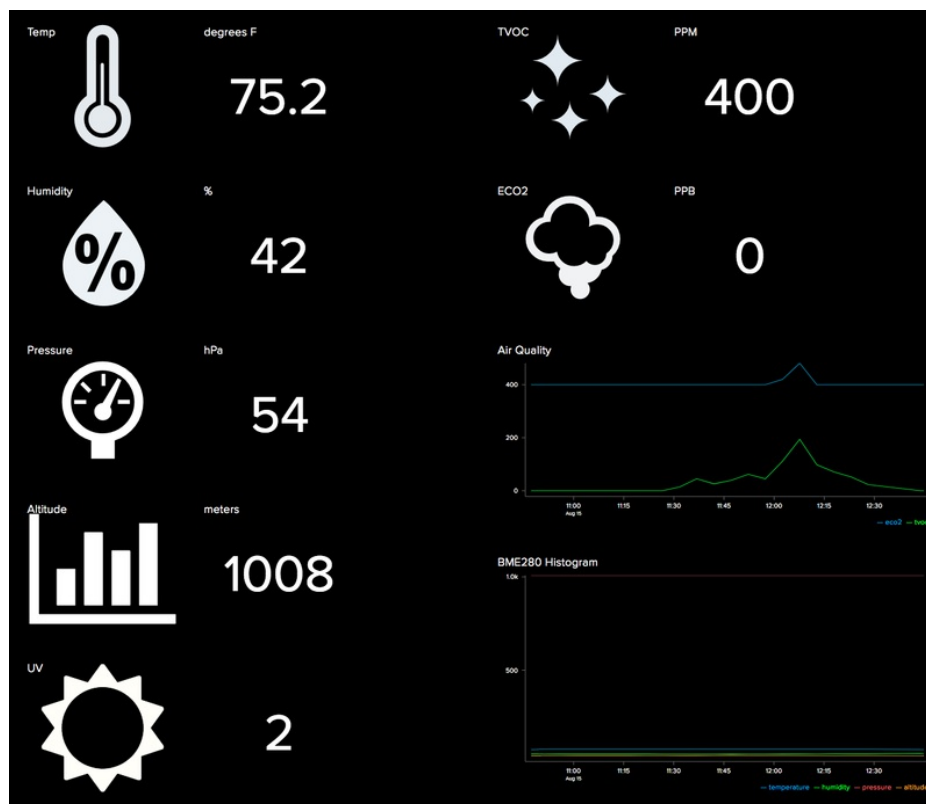
Create

If you navigate back to the dashboard, you'll notice the thermometer icon is now displayed. We're going to repeat this process for each feed we'd like to display.

We set up a dashboard using the following icons, but you can use anything from the [Icon Listing page \(https://adafru.it/C8S\)](https://adafru.it/C8S) you'd like:

- Humidity, by setting the feed value to **w:humidity**
- Temperature, **w:thermometer**
- Pressure, **w:barometer**
- Altitude, **bar-chart**
- eCO2, **w:smoke**
- TVOC, **tvoc-icon**
- UV, **uv-feed-block**

Our dashboard uses the values above. We also added two **histogram blocks** to take a peek at some of the values over an hour.



## Adding Text Blocks

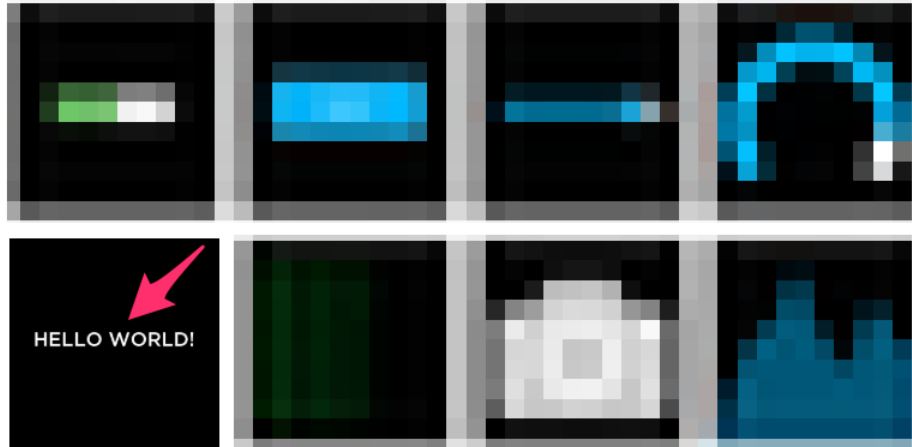
The icon block is a great way to graphically display what types of data are associated with a feed. We need a way to add the data from the sensors. To do this, we'll need to add **seven text blocks** to hold feeds for our sensor data.

Let's start by creating a text block to hold the temperature data from the BME280 sensor. From the dashboard, **add a block** and **select the text block**.

## Create a new block



Click on the block you would like to add to your dashboard. You can always come back and switch the block type later if you change your mind.



Choose the **data feed** (we're going to use the *temperature* feed) to display:

## Choose feed

**Text:** A text block can be used to send data as we

If you have lot of feeds, you may want to use the

Group / Feed
<input type="checkbox"/> My Feeds
<input checked="" type="checkbox"/> temperature

Then, select a **large** font size.

## Block settings



In this final step, you can give your block a title and see a preview of how it will look. Customize the look and feel of your block with the remaining settings. When you are ready, click the "Create Block" button to send it to your dashboard.

Block Title (optional)

Font Size

Large ▾

Block Preview



Text A text block can be used to send data as well as view data.

Test Value

◀ Previous step

Create block

Repeat this for the **humidity**, **pressure**, **altitude**, **temperature**, **uv**, **tvoc**, and **eco2** feeds.

While you can't "see" them on the background (since those feeds are currently empty), we can view them by clicking the **green gear** on the dashboard.



The outlines of the text blocks should now be visible. Reorganize the blocks however you'd like - we can always change them later.





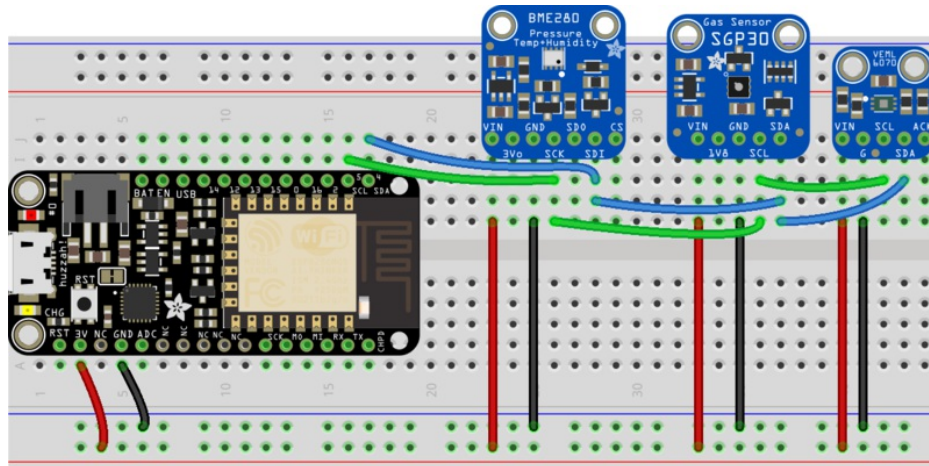
Next, let's wire everything up!

# Arduino Wiring and Assembly

## Wiring

We will be connecting multiple I2C devices to a single I2C controller using only two wires - SCL and SDA.

- *Interested in learning more about I2C Addressing?* [Check out our learn guide on connecting multiple I2C devices \(https://adafru.it/BK0\).](https://adafru.it/BK0)



Make the following connections between the **Adafruit Feather Huzzah** and the **BME280**:

- Feather 3V to BME280 Vin
- Feather GND to BME280 GND
- Feather SCL to BME280 SCK
- Feather SDI to BME280 SDI

Make the following connections between the **BME280** and the **SGP30**:

- BME280 SCK to SGP30 SCL
- BME280 SDI to SGP30 SDA
- Feather 3V to SGP30 Vin
- Feather GND to SGP30 GND

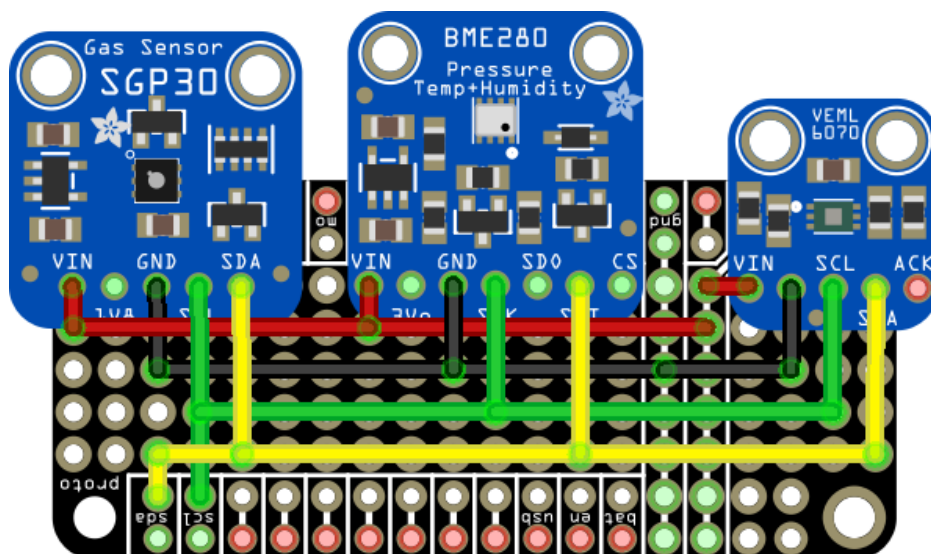
Make the following connections between the **VEML6070** and the **SGP30**:

- SGP30 SCL to VEML6070 SCL
- SGP30 SDA to VEML6070 SDA
- Feather 3V to VEML6070 Vin
- Feather GND to VEML6070 GND

## Assembly

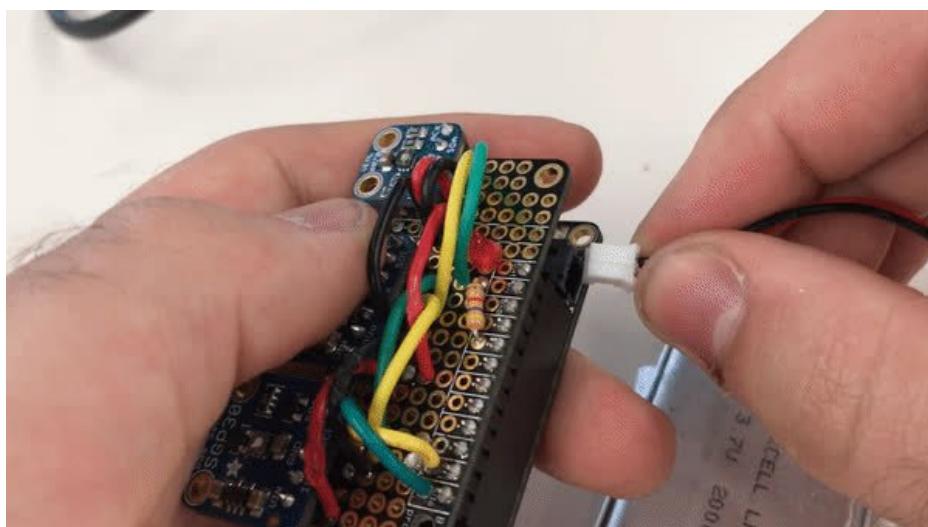
If you'd like to make this portable using the [Adafruit Featherwing Proto \(https://adafru.it/wfu\)](https://adafru.it/wfu) and a project enclosure, we have some great news, You can fit *all* the sensors on the FeatherWing Proto. The wiring between the sensors and the feather are identical to the connections made above.

**Note:** The ACK pin for the VEML6070 is unused, so we're going to hang it off the side so we can fit everything on the board.



The [small plastic project enclosure](https://adafruit.it/C8T) (<https://adafruit.it/C8T>) is the perfect size for a feather and a medium-sized LiPo battery. It's also rugged and has a waterproof gasket seal.

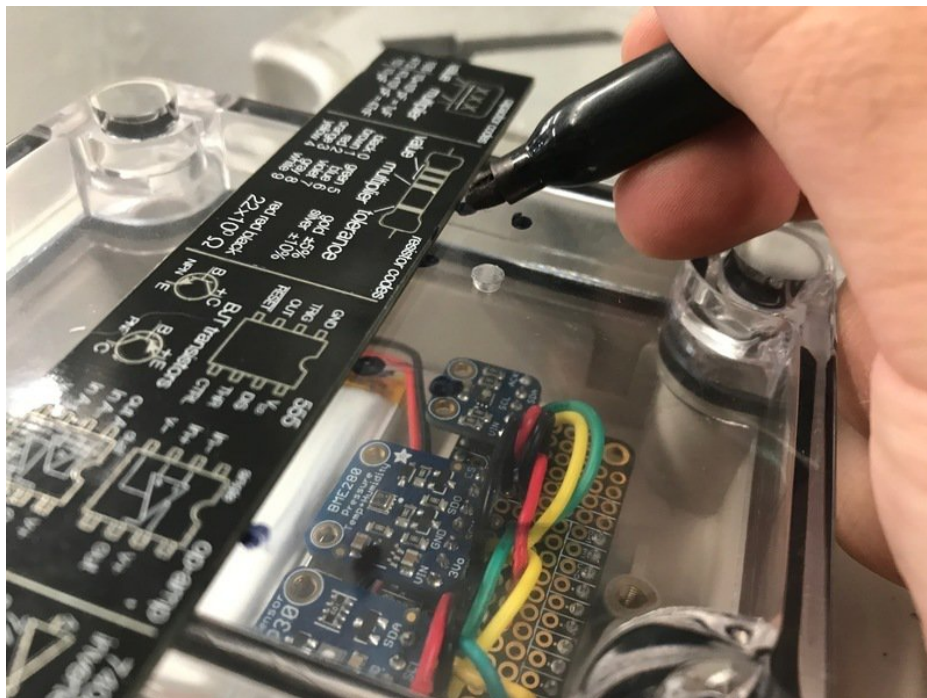
Connect the Feather Huzzah's JST port to a [lipo battery](https://adafruit.it/C8U) (<https://adafruit.it/C8U>) using the included JST connector on the battery:



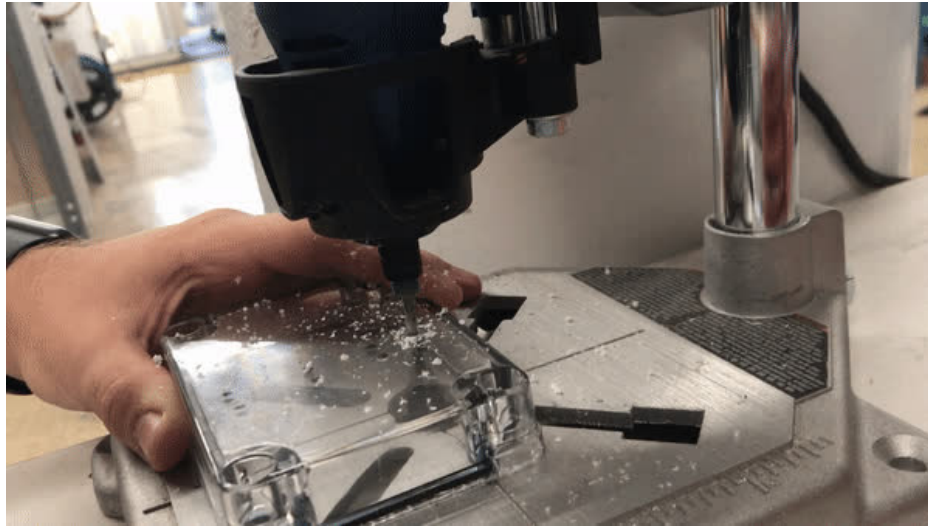
Next, we're going to test-fit the feather and the lipo battery in the project enclosure. Use a non-permanent marker to outline the positions of the battery and the feather on the bottom of the case.



Put the lid on the box and screw it down to ensure everything fits correctly. Use a ruler and a marker to mark locations for holes on the plastic lid of your project box. Be sure to mark the locations closer to the sensors.



Let's drill some holes! I used a rotary tool with a 4mm drill bit mounted to a drill press. Be sure to hold the lid tightly, use a low RPM on your tool, and watch your fingers!

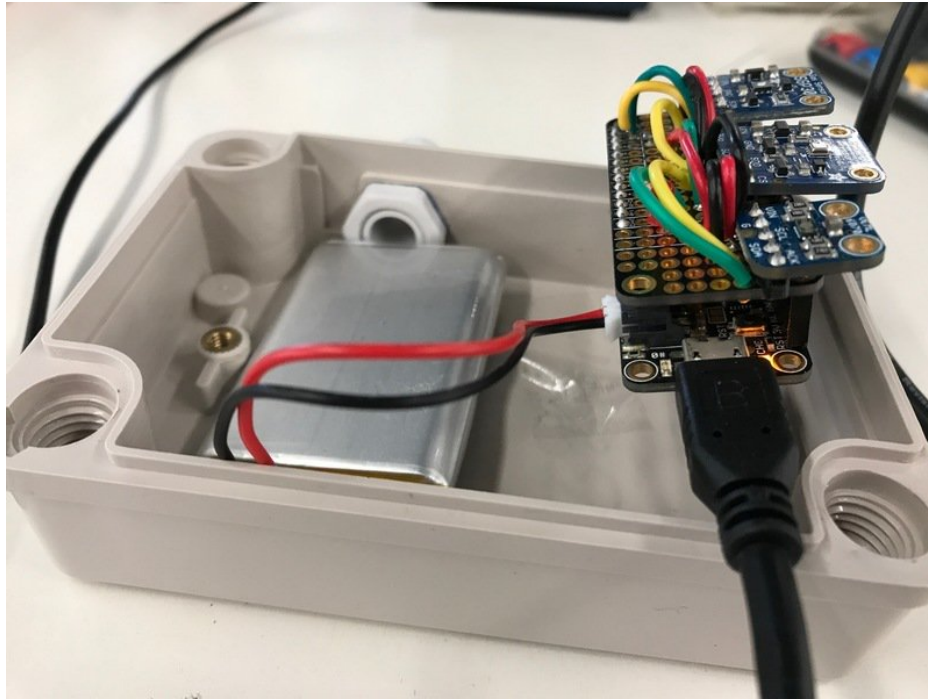


Next, we're going to attach a [cable gland \(https://adafru.it/C8V\)](https://adafru.it/C8V) to seal off the USB cable from the elements. Drill a 5/8" hole on the side of the enclosure where the Feather HUZZAH's USB port is located. The cable gland has integrated screw threads - twist it into the hole you drilled and attach the retaining nut.





You'll need to cut and splice the Micro USB cable to fit it into the enclosure through the cable gland. Plug a Micro-USB cable into the Feather Huzzah and your computer. The green **CHG** LED should turn on to indicate the lipo is charging.



Next, we're going to load the environmental monitoring code on your Feather.

## Arduino Setup

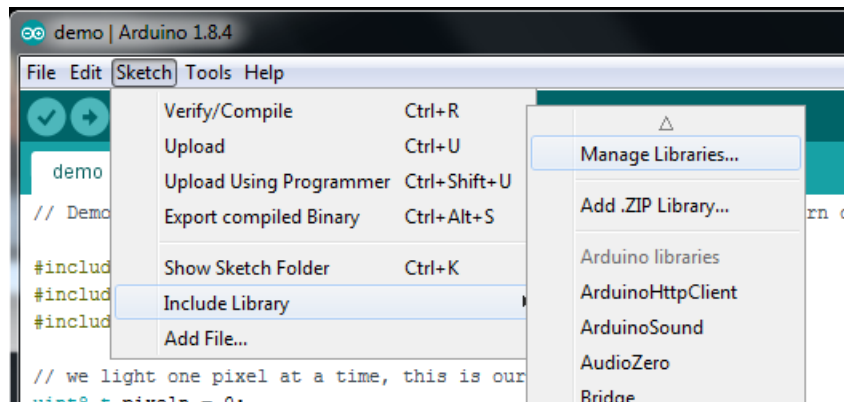
You should first go through the setup guide associated with your selected hardware, and make sure you have internet connectivity before continuing. The following links will take you to the guides for your selected platform:

- [HUZZAH ESP8266](https://adafru.it/ufK) (<https://adafru.it/ufK>)
- [HUZZAH ESP32](https://adafru.it/C8W) (<https://adafru.it/C8W>)
- [Feather MO WiFi](https://adafru.it/C8X) (<https://adafru.it/C8X>)
- [WICED Feather](https://adafru.it/C8Y) (<https://adafru.it/C8Y>)

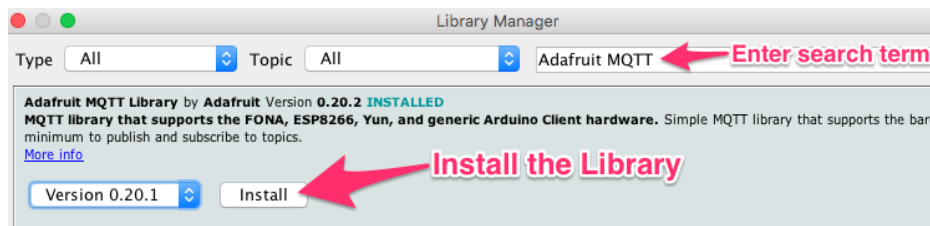
To read sensor data from the SGP30, you will need to [install the Adafruit\\_SGP30 library \(code on our github repository\)](#) (<https://adafru.it/BnZ>). It is available from the Arduino library manager so we recommend using that.

- Not sure how to use the **Library Manager**? [We have a great learn guide about installing libraries](#) (<https://adafru.it/dit>):

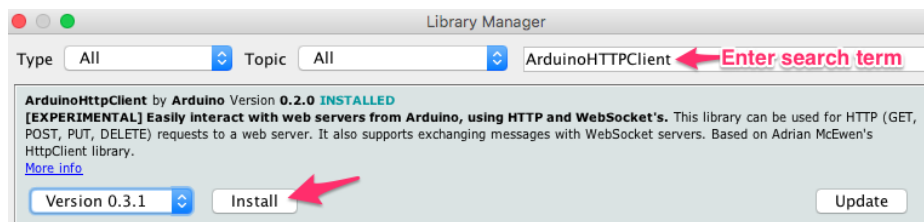
From the Arduino IDE, open the **Library Manager**



Enter **Adafruit MQTT** into the search box, and click **Install** on the **Adafruit MQTT** library option to install version 0.20.2 or higher.

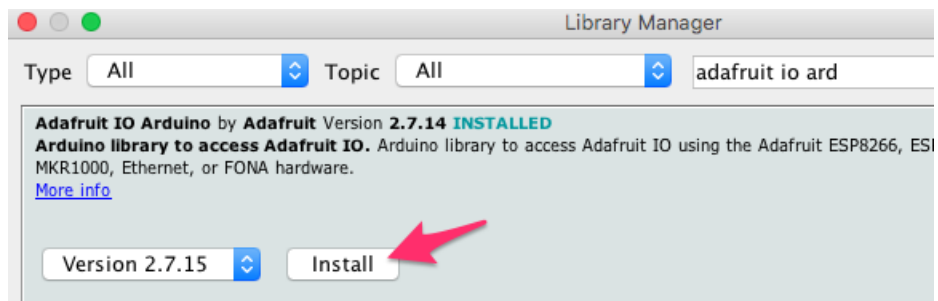


Enter **ArduinoHttpClient** into the search box, and click **Install** on the **ArduinoHttpClient** library option to install version 0.3.0 or higher.

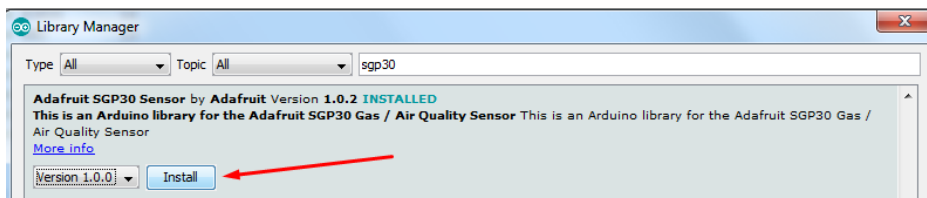


If you haven't installed our Adafruit IO Arduino Library or if you have an older version of this library, you will need to

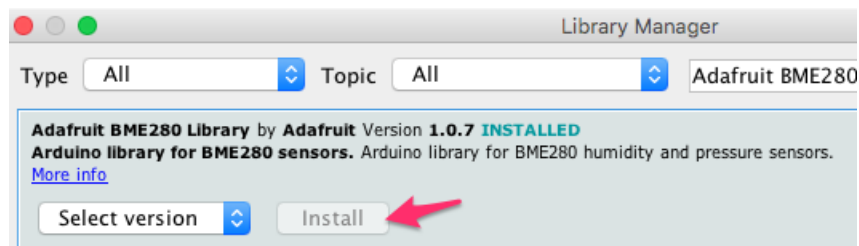
make sure you install version 2.7.15 of the Adafruit IO Arduino library installed before continuing:



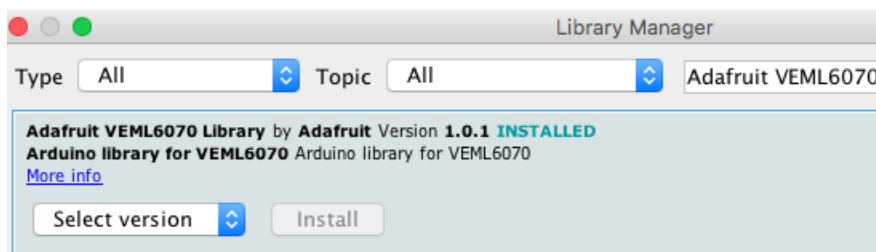
Type in **Adafruit SGP30** to search for the library. Click **Install**.



Next, we're going to install the **Adafruit BME280 Sensor Library**. In the Library Manager, type in **Adafruit BME280** to search for the library. Click **Install**.

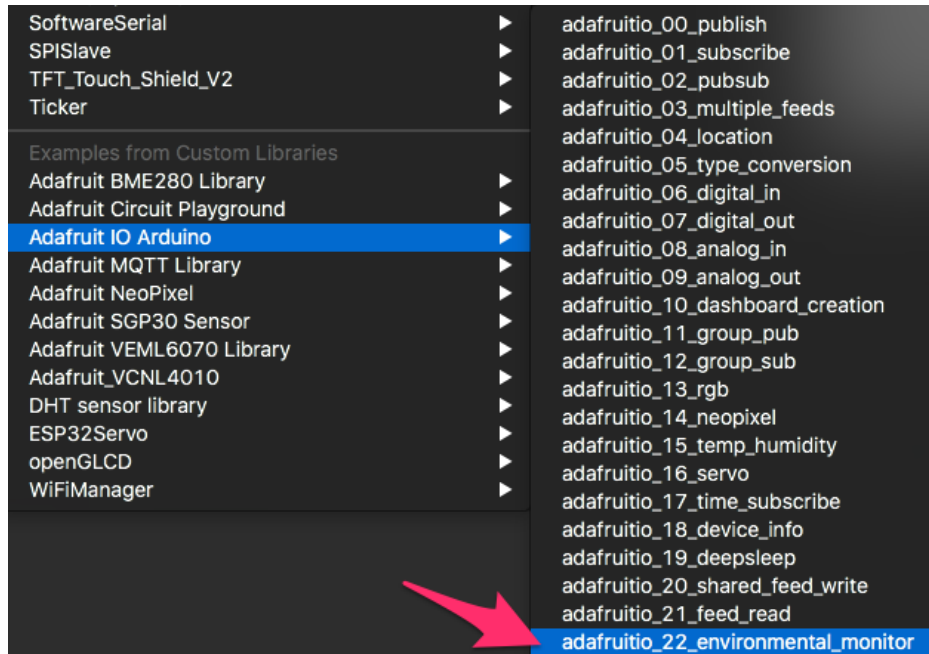


Finally, we'll install the **Adafruit VEML6070** library. In the Library Manager, type in **Adafruit VEML6070** to search for the library. Then, click **Install**.

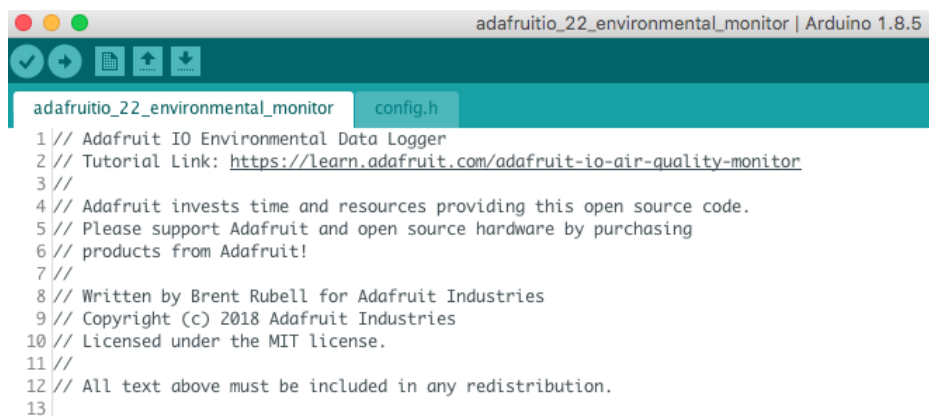


The code for this learn guide is contained within the Adafruit IO Arduino Library's examples.

To open it from the Arduino IDE, navigate to: **File -> Examples -> Adafruit IO Arduino -> adafruitio\_22\_environmental\_monitor**



At this point, you should see two tabs open on your Arduino IDE: **adafruitio\_22\_environmental\_monitor** and **config.h**



Next, we're going to configure our Feather for use with Adafruit IO.



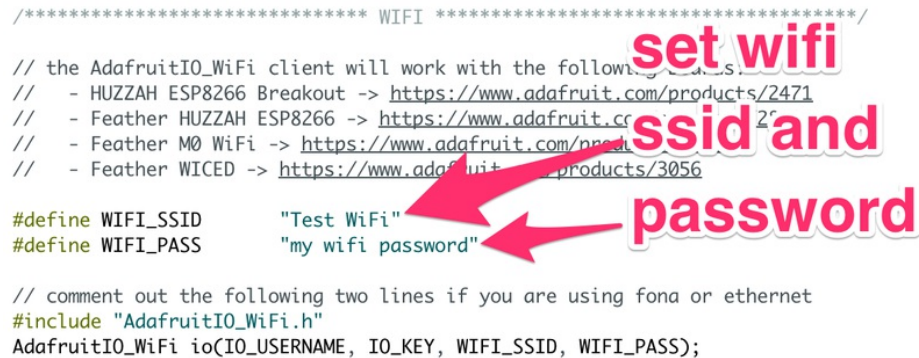
## Arduino Network Config

To configure the network settings, click on the **config.h** tab in the sketch. You will need to set your Adafruit IO username in the **IO\_USERNAME** define, and your Adafruit IO key in the **IO\_KEY** define.



## WiFi Config

WiFi is enabled by default in **config.h** so if you are using one of the supported WiFi boards, you will only need to modify the **WIFI\_SSID** and **WIFI\_PASS** options in the **config.h** tab.

A screenshot of the Arduino IDE showing the WiFi configuration section of 'config.h'. A large red text overlay with a white outline reads 'set wifi ssid and password'. Two red arrows point from this text to the 'WIFI\_SSID' and 'WIFI\_PASS' definitions. The code includes a comment about the AdafruitIO\_WiFi client and lists supported boards with their respective URLs. The definitions are: '#define WIFI\_SSID "Test WiFi"' and '#define WIFI\_PASS "my wifi password"'. Below these are comments about commenting out the following lines if using fona or ethernet, followed by '#include "AdafruitIO\_WiFi.h"' and 'AdafruitIO\_WiFi io(IO\_USERNAME, IO\_KEY, WIFI\_SSID, WIFI\_PASS);'.

```
/***** WIFI *****/  
  
// the AdafruitIO_WiFi client will work with the following boards:  
// - HUZZAH ESP8266 Breakout -> https://www.adafruit.com/products/2471  
// - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2472  
// - Feather M0 WiFi -> https://www.adafruit.com/products/2473  
// - Feather WICED -> https://www.adafruit.com/products/3056  
  
#define WIFI_SSID "Test WiFi"  
#define WIFI_PASS "my wifi password"  
  
// comment out the following two lines if you are using fona or ethernet  
#include "AdafruitIO_WiFi.h"  
AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

## FONA Config

If you wish to use the FONA 32u4 Feather to connect to Adafruit IO, you will need to first comment out the WiFi support in **config.h**

```

/***** WIFI *****/

// comment out default
// wifi config lines
// - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2821
// - Feather WiFi -> https://www.adafruit.com/products/3010
// - WiFi -> https://www.adafruit.com/products/3056

#define WIFI_SSID "Test WiFi"
#define WIFI_PASS "my wifi password"

// comment out the following two lines if you are using fona or ethernet
// #include "AdafruitIO_WiFi.h"
// AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);

```

Next, remove the comments from both of the FONA config lines in the FONA section of **config.h** to enable FONA support.

```

/***** FONA *****/

// the AdafruitIO_FONA library is:
// - Feather 32u4 FONA -> https://www.adafruit.com/product/3027

// uncomment the following two lines if you are using fona
// uncomment both
// fona config lines
// comment out the AdafruitIO_WiFi client in the WIFI section
#include "AdafruitIO_FONA.h"
AdafruitIO_FONA io(IO_USERNAME, IO_KEY);

```

## Ethernet Config

If you wish to use the Ethernet Wing to connect to Adafruit IO, you will need to first comment out the WiFi support in **config.h**

```

/***** WIFI *****/

// comment out default
// wifi config lines
// - Feather HUZZAH ESP8266 -> https://www.adafruit.com/products/2821
// - Feather WiFi -> https://www.adafruit.com/products/3010
// - WiFi -> https://www.adafruit.com/products/3056

#define WIFI_SSID "Test WiFi"
#define WIFI_PASS "my wifi password"

// comment out the following two lines if you are using fona or ethernet
// #include "AdafruitIO_WiFi.h"
// AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);

```

Next, remove the comments from both of the Ethernet config lines in the Ethernet section of **config.h** to enable Ethernet Wing support.

```

/***** ETHERNET *****/
// the Adafruit Ethernet FeatherWing boards:
// - Ethernet FeatherWing -> https://www.adafruit.com/products/3201
// Comment the AdafruitIO_WiFi client in the WiFi section
// and comment out the AdafruitIO_WiFi client in the WiFi section
#include "AdafruitIO_Ethernet.h"
AdafruitIO_Ethernet io(IO_USERNAME, IO_KEY);

```

**uncomment both ethernet config lines**

Next, we will look at how the example sketch works.

## Arduino Code

The code waits ten seconds between sensor reads and publishing to Adafruit IO. We can adjust this by increasing or decreasing the `READ_DELAY` variable at the top of our code.

```
// Delay between sensor reads, in seconds
#define READ_DELAY 10
```

The next chunk of code sets up feed instances to hold the data produced by the sensors.

```
// set up the feeds for the BME280
AdafruitIO_Feed *temperatureFeed = io.feed("temperature");
AdafruitIO_Feed *humidityFeed = io.feed("humidity");
AdafruitIO_Feed *pressureFeed = io.feed("pressure");
AdafruitIO_Feed *altitudeFeed = io.feed("altitude");
// set up feed for the VEML6070
AdafruitIO_Feed *uvFeed = io.feed("uv");
// set up feeds for the SGP30
AdafruitIO_Feed *tvocFeed = io.feed("tvoc");
AdafruitIO_Feed *ecO2Feed = io.feed("ecO2");
```

In the setup function, we make calls to `setupBME280()` and `setupSGP30()`, which set up both of these sensors. We're also going to set up the VEML6070 by calling `uv.begin()`, then connect to Adafruit IO. The code will wait until you have a valid connection to Adafruit IO before continuing with the sketch.

If you have any issues connecting, check `config.h` for any typos in your username, key, or WiFi credentials.

```

void setup() {
  // start the serial connection
  Serial.begin(9600);

  // wait for serial monitor to open
  while (!Serial);

  Serial.println("Adafruit IO Environmental Logger");

  // set up BME280
  setupBME280();
  // set up SGP30
  setupSGP30();
  // setup VEML6070
  uv.begin(VEML6070_1_T);

  // connect to io.adafruit.com
  Serial.print("Connecting to Adafruit IO");
  io.connect();

  // wait for a connection
  while (io.status() < AIO_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }

  // we are connected
  Serial.println();
  Serial.println(io.statusText());
}

```

The code inside the `loop()` obtains values from the sensors and saves their current state. The SGP30 sometimes fails on read (due to timing). If this occurs, we'll set the measurements of `tvocReading` and `ECO2Reading` to -1 so we can catch this type of error in our dashboard.



```

Serial.println("Reading Sensors...");

// Read the temperature from the BME280
temperatureReading = bme.readTemperature();

// convert from celsius to degrees fahrenheit
temperatureReading = temperatureReading * 1.8 + 32;

Serial.print("Temperature = "); Serial.print(temperatureReading); Serial.println(" *F");

// Read the pressure from the BME280
pressureReading = bme.readPressure() / 100.0F;
Serial.print("Pressure = "); Serial.print(pressureReading); Serial.println(" hPa");

// Read the altitude from the BME280
altitudeReading = bme.readAltitude(SEALEVELPRESSURE_HPA);
Serial.print("Approx. Altitude = "); Serial.print(altitudeReading); Serial.println(" m");

// Read the humidity from the BME280
humidityReading = bme.readHumidity();
Serial.print("Humidity = "); Serial.print(humidityReading); Serial.println("%");

// VEML6070
uvReading = uv.readUV();
Serial.print("UV Light Level: "); Serial.println(uvReading);
if(! sgp.IAQmeasure()){
  tvocReading = -1;
  ec02Reading = -1;
} else {
  tvocReading = sgp.TVOC;
  ec02Reading = sgp.eCO2;
}

```

The final chunk of the `loop()` function sends sensor data to the feeds associated with that data. We also delay the polled loop to wait for the temperature to change.

```

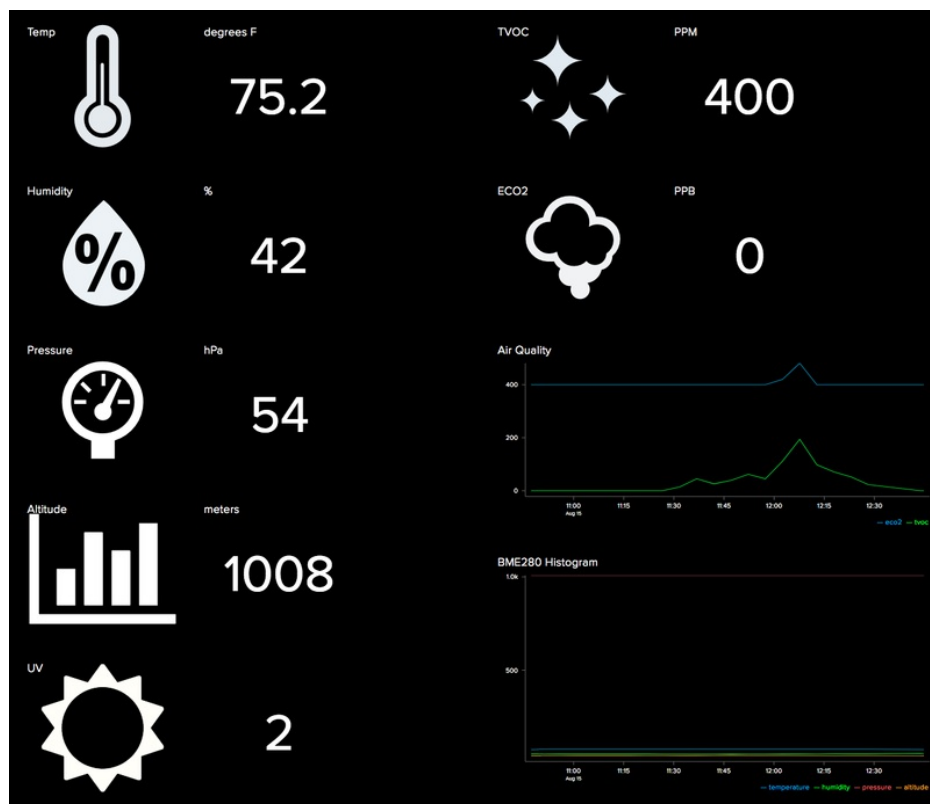
// send data to Adafruit IO feeds
temperatureFeed->save(temperatureReading);
humidityFeed->save(humidityReading);
altitudeFeed->save(altitudeReading);
pressureFeed->save(pressureReading);
uvFeed->save(uvReading);
ec02Feed->save(ec02Reading);
tvocFeed->save(tvocReading);

```

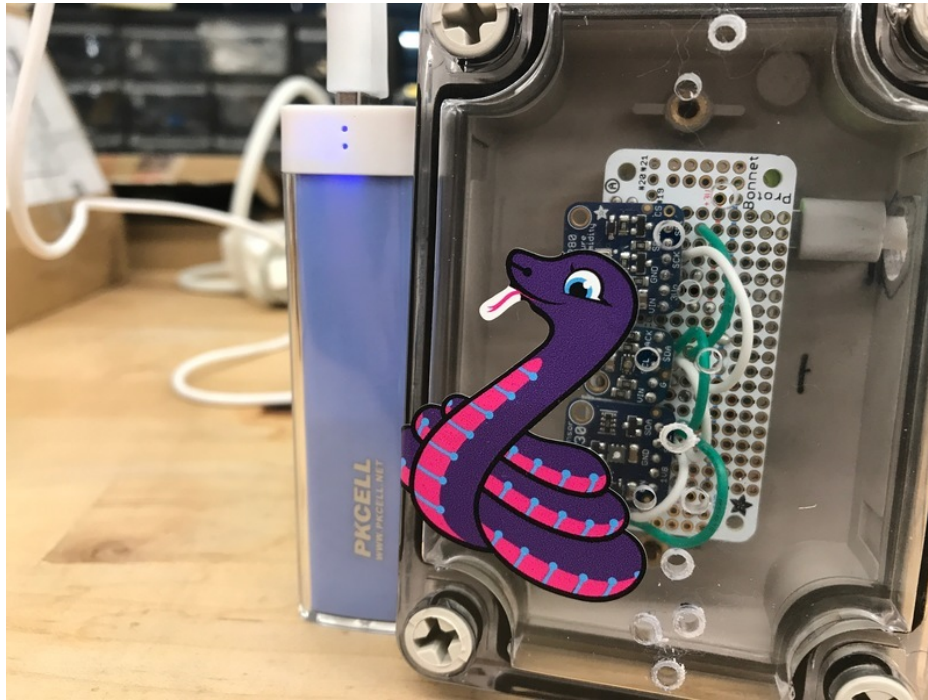
Upload the sketch to your board and open the Arduino Serial Monitor (**Tools -> Serial Monitor**). You should see the board connecting to Adafruit IO, obtain sensor readings, and send them to Adafruit IO:

```
Adafruit IO Environmental Logger
BME Sensor is set up!
Found SGP30 serial #064F41A
Connecting to Adafruit IOAdafruitIO::connect()
.
Adafruit IO connected.
Reading Sensors...
Temperature = 82 *F
Pressure = 1006 hPa
Approx. Altitude = 56 m
Humidity = 43%
UV Light Level: 3
TVOC: 0 ppb eCO2: 400 ppm
```

Check your Dashboard on Adafruit IO and you should see your dashboard populated with values. You should also see them change every ten seconds.

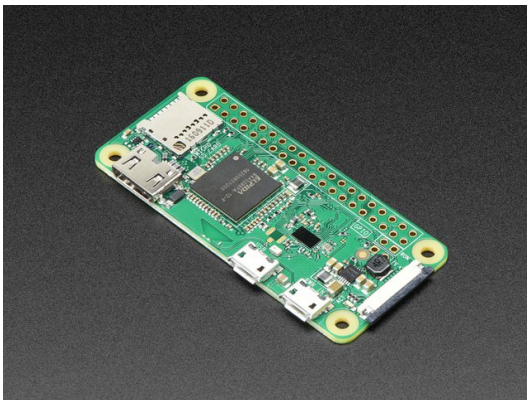


## Python Wiring and Assembly



We're going to make an environmental monitor using a **Raspberry Pi Zero W** and a **Perma Proto Bonnet**. We'll wire it up, build an enclosure for it, install dependencies, and program it with Python & [CircuitPython](https://adafruit.it/Bxq) (<https://adafruit.it/Bxq>)

The Pi Zero W has **built-in WiFi** - which is great for connecting our environmental monitor to Adafruit IO. It's also smaller than a regular Raspberry Pi 3, making it the perfect size to bring with you (monitor the air in the subway station) or stick it in a small corner of your room.



Raspberry Pi Zero W

\$10.00  
IN STOCK

[ADD TO CART](#)

---

We're going to be building our own pHAT for monitoring environmental data by using a Perma Proto Bonnet.



Adafruit Perma Proto Bonnet Mini Kit

\$4.50  
IN STOCK

ADD TO CART

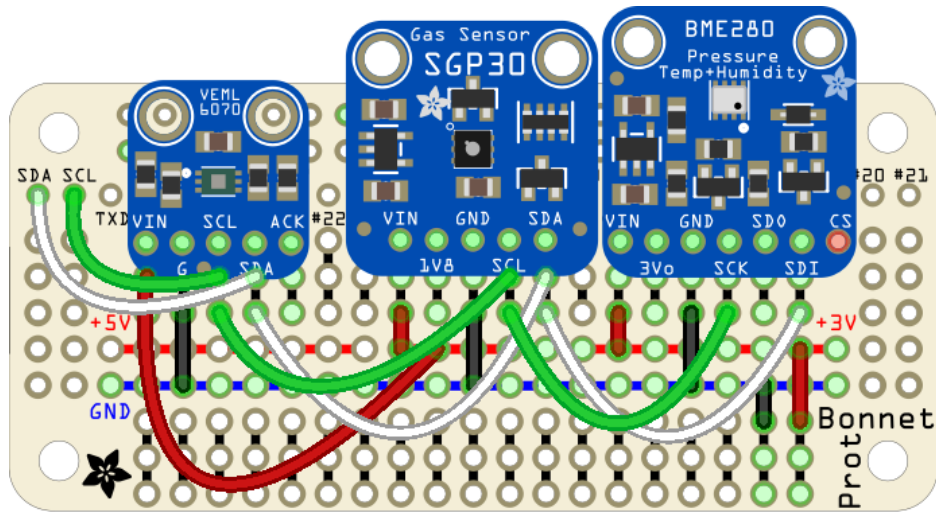
## Wiring

First, solder the included 20x2 GPIO Header to the Perma Proto Bonnet



We will be connecting multiple I2C devices to a single I2C controller using only two wires - SCL and SDA.

- *Interested in learning more about I2C Addressing?* [Check out our learn guide on connecting multiple I2C devices \(https://adafru.it/BK0\).](https://adafru.it/BK0)



**Note:** The SDA and SCL lines are located at the top- of the Proto-Bonnet and do not run down the side.

Make the following connections between the **Raspberry Pi** and the **VEML6070**:

- Pi 3V to VEML6070 Vin
- Pi GND to VEML6070 GND
- Pi SCL to VEML6070 SCK
- Pi SDI to VEML6070 SDI

Make the following connections between the **VEML6070** and the **SGP30**:

- VEML6070 SCL to SGP30 SCL
- VEML6070 SDA to SGP30 SDA
- Pi 3V to SGP30 Vin
- Pi GND to SGP30 GND

Make the following connections between the **SGP30** and the **BME280**:

- VEML 6070 SCL to BME280 SCK
- VEML 6070 SDA to BME280 SDI
- Pi 3V to VEML6070 Vin
- Pi GND to VEML6070 GND

## Assembly

The [small plastic project enclosure \(https://adafruit.it/C8T\)](https://adafruit.it/C8T) is the perfect size for a Pi Zero W and Environmental Logger Bonnett.



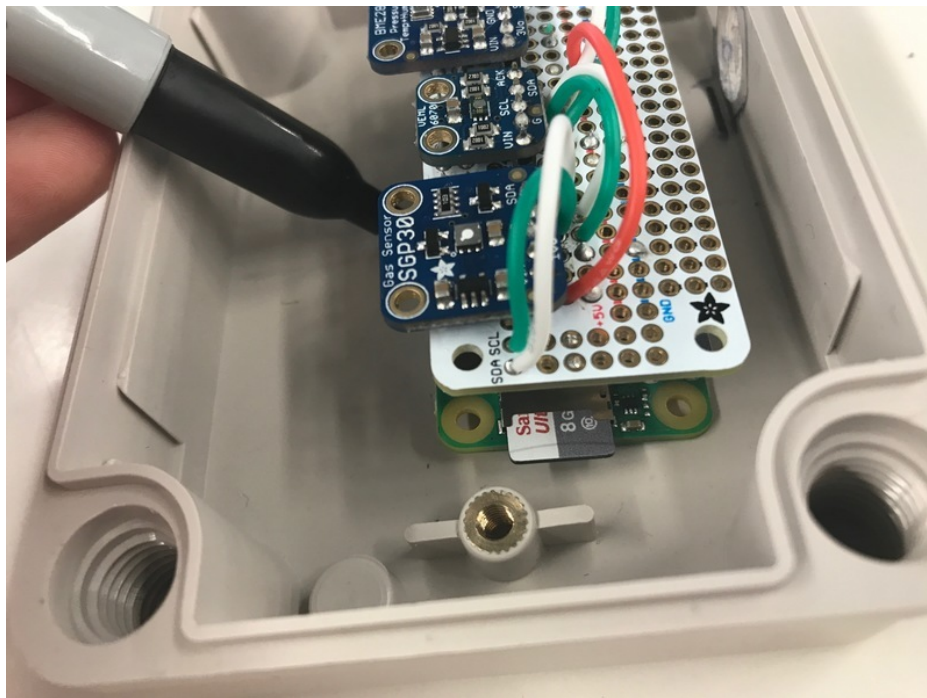


Small Plastic Project Enclosure - Weatherproof with Clear Top

\$9.95  
IN STOCK

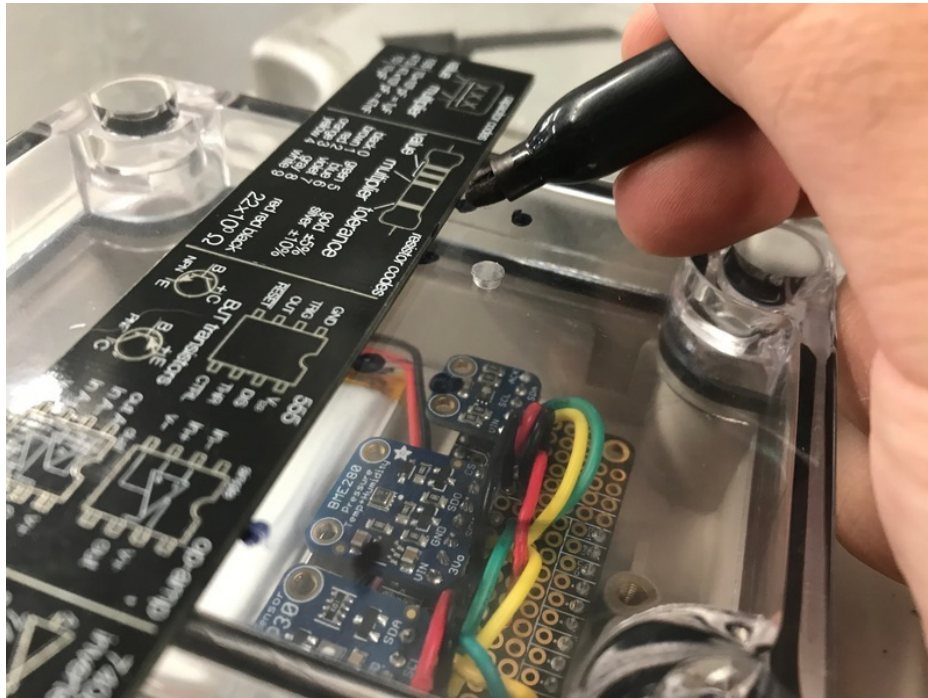
ADD TO CART

We're going to test-fit the Pi in the project enclosure. Use a non-permanent marker to outline the positions of the Pi on the bottom of the case.

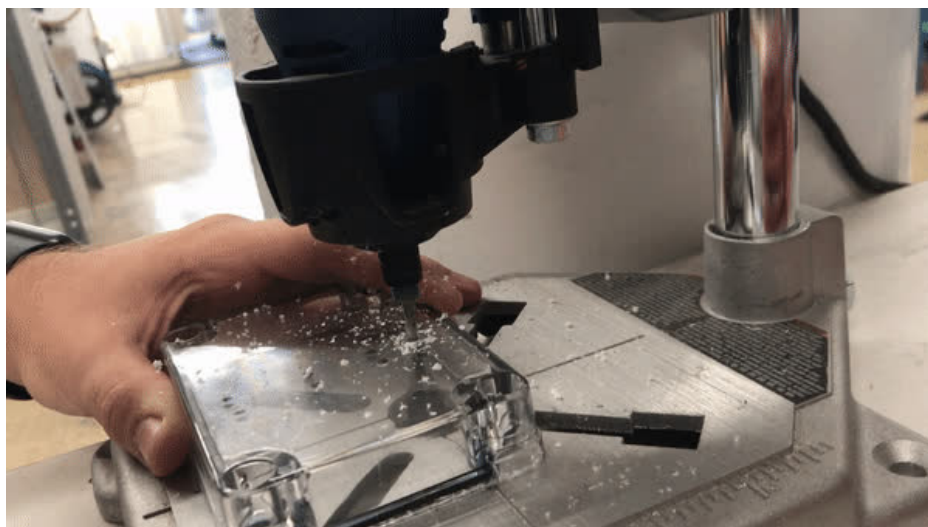


Put the lid on the box and screw it down to ensure everything fits correctly. Use a ruler and a marker to mark locations for holes on the plastic lid of your project box. Be sure to mark the locations closer to the sensors.





Let's drill some holes! I used a rotary tool with a 4mm drill bit mounted to a drill press. Be sure to hold the lid tightly, use a low RPM on your tool, and watch your fingers!



Next, let's set up our Raspberry Pi to be used with both CircuitPython and our sensors.

Next, we're going to attach a [cable gland \(https://adafruit.it/C8V\)](https://adafruit.it/C8V) to seal off the USB cable from the elements. Drill a 5/8" hole on the side of the enclosure where the Pi's USB port is located. The cable gland has integrated screw threads - twist it into the hole you drilled and attach the retaining nut.

You'll need to cut and splice the Micro USB cable to fit it into the enclosure through the cable gland. Plug a Micro-USB cable into the Pi Zero W.

## Making it Portable

If you want to take your Environmental Monitor on-the-go, you might want to add a Rechargeable USB Battery Pack.

We used a [smaller rechargeable model with a 2200mAh capacity \(https://adafru.it/e2q\)](https://adafru.it/e2q). Charge up your battery pack, and then plug it into the Pi's **PWR\_IN** port.

## Python Setup

If you're following along with a Raspberry Pi, Beaglebone or any other supported small linux computer, we'll use a special library called [adafruit\\_blinka](https://adafru.it/BJJ) (<https://adafru.it/BJJ>) (named after Blinka, the CircuitPython mascot (<https://adafru.it/BJT>)) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides. It's CircuitPython, on Pi!

If you haven't set up Blinka and the Adafruit IO Python Library yet on your Raspberry Pi, follow our guide:

- [Blinka and Adafruit IO Setup](https://adafru.it/BMB) (<https://adafru.it/BMB>)

## Enable I2C

We use two pins on the Pi (SDA/SCL) to communicate over I2c with the PCA9685. You only have to do this step *once* per Raspberry Pi, the I2C interface is disabled by default.

- [Enabling I2C](https://adafru.it/dEO) (<https://adafru.it/dEO>)

Once you're done with this and have rebooted, verify you have the SPI devices with the command:

```
sudo i2cdetect -y 1
```

The output from running this command should look like the following:

```
pi@io-pi:~ $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- 38 39 -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- 58 -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:  -- -- -- -- -- -- -- -- 77
```

## Installing required CircuitPython Libraries

You'll need libraries to communicate with the VEML6070, SGP30, and BME280 sensors. Since we're using Adafruit Blinka (CircuitPython), we can install CircuitPython libraries straight to our Raspberry Pi.

Run the following command from your terminal to install the [Adafruit\\_CircuitPython\\_VEML6070 Library](https://adafru.it/C6-) (<https://adafru.it/C6->):

```
sudo pip3 install adafruit-circuitpython-veml6070
```

Next, install the [Adafruit\\_CircuitPython\\_SGP30](https://adafru.it/Bn-) (<https://adafru.it/Bn->) library

```
sudo pip3 install adafruit-circuitpython-sgp30
```

Then, install the [Adafruit\\_CircuitPython\\_BME280](https://adafru.it/BfX) (<https://adafru.it/BfX>) library

```
sudo pip3 install adafruit-circuitpython-bme280
```

## Python Code

First, we're going to import the libraries used by our code to read sensor values, use CircuitPython on our Pi, and read/write to Adafruit IO using the REST client.

```
# Import standard python modules
import time

# import Adafruit Blinka
import board
import busio

# import sensor libraries
import adafruit_sgp30
import adafruit_veml6070
import adafruit_bme280

# import Adafruit IO REST client
from Adafruit_IO import Client, Feed, RequestError
```

Next, we'll set up our code with our Adafruit IO username and secret key. If you need these values, [navigate to your Adafruit IO profile \(https://adafru.it/BmD\)](https://adafru.it/BmD). We'll also create an instance of the REST client with these values.

```
# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Create an instance of the REST client
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
```

The code next assigns feeds, or creates and assigns them. Then, we create an I2C busio object and create objects for the VEML6070, BME280, and SGP30 sensors.

```
# Create busio I2C
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)
# Create VEML6070 object.
uv = adafruit_veml6070.VEML6070(i2c)
# Create BME280 object.
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
bme280.sea_level_pressure = 1013.25
# Create SGP30 object using I2C.
sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)
sgp30.iaq_init()
sgp30.set_iaq_baseline(0x8973, 0x8aae)
```

in the `while True` loop, we read data from the SGP30, VEML6070, and the BME280. The VEML6070 samples ten times before arriving at a final data.

```
# Read SGP30.
eCO2_data = sgp30.co2eq
tvoc_data = sgp30.tvoc

# Read VEML6070, sample ten times.
for j in range(10):
    uv_data = uv.read

# Read BME280.
temp_data = bme280.temperature
# convert temperature (C->F)
temp_data = int(temp_data) * 1.8 + 32
humid_data = bme280.humidity
pressure_data = bme280.pressure
alt_data = bme280.altitude
```

Then, we send our data to Adafruit IO. Since we'll be sending a lot of data to our feeds, we added a pause between sending each sensor's data to its corresponding Adafruit IO Feeds.

```
# Send SGP30 Data to Adafruit IO.
print('eCO2:', eCO2_data)
aio.send(eCO2_feed.key, eCO2_data)
print('tvoc:', tvoc_data)
aio.send(tvoc_feed.key, tvoc_data)
time.sleep(SENSOR_DELAY)
```

Finally, we wait for `LOOP_DELAY` minutes. If you want to report data more quickly or slowly, adjust the `LOOP_DELAY` variable at the start of the code.

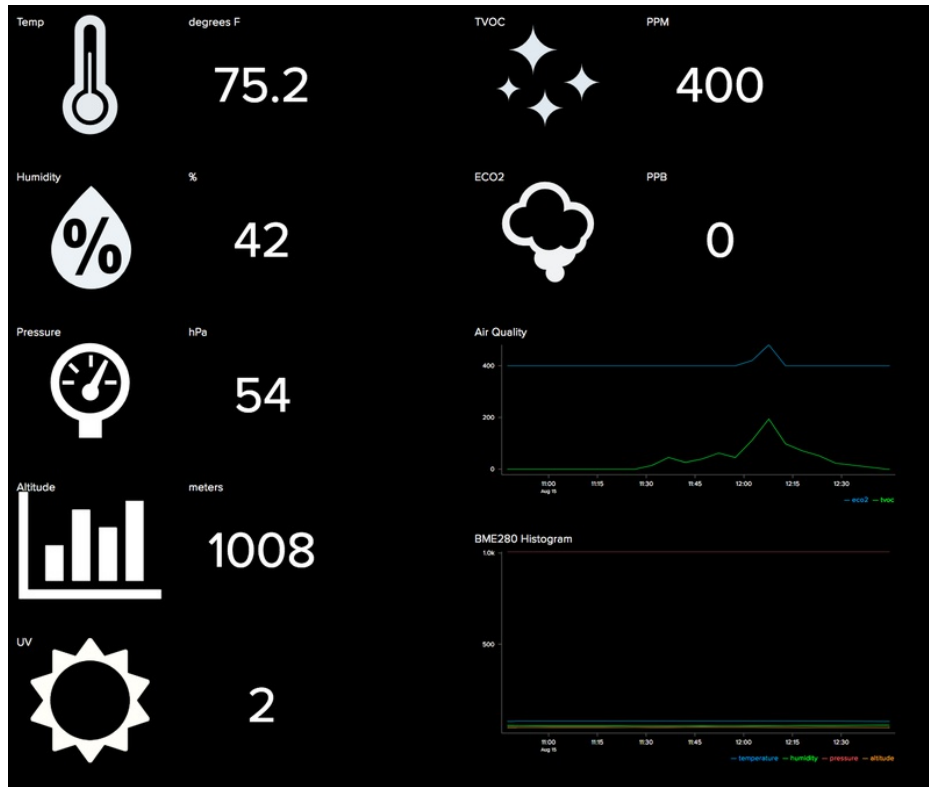
Make sure you have your Adafruit IO Username and Key set up and that your Pi is connected to the internet. Then, enter the following in your terminal to run the code:

```
python3 environmental_monitor.py
```

You should see the following output in your terminal:

```
Reading sensors...
sending data to adafruit io...
eCO2: 400
tvoc: 0
UV Level: 3
Temperature: 75.2 C
Humidity: 54.8 %
Pressure: 1008.1 hPa
Altitude = 42.68 meters
```

Your dashboard should populate with values. Note that it will take the SGP30 at least ten reads before it obtains a baseline measurement.



## Code

```
"""
'environmental_monitor.py'
=====
Example of sending I2C sensor data
from multiple sensors to Adafruit IO.

Tutorial Link: https://learn.adafruit.com/adafruit-io-air-quality-monitor

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Author(s): Brent Rubell for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

Dependencies:
- Adafruit_Blinka (CircuitPython, on Pi.)
  (https://github.com/adafruit/Adafruit_Blinka)
- Adafruit_CircuitPython_SGP30.
  (https://github.com/adafruit/Adafruit_CircuitPython_SGP30)
- Adafruit_CircuitPython_VEML6070.
  (https://github.com/adafruit/Adafruit_CircuitPython_VEML6070)
- Adafruit_CircuitPython_BME280.
  (https://github.com/adafruit/Adafruit_CircuitPython_BME280)
"""

# Import standard python modules
```



```

import time

# import Adafruit Blinka
import board
import busio

# import sensor libraries
import adafruit_sgp30
import adafruit_veml6070
import adafruit_bme280

# import Adafruit IO REST client
from Adafruit_IO import Client, Feed, RequestError

# loop timeout, in seconds.
LOOP_DELAY = 10

# Set to your Adafruit IO key.
# Remember, your key is a secret,
# so make sure not to publish it when you publish this code!
ADAFRUIT_IO_KEY = 'YOUR_AIO_KEY'

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'YOUR_AIO_USERNAME'

# Create an instance of the REST client
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

try: # if we already have the feeds, assign them.
    tvoc_feed = aio.feeds('tvoc')
    eco2_feed = aio.feeds('eco2')
    uv_feed = aio.feeds('uv')
    temperature_feed = aio.feeds('temperature')
    humidity_feed = aio.feeds('humidity')
    pressure_feed = aio.feeds('pressure')
    altitude_feed = aio.feeds('altitude')
except RequestError: # if we don't, create and assign them.
    tvoc_feed = aio.create_feed(Feed(name='tvoc'))
    eco2_feed = aio.create_feed(Feed(name='eco2'))
    uv_feed = aio.create_feed(Feed(name='uv'))
    temperature_feed = aio.create_feed(Feed(name='temperature'))
    humidity_feed = aio.create_feed(Feed(name='humidity'))
    pressure_feed = aio.create_feed(Feed(name='pressure'))
    altitude_feed = aio.create_feed(Feed(name='altitude'))

# Create busio I2C
i2c = busio.I2C(board.SCL, board.SDA, frequency=100000)
# Create VEML6070 object.
uv = adafruit_veml6070.VEML6070(i2c)
# Create BME280 object.
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
bme280.sea_level_pressure = 1013.25
# Create SGP30 object using I2C.
sgp30 = adafruit_sgp30.Adafruit_SGP30(i2c)
sgp30.iaq_init()
sgp30.set_iaq_baseline(0x8973, 0x8aae)

# Sample VEML6070
def sample VEML():

```

```

    for j in range(10):
        uv_raw = uv.read
    return uv_raw

while True:
    print('Reading sensors...')
    # Read SGP30.
    eCO2_data = sgp30.co2eq
    tvoc_data = sgp30.tvoc

    # Read VEML6070.
    uv_data = sample_VEML()

    # Read BME280.
    temp_data = bme280.temperature
    # convert temperature (C->F)
    temp_data = int(temp_data) * 1.8 + 32
    humid_data = bme280.humidity
    pressure_data = bme280.pressure
    alt_data = bme280.altitude

    print('sending data to adafruit io...')
    # Send SGP30 Data to Adafruit IO.
    print('eCO2:', eCO2_data)
    aio.send(eCO2_feed.key, eCO2_data)
    print('tvoc:', tvoc_data)
    aio.send(tvoc_feed.key, tvoc_data)
    time.sleep(2)
    # Send VEML6070 Data to Adafruit IO.
    print('UV Level: ', uv_data)
    aio.send(uv_feed.key, uv_data)
    time.sleep(2)
    # Send BME280 Data to Adafruit IO.
    print('Temperature: %0.1f C' % temp_data)
    aio.send(temperature_feed.key, temp_data)
    print("Humidity: %0.1f %" % humid_data)
    aio.send(humidity_feed.key, int(humid_data))
    time.sleep(2)
    print("Pressure: %0.1f hPa" % pressure_data)
    aio.send(pressure_feed.key, int(pressure_data))
    print("Altitude = %0.2f meters" % alt_data)
    aio.send(altitude_feed.key, int(alt_data))
    # avoid timeout from adafruit io
    time.sleep(LOOP_DELAY * 60)

```