# Table of Contents

**Introduction** 5  
- Hello! 5  

**What is Adafruit IO?** 5  
- Internet of Things, the Adafruit way 5  

**What Hardware or Transport to Use** 7  
- Hardware 7  
- Arduino 7  
- Python 9  

**How Adafruit IO Works** 10  
- How Adafruit IO Works 10  
- The Adafruit IO Back End 10  
- Accounts and the Adafruit IO Key 11  
- Adafruit IO Feeds 11  

**Creating a New Feed** 12  
- Step by Step 12  

**Sending Data From Your Thing** 16  
- Code 16  
- Linux + Python Example 16  
- Microcontroller + Arduino Example 18  

**Advanced Feeds** 22  
- Feed Options 22  

**WebHooks** 26  
- Communicating with Other Sites 26  

**Adafruit IO Front End--Dashboards** 29  
- User Interface with Feeds 29  
- Lines, and Toggles, and Buttons oh my! 30  

**Actions** 35  
- Scheduled vs. Reactive Actions 36  
- Scheduled Actions 36  
- Reactive Actions 37  

**Services** 38  
- IFTTT and Zapier 39  

**Adafruit IO Plus Integrations** 39  
- Integrations and Dark Skies 39  

**Who is Adafruit.IO for?** 41  
- Entry-Level 41  
- Makers 42  
- Engineers 42
IO Plus

- Upgrades

Conclusion

- The Future
Hello!

Hello, and welcome to episode 4 of Adafruit and Digi-Key’s “All the Internet of Things”! A six-part series that covers everything you need to know about IoT. In the previous episode, we discussed services -- services are what you use to route events to and from your IoT devices, coordinate real-time communication between multiple devices, and record, process and visualize the data that your “things” are producing. We also talked about some large cloud services such as Amazon AWS IoT, Google Cloud IoT, and Microsoft Azure IoT Suite.

You might find that although these services provide robust security and sophisticated deployment tools, getting started with using their comprehensive technology is intimidating.

If only there were some IoT service that proved easy to understand and implement...maybe with high quality documentation with examples, learning guides, and community support...and, maybe just maybe, an IoT service that provided opens source solutions with a low to no-cost point of entry...

Well, on today’s episode, we're happy to do a formal introduction to just that -- our very own Adafruit IO!

What is Adafruit IO?

Internet of Things, the Adafruit way

Here at Adafruit, we manufacture, support and sell all of these amazing sensors, LEDs, and robotics. So, naturally, we wanted a good way to interact with them over the Internet. We’ve covered a bunch of great services for datalogging and communicating with your controller over the web, but we wanted a service that fit the needs of the Prototyping & Maker communities AND was designed for engineers of all skill levels. So we decided to build our own system - that is how Adafruit IO got started.
Like many of the services we've introduced, we maintain both MQTT (https://adafru.it/EfZ) and REST (https://adafru.it/uff) APIs, which is how you will be communicating with Adafruit IO over the Internet. But you don’t have to be an expert programmer! We’ve built robust (https://adafru.it/Ef-) client (https://adafru.it/Eg0) libraries (https://adafru.it/Ean) with lots of examples (https://adafru.it/Eg1), so you can probably start with some ready-to-go code.

Once you’ve got a device connected, you can control and monitor using configurable dashboards. The web dashboards come with a dozen widgets that allow for easy 2-way interaction with your devices. You’ll get buttons, gauges, maps, sliders.

Outside of the dashboard, you can create triggers to, say, email you when the water sensor value goes above 9000.

Adafruit IO (https://adafru.it/Eg2) is available and free for anyone with an Adafruit account. For the vast number of people starting out with IoT, the basic account will do everything they need. Adafruit IO plus (https://adafru.it/Eg3) members get some nifty extras and more bandwidth.

This video and guide will go over all things Adafruit IO including how to get started and use Adafruit IO to make the most out of your projects.

Now that you know what you can expect when using Adafruit IO, let’s focus on how to get started!

For more information on Adafruit IO please check out this learn guide! (https://adafru.it/BRB)
Or if you'd like more information on REST and MQTT we have a learn guide (https://adafru.it/Bor) for that too!

What Hardware or Transport to Use

Hardware

Some services lock you into using just their hardware or transport (https://adafru.it/ChW) - Adafruit IO is agnostic about what it is that's connecting. In theory, anything that can send messages over the Internet will work. But we do have some recommendations.

Arduino

The Adafruit Feather line of microcontrollers (https://adafru.it/Eg4) are low cost, have built-in networking and battery charging, and are easy to program with our Arduino libraries (https://adafru.it/fpd).

If you are planning to use WiFi, the Adafruit Feather HUZZAH with ESP8266 (https://adafru.it/n6A) is a common choice for ultra-low cost, but has few GPIO.

Adafruit Feather HUZZAH with ESP8266 (https://adafru.it/n6A) is our take on an 'all-in-one' ESP8266 WiFi development board with built in USB and battery charging. Its an ESP8266 WiFi module with all the extras you need, ready to rock!
For more power but still keeping the built-in WiFi, the Adafruit HUZZAH ESP32 (https://adafruit.it/wcN) is an upgrade with more pins and a beefier chip.

The Adafruit HUZZAH32 is our ESP32-based Feather (https://adafruit.it/wcN), made with the official WROOM32 module. We packed everything you love about Feathers: built in USB-to-Serial converter, automatic bootloader reset, Lithium Ion/Polymer charger, and just about all of the GPIOs brought out so you can use it with any of our Feather Wings.

The Adafruit Feather M0 WiFi w/ ATWINC1500 (https://adafruit.it/n6f) board gives you SAMD21 ARM Cortex with a separate WiFi module, good if you need native USB, or have existing ARM code to run and don’t want to use a different core chipset. These would be programmed in C.
And of course, you can also use Bluetooth Low Energy and tether using our Bluefruit App (https://adafru.it/Eg5). For that, we recommend our nRF52832 (https://adafru.it/vAx) or nRF52840 (https://adafru.it/DLQ) Bluefruit Feathers, and you'll also need an iOS or Android phone that will stay on all the time to perform the data transfer.

Python

If battery life and portability isn’t as important, Raspberry Pi (https://adafru.it/Eg6) computers and other single-board computers are also a great choice. These tend to have Ethernet as well as WiFi built in. The Linux operating system will take care of all of the networking for you, so it’s incredibly easy to get started, and you can use Python (https://adafru.it/Eg0) which is higher level and easier to use than C.

The Raspberry Pi 3 Model B+ (https://adafru.it/Bi1) boasts a 64-bit quad core processor, 5GHz wireless LAN and faster (300mbps) Ethernet. If you'd like to have a beefier hardware platform to use with Adafruit IO and possibly some more intensive libraries (such as OpenCV) - this is the ideal hardware platform for your project.
The Raspberry Pi Zero W ([https://adafruit.it/vMD](https://adafruit.it/vMD)) offers all the benefits of the full-sized Pi and built-in WiFi. This board is perfect for Adafruit IO projects which require a small form factor.

For more information on using Adafruit IO with our Feather Huzzah ESP8266, please check out this learn guide ([https://adafruit.it/ufK](https://adafruit.it/ufK))!

How Adafruit IO Works

There's two halves to every service online - the Back End, where data is handled, and the Front End, where you use a web browser to interact. For most Internet services you use on a day-to-day basis, you don’t deal with the Back End at all - it doesn’t matter how that restaurant got all their dishes listed on the website, as long as you can order them for delivery.

On Adafruit IO, you'll be dealing with both halves.

The Adafruit IO Back End

The back end is where you'll be sending your sensor data to and from our server, it's literally named **Adafruit IO** ([https://adafruit.it/eIC](https://adafruit.it/eIC)) (very easy to remember). You can do send/receive data in two ways, with REST or MQTT. Which one you use will depend a lot on your hardware, bandwidth, and language needs. We covered these extensively in previous videos so go watch those if you haven’t yet. We recommend REST for your first project, it's easier to understand and debug.

Either way, once you’ve picked your hardware and transport, check how you want to program it and choose one of our provided libraries. We’ve got code and examples for **Arduino C/C++** ([https://adafruit.it/fpd](https://adafruit.it/fpd)), **Python** ([https://adafruit.it/DOJ](https://adafruit.it/DOJ)), **node.js** ([https://adafruit.it/Eg7](https://adafruit.it/Eg7)) and even **Golang** ([https://adafruit.it/Bvt](https://adafruit.it/Bvt))!
Accounts and the Adafruit IO Key

Start by registering online for an account at Adafruit (https://adafruit.it/dAQ).

Once you’re logged in, you can retrieve your API key.

Click View AIO Key on the sidebar, then copy the Active Key to somewhere safe - we’ll be using it later.

This key is important, it unlocks access to Adafruit IO. You’ll be programming it and your username into every one of your projects. If your key is lost or compromised, you can always generate a new key, but you’ll have to update all your existing projects. Don’t share your key with others!

Adafruit IO Feeds

Now let’s check out the way we store data in Adafruit IO - after all that’s what we’re here for! Data on IO is kept in time-series databases called feeds. Each feed contains time-stamped data points. The data points don’t have to be numbers, they can be any type of data.

For example, you can store numbers - say for environmental data like temperature or humidity.

Numbers can also be used for controlling hardware, like a robot’s motor throttle.
Human-readable ASCII strings are also common, often displayed either on a dashboard or on the device's display. Finally, binary data can be saved as well - we strongly recommend encoding it in base64 so you can store/retrieve it with REST and JSON. For example, JPG encoded photos could be base64 encoded, then uploaded. Or if you have a firmware OTA update, you could store it as ASCII Intel HEX or base64 encoded data.

In addition to the main storage data chunk, you can also attach metadata - the first and most obvious metadata is the Time that the data was entered into Adafruit IO - this is handled for you when you post data to the feed. The second is Location, where you can set where on Earth the data came from.

That metadata is something you have to add yourself, we don’t provide a geotagging service at this time, but if you have GPS module or use a geo-IP service, you can add it on every post. Location addition is only possible via the REST API, you can see the JSON formatting for adding it here, (https://adafruit.io/uff)

Creating a New Feed

Step by Step

Let’s create a new feed to demonstrate how this all works.
Log into your Adafruit IO account and visit the Feeds page (https://adafruit.io/ mxC).

Now in Actions select Create a New Feed - we’ll call this one Temperature.

Voila, a new feed has been created!

Next, to see what happens if incorrect data is added. This time, when we add data - let’s put in a string instead of a number.

Now to test it out by manually adding a value. Click Add Data and type in the current temperature. Once you've done that, you'll see the feed automatically update with the timestamp metadata.
From the temperature feed, click Add Data and type in the current temperature. Once you’ve done that you’ll see the feed automatically update with the timestamp metadata.

Next, let’s see what happens if we add incorrect data. This time, when we Add lets put in a string instead of a number. You’ll notice that Adafruit IO doesn’t stop you from adding it, it does not do any type checking!

That’s something you can use if it’s helpful, or make sure your code filter out.
We can remove data points, such as this incorrect one, quite easily by selecting it and then clicking Remove Selected Data.

While I'm here, you ought to know, you can always Download All Data which will fetch all the data in the feed as a structured JSON or CSV text file.
For more information on using feeds in Adafruit IO, check out this Learn Guide (https://adafru.it/ioA).

Sending Data From Your Thing

Code

Now that we've got the feed set up, we'll want to automatically send data from our "thing". Let's show how to do that with both a Raspberry Pi and Python as well as a Feather ESP8266 and C++. For our sensor, we'll use the ADT7410 (https://adafru.it/EgN) from Analog Devices, which is a high precision temperature sensor with an I2C interface.

Linux + Python Example

Click here for info on how to build this project in detail!

https://adafru.it/Eg8

Let's start with a Raspberry Pi with Python. This assumes we've already set up the Pi's operating system and WiFi. We wire up the sensor with I2C and run the Python script on this page (https://adafru.it/Eg9).

---

'adafruit_io_adt7410.py'

Example of sending temperature values to an Adafruit IO feed using an ADT7410 breakout.

Dependencies:
- Adafruit_Blinka (https://github.com/adafruit/Adafruit_Blinka)
- Adafruit_CircuitPython_SSD1306

# SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Example of sending temperature values to an Adafruit IO feed using an ADT7410 breakout.

import adafruit_adt7410

# Initialize the ADC

# Adjust the gain of the ADC

# Initialize the temperature sensor

# Set the temperature to Celsius

# Get the current temperature

# Send the temperature to Adafruit IO

# Print the temperature

# Close the temperature sensor

# The example code above sends the temperature values to Adafruit IO feed using an ADT7410 breakout.
# Import standard python modules
import time

# import Adafruit SSD1306 OLED
from PIL import Image, ImageDraw, ImageFont
import adafruit_ssd1306

# import Adafruit IO REST client
from Adafruit_IO import Client

# import Adafruit CircuitPython adt7410 library
import adafruit_adt7410

# import Adafruit Blinka
import board
import busio
import digitalio

# Delay between sensor reads, in seconds
DELAY_SECONDS = 30

# 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 = 'ADAFRUIT_IO_KEY'

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

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

# Set up `temperature` feed
pi_temperature = aio.feeds('temperature')

# Set up OLED
i2c_bus = busio.I2C(board.SCL, board.SDA)
oled_reset = digitalio.DigitalInOut(board.D21)
disp = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c_bus, reset=oled_reset)
# Clear display.
disp.fill(0)
disp.show()

# Create blank image for drawing.
# Make sure to create image with mode '1' for 1-bit color.
width = disp.width
height = disp.height
image = Image.new('1', (width, height))
# Get drawing object to draw on image.
draw = ImageDraw.Draw(image)
# `sudo apt-get install ttf-mscorefonts-installer` to get these fonts
font_big = ImageFont.truetype('/usr/share/fonts/truetype/msttcorefonts/Arial_Bold.ttf', 16)
font_small = ImageFont.truetype('/usr/share/fonts/truetype/msttcorefonts/Arial_Bold.ttf', 12)

adt = adafruit_adt7410.ADT7410(i2c_bus, address=0x48)
adt.high_resolution = True
time.sleep(0.25)  # wait for sensor to boot up and get first reading

while True:
    # clear screen
    draw.rectangle((0, 0, width, height), outline=0, fill=0)
# Read the temperature sensor
```
tempC = adt.temperature
```
```
tempC = round(tempC, 2)
```

# display the temperature on the OLED
```
draw.text((0, 0), "Temp: %0.2F °C" % tempC, font=font_big, fill=255)
```

# Send temperature to Adafruit IO
```
print('Sending temperature {0} °C to Adafruit IO'.format(tempC))
draw.text((0, 16), "Sending...", font=font_small, fill=255)
disp.image(image)
disp.show()
```
```
aio.send(pi_temperature.key, tempC)
draw.text((0, 16), "Sending...Done!", font=font_small, fill=255)
disp.image(image)
disp.show()
```
```
# Delay for DELAY SECONDS seconds to avoid timeout from adafruit io
time.sleep(DELAY SECONDS)
```

As you can see, this is very simple and straightforward - we only have to enter in our Username and AIO Key. The code will connect to the feed, initialize the sensor, and then execute a loop, sending the temperature every 30 seconds.

Meanwhile, over at the Adafruit IO website, we can monitor what data is coming in. This is very handy if you’re having an issue - you can see the connected IP addresses and what feeds they are writing to. You can also see errors.

Perhaps a typo snuck into your code, it may appear here!

### Microcontroller + Arduino Example

Click here to follow along and build this project in detail!

https://adafruit.it/Egb
Likewise, here's an example using the ESP8266 Feather and an ADT7410 wired up on a breadboard. With the ESP Feather, we're stuck with a microcontroller rather than a full computer, so we don't use Python, instead we'll stick to "Arduino" C++. The tradeoff is that we have much lower power usage, and faster boot, but we also have to be more careful with memory and link management. Check out the Arduino code on this guide (https://adafru.it/Egb).

```cpp
// SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
// SPDX-License-Identifier: MIT

// Adafruit IO ADT7410 Example
// Adafruit invests time and resources providing this open source code.
// Please support Adafruit and open source hardware by purchasing
// products from Adafruit!
// Written by Ladyada and Brent Rubell for Adafruit Industries
// Copyright (c) 2019 Adafruit Industries
// Licensed under the MIT license.
// All text above must be included in any redistribution.

//******************** Adafruit IO Config ********************

// visit io.adafruit.com if you need to create an account,
// or if you need your Adafruit IO key.
#define IO_USERNAME "YOUR_IO_USERNAME"
#define IO_KEY "YOUR_IO_KEY"

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

// the AdafruitIO WiFi client will work with the following boards:
// - Feather M0 WiFi -> https://www.adafruit.com/products/3010
// - Feather WICED -> https://www.adafruit.com/products/3056
#define WIFI_SSID "WIFI_NAME"
#define WIFI_PASS "WIFI_PASS"

// 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);

//******************** Configuration ********************

// time between sending data to adafruit io, in minutes
#define MESSAGE_WAIT_SEC (15 * 60)

//******************** Example Starts Here ********************
```
// adt7410 sensor
#include "Adafruit_ADT7410.h"
// featherwing oled
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <SPI.h>

// Create the OLED display object
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, &Wire);

// Create the ADT7410 temperature sensor object
Adafruit_ADT7410 tempsensor = Adafruit_ADT7410();

// set up the 'temperature' feed
AdafruitIO_Feed *huzzah_temperature = io.feed("temperature");

void setup()
{
    // start the serial connection
    Serial.begin(115200);

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

    Serial.println("Adafruit IO ADT7410 + OLED");
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // Address 0x3C for 128x32
    Serial.println("OLED begun");

    // Show image buffer on the display hardware.
    // Since the buffer is intialized with an Adafruit splashscreen
    // internally, this will display the splashscreen.
    display.display();
    delay(1000);

    // Clear the buffer.
    display.clearDisplay();
    display.display();

    // Make sure the sensor is found, you can also pass in a different i2c
    // address with tempsensor.begin(0x49) for example
    if (!tempsensor.begin())
    {
        Serial.println("Couldn't find ADT7410!");
        while (1)
        {
        }
    }

    // sensor takes 250 ms to get first readings
    delay(250);

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

    display.clearDisplay();
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 0);
    display.print("Connecting to IO...");
    display.display();

    // wait for a connection
    while (io.status() < AIO_CONNECTED)
    {
        Serial.print(".");
        delay(500);
void loop()
{
    // io.run(); is required for all sketches.
    // it should always be present at the top of your loop
    // function. it keeps the client connected to
    // io.adafruit.com, and processes any incoming data.
    io.run();

    // Read and print out the temperature, then convert to °F
    float c = tempsensor.readTempC();
    Serial.print("Temp: ");
    Serial.print(c);
    Serial.println("C");

    // clear the display
    display.clearDisplay();
    display.display();

    // print to display
    display.setTextColor(WHITE);
    display.setTextSize(2);
    display.setCursor(0, 0);
    display.print("Temp:");
    display.print(c);

    // send data to adafruit io
    display.setCursor(15, 20);
    display.setTextSize(1);
    display.print("Sending...");
    display.display();
    Serial.println("Sending to Adafruit IO");
    delay(1000);
    huzzah_temperature->save(c, 0, 0, 0, 2);

    // sent to IO
    display.clearDisplay();
    display.display();
    display.setTextSize(2);
    display.setCursor(0, 0);
    display.print("Temp:");
    display.print(c);
    display.setTextSize(1);
    display.setCursor(15, 20);
    display.print("Sending...Done!");
    display.display();

    Serial.print("Waiting ");
    Serial.print(MESSAGE_WAIT_SEC);
    Serial.println(" seconds");
    // wait 15 minutes between sends
    for (int i = 0; i < MESSAGE_WAIT_SEC; i++)
    {
        delay(1000);
    }
}
Like the Python code, it's pretty simple - we need our Username and AIO Key. We also need the WiFi credentials since there's no operating system to do that for us. Like the Pi script, we connect, initialize the sensor and send temperature data every 10 seconds.

And at Adafruit IO, we have the same monitored data:

![Live Data](image)

Once we're sure that the monitor is properly indicating what data is coming in, we can check our feeds. Aha! We've got our temperature data feeds from both the Pi and Feather.

For more information on the basics of using Adafruit IO with temperature and humidity sensors, check out this learn guide (https://adafruit.it/uFD)!

Advanced Feeds

Feed Options

Now that your feed is running, let's see what else we can do. Adafruit is always adding new ways to use and customize your feeds. When you select the feed, you'll have a few options running down the right side of the screen.
First up is the Feed Info - which is sort of what you expect, you can change the human-readable name, the feed name, as well as add a description. We also have a URL reference for when you program in your devices.
Next down is Privacy - you can make feeds public or private. Public feeds aren’t writable by others, but they can see the data that goes into the feed, even if they don’t have an Adafruit IO account!

They can view the feed webpage using any browser, or fetch the JSON data over REST from any computer. MQTT usage will require an account. Private feeds are only visible to you, and you need to be logged in to Adafruit IO.

You might want more advanced sharing, since Privacy is a bit extreme - either everyone can see it or no-one can.

Sharing lets you share read or read-write privileges with other Adafruit IO account holders. They’ll need to have an account of course, and the feed is still owned by your account, but this means you don’t have to share API Keys with others (which you shouldn’t do!)

Feed History allows you to turn on or off data storage per feed. By default the History setting is on, which allows you to track the history of a data feed.

You may want to turn this off, especially if you have a feed that has a big data payload, like binary data, and you don’t need previous values - remember that Free accounts on Adafruit IO have a limit of 30 days of history and a limit on the total storage amount.
Next up is Notifications. This is a pretty useful feature that is missing from many services - it is common to have a way to forward messages from your feeds, but often you want the opposite - a way to know if your feed hasn’t received messages. If you’ve programmed computers or microcontrollers, this is sort of the equivalent of a watchdog timer or a supervisory circuit.

Pick a time range that is at least two times larger than your expected data interval - from 10 minutes to 1 week, and you’ll get an email to your Adafruit IO account to let you know that the feed hasn’t been updated and you should check it out.

We’ll skip Webhooks for now, and go to Disable Feed. Free accounts have a limit on the number of feeds, so after you’ve used a feed, and you’re done with it, you may want to disable it. This keeps the data available for download, but doesn’t quite delete it. Once disabled, you can’t re-enable it!

Last on the list is Licensing, which is useful if you plan on making your feeds public. You may want to set a Creative Commons license so people know your expectations for what they can do with your data. For more information about Creative Commons license types, visit [https://creativecommons.org/](https://creativecommons.org/) (https://adafruit.it/eZc).
WebHooks

Communicating with Other Sites

OK, finally lets zoom back to WebHooks. As we mentioned in earlier videos, nearly every website and service on the internet has ways to communicate with other sites. While REST is common, you can only ‘pull’ data over REST, which makes it high-traffic when you’re constantly polling for new data. Webhooks are how REST-API supporting sites support data ‘pushing’. For example, say you wanted to get a user’s latest Twitter message. Instead of constantly connecting to the twitter API every minute to check if a new message has been posted, you can ask Twitter to update a webhook URL on each post. That means Twitter will contact you when there’s new data. But, as you can imagine, you need a webserver to listen for that posting. In this case, Adafruit IO can act as that webhook destination.

Adafruit IO only supports receiving data at this time, we need a service that will publish data into a feed.
Once you have the raw webhook, we need to publish data to it. Once data is published to the raw webhook, it'll appear on the Adafruit IO Feed.

One of the big benefits of webhooks is when you have services that don’t update that often, but you want quick updates when they do. Also, most web services require an API key or authentication - which can be annoying to manage. Webhooks are often ‘free’ with services - so they can easily provide that glue to let you integrate Adafruit IO with other sites with little fuss.

Now, click WebHook to create a URL for data to feed into. Now, if you happen to have control over the webhook publication data to get it into the right format, you can get the data to publish values directly to a feed. That’s fairly complex though. It’s easier to have the webhook be a raw-data destination, where we’ll get the full JSON publication. Once you have the raw webhook, we need to publish data to it.
Lets use github.com, which has extensive webhook support. In your account, go to Settings and then Webhook. Lets set up a webhook that will publish to our URL, make sure its posting json data, and we can customize what will trigger the webhook. In this case, let’s have it go off when someone stars one of the Adafruit repositories. Save it and check the feed, you’ll see a new message arrived, that’s their way of testing. Now we can star some repositories, and see new entries come in immediately - way faster than trying to poll github.

One of the big benefits of webhooks is when you have services that don’t update that often, but you want quick updates when they do. Also, most web services require an API key or authentication - which can be annoying to manage. Webhooks are often ‘free’ with services - so they can easily provide that glue to let you integrate Adafruit IO with other sites with little fuss.
Adafruit IO Front End--Dashboards

User Interface with Feeds

Now that we’ve got data streaming into feeds on Adafruit IO, and learned of all the cool tricks you can do with your feeds, we can now view and analyze the projects data in real-time. If you want to interface with the feed data, you can always query and fetch data on any computer using the REST or MQTT API. There’s plenty of libraries for a computer programming language to do so.

Ultimately, almost every Internet of Things project will, at some level, need to provide a way for the things to deal with the humans. User interface considerations can range from a mobile app that allows a user to turn lights on and off and adjust the volume on a music player, to output features such as status monitors, analytics, and data visualization dashboards.

However, making a custom app with a user interface can be a ton of work. You can create a quick and simple UI using the built in dashboard capability in Adafruit IO. Of course, you should start with making all your feeds and getting them working well.

Once that’s done, you can visit the Dashboard (https://adafruit.io/eIS) section of the site. Free accounts can have up to 5 Dashboards, and they can have as many elements as you like.

One thing to note about elements is that some of them pull from feed data, some push to your feed, and some do both. We’ll explain as we go through them. Note that since all the elements read from the underlying feeds, they can affect each other, which can be great for adding quick visual feedback!

From your account, click Dashboards and create a new dashboard. A dashboard is just a webpage you can edit. Click Plus to add a block. There are a wide range of great looking, functional blocks.

Lines, and Toggles, and Buttons oh my!

OK, of course we’ve got that temperature data feed created earlier - let’s display that data. We’ve got two options for reading values - we can put down a line chart block. This will plot the data much like the feed information page. That’s good for historical trend data. For instantaneous readings, with a nicer look than the plain text block, try the gauge block. Both of these are read-only blocks.
You can click edit to customize your blocks. For example, we can set the min/max value for the gauge, note that the preview on the right will update immediately. Each block has various customizations that you can be adjusted. Try the Test Value area to see what different values will look like! Click update when you're done.

Now that we’ve seen a few reading blocks, check out the momentary push button. This is sort of what it sounds like, its a button graphical element. When you click the button with your mouse or finger on touchscreen, it will send a message, let’s say “ON” to the attached feed. When released, it sends another message, let’s say “OFF”. It will only write data to a feed, not read data. You can change what data is written in the configuration page.
With non-numerical data, the plot and gauge won’t work. The text element will show the last value written to a feed. Let’s add one. We can edit the dashboard to place it right next to the button. Now note that one element is writing to a feed, and one is reading. So when I click the push button, the text updates immediately to match!

If you need more than just the latest value, the Stream block is sort of like a small feed monitor, it will display date-stamped feed information. We always recommend keeping one of these on each dashboard so you can easily monitor your feed data.

The ON/OFF switch is much like the button and is one of the few elements that’s both read and write.

Let’s add one so we can see how that works. Like the push button, it will send a message to a feed when pressed, and will alternate between ON and OFF (or whatever text you like).
Click the switch and you’ll see the feed/text block update. Unlike the button, it also reads from the feed. So if you press the momentary button...aha! The switch also updates itself.

There’s a few more elements we won’t go into detail - and there may be even more Adafruit has added since recording this video.

Map uses Open Street Map data to plot your location. You can use this if you have the geolocation metadata set, as mentioned earlier.
If you need a lot of buttons, you can use either the numberpad or the remote control elements. These give you either 12 or 21 momentary buttons, and will write a single number to a feed.

The Image element is good if you have base64-encoded images in a feed, it will decode JPG files automatically, great if you’re making a remotely viewable camera feed.
The Color picker will read or write data from a feed, you can select a color and it will write the corresponding web hex color to the feed, or it will display a color if the feed contains a valid hex color.

Status indicator is a very simple circle that will change colors based on a numeric condition. It's even simpler than the gauge block, good when you just want to know if the temperature has fallen below a certain value.

Finally, the Icon block will decode text within the feed value and display a corresponding icon, sort of an advanced status indicator with a wide range of shapes. You can't change the color via the feed, just the displayed icon.

For more information about dashboards and all the cool things they can do, check out this learn guide (https://adafruit.it/f5m)!

Actions

Actions were previously called Triggers. Screenshots may show trigger terminology yet.
Scheduled vs. Reactive Actions

While you can build just about any kind of IoT device or project with the feeds and dashboards we’ve demonstrated, we can add a little more logic and integration using actions and services.

Actions are a way to do something when a certain situation occurs. There’s two kinds of actions supported at this time: Scheduled and Reactive.

Scheduled Actions

Scheduled actions are the simplest. They’re a little like the Feed Notifications we mentioned earlier. You can simply have a feed value emailed to yourself on a set schedule. For example, here’s an example email from a scheduled action for my temperature feed.
Reactive Actions

Reactive Actions are much more advanced, and can integrate basic logic. At a basic level, they check if a feed value is somehow comparable to a value or to even another feed. If so, you’ll be able to send an email notification, post a webhook or even publish a message to another feed. For example, you can set up a reactive trigger to notify you when the temperature falls below a certain value. When paired with Feed Notification you can easily keep track of the health of your feed and also when something’s gone wrong.
Services

Notifications and triggers are handy for keeping tabs on your feeds, and the webhooks capabilities give you a lot of integration options. But, you may want to integrate into more services besides emails, and sometimes the webhook data is not formatted in the exact right way. Service integration in Adafruit IO let's you supercharge your feeds.
IFTTT and Zapier

Both IFTTT (https://adafru.it/fYY) and Zapier (https://adafru.it/Ehn) are ‘cross-linking’ services that are popular and wide ranging. They basically take data and shuffle it between services. We have integration with both of these services, and best of all, they’re free! Once you sign up, you can create triggers, much like the ones that are built into Adafruit IO but with much more control and possibilities.

For hands on experience with IFTTT check out this learn guide (https://adafru.it/Ehx) that will teach you to make a small mailbox to sit on your desk and notify you about incoming emails. When a new email is received, it raises a flag and gently lowers it back down.

Or you can check out this learn guide (https://adafru.it/jdq) that will show you how you can use an Adafruit HUZZAH ESP8266 WiFi microcontroller board with a door sensor to email/tweet/text you when your door has opened!

Zapier (https://adafru.it/Ehn) is similar to IFTTT, so we won’t demo both - they do have some differences in integration capabilities, so we recommend signing up for both and seeing which one gives you the connection you’re looking for. One thing to note is that both IFTTT and Zapier have some delay to them, you wont have messages come and go instantly the way webhooks do. Sometimes there can be up to 15 minutes delay, or even a dropped message could occur. You’ll want to do plenty of experimentation to figure out whether it makes sense to use Adafruit IO Triggers, notifications, webhooks or Zapier/IFTTT.

For more information on Zapier feel free to check out this learn guide (https://adafru.it/Ehy) that goes more in depth.

And if you'd like to try a project with Zapier we have a wonderful learn guide (https://adafru.it/Ehz) for making Adafruit IO Time tracking Cube!

Adafruit IO Plus Integrations

Integrations and Dark Skies

Some integrations are available for Adafruit IO Plus members only. For example, we have a Weather service that hooks into Dark Sky (https://adafru.it/Eho), a premier forecasting service. Once you start building IoT projects, you’ll notice that getting
weather data on demand isn’t easy - almost every service requires authentication and most require payment. With our Dark Sky integration, it’s quite easy to get weather data whenever you like, which you can use to make projects that react to current conditions in the local area, or really anywhere you like.

Once you have upgraded to Adafruit IO Plus (https://adafruit.it/Eg3), add a location to your service. Say, Thief River Falls, Minnesota. Now you can query the forecast or current weather for that location. For example, here’s some Python code that will get the current weather condition and forecast, and print it out. When I run it, we see...it’s crisp and cool this winter afternoon.

```python
# Import JSON for forecast parsing
import json
# Import Adafruit IO REST client.
from Adafruit_IO import Client, Feed, RequestError

# Set to your Adafruit IO key.
ADAFRUIT_IO_USERNAME = 'YOUR_IO_USERNAME'
ADAFRUIT_IO_KEY = 'YOUR_IO_PASSWORD'

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

# Grab the weather JSON
weather = aio.receive_weather(1234)
weather = json.dumps(weather)
forecast = json.loads(weather)

# Parse the current forecast
current = forecast['current']
print('Current Forecast')
print('It is {0} and {1}.'.format(current['summary'], current['temperature']))

# Parse the two day forecast
forecast_days_2 = forecast['forecast_days_2']
print('Weather in Two Days')
print('It will be {0} with a high of {1}F and a low of {2}F.'.format(
```

©Adafruit Industries
We’ve gone over just about all of the amazing features that Adafruit IO provides, and Adafruit is constantly adding more. In fact, there are probably more services and features that have been added since. You can always get updates on the latest news by checking out the Adafruit blog (https://adafruit.it/Ehp), and the most recent documentation, available here (https://adafruit.it/uff)! 

Who is Adafruit.IO for?

Whew! You should now have a good sense of what you can do with Adafruit IO - how feeds, dashboards and services all work together to get your data stored, plotted and interacting. Compared to many IoT services, there’s a lot you can do! So who is Adafruit IO for?

The short answer is, everyone!

Entry-Level

Adafruit IO can be used by entry-level Makers who want to make a remote-control lamp, to advanced engineers who want to automate their factory. Scientists could use IO to log experiment data in real-time and download or organize gathered data (https://adafruit.it/Ehq). Home automators can connect sensors to their HVAC system to minimize power usage. Roboticists can control and monitor their mechatronics remotely (https://adafruit.it/Ehr) from around the world. Students can build interactive
art creations that tie with social media services and light up (https://adafru.it/Ehs) whenever their posts get a like. With the powerful single board computers and low cost microcontroller boards available, even a first project or a custom one-off product can now be connected and controlled over the Internet.

Makers

We’ve designed IO to be simple and quick to start, for experimenters and Makers of all sorts. Sending data to the cloud is easy to implement with Adafruit IO, pre-existing projects can be converted easily to an Adafruit IO project, control their projects from the Internet. Add Internet-connectivity to household appliances, wearables, art installations, etc.

Engineers

But behind this simplicity, we have made Adafruit IO robust enough to handle the demands of industrial-level engineering without the complexity of enterprise services. We have pre-built APIs (https://adafru.it/ikf) to have code communicate with Adafruit IO, data-logging, webhooks and service integration so you can easily prototype an IoT product before taking the next step and scaling for production. It’s a great place to put together prototypes, and build customized projects for small businesses. We’ll note that it’s not aimed at building and commercializing business-to-customer or business-to-business IoT projects, check out our previous Services (https://adafru.it/Efd) video for some suggestions on enterprise-level IoT services. Those will be a big step up in complexity, and won’t have the same friendly integrations so they’re a good step up once you’ve got Adafruit IO going.

If you’re curious about what people are using Adafruit IO for and maybe get some inspiration or validation for your ideas, please check out the Adafruit discord channel (https://adafru.it/Eht) (#help-with-adafruitio channel), our Adafruit IO forum (https://adafru.it/plC), the Adafruit blog (https://adafru.it/Ehp), and the Adafruit Learning System, where we’ve got dozens of IoT projects with code and wiring diagrams, from the basics of a button, to how to automate an entire house (https://adafru.it/iRB).
IO Plus

Upgrades

Now you’re probably really excited to get started with Adafruit IO, maybe you even have some ideas for your first IoT build. The good news is getting started with Adafruit IO is easy and...free! That’s right, the introductory account that can do just about everything we’ve shown in this video is free with your Adafruit account. Adafruit IO’s basic service gives you:

- 30 data points per minute
- 30 days of data storage
- 10 feeds
- 5 dashboards
- Notifications, triggers and service integrations

After you’ve gotten comfortable with the basic Adafruit IO service, and have had a chance to try out the APIs and services, you may want more and you’re ready to power up you can upgrade to Adafruit IO Plus for:

- 60 data points per minute
- 60 days of data storage
- unlimited dashboards
- and unlimited feeds
- Improved integrations including emailed triggers and Dark Skies weather service

You can subscribe monthly for $10 (https://adafru.it/Eg3). If you’d like to save some money there is even a yearly pass card, available for $99 on Digikey.com (https://adafru.it/Ehu) that will give you a full year of Adafruit IO Plus.

Each purchase gets you all the goodies we’ve mentioned, any new services and integrations we’ll be adding, and help support and improve Adafruit IO!
Conclusion

The Future

We hope that after this, you have a better understanding of what Adafruit IO can do for your upcoming IoT project! Feel free to check out some examples of Adafruit IO projects on the Adafruit Learning System (https://adafru.it/iRB) or read more documentation about Adafruit IO here (https://adafru.it/BRB).

Or (as suggested) visit https://io.adafruit.com/ (https://adafru.it/fsU) and start building your own internet of things project today!