

Introduction

This document describes how to get data from a Browan TBHH100 temperature and humidity sensor to flow in real time to a Google Sheet.

The end-to-end data flow is as follows:

TBHH100 > Helium Hotspot > Helium Console > Pipedream > Google Sheet

Table of Contents

Contents

Introduction	1
Table of Contents	1
References	1
Preconditions	2
High Level Steps	3
Detailed Level Steps	3
Appendix A – Sample TBHH100 Date from the Helium Console Debugger	24

References


The table below lists the learning resources used to support this effort.

ID	Topic	Reference	Description
1	TBHH100	https://github.com/helium/console-decoders/tree/master/Browan/TBHH100 <ul style="list-style-type: none"> Decoder.js RM_Temperature _ Humidity Sensor.pdf BrowanTBHH100data_Decoder.pdf 	TBHH100 in the Helium decoder Github repository <ul style="list-style-type: none"> JavaScript decoder function Reference Manual for the TBHH100 How to interpret the payload of a TBHH100 and info on the decoder
2	Helium	<ul style="list-style-type: none"> https://developer.helium.com/console/adding-devices https://developer.helium.com/console/functions helium/console-decoders repository https://developer.helium.com/console/labels 	<ul style="list-style-type: none"> How to add a device to the console How to use/create a Function in the console GitHub repository of Console decoders How to use labels in the console
3	Pipedream	<ul style="list-style-type: none"> https://pipedream.com/workflows 	<ul style="list-style-type: none"> Pipedream Web Page where your workflows are found
4	Mike's Documents	https://github.com/mikedsp/helium/tree/master/MyDocuments <ul style="list-style-type: none"> PipeDreamBootstrap - 	Mike's Helium Document GitHub Repository <ul style="list-style-type: none"> Detailed notes on how to set up a Pipedream workflow to send data from Twitter to a

		TweetToGoogleSheet-SHARE.pdf • HowTo_BrowanTBHH100_to_GoogleSheet-SHARE.PDF	Google Sheet • (this document) How to get data from a Browan TBHH100 temperature and humidity sensor to flow in real time to a Google Sheet.
--	--	--	---

Preconditions

The table below lists what is needed to complete this project.

Precondition	Description
TBHH100	 <p>This is the temperature and humidity sensor. I got mine from Cal-Chip Connect here: https://www.calchipconnect.com/shop/temperature-and-humidity-sensor</p>
In range of a Helium Hotspot	<p>Your TBHH100 has to be in range of a Helium hotspot. If you don't already own a hotspot and have it near you or you're not within range of someone else's hotspot, your best bet is to get a free Helium hotspot via Emrit – that's what I did.</p> <ul style="list-style-type: none"> • https://www.emrit.io/ • https://www.emrit.io/signup/
Pipedream Account	<p>Pipedream will serve as the integration piece between Helium and the Google Sheet. You can sign up for a free account here:</p> <ul style="list-style-type: none"> • https://docs.pipedream.com/sign-up/ <p>You don't have to be a computer programmer to use Pipedream, but it will certainly be easier to understand how to use pipedream if you have a programming background – especially if you have already done Node.js programming and have seen JSON before. If you don't have this background, don't despair as some grit and determination will suffice. Web interfaces and Node.js was completely new to me. When I first looked at Pipedream my head exploded. I learned enough to get the job done by painstakingly following the steps in this 4 minute video, which walks through a project to send Twitter tweets to a Google Sheet. Total time investment was about 6 hours.</p> <ul style="list-style-type: none"> • https://www.youtube.com/watch?v=hJ-KRbp6EO8
Helium Console Account w/ some Data Credits	<ul style="list-style-type: none"> • It's free to create a Helium Console account • Once the account is created, you can purchase data credits with a credit card from the Data Credits page. I purchased 1,000,000 credits for \$10. With my 3 devices (2 object locators and the TBHH100) that's enough credits for over 1 year.

High Level Steps

High level steps are as follows

1. Get the TBHH100 set up in the Helium Console
 - a. Apply a decoder function so you can read/decode the data from the device
2. Create a pipedream endpoint (via a new Pipedream Workflow)
 - a. This is so you have somewhere to send data from the Helium Console
3. Create an HTTP Integration in the Helium Console
 - a. Use the URL from the pipedream endpoint in the previous step
 - b. Use labels to connect the integration to the TBHH100 device and the decoder function
4. Verify that Data is flowing from the TBHH100 > Helium Console > Pipedream workflow
5. Create a Google Sheet
 - a. Make 1 column for each data element you see (or are interested in) when looking at the TBHH100 device in the Helium Console with debug turned on
6. Complete the development of the Pipedream workflow
 - a. Make the connection to your Google Sheet
 - b. Drop the trigger data into a Pipedream step that connects to your Google Sheet
7. (optional) Do some data manipulation of the sensor data in the Pipedream workflow to make it read better in the Google sheet, e.g. the sensor is sending the timestamp in Unix
8. (optional) Enjoy the sensor data flowing into your Google Sheet. Share the Sheet with a friend.

Detailed Level Steps

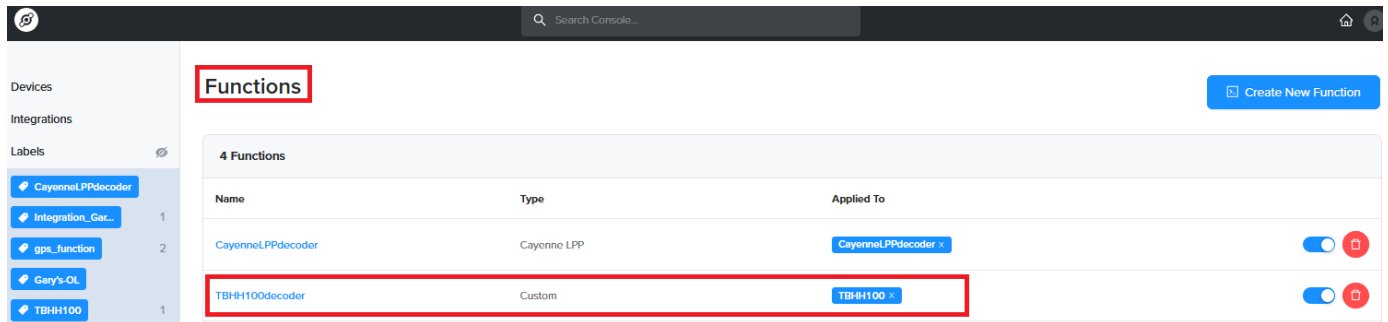
1. Get the TBHH100 set up in the Helium Console
The end result of this step is that you will have a TBHH100 in your Helium Console and you'll be able to read the data sent from the device in the Helium debugger.

The 2 screenshots below show the end result of this step. Note that I have a TBHH100-to-Cayenne integration already set up and running. That's what the label, *tbhh100-int*, is for. You can ignore that for this project.

Device Name	Device EUI	Labels	Integrations	Frame Up	Frame Down	Packets Transferred	DC Used	Date Activated	Last Connected
TBHH100915012992-wApkey	5840C00000110F13					0	0	Aug 17, 2020 8:27 PM	
TBHH100	5840C0000011C1D4	TBHH100	cayenne-TBHH100	1192	2	321	231	Aug 3, 2020 5:58 PM	Aug 23, 2020 7:53 AM

(above) End Result - TBHH100 in the Helium Console.

- Device Name = TBHH100
- Label = TBHH100 (this is the label I applied to the decoder function I created)
- 321 Packets transferred since the device was activated on Aug 3, 2020



(above) End Result – custom decoder for the TBHH100 with the label, *TBHH100*

The sub steps below walk you through getting step 1 completed.

a. Add the TBHH100 to your Helium Console

Follow the Helium instructions here: <https://developer.helium.com/console/adding-devices>

b. Create a custom decoder function and assign a label to it

Follow the 'Creating a Decoder' Helium instructions here: <https://developer.helium.com/console/functions>

- From the [helium/console-decoders repository](https://github.com/helium/console-decoders), use the decoder.js code in the Browan/TBHH100 folder
- Add a new label called TBHH100

TBHH100decoder

Function Details

Update Function

TBHH100decoder

Decoder



Custom Script



Clear

Custom Script

```

0 function Decoder(bytes, port) {
1
2     var params = {
3         "bytes": bytes
4     };
5
6     // VOC Measurement
7     // Disabled on the TBHH100, i.e. is always ffff so comment this section out
8     /*
9     voc = (bytes[7] << 8) | bytes[6];
10    if (voc === 65535) {
11        voc_error = true;
12    } else {
13        voc_error = false;
14    }
15    */
16
17    // CO2 Measurement
18    // Disabled on the TBHH100, i.e. is always ffff so comment this section out
19    /*
20    co2 = (bytes[5] << 8) | bytes[4];
21    if (co2 === 65535) {
22        co2_error = true;

```

Labels Applied To

Labels are necessary to apply Functions to devices

Add a Label

Search or Create Label...

Add

Attached Labels

TBHH100 x

(above) Creating the TBHH100 decoder function

- Function name = 'TBHH100decoder'
- Code copied in from the decoder.js file in the Browan/TBHH100 folder in the Helium decoder repository
- Label *TBHH100* created and attached to the function

- c. Apply the decoder function to the TBHH100 device by applying the *TBHH100* label created in the previous step to the TBHH100 device

You can see this in the screenshot under step d, below, where the label, *TBHH100*, is highlighted in red. For more information on the use of labels, check the Helium documentation here:

<https://developer.helium.com/console/labels>

- d. Looking at the TBHH100 device in the console, with the debugger turned on, verify that the decoder is working by making sure you can see/read the device data in the debug window.

Note that the TBHH100 transmits data when it senses a 2C temperature change or every few hours. There is no button on the device to force a data transmission. A good way to ‘force’ a data transmission is to place it in or out of your freezer.

The screenshot shows the Helium Console interface. On the left, a sidebar lists various categories: Devices, Integrations, Labels, Functions, Organizations, Users, and Data Credits. Under the 'Labels' section, several labels are listed, including 'CayenneLPDecoder', 'Integration_Ga...', 'gps_function', 'Gary's-OL', 'TBHH100', 'tbhh100-int', and 'cayenne1'. The 'TBHH100' label is highlighted with a red box. The main area displays the 'Device Details' for 'TBHH100'. Fields include Name (TBHH100), UUID (759ab813-673b-45fb-bb8e-6abdb66e7860), Device EUI (58A0CB000), App EUI (58A0CB0000210000), App Key (B9EAC6075), Activation Method (OTAA), Attached Labels (TBHH100, tbhh100-int), and Associated Integrations (cayenne-TBHH100). A red box highlights the 'TBHH100' label in the 'Attached Labels' section. On the right, the 'Debug' window shows a packet of data with a red box highlighting the 'decoded' section, which contains the following JSON:

```
{
  "description": "Connection established",
  "id": "b2f47019-1df0-4149-8887-473fe9b7f469",
  "name": "cayenne-TBHH100",
  "status": "success"
}, {
  "debug": {
    "req": {
      "body": {
        "app_eui": "58A0CB0000210000",
        "dc": {
          "balance": 1006488,
          "nonce": 1
        }
      }
    },
    "decoded": {
      "payload": {
        "batt": 3.6,
        "bytes": [
          8,
          251,
          40,
          24,
          255,
          255,
          255,
          255
        ],
        "rh": 24,
        "temp": 8
      }
    },
    "status": "success"
  }
}
```

(above) Looking at the TBHH100 device in the Helium Console with the debugger turned on

- “decoded” means the decoder has been applied to the data payload from the device
- Relative Humidity = 24%
- Temperature = 8C
- If you want a deeper understanding of the data, check out the two .pdf files in [the Browan/TBHH100 folder in the Helium decoder repository](#)

At this point, you’ve completed step 1 and you should now have a TBHH100 in your Helium Console and you are able to read the temperature and humidity data being sent from the device to the Helium Console.

2. Create a pipedream endpoint (via a new Pipedream Workflow)

After this step you’re going to create an HTTP integration in the Helium Console. When you do that, one of the parameters you’ll need is the endpoint (or the HTTP address to send the data). So in step 2, we’re going to do the initial Pipedream workflow creation in order to obtain the endpoint, but we’ll finish the Pipedream work after we create/setup the HTTP integration in the Helium Console.

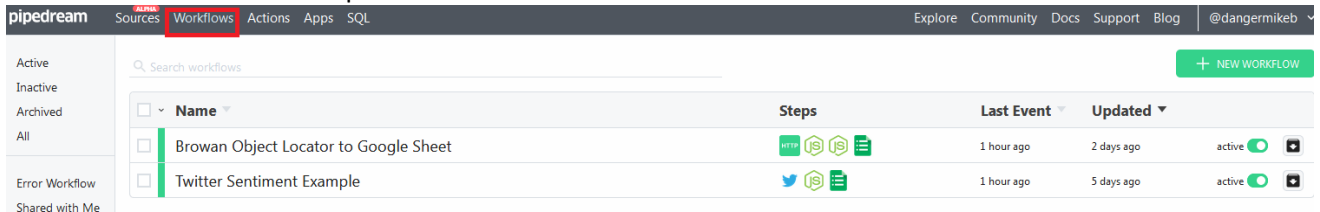
Here’s what we’re going to do in step 2

- Go to the Workflow tab in Pipedream
- Create a new workflow and set “HTTP/Webhook” as the trigger

- Give the workflow a Name

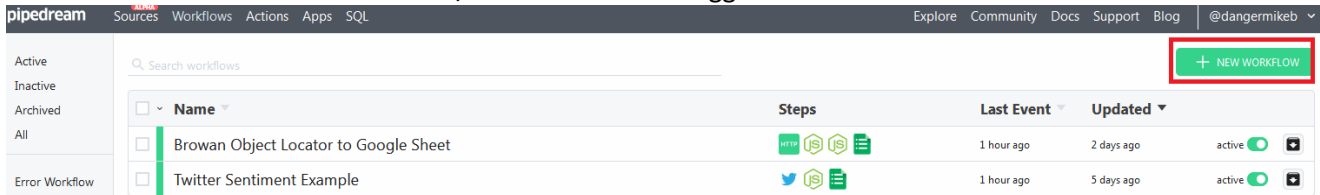
Now let's walk through these steps.

a. Go to the Workflow tab in Pipedream

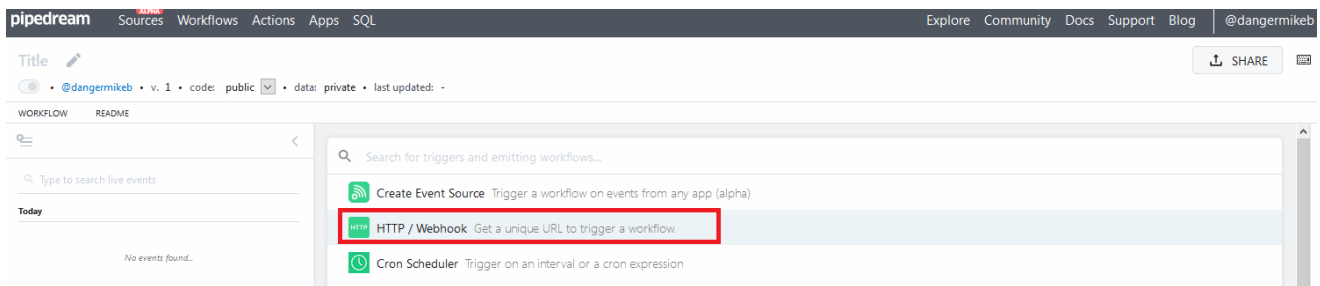


(above) On the Workflow tab of Pipedream

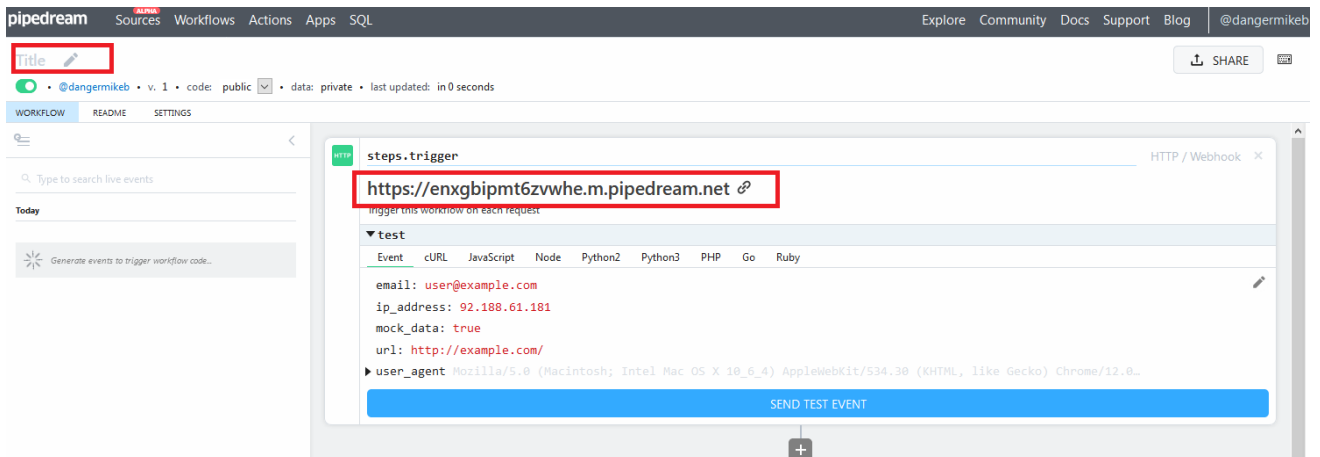
b. Create a new workflow and set “HTTP/Webhook” as the trigger



(above) Initiate the creation of a new workflow



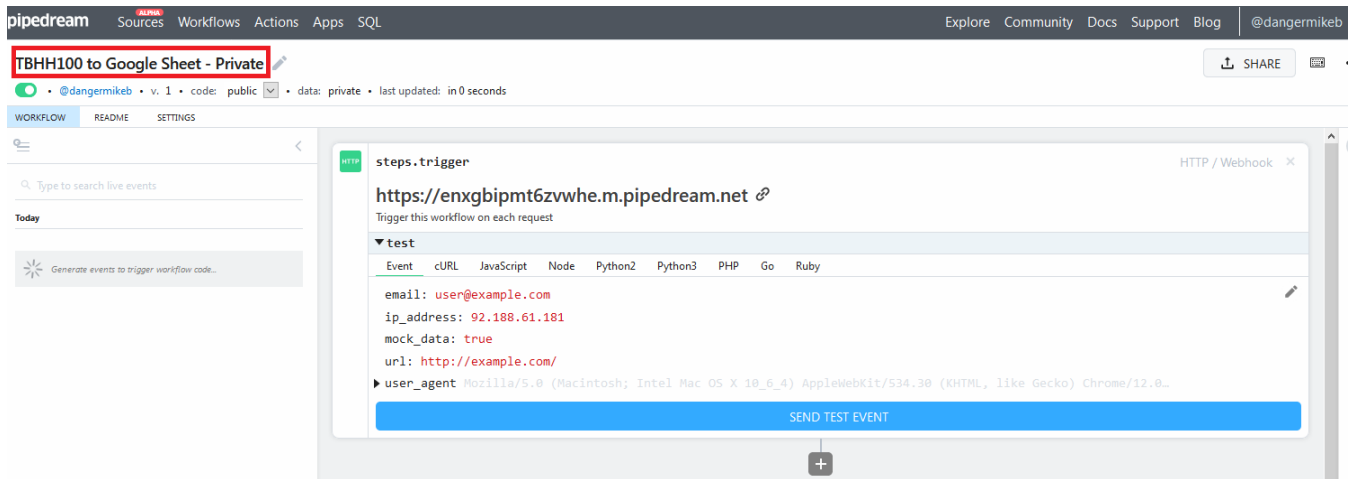
(above) Selecting HTTP/Webhook as the trigger



(above) New workflow with an HTTP/Webhook trigger.

- Notice that there is no Title (name) yet for the workflow
- The URL right under 'steps.trigger' is the endpoint you'll use in when you set up the HTTP integration in the Helium Console

c. Give the workflow a name



(above) Workflow name set to 'TBHH100 to Google Sheet – Private'.

3. Create an HTTP Integration in the Helium Console

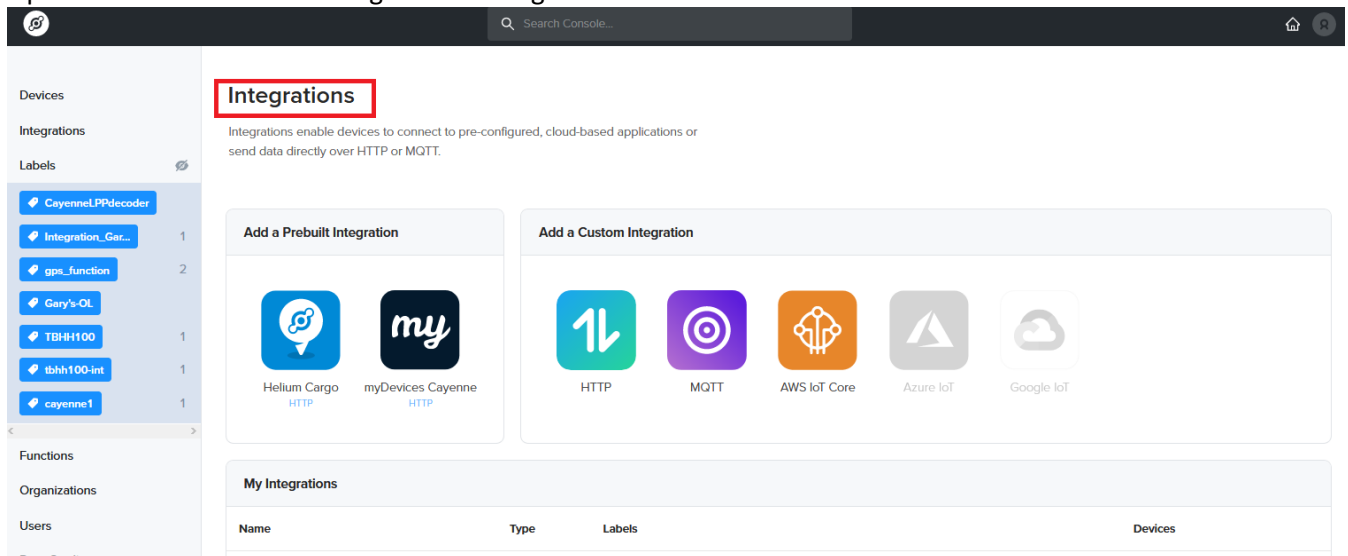
Now that we have the location in Pipedream where we want to send the device data (from step 2), we're going to create a Custom HTTP integration in the Helium Console to send the data. After this step we'll go back into Pipedream and work on getting that data over to a Google Sheet.

Here's what we're going to do in step 3

- Open Helium Console and go to the Integrations tab
- Initiate the creation of a Custom HTTP Integration
- Complete the form
 - Specify the URL of the Pipedream endpoint
 - Name the integration
 - Apply the TBHH100 decoder label that is already applied both the TBHH100 Decoder function and device
- Create the Integration

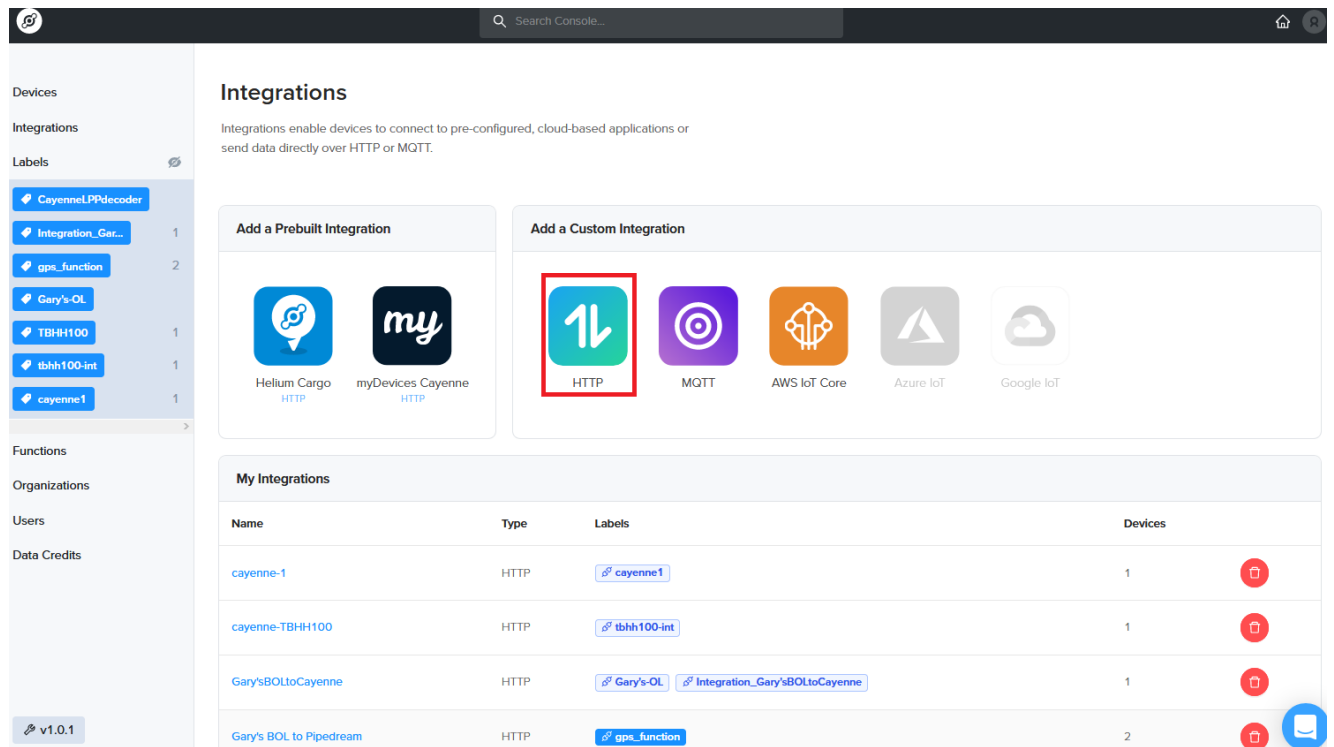
Now let's walk through these steps.

a. Open the Helium Console and go to the Integrations tab

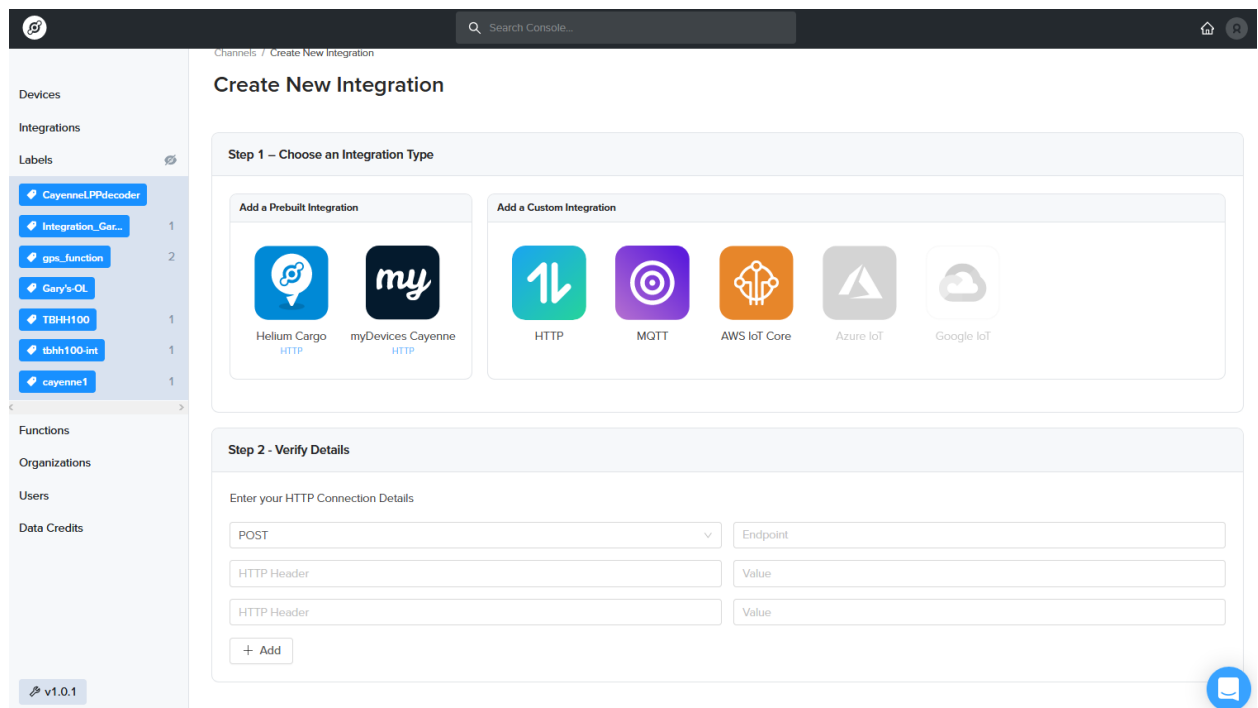


(above) Looking that the Integrations tab of the Helium Console

- b. Select 'HTTP' from the Custom Integration options

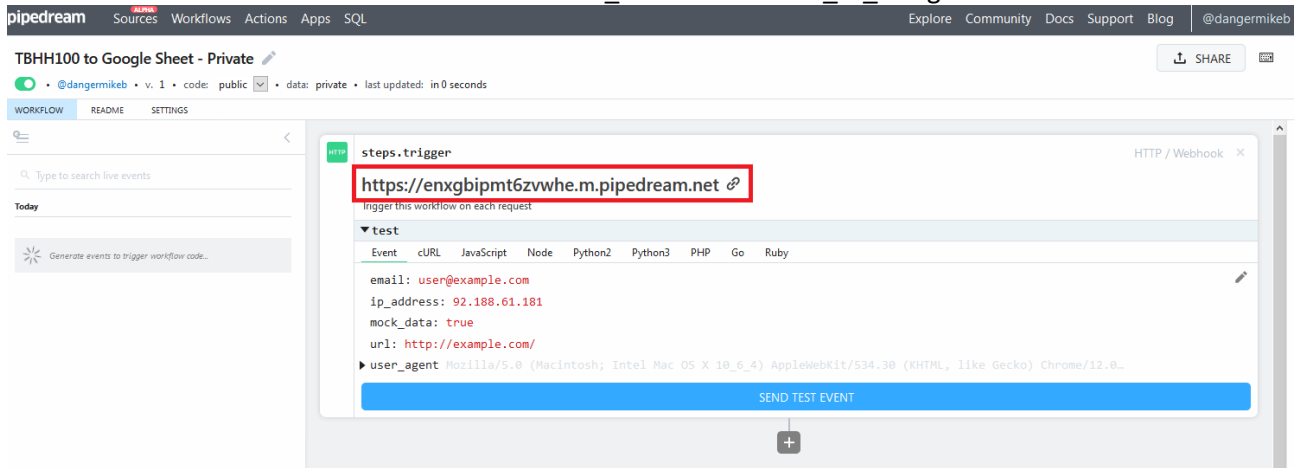


(above) Initiating the creation of a new Custom HTTP Integration by selectin 'HTTP'

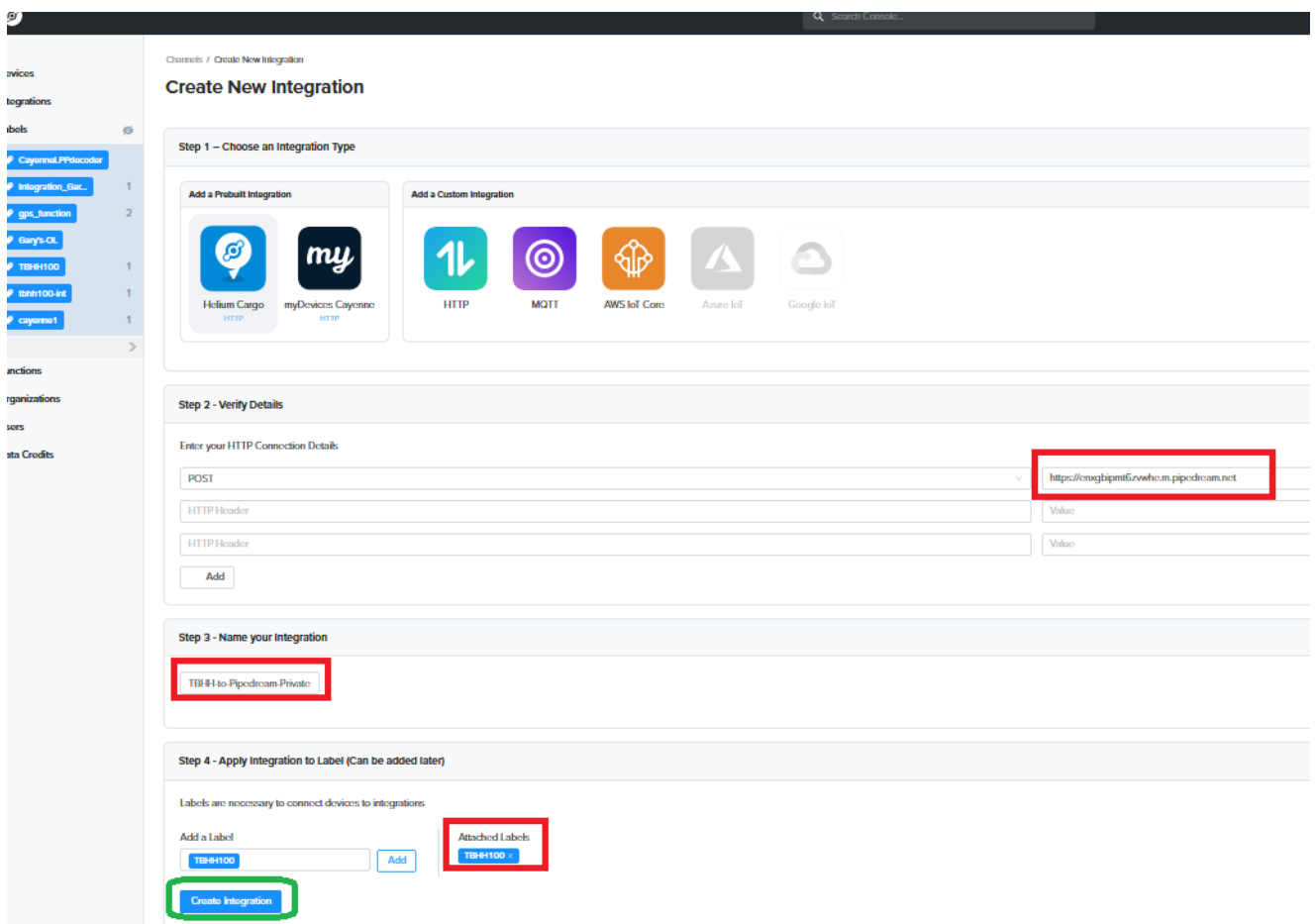


(above) New Integration Form – nothing filled in yet

- c. Fill in 3 elements of the Integration form, then select the 'Create Integration' button



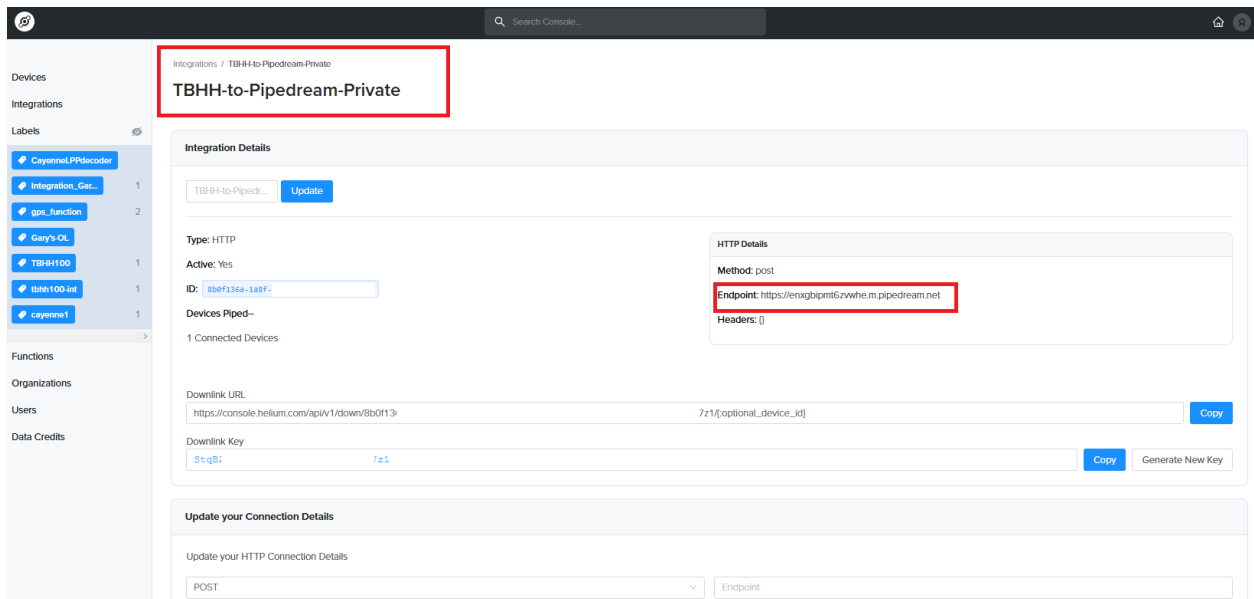
(above) Shown as a reminder – the Pipedream workflow URL



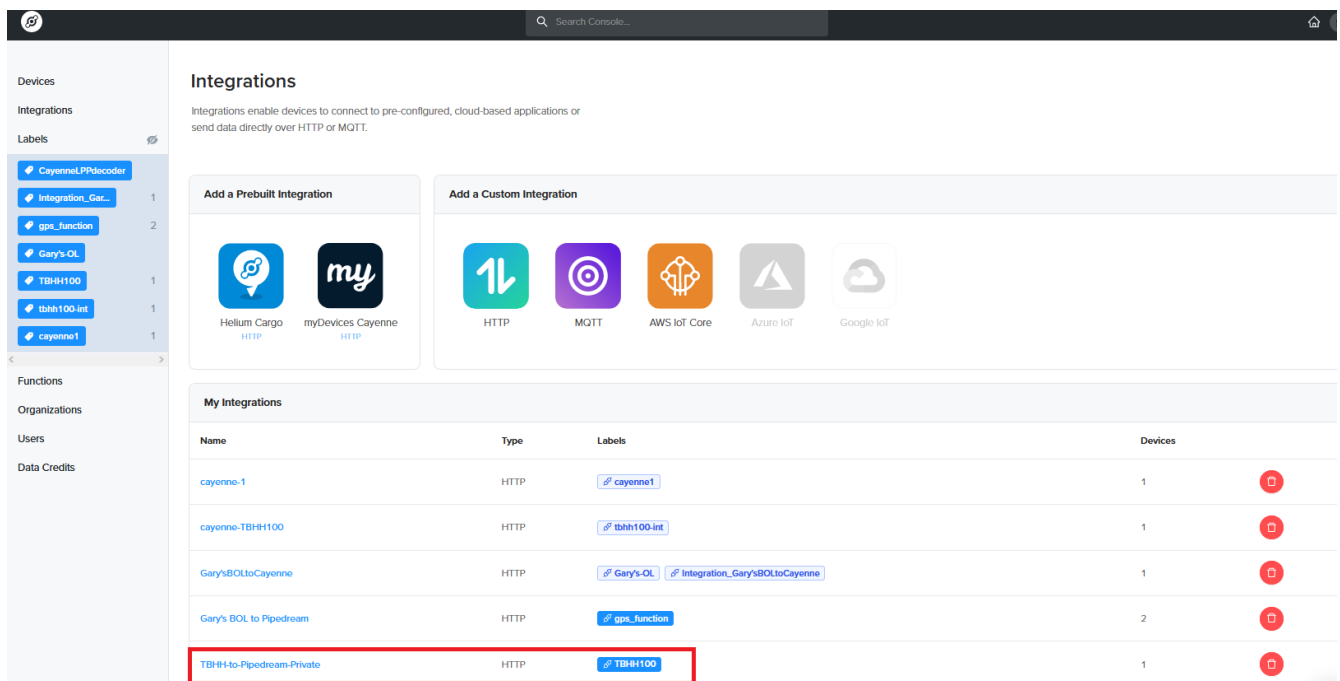
(above) Completed New Integration form, ready to hit the 'Create Integration' button

- POST Endpoint = <https://enxgbipmt6zvwhe.m.pipedream.net>
- Integration name = "TBHH-to-Pipedream-Private"
 - Pick a name that means something to you. I chose this name because I don't intend to share the Google Sheet output with the general public. I intend to set up a similar TBHH100-to-Google Sheets integration to a Google Sheet that I intend to provide read-only access to anyone. I will name that integration, "TBHH-to-Pipedream-Public"
- Label Added = TBHH100

- Note it is very important that you apply the same label to the integration that you applied to the TBHH100 decoder function and the TBHH100 device. If you don't do this, you won't be able to decode the data in your Pipedream workflow. See the [Helium Label documentation](#) for more details.



(above) HTTP integration called 'TBHH-to-Pipedream-Private' has been created in the Helium Console
 Note – I have cut out part of the Downlink URL, ID, and Downlink key for privacy reasons.



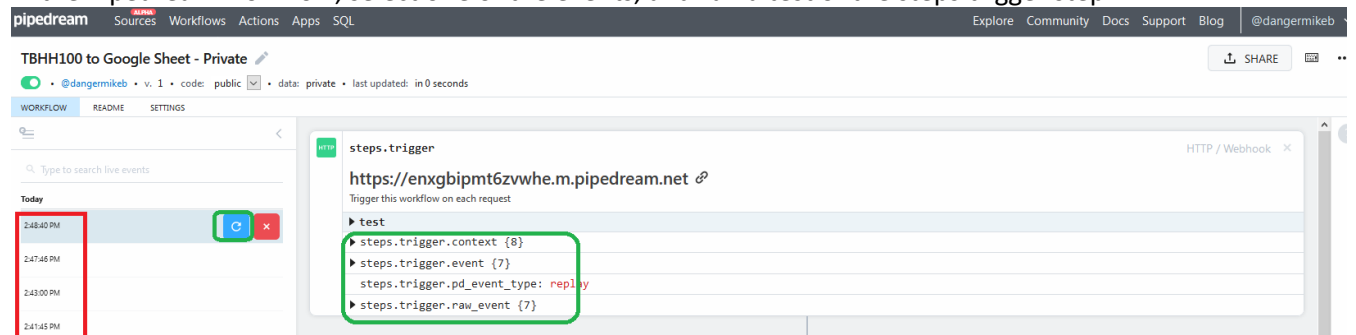
(above) 'TBHH-to-Pipedream-Private' now shows up in your list of Integrations

- Verify that Data is flowing from the TBHH100 > Helium Console > Pipedream workflow

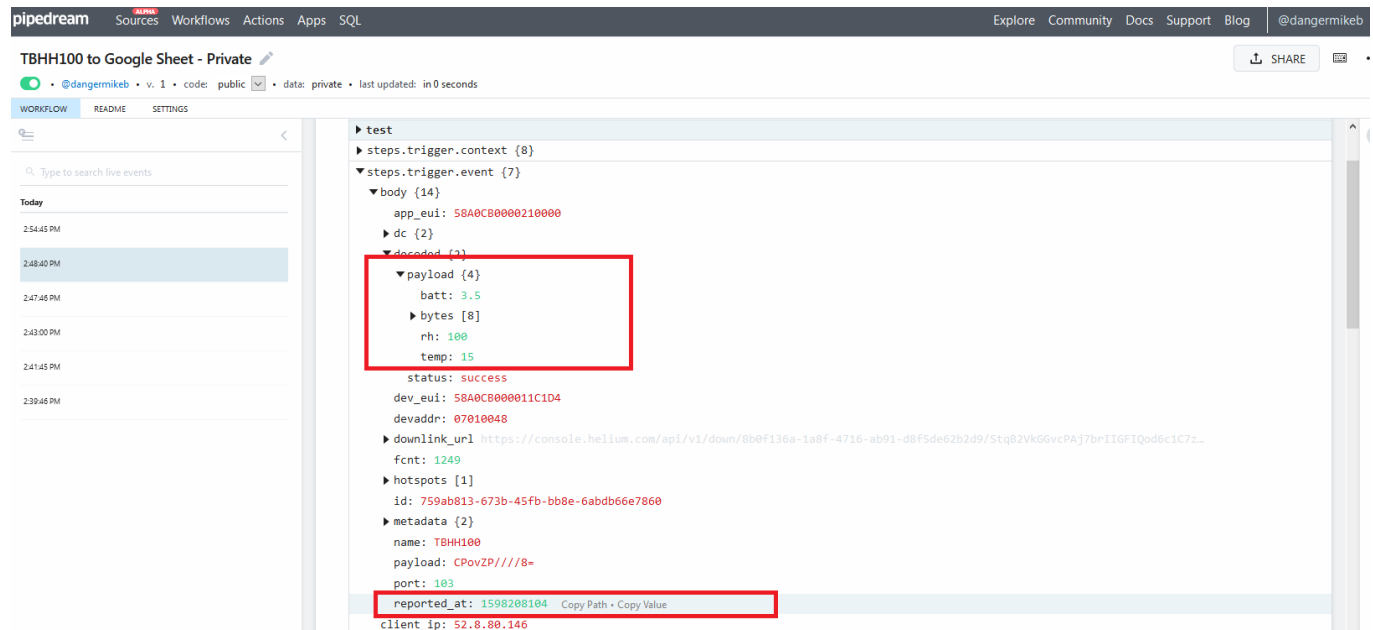
If you have set up the HTTP integration correctly, you should start to see some events showing up in your Pipedream workflow

Note – it may take from 5-15 minutes before you start seeing events. You’ve got to trigger a temperature change in the TBHH100 (take it out of the freezer) and wait for device to notice that. And sometimes it takes the console a while before it starts picking up events. If after 15 minutes you are not seeing data events in Pipedream, double check your labels. Also – turn the console debugger on and wait for it to catch an upload from the TBHH. Appendix A shows what your data should look like in the Console debugger.

In the Pipedream workflow, select one of the events, and run a test of the steps.trigger step



(above) Pipedream workflow with 4 events received (red box on the left). The green boxed items correspond to using the most recent event (2:48:40pm) as a test. You can open up the items in the green box on the right to view the information that is available to you.

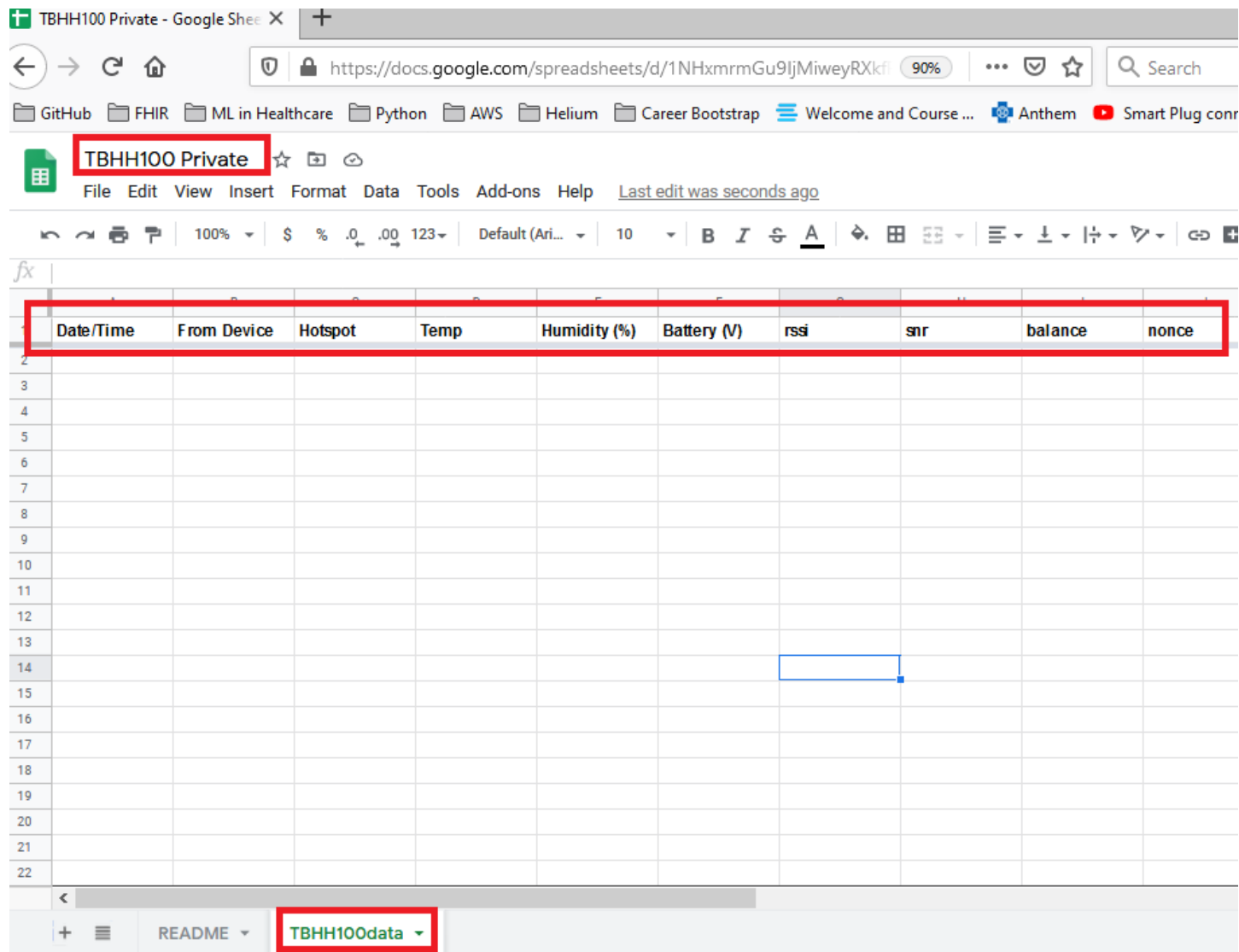


(above) Opening up the steps.trigger.event to see the information in there.

Hint: There is lots of good and interesting stuff in steps.trigger.event! In fact, it’s looking at the available data in the event that helps me figure out what data I want to see and how to set up my Google Sheet.

5. Create your Google Sheet

Create a Google Sheet, give it a Title, name the Sheet, and make a column for each data element you want to see.



(above) Google sheet

- Title = 'TBHH100 Private'
- 10 data columns
- Sheet is named 'TBHH100data'

Note: You don't have to know all the columns you want to see at first. Once you've got your connection from Pipedream to the Google Sheet going and you know how to find data in your event, you can add/remove data elements that posted to the Google Sheet very easily.

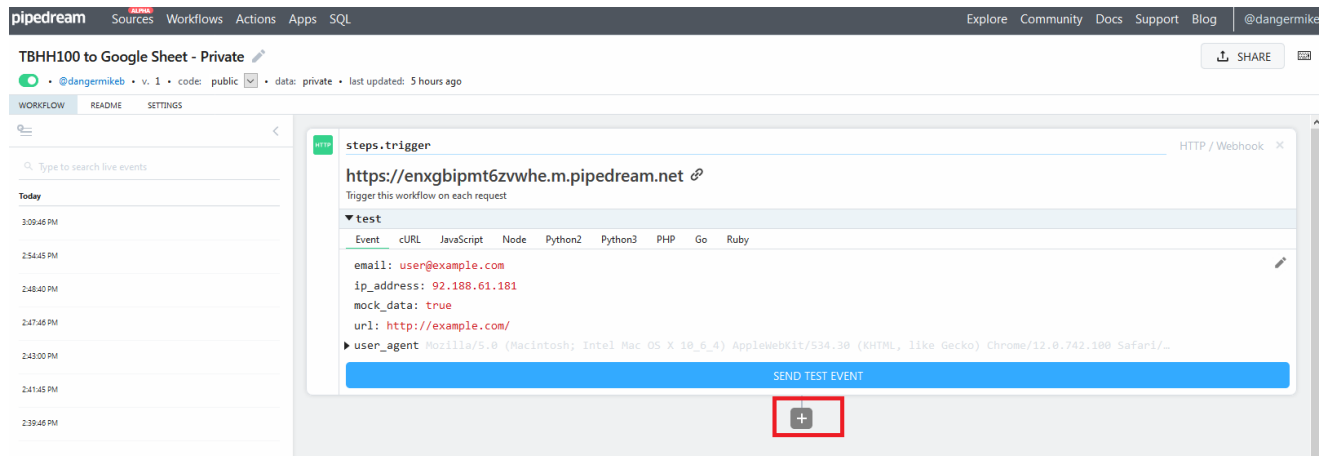
6. Complete the initial development of the Pipedream workflow

Here's what we're going to do

- Add a 'Google Sheet – Add Single Row to a Sheet' action to the workflow
- Connect to Google Sheets
- Configure the Spreadsheet ID and sheet ID
- Set the data parameters to send to the Google Sheet

Let's see how that looks....

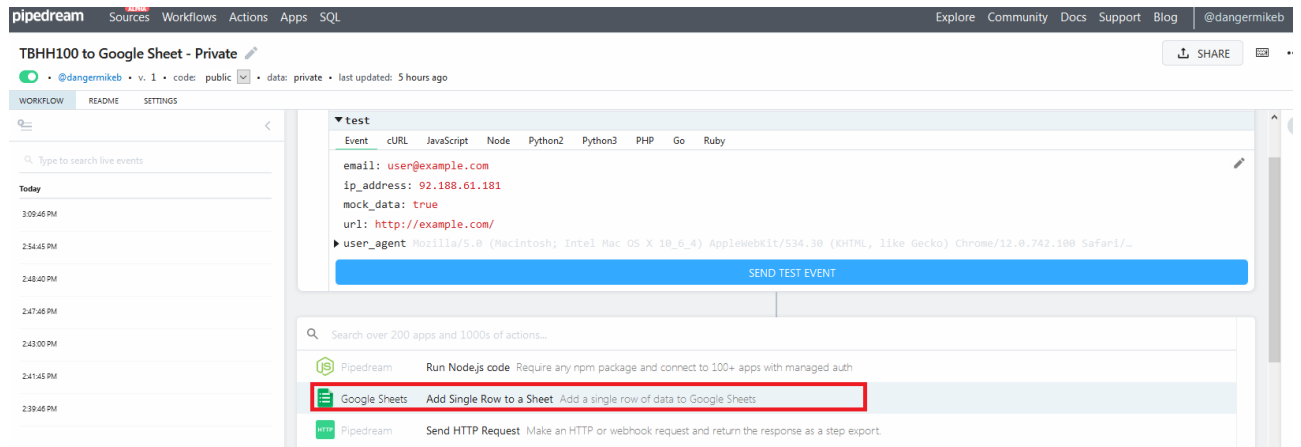
a. Add a new action



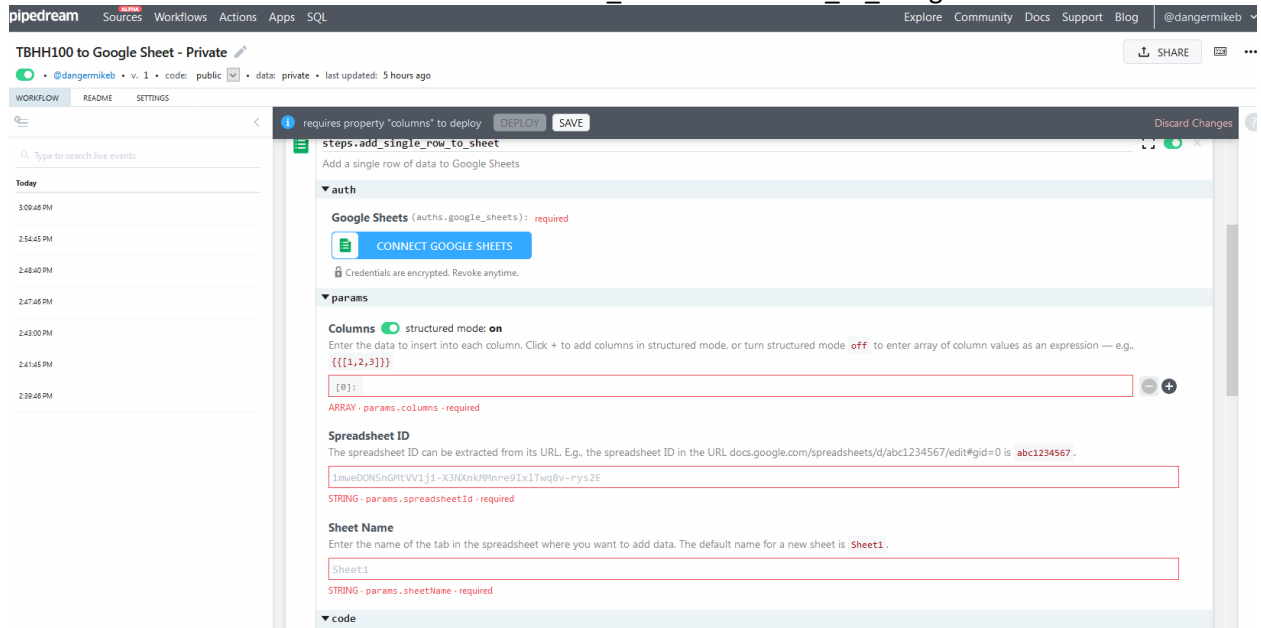
(above) Select '+' to add a new action

If you sent a test event, the '+' may not be there and instead you may skip to step b

b. Of the action types, select 'Google Sheet – Add Single Row to a Sheet'

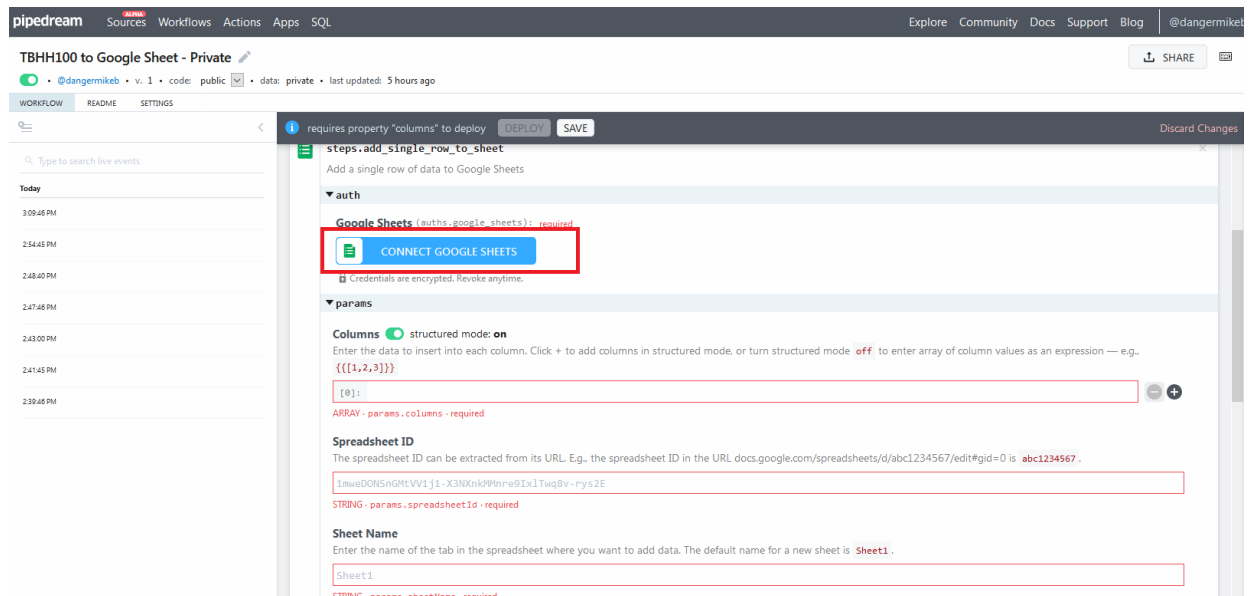


(above) Selecting 'Google Sheet – Add Single Row to a Sheet'

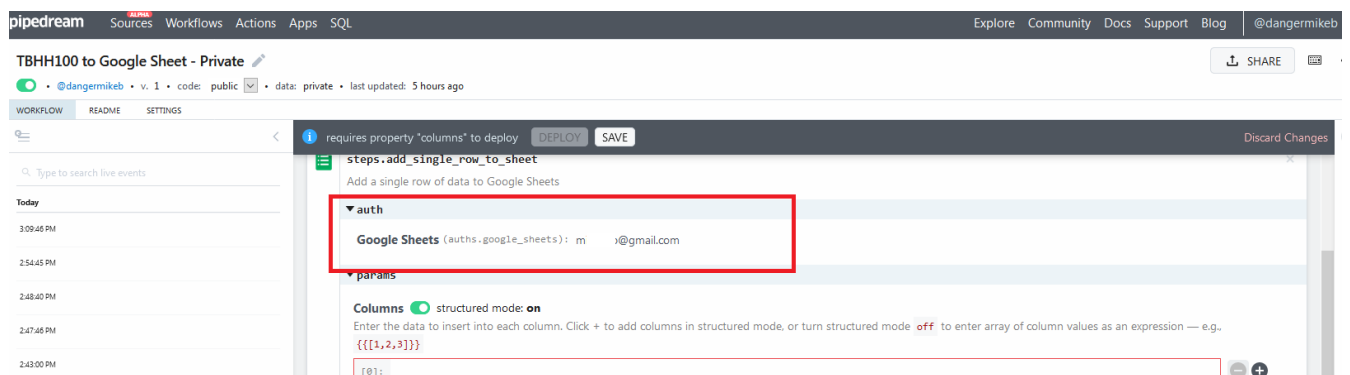


(above) Result of selecting 'Google Sheet – Add Single Row to a Sheet'

c. Make the connection to Google Sheets



(above) Making the connection Google Sheets



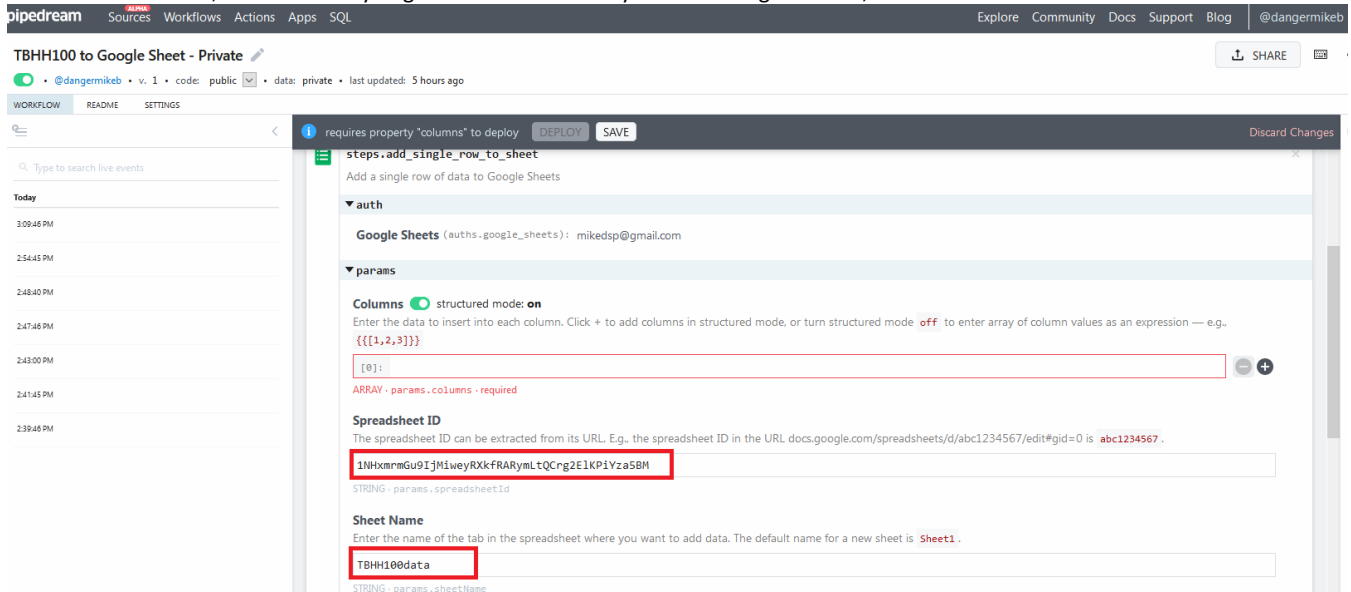
(above) Result of making the connection to Google Sheets (I've cut out part of my email address)

d. Configure the Spreadsheet ID and sheet ID parameters

For the Spreadsheet ID, copy the URL of your Google Sheet and extract the unique identifier. As an example, see the yellow part of the URL below.

<https://docs.google.com/spreadsheets/d/1NHxmrmGu9IjMiweyRXkFRARymLtQCrg2EIKPiYza5BM/edit#gid=0>

For the Sheet ID, use the name you gave the Sheet where your data will go. For me, the sheet is called 'TBHH100data'

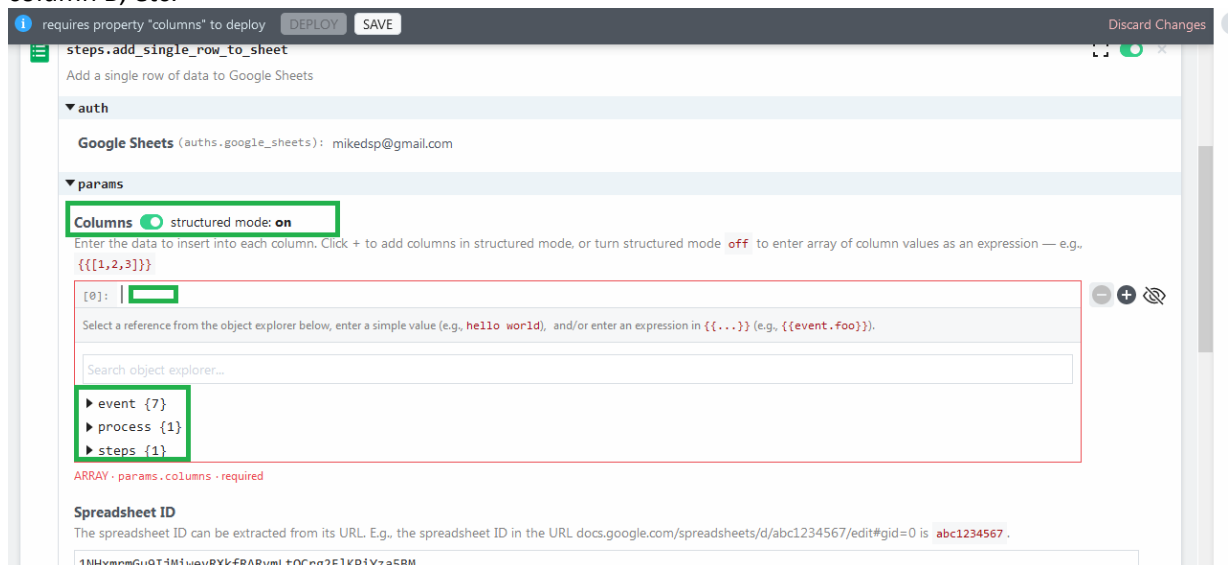


(above) Spreadsheet ID and Sheet Name filled in with details of the Google Sheet created in the step 6

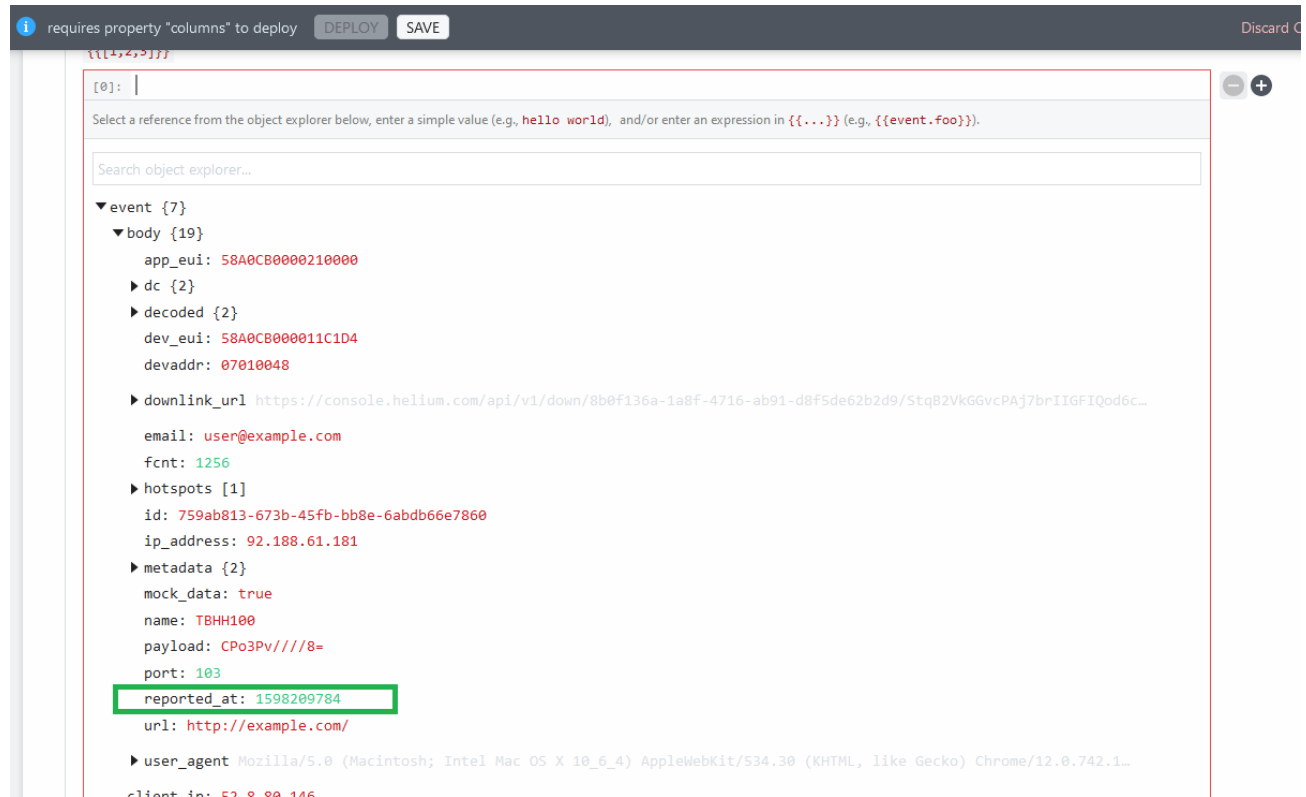
e. Add in the data parameters you want to send.

Left click into the red box in the step and options to select data from event/process/steps will appear (see the green boxes in the screenshot below). Open the event and you'll see values you can choose from. **Note – make sure 'structured mode' = on.**

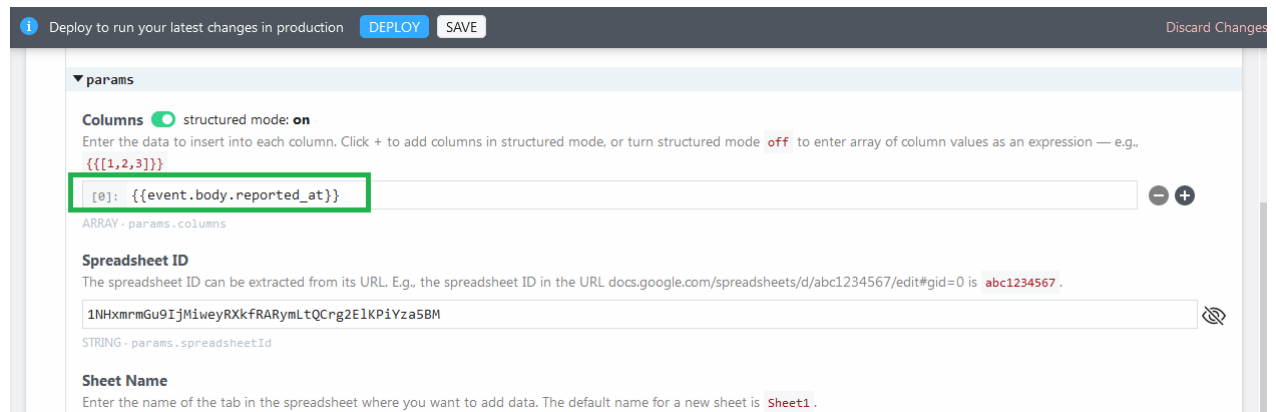
The first parameter you specify will go into column A in the Sheet, the second parameter will go into column B, etc.



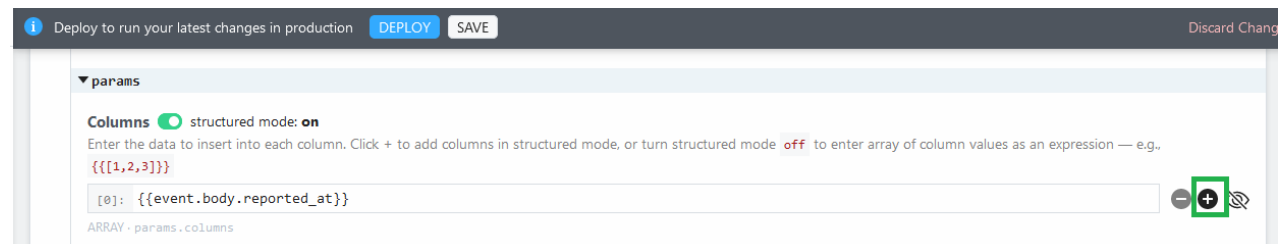
(above) Getting ready to specify the data to go into column A



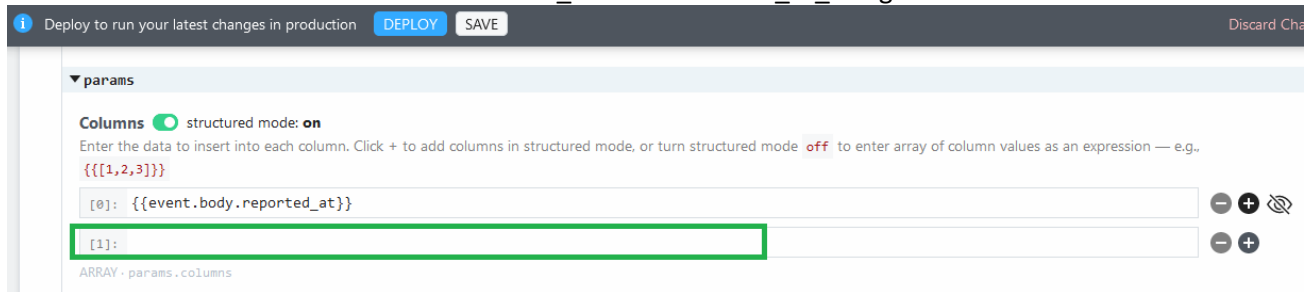
(above) Selecting the 'reported_at' value



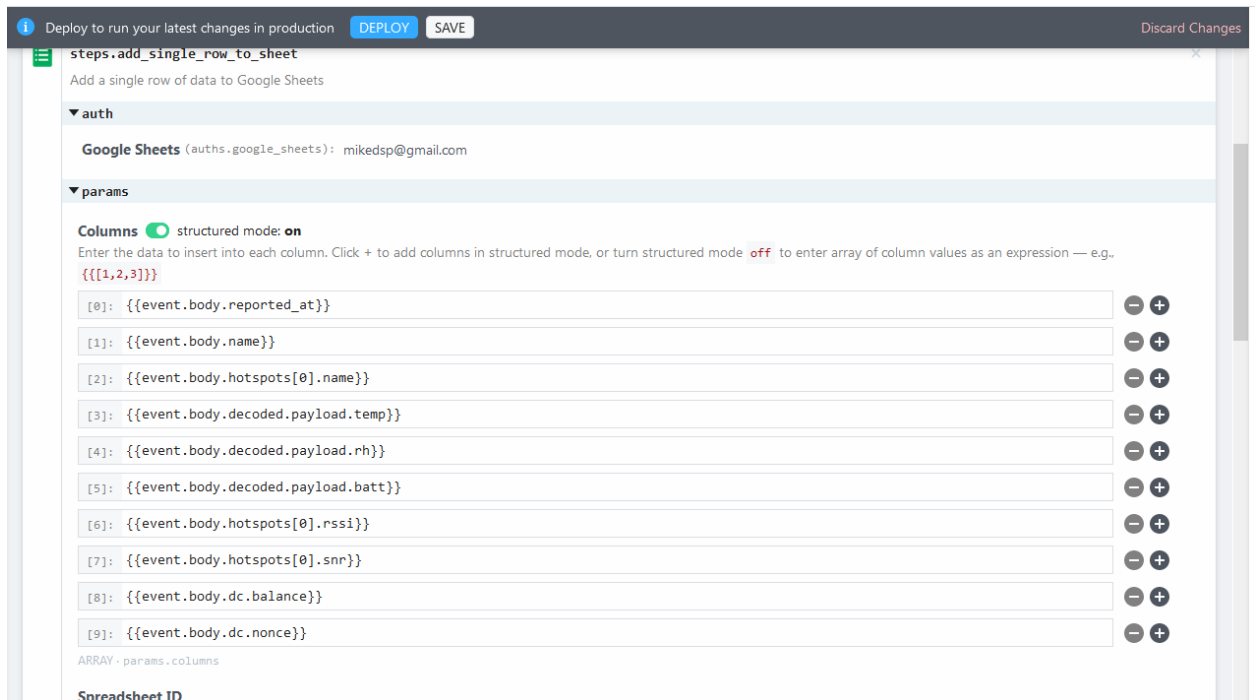
(above) The value that appears in event.body.reported_at will go into column A of the Sheet



(above) Use the '+' at the right to add another parameter



(above) ready for the next parameter to be specified

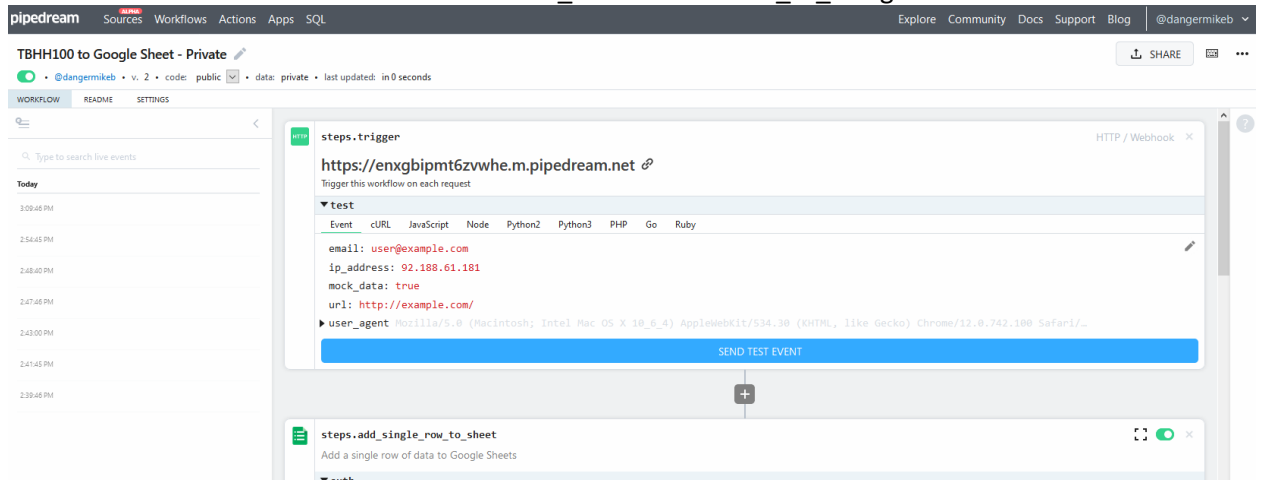


(above) All 10 parameters identified in the Google Sheet are specified

f. Deploy your code and send a test event

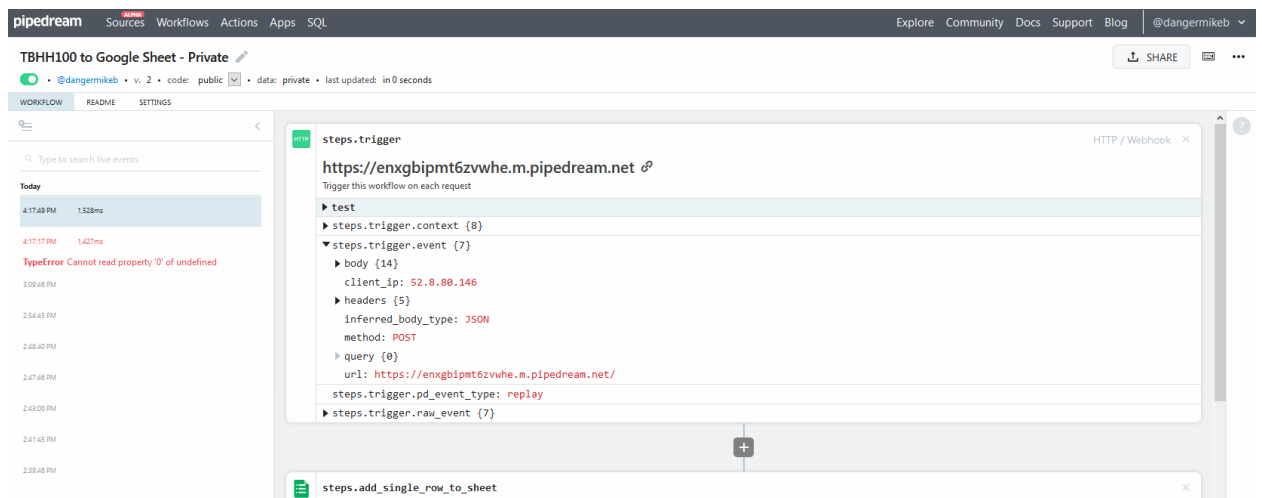


(above) Deploy your code

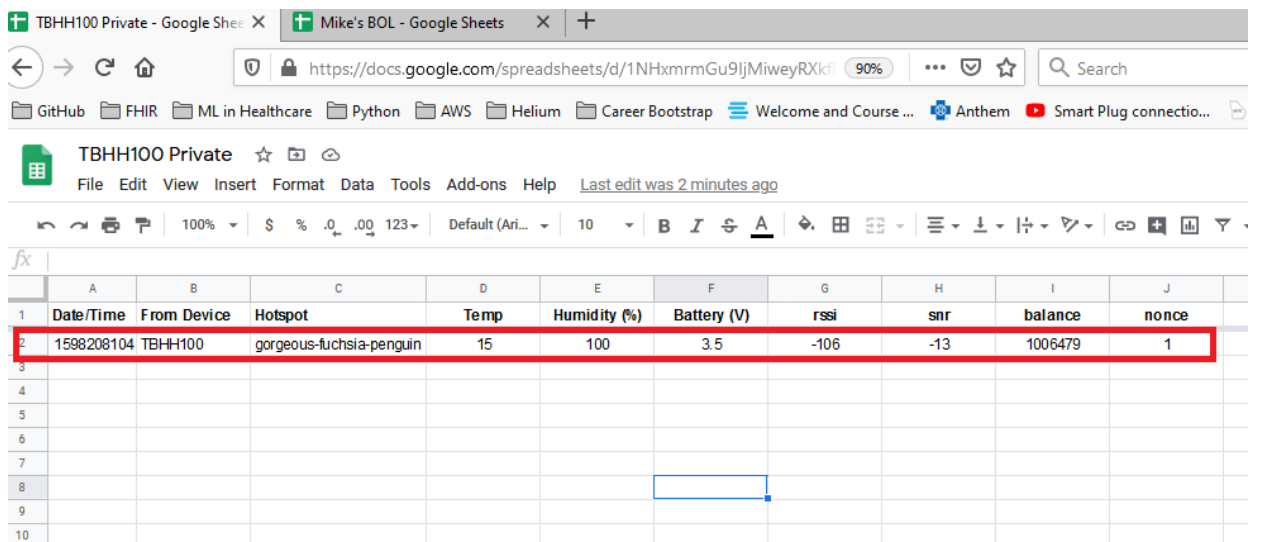


(above) Code has been deployed and we're ready to send a test event

Hit 'SEND TEST EVENT' which should inject the most recently received Event on the left side of your screen into the workflow



(above) What the workflow looks like after sending the test event



(above) What you should see in your Google Sheet if everything is working – 1 row of data corresponding to the data that was in your test event

7. (optional) Do some data manipulation of the sensor data in the Pipedream workflow to make it read better in the Google sheet.

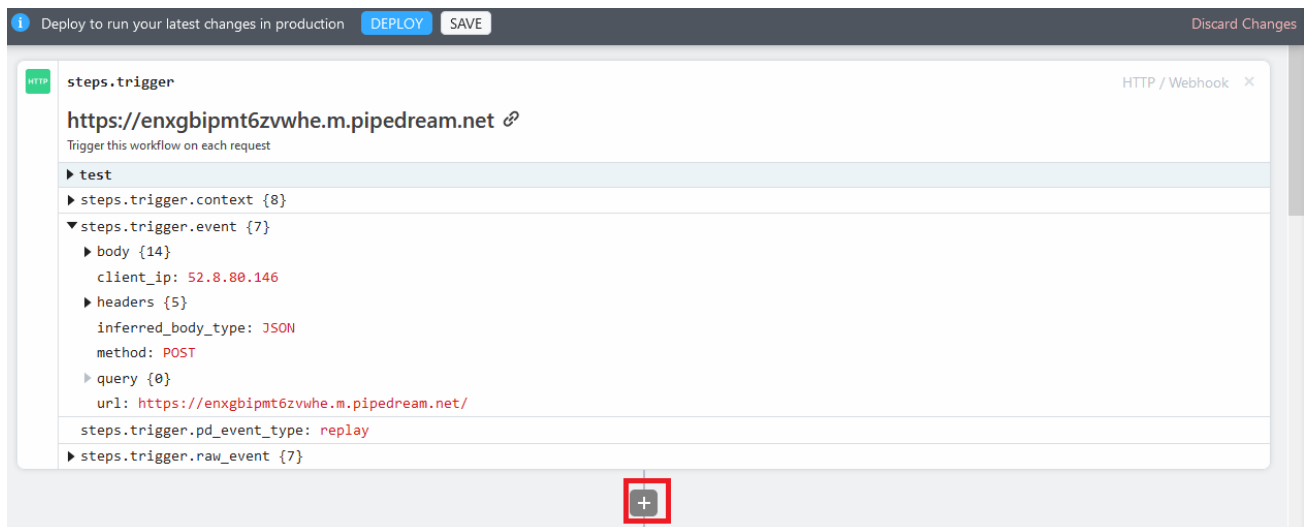
If you look closely at the data in the Google Sheet, you'll see that the Date/Time is a very large integer and that the temperature is being displayed in Celsius. The timestamp in Unix format - i.e. the number of seconds since 1970. That's not readable in the Google Sheet.

If you want to clean that up, you can do that with fancy footwork in the Google Sheet or in the Pipedream workflow before you send it to the Sheet. I found it easier to do the cleanup in Pipedream – and this is from a guy who learned what Node.js is 1 week ago. The point being, it's not too hard to do simple data manipulation in Pipedream.

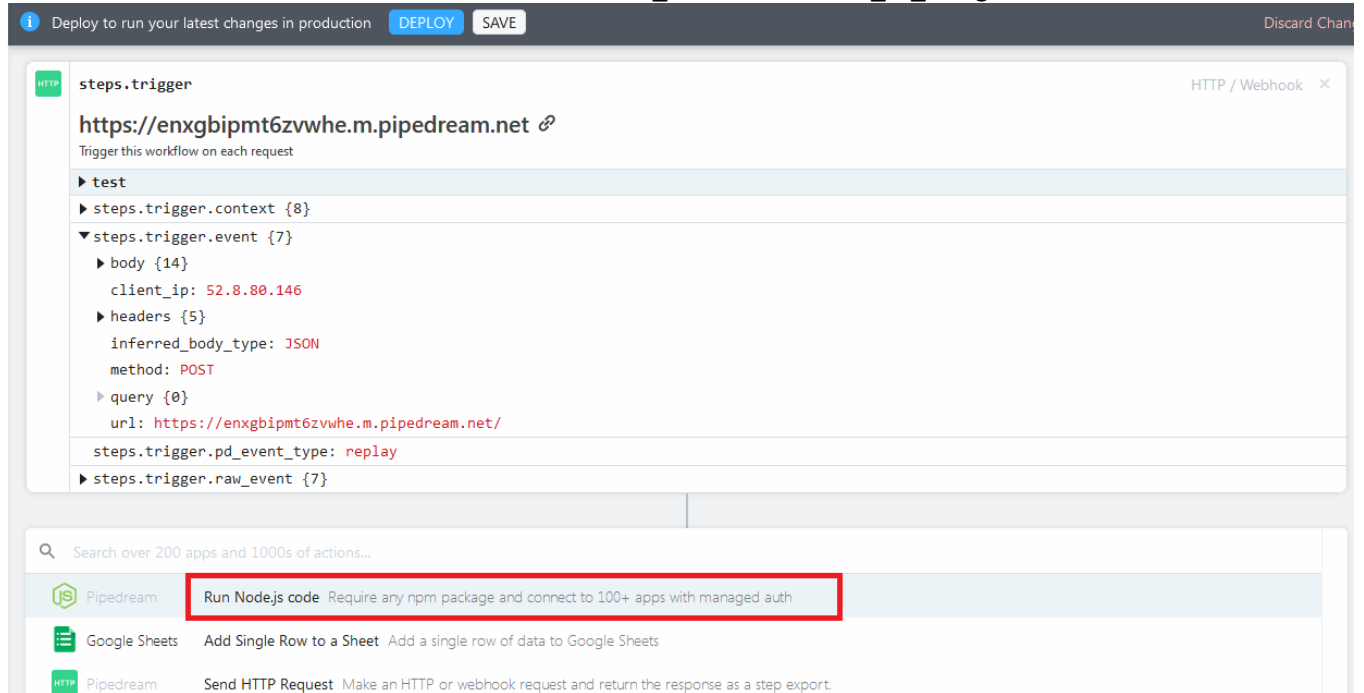
So what we're going to do is to find a Node.js function for a time conversion and one for the temperature conversion. I did that with some simple Google searching and ended up with was the following 2 npm packages:

- Unixtimejs
- Celcius

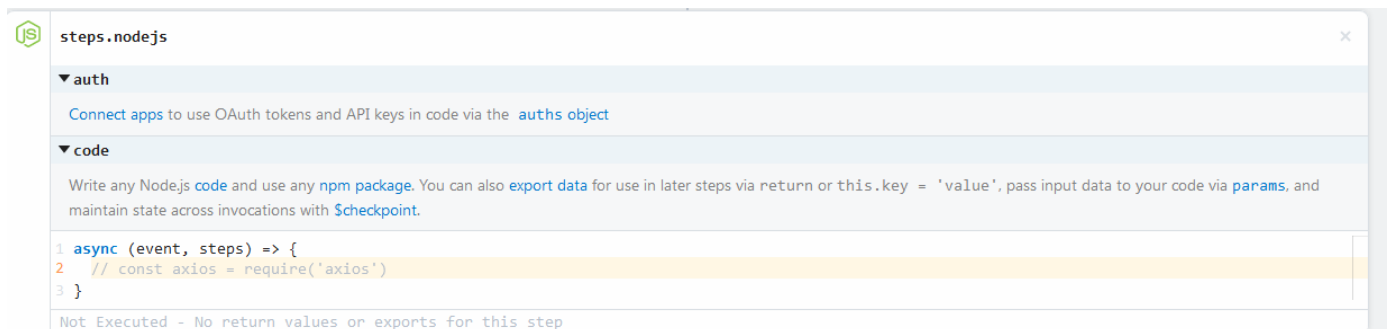
Once you have found your Node.js functions, add a Node.js step after the trigger and create a time conversion, then add a Node.js step after that for the temperature conversion. Finally, go back into your Google Sheet step and update the parameters using the *event.body.reported_at* and *event.body.decoded.payload.temp* to use the output values from the Node.js steps you added so that the converted time and converted temperature are placed into the Sheet.



(above) About to add a new step after the trigger



(above) Select 'Run Node.js code'



(above) New, default, node.js step



(above) Time conversion step

- renamed to 'step.convert_out_of_unix'
- operating on *event.body.reported_at* from the trigger event

```

async (event, steps) => {
  const { toFahrenheitFmt } = require('celsius');

  var Ftemp = toFahrenheitFmt(event.body.decoded.payload.temp); // 97
  console.dir(Ftemp)
  return Ftemp
  //toFahrenheit(36.68, 3); // 98.024
  //toFahrenheit('-40.691 degrees C', 5); // -41.24380
}

```

Not Executed - No return values or exports for this step

(above) Temp conversion step

- renamed to 'steps.C_to_F'
- operating on `event.body.decoded.payload.temp` from the trigger event

Deploy to run your latest changes in production **DEPLOY** **SAVE** Discard Change

auth

Google Sheets (auths.google_sheets): mikedsp@gmail.com [unlink](#)

params

Columns ☒ structured mode: on
Enter the data to insert into each column. Click + to add columns in structured mode, or turn structured mode **off** to enter array of column values as an expression — e.g., `{{[1,2,3]}}`

[0]:	<code>{{steps.convert_out_of_unix_time.\$return_value}}</code>	- +
[1]:	<code>{{event.body.name}}</code>	- +
[2]:	<code>{{event.body.hotspots[0].name}}</code>	- +
[3]:	<code>{{steps.C_to_F.\$return_value}}</code>	- +
[4]:	<code>{{event.body.decoded.payload.rh}}</code>	- +
[5]:	<code>{{event.body.decoded.payload.batt}}</code>	- +
[6]:	<code>{{event.body.hotspots[0].rssi}}</code>	- +
[7]:	<code>{{event.body.hotspots[0].snr}}</code>	- +
[8]:	<code>{{event.body.dc.balance}}</code>	- +
[9]:	<code>{{event.body.dc.nonce}}</code>	- +

ARRAY · params · columns

(above) The time and temperature parameters in the Google Sheet step updated to use the conversion outputs from the Node.js steps

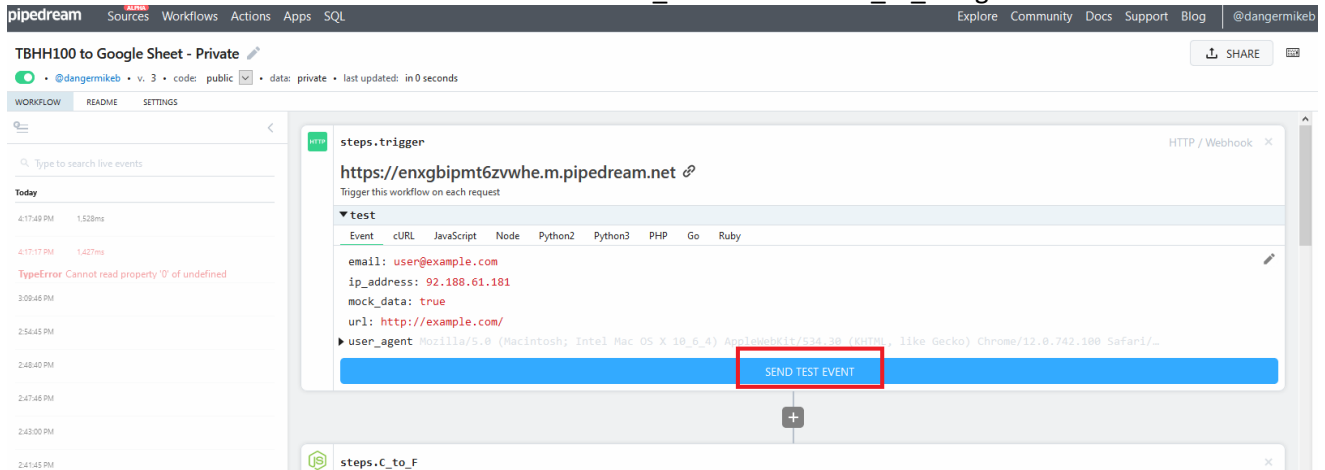
Deploy to run your latest changes in production **DEPLOY** **SAVE** Discard Change

steps.convert_out_of_unix_time

auth

Connect apps to use OAuth tokens and API keys in code via the **auths** object

(above) Deploy your changes



(above) Send a test event

The screenshot shows a Google Sheet titled 'TBHH100 Private' with data from the TBHH100 device. The data includes Date/Time, From Device, Hotspot, Temp, Humidity (%), Battery (V), rssi, snr, balance, and nonce. A red box highlights the 'Temp' column.

	A	B	C	D	E	F	G	H	I	J
1	Date/Time	From Device	Hotspot	Temp	Humidity (%)	Battery (V)	rssi	snr	balance	nonce
2	1598208104	TBHH100	gorgeous-fuchsia-penguin	15	100	3.5	-106	-13	1006479	1
3	Sun, 23 Aug 2020 21:06:43 GMT	TBHH100	gorgeous-fuchsia-penguin	27 °F	20	3.6	-57	12.5	1006474	1
4	Sun, 23 Aug 2020 21:10:44 GMT	TBHH100	clever-orange-mustang	18 °F	23	3.6	-65	13.19999981	1006473	1
5	Sun, 23 Aug 2020 21:32:45 GMT	TBHH100	clever-orange-mustang	0 °F	40	3.6	-82	11	1006472	1
6	Sun, 23 Aug 2020 21:38:46 GMT	TBHH100	clever-orange-mustang	0 °F	45	3.6	-83	11	1006471	1
7										
8										
9										
10										

(above) Google Sheet – with formatted Time and Temp coming in.

Note – at the time I took this screenshot, several additional readings had been sent from the TBHH100 device. The device is sitting in the freezer.

- (optional) Enjoy the sensor data flowing into your Google Sheet. Share the Sheet with a friend. Share the Sheet with the world.

Using Google access controls, you can share your Google Sheet. One sharing mechanism is to Publish your document, which can make the document visible in READ-ONLY mode to anyone with the URL.

To test out the instructions in this document, I created another Pipedream integration, this time sending the data from the same device to a Google Sheet that I have published here:

https://docs.google.com/spreadsheets/d/e/2PACX-1vR94gWpsaT8EjUoQN49DOcQio777sAPBJIL1Gz2nGywLk39j45y73b_dDimFn0HQ4kLOagLYaAfrfq9/pubhtml

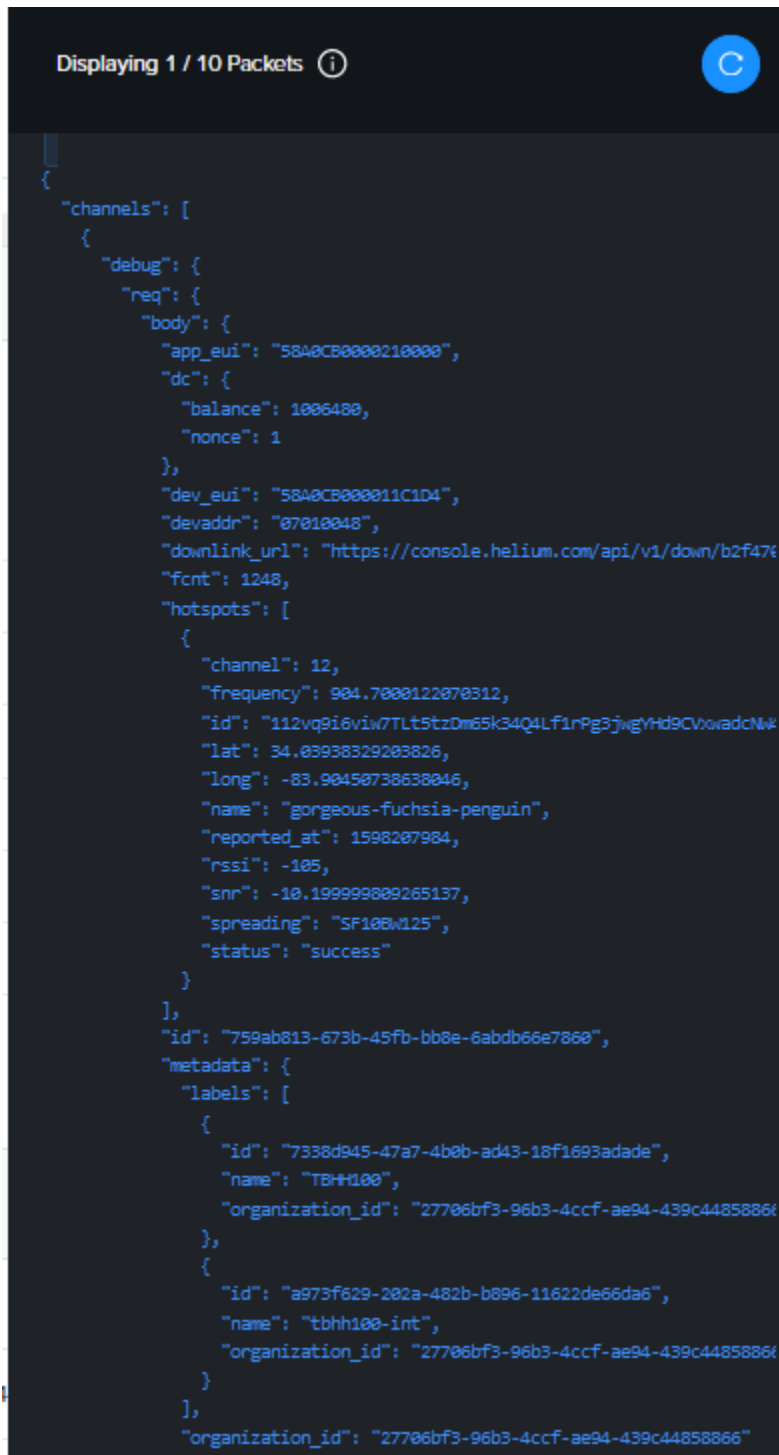
Clicking on the link above should take you to a real time view of the data from my TBHH100 sensor.

Instructions for Publishing a Google Sheet can be found here:

https://support.google.com/docs/answer/183965?visit_id=637338598692878804-1883751722&rd=1

Appendix A – Sample TBHH100 Data from the Helium Console Debugger

The 2 screenshots below show what the data looks like when the Console Debugger captures TBHH100 with a TBHH100 decoder applied to the device.



```
{
  "channels": [
    {
      "debug": {
        "req": {
          "body": {
            "app_eui": "5840CB0000210000",
            "dc": {
              "balance": 1006480,
              "nonce": 1
            },
            "dev_eui": "5840CB000011C1D4",
            "devaddr": "07010048",
            "downlink_url": "https://console.helium.com/api/v1/down/b2f476",
            "fcnt": 1248,
            "hotspots": [
              {
                "channel": 12,
                "frequency": 904.7000122070312,
                "id": "112vq9i6viw7TLt5tzDm65k34Q4Lf1rPg3jwgyHd9CVowadCNW",
                "lat": 34.03938329203826,
                "long": -83.90450738638046,
                "name": "gorgeous-fuchsia-penguin",
                "reported_at": 1598207984,
                "rssi": -105,
                "snr": -10.199999809265137,
                "spreading": "SF10BW125",
                "status": "success"
              }
            ]
          },
          "id": "759ab813-673b-45fb-bb8e-6abdb66e7860",
          "metadata": {
            "labels": [
              {
                "id": "7338d945-47a7-4b0b-ad43-18f1693adade",
                "name": "TBH100",
                "organization_id": "27706bf3-96b3-4ccf-ae94-439c44858866"
              },
              {
                "id": "a973f629-202a-482b-b096-11622de6da6",
                "name": "tbhh100-int",
                "organization_id": "27706bf3-96b3-4ccf-ae94-439c44858866"
              }
            ]
          },
          "organization_id": "27706bf3-96b3-4ccf-ae94-439c44858866"
        }
      }
    }
  ]
}
```



```

{
  "debug": {
    "req": {
      "body": {
        "app_eui": "58A0CB0000210000",
        "dc": {
          "balance": 1006480,
          "nonce": 1
        },
      },
      "decoded": {
        "payload": {
          "batt": 3.5,
          "bytes": [
            8,
            250,
            45,
            100,
            255,
            255,
            255,
            255
          ],
          "rh": 100,
          "temp": 13
        },
        "status": "success"
      },
    },
    "dev_eui": "58A0CB000011C1D4",
    "devaddr": "07010048",
    "downlink_url": "https://console.helium.com/api/v1/down/8b0f1:",
    "fcnt": 1248,
    "hotspots": [
      {
        "channel": 12,
        "frequency": 904.7000122070312,
        "id": "112vq9i6viw7TLt5tzDm65k34Q4Lf1rPg3jwgYHd9CVxwadcNw",
        "lat": 34.03938329203826,
        "long": -83.90450738638046,
        "name": "gorgeous-fuchsia-penguin",
        "reported_at": 1598207984,
        "rssi": -105,
        "snr": -10.199999809265137,
        "spreading": "SF100w125",
        "status": "success"
      }
    ]
  }
}

```

