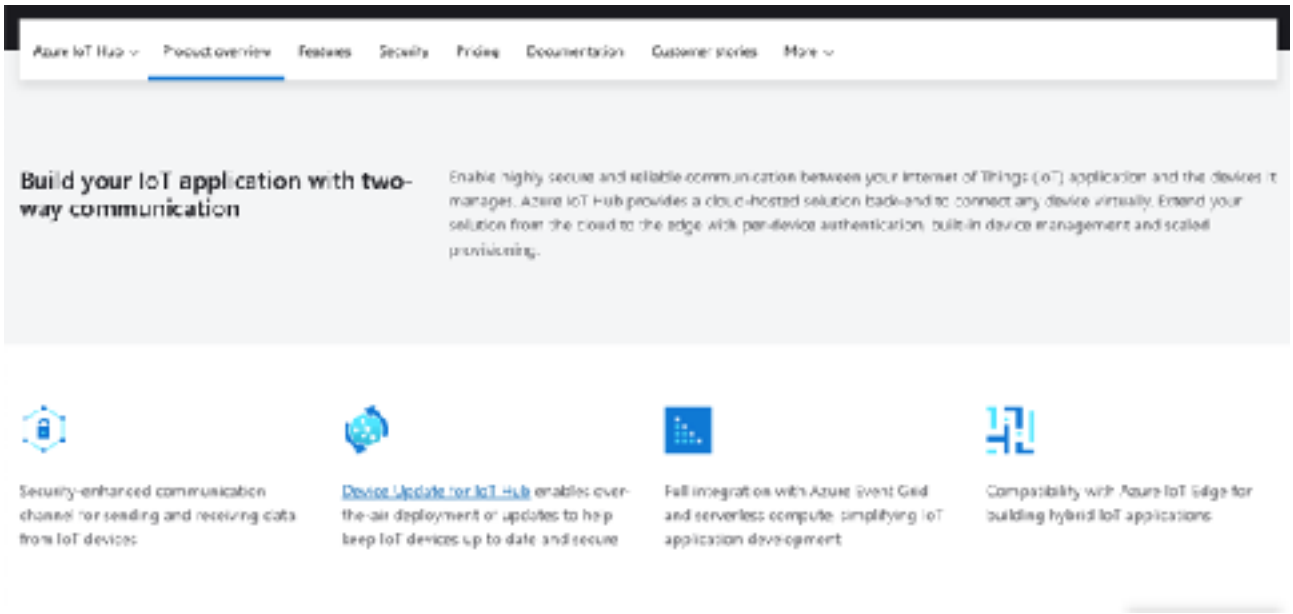
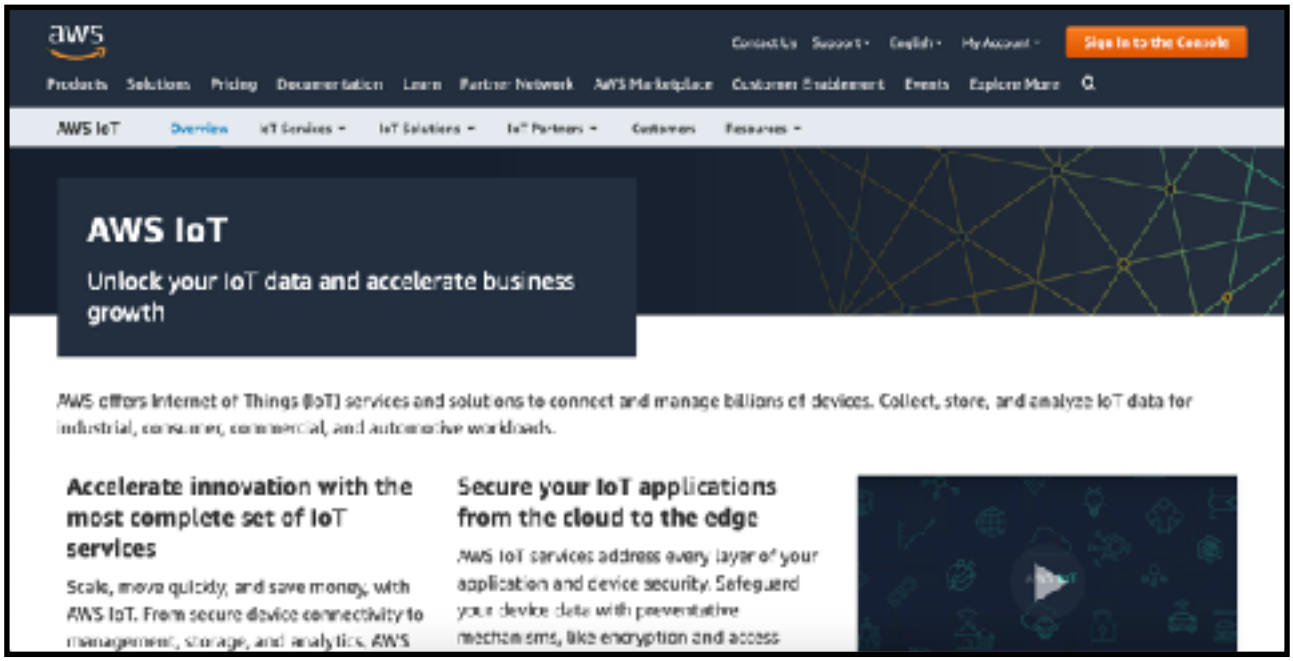
Microsoft Azure



While Microsoft Azure offer a free service, you may find yourself quickly growing out of this free level and getting charged as Azure is aimed for the more business end of the IOT market.

Amazon AWS



Amazon Web Services (AWS)

To use the Amazon Web Services Internet of Things (AWS IOT onwards) you first need to register with the AWS site. If you have an Amazon account then you can now sign in with these credentials.

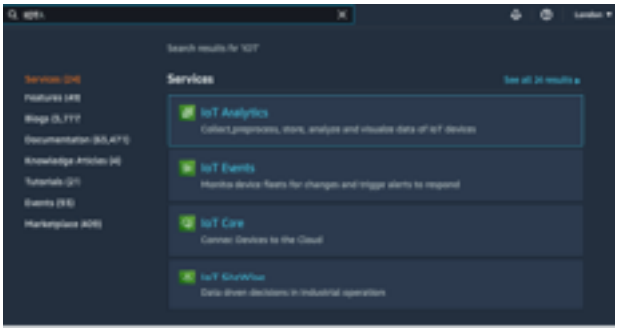
Once you log in and win the fight with captcha, you will be taken to this page:

This page will gave you some getting started guides and a list of costs you may build up once you start using the services.

At the top of the screen next to the name you registered as you will find a location, this must be set your geographically local region or you may get some unexpected finical charges. Another note about the region servers is that not all AWS services are available to all regions.

Next click on the search bar and type in “IOT”

This will bring up a list of AWS services that match the search string:



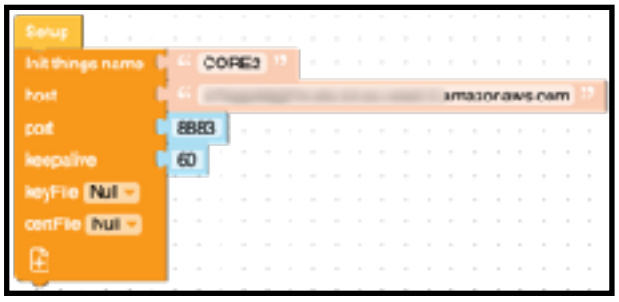
We will be using IOT Core in this guide and so click on IOT Core to be taken to the AWS IOT service.

It is worth looking through the guides and documents in order to understand the IOT service and how to use it before moving on.

In order for a device (or thing as AWS calls them,) to connect to the AWS services we have to create the thing, attach security certificates, polices and rules and then program them into the device. While the AWS Console used to set up the device has changed, the instruction created by M5Stack found here: https://docs.m5stack.com/en/uiflow/iotcloud/aws along with the UIFLOW example is still valid.

There is a little variation between the M5Stack guide and my rolling guide but that is because I hope to clear up some issues that new users may have including describing the various blocks available in UIFLOW.

This is the AWS config block that is used to hold the settings AWS needs in order to allow an M5Stack device to connect and also to call the AWS library into the code for the rest of the blocks to operate.



Init Things Name is used to connect a text block that will hold a name you give to you device. This name must be matched on the AWS IoT page:

The Host block is used to hold the AWS Endpoint that your thing will used to connect to AWS:

Port number has to be set to 8883 according to this AWS page https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html

The **Keepalive block** is used to set a time in seconds that a connection will be kept open between a device and the AWS services. If not set, the the connection between the device and AWS will be closed if there is no messages being set from the device.

The Key file drop down menu is used to select any Private Key files stored on the M5Stack device. This must be set to the Private key file that must be named as **Private.Pem.key.**

The Certfile drop down menu is used to select the device certificate that must be named **Certificate.pem.cert**.

The keys and certificates are generated by AWS when you create you policies needed for a device to connect to AWS services.

The icon on the bottom with the + sign is used to add certificates and keys to the M5Stack controllers however, we can also use UIFlows, file manager dialog to upload keys and certificates to M5Stack controllers. The

UIFlow manage is found in the Icons on the top of the screen on the right hand side of UIFlow.

Clicking on this icon will display the manager dialog.

Click on “Certificates “ and the certificate folder is opened, clicking “Add Certificate” opens the file dialog opened by clicking the “+” icon on the block. In the above image you can see that I cave the Device Certificate, Private key and Public keys uploaded and named as UIFlow expected them.

The Micropython code for this block is as follows:

```
from IoTcloud.AWS import AWS
```

This is just a set of instructions that define setting that code will need to know in order to work.

AWS Start

The AWS start block is used to start AWS dedicated code. This block must be placed outside and before the main programming loop or AWS will think it is some kind of attack and block the repeated service creation calls.

AWS SubScribe

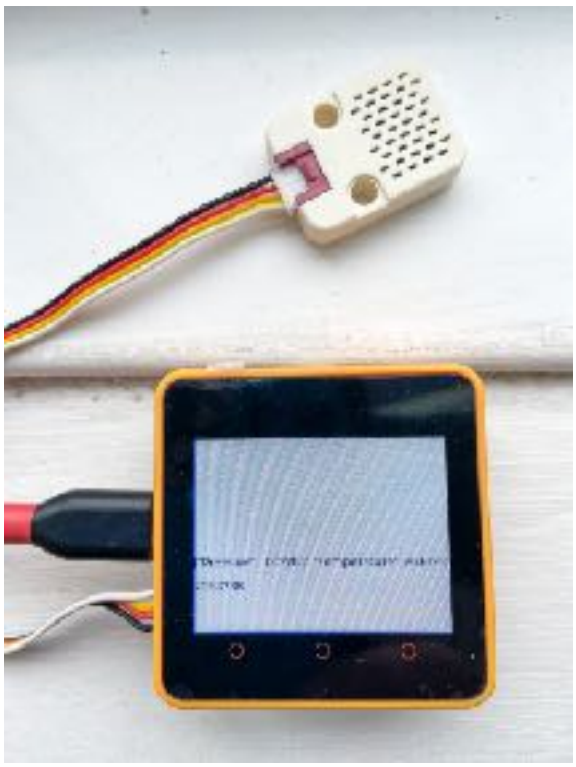
The AWS Subscribe is a standalone function loop that runs independent to our main AWS code once a connection to the AWS IoT service has been established. The Block will attempt to connect to an AWS topic and download data from it.

The Above screenshot is the topic the AWS IoT test server used by the demo I will share later in this section.

Get Topic Data

The Get Topic Data block is used within the AWS Subscribe function loop to allow an M5Stack controller to read data sent to the AWS IoT server.

In this example I am using a label to display data from the AWS IoT topic on the M5Stack screen. In the following photo you can see how the data is displayed on the screen of an M5Stack.



Publish Topic

The Publish Topic block is used to send data to an AWS service. The first space is for a text block containing the topic and the second space is for the data. Data can be plain text but its better if it is JSON formatted data for example:

This is another snippet from the example that just shows how the Publish Topic block works.

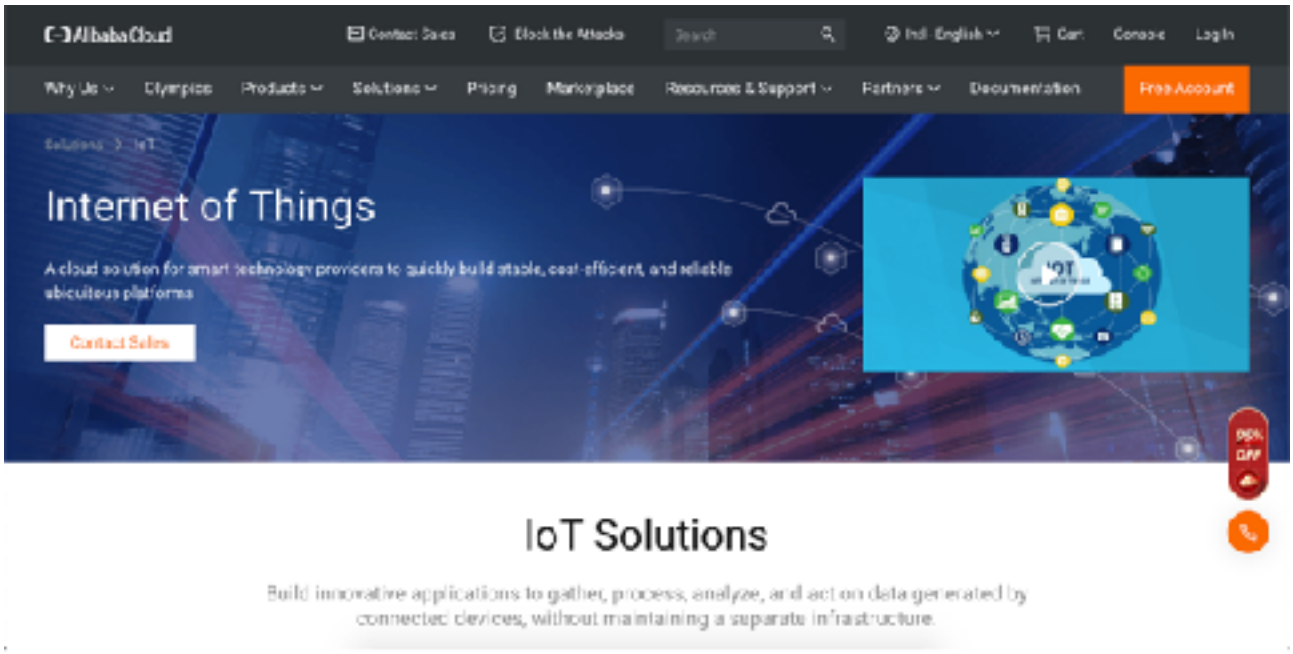
This is the complete program for reading data from the M5Stack ENVIII Unit and sending it to AWS IoT.

The Micropython code for this is shown on the next page.

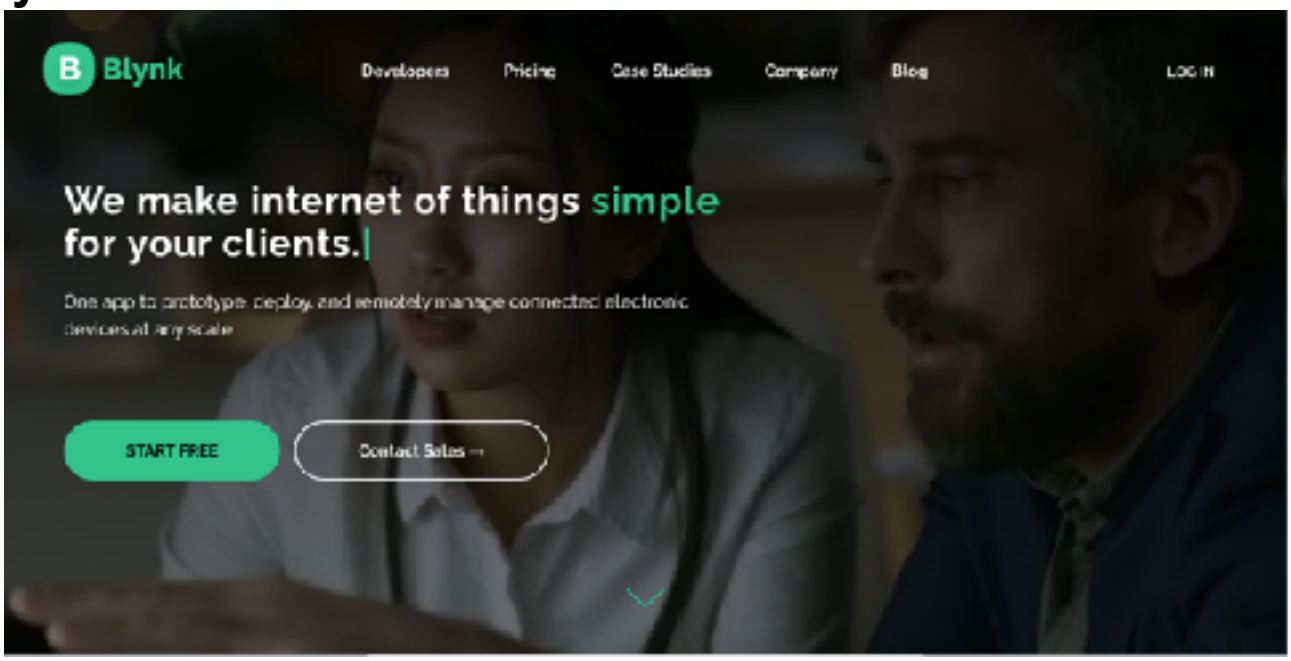
```
from m5stack import *
from m5stack_ui import *
from uiflow import *
from IoTcloud.AWS import AWS
import json

import time
import unit

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)
env3_0 = unit.get(unit.ENV3,
unit.PORTA)
```

Tencent

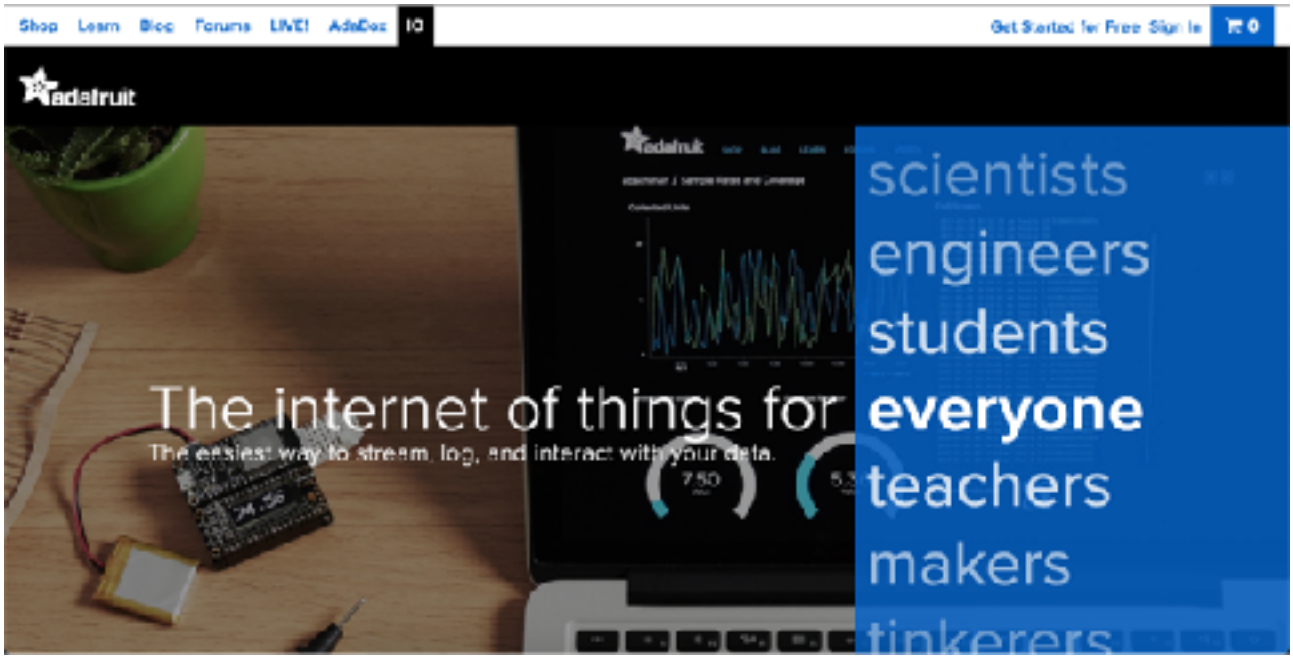
AliIoT



Blynk



While users are able to connect devices to and interact with the Blynk service, Blynk offers a phone app to interface with the web services and IOT devices but also provides the code so that you can set up a private Blynk server on a Raspberry pi.



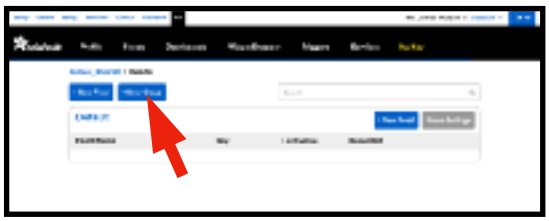
Adafruit IO

To. Use the Adafruit IO servers you need to register for a free account or if you are already registered on Adafruit website you just sign in.

Once you register or sign in you will be taken to the profile page:

In order to send data to io.adafruit.com you need to create a feed. Click on Feed and you will be taken to the Feed page:

Next click on “View All” to be taken to the feed editor:



This page shows that there are no feeds and so we need to click on “New Feed” to create one:

Enter a name for the feed and give the feed a description. Click on “Create” and you will be returned to the feed screen with the new feed listed:

Now that we have created a Feed, weed need to create a dashboard to show the feed. To do that, click on “Dashboard” to open the Dashboard Page:

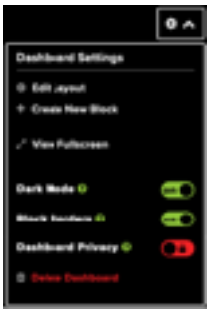
Like with the flow page, this shows that there is no dashboard set up and so we need to click on “View All” in order to get the dashboard editor:

Click on the “New Dashboard” button to open the dashboard dialog and give it a name and description:

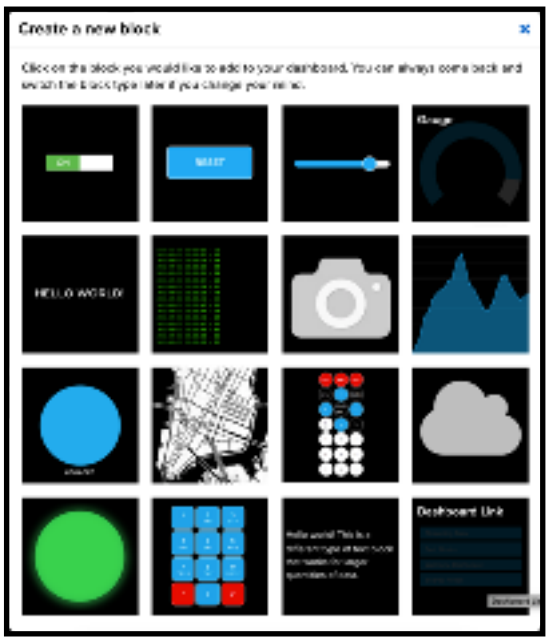
Click on the create button and we return to the dashboard screen showing our newly created dashboard:

Click on the newly created “Environmental Dashboard” and the dashboard editor will open:

Click on the gear shape to open the dashboard menu:



Now click on the “Create New Block” and the block window will open:



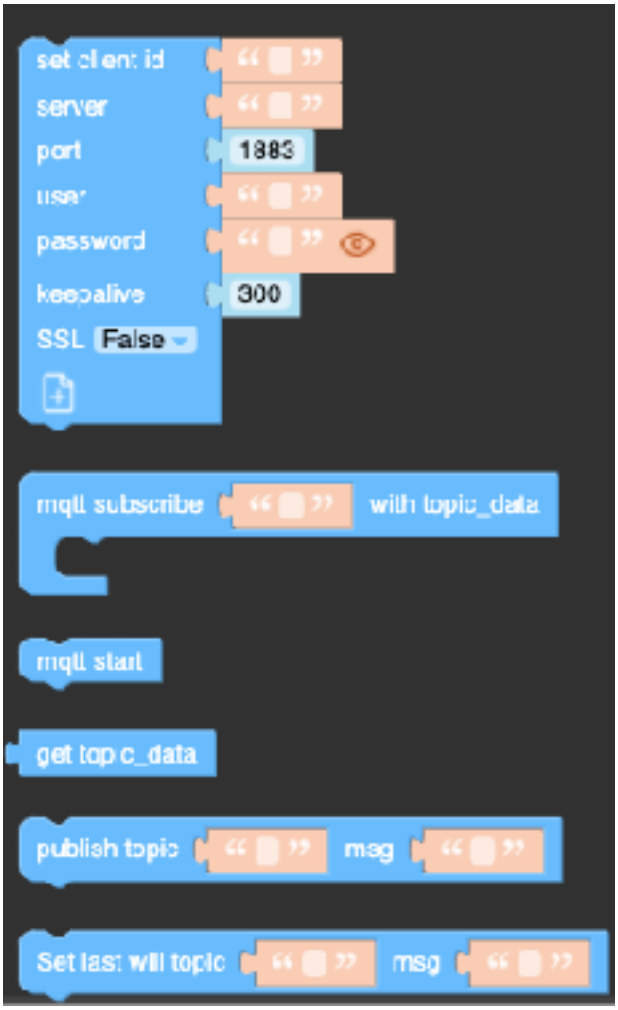
Click on the “Stream” block and the next window will ask you to chose a feed:

Click on the box next to “ENV Reading”, the button on the bottom will now change to “Next Step” Click on it and the settings window will appear:

Leave all the setting as they are and just click on “Create Block” and we will be returned to the dashboard screen showing the blank block ready to receive data that we send to it.

That is everything we need to set up on the io.adafruit.com side of thangs and so now we can move on to the M5Stack UIFlow code side.

To send data to io.adafruit.com we will use the MQTT blocks found in UIFLOW.



The first block contains settings that MQTT needs in order to connect. These settings are as follows:

Set Client ID need to be a unique name for each device otherwise, if you try to reuse the Client stream the stream will be immediately disconnected preventing both devices from access the stream.

Server is io.adafruit.com.

Port is set to 1883 by default,

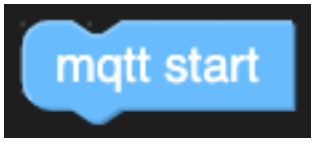
User is the user name you used to registered adafruit.io

Password in Adafruit’s case is the key you download from the key window

Keep Alive is used to maintain a connection to a MQTT service. Some services will automatically close the connection between an IOT device and service if no data is received after a short time.

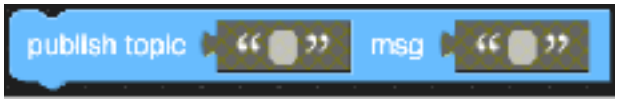
SSL is used if the IOT service provides SSL encryption certificates and keys. Unfortunately io.adafruit.com doesn't provide these (or I'm yet to find them) and so this is set to false.

Next we need to add the MQTT Start block:



This must be placed before the main loop otherwise the program will continuously attempt to make a connection and block access to the server.

Now that the setup is complete, it is time for the code that will capture data and send it to the feed. This consists of a publish block:



The Publish Topic space is used to contain the feed address (which in the example is **James_Purcell/feeds/m5test**).

The MSG box is where we place in the data we want sent to the feed. In the example that follows, all I did was used the Get ENV3.0 Temperature block.

There is a limit to how much data can be sent to io.adafruit.com and so we need to add a Wait block of 5 seconds to prevent data overload. This limit is only on the free service and if you pay, this limit is removed.

A step that is not essential but I use to monitor the data being sent is to add a Label block with the Get ENV3.0 Temperature block inside it to show the temperature readings being sent on the M5Stack screen.

And that is all for the basic demo, below is the complete UIFlow code and the matching UIFLow code:

```
from m5stack import *
from m5stack_ui import *
from uiflow import *
from m5mqtt import M5mqtt
import time
import unit

screen = M5Screen()
screen.clean_screen()
screen.set_screen_bg_color(0xFFFFFF)
env3_0 = unit.get(unit.ENV3, unit.PORTA)

label0 = M5Label('Text', x=12, y=27, color=0x000,
font=FONT_MONT_14, parent=None)
label1 = M5Label('Text', x=-11, y=135,
color=0x000, font=FONT_MONT_14, parent=None)

m5mqtt = M5mqtt('Core20101', 'io.adafruit.com',
1883, 'James_Purcell',
'b596ce7231ac6ddd9d136ca1e4950c9c345a0b52', 300)
m5mqtt.start()
while True:
  m5mqtt.publish(str('James_Purcell/feeds/
m5test'),str((env3_0.temperature)))
  label0.set_text(str(env3_0.temperature))
  wait(5)
  wait_ms(2)
```