

IoT With M5Stack and UIFlow - Volume 1

UIFlow Version 1.9.X

Written By Adam Bryant.



M5STACK



AWS IoT

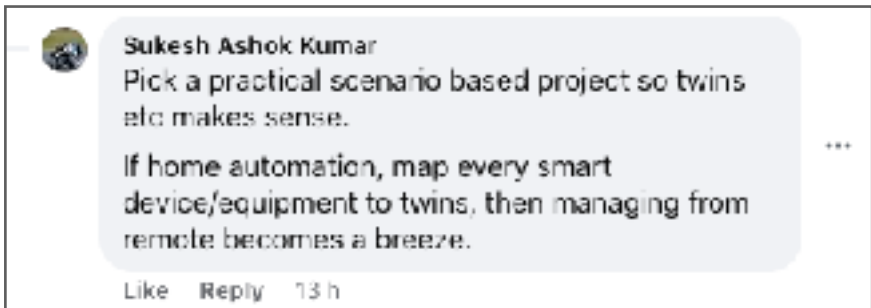
IoT With M5Stack and UIFlow - Volume 1	1
Introduction.	1
What Is IoT?	2
What devices produced by M5Stack can be used for IoT?	2
What software is needed for IoT?	2
What hardware will I be using for this guide?	3
M5Stack Controllers.	3
Core2/ Core2 AWS	3
M5Stack Sensors.	4
ENV Sensor	4
Watering Unit	6
Soil Moisture Sensor.	8
The Scenario.	10
Data Collection.	11
Security Features	12
Digital Twin	13
M5Stack EZData (Part 1)	15
Introduction to M5Stack EZData.	16
Save Value to Topic With Token.	16
Get Value from Topic with Token	19
Remove Topic	20
Save Value to List	20
Get Value From List	22

Get Current ISODateTime	24
Microsoft Azure	27
Introduction.	28
Azure IoT UIFlow Blocks.	40
IoT Hub SAS Authentication	42
IoT Hub X.509 Authentication	42
IoT Central Connection Block.	43
Azure Start	44
Subscribe C2D Message	44
Publish D2C Message	45
Subscribe Direct Method	45
Subscribe Twin Desired Payload.	46
Update Twin Reported Properties.	46
Retrieve Twin Properties.	47
Creating an IoT Central Applications.	48
Amazon Web Services (AWS)	71
Introduction.	72
AWS Start	85
AWS SubScribe	85
Get Topic Data	85
Publish Topic	87
MQTT	91
Introduction.	92

MQTT Credentials	93
MQTT Start	94
Publish to Topic with Message	95
MQTT Subscribe with Topic Data	95
Get Topic Data	96
Set Last Will Topic Message	96
Adafruit IO	99
Introduction.	99
Useful Forms.	114
Access Key List.	116
In Closing.	117

Introduction.

When I started writing this Internet of Things guide (hereafter abbreviated to **IoT**), I was only concerned with understanding the functions available from within UIFlow however, after fighting with Microsoft Azure and a general lack of understanding of how the IoT functions worked, Facebook member Sukesh Ashok Kumar came up with the following suggestion:



Taking this comment to heart, I decided that I would use my greenhouse electronic project as the scenario for this IoT guide and so you will see repetition of the project setup for giving examples on the various functions.

Thank you Sukesh for giving me a sense of direction for this guide.

What Is IoT?

Internet of Things (IoT) is a collection of hardware and software platforms that work together to allow real world devices to connect to each other over the Internet in order to send data and commands to each other.

What devices produced by M5Stack can be used for IoT?

Surprisingly most of the M5Stack controllers from the Core 2 series to the ATOM and Stamp Series can be used for IoT as long as they have access to the internet via wifi. Not only can the controllers be used for IoT but some units now use the ESP32 microcontroller used in the main M5Stack Controllers.

What software is needed for IoT?

IoT software is split into two main categories, Firmware and Remote Services.

Firmware (in the case of this book) is created in UIFlow and uploaded to the controllers and Remote Services is software installed to servers that can be in the same wifi network on servers hosted in the cloud (internet for us oldies).

What hardware will I be using for this guide?

Throughout the guide you will see me using the same hardware for the various projects. This hardware is mostly purchased from M5Stack and will have links to the products on the Official M5Stack store. Where products are not made by M5Stack I will provide a link to the Amazon UK shop as an example location to buy them.

M5Stack Controllers.

Core2/ Core2 AWS



Here I will mostly be using the Core2 AWS as my Core2 is in use on other projects. I am recommending the Core2 series as they have more ram than the Core, Stick, Atom and, stamp series but I have just managed to get the Atom to run on AWS IoT services.

To Purchase a Core2:

<https://shop.m5stack.com/collections/m5-controllers/products/m5stack-core2-esp32-iot-development-kit?ref=pfpqkvphmgr>

To purchase a Core 2 AWS:
<https://shop.m5stack.com/collections/m5-controllers/products/m5stack-core2-esp32-iot-development-kit-for-aws-iot-edukit?ref=pfpqkvphmgr>

M5Stack Sensors.

ENV Sensor

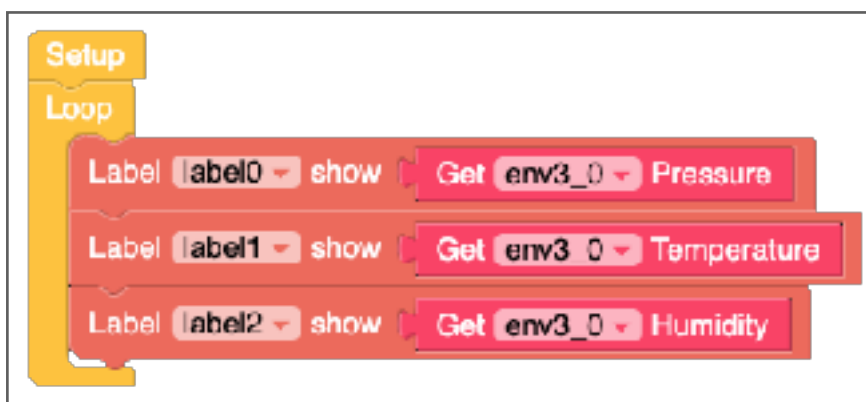


While you can use any of the ENV sensor versions, I am using the ENV3 Unit and it was brought just for this series of projects. The ENV sensor is connected to port A of the Core2 AWS.

In UIFlow there are three functions available that return the Temperature, Pressure and humidity.



In order to use these blocks we need to create a simple script that contains three labels to show the values as shown below.



This is only a simple example and later in the book I will show you how to extend this example for the various IoT services.

To purchase the ENV Sensor from M5Stack, please visit:
<https://shop.m5stack.com/collections/m5-sensor/products/env-iii-unit-with-temperature-humidity-air-pressure-sensor-sht30-qmp6988?ref=pfpqkvphmgr&variant=40187936309420>

Watering Unit



The Watering Sensor consist of a small water pump and capacitive moisture sensor built into one unit that is connected to the Cores using Port B of the Core2's base. The pump comes with two short hoses but the inlet and outlet is compatible with 4.6mm (3/16") irrigation hoses and accessories from companies like Gardena, Hozelock and others that uses the 4.6mm (3/16") irrigation hose size.

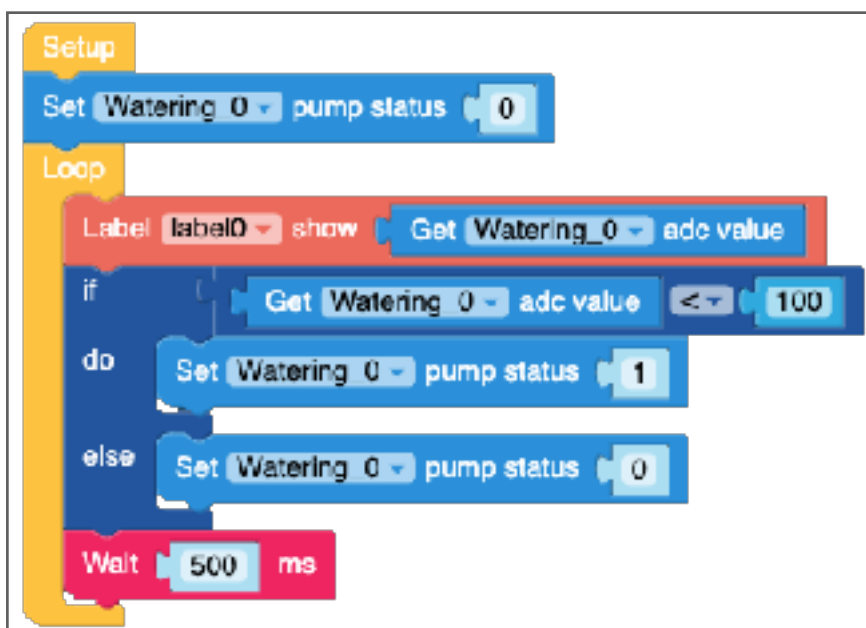
In UIFlow the Watering Unit has 1 value block and two function blocks:



We can use code similar to the ENV3 example to read the moisture sensor on the unit:



However, to control the pump we need a few more blocks.



To purchase the Watering Unit from M5Stack, please visit:
<https://shop.m5stack.com/collections/m5-sensor/products/watering-unit-with-mositure-sensor-and-pump?ref=pfpqkvphmgr>

Soil Moisture Sensor.



While the Soil Moisture sensor could be used in the projects it is not recommended. These sensors are resistive and have exposed copper in order to sense moisture levels. This exposed copper means that they have a short life as the copper gets corroded.

If you wish to use this in UIFLOW, the unit only has two blocks that return values:



And we can use code similar to the ENV sensor example code:



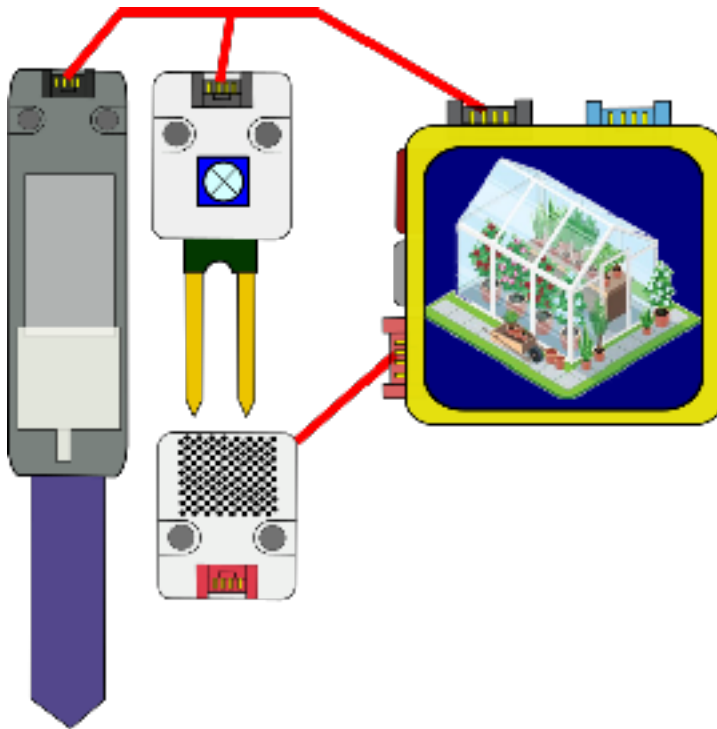
To purchase the Soil Moisture Sensor from M5Stack, please visit:

<https://shop.m5stack.com/collections/m5-sensor/products/earth-sensor-unit?ref=pfpqkvphmgr&variant=16804783882330>

The Scenario.

The scenario that I will be using for all the IoT examples is that the Core2 AWS will act as the Greenhouses IoT Hub, the sensors and pumps will connect to the Core2 AWS that will send the data to and from the sensors to the IoT services and react on preset events triggering the pumps.

Below is a simple diagram of how the hardware is connected together.



Data Collection.

In order to understand the data collected by the Core2 that is sent to the IoT services, the information needs to be separated in to data categories.

These categories are as follows:

Telemetry - Telemetry is a stream of values sent from the device, typically from a sensor.

Properties - Properties represent point-in-time values.

Commands - You can call device commands from IoT Central.
Commands optionally pass parameters to the device and receive a response from the device.

For this project to work I need to declare telemetry values that will need to be collected will be

- Wartering Unit Moisture,
- ENV Sensor Temperature,
- ENV Sensor Humidity,
- ENV Sensor Pressure,
- Earth Sensor Analogue,
- Earth Sensor Digital.

While we could combine the data into one stream using JSON, this causes issues with data logging and graphing. For this book I will keep the data separated into their own stream (or feed as some services call them) in order to better track the data and create charts of the data.

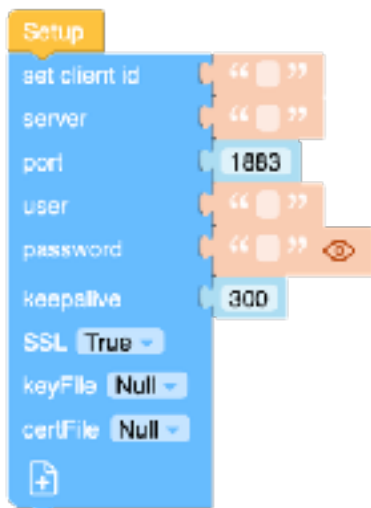
In addition to the telemetry values I will also need to define commands for:

- Watering Unit Pump,
- Core2 LED Colours.

In each of the services I will show you how to collect the telemetry data and control the pumps and LEDS.

Security Features

In the old days of computing all you needed to secure an IoT service was a username and password. As technology has moved on, the user name and password has become insecure and prone to being hacked. While some less important service providers still allow simple User name and password access (like M5Stack test IoT service EZData) most have moved on to use User encrypted keys and certification. To use Keys and certification with the MQTT blocks you need to make sure that the SSL dropdown box is set to true.



And the Key file and cert selection box's become accessible to select pre saved Keys and certificates.

Microsoft Azure on the other hand used a individually generated string with is connected to a virtual Digital twin.

Digital Twin

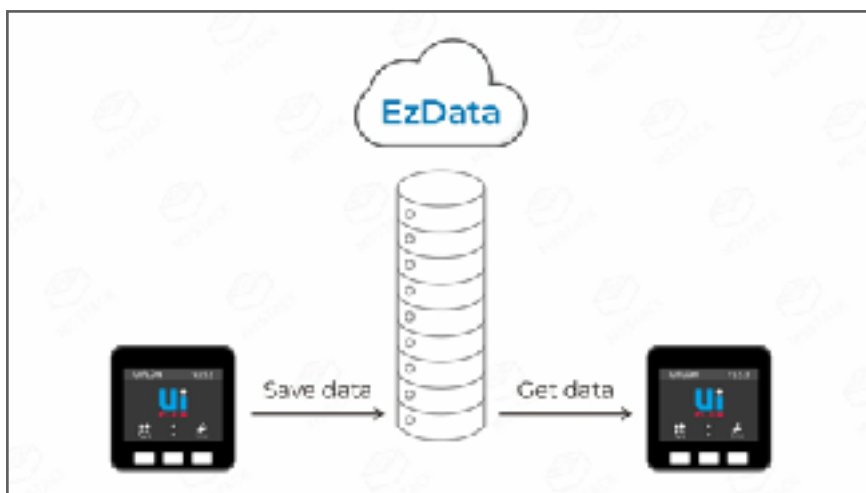
Digital twins are virtual representations off a real world device that live on an IoT services web server. The digital twin tell the services what data is being generated by the IoT device, how to categorise the data and how to process the generated data.

A idea of the digital twin can take some time to understand but once you grasp its use then designing IoT services become easier to understand.

M5Stack EZData (Part 1)



M5STACK



Introduction to M5Stack EZData.

EZData is an IoT solution provided by M5Stack using their own servers that allows us to send data from one device to another device via the cloud and an internet connection.

Save Value to Topic With Token.

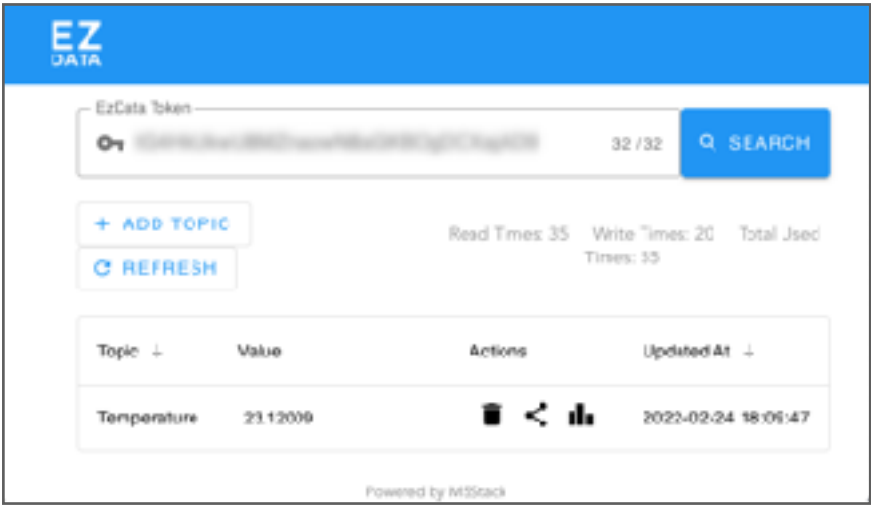


The Save Value to Topic with Token block is used to send data from an M5Stack devices to the EZData service instance defined with the random generated token. The token is generated in UIFlow and automatically placed in the block when the block is added to a program.

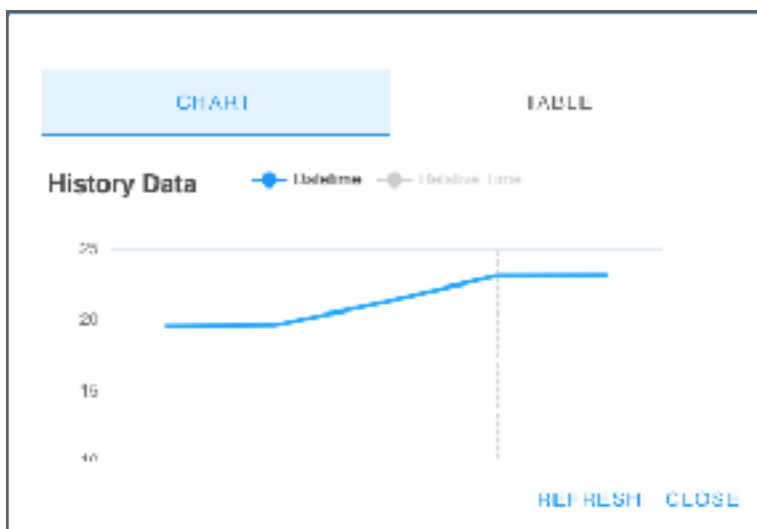
The following example retrieves the temperature from the ENV3 sensor Unit and writes it to the topic Temperature on the EZData server.



When run with a valid token the results on the EZData server is as follows:



Clicking on the bin icon will delete the topic, the arrow icon creates links to the data that has been sent to the server and the Bar chart icon shows a chart of readings:



While clicking on “Table” shows the transmitted data also know as telemetry:

CHART TABLE

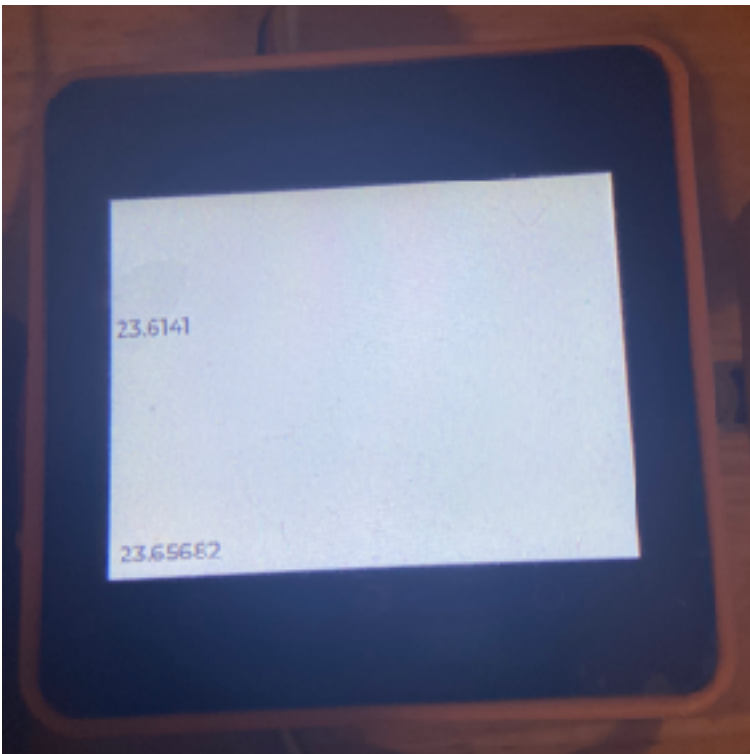
Datetime	Value
2022-02-24 18:09:50	23.14946
2022-02-24 18:09:47	23.12009
2022-02-24 17:57:17	21.25887

REFRESH CLOSE

Get Value from Topic with Token



The Get Value from Topic with Token block is used for retrieving data from the EZData service. Unlike the Save Value block which is a function, the Get value requires additional blocks to work like a label block. The following example uses another Label to retrieve data from the EZdata topic temperature.



Remove Topic

In order to remove a topic from EZData you can click on the Bin icon in EZData page or you can use the following block in UIFLow:



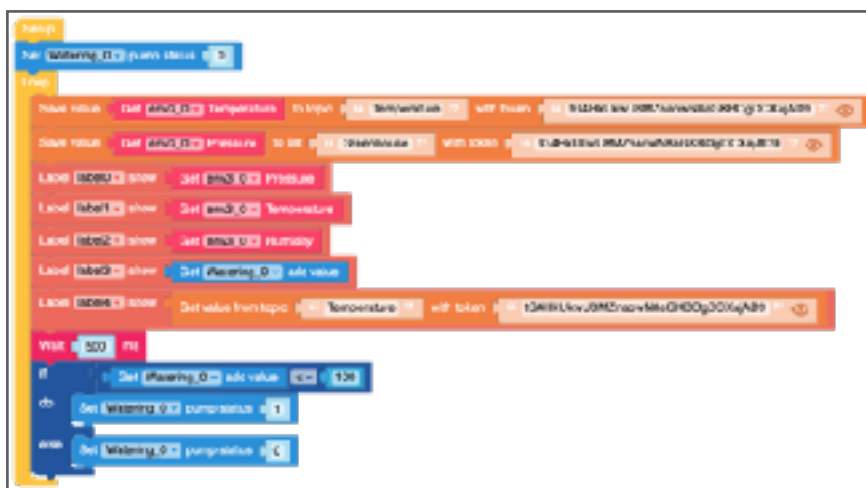
The Remove Topic block takes the name of the the topic you want to remove and the token that points to the individual server instance that is assigned to you.

Save Value to List

So far I have just shown how to write, retrieve and delete single entries. Next I will show you how to work with lists in EZData. To start using lists we need to create one using the following block:



If we add the Save Value To List block to the existing code:



We get the following in EZData:

EZ DATA

EzData Token

1G4HkUkwU8MznacwV8aGKB0gDCXajAD9

32 / 32

SEARCH

+ ADD TOPIC

REFRESH

Read Times: 44

Write Times: 20

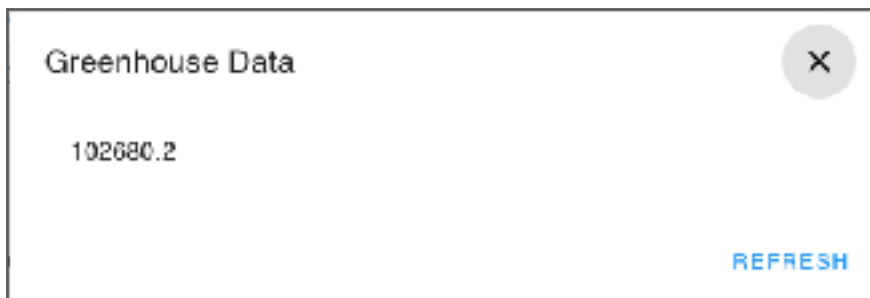
Total Used Times: 72

Topic ↓	Value	Actions	Updated At ↓
Greenhouse	View List +		2022-02-26 10:14:26
Temperature	24.71427		2022-02-26 10:14:21

Powered by M5Stack

Clicking on “View List” brings up the following:

21



Unfortunately, if we try to add the other readings from the ENV3 to the list, the Core2 will crash without an error and only the first entry sent to the list will appear.

A second issue with the list is that for some reason when the “List block” is used, the “Get Value from topic block” fails to retrieve any data.

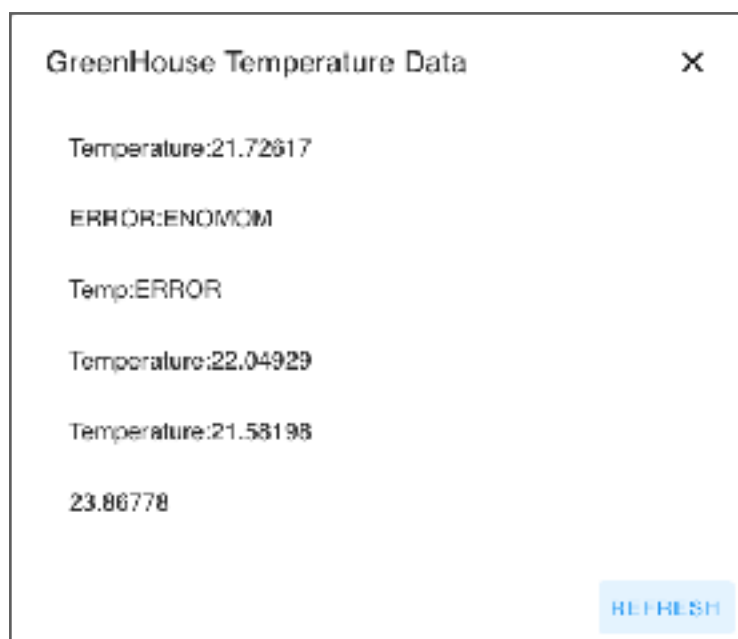
[Get Value From List](#)

To retrieve an entry from a list we use the “Get Value From List” block:



The “Get value from list” is used to define which list you want to retrieve data from, “Offset” is used to define which entry in the list you want retrieved, “Count” is how many items you want from the list after the initial entry and “With Token” is the EZData instance assigned to the UIFlow session.

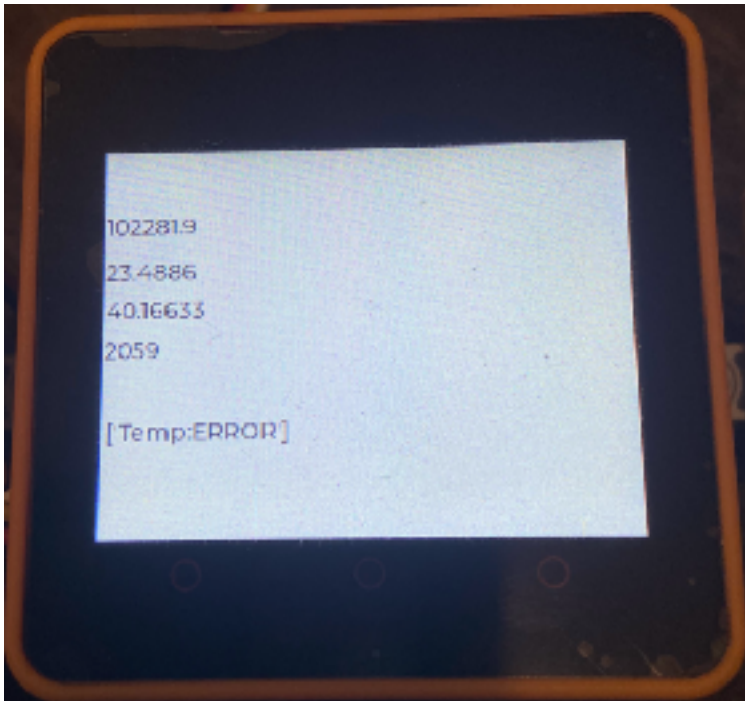
In the following list example I added some random entries to the list to make it easier to see:



When I run the following modified example:



The Core2 AWS display will show the third entry because the offset starts at 0 and not 1.



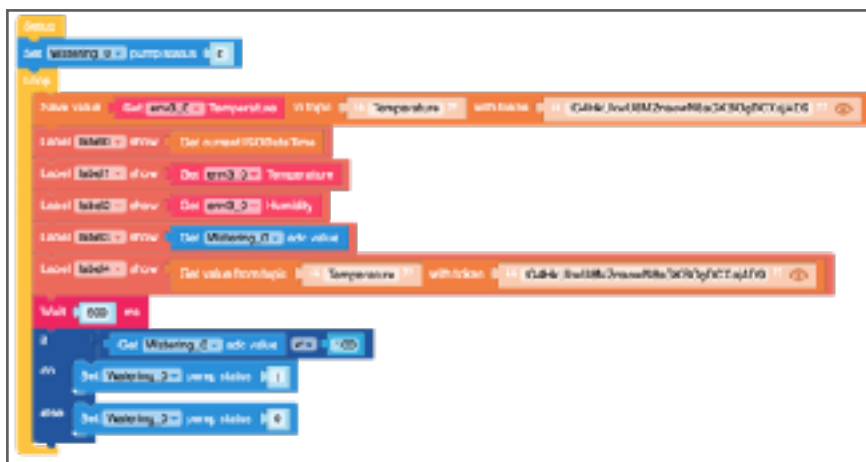
[Get Current ISODateTime](#)

The last block to be found in the first section of EZData block is the “Get Current ISODateTime” block.

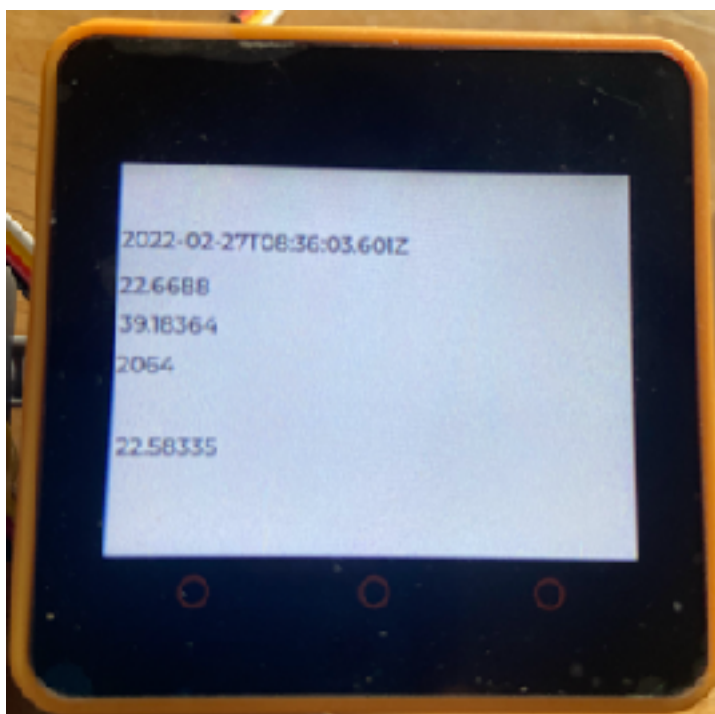


The “Get Current ISODateTime” retrieves the timestamp for the EZData instance clock supplied by the servers own clock.

In the following example I have replaced the cable that was showing the pressure with the “Get Current ISODateTime”:



Which when run shows the current server time.



Microsoft Azure

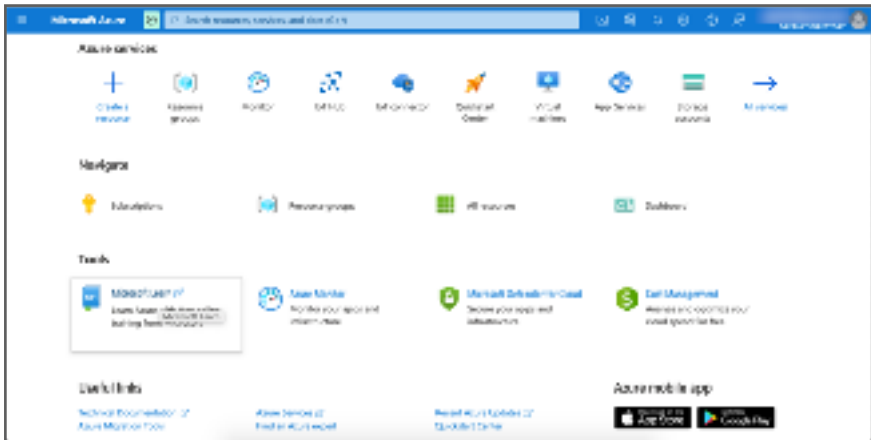


Introduction.

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.

Azure is an exercise in utter frustration if you are only getting started and the M5Stack FB group has been on the receiving end of my many frustrations.

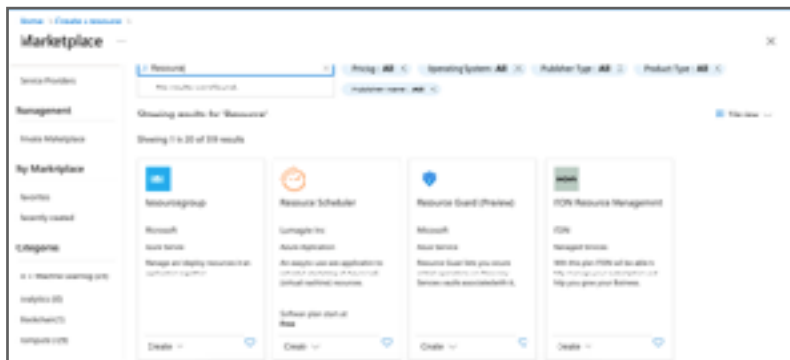
In order to get started with Azure, you need to register. If you haven't already, follow all the on-screen prompts to register and then you will end up at the dashboard



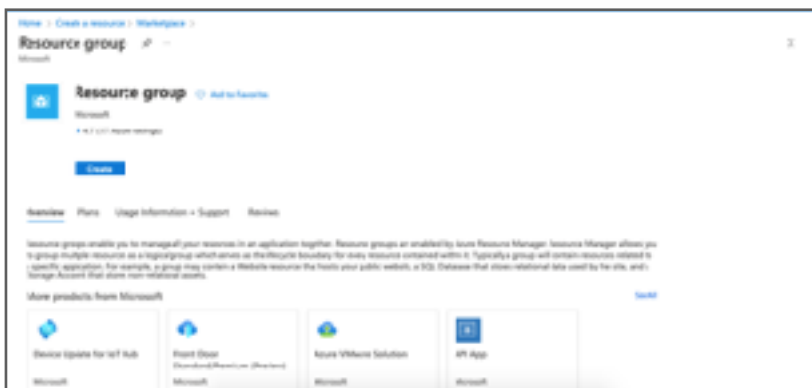
In order to get started we first need to create a resource. Click on the “Create a Resource” and the next screen will appear.



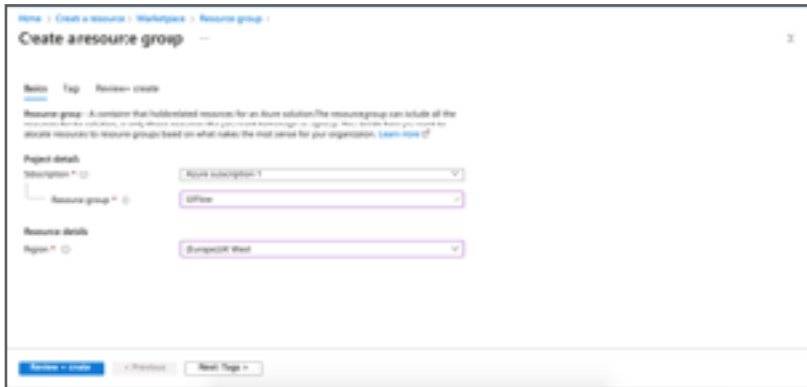
Type Resource into the search box and hit enter.



Click on Resource,



Click on the “Create” Button,



Home > Create a resource > Marketplace > Resource group

Create a resource group

Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources that are related to a single solution, or you can create separate resource groups for different solutions. You can create resource groups based on what makes the most sense for your organization. [Learn more](#)

Project details

Subscription * (v) Azure subscription 1

Resource group * (v) offshore

Resource details

Region * (v) Europe (UK West)

[Review + create](#) [+ Previous](#) [Next: Tags >](#)

Give the resource a name and select the region of the world you are in (in my case I’m in the west of the UK and so chose UK West). Next click on “Next: Tags” .



Home > Create a resource > Marketplace > Resource group

Create a resource group

Review + create

Tags

Apply tags to your Azure resources to logically organize them by categories. A tag consists of a key (name) and a value. The names cannot contain slashes and tags are case sensitive. [Learn more](#)

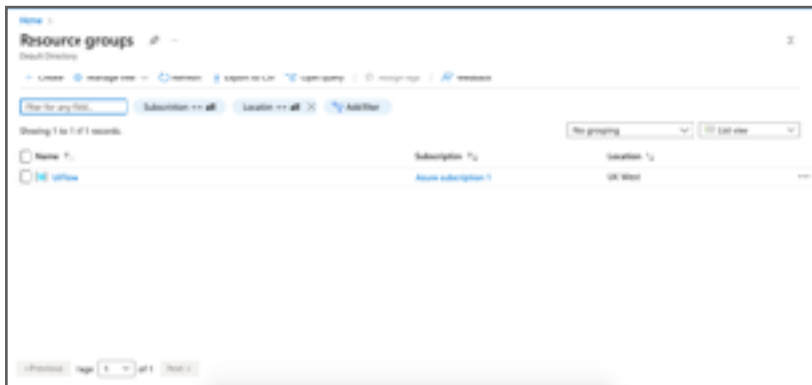
Name	Value	Resource
offshore-uk-west	offshore-uk-west	Resource group
		Resource group

[Review + create](#) [+ Previous](#) [Next: Review + create >](#)

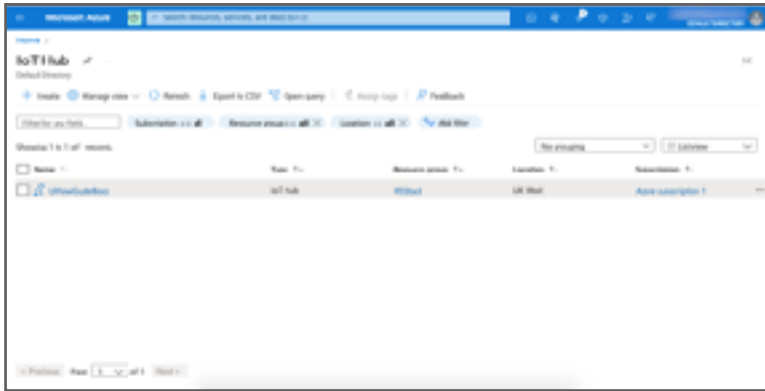
Click on the empty box’s and select the items that appear as default and then click on the Review create button.



Double check the setting are correct and then click on “Create”.

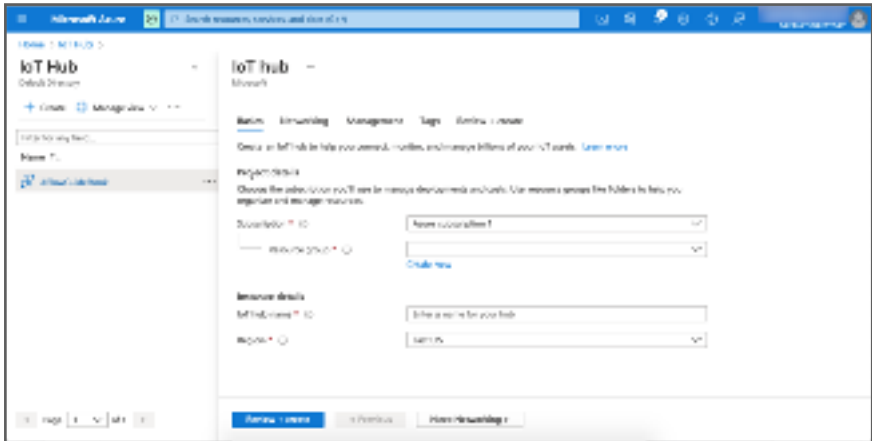


Now that we have created a resource group we can now create an IoT Hub. Click on the IoT Hub Icon to be take to the IoT Hub page.



This page is normally empty as there won't have been a hub created yet.

Click on the “Create” button and the hub creation window will open.



Leave “Azure subscription 1” box as it is, click on the dropdown arrow to select a resource group, then click on “Next: Networking”.



Leave this as Public and click “Next:Management”.

Home > IoT Hub >

IoT hub

Microsoft

Basics Networking **Management** Tags Review + create

Each IoT hub is provisioned with a certain number of units in a specific tier. The tier and number of units determine the maximum daily quota of messages that you can send. [Learn more](#)

Scale tier and units

Pricing and scale tier *

F1: Free tier

[Learn how to choose the right IoT hub tier for your solution](#)

Number of F1 IoT hub units

0

Determines how your IoT hub can scale. You can change this later if your needs increase.

Defender for IoT

Off

Microsoft [Defender for IoT](#) is a separate service which adds an extra layer of threat protection for Azure IoT Hub, IoT Edge, and your devices. You will be charged separately for this service. Defender for IoT may process and store your data within a different geographic location than your IoT Hub. [Learn more](#)

Scaling and scaler tier	F1	Device-to-cloud messages	Enabled
Messages per day	8000	Message routing	Enabled
Cost per month	000 USD	Cloud-to-device commands	Enabled
Defender for IoT	Disabled	IoT Edge	Enabled
		Device management	Enabled

Role-based access control

Change the permission model to Azure role-based access control (RBAC) only, or to a combination of shared access policies and RBAC. [Learn more](#)

☐ RBAC only

☒ Shared access policy + RBAC

To manage the elements within an instance, a user needs access to IoT Hub data/APIs. Select the suggested role below to grant yourself full access to the APIs. You can also use Azure Control (IAM) to choose appropriate roles later. [Learn more](#)

☐ Assign me to the IoT Hub Data Contributor role

Advanced settings

Scale

Device-to-cloud partitions

0

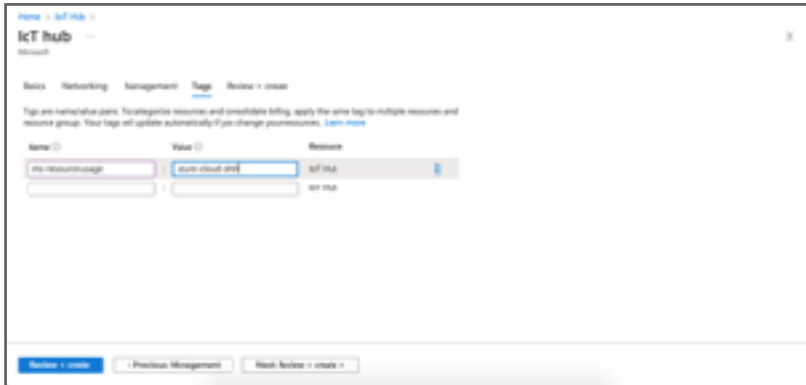
2

[Review + create](#)

[< Previous: Networking](#)

[Next: Tags >](#)

Change the Pricing and Scale tier to “F1: Free tier”, then scroll down and click on the box next to “Assign me to the IoT Hub Data Contributor role”, then click on “Next: Tags”.



The screenshot shows the 'Tags' configuration page for an IoT Hub. The breadcrumb navigation at the top is 'Home > IoT Hub > Tags'. The page title is 'IoT Hub' with a sub-header 'Microsoft'. Below the navigation tabs (Basics, Networking, Management, Tags, Review + create), there is a descriptive paragraph about tags. A table with three columns: 'Name', 'Value', and 'Resource' is present. The first row has 'my-resource-tag' in the Name column, 'azure-iot-hub' in the Value column, and 'iot-hub' in the Resource column. Below the table, there are buttons for 'Review + create', '< Previous: Management', and 'Next: Review + create >'. The 'Review + create' button is highlighted in blue.

Name	Value	Resource
my-resource-tag	azure-iot-hub	iot-hub

Click on the box's again to assign the tags and then click the “Next: Review + Create” button.

Home > IoT Hub >

IoT hub

Microsoft

✓ Validation passed.

Basics Networking Management Tags **Review + create**

Basics

Subscription	Azure subscription 1
Resource group	UIFlow
Region	UK West
IoT hub name	UIFlow
Disaster recovery enabled	Yes

Networking

Connectivity configuration	Public access
Private endpoint connections	None
Allow public network access	Enabled

Management

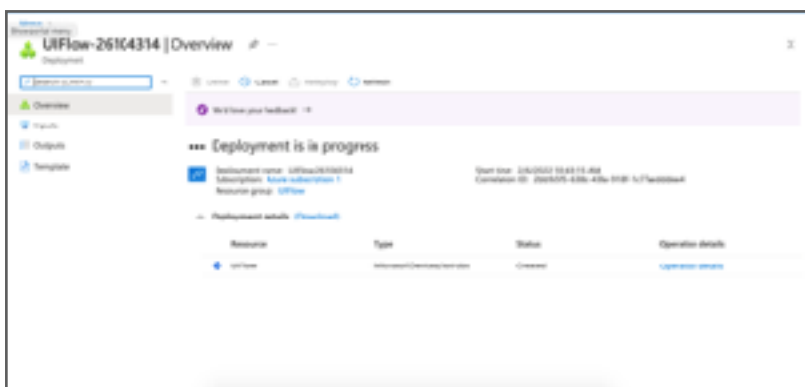
Pricing and scale tier	F1
Number of F1 IoT hub units	1
Messages per day	8,000
Device-to-cloud partitions	2
Cost per month	0.00 USD
Defender for IoT	Disabled

Tags

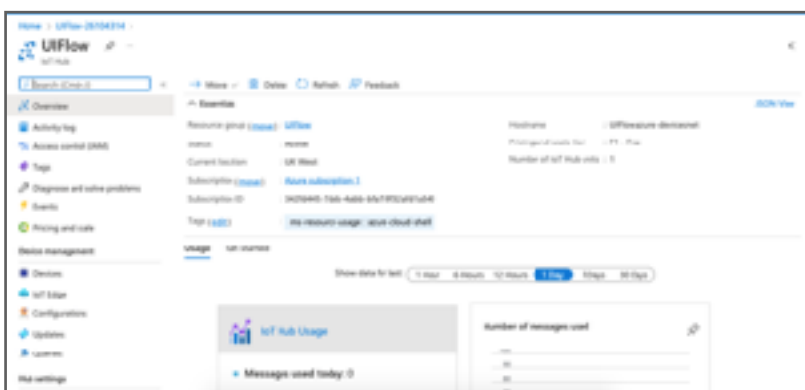
ms-resource-usage	azure-cloud-shell
-------------------	-------------------

Create < Previous: Tags Next > [Automation options](#)

Review the setting and then click on “Create” to create the hub.

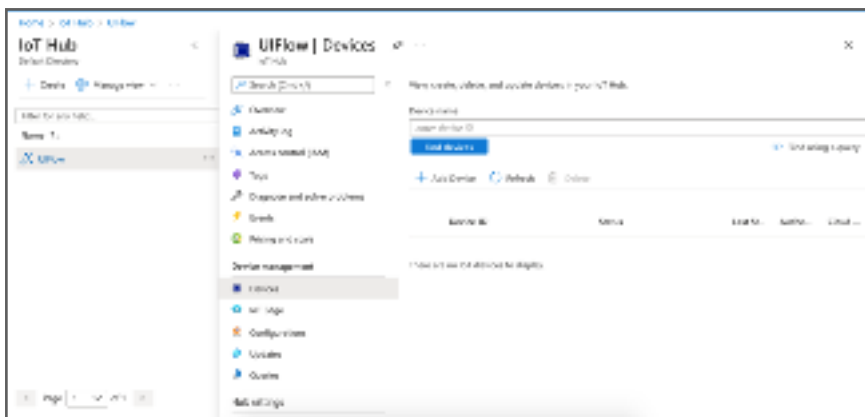


Deployment may take a while but when it is finished, you will have a button appear at the bottom that will take to back to the IoT Hub page.

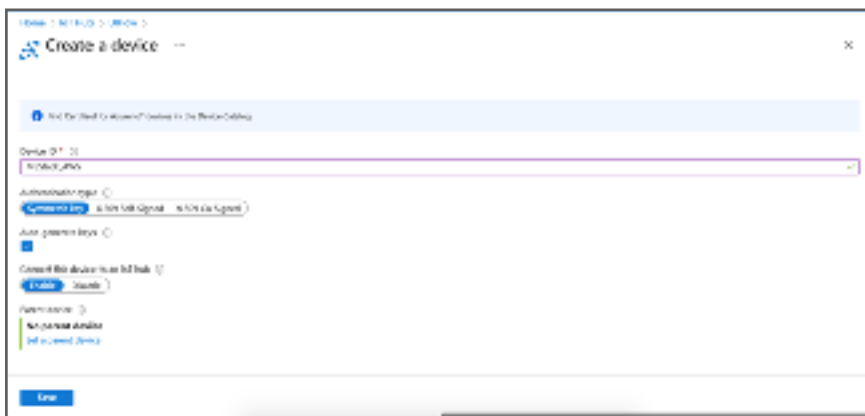


The page now shows the IoT Hub created and a set of charts showing various metrics. At the moment, nothing will appear in the charts because we haven't connected any devices.

Now that the resources and hub have been created, it is time to add a device. Click on the IoT Hub and then click on the hub created earlier to open it.



Under device management, click on “Devices” to open the device page. To create a device click on “+ Add Device”.



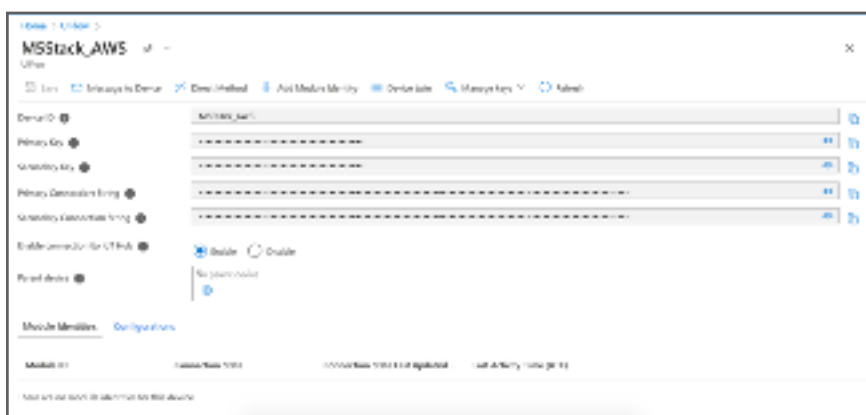
Type in a name which Azure will use to identify the device. In this guide I am using an M5Stack Core2 AWS and so I just type “M5Stack_AWS”.

Make sure all other settings are as shown and press save to create the device and return to the devices page.



If the device ID doesn't appear, wait a few seconds and then click refresh, this reloads the page listing the device as the page doesn't always get refreshed during creation.

Next click on the Device ID to load the devices options.

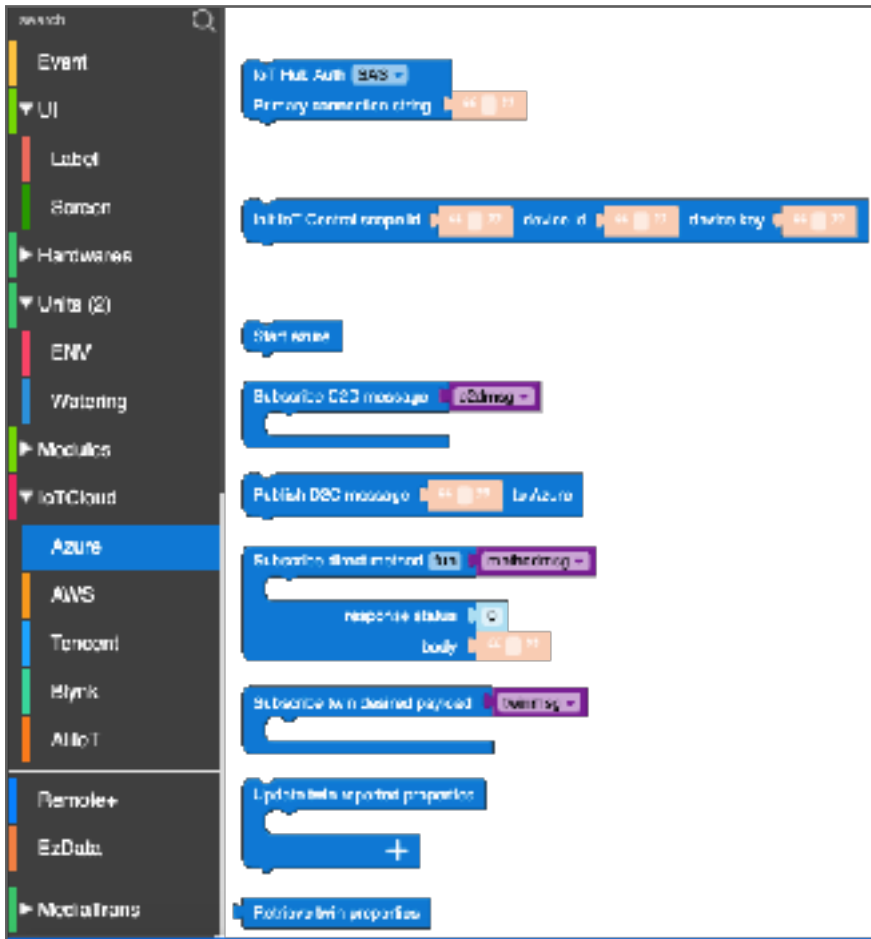


This will bring up this page showing the connection keys that will be needed for connecting hardware.

And that is everything needed to set up the Azure IOT server. Next it is time for the M5Stack side of programming.

Azure IoT UIFlow Blocks.

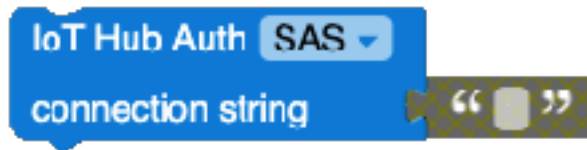
The UIFlow blocks used to access Azure IOT services are found in UIFlow under the IoT Cloud Menu:



In order to test the connection to Azure IoT Hub I used the following example:

Following is an explanation of the blocks and how to build an example code with them.

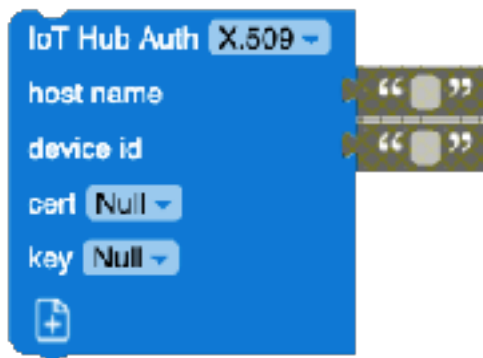
[IoT Hub SAS Authentication](#)



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

The drop down box allows you to select between SAS or X.509 as the authentication method and the text box is used to hold the connection string that need to be generated before connecting a hardware device to the Azure IoT Cloud.

[IoT Hub X.509 Authentication](#)



The Connection String box is where you type in the Primary connection string which UIFlow uses to generate the SAS keys during runtime.

The Micropython code for the SAS block consist of only one line:

```
azure = IoT_Hub(connection_string='')
```

While the X509 block consists of the following code:

```
azure = IoT_Hub(device_id=, host_name=, ssl=True,  
cert_file_path=", private_key_path=')
```

I need to point out that when the config block is added to the program, a new import function consisting of the following line:

```
from IoTcloud.Azure import IoT_Hub
```

is added to the list of imported functions.already visible in Micropython.

```
from m5stack import *  
from m5stack_ui import *  
from uiflow import *  
from IoTcloud.Azure import IoT_Hub
```

Next we have the IoT Hub Central connection block.

[IoT Central Connection Block.](#)



Unlike the previous SAS block that needs only the primary Key, the IoT Hub Central block needs three values that are obtained when we try to connect to hardware from within the IoT Hub Central App.

Device connection groups

ID scope

Device ID

Choose the authentication type for this device. You can choose the type if you want to.

Authentication type

Shared access signature (SAS)

Key

Shared Access Signatures (SAS) use security tokens and keys to connect to IoT Central. Use the SAS keys from the default enrollment group shown below to register your device.

[Learn more](#)

Primary key

Secondary key

Done

Next we have the Azure Start block.

[Azure Start](#)

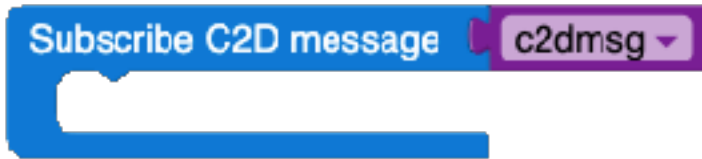


This block must be placed outside and before any program loops as it can only be run once. Placing it inside a loop will result in the Azure program trying to open multiple new connections to the Azure IoT Cloud and will result in the device being blocked as it could be perceived to be an attack attempt.

The Micropython code for this block is as follows:

```
azure.start()
```

[Subscribe C2D Message](#)



C2D is an abbreviation of Cloud to Device and is used to send messages from Azure web services to devices.

Subscribe C2D Message connects to a message server and is used to receive message.

The Miscropython code for this block is

```
def azure_C2D_cb(msg_data):  
    global c2dmsg  
    c2dmsg = msg_data  
    pass
```

[Publish D2C Message](#)

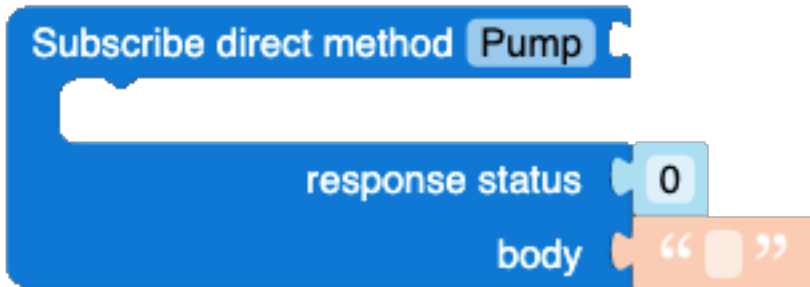


Publish D2C is used to send data to the Azure cloud message system.

The Miscropython code for this block is

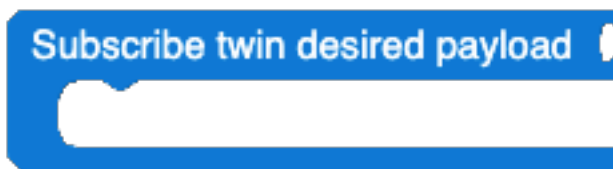
```
azure.publish_D2C_message(str())
```

[Subscribe Direct Method](#)



```
def azure_direct_Pump(payload, rid):  
    global pumpmsg  
    _ = payload  
  
    azure.response_direct_method(0, rid, body="")
```

Subscribe Twin Desired Payload.



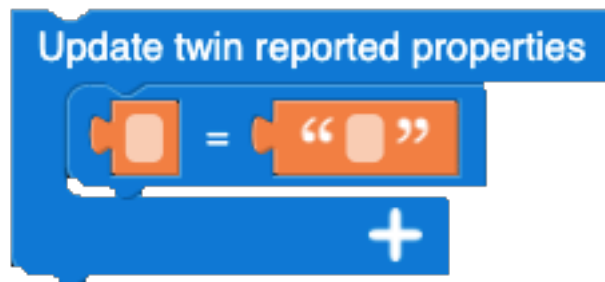
Subscribes to the Azure Device Twin system with the specified payload.

```
def azure_desired_cb(payload):  
    global pumpmsg  
    pumpmsg = payload
```

Update Twin Reported Properties.



Used to update properties in the Azure Device Twin
If you click on the “+” sign a block will appear to allow you send a Key and value pair.

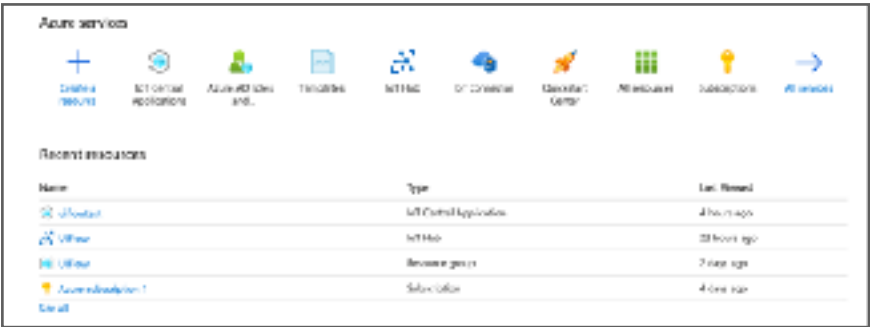


[Retrieve Twin Properties.](#)



Retrieve Twin Properties returns the values received by Azure’s Device Twin system

Creating an IoT Central Applications.



From The IoT hub locate IoT Central Applications and click on it to open the IoT Central page.



Click on “+ Create” to start creating a new application.

Home > IoT Central Applications >

IoT Central Applications

Default Directory

+ Create ⚙️ Manage view ...

Filter for any field...

Name ↑

uiflowtest ...

IoT Central Application

IoT Central Application

Resource name * ✓

Application ID * ✓ [api.iotcentral.com](#)

Subscription * ▼

Resource group * ▼ [Create new](#)

Pricing plan * ▼ [Learn more about pricing](#)

Template * ▼ [Learn more about application templates](#)

Location * ▼

< Page 1 of 1 >

[Create](#) [Automation options](#)

Give the resource a name but it must be all lowercase as the name will be used for the web address to the application.

Chose a Subscription. In my case I only had one choice.

Choose a Resource group. The Resource group I chose I created earlier in the guide.

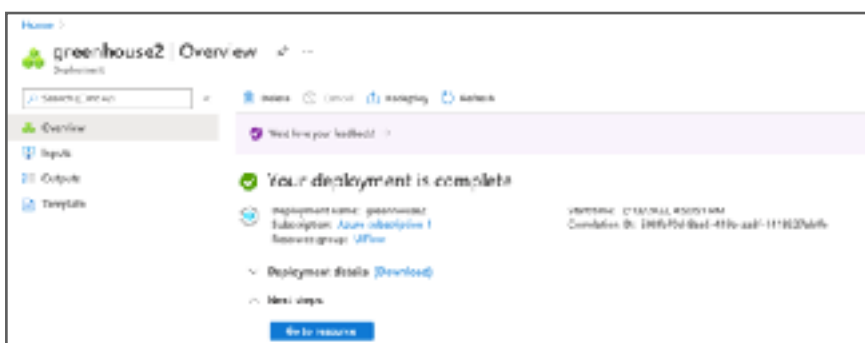
Select a Pricing plan, Standard 2 allows up to two free devices to connect to the application.

Set Location to the location in the world you are based in.

After filling in the required boxes click on “Create”

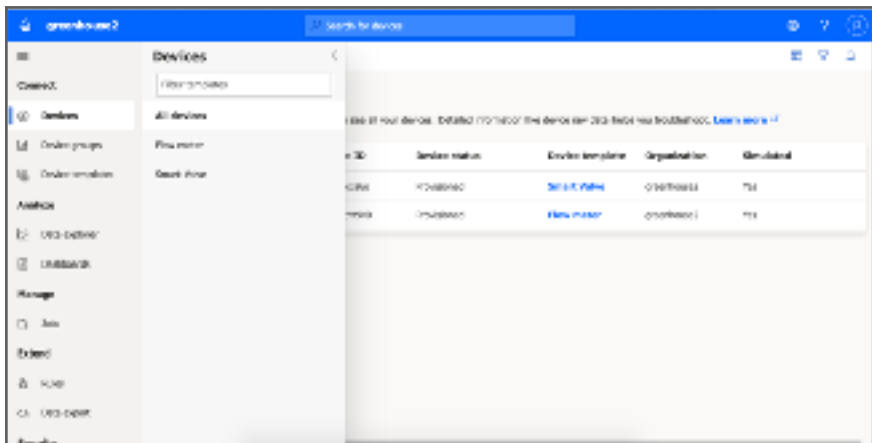


Wait for deployment to finish and when the “Goto Resource” button appears, click on it.



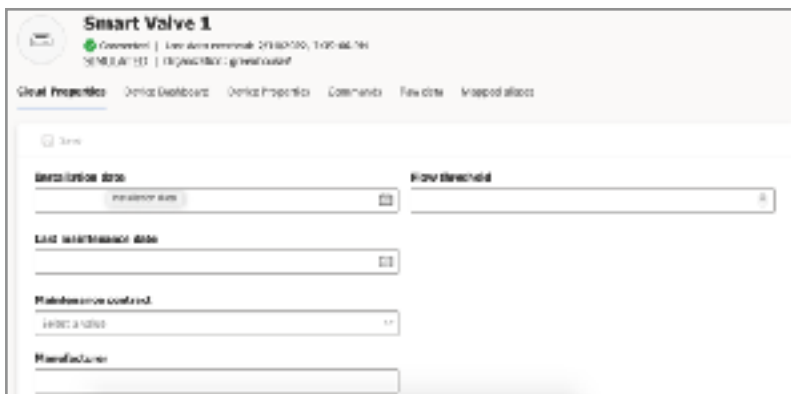
To view the application and to begin customising the application, click on the link to the right of the page.

The application screen will open to show the following



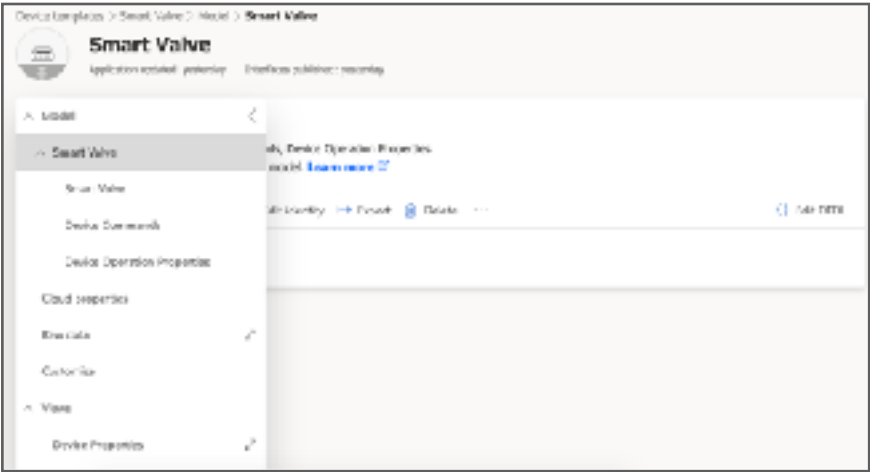
Here we can see that the page has been populated with examples that show us how the application has been built. Along with virtual devices that are used to simulate data.

If we click on a device we can see several pages of information and device data layouts.





And if we go to Device Templates:



We can see various screens that show how the data is to be sorted and reacted on by the application.

Device templates > Smart Valve > **Customize**



Smart Valve
 Application updated yesterday Interface published yesterday

5


Customize your interface Published

Decide what and how you want to apply custom settings to your interface capabilities (for example, if you want to specify minimum and maximum target ranges).

[Save](#)

Display name	Name (Interface)	Capability type	Semantic type ⓘ		
<input type="text" value="Temperature"/>	Temperature (Smart Valve)	Intensity	Temperature ✓	✕	▼
<input type="text" value="Actuator Position"/>	ActuatorPosition (Smart Va...	Property	None ✓	✕	▼
<input type="text" value="Rotation Speed"/>	RotationSpeed (Smart Valve)	Property	None ✓	✕	▼
<input type="text" value="Pressure"/>	Pressure (Smart valve)	Temperature	Pressure ✓	✕	▼

Device templates > Smart Valve > **Cloud properties**



Smart Valve
 Application updated yesterday Interface published yesterday

7

Cloud properties Published

Give your cloud property a display name and a friendly name, and choose a semantic type that describes the type of data the router will collect. Then choose how that data will be measured and displayed.

[List](#) [+ Add cloud property](#)

Display name	Name *	Semantic type ⓘ		
<input type="text" value="Flow (m³/sec)"/>	FlowThreshold	None ✓	✕	▼
<input type="text" value="Humidifier"/>	Humidifier	None ✓	✕	▼
<input type="text" value="Maintenance contact"/>	MaintenanceContact	None ✓	✕	▼
<input type="text" value="Last maintenance date"/>	LastMaintenanceDate	None ✓	✕	▼

Device templates > Smart Valve > Model 3 Smart Valve > Device Capabilities

Smart Valve

application opened previously | interface defined previously

>

Device Capabilities

Selected interface: | **Unselected:**

Create a specialized interface that inherits capabilities from other, more general interfaces (for example, a "Coffeepot Robot" interface that extends a "Robot" interface). [Learn more](#)

Save | Add capability | Edit identity | Version | Export | Delete | [Edit HTML](#)

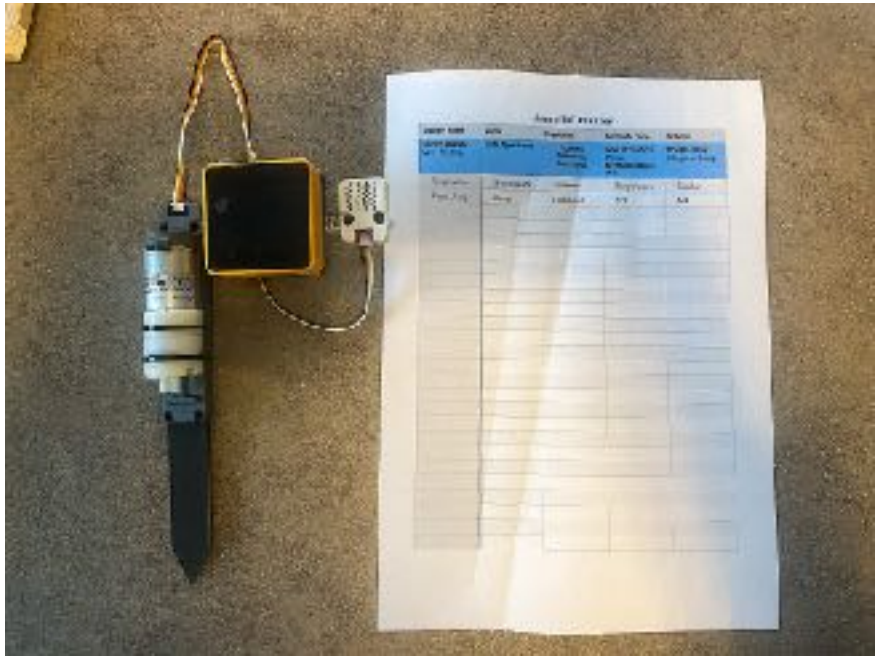
Display name	Name *	Capability type *	Semantic type		
Set Valve Position	SetValvePosition	Command		X	▼
Close Valve	CloseValve	Command		X	▼
Open Valve	OpenValve	Command		X	▼
Time Remaining	TimeRemaining	Property	None	X	▼

At the end of this book I have made a form which you can copy and print to make notes on the various functions you will need in your own projects. I have added the Temperature device as an example of an input and the moisture sensor/pump as an example of a command/output to get started.

Azure IoT Planner

Display Name	Name	Capability	Semantic Type	Schema
Show as Data Type Heading.	Data Type Name.	Property, Telemetry, Command	Type of reading (Temp, Pressure, Speed, etc.)	Double, Float, Integer, or Long
Temperature Sensor	Temperature	Telemetry	Temperature	Available
Moisture Pump	Pump	Command	NOT	NOT

In the following picture I have the initial hardware set up in a basic configuration. Unfortunately the Core2 AWS only has one Port A and One Port B and so I can only connect the ENV to port A and the moisture Sensor/Pump to port B.



Next to the hardware is the planner form. The two example entries are for the hardware but more entries are needed. In the following table I list the additional entries that need to be added:

Azure IoT Planner-1

Display Name	Name	Capability	Semantic Type	Schema
Shown as Data type Heading.	Data Type Name.	Property, Telemetry, Command	Type of reading (Temp, Pressure, Speed, etc)	Double, Float, Integer or Long
<i>Humidity</i>	<i>Humidity</i>	<i>Telemetry</i>	<i>Percent</i>	<i>Double</i>
<i>Pressure</i>	<i>Pressure</i>	<i>Telemetry</i>	<i>Pascal</i>	<i>Double</i>
<i>Temperature</i>	<i>Temperature</i>	<i>Telemetry</i>	<i>Temperature</i>	<i>Double</i>
<i>Moisture Sensor</i>	<i>Moisture</i>	<i>Telemetry</i>	<i>Percent</i>	<i>Double</i>
<i>Water Pump On</i>	<i>PumpOn</i>	<i>Command</i>	<i>N/A</i>	<i>N/A</i>
<i>Water Pump Off</i>	<i>PumpOff</i>	<i>Command</i>	<i>N/A</i>	<i>N/A</i>

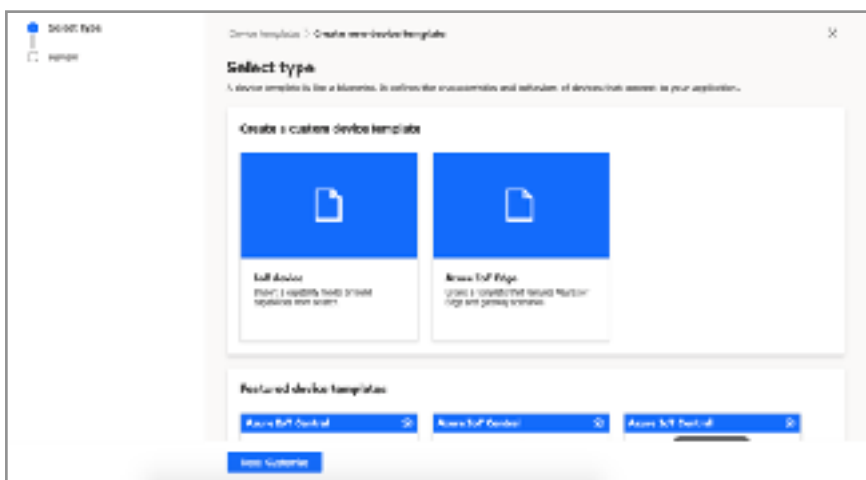
Now that I have made a list of the functions I need in the Azure application, we need to set the Azure IoT Application to receive and handle the data to be received from the Core2 AWS.

Reopen the Azure Application and go to the Templates Section and delete the existing templates. As we made a list of the functions we need (shown above) based on the information in those templates, we don't need them any more.

Click on the "New" button at the top of the screen to start creating the new template:



This opens the Template creation panel that allows us to create a new IoT device template, create an IoT Edge device template, or import an existing device template. As no template exist for the Core2 AWS or any other M5Stack device, we need to click on the IoT device button.



Click on “IoT Device” and then click on the “Customise” button that appears at the bottom of the screen to move on.

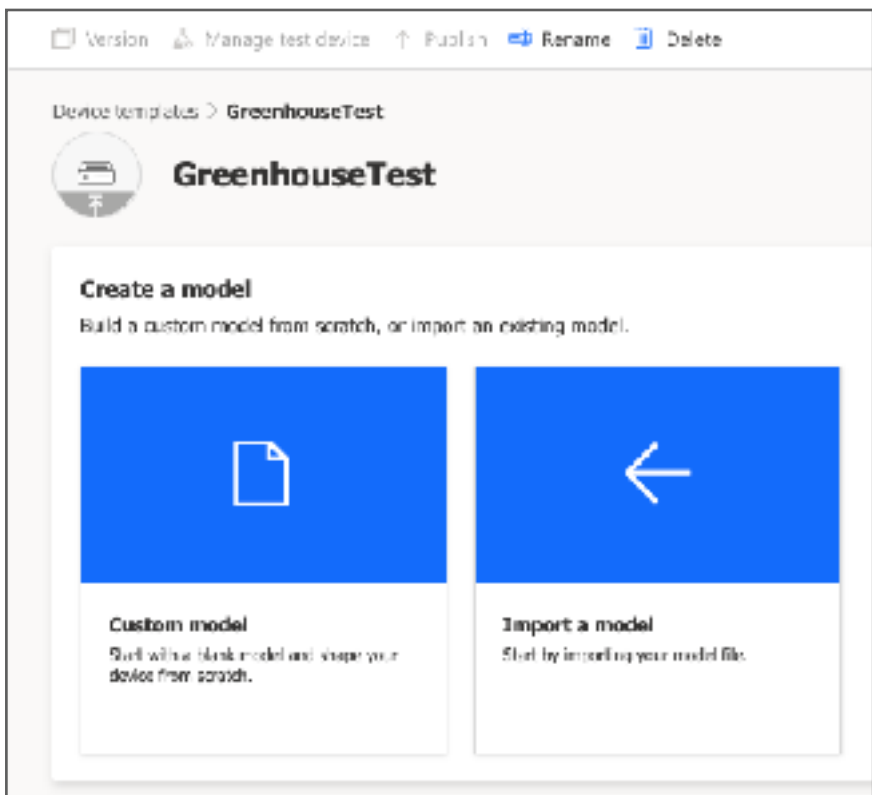


Give the template a name and then click the “Next: Review” button to move on:



As this shows what we have filled in so far, click on the “Create” button to make the template.

Now the template is created we can move on to adding the hardware model that will tell the application how to handle the data being sent to it.



Click on “Custom Model” and this take us to the page allowing us to set up how the data will be handled.



Click on “+ Add Capability” button to start adding the functions.

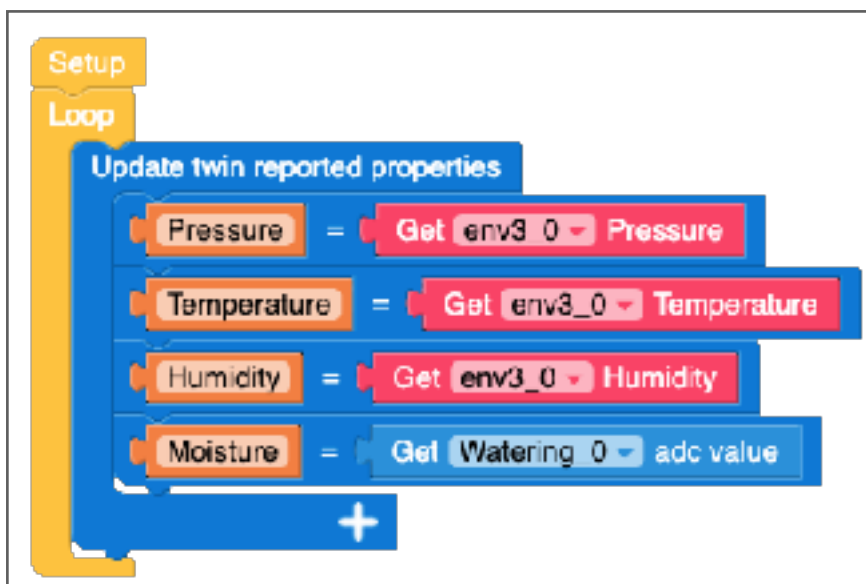


Here I show how the inputs are handled. The way the “Name” is typed here is important as this tells the application what part of the data contains what.

Setting these to properties will allow them to be shown in the data of the Application:

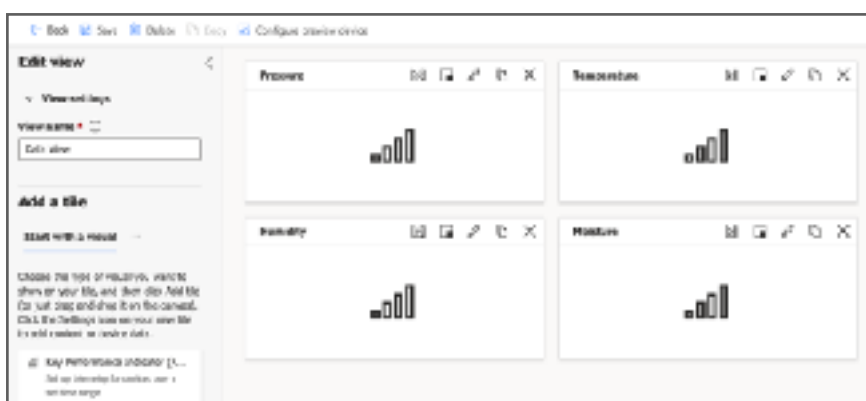
Interface / Hu...	Interface / Moi...	Interface / Pre...	Interface / Te...
4B.59083	2063	100541.4	22.71152
4B.37873	2064	100541.9	22.65545
4B.50385	2062	100543	22.58335
4B.59388	2062	100540.1	22.68215
4B.61219	2064	100539.5	22.6688
4B.52674	2063	100539.6	22.68215
4B.59388	2043	100539.7	22.59665
4B.61829	2063	100537.6	22.61272
4B.65034	2060	100535.1	22.65545
4B.44434	2061	100534.5	22.72488

This must be exactly as typed in the “Twin Properties” box in UIFlow or the data will not be recognised.



Clicking on the arrow next to each function we create allows us to customise the function by setting the units that the function data is being sent in and the application needs to show it as.

Skip down to views, click on create view, give the view a name and drag four bar graphs onto the area on the right which will be used to display data sent from the Core2 AWS to the application.



Click on the pencil icon to open the configuration for each bar graph, give each a different name and click on the “+ Add Capability” button to select the telemetry stream you wish it to show.

Configure bar chart

Title *

Pressure

Show legend

Or

Show X axis

Or

Show Y axis

Or

Display range

Past 30 minutes

Telemetry

Pressure

Average

+ Capability

Update

Cancel

Click on “Update” to apply the setting to the bar graphs and once all four are set, click on the “Save button to save and close the view window returning us to the main template window. Ignore the commands functions I have added in my screen shots and click on “Publish” at the top of the screen to make the template live.

Next go to “Devices” and click on the “+New” button to create a new device:

Create a new device

To create a new device, select a device template, a name, and a unique ID. [Learn more](#)

Device name

Cine2 AWS

Device ID

2H8vdyke

Organization

UFT Design

Organization

Device template

Greenhouse

Simulate this device?

A simulated device generates telemetry that enables you to test the behavior of your application before you connect a real device.

☐ No

Create

Cancel

Give the device a unique name, set the template to the template we just created and then click on the “Create” Button.

Devices

Filter templates

All devices

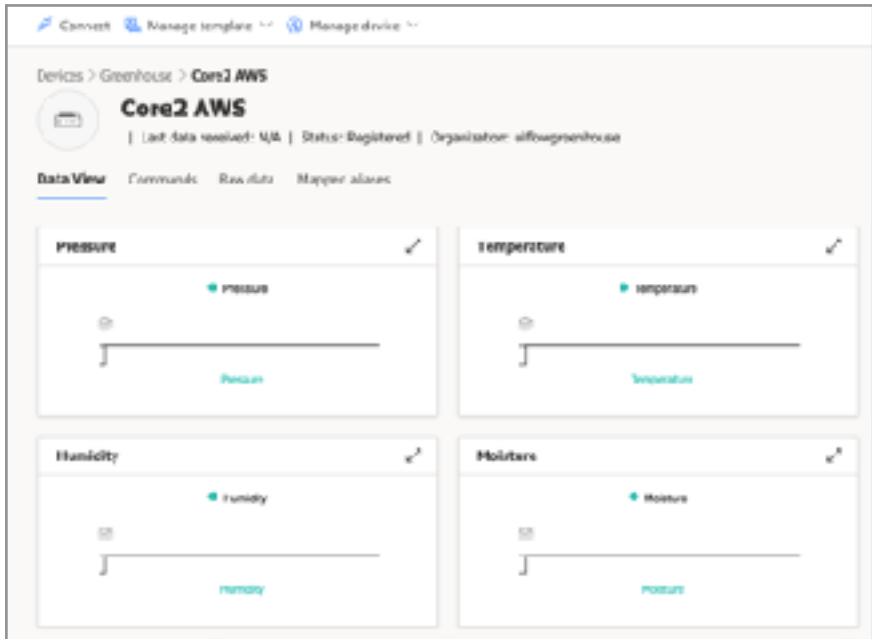
Goodhouse

All devices

Device name tags reveal all your devices. Detailed information for each tag will help you troubleshoot. [Learn more](#)

Realtime name	Device ID	Device status	Device template	Organization	Simulated
Cine2 AWS	2H8vdyke	Tested	Greenhouse	UFT Design	No

Click on the device to open it:



and at the top of the screen click the “Connect” button to open the page containing the connection keys:

Device connection groups

ID scope ⓘ

IncDME47B3

Device ID ⓘ

2h0tutDyke

Choose the connection type for this device. You can change this later if you need to.

Authentication type

Shared access signature (SAS)

Key ⓘ QR code

Shared Access Signatures (SAS) use security tokens and keys to connect to IoT Central. Use the SAS keys from the default enrollment group shown below to register your device.
[Learn more](#)

Primary key ⓘ

vi+g5hB5PF0zSpk58ntCFBRJ6zFw/G3WygV0Lz/Apw=

Secondary key ⓘ

ReyM0uxDCHLs5BN3zWgr7s52Q/U0gAQ0/VymQJ7hEs=

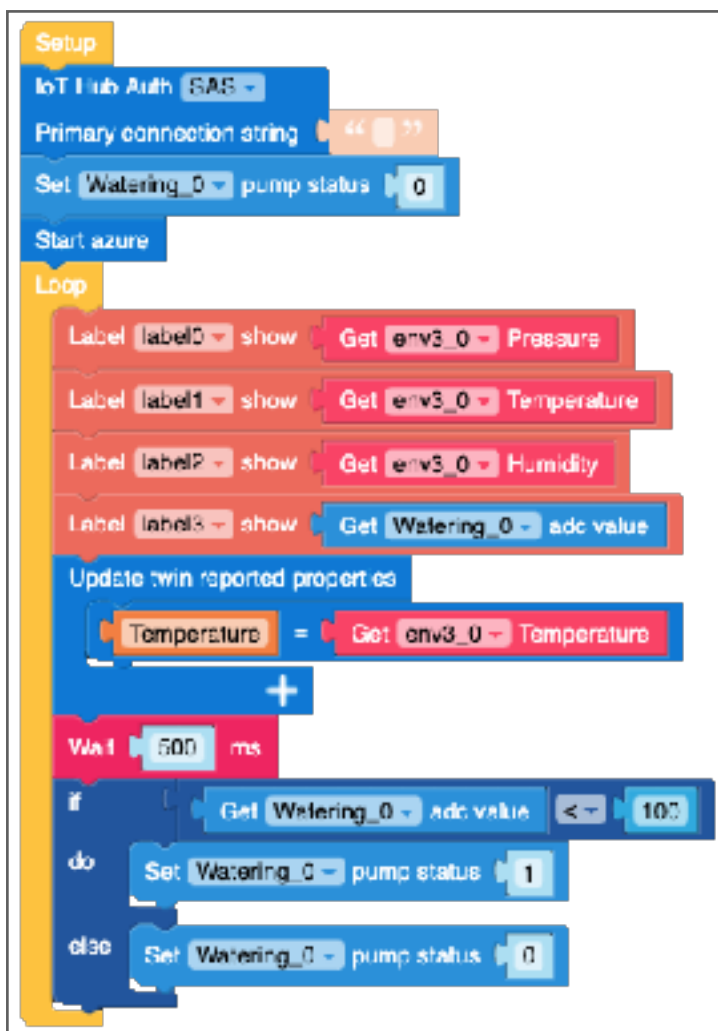
Close

Make a note of these keys and strings as they will need to be entered into the IoT Central connection block of the UIFlow code. Click on close and now we can move on to UIFlow and start creating the code.

First we start by merging the two example programs given in the hardware section into one program:



Next we need to connect the Core2 AWS to The Azure example application using the Azure IoT Central Connection block and then add the updated Twin Properties block that sends the captured data to the Azure Application.



Copy the code as shown above, replace the Scope ID, Device ID and Device key with your own keys and save the program to the Core2 AWS so that it starts running and sending data.

Alas, as the Azure IoT Central is only free for 30 days, I am unable to get the pump working and continue using the Azure services without paying.

Maybe one day when finances are more willing I may be able to revisit Azure services but until then I apologise for not being able to finished the Azure side of the guide.

Amazon Web Services (AWS)

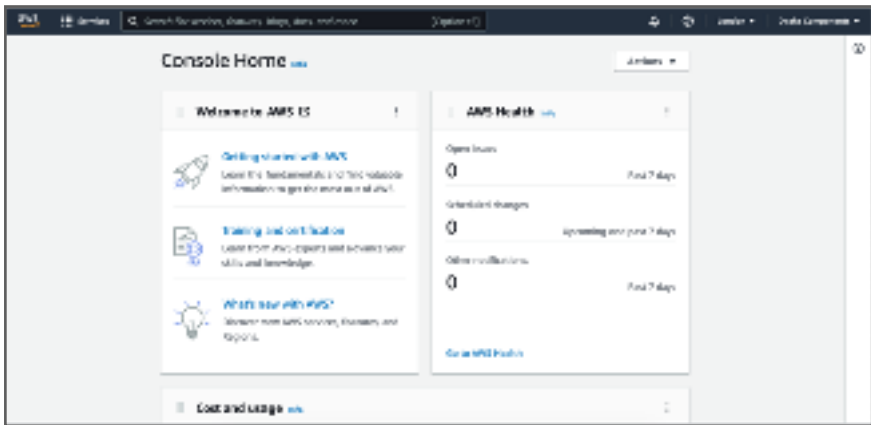


AWS IoT

Introduction.

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:

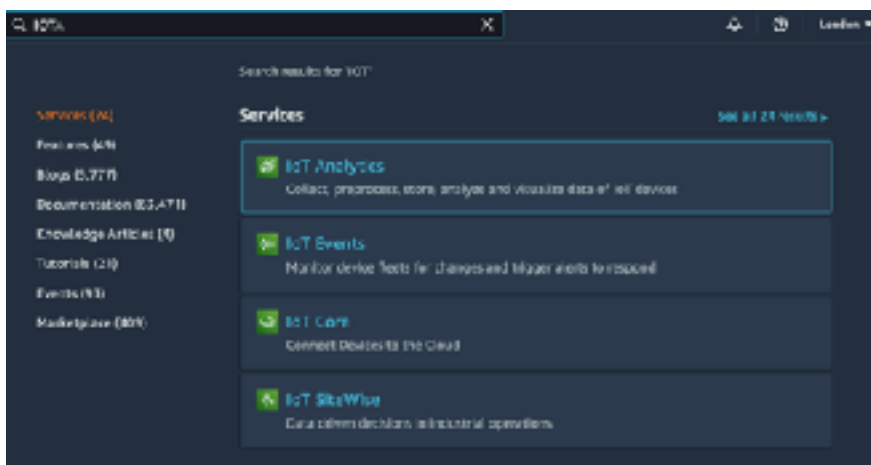


This page will give you some getting started guides and a list of costs you may build up once you start using the services.

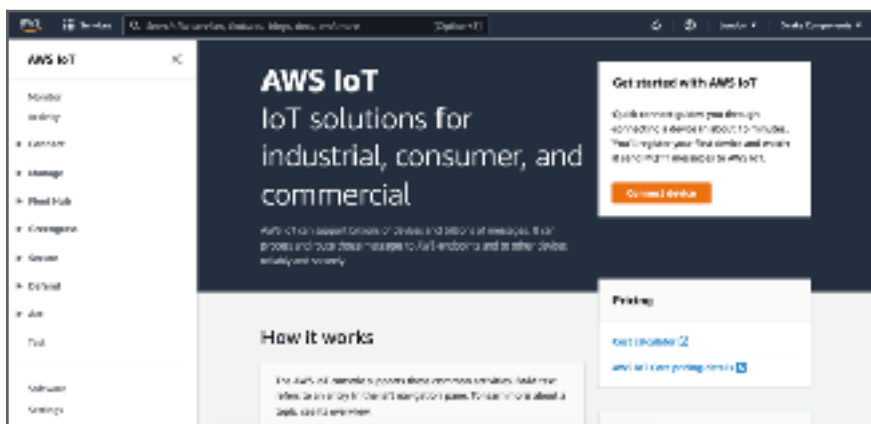
At the top of the screen next to the name you registered as you will find a location, this must be set your geographically local region or you may get some unexpected financial charges. Another note about the region servers is that not all AWS services are available to all regions.

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.



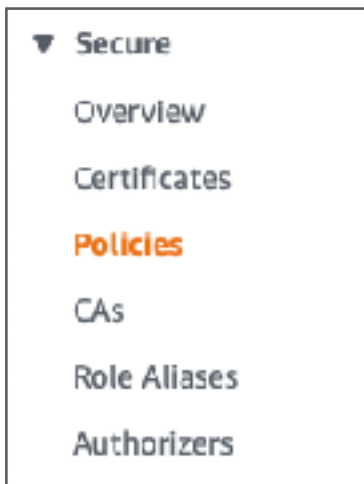
It is worth looking through the guides and documents in order to understand the IOT service and how to use it before moving on.

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, policies 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 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.

Before we can create a thing that is the AWS IOT representation of the M5Stack, we need to create a policy which allows the M5Stack to communicate with the AWS IOT service. To create a policy need to scroll down through the menu of the left to find secure, click on it to open the menu and click on policy.



This opens the policy page:



Here you can see that I already have a policy created but I will still show you how to create the policy.

Click on the “Create Policy” button:



Give the policy a name and in the next box, click on JSON to open the policy rules panel.



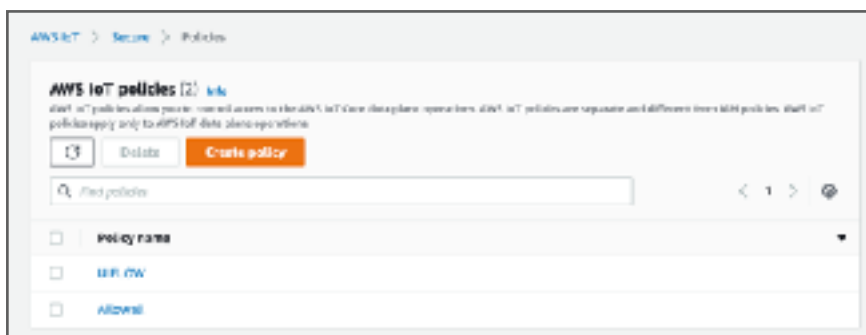
In the code box change:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "",
      "Resource": ""
    }
  ]
}
```

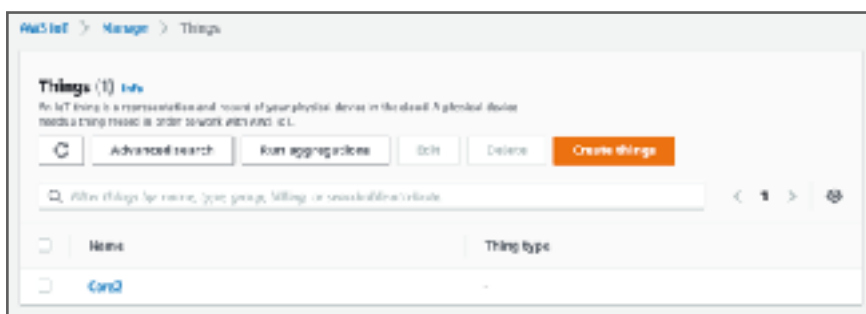
To

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    }
  ]
}
```

Click on “create” to create the policy and return to the policy page that will now show out newly created policy.



Now the policy is created we can now create a thing. Scroll up to “Manage” and click on “Things” to open the Things page:



Click on the “Create Thing” button to open the create thing page:

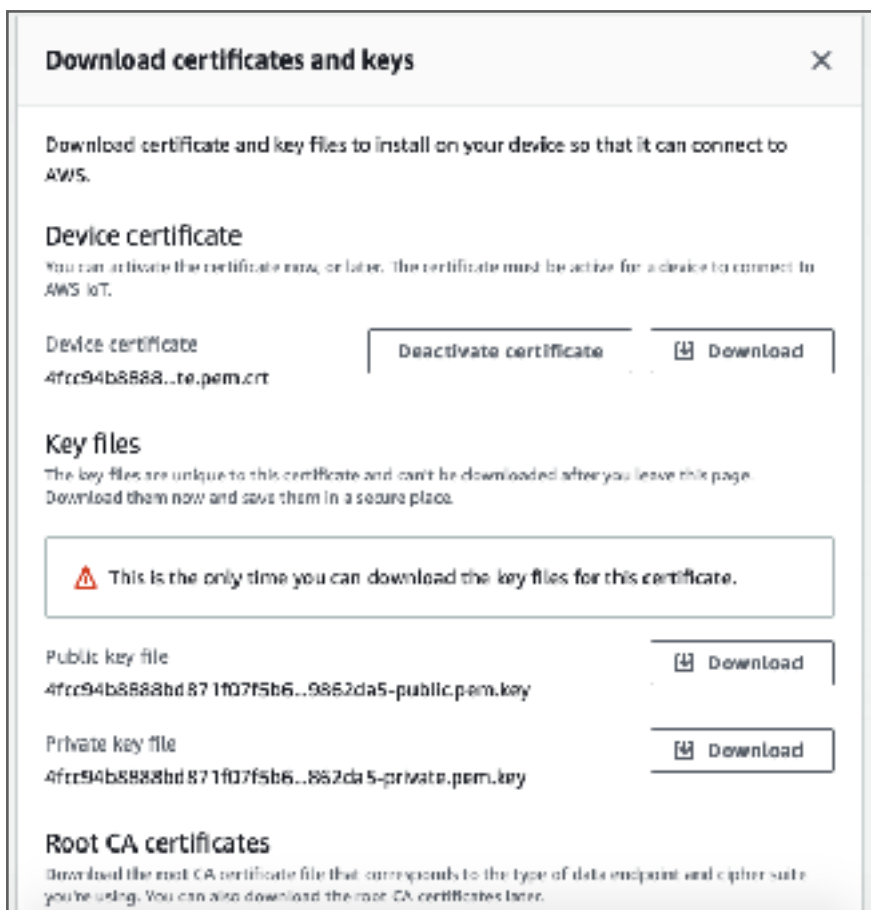


As this is the first time you are setting up a thing make sure that “Auto-generate a new certificate (recommended)” is selected before clicking on “Next”



This page allows us to select the policy that was created earlier. (Here I am using the existing policy).

Clicking on “Create Thing” will create the thing and bring up the device certificates.



Important note: These keys and codes do not work on my account As they are now deactivated and now deleted!

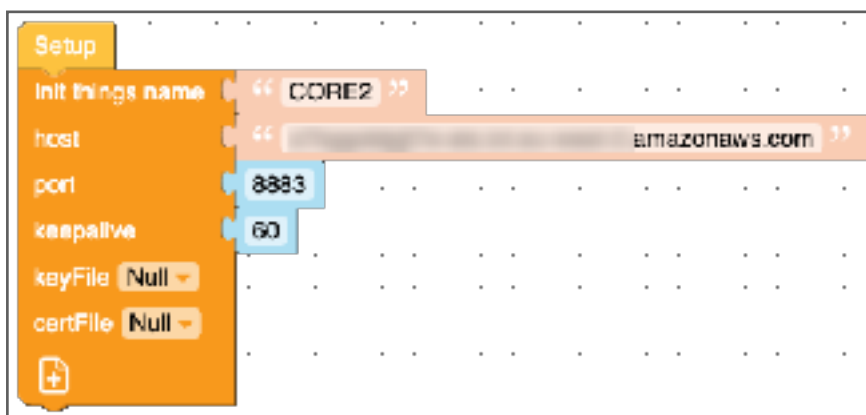
Download the Certificates and keys and then click on “Done” to close the panel and return to the Thing page sowing our newly created thing.



And that is everything needed to set up the AWS IOT server. Next it is time for the M5Stack side of programming.

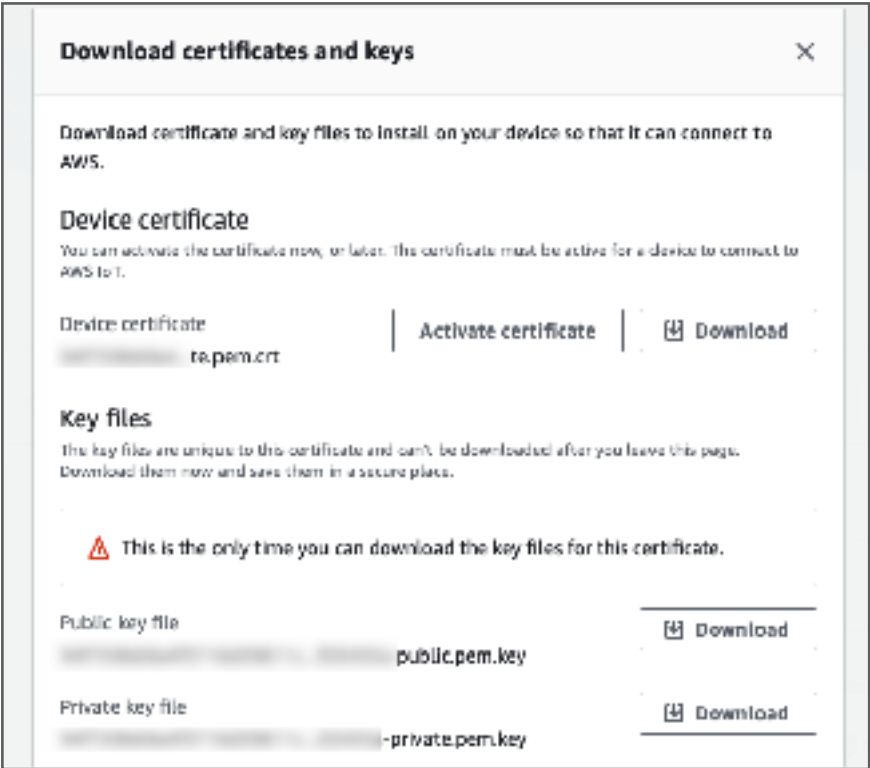
The UIFlow blocks used to access AWS IOT services are found in UIFlow. Following is an explanation of the blocks and how to build an example code with them.

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 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 have 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

aws = AWS(things_name="", host="", port=0, keepalive=0,
cert_file_path="", private_key_path=“)
```

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.

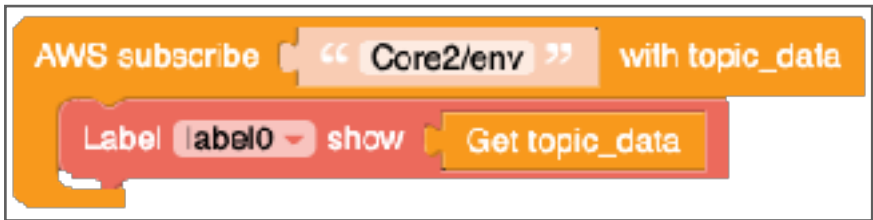


The Above screenshot is the topic the AWS IoT test server used by the demo I will share later in this section.

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.



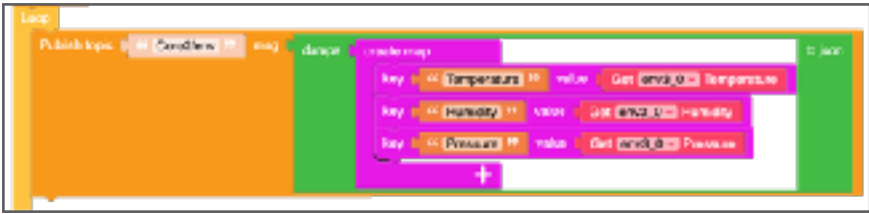
In this example I am using a label to display data from the AWS IoT topic on the M5Stack screen. In the following photo you can see how the data is displayed on the screen of an M5Stack.



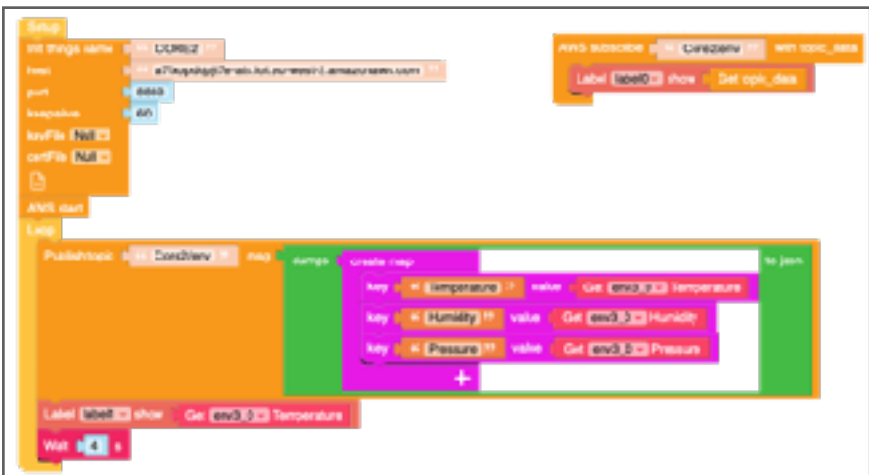
Publish Topic



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 ENVI8 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(0xFFFFFFFF)
env3_0 = unit.get(unit.ENV3, unit.PORTA)

label0 = M5Label('Text', x=-2, y=147, color=0x000,
font=FONT_MONT_14, parent=None)
label1 = M5Label('Text', x=0, y=176, color=0x000,
font=FONT_MONT_14, parent=None)

def fun_Core2_env_(topic_data):
    # global params
    label0.set_text(str(topic_data))
    pass

aws = AWS(things_name='CORE2', host='a7lxppddpjt7e-
ats.iot.eu-west-2.amazonaws.com', port=8883, keepalive=60,
cert_file_path="/flash/res/certificate.pem.crt",
private_key_path="/flash/res/private.pem.key")

aws.subscribe(str('Core2/env'), fun_Core2_env_)
aws.start()
while True:
    aws.publish(str('Core2/env'),str((json.dumps({'Temperature':
(env3_0.temperature), 'Humidity':(env3_0.humidity), 'Pressure':
(env3_0.pressure)}))))))
    label1.set_text(str(env3_0.temperature))
    wait(4)
    wait_ms(2)

```

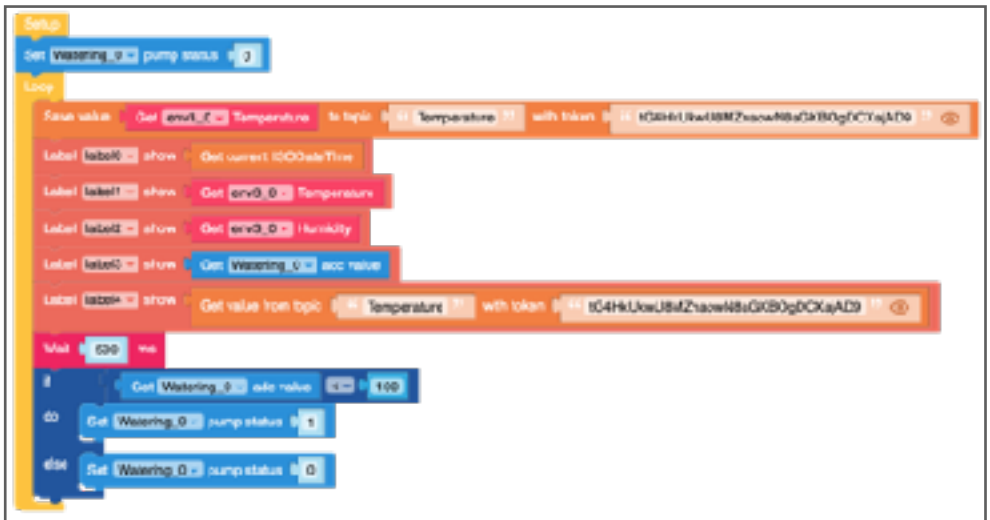

MQTT



Introduction.

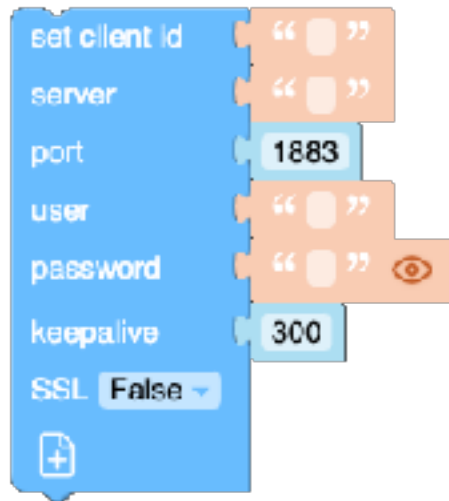
So far I have shown you how to use IoT services that have dedicated blocks available in UIFlow. In this section I will show you how to connect to other services using the MQTT blocks available in UIFlow.

As always, we start with the basic program:



Next we add the MQTT “Set Client ID” block.

MQTT Credentials



The MQTT Credentials block is used to hold the access credentials needed to log into an MQTT server as well as assign any SSL certificates that may be required to access the MQTT servers. These credentials 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 the MQTT/IoT server address.

Port is set to 1883 by default,

User is the user name you used to registered with an MQTT/IoT service.

Password hold the Server access password or access key.

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.

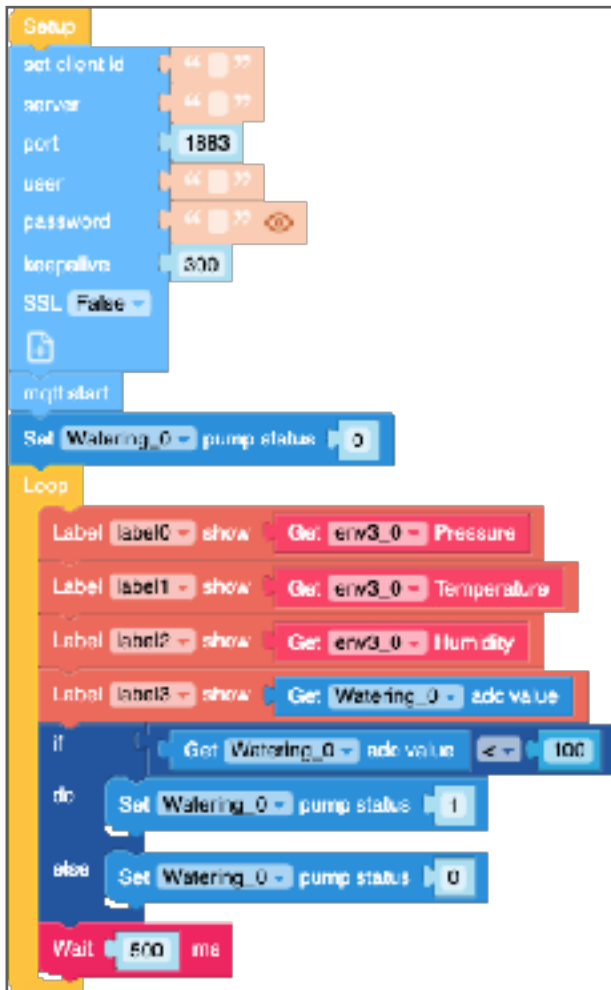
SSL is used if the IOT service provides SSL encryption certificates and keys. Setting this to true brings up additional boxes that allow you to assign the various certificates.

MOTT Start

Next we have the MQTT Start block:



This block is used to start the MQTT service on the Core2 and goes directly after the MQTT Credentials block:



[Publish to Topic with Message](#)

In order to send data to an MQTT Server we need to use:



The Topic is where the data is to be listed and the message contains the data/telemetry from the sensor connected to the Core2.

In the following sample of code based on the previous example: the temperature reading from the ENV3 unit are send to the Temperature topic:



The sample shows that I have just added the publish block to the beginning of the loop before the label blocks.

[MQTT Subscribe with Topic Data](#)

In order to receive data/telemetry from the MQTT server, we first need to subscribe to a topic using the following block:



The MQTT Subscribe block is a loop function that constantly checks a specified subject for updated data/telemetry.

[Get Topic Data](#)

In order to display the data/telemetry retrieved we use the Get Topic Data block:



The Get Topic block only returns the data as a value and so need to be placed in a block like a label block in order to display the data on the Core2's Screen.

The following example when added to the current program and run will show the retrieved temperature reading at the bottom of the Core2's screen.



[Set Last Will Topic Message](#)



Adafruit IO



Introduction.

Adafruit IO is an IoT/MQTT service provided by Adafruit industries and has been designed to allow quick and easy setup and connection of hardware to the internet.

Unlike AWS and Azure (as of this book going to press) adafruit.io does not have dedicated UIFlow blocks. Over the next sever pages I will show you how to use UIFlows MQTT blocks to connect to the adafruit.io servers.

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

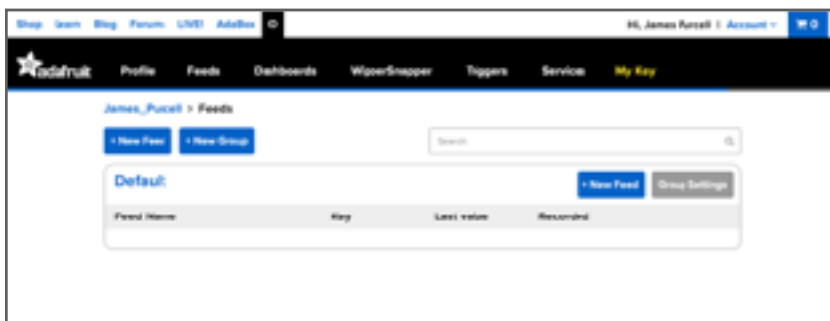
Once you register or sign in you will be taken to the profile page:



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:

Create a new Feed

Name

ENV Reading

Maximum length: 128 characters. Used: 11

Description

Receiving values from the M5Stack ENV3

Cancel Create

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:

MyFeeds

ENV Reading

Feed Name	Env	Last value	Recorded
ENV Reading	env-reading	25.00	25.00

We can also create groups of feeds by clicking on the ‘Create Group’ button and naming it like we did for the feed:

Create a new Group

Name

Greenhouse

Description

Show detailed group JSON record

Cancel

Create

Click on create and a feed group will appear under feeds:

James.Percell > Feeds

+ New Feed

+ New Group

Search

Default

+ New Feed

Group Settings

Feed Name	Key	Last value	Recorded
-----------	-----	------------	----------

Greenhouse

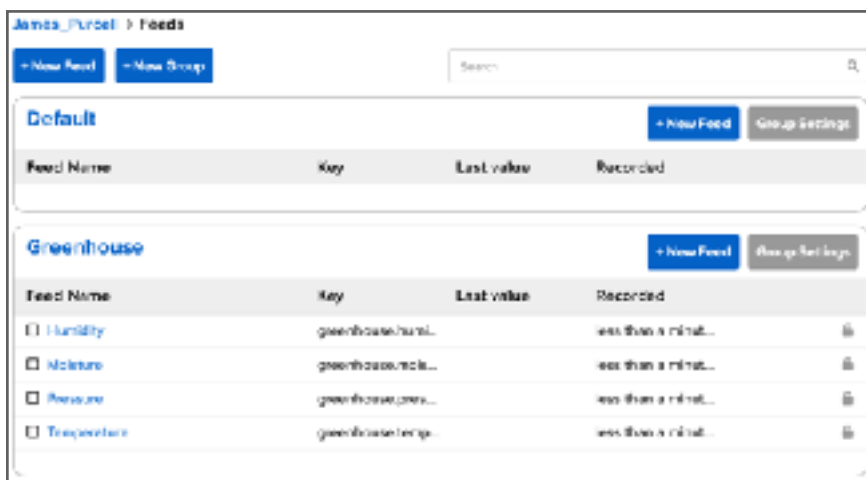
+ New Feed

Group Settings

Feed Name	Key	Last value	Recorded
-----------	-----	------------	----------

Loaded 1/0/0 records

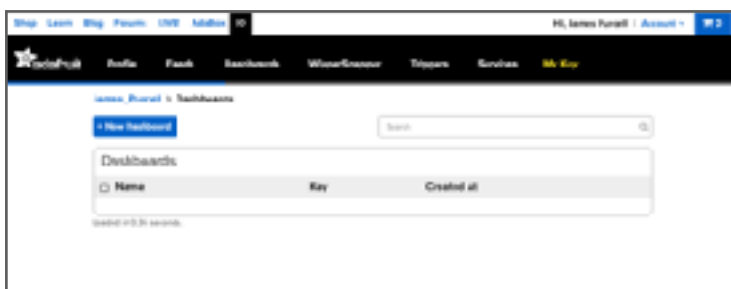
Now click on the “+ New Feed” button in the group box and add separate feeds for the data streams we will be sending:



Now that we have created the feeds, we 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:

 This screenshot shows a modal dialog box titled 'Create a new Dashboard'. It has a close button (X) in the top right corner. The dialog contains two input fields: 'Name' and 'Description'. The 'Name' field is filled with the text 'Greenhouse'. Below the 'Description' field, there are two buttons: 'Cancel' and 'Create'.

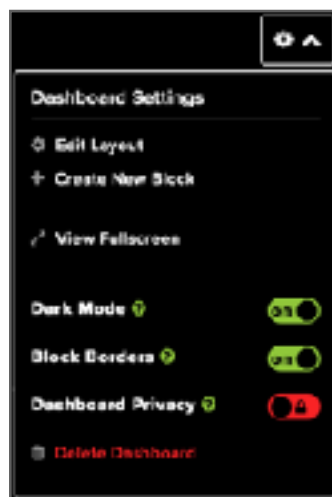
Click on the “Create” button and adafruit.io will return to the dashboard screen showing our newly created dashboard:



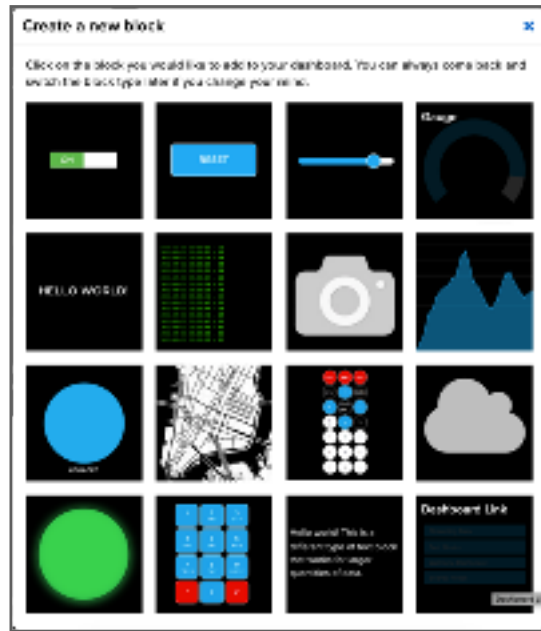
Click on the newly created “Environmental Dashboard” and the dashboard editor will open:



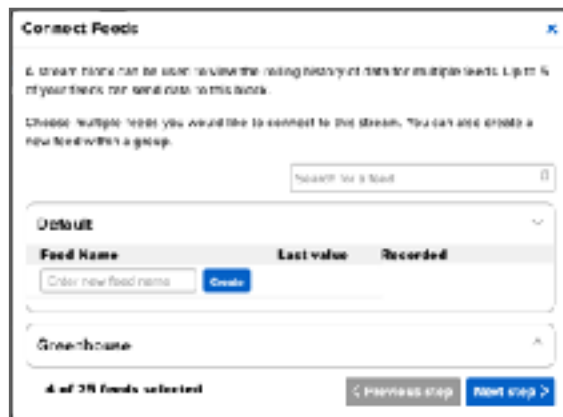
Click on the gear shape to open the dashboard menu:



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:



As you can see in the previous image, there are no feeds visible but there is the Greenhouse group. Click on the arrow to the right of Greenhouse to open the grouped feeds:

Connect Feeds

A stream block can be used to view the rolling history of data for multiple feeds. Up to 5 of your feeds can send data to this block.

Choose multiple feeds you would like to connect to this stream. You can also create a new feed within a group.

Search for a feed

Default

Feed Name	Last value	Recorded
<div>Enter new feed name</div>		<div>Create</div>

Greenhouse

Feed Name	Last value	Recorded
<div><input checked="" type="checkbox"/> Humidity</div>		<div>about 6 hours</div> <div></div>
<div><input checked="" type="checkbox"/> Moisture</div>		<div>about 6 hours</div> <div></div>
<div><input checked="" type="checkbox"/> Pressure</div>		<div>about 6 hours</div> <div></div>
<div><input checked="" type="checkbox"/> Temperature</div>		<div>about 6 hours</div> <div></div>

Enter new feed name

Create

4 of 25 feeds selected

< Previous step

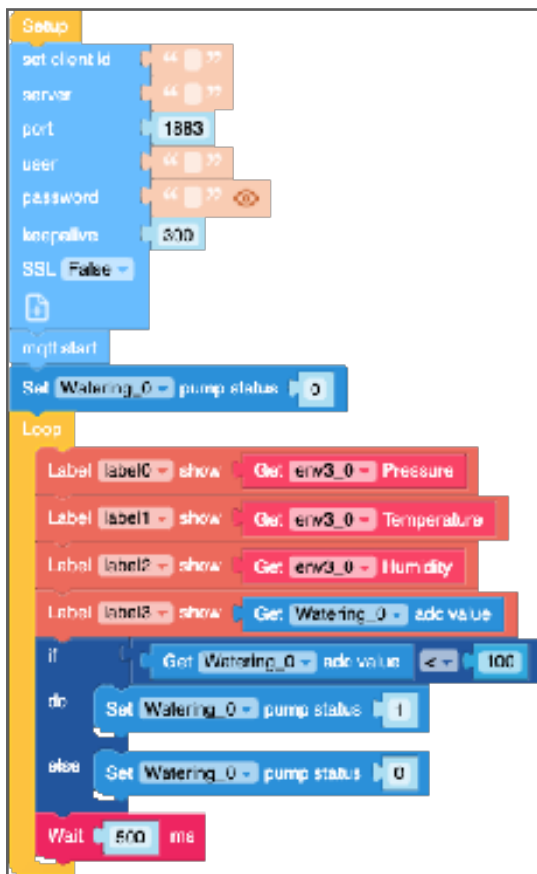
Next step >

Click on the box's next to the feeds to select them and the button on the bottom will now change to "Next Step" Click on it and the settings window will appear:

That is everything we need to set up on the io.adafruit.com side of things 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.

Starting with the Sample code from the MQTT section:



Enter the details into the configuration block 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

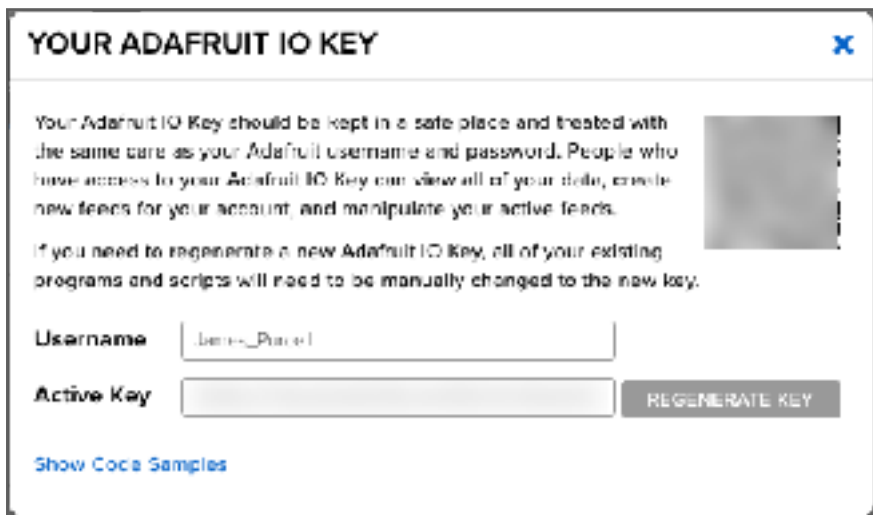
immediately disconnected preventing both devices from access the stream.

Server is io.adafruit.com.

Port is set to 1883 by default for none secure connections and 8883 for SSL encrypted connections.

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



YOUR ADAFRUIT IO KEY

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

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

Username

Active Key

[Show Code Samples](#)

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.

When asked about certificate and key files Adafruit can be a little reserved when answering and will point too examples that use a secure connection without these files.

Next we add the Publish Topic block before the Label blocks and set the topic to the MQTT Feed address found By clicking on the gear next to Feed Info that will bring up the following:



Edit Feed

Name
Temperature
Maximum length: 125 characters. Used: 11

Key
temperature
Changing the key will change API URLs and MQTT subscription topics. The only characters we permit are lower case english letters ('a' to 'z'), numbers, and dash ('-').
[See our guide to naming things in Adafruit IO](#) for more information about how we handle the formatting of **names** and **keys**.

Current Endpoints

Web https://io.adafruit.com/James_Purcell/feeds/

API https://io.adafruit.com/api/v2/James_Purcell/

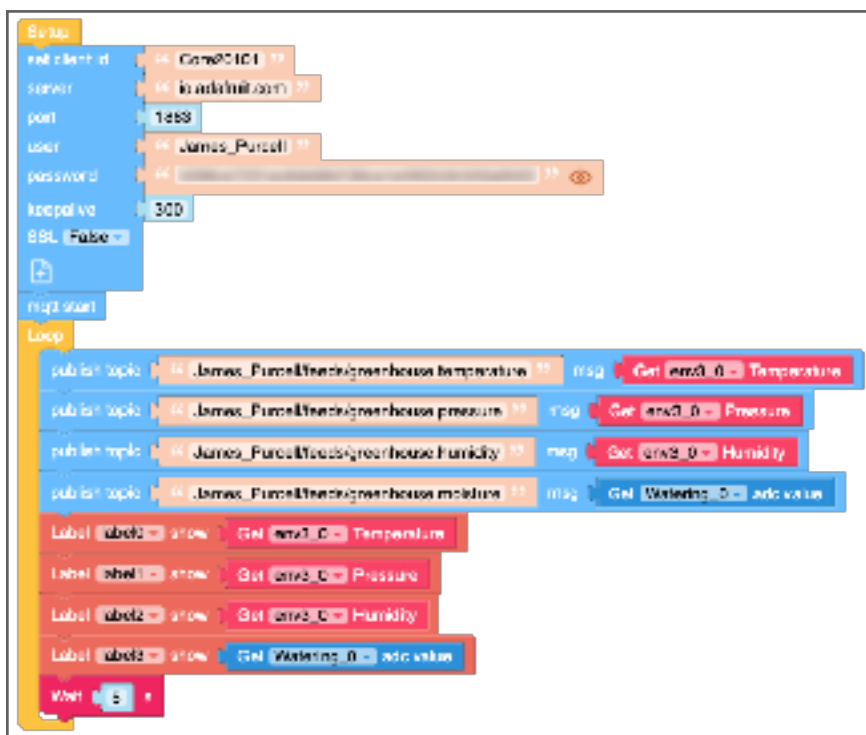
MQTT by Key [James_Purcell/feeds/greenhouse.temperature](#)

Description

[Show detailed feed JSON record](#)

[Cancel](#) [Save](#)

Copy the link (highlighted) next to MQTT by Key and paste it into the topic space and replace for all four of the sensors data streams.

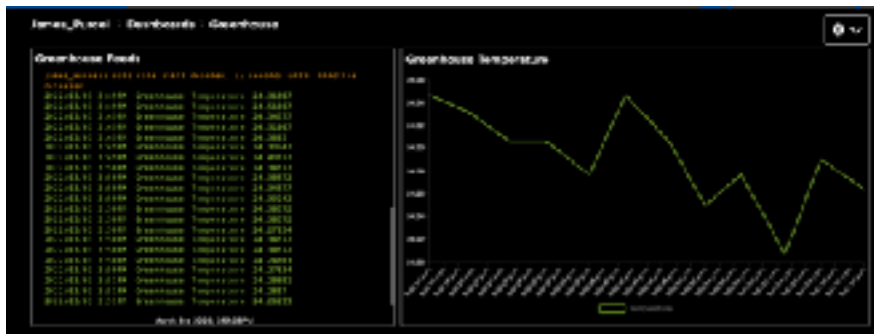


For the sake of simplicity, I have removed the pump control blocks in order to concentrate on only the data being sent to Adafruit.io.

If we click on each of the feeds we can see a chart of the data streams but when we go back to the dashboard we get a data throttle warning. This is an issue with the free service of adafruit.io and is caused by four data streams being sent at the same time.

Unfortunately the only way around this is to remove three of the streams leaving only the temperature stream available.

Now if we go into adafruit.io and view the dashboard we see live the live data appearing without a throttle warning.



Here I have added a line graph to show how the temperature is fluctuating.

Useful Forms.

Azure IoT Planner

Display Name	Name	Capability	Semantic Type	Schema
Shown as Data type Heading.	Data Type Name.	Property, Telemetry, Command	Type of reading (Temp, Pressure, Speed, etc)	Double, Float, Integer or Long
Temperature	Temperature	Telemetry	Temperature	Double
Water Pump	Pump	Command	N/A	N/A

You are free to Copy and print this form for private use.

Access Key List.

Azure Primary Access Key:

Azure Secondary Access Key:

Azure Stream End Points:.....

.....

.....

.....

.....

.....

.....

EZData Access Token:

.....

.....

MQTT User ID:

MQTT Password:

.....

MQTT Access Key if required (replaces password):

.....

.....

.....

MQTT Notes:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

In Closing.

Thank you for taking the time to read through my guide. If you have any questions, please feel free to ask and I will put the answer in the next update of the book that is due after the UIFlow 2.X update.

If you have any ideas and suggestions on improving these guides please let me know.

If you found this guide useful you can subscribe to my Youtube channel found here:

[https://www.youtube.com/channel/
UCn5qzjLQdVz9K8SM0ojJAEA](https://www.youtube.com/channel/UCn5qzjLQdVz9K8SM0ojJAEA)

You can find me on [hackster.io](https://www.hackster.io/AJB2K3) here: [https://www.hackster.io/
AJB2K3](https://www.hackster.io/AJB2K3)

In the official FB group here: [https://www.facebook.com/
groups/m5stack](https://www.facebook.com/groups/m5stack)

On the Official forums found here: [https://
community.m5stack.com](https://community.m5stack.com)