Air Quality, Lora and Azure Functions

Making a useful connected device with Azure.

Rob Miles

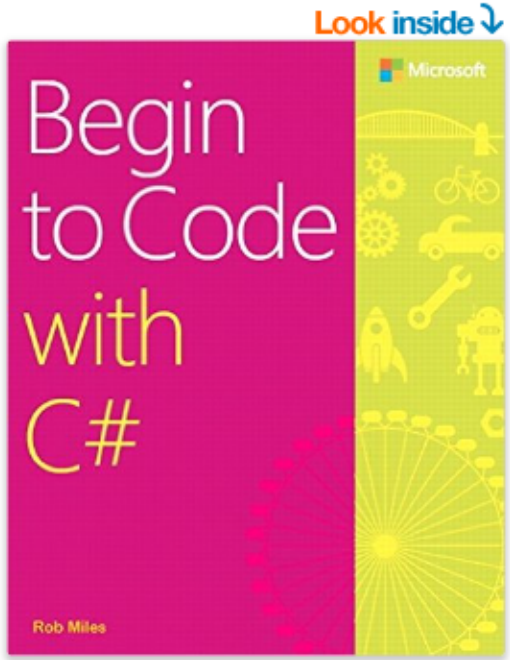
www.robmiles.com

Overview

- About Rob
- Why make an air quality sensor?
- How do you measure air quality?
- Building a device
- Connecting a device using MQTT
- The Azure IoT Hub and MQTT
- Using Azure Functions with MQTT
- LoRa and Azure

About Rob:

- Taught Computer Science at Hull University for many years
  - In charge of twisting minds and crushing dreams
- A Microsoft MVP
- Blogs at: www.robmiles.com
- Tweets at: @robmiles
- Writes books…..




See all 2 images

Begin to Code with C# Paperback – 9 Sep 2016

by Rob Miles (Author)

★★★★★ 1 customer review

See all formats and editions

Kindle Edition	Paperback
£15.86	£16.69
Read with Our Free App	8 Used from £10.77 39 New from £11.05

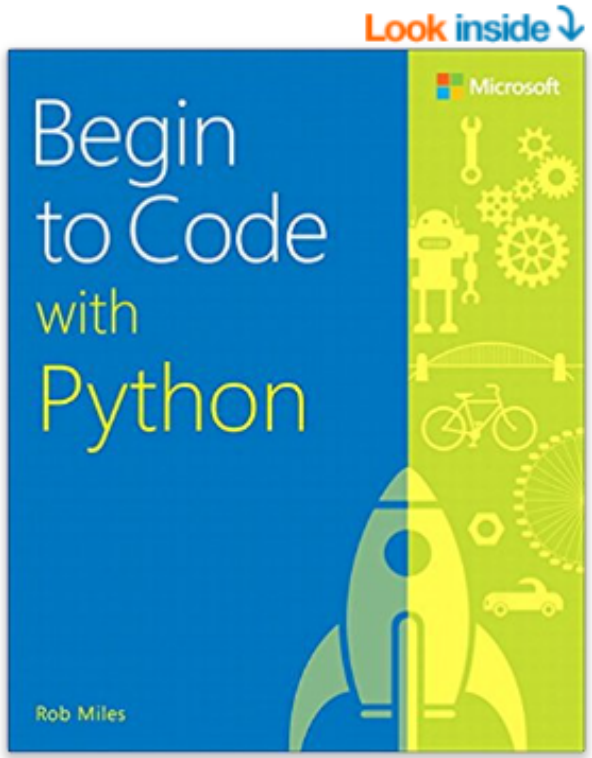
Want it delivered by tomorrow, 23 Nov.? Order within 2 hrs 30 mins and choose **One-Day Delivery** at checkout. Details

Note: This item is eligible for **click and collect.** Details

Become a C# programmer—and have fun doing it!

Start writing software that solves real problems, even if you have absolutely no programming experience! This friendly, easy, full-color book puts you in total control of your own learning, empowering you to build unique and useful programs. Microsoft has completely reinvented the beginning programmer's tutorial, reflecting deep research into how today's beginners learn, and why other books fall short. *Begin to Code with C#* is packed with innovations, from its "Snaps" prebuilt operations to its "Make Something

Read more



Begin to Code with Python Paperback – 8 Dec 2017

by Rob Miles (Author)

6 reviews from Amazon.com

See all 2 formats and editions

Kindle Edition	**Paperback**
£18.99	**£29.99** prime
Read with Our **Free App**	5 Used from £30.19 29 New from £18.00

Want it delivered by Sunday, 6 May? Order within 23 hrs 57 mins and choose **One-Day Delivery** at checkout. Details

Note: This item is eligible for **click and collect.** Details

This full-color book will inspire you to start solving problems and creating programs with Python, even if you have absolutely no programming experience. It's not just friendly and easy: it's the first Python beginner's guide that puts you in control of your own learning, and empowers you to build unique programs to solve problems you care about.

Why make an Air Quality Sensor?

How do we currently measure air quality?

- The Met Office weather forecast and climate prediction model has been developed to include air quality forecasting in a new model configuration called AQUM.
- Air quality is determined by the following factors:
  - Emissions of pollutants
  - Transport and dispersion of pollutants by winds
  - Chemical reactions amongst reactive gases and aerosols
  - Removal processes, such as rain and deposition on surfaces.
- The Met Office model uses UK and European maps of annual average pollutant emissions to simulate the release of chemical species into the atmosphere.

Calculated readings

- The thing to remember here is that a lot of these readings are created by the use of software models
- There are some readings that are entered into the system, but these are few and far between
- We thought it might be interesting to try and find out if we could use cheap air quality sensors to improve on the resolution of the readings and learn things about local air quality
- It has turned out to be very interesting….

Good and bad air quality

- This table shows the mapping between air quality values and what they mean for us
- “Professional” sensors will also read the amount of Nitrogen Oxide and ozone
- These sensors are quite expensive and so we thought we’d start with particles

AQI	$PM_{2.5}$ ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	Air Quality Descriptor
0–50	0.0–15.4	0–54	Good
51–100	15.5–40.4	55–154	Moderate
101–150	40.5–65.4	155–254	Unhealthy for Sensitive Groups
151–200	65.5–150.4	255–354	Unhealthy
201–300	150.5–250.4	355–424	Very unhealthy

Good and bad air quality

- This table shows the mapping between air quality values and what they mean for us
- “Professional” sensors will also read the amount of Nitrogen Oxide and ozone
- These sensors are quite expensive and so we thought we’d start with particles

AQI	$PM_{2.5}$ ($\mu g/m^3$)	PM_{10} ($\mu g/m^3$)	Air Quality Descriptor
0–50	0.0–15.4	0–54	Good
51–100	15.5–40.4	55–154	Moderate
101–150	40.5–65.4	155–254	Unhealthy for Sensitive Groups
151–200	65.5–150.4	255–354	Unhealthy
201–300	150.5–250.4	355–424	Very unhealthy

Measuring air quality

- We decided to measure the density of smoke particles in the air
- These are produced by vehicles and also by burning fossil fuels
- The ones we are interested in have a radius of less than 2.5 microns
- This is around a 25^{th} of the width of a human hair
  - This is smaller than pollen grains (they are 10 microns)
- We can buy sensors that can detect these particles and give an output that tells us the number of micrograms of particles there are per cubic meter of air

Technology as an agent of change

- I strongly believe that you can use technology to change the way that people behave, and to make things better
- If we can make people better informed of the consequences of their actions we might be able to change what they do
  - Plastic bags and doggy poo are good examples of successes in this area
- For example, if we end up producing evidence that a large number of vehicles at the school gates produces peaks in air pollution, perhaps people might not use their cars to take their kids to school quite so much

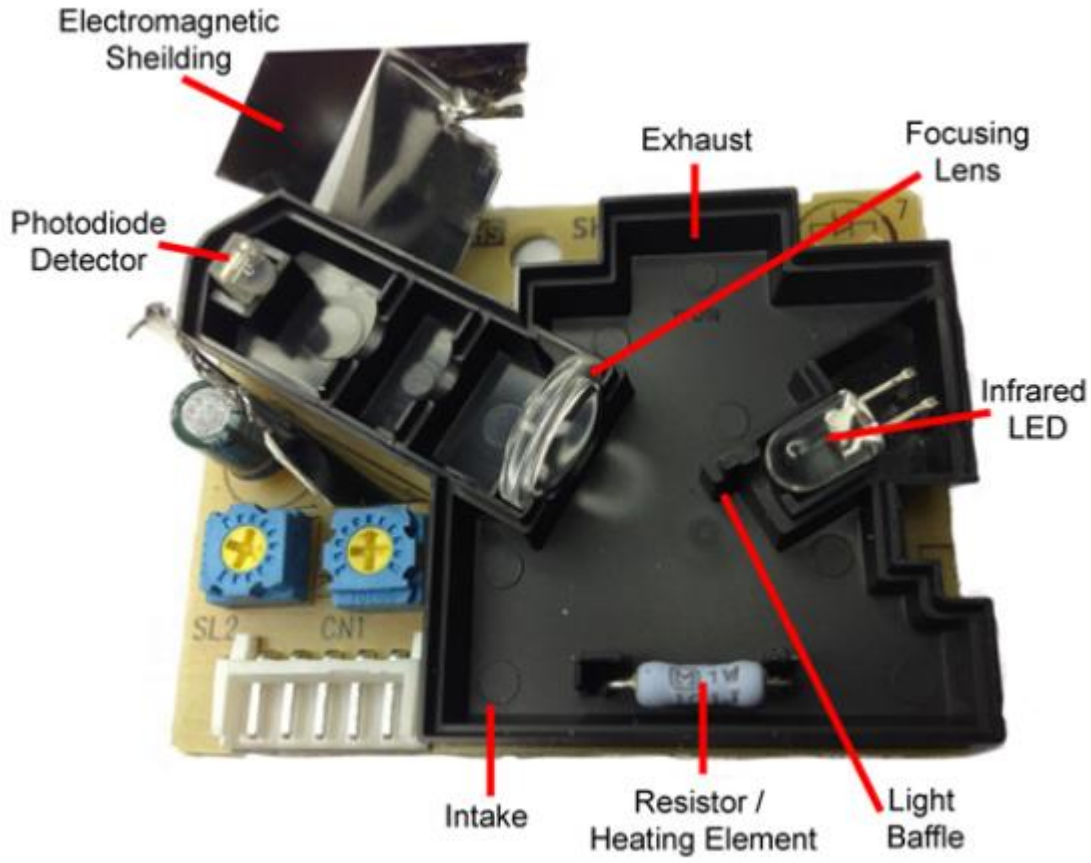
A General Note about projects

- If you want to learn how to use a particular technology one of the best ways is to try and build something
- Just learning stuff by reading books and watching YouTube videos does not work – you need to get making things
- When you start building things you find out what the **real** problems are
- We thought that the hard part of our project would be making the sensors
- This turned out not to be the case, but more of that later……

How do you measure Air Quality

How a sensor works

- The dust sensors work by shining a beam of light through the air and then detecting the light scattered by the particles in the air
- The air can be moved by a fan, or by using a heater that causes convection
- You can buy sensors that work like this for around 5 pounds



The sensor we like

Nova PM sensor SDS011 High precision laser pm2.5 air quality detection sensor module Super dust dust sensors, digital output

4.9 (354 votes) | 1183 orders

Price: £ 13.73 / piece

Shipping: Free Shipping to United Kingdom via AliExpress Standard Shipping

Estimated Delivery Time: 20-40 days

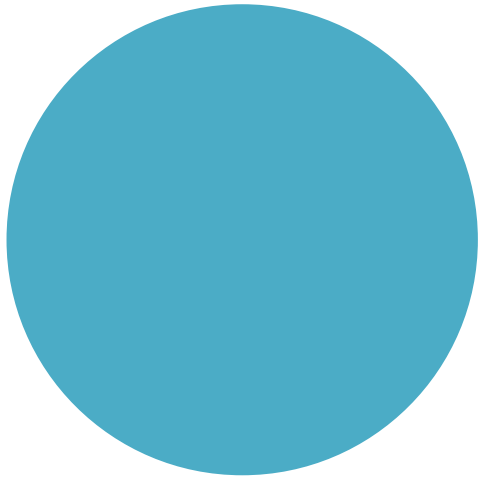
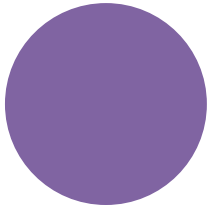
Quantity: − 1 + piece (89199 pieces available)

Buy Now | Add to Cart | 1369

Return Policy: Returns accepted if product not as described, buyer pays return shipping fee; or keep the product & agree refund with seller. View details

Seller Guarantees: On-time Delivery 60 days

- This is the Nova SDS011 sensor
- It has been used successfully on numerous air quality projects
- Has a pipe connection for the inlet and a fan that moves the air
- Can connect it to a PC or an embedded device

Building a device

Building a sensor node device

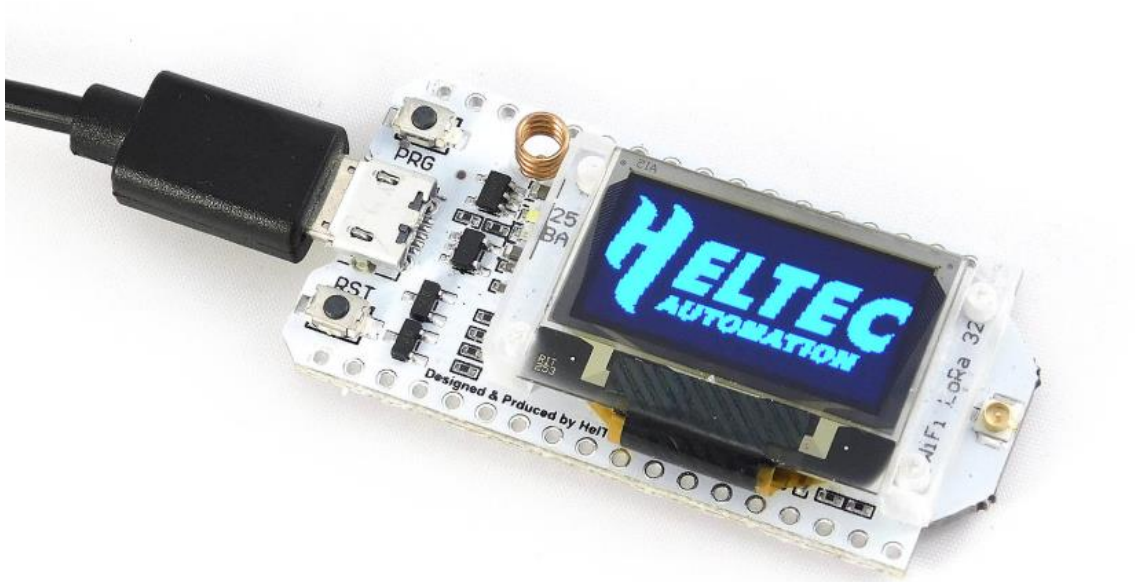
- A sensor node needs a micro-controller to get data from the sensor and send it into the server backend
- It could do other things too, for example drive a display
- Hardware for embedded devices is incredibly cheap
- They are rather powerful
- They can be programmed using Arduino platform
  - This provides a whole set of libraries and a development environment for embedded devices

The esp8266 is an awesome chip….

- Lots of WiFi options
  - WiFi client over a serial port
  - Fully programmable in C++ just like the Arduino
  - WiFi access point and web server
  - Support for UDP, TCP, secure sockets and mDNS
  - Very easy to use with many examples

- Making a connected client device
  - Lots of ways to do this
  - We're going to use MQTT but you can use it as a web server, or even a WiFi access point (or both)
- I use the Wemos platform – around two pounds fifty a pop…

Enter the ESP32



- The company that made the ESP8266 has now made its successor - the ESP32
- This is a dual core device with 16M of RAM clocked at 240MHz
- It costs around a fiver
- You can program it with the Arduino IDE or Python
- The Heltec version costs a bit more (12 pounds) but includes an OLED screen and a LoRa (Low powered Radio) device of which more later

The m5stack for pretty hardware

- If you want to make a device that you can show people, take a look at the m5stack device
- This is ESP32 based but it has a colour screen and some buttons, as well as a microphone and a speaker
- You can “stack” modules underneath the core module to add extra functions

The Azure IoT Devkit for industrial strength

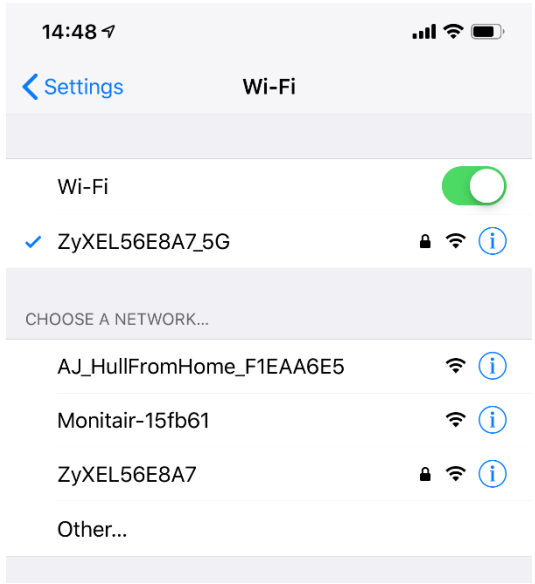
- This is a nifty little device that has a beefy processor and a nice collection of sensors
- It has a two colour OLED display and will fit into a BBC Microbit socket
- It contains secure storage
  - It can be made impossible to extract information from a device in the field
  - Devices can be secured using X500 certificates
- Some very good examples available from the Azure IoT team

Programming your connected hardware

- Making your own connected hardware is cheap and fun
- You can program an embedded device in a variety of languages
- I like to use C++
- The software is free and can be used from within Visual Studio and Visual Studio Code
- The programming environment works with a variety of devices
- There are pre-built libraries for lots of devices and services

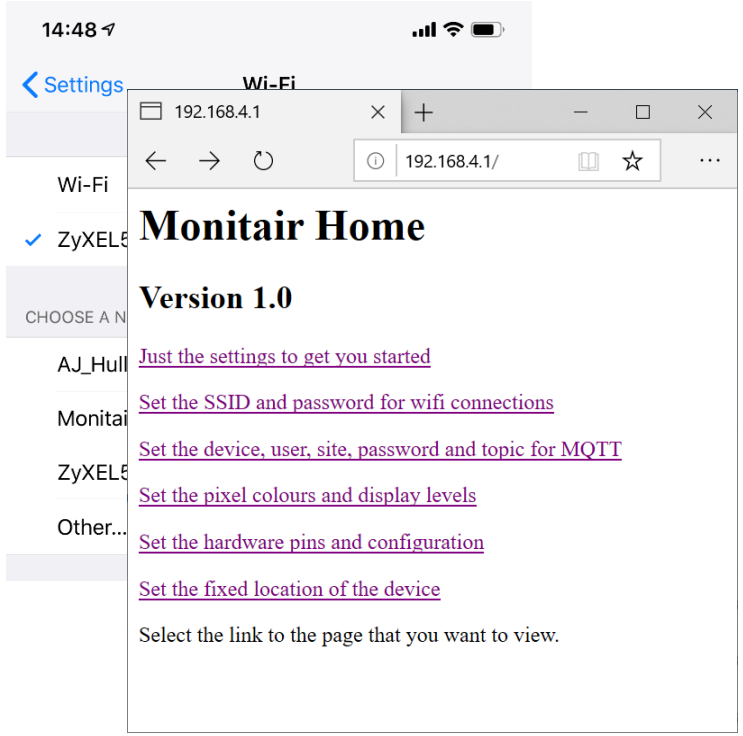
The Monitair Sensor Node Firmware

- The sensor has been designed to be used as a connected appliance
- It can operate as a WiFi access point for network configuration



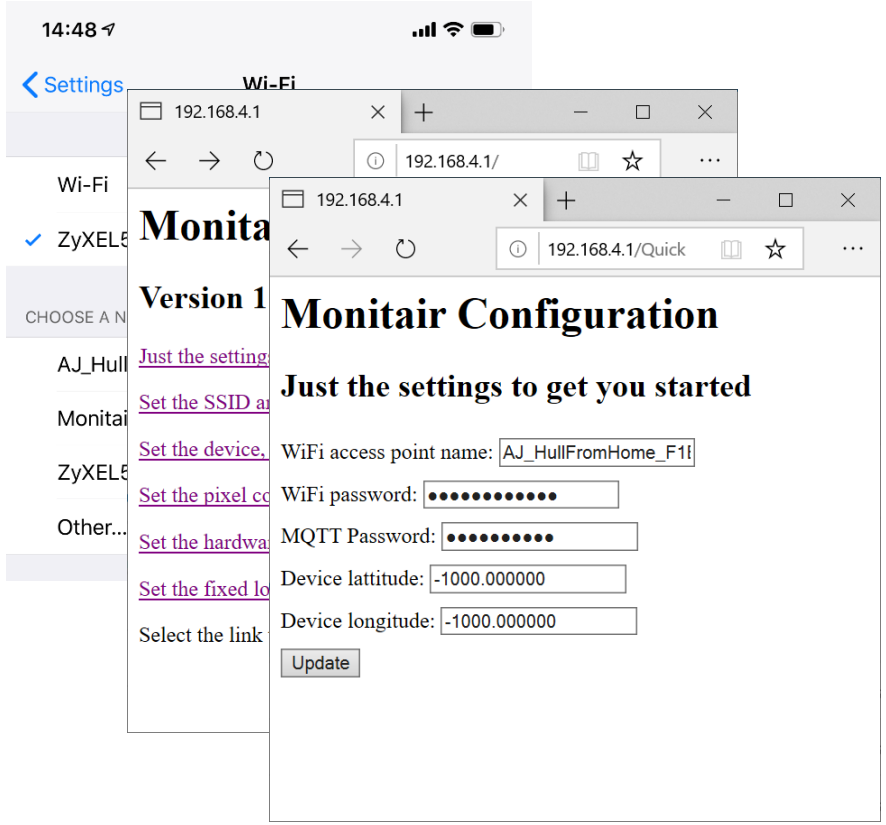
The Monitair Sensor Node Firmware

- The sensor hosts a web page from which the user can configure WiFi and MQTT settings



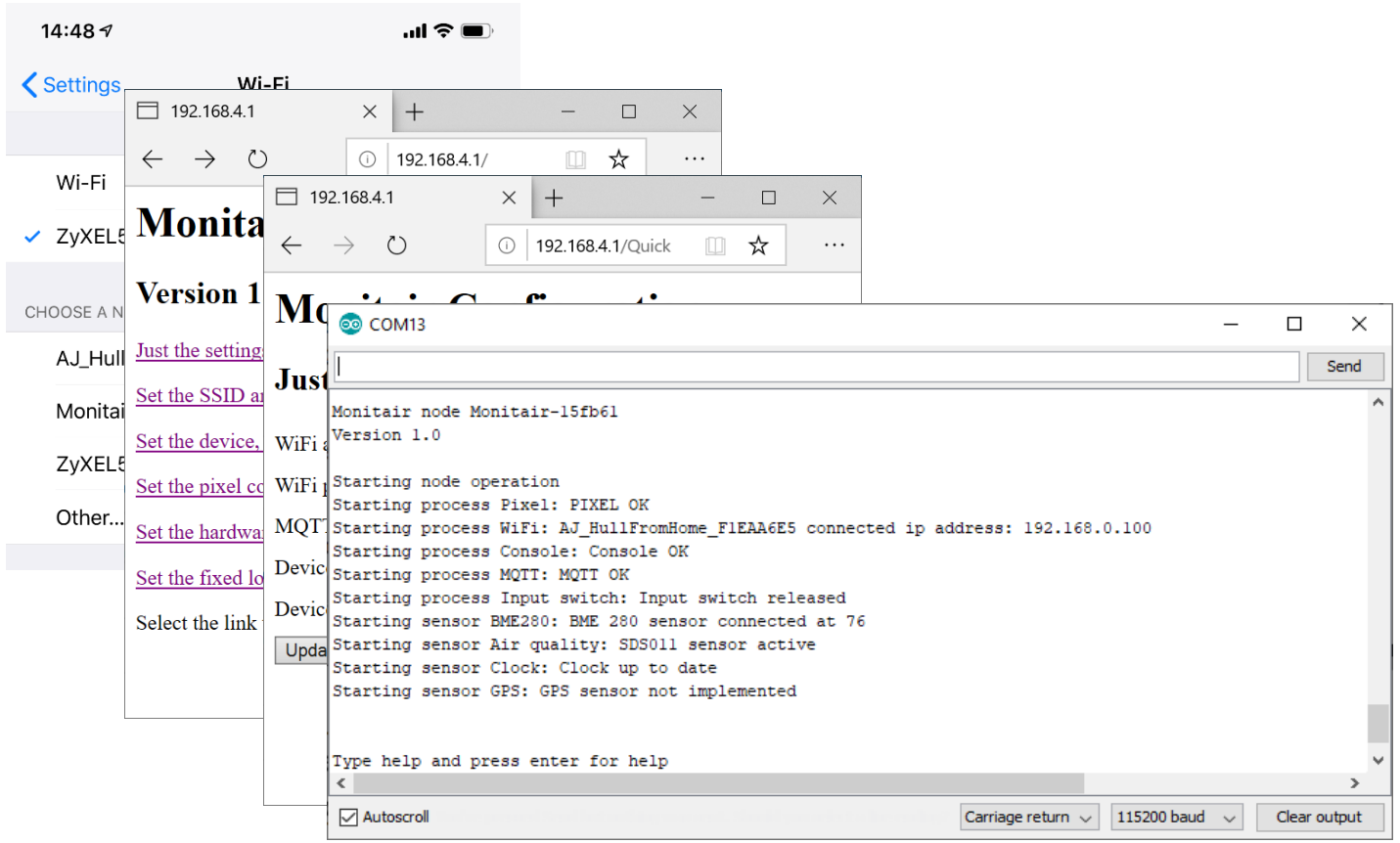
The Monitair Sensor Node Firmware

- There is also a “quick configuration” page for popular settings

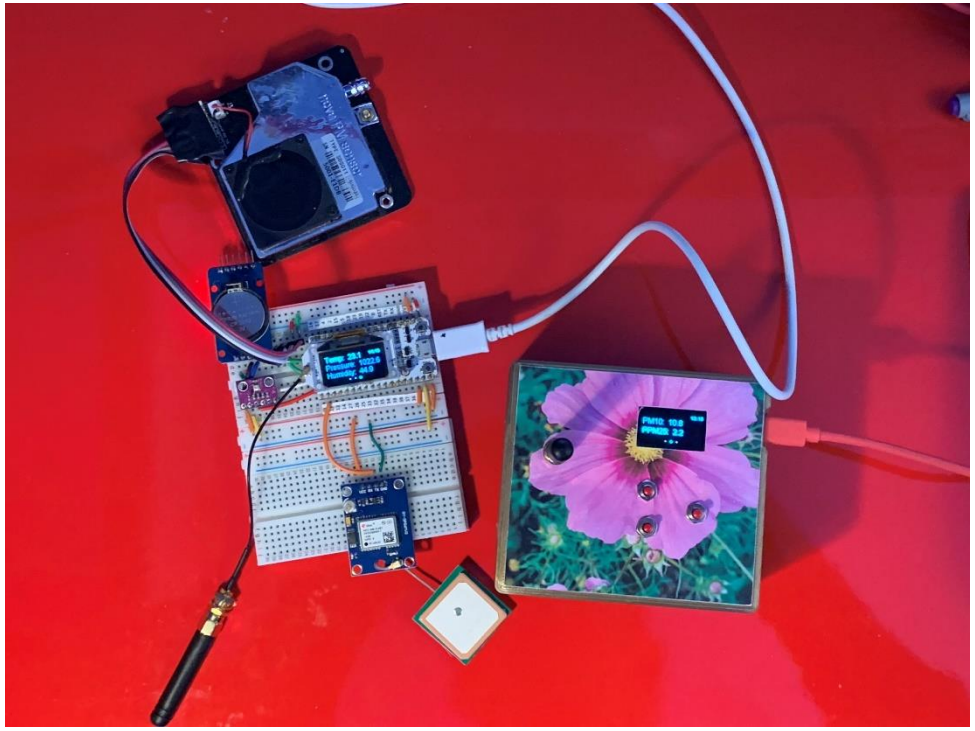


The Monitair Sensor Node Firmware

- ..as well as a serial interface for factory configuration and testing



My first sensor



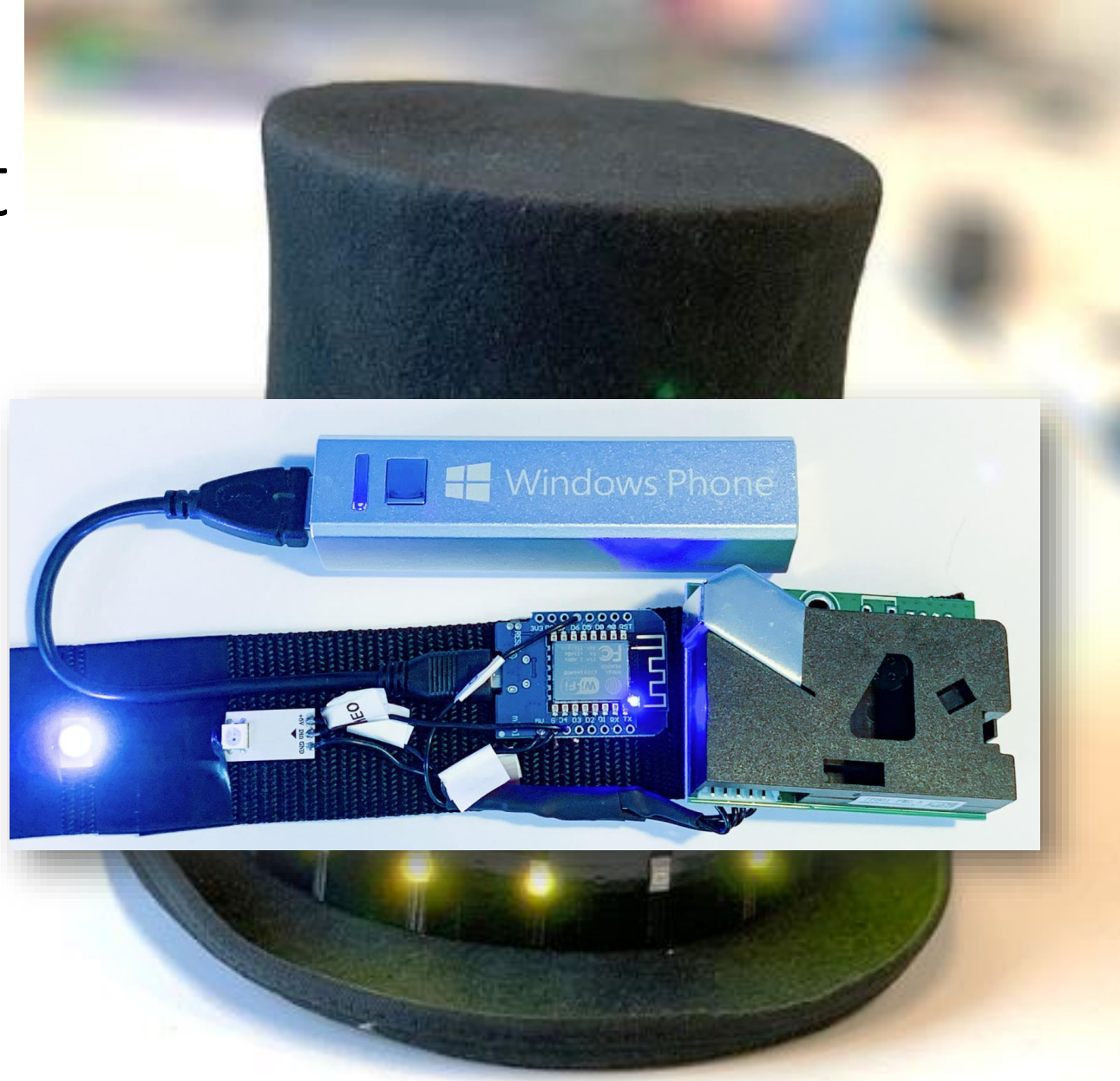
- This is my first “proper” sensor in “breadboard” and finished versions
- It measures temperature, pressure, humidity and particle density
- The readings can be sent over MQTT or LoRa to a server
- It can be configured using strings of JSON that can be sent over MQTT, serial connection or LoRa
- It also has a GPS receiver to tag readings with their location

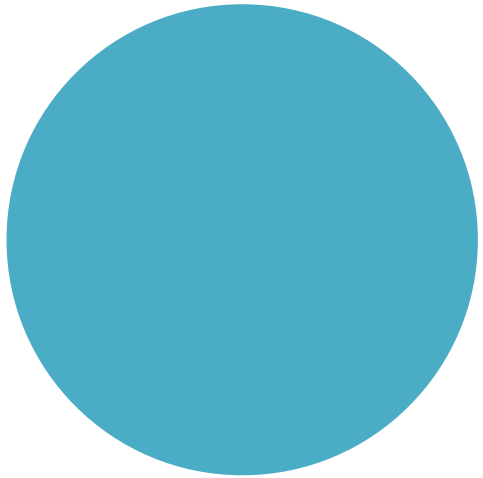
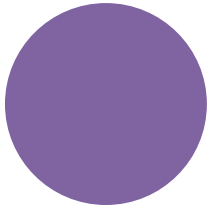
The Air Quality Top Hat

- I built the Monitair software into a top hat
- It uses a ZPHO1 sensor
  - This is not very reliable
  - But it only costs around a fiver
- The hat uses neopixels to show the air quality around the wearer

The Air Quality Top Hat

- I built the Monitair software into a top hat
- It uses a ZPHO1 sensor
  - This is not very reliable
  - But it only costs around a fiver
- The hat uses neopixels to show the air quality around the wearer
- The Wemos and the sensor are attached to a hatband



Connecting a device using MQTT

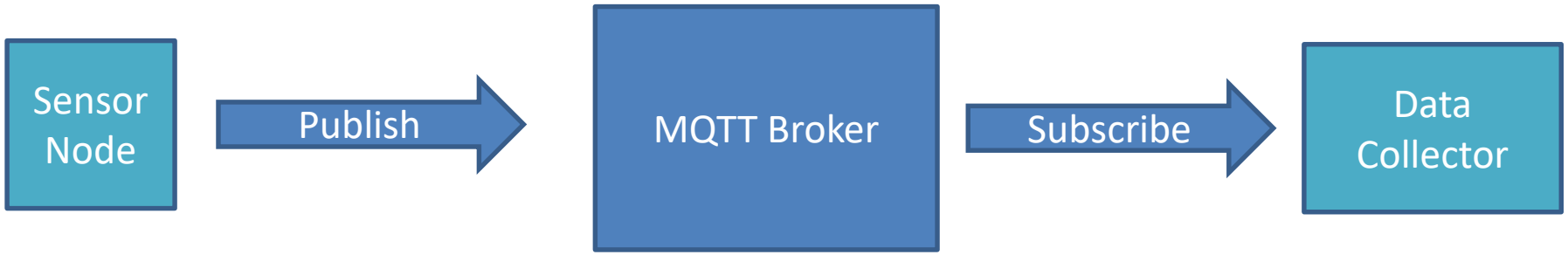
IoT device connectivity

- The devices that we have looked at all have WiFi
- You can use them to create network connections so they can use datagrams or connections
  - They will work as web servers
  - They can also support secure sockets
- You can connect to network services using restful connections
- However, the IoT community makes a lot of use of MQTT (Message Queue Telemetry Transport)
- This is a very easy way to hook sensors and actuators together

Message Queue Telemetry Transport

- MQTT is a way to connecting sensors to endpoints
  - It has a publish/subscribe architecture
- The communication can run over serial or WiFi and is based on a simple packet structure
- People have different opinions of how good it is, but it is very popular and also supported by the Azure IOT Hub
- It also runs (surprise surprise) on the esp8266
- It is a great way to create cheap, connected, sensors

The MQTT broker



- The MQTT broker accepts messages and passes them on to subscribers that have registered as listening to an endpoint topic
- Sensor nodes can subscribe to topics so that they can be sent commands
- MQTT messages are just blocks of bytes
- We encode them into JSON strings

Connecting Arduino devices to MQTT

- I use **PubSubClient** for Arduino devices
- You can add it to your Arduino solution as you would any other library
- It works well on the ESP8266 and ESP32 devices
- It can talk to any MQTT broker, including Azure the one provided by Azure IoT Hub (as long as you use secure sockets for the connection)
- There is also a Microsoft client you can add to an Arduino project

Connecting to an MQTT broker

```
mqttPubSubClient->setServer(settings.mqttServer, settings.mqttPort);
mqttPubSubClient->setCallback(callback);
mqttPubSubClient->connect(settings.deviceName, settings.mqttUser,
                          settings.mqttPassword);
```

- Setting up an MQTT client is simple enough
- We need some configuration information that identifies the device to the broker

Set the server

```
mqttPubSubClient->setServer(settings.mqttServer, settings.mqttPort);
mqttPubSubClient->setCallback(callback);
mqttPubSubClient->connect(settings.deviceName, settings.mqttUser,
                          settings.mqttPassword);
```

- This statement sets up the server
- The **`mqttServer`** element is the network address of the server
- The **`mqttport`** is the TCPIP port to be used
  - Open data 1883
  - Secure Sockets 8883

Assign a function for callbacks

```
mqttPubSubClient->setServer(settings.mqttServer, settings.mqttPort);
mqttPubSubClient->setCallback(callback);
mqttPubSubClient->connect(settings.deviceName, settings.mqttUser,
                          settings.mqttPassword);
```

- This statement identifies the function to be called when the broker sends an MQTT message to a topic the device has subscribed to
  - Our application must contain a function called **`callback`**
- This is how we can use MQTT to control a device

Connect the device

```
mqttPubSubClient->setServer(settings.mqttServer, settings.mqttPort);
mqttPubSubClient->setCallback(callback);
mqttPubSubClient->connect(settings.deviceName, settings.mqttUser,
                          settings.mqttPassword);
```

- This call actually makes the connection
  - The device name is the MQTT device name
  - On a standard MQTT broker the **`mqttUser`** is the username for the broker and the **`mqttPassword`** is the password
  - This is not particularly secure – anyone with the broker username and password can add their own devices and subscribe to endpoints

Sending MQTT Messages

```
mqttPubSubClient->publish("airquality/data", buffer);
```

- A message is a string of bytes which is published on a given topic
- Topics are hierarchical and can contain wildcards which allow a subscriber to receive from collections of sources

Receiving MQTT Messages

```
mqttPubSubClient->subscribe(settings.mqttSubscribeTopic);
```

- The node can nominate a topic which it is interested in
- The broker will relay messages sent to that topic onto that node
- This will cause the callback function to be called each time a message for that topic arrives

Keeping the MQTT connection alive

```
mqttPubSubClient->loop();
```

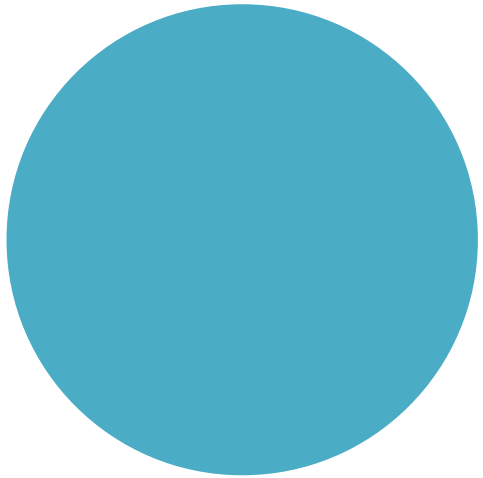
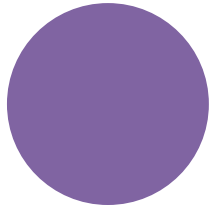
- MQTT clients send a message every 9 seconds to keep the MQTT connection alive
- The node must make regular calls to the loop method in the MQTT connection object to check for incoming messages and send the heartbeat message
- This is how the broker determines which clients are connected

MQTT housekeeping

- There are a number of different Quality of Service (QoS) levels
  - QoS 0 – the message is sent once and will be received once or never (like a datagram)
  - QoS 1 – the message will be delivered at least once (sender waits for an acknowledgement and resends if one is not received)
  - QoS 2 – the message will be delivered exactly once
- Azure IoT hub uses QoS level 2
- Stations can nominate a “last wishes” message to be sent to subscribers if their connection is lost

MQTT Resources

- I use **PubSubClient** for the Arduino based sensor nodes
  - Install it as any other Arduino library
- **Eclipse Paho** has MQTT software for a wide range of platforms
  - https://www.eclipse.org/paho/
- If you want to run your own MQTT broker (perhaps for a home network) take a look at **Mosquitto**
  - https://mosquitto.org/
- The **NodeRed** tool is a great way to create flows of data between devices
  - https://nodered.org/

The Azure IoT Hub and MQTT

Azure, MQTT and embedded devices

- Azure IoT Hub provides a complete IoT device management framework
- This includes device management and simulation
  - Devices can be created and managed securely and programmatically
  - You can use it to create “proper” IoT device networks
- Azure and MQTT
  - The Azure IoT Hub will interact with MQTT messages
  - These can be passed on to your backend Azure applications and Azure applications can also be sent to MQTT devices

The Azure IoT Hub

- MQTT enabled devices can connect to Azure IoT hub, publish messages and subscribe to topics
- The received messages then allow you to do lots of lovely things with your connected devices
- There is also a lot of device management support too
- … and it is all done over secure channels

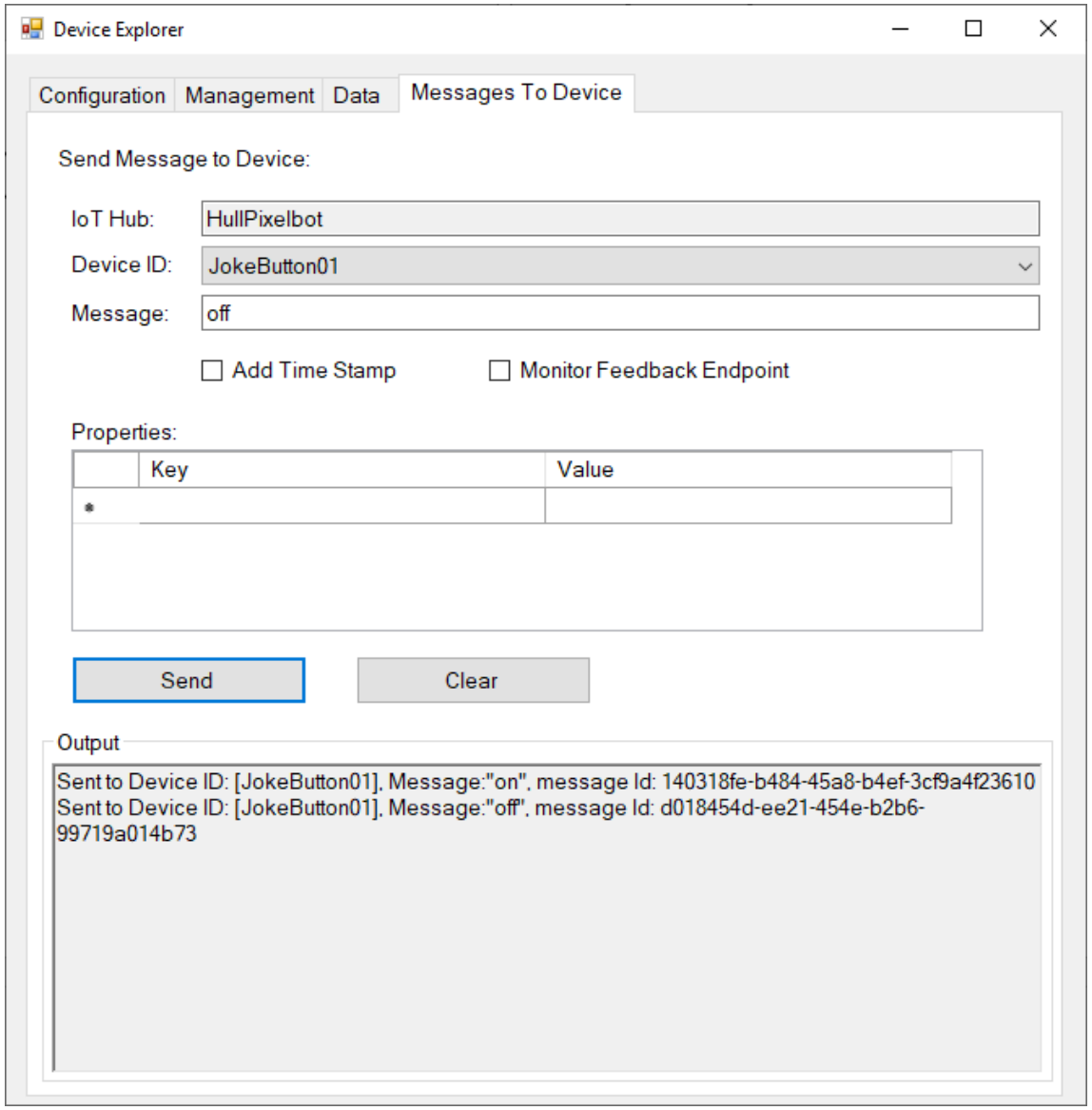
Connecting to MQTT – Azure Iot Hub

```
mqttPubSubClient->setServer(settings.mqttServer, settings.mqttPort);
mqttPubSubClient->setCallback(callback);
mqttPubSubClient->connect(settings.deviceName, settings.mqttUser,
                          settings.mqttPassword);
```

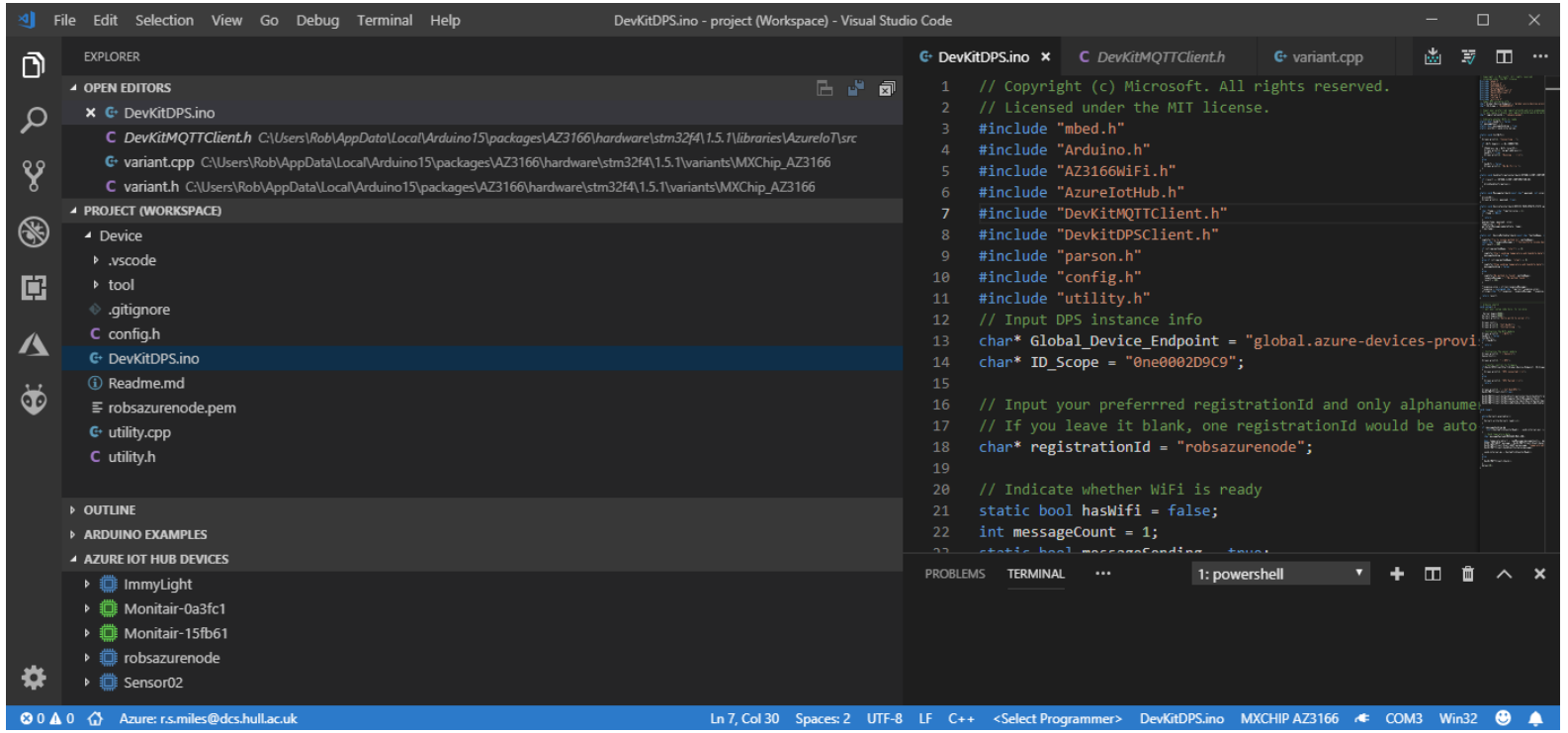
- `mqttServer` – address of server
- `mqttPort` – 8883 (secure sockets only)
- `deviceName` – name of the device
- `mqttUser` – unique username for device
- `mqttPassword` – Shared Access Signature (SAS) key for the device

Device Explorer

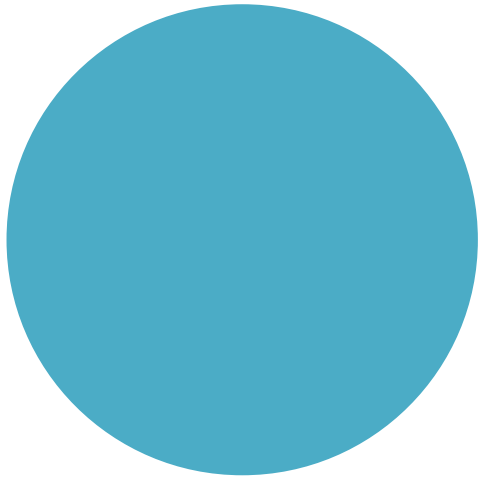
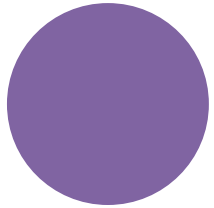
- The device explorer provides device management and testing
  - This is available in source form
- We can view messages from connected clients and send messages to them as well
- This is not the only way to provision devices
- There is also an api you can use to build a workflow if you have lots of devices



Visual Studio Code



- Visual Studio Code is a great place to create solutions
- You can register and monitor devices on the Azure IoT Hub

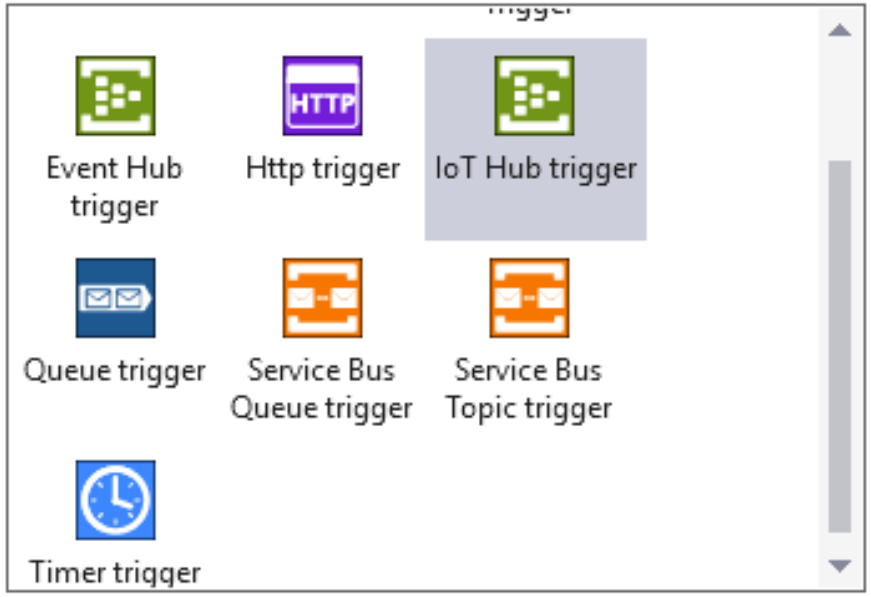
Using Azure Functions and MQTT

Azure Functions

- An Azure Function is a lump of code that runs in the cloud when an event occurs
- As a developer you just have to create the code and deploy it into Azure
- There is no need to create a server
- You only pay for the time your function is active, not the time that your server is running

Azure Functions Events

- You can fire off an Azure function on a variety of trigger events including web requests and timed events
- We are going to trigger the function when an incoming message is received from a node
- The function will store the incoming value in Azure Table Storage



Azure Table Storage

- To keep thing simple I'm going to store the readings in Azure Table Storage
- This stores each reading as a row in a table
- Rows are defined by mapping a POCO (Plain Old CLR Object) value
- In my case I'm creating an instance of a C# class
- The instance must have two members used to index the table:
  - `PartitionKey` – broad categorisation of the data
  - `RowKey` – unique value for any given `PartitionKey` value

My Air Quality Reading


```
{ "dev":"Monitair-15fb61","temp":24.73,"humidity":41.33,
"pressure":1017.30,"PM10":13.00,"PM25":5.10,"timestamp":"1551432371"}
```


- This is the MQTT message that is sent by the sensor node to the server
- It is formatted using JSON
- This needs to be stored on the server

My Air Quality Class

```
public class AirQReading
{
    [JsonProperty("dev")]
    public string PartitionKey { get; set; }
    [JsonProperty("timestamp")]
    public string RowKey { get; set; }
    public float PM10 { get; set; }
    public float PM25 { get; set; }
    [JsonProperty("temp")]
    public float Temp { get; set; }
    [JsonProperty("humidity")]
    public float Humidity { get; set; }
    [JsonProperty("pressure")]
    public float AirPress { get; set; }
}
```

- The device property is mapped onto the **`PartitionKey`**
- The time property is mapped onto the **`RowKey`**

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- This is the function that receives data and store it in the table

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
        Connection = "IoTHubConnectionString")]EventData message,
        TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- This is the name of the function

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- Identifies the table into which the readings will be placed
- The storage connection string is given in the settings for the project

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- This is the MQTT topic to be monitored
- This monitors messages from all devices (# is a wildcard)

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- This maps to an SAS connection string held in the function settings file
- **You need to copy this setting to your function when you deploy it**

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
        Connection = "IoTHubConnectionString")]EventData message,
        TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- This is the message instance that will be delivered into the function call by Azure IoT hub

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- The function can write messages into this log

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- Convert the message payload into a string

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- Create an empty **`AirQReading`** instance

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

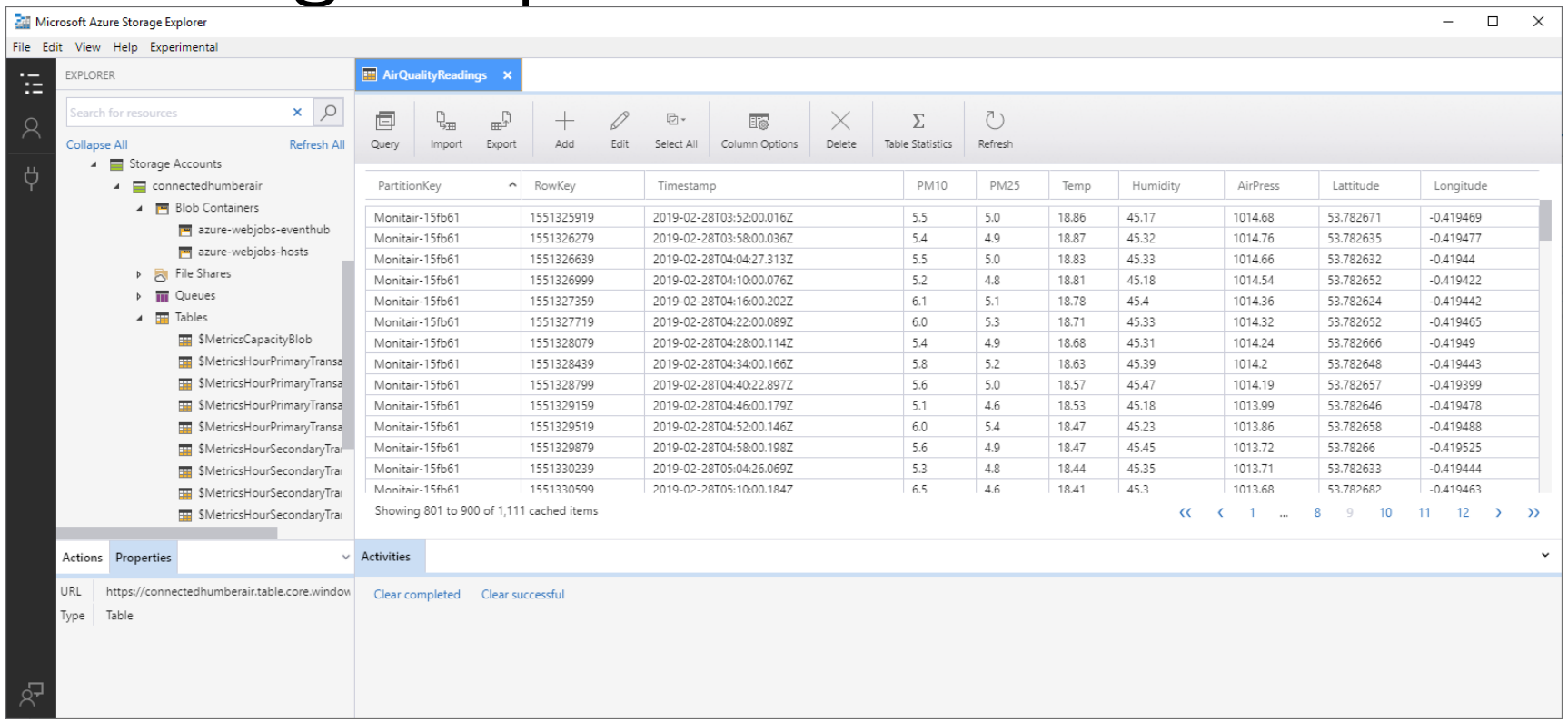
- Populate the **`AirQReading`** instance with values from the received JSON string (missing values are left at default values)

My Azure Function

```
[FunctionName("DataReceiver")]
[return: Table("AirQualityReadings")]
public static AirQReading Run([IoTHubTrigger("devices/#",
          Connection = "IoTHubConnectionString")]EventData message,
          TraceWriter log)
{
    string readingJson = Encoding.UTF8.GetString(message.GetBytes());
    AirQReading result = new AirQReading();
    JsonConvert.PopulateObject(readingJson, result);
    return result;
}
```

- Return the POCO object that will be stored in the table

Azure Storage Explorer



- You can use Azure Storage Explorer to view the received data

Demo

An Azure connected Air Quality top hat

LoRa and Azure

What is LoRa?

Lo w Powered Ra dio

Lo ng Ra nge

Low powered radio

- Designed for use in battery powered devices
  - Battery life measured in years
- LoRa radio transmitters are cheap and easy to add to a device
- Uses “Spread Spectrum Technology”
  - Messages are sent “below the noise” as packets of data
- Best regarded as a form of “SMS” message rather than a continuous telephone call
  - There are limits on the message size and the number of messages you can send in a given time

Long Range

- Range up to 15-20 km
  - (although this depends a lot on conditions – take it with a pinch of salt)
- Lora wavebands
  - **868 MHz for Europe**
  - 915 MHz for North America
  - 433 MHz band for Asia
- You don’t need a licence to use the LoRa band
  - But you should be using properly certified devices and not breach the usage conditions – if you’re doing this properly

LoRa “peer to peer” connection

- You can use LoRa to connect two devices together
  - Think of this as a car remote keyfob with a really long range
- Messages sent by one LoRa device will be received by the any other LoRa device that is listening
- You would need to devise your own station addressing scheme
- You may also need to add security in the form of packet encryption and verification

LoRaWan

- You can also use a LoRa device as part of a larger network
- A LoRa embedded device (an *endpoint*) will be associated with a given LoRa application
- Within an application each LoRa device has a unique address
  - If you were making a “cow tracker” you’d attach an endpoint to the cow
- Data between the endpoint and the gateway is encrypted
- A LoRa gateway forwards all endpoint messages to a LoRa server
- The server sends messages onto backend applications
- This forms a *LoraWAN* (LoRa Wide-Area Network)

LoRaWAN cow tracking

endpoints

Fit cows with LoRa endpoint devices that contain a GPS tracker and a LoRa wireless transmitter

Devices send location information every few hours

LoRaWAN cow tracking

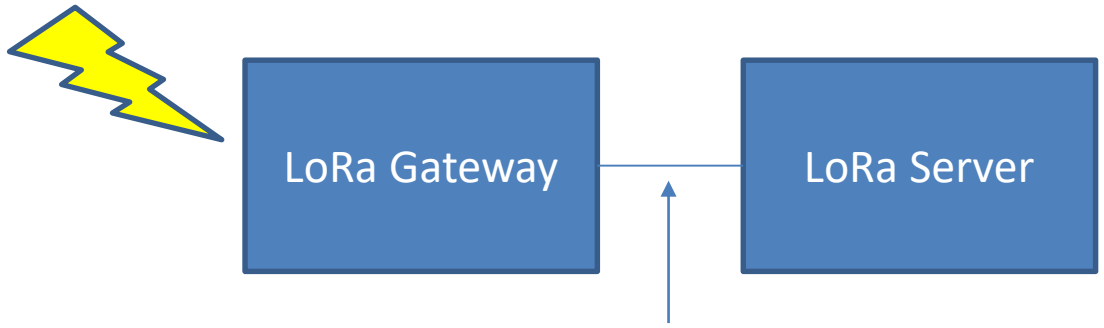
Endpoints send messages to a LoRa Gateway

endpoints

LoRaWAN cow tracking

LoRa Gateway forwards messages to the LoRa Server

endpoints

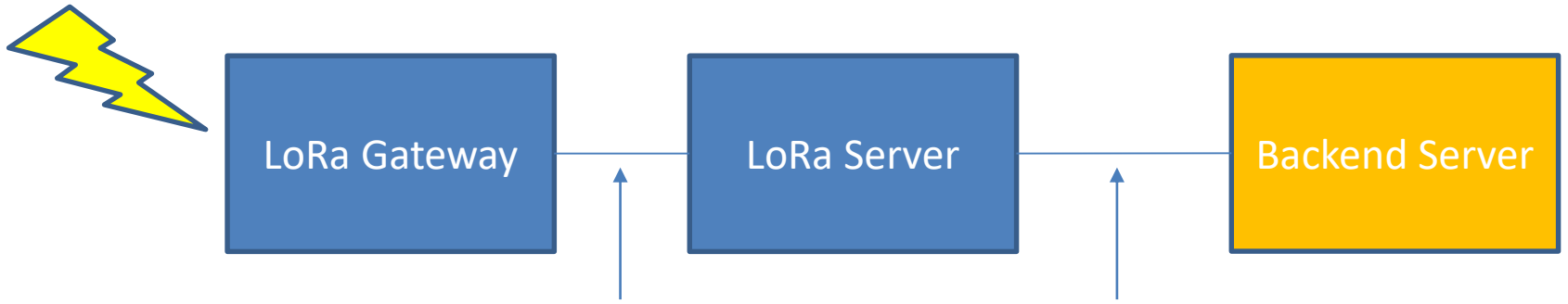


network connection

LoRaWAN cow tracking

LoRa Server sends messages to your backend applications

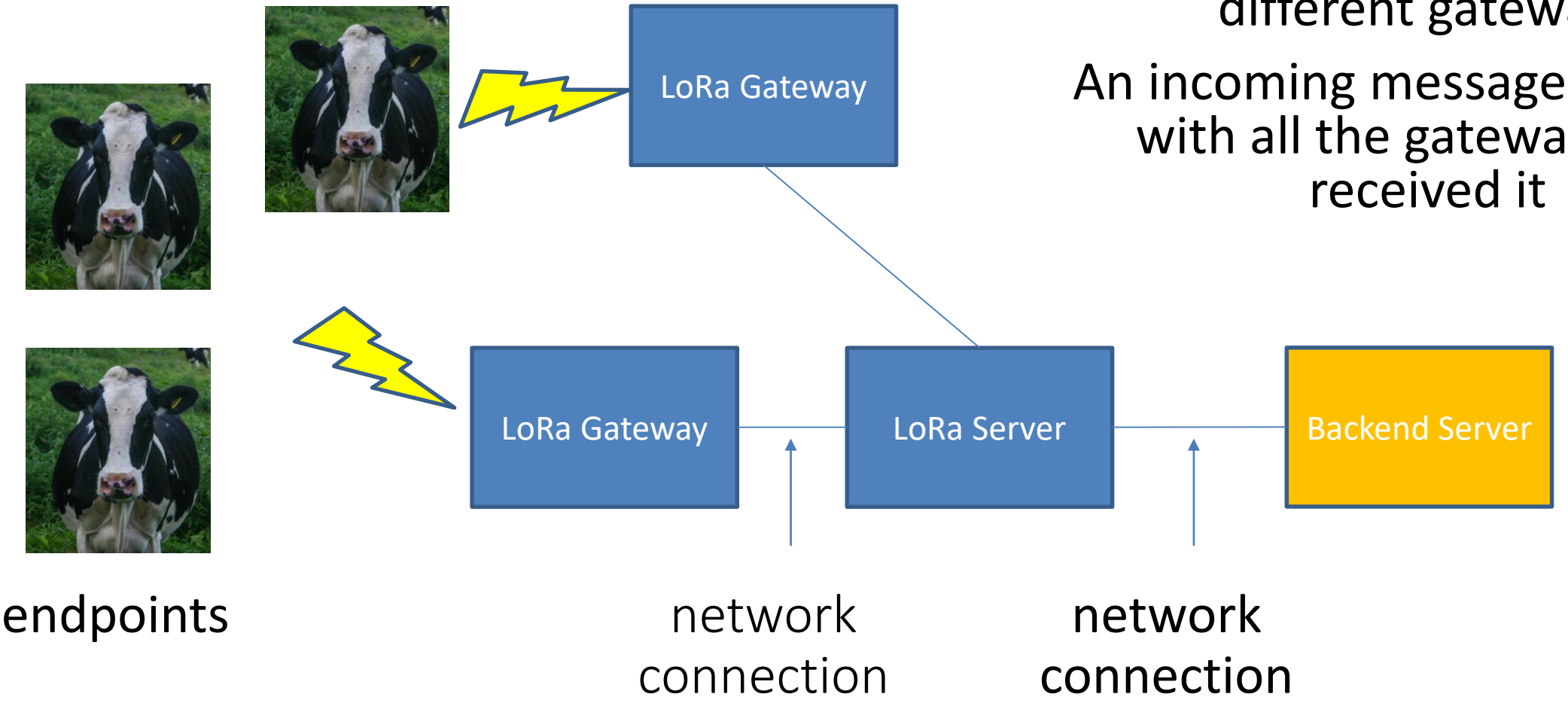
endpoints



network connection

network connection

LoRaWAN cow tracking



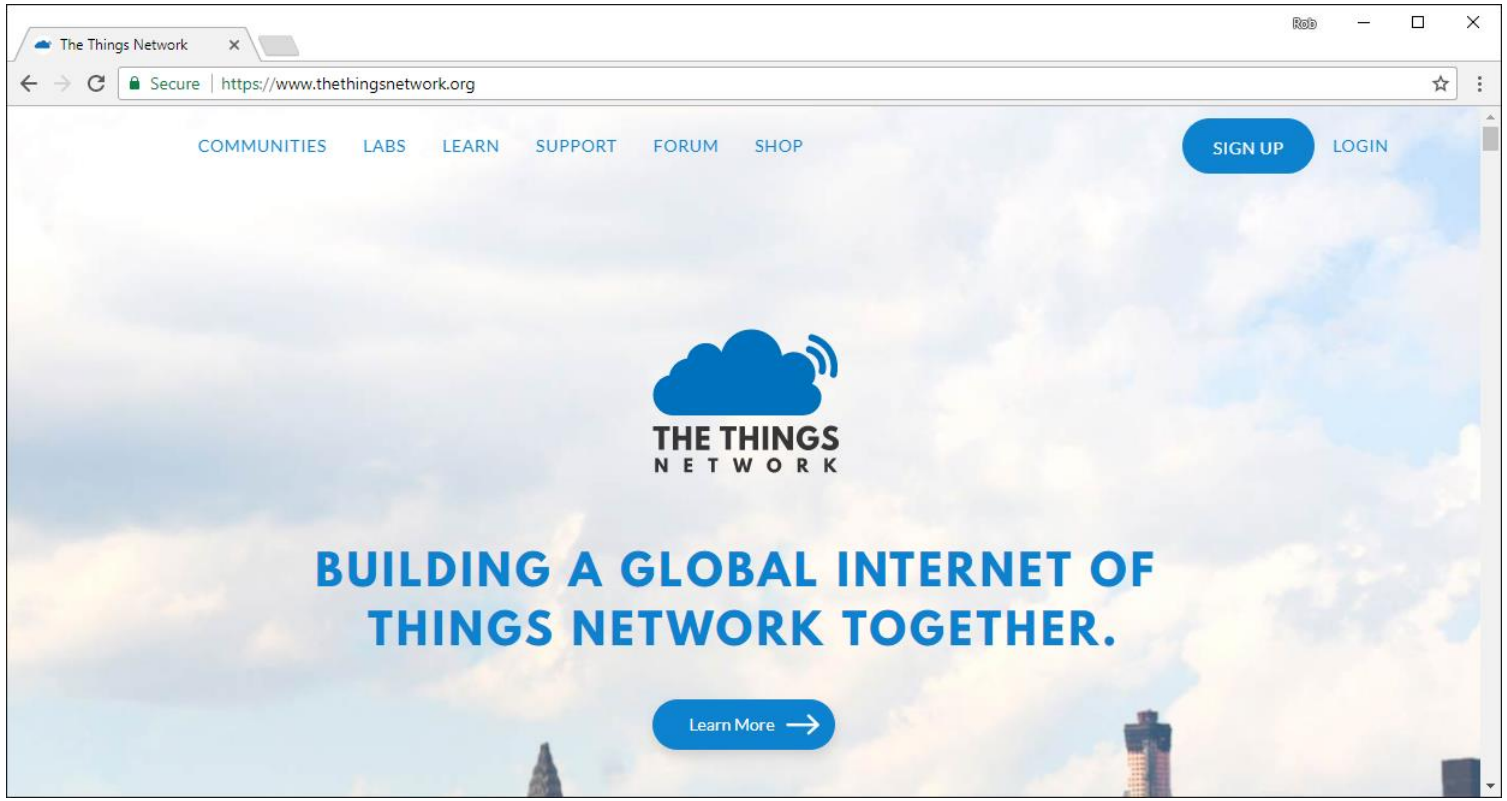
The Lora server manages multiple message from different gateways

An incoming message is tagged with all the gateways that received it

What is a gateway?

- A gateway has a LoRa radio receiver and a network connection
  - Receives messages from the endpoint and forwards them to a LoRa server
- You can use LoRa endpoint devices as primitive gateways
  - But they don’t expose the full functionality as they are only single channel devices
- The cheapest “proper” LoRa gateway is around 120 pounds and runs on a Raspberry Pi
- Best placed high up and outdoors

The Things Network



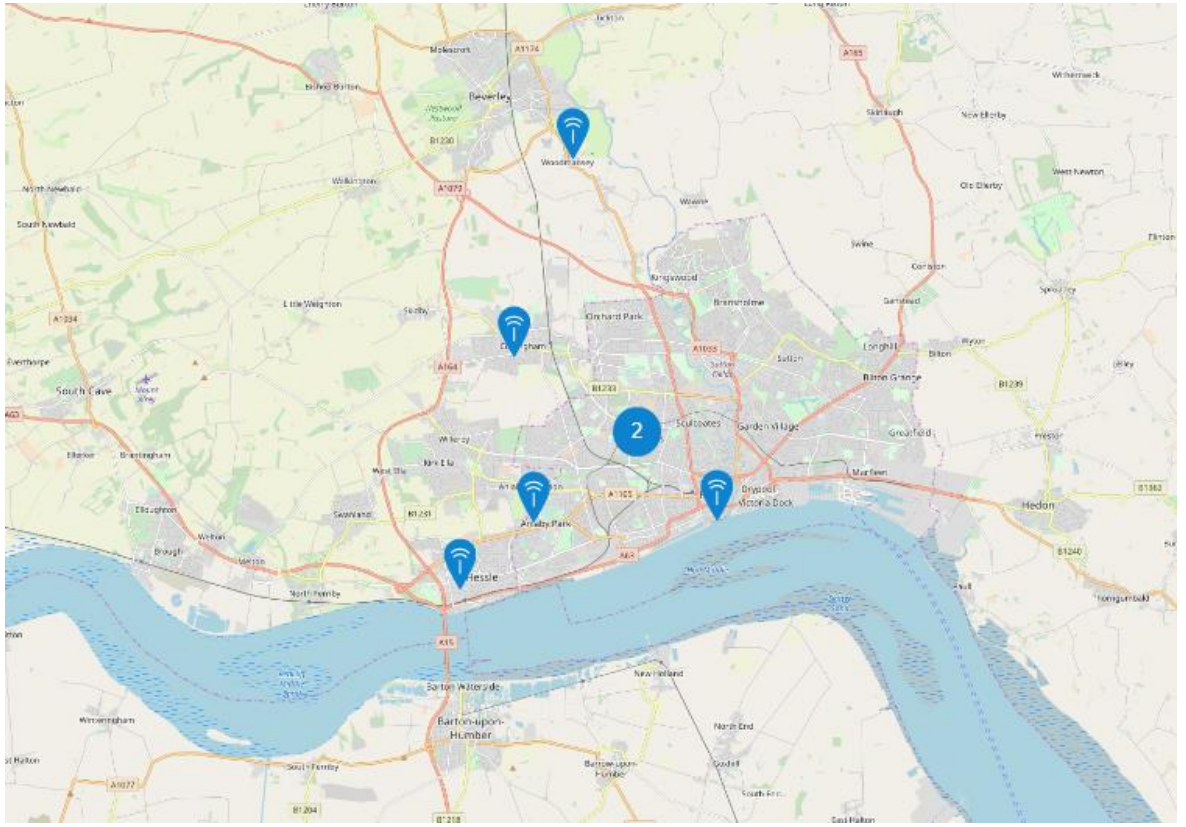
- The Things Network underpins a worldwide network of open LoRa gateways

The Things Network

- Building networked communities using LoRa
  - Provides the server backend for LoRaWAN applications
  - Creates open source software and hardware which you can use to build your own bespoke LoRa network
  - Sells LoRa devices on Kickstarter
- You can buy your own gateway and register it on The Things Network
  - Any LoRa endpoint can then use your gateway as a conduit onto The Things Network
  - The Things Network will host your LoRa applications and pass your endpoint data into your own backend servers

LoRa gateways in Hull

- There are a number of gateways in Hull which are attached to The Things Network
- We are trying to get more of them installed



What is a server?

- The LoRa server receives messages from the gateways, identifies ones that are for applications it knows about, sorts out multiple messages and then forwards them on to the application backend
- You can create your own servers, but for testing you can use those provided by The Things Network (TTN) for free
- You can register your gateways on The Things Network and then create your applications and connect your servers to them
  - A great way to get started, but for “proper” services you would want to have your own infrastructure

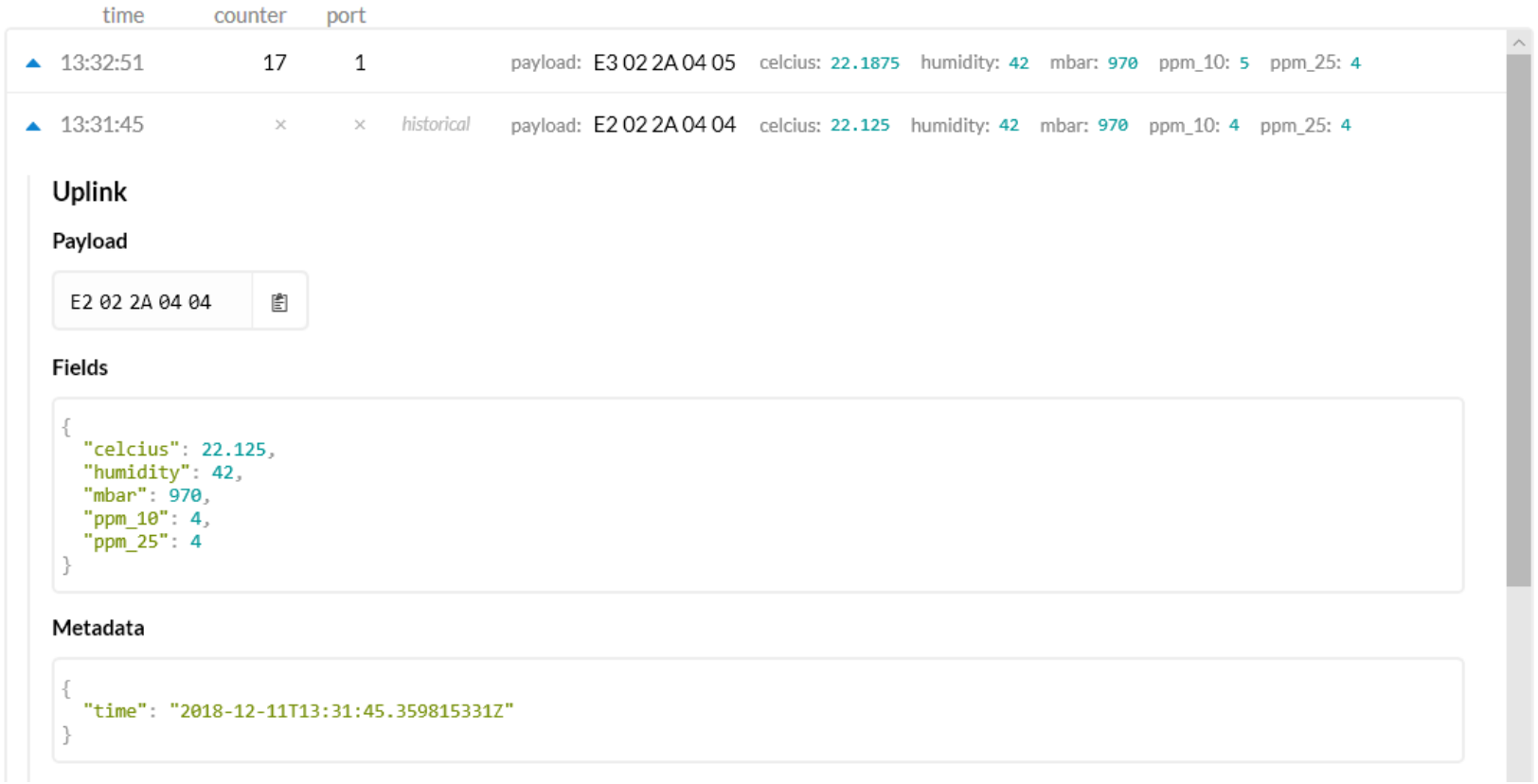
LoRa Security

- Because LoRa is a broadcast medium using public frequency bands anyone can eavesdrop on any message
- An endpoint is associated with a particular *application* which is identified in each LoRa packet that the endpoint sends
- Each application has an encryption key
- Keys can be “baked in” to a device or deployed via the LoRa network
- In addition, a given network session is encrypted by means of a network session key
  - Based on AES-128 (802.15.4 security)

Endpoint activation

- No such thing as “default password” for a LoRa device
- An endpoint must be *activated* before it can be used on a LoRa network
- Two forms of activation:
- Activation By Personalisation (ABP):
  - Credentials are “burned in” to the endpoint before it is deployed
- Over The Air Activation (OTAA):
  - Endpoint is deployed containing an *Application Root Key* which is used to authenticate a setup process that produces credentials to be stored in the endpoint

Application data



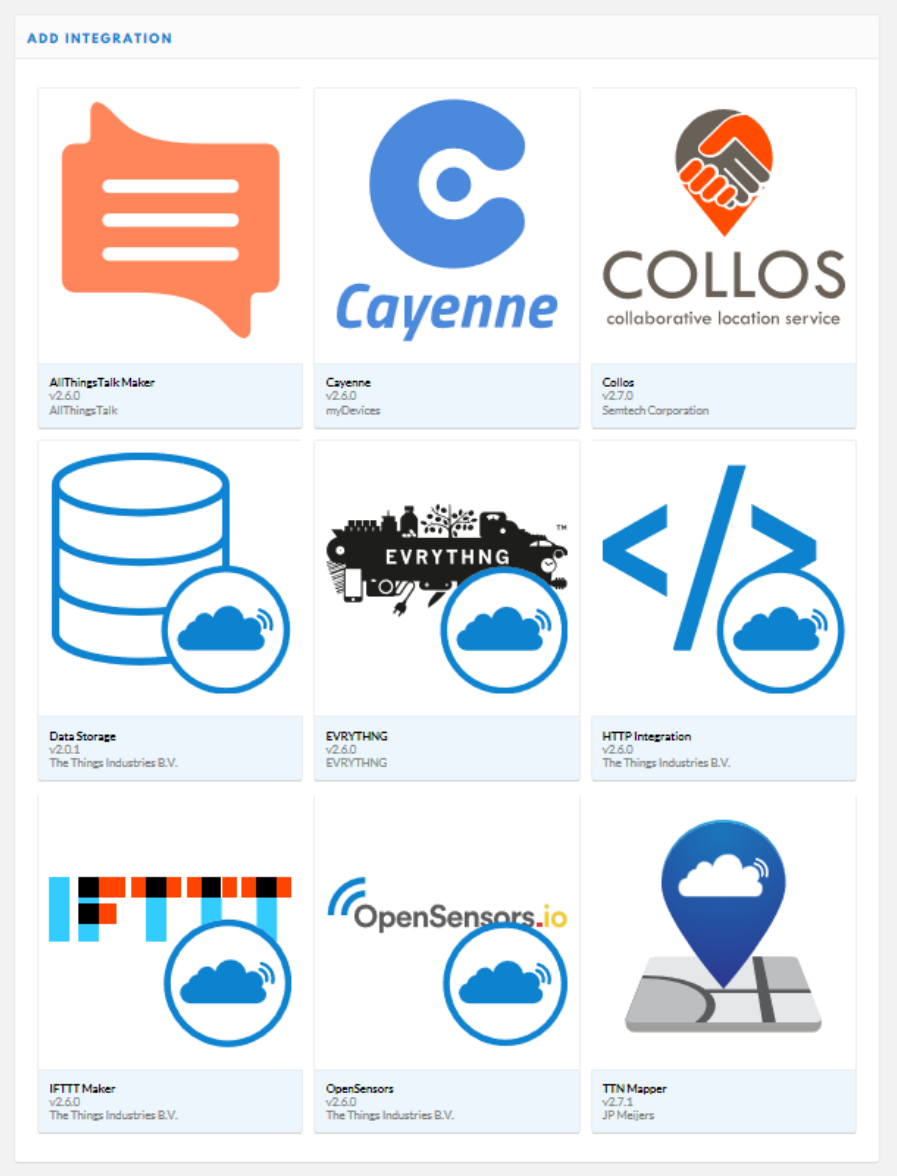
- These are packets received from an endpoint
- The data values are encoded at the node and decoded on receipt

Application metadata

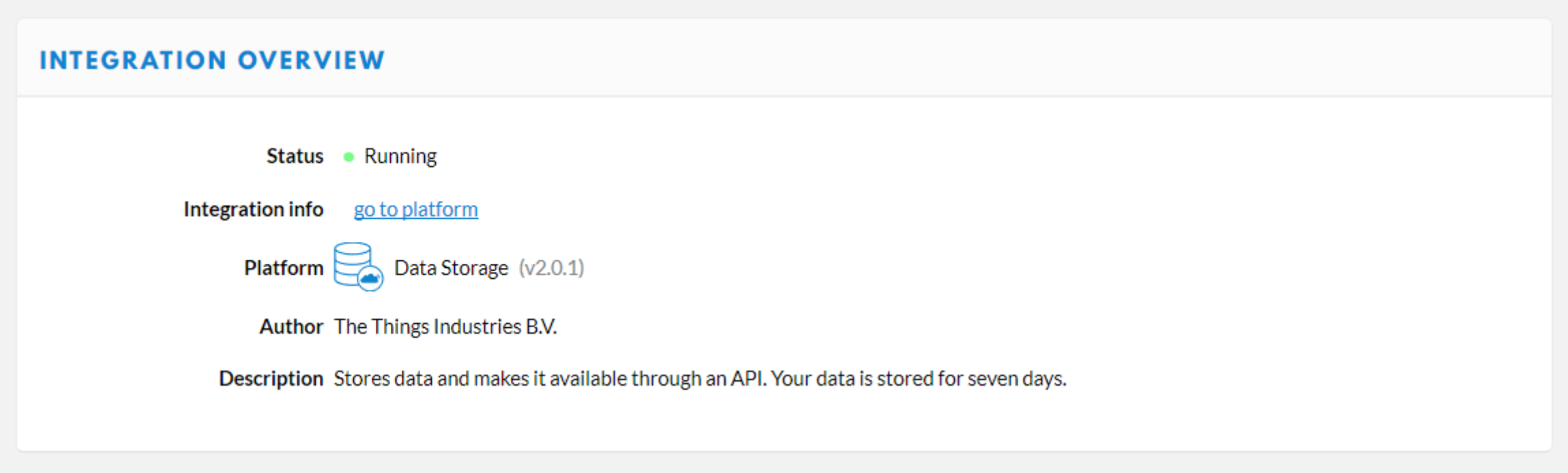
- This is the metadata that gets also gets pushed up to the application
- It contains details of the gateways that received the packet

Integrations

- The Things Network provides a set of “integrations” that you use to send LoRa messages into your application
- You can use http GET/POST, or MQTT or IFTT
- They also provide a database for short term storage (7 days)

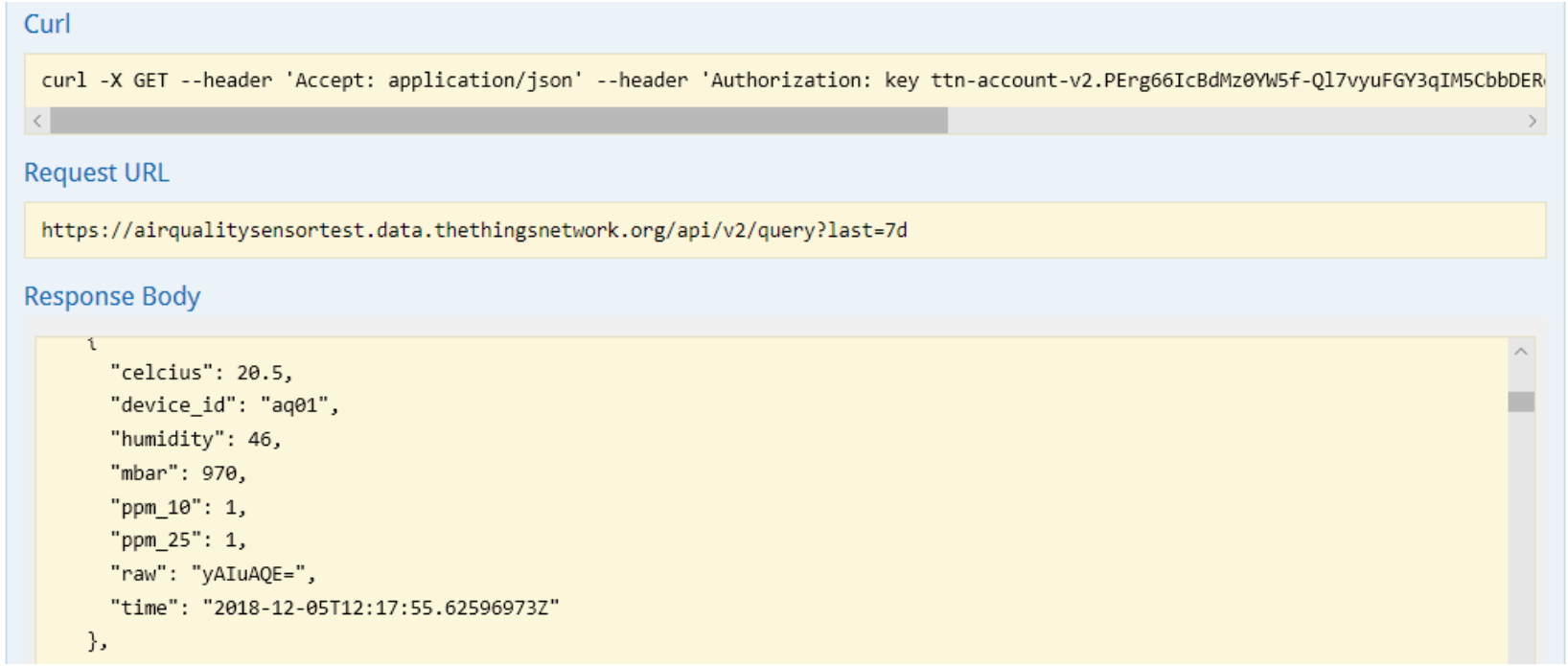


Data Storage



- This integration will store your data for 7 days
- There is a restful interface for getting readings back

Reading back data



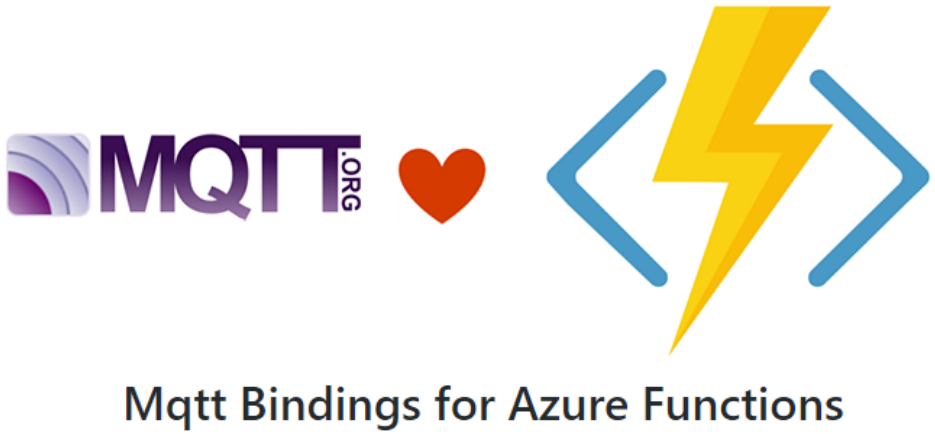
- It is easy to pull data back from the Things Network
- The API is defined by Swagger

Sending messages to a LoRa endpoint

- A LoRa endpoint will not normally be listening for messages from the gateway
  - This is to save power
- Class A
  - Listen for a brief interval after the endpoint has sent something
- Class B
  - Listen for a brief interval at scheduled times
- Class C
  - Nearly continuous listening (not suitable for battery powered endpoints)

LoRa to Azure IoT

- The Azure IoT Hub works with MQTT
  - Remote devices can publish and subscribe to topics it exposes

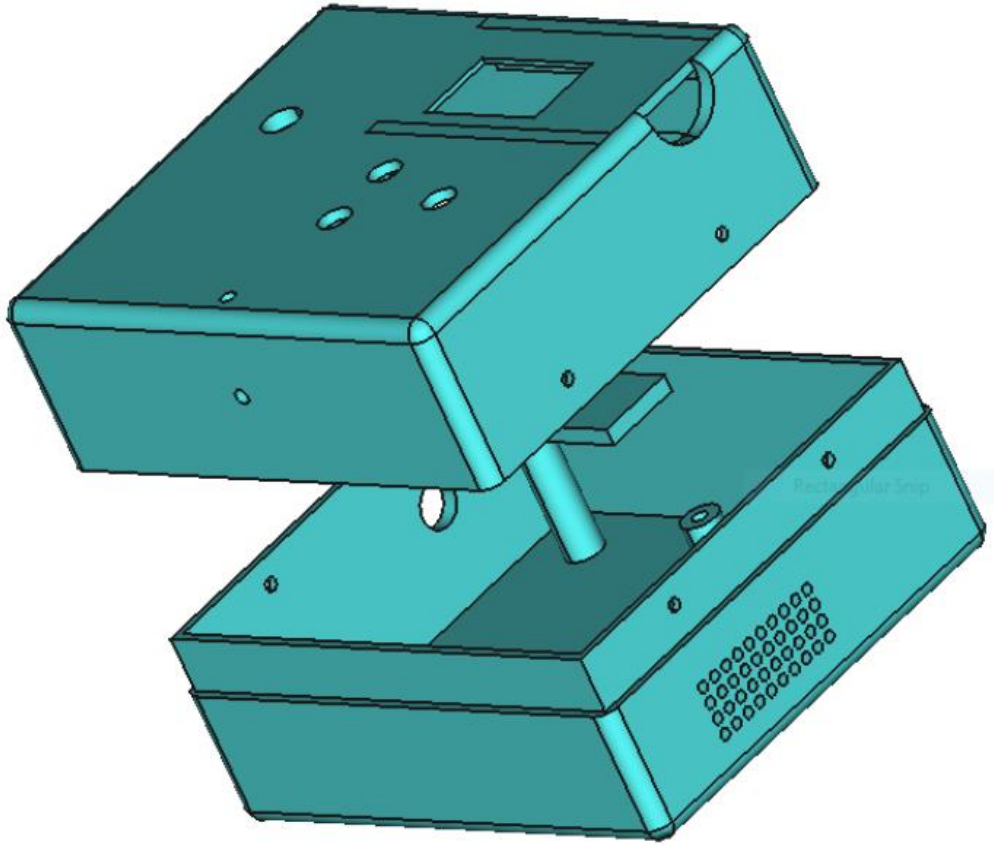


- However, you need something extra to connect Azure IoT Hub to an MQTT broker
  - You need this because The Things Network acts as an MQTT broker to expose LoRa messages to clients
- There are some MQTT bindings for Azure functions I'm playing with
- You could also use a timed Azure Function to download batches of data

All on GitHub

- All my sensor designs are on GitHub
  - This includes code and 3D printable files for the cases
- These devices would serve as the basis of any remote controlled sensor or actuator

  `github.com/CrazyRobMiles/AirQuality`



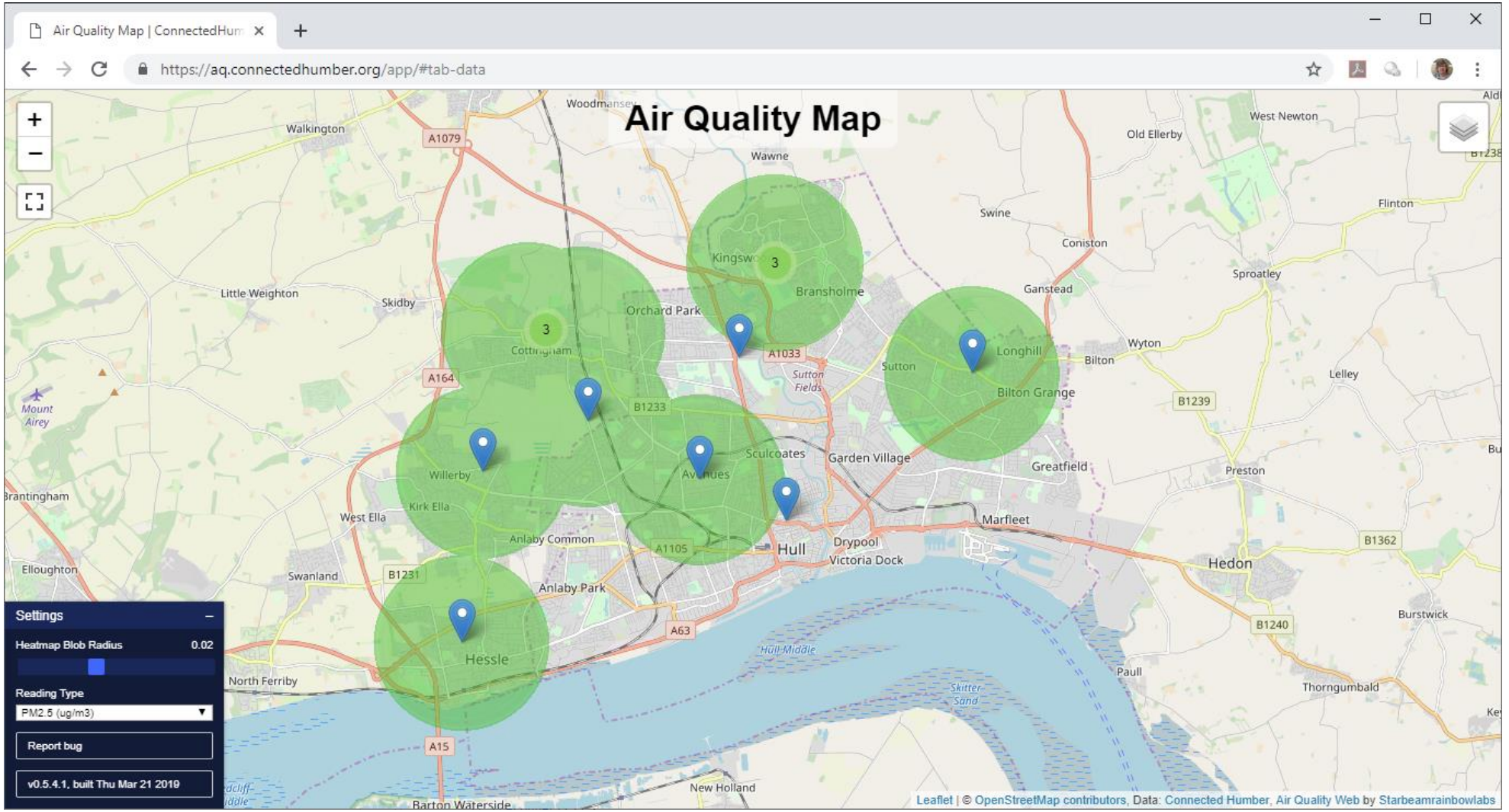
Connected Humber

- Connected Humber is a community group that is building sensors, deploying them and then analysing the results
- We meet up at c4di in Hull on the first and third Thu. of each month

`www.connectedhumber.org`

`gettogether.community/connected-humber`

aq.connectedhumber.org



Summary

- Building connected devices is easy and cheap
- The Azure IoT provides industrial strength support for MQTT connected devices
- You can use Azure Functions to bind to events generated by devices
- You can use Azure Table Storage to hold incoming data
- LoRa: low-power long-range networking moving small data packets
- LoRa gateways and apps can be attached to The Things Network
- You can use MQTT to Azure to receive LoRa data

www.robmiles.com connectedhumber.org