i-UG Open Source Education
for IBM i

Node-Red

LESSON 9

1. What is MQTT?

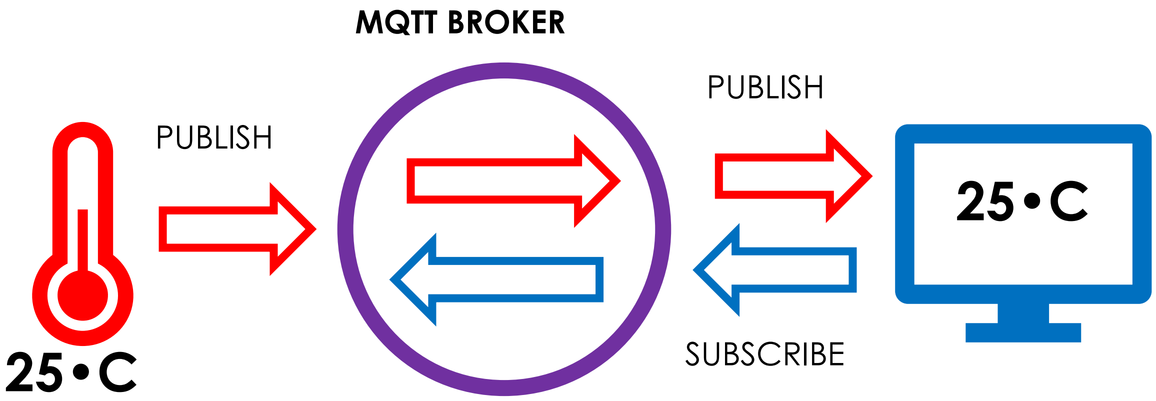
MQTT (Messaging Queuing Telemetry Transport) is a standard messaging protocol for the IoT (Internet of Things), it's extremely lightweight publish/subscribe messaging for machine-to-machine communication and makes up the basis of how we can manage traffic from remote locations and send instruction to remote devices at very low-to-no cost.

Using an MQTT broker, we can have a device/sensor publish information to the broker, then have the broker send this to a subscribing device, and this can be with multiple devices.

2. What is an MQTT Broker?

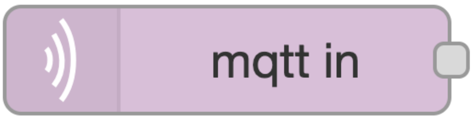
MQTT brokers are centralised servers that facilitate the message exchange process in the MQTT protocol's publish/subscribe communication model. As intermediaries, the MQTT broker manages connections between clients, receiving published messages and distributing them to appropriate subscribers based on their topic subscriptions.

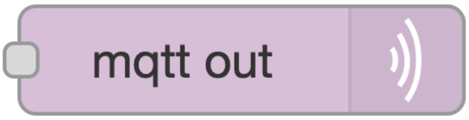
Brokers maintain a list of connected clients, their subscriptions, and Quality of Service (QoS) levels. They handle message delivery based on the specified QoS level, ensuring reliable communication even in challenging network conditions.



3. MQTT and Node-RED

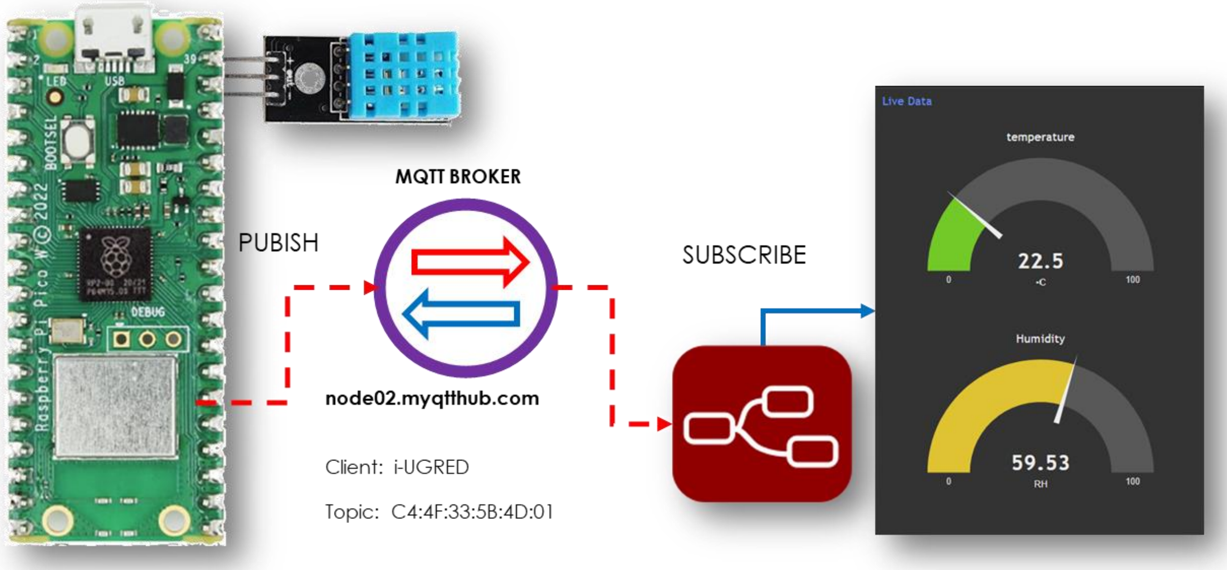
We will be using two new nodes for this. The MQTT IN Node and the MQTT OUT Node.





4. Lesson Overview:

In this lesson, we are going to use a temperature and humidity sensor to receive live data that we can display on a node-red dashboard and create a warning system to advise if the temperature rises above a certain level.



Within our Node-RED, we will be a Subscriber. In short, our MQTT IN Node will connect to an MQTT broker (We are using a broker on the IBM Power system –ActiveMQ). In the Broker, we have set up a specific topic (In our case for the Temperature sensor it is 'C4:4F:33:5B:4D:01') and so that Topic will be the conduit for us to read the temperature that is being picked up by the Temperature Sensor and published to the MQTT Broker.

There is an 'MQTT OUT equivalent' on the Temperature Sensor which is set to send its Temperature information to the specific topic on our MQTT Broker 'C4:4F:33:5B:4D:01'.

The Broker's job is to make this available to any subscriber to that topic, so for us, we will use the MQTT IN Node as above, and receive the readings that the Broker holds.

Just for completeness, and to demonstrate the use of being a Publisher, we will also set up to Publish some data that can be received and read by Node-RED.

N.B. Each MQTT Broker has its own way of registering, setting up and managing, so in this course, setting up an MQTT Broker will not be covered.

We have an MQTT Broker that we will use and topics set up for this education and we will simply attach to those points and use the stock credentials to validate and gain access.

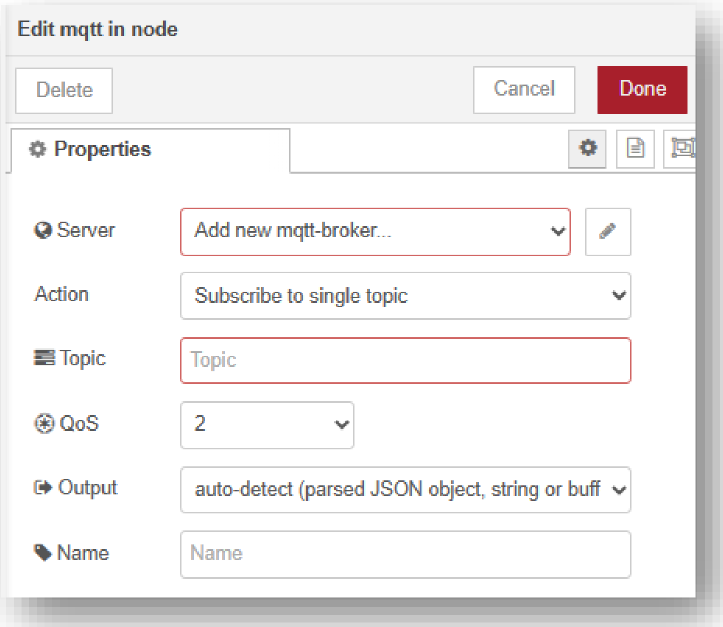
STAGE 1

To start, lets copy the Flow from the earlier Lesson 5 – Engaging with the UI – Output.

So, Drag in an *MQTT in* Node and open it and review the information. It should be displaying the MQTT information as below, but if not, then we'll reset it here.

First, we need to point it to our MQTT Broker and set the topic that we will use.

Open the MQTT in Node: -



Edit mqtt in node

Server: Add new mqtt-broker...

Action: Subscribe to single topic

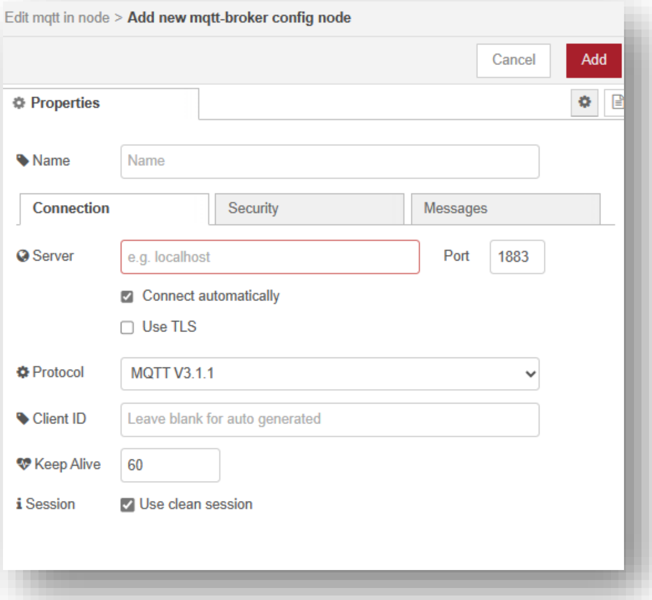
Topic: Topic

QoS: 2

Output: auto-detect (parsed JSON object, string or buff

Name: Name

Click the pencil to edit the Server: -



Edit mqtt in node > Add new mqtt-broker config node

Name: Name

Server: e.g. localhost

Port: 1883

☑ Connect automatically

□ Use TLS

Protocol: MQTT V3.1.1

Client ID: Leave blank for auto generated

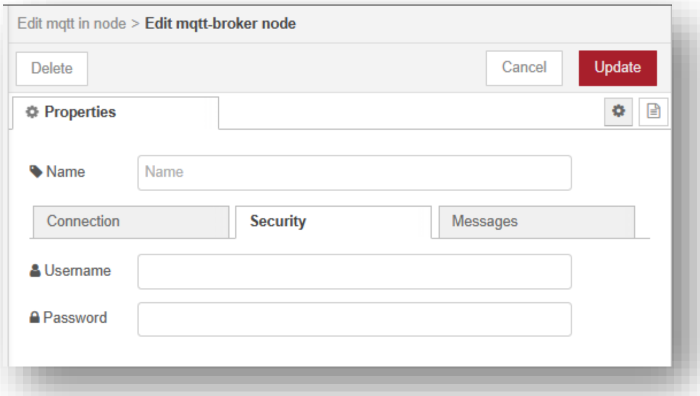
Keep Alive: 60

Session: ☑ Use clean session

Set the Server to: ersc.ddns.net Set the Port to : 1883

Leave the remaining boxes as they are.

Then on the Security Tab: -



Edit mqtt in node > **Edit mqtt-broker node**

Name: Name

Username:

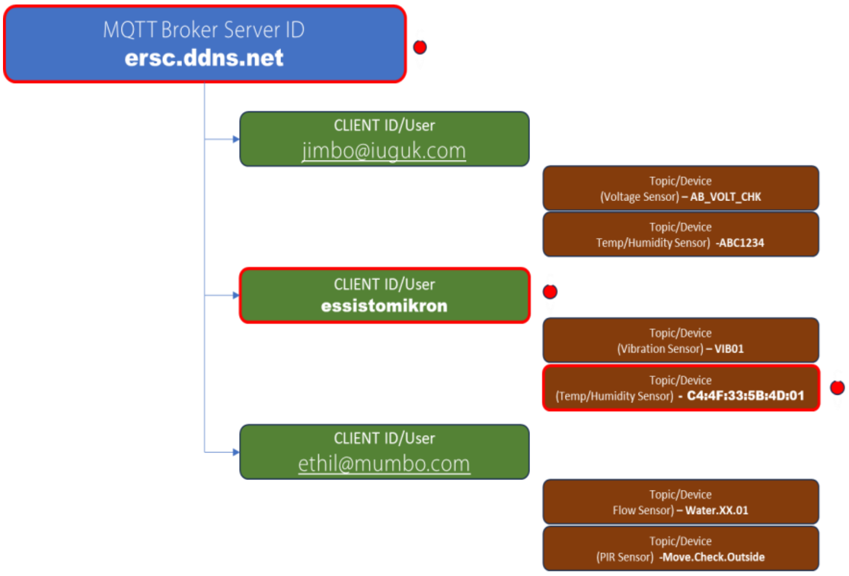
Password:

Set the User name to: essistomikron, Set the Password to: xCYFgVEx-8wfhFXfn

Click Update/Add. As you return to the Edit MQTT IN Node panel, enter the topic: C4:4F:33:5B:4D:01

Click Done.

What we are doing here is simply giving the *MQTT in* Node the address of our chosen MQTT Broker Service (*ersc.ddns.net*), the Credentials we will use to let the MQTT Broker who we are (*User: essistomikron, Password: xCYFgVEx-8wfhFXfn*), then which of the many Topics in our stack that we want information from as it arrives (*C4:4F:33:5B:4D:01*).



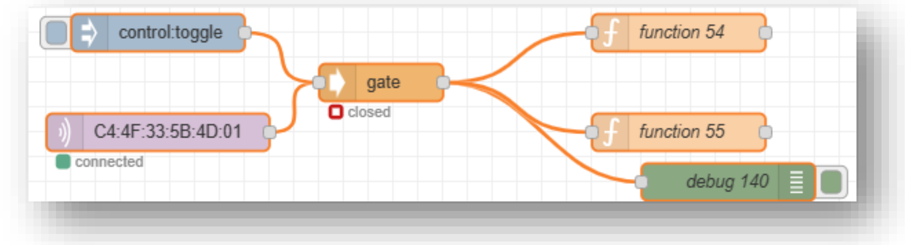
Deploy the flow to see if the node successfully connects to the node, there should be a green square with 'connected' written next to it directly under the node.

Once you have a 'connected' indicator on the MQTT in Node, we will delete the Inject Node that we have imported and drag in a new Inject Node. These nodes can't be connected so don't try to just yet. Inside the inject node you must set the msg.payload to *toggle* (a String) and the msg.topic to *control (Again, a String)*. You will appreciate the use of this Inject Node as we progress.

Next bring in a gate node and connect both the inject node <u>and</u> the MQTT in Node to it. This will give us a controllable platform where we can allow or deny access from the MQTT sensor information to our gauge displays.

Now, the purpose of the two Function Nodes we have copied will essentially not change, however, they will be 'fed' differently. They will not need changing as the code required is the same.

Attach a Debug Node to the right hand side of the Gate Node. It should look a bit like this: -



N.B. If you are looking at the GitHub Code, this is essentially Gauges Stage 2 linked on the Stage 1

Click on the Inject Node to open the Gate Node. The Debug information in the right hand sidebar, should be populating. Once you have a few of these, hit the Inject Node again to toggle the gate to Closed.

Let's have a look at the information in the Debug . This is what the sensor is sending! Each entry in the Debug list will look like this: -

```
21/07/2023, 12:35:14   node: debug 169

C4:4F:33:5B:4D:01 : msg.payload : Object

  ▶ { data: object, info: object }
```

This is a JSON Object called msg.payload. Inside this object are two more objects,. They are called **data** and **info**. We will explain these below but let's just take a look at JSON objects.

What is JSON?

JSON stands for JavaScript Object Notation. The JSON format is text only!
JSON is a lightweight data-interchange format
JSON is plain text written in JavaScript object notation
JSON is used to send data between computers
JSON is language independent

The data in the JSON is in name/value pairs

Data is separated by commas
Curly braces hold objects
Square brackets hold arrays

In JSON, values must be one of the following data types:

a string
a number
an object (JSON object)
an array
a Boolean
null

You will use these often in Node-RED as they are a great vehicle for moving data between nodes and are very understandable. Take some time to research JSON objects as they will be valuable for you in the future. For now, we will just cover enough about JSON objects for you to work with them in the lessons.

So, let's just have a look at our JSON Object.

The JSON in our Debug list looks like this: -

```
21/07/2023, 12:35:14   node: debug 169

C4:4F:33:5B:4D:01 : msg.payload : Object

  ▶ { data: object, info: object }
```

Let's expand this. Click the Arrow. You will then see this: -

```
21/07/2023, 12:35:14   node: debug 169

C4:4F:33:5B:4D:01 : msg.payload : Object
▼   Object
▶   data: object
▶    info: object
```

Clicking on each of the arrows, you eventually see this:-

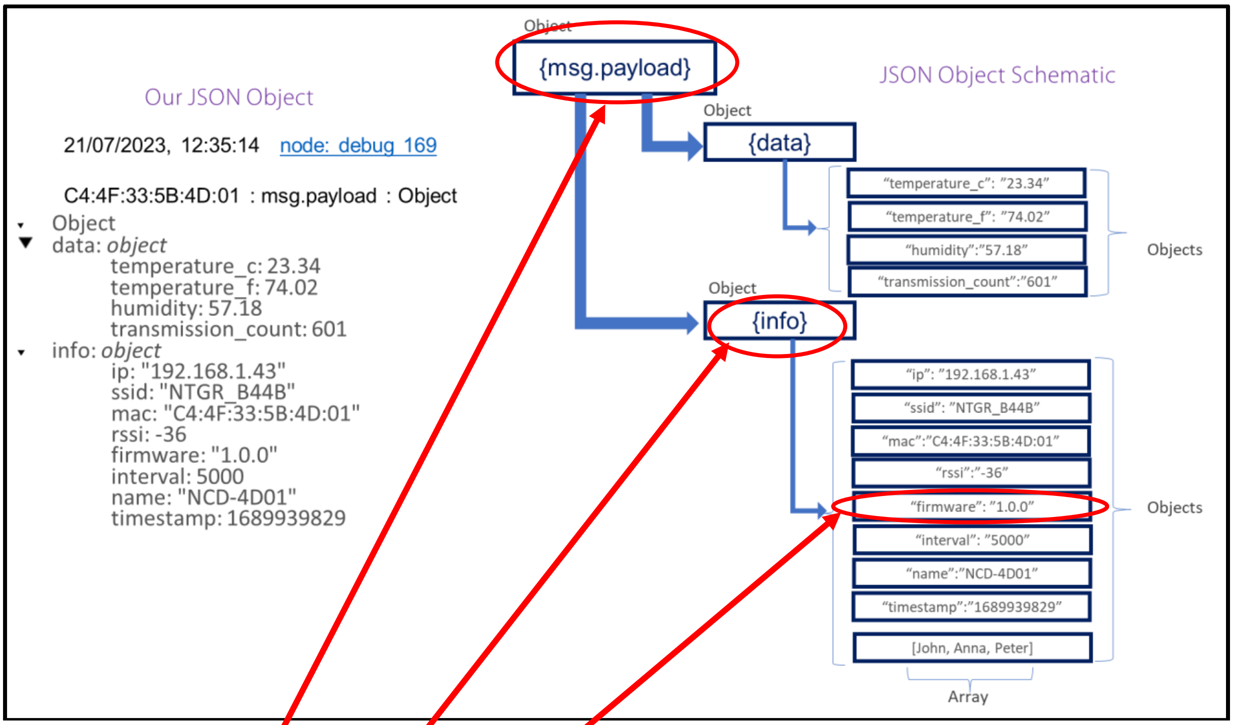
```
21/07/2023, 12:35:14   node: debug 169

C4:4F:33:5B:4D:01 : msg.payload : Object
▼   Object
▼   data: object
        temperature_c: 23.34
        temperature_f: 74.02
        humidity: 57.18
        transmission_count: 601
```

```
▼  info: object
        ip: "192.168.1.43"
        ssid: "NTGR_B44B"
        mac: "C4:4F:33:5B:4D:01"
        rssi: -36
        firmware: "1.0.0"
        interval: 5000
        name: "NCD-4D01"
        timestamp: 1689939829
```

As you can see, we don't just receive two simple numbers, one for temperature and one for humidity. All of this information (under payload) counts as the payload, but we just want temperature_c and humidity, hence the function nodes.

The way we read them in Node-RED is simple. If we wanted to get any of these values, we would start at the top and gradually move deeper into the object.

Here is how they are constructed: -



Our JSON Object

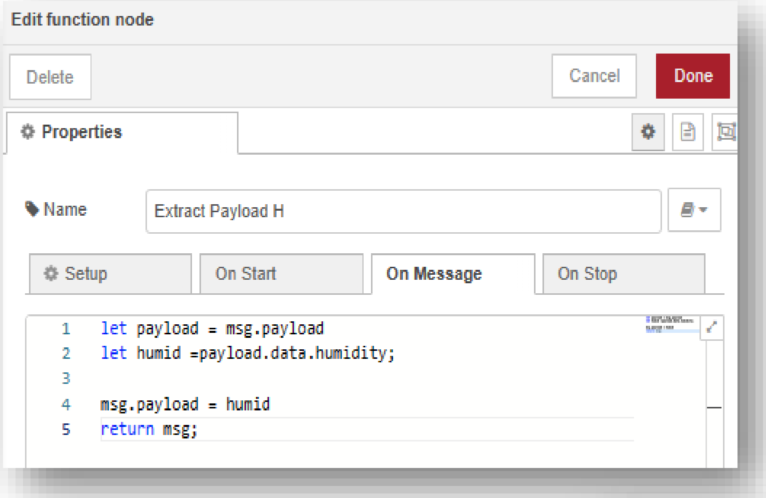
```
21/07/2023, 12:35:14   node: debug 169

C4:4F:33:5B:4D:01 : msg.payload : Object
▾  Object
▼  data: object
        temperature_c: 23.34
        temperature_f: 74.02
        humidity: 57.18
        transmission_count: 601
▾  info: object
        ip: "192.168.1.43"
        ssid: "NTGR_B44B"
        mac: "C4:4F:33:5B:4D:01"
        rssi: -36
        firmware: "1.0.0"
        interval: 5000
        name: "NCD-4D01"
        timestamp: 1689939829
```

So, if we wanted to get the firmware level, in the function node we would refer to the msg and access as: -

msg.payload.info.firmware we will get **'1.0.0'**

For our purposes, we will need temperature_c and humidity, so let's construct this in a logical way: -

In each of the function nodes we need to set three lines of code to define which part we are looking for: -



Name: Extract Payload H

```
let payload = msg.payload
let humid =payload.data.humidity;

msg.payload = humid
return msg;
```

Let payload = msg.payload

(defines the term payload as information in the payload part of the Node-RED msg)

Let humid = payload.data.humidity

(defines a variable - humid as the information under payload > data > humidity)

msg.payload = humid

(defines the payload going forward as the information defined by humid)

Let payload = msg.payload

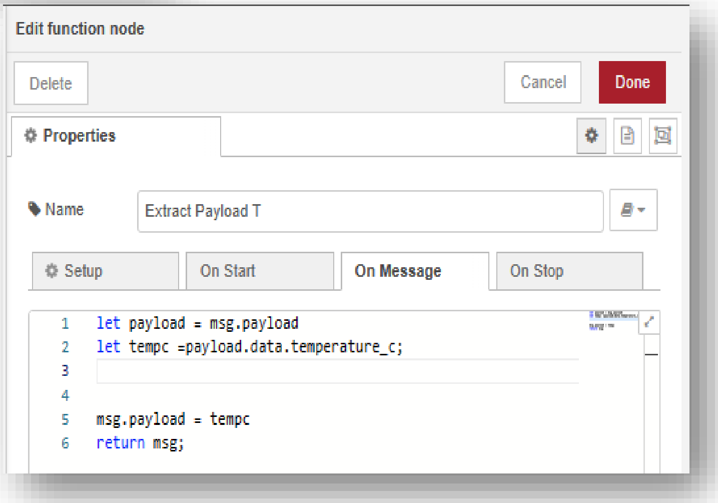
(defines the term payload as information in the payload part of the Node-RED msg)

Let tempc = Payload.data.temperature_c

(defines a vatiable - tempc as the information under payload > data > temperature_c)

msg.payload = tempc

(defines the payload going forward as the information defined by tempc)



Name: Extract Payload T

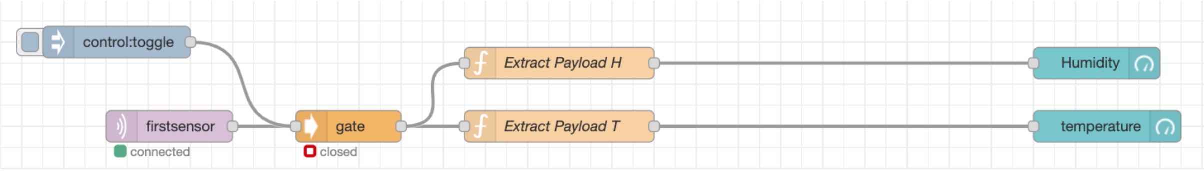
```
let payload = msg.payload
let tempc =payload.data.temperature_c;


msg.payload = tempc
return msg;
```

The result of doing this gives us the exact information we want in a form that can be directly inserted into some gauges, giving us a live readout in a visual format. We will be passing the data we retrieve (temperature or humidity) on to the next node – a Gauge Node – in the msg.payload. Remember, as you set up each Function node, it will be helpful to name each node to indicate the data set they are retrieving.

5. STAGE 2

So now, let's add some gauges, one from each function node. Let's start by just reviewing the two new Gauge Nodes. Again, it will be helpful to name each gauge node with which data set they are going to display.

Apply the dashboard settings you want. It is important that you set up a new Tab as otherwise, these Gauges will appear as duplicated on your earlier Gauges Tab. Confusing.

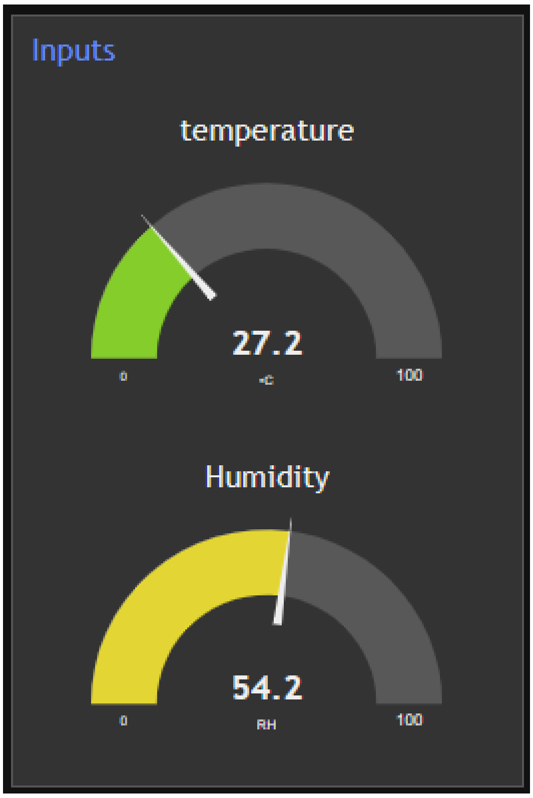


So, now Deploy your Flow.

You will see that the Gate Node is still displaying a red Closed indication.

Look at the UI for the Tab you have selected. You will see the two Gauges but they will be empty.

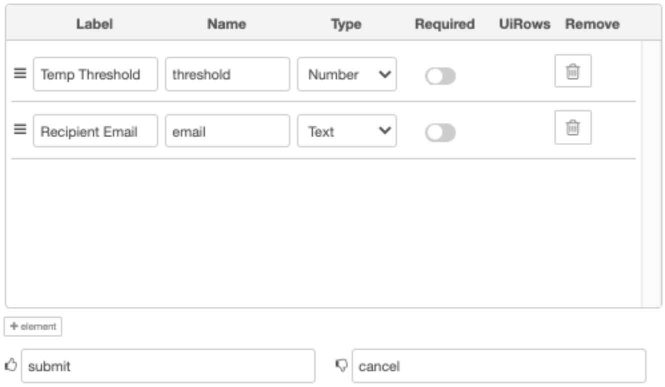
Now, return to the flow and click the Inject (Toggle) Node. It will set the Gate to a green Open indicator.



Now look at the UI and you will see the temperature and humidity in the Manchester DC.

a. Automating and advising

Next we want to set up a warning system that will warn us, if the sensor goes above a certain level, via email. For this we will need both the address to send the warning to and the level at which we want to send the message. We can manage this, and be able to change it whenever we want, by putting in a Forms Node. We have learned about Forms Nodes and e-mail Nodes in earlier lessons.



Label	Name	Type	Required	UiRows	Remove
Temp Threshold	threshold	Number	○		🗑
Recipient Email	email	Text	○		🗑

Start with a form node, this will let us write the email we want the warning to be sent to and choose the temperature threshold, within the dash. Create two form elements within the node, one for the temperature threshold set to a number type, and one for the recipient email set to a text type.

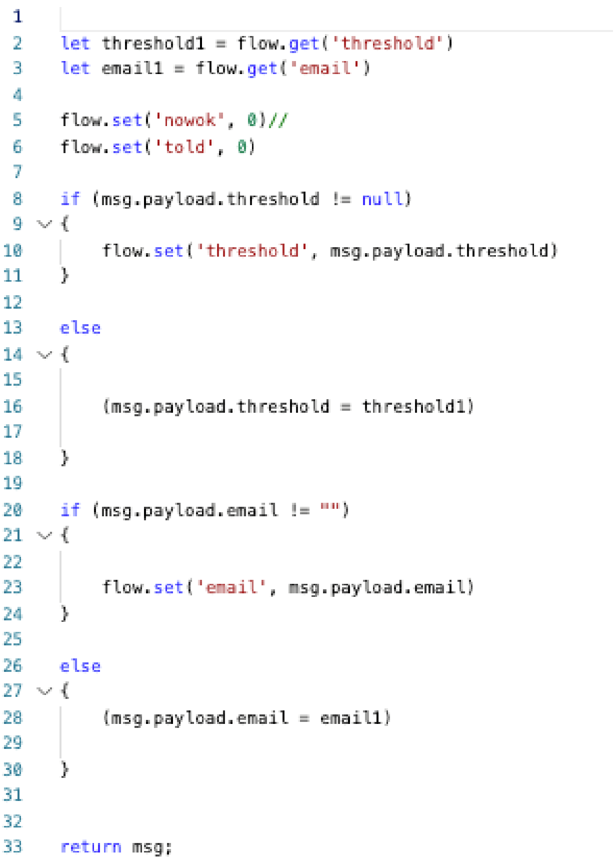
Now bring in a function node, the purpose of which is to take the information from the form entries and make it useable. This code is what you need to put in. It works as follows…

First of all, we are going to introduce a 'flow' variable. This means that we can set up a variable, just as we have before, but instead of this variable being available just within the Function Node we are coding at that time, a flow variable will be available to any part of the whole Flow.

We are then going to set two of these flow variable to use as switches later.

The other two will maintain both the temperature threshold we want to measure to and the e-mail of the person that should receive an Alert if the threshold is breached.

Note that we are NOT going to overwrite the flow variable either the email or the threshold if the user does not enter one!



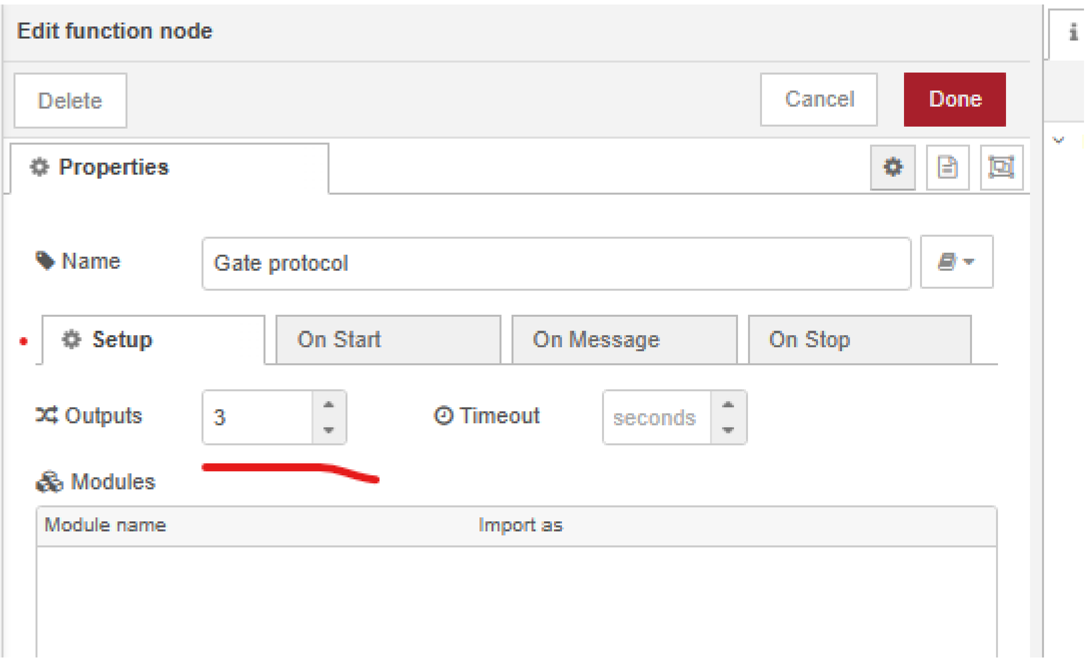
```

let threshold1 = flow.get('threshold')
let email1 = flow.get('email')

flow.set('nowok', 0)//
flow.set('told', 0)

if (msg.payload.threshold != null)
{
    flow.set('threshold', msg.payload.threshold)
}

else
{

    (msg.payload.threshold = threshold1)

}

if (msg.payload.email != "")
{

    flow.set('email', msg.payload.email)
}

else
{
    (msg.payload.email = email1)

}


return msg;
```

Now drag in another Function Node. We will call this the Gate Protocol Function Node.

Click the 'Setup' Tab.

We are going to give this Function Node 3 potential outputs. The output channel we will use will depend on the data it is receiving and the results of our processing. This is a very useful feature and one which you will use often in future.

So, set the Outputs box to 3: -



Edit function node

Name: Gate protocol

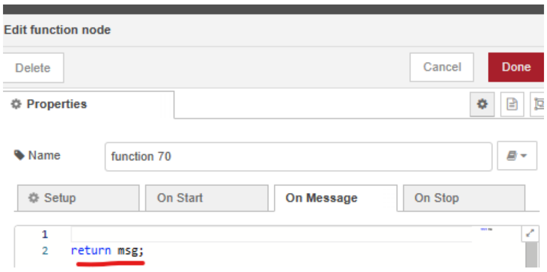
Outputs: 3

Timeout: seconds

Modules

Module name	Import as

Now click on the 'On Message' tab. This is where we will be putting the Code. However, let's just have a brief overview of what we are going to do in order to select which channel we are going to use.



Edit function node

Name: function 70

```

return msg;
```

One small feature that you have been using (because you HAVE to) is the closing of code in a Function Node – return msg;

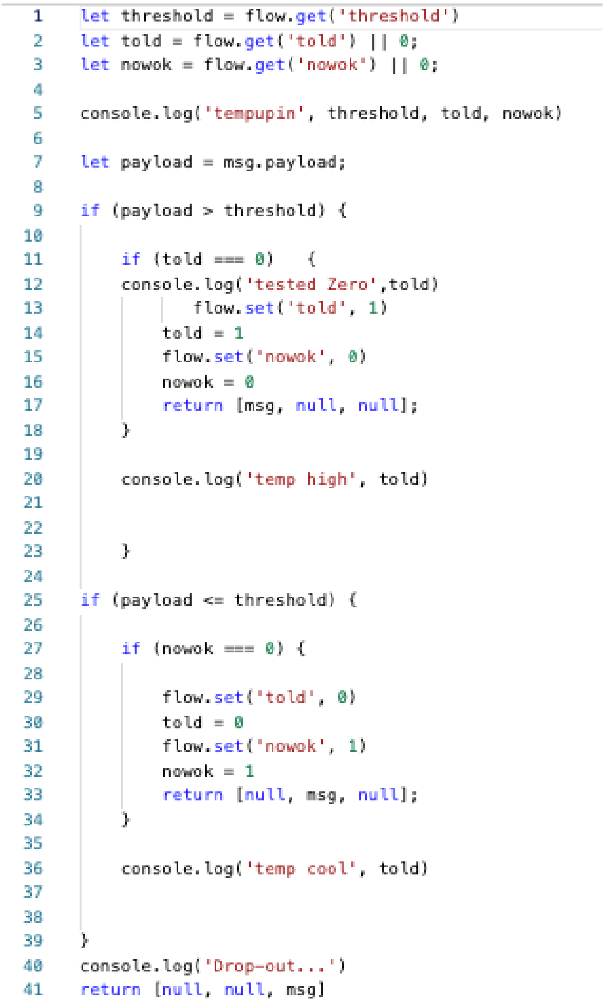
This closes the program and makes sure that the msg is passed on to the next Node.

We are now going to alter this return statement To only return ONE of the three output ports. The way we do this is like this: -

```
return [null, null, msg]
```

This will open port 3 and allow the message (msg) to flow through. It will close the other two ports, so no msg will flow through port 1 and 2.

So, it stands to reason that `return [null, msg, null]` will open port 2 and allow the message (msg) to flow through. It will close the other two ports, so no msg will flow through port 1 and 3. etc.



```
let threshold = flow.get('threshold')
let told = flow.get('told') || 0;
let nowok = flow.get('nowok') || 0;

console.log('tempupin', threshold, told, nowok)

let payload = msg.payload;

if (payload > threshold) {

    if (told === 0)    {
    console.log('tested Zero',told)
            flow.set('told', 1)
        told = 1
        flow.set('nowok', 0)
        nowok = 0
        return [msg, null, null];
    }

    console.log('temp high', told)


    }

if (payload <= threshold) {

    if (nowok === 0) {

        flow.set('told', 0)
        told = 0
        flow.set('nowok', 1)
        nowok = 1
        return [null, msg, null];
    }

    console.log('temp cool', told)


}
console.log('Drop-out...')
return [null, null, msg]
```

The code is set to measure the data from the sensor against the temperature threshold given by the form.

If the temperature is above the threshold, we want to send out a warning message to the recipient that we have in our e-mail flow variable, advising of the breach.

However, we don't want to send an e-mail every 5 seconds whilst the temperature remains above the threshold, so we are using an interlock method to make sure that we only sent it once.

If the temperature then falls below the threshold, we want to advise the recipient that this is the case, and again, we only want to send this once.

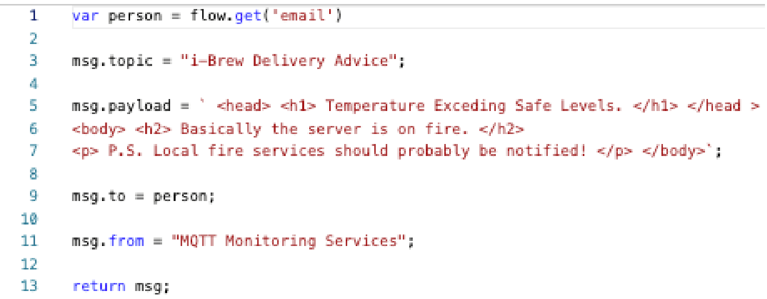
If it never has gone over the threshold, then we don't want to advise anyone.

Now, we are going to rely on another Function Node updating the switches so that we know where we are, hence the need for these to be flow variables. Note the return msg usage WITHIN the IF groups!!

Take a careful look at the code (you can import it from GitHub) to see how we are doing this. If you can think of another way, go ahead and try it.

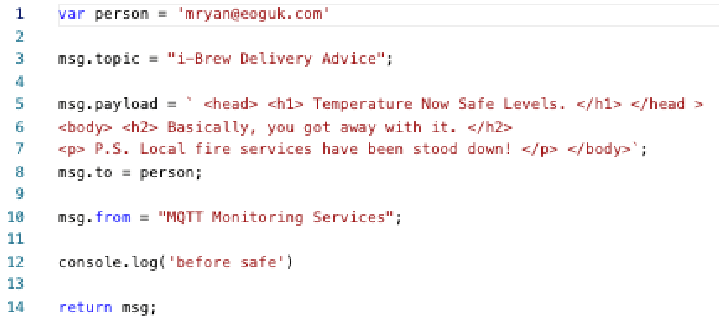
Next, we need to set up a mechanism to send a warning. Whilst we send the message, we also want to record that we have just done this, so that we don't send it again. The next message will (hopefully) be an advisory message saying that the temperature is now within tolerance.

Send Warning



```
var person = flow.get('email')

msg.topic = "i-Brew Delivery Advice";

msg.payload = ` <head> <h1> Temperature Exceding Safe Levels. </h1> </head >
<body> <h2> Basically the server is on fire. </h2>
<p> P.S. Local fire services should probably be notified! </p> </body>`;

msg.to = person;

msg.from = "MQTT Monitoring Services";

return msg;
```

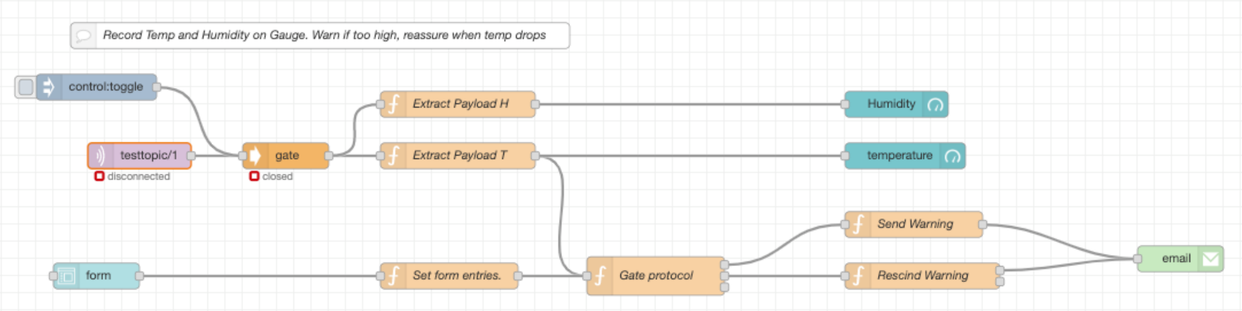
Rescind Warning



```
var person = 'mryan@eoguk.com'

msg.topic = "i-Brew Delivery Advice";

msg.payload = ` <head> <h1> Temperature Now Safe Levels. </h1> </head >
<body> <h2> Basically, you got away with it. </h2>
<p> P.S. Local fire services have been stood down! </p> </body>`;
msg.to = person;

msg.from = "MQTT Monitoring Services";

console.log('before safe')

return msg;
```

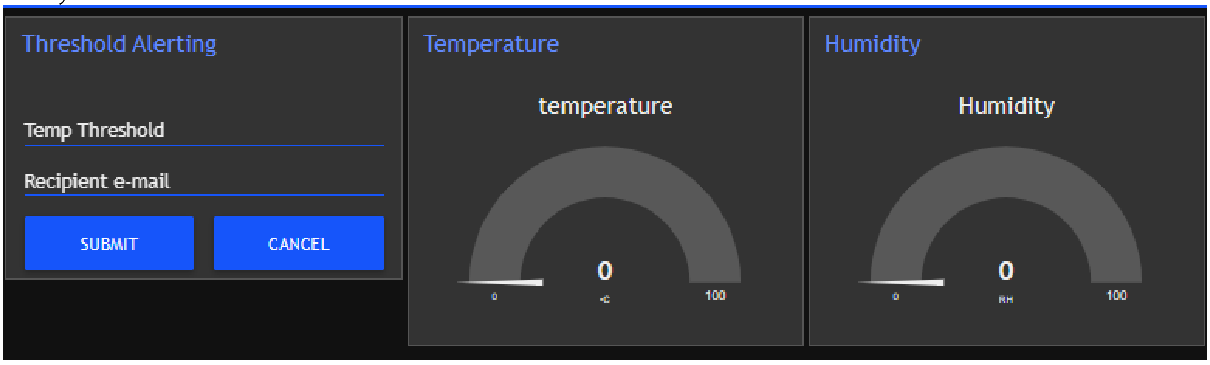
So, drag in two more Function Nodes and put the Send Warning code below into Send Warning Function Node, and put the Rescind Warning code into the Rescind Warning Function Node.

As you can see from these Nodes, we are building up the message that we want to send. Now to finish the flow we just need to add an email node. Your completed flow should be looking something like this: -



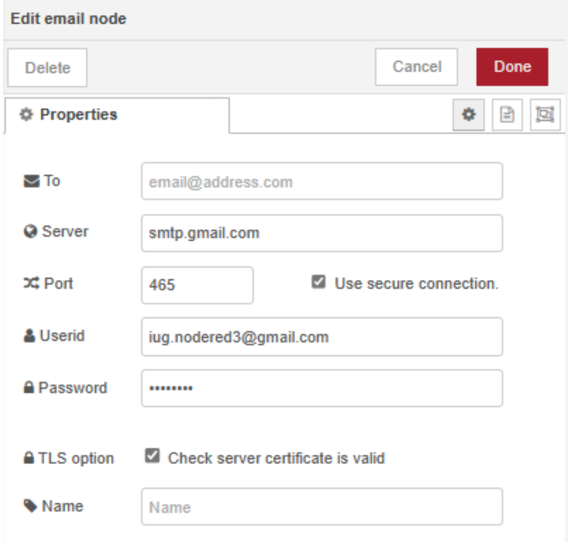
N.B. If you are referencing the GitHub code, this will be the Gauges Stage 1, Stage 2and the Warning group.

And your dashboard should resemble this.



b. The e-mail Node

Drag in an email (Out) Node. Strangely, the IN and OUT nodes are not notated, so we want the one with the connector on the LEFT, so we can put stuff into it. Open the email Node: -



Edit email node

Delete | Cancel | Done

Properties

To	email@address.com
Server	smtp.gmail.com
Port	465 ☑ Use secure connection.
Userid	iug.nodered3@gmail.com
Password	••••••••
TLS option	☑ Check server certificate is valid
Name	Name

Setting up a mail server is a whole subject on its own, so we will adopt a previously set up service using Google so that we can send emails from our Node-RED flow.

Enter the details as follows into the node:

Server:	snmp.gmail.com
Port;	465
Userid	iug.nodered3@gmail.com
Password	This is in GitHub

You can give it a name, but the default email, is good. This is the vehicle to send out the email, which you just constructed in the two Function Nodes. You can play around with the content in the Function Nodes to personalise your email.

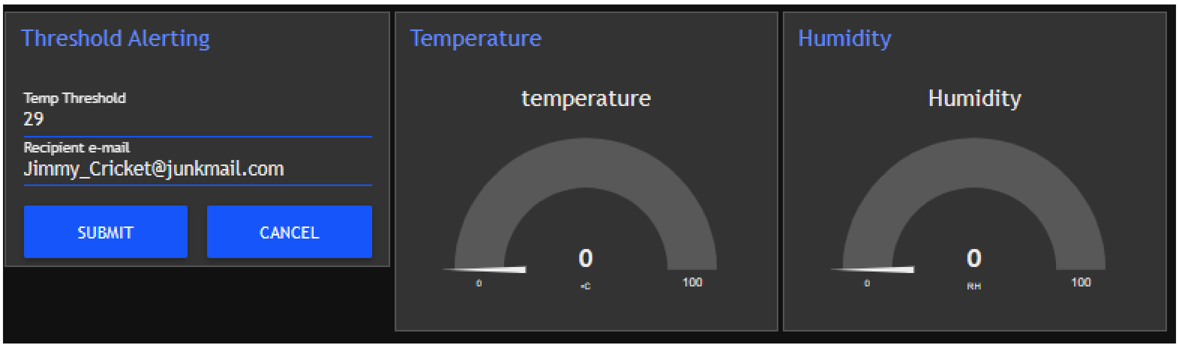
Check that you have followed the code carefully and the all of the Nodes are connected correctly. Then, Deploy.

c. Operating the warning system

So now, nothing is happening…

That's because we have a couple of jobs to do.

First thing is to set a temperature threshold and to give the warning system a recipient to send emails to. So, go to the UI on the Sensors tab: -



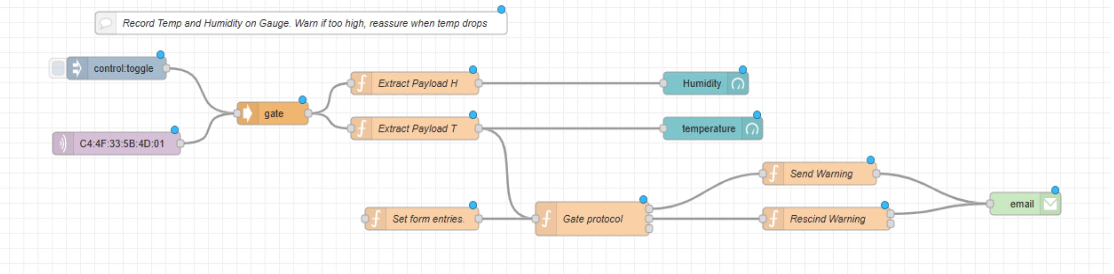
Threshold Alerting

Temp Threshold
29

Recipient e-mail
Jimmy_Cricket@junkmail.com

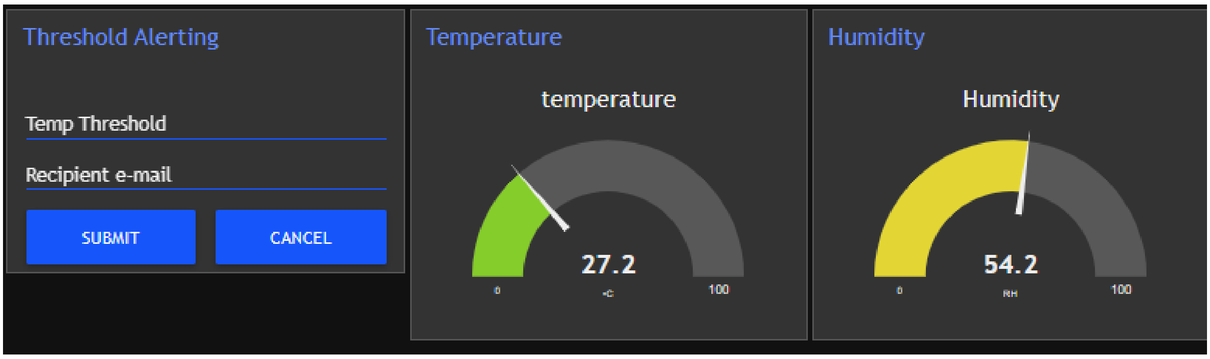
Key in a threshold number. Keep it quite high to begin with, say, 29˚**c (29**). Also, key in your email address or an email address that you have access to, to check.

Then, return to the Flow: -



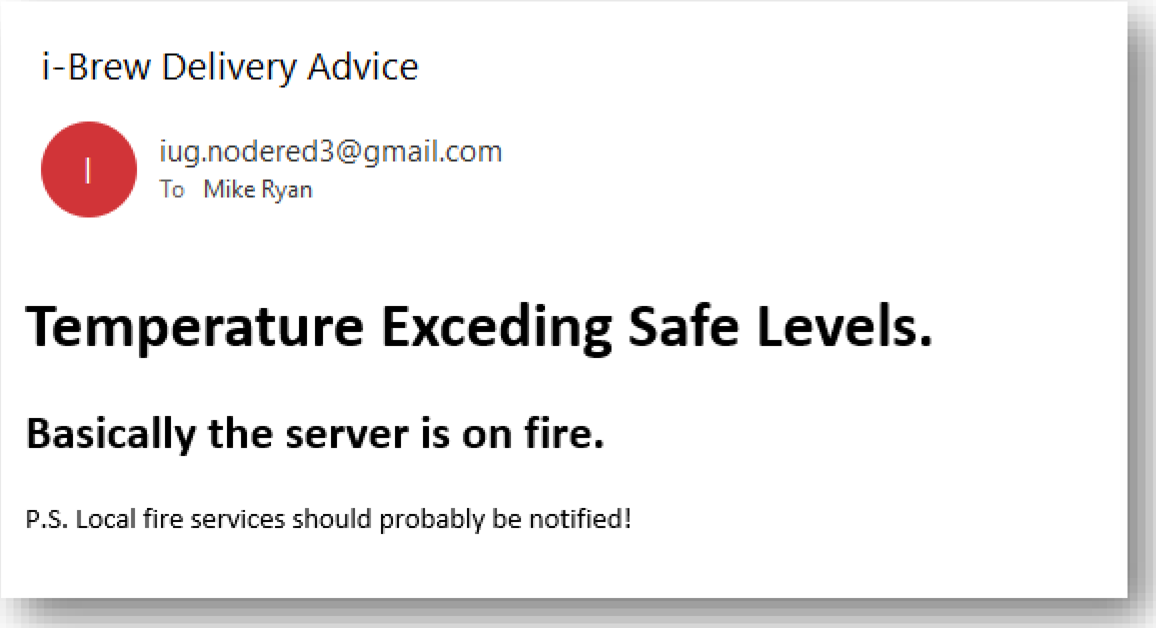
You will notice that the Gate Node (Not the Gate Protocol Function Node) is showing a Red Closed indication. Click the Toggle Inject Node. This will open the Gate Node and let the traffic from the MQTT Broker through.

Now go to the UI. It should be showing realistic readings in the Gauges: -



So, all is OK and we have not sent anyone any e-mails. Good.

A simple way to check out alerting system now is to change the threshold level. So, go to the UI and put in a Temp Threshold of 21. So now, the temperature in Manchester will be higher than the threshold, so it should send an email…



i-Brew Delivery Advice

I

iug.nodered3@gmail.com
To Mike Ryan

Temperature Exceding Safe Levels.

Basically the server is on fire.

P.S. Local fire services should probably be notified!

Play around with the threshold levels and see that the correct email comes out… and only once per incident!

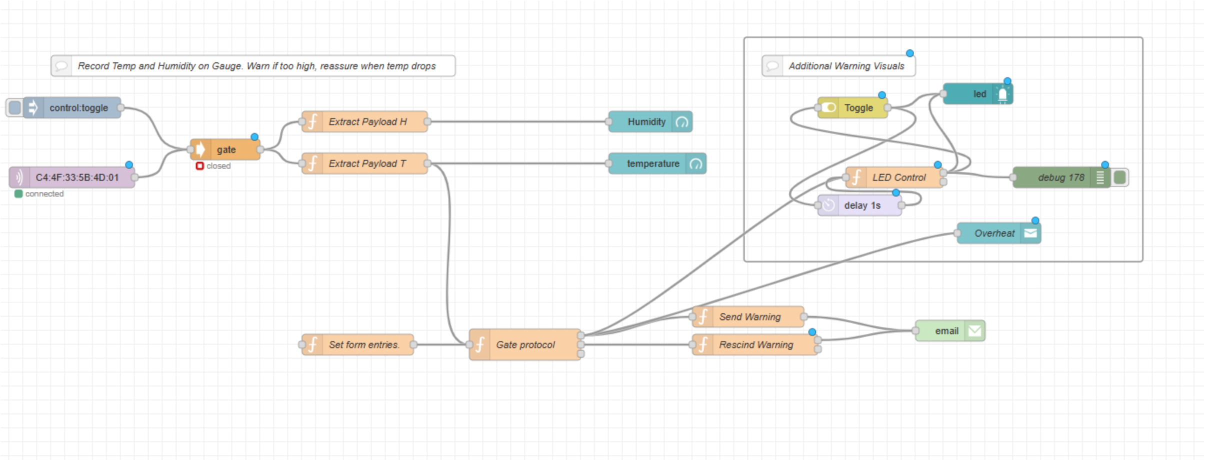
d. Confirming a point

Just to confirm a point, set up an MQTT OUT Node. Make up a unique Topic. Set up an MQTT IN Node and use the SAME unique topic. Add an Inject Node to the MQTT OUT Node, add a Debug Node to the MQTT IN Node. In the Inject Node, change the msg.payload to a String and key in 'Hello World'

Although it is all within one Node, it is actually sending the Hello World message out to the MQTT Broker in Manchester and you are reading in – on that topic – from the MQTT Broker in the MQTT IN Node and this shows in the Debug Node.

e. Upping the Game.

Now, If you want to extend knowledge a little and show some better warnings, bring in the GitHub version and study the 'Additional Warning Visuals' group.



This puts additional information out on the Dashboard to warn users of what is happening.

This is down to you now. You've learned a lot to get here, you can now try and impress by experimenting with this kind of alerting and, basically, show off.

It is all yours. You are unlikely to break it. I think you are now ready for the brewery…

And with that, the lesson is concluded. If you didn't manage to make one of the flows work then you can copy one of the flows from our GitHub repository here, and see where you went wrong. These functions, although they seem simple, are the backbone of many useful applications of Node-Red for major and small businesses.

https://github.com/JoshRyanEOG/i-UG_Education_Node-Red