

Edison, Grove, and Node-RED

Hands-On | Lab 2 of 2

Author: Stefania Kaczmarczyk | slkaczma@us.ibm.com | [@stefania_kaz](https://twitter.com/stefania_kaz)



Part One: Connecting the Edison to the Internet of Things Foundation

This lab assumes that you have already followed the Quickstart set-up for the Edison at intel.com/edison/getstarted, assembled the Groove header/temperature sensor, and successfully tested the sensor using grovetemp.js. This lab continues to use the Intel XDK IDE.

1. To start, we need to get the IP address for the Edison by going to <http://edison.local>. We replace edison with the unique name given to the device during setup, if that option was chosen.

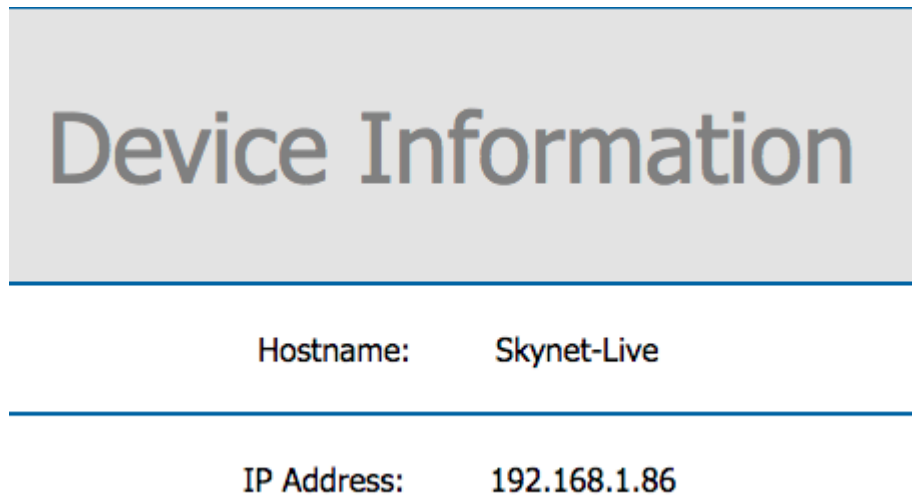


Figure 1-1: Device IP address

2. The IoT Foundation has created several quickstart recipes for getting a device connected to the foundation and pulling some test data. For this lab we'll use the Galileo recipe with a few modifications. To start, we skip to Step2 of the Connect section and continue through to step 3-f (Run the sample) in Quickstart Recipe for the Galileo:

<https://developer.ibm.com/iotfoundation/recipes/intel-galileo/>

d) Enter the command: `chmod +x setup.sh`

e) Run the command: `./setup.sh`

f) Run the sample by entering: `node ibm-iot-quickstart.js`

Figure 1-2: Stopping step for quickstart recipe

3. When we reach **Connect Step 3 Part f**, we need to modify the CPU thermal file the javascript is reading in order to pull information from a file that has some content. To do this we'll open the file in the default UNIX visual editor VI:

```
vi ibm-iot-quickstart.js
```

4. With the javascript file open, press **Shift+I** to enter Insert mode.

- Now we just need to update the the fs.readFile line to read **thermal_zone1** instead of thermal_zone0:

```
fs.readFile('/sys/class/thermal/thermal_zone1/temp','utf8')
```

- To exit the editor, we hit the **Esc** key and type a **colon (:)**. Then enter **wq** to write out the modified file and quit.

TIP: For more information on VI commands: <http://www.cs.rit.edu/~cslab/vi.html>

- Now we run the sample using the command in step f and continue to the Visualize portion of the recipe. If successful we should see the message “Device connected” and a visualization of our data at the bottom of the page.

A green dot followed by the text "Device connected at 10:56:51 AM".

Figure 1-3: Device connected status on IoT Foundation quickstart web page

- Now that we know our device can transmit to the IoT Foundation we need to register the device and get temperature data from the sensor rather than the CPU. Switch to Bluemix (<https://console.ng.bluemix.net/>) and login.
- To start we will simply add the **Internet of Things service** by selecting **CATALOG** from the navigation menu and scrolling to the Internet of Things section. We need the Internet of Thing service so we select it's hexagon.

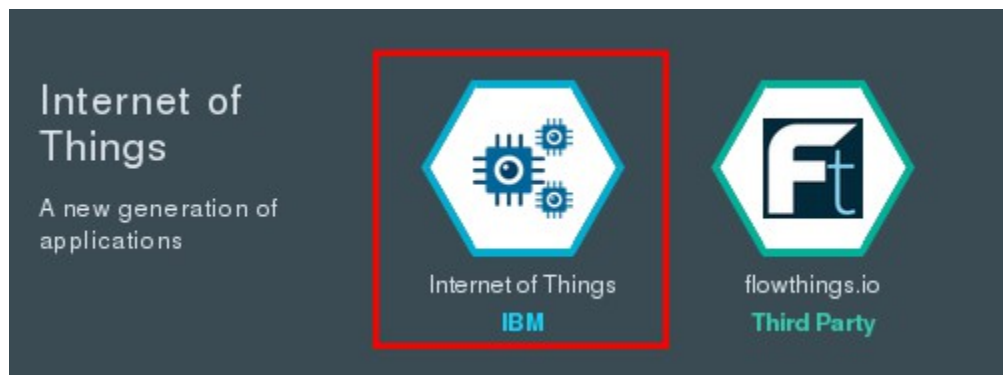


Figure 1-4: Bluemix Internet of Things service section

10. This new window gives an overview of the service to include the pricing scheme and a link to VIEW DOCS for more documentation on the service. For now we just need to give the service a **unique name** and select **CREATE**.



The screenshot shows a form with two fields. The first field is labeled 'App:' and has a dropdown menu with 'Leave unbound' selected. The second field is labeled 'Service name:' and has a text input containing 'sk-iot'. The 'Service name' field is highlighted with a red border.

Figure 1-5: Create service form

11. The service takes us to a welcome screen. To get to our foundation dashboard we simply select **Launch dashboard** under “Connect your devices”.



Figure 1-6: Launch dashboard button

12. The INFO tab lists steps for adding devices, so we start by selecting the **DEVICES** tab and then the plus sign to **Add Device**.

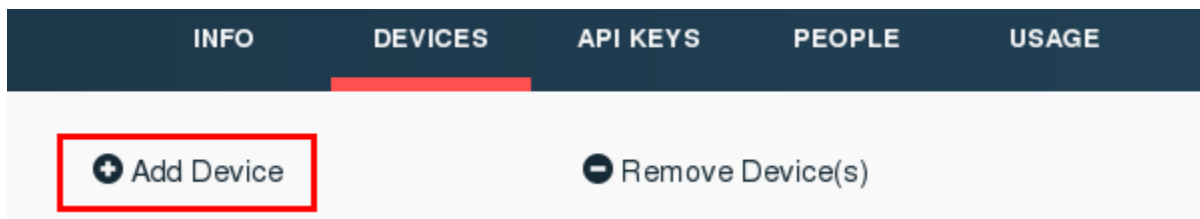
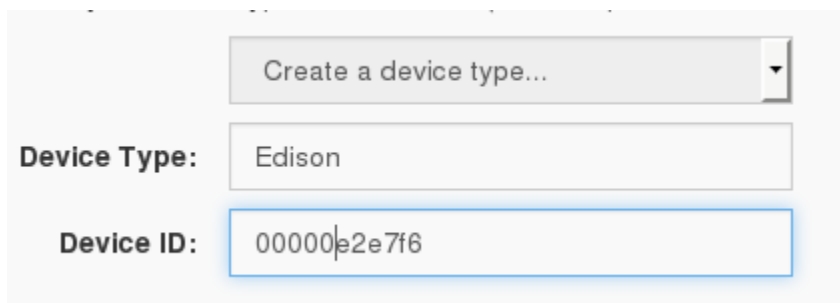


Figure 1-7: The DEVICES tab in IoT Foundation

13. Device Type can be any user generated name, but we'll use Edison for ease. Then we need to enter the MAC address we used in the Visualization step.



The screenshot shows a form with three fields. The first field is a dropdown menu labeled 'Create a device type...' with 'Edison' selected. The second field is labeled 'Device Type:' and has a text input containing 'Edison'. The third field is labeled 'Device ID:' and has a text input containing '00000p2e7f6'. The 'Device ID' field is highlighted with a blue border.

Figure 1-8: Adding a new Edison device

14. Select **Continue**. We need to make sure to copy the text in Step 1 into a text document and save it as it is non retrievable once this view is closed.

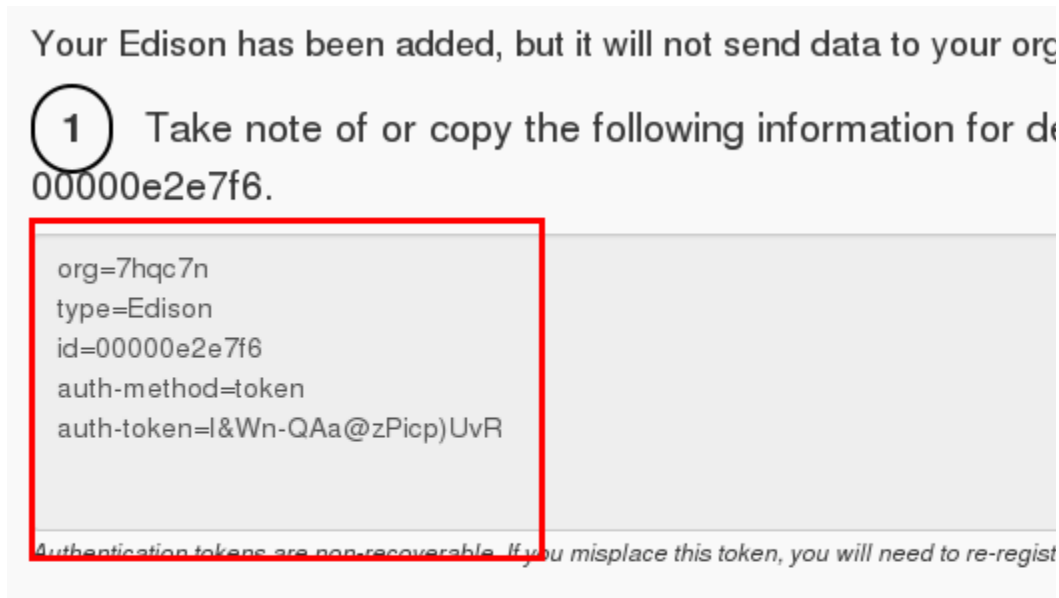


Figure 1-9: Connection information for device

15. Once we select done we are taken back to our DEVICES view. The Edison appears under Device Type with a status of Unavailable.

<input type="checkbox"/>	Device Type	Device ID	Last Event
<input type="checkbox"/>	◆ Edison	00000e2e7f6	Unavailable

Figure 1-10: The status of the newly added Edison

16. Now we need to get some data up to the IoT Foundation. We navigate back to the Intel XDK IoT Edition application and create a new project using a Blank Template. Copy the contents of this **main.js** file from GitHub into the main.js file.

<https://github.com/slkaczma/edisongroovetempbluemix/blob/master/main.js>

This file contains a few lines from grovetemp.js that we already worked with as well as some error and connection handling from the ibm-iot-quickstart.js file. Note there are some lines that are commented out, those will be used later.

17. We are now going to combine details from this temperature reader with a connection object for the IoT Foundation. First we open **package.json** to add the required modules to dependencies.

```
"dependencies": {  
  "mqtt": "^0.3.8",  
  "getmac": "^1.0.6",  
  "properties": "^1.2.1"  
}
```

18. Now, we right-click on the file structure and select **New File**. We name the file **device.cfg** and paste the contents of Step 14 above into this file.

19. We save the files and then build, upload, and run the project. If successful we should see the contents of the message object displayed in the console. Additionally, to confirm the messages are actually making it to the foundation, we can go back to our **IoT Foundation dashboard** and select the **DEVICES** tab. The Edison should now have a green dot next to it with a Last Event of 'Just now'.




<input type="checkbox"/>	Device Type	Device ID	Last Event
<input type="checkbox"/>	 Edison		 Just now

Figure 1-11: Edison with active data

20. Selecting the plus icon next to the **Last Event** status opens up a listing of the last 10 inbound events. If we click one of the events we can see the contents of the message, our Celsius and Fahrenheit data.

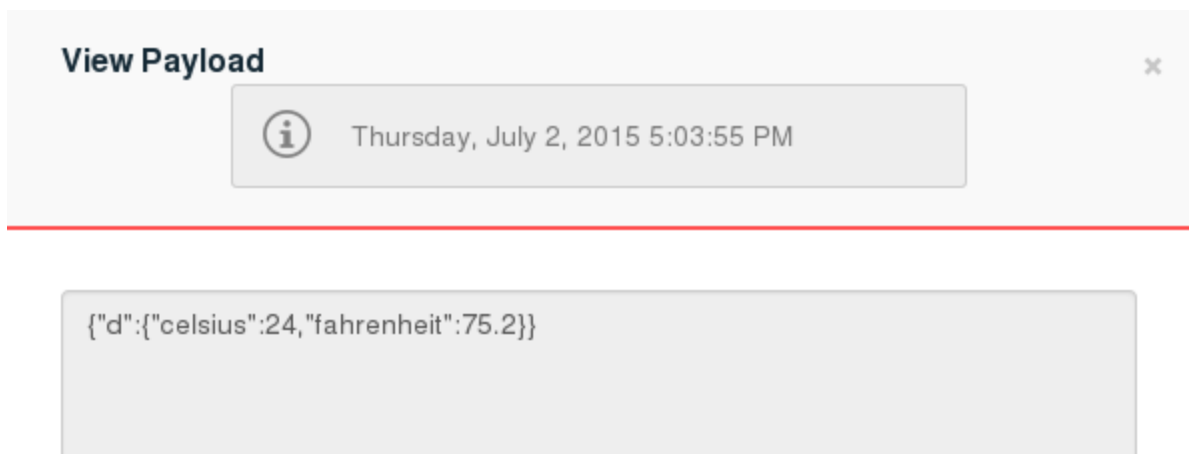


Figure 1-12: Contents of a Last Event payload

We're all connected up to the IoT Foundation!
Now to build an app that uses that data!

Part Two: Create an App Using the Node-RED Visual Editor

1. We start by navigating to Bluemix.net and logging in.
2. Now, select the **CATALOG** from the navigation menu and select the **Internet of Things Foundation Starter** from the Boilerplates.

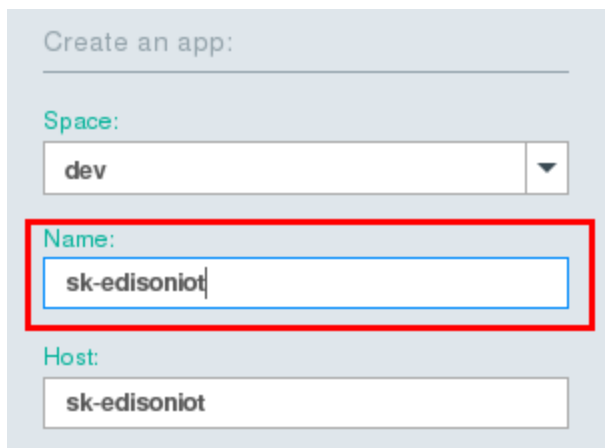


Internet of Things
Foundation Starter

IBM

Figure 2-1: The Internet of Things Foundation Starter boilerplate icon

3. Like with the IoT Service, we give the app a unique name. The host data will fill in to mimic the Name.



Create an app:

Space: dev

Name: sk-edisoniot

Host: sk-edisoniot

Figure 2-2: Enter a name in the Name text box of add app form

4. We select **CREATE** and the application staging screen displays. Select **Overview** from the left menu to get to the application dashboard.



Figure 2-3: Application Overview screen

TIP: If you can't see the left menu, select the double carrot next to 'Back to Dashboard'.

- When APP HEALTH display “**Your app is running**” we select the route for the app (it's web address) to open the Node-RED launch screen and confirm the app is running properly.

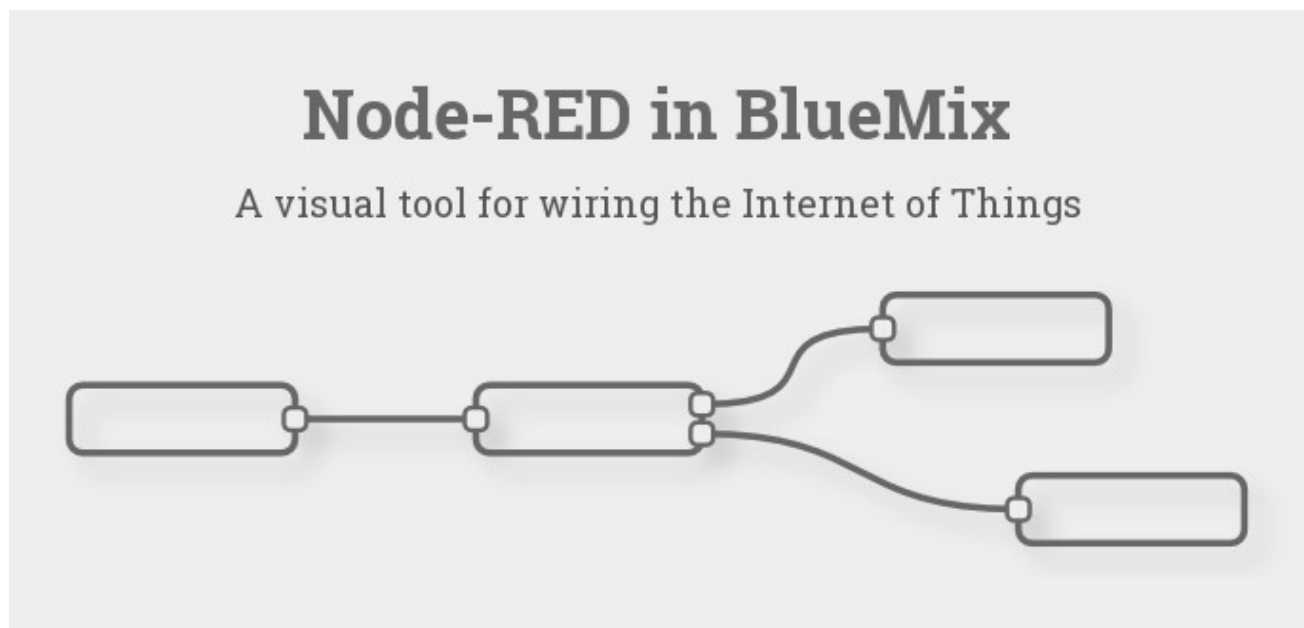


Figure 2-4: Node-RED launch screen

TIP: Routes is beneath the name of the application next to the application icon.

- We now need to bind our IoT service from Part One to this application. From our app Overview we select the **+ BIND SERVICE OR API** button.
- In the Add Service window we select our iotf-service and then select **ADD**.

Add Service to 'sk-edisoniot' Application

Select the previously used service instance that you want to add to your applicat



	NAME	SERVICE	VERS
<input type="radio"/>	 pi-tweetmewatson	personality_insights	
<input checked="" type="radio"/>	 sk-iot	iotf-service	

Figure 2-5: Add Service window

- When the application prompts for a restage we select **RESTAGE** and wait for the app to redeploy.

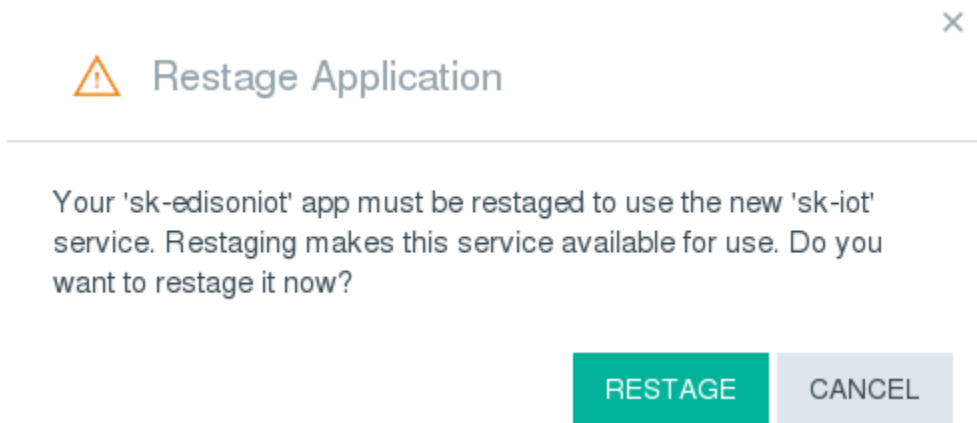


Figure 2-5: Restage prompt

- Once the app health reaches running again, we select our app Route again to open the Node-RED launcher. To launch the Node-RED editor click the red **"Go to your Node-RED flow editor"** button.

This boilerplate comes with a starter flow that just so happens to be for temperature data. That's half the work done for us!

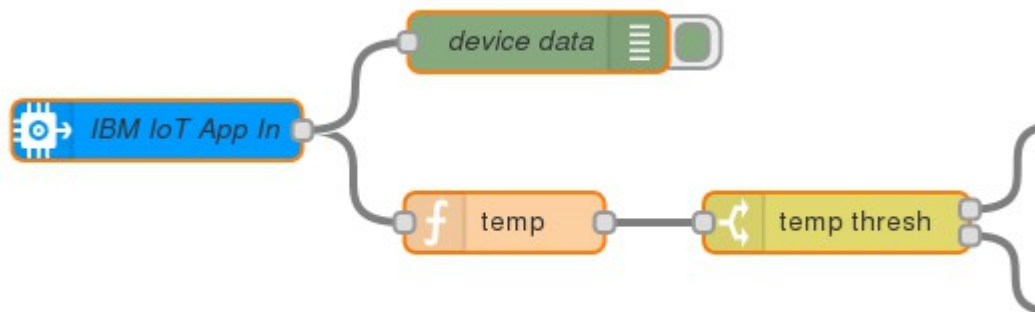


Figure 2-6: Starter flow on the Node-RED workspace

10. We need to make a few minor changes. To start, double-click the blue **IBM IoT App In** node to open the edit window. The following changes will switch from the quickstart to the Edison in our IoT Foundation:
- Authentication – Bluemix Service
 - Device Type – deselect All and enter Edison
 - Device Id – the MAC address of the Edison

Edit ibmiot in node

Authentication

Input Type

Device Type ☐ All or

Device Id ☐ All or

Event ☒ All or

Format ☐ All or

Name

Use the Input Type property to configure this node to receive Events sent by IoT Devices, Commands sent to IoT Devices, Status Messages referring to IoT Devices, or Status Messages referring to IoT Applications
Check the info tab, to get more information about each of the fields

Ok Cancel

Figure 2-7: Edit ibmiot in node window

Select **Ok** to save the changes and exit the window.

11. The **Deploy** button at the top of the screen is now a vibrant red. To test our input we select the Deploy button and switch to the debug tab. This debug tag is receiving data via the green device data node in our flow.



Figure 2-8: Deploy button at top of Node-RED page and debug tab

12. We should now see the message object from our Edison. If we want to stop the messages we can toggle the green tab on the right side of the **device data** node.

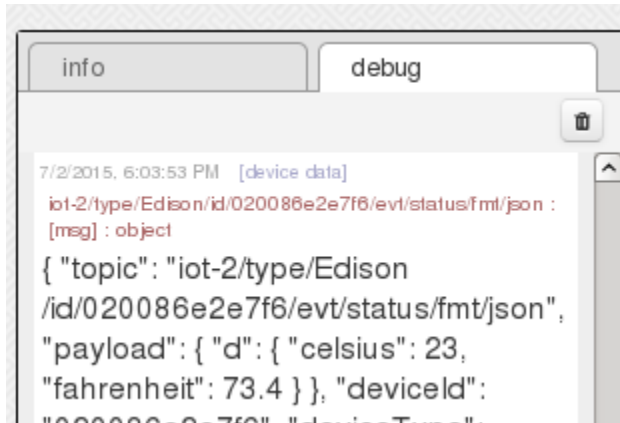


Figure 2-9: Debug tab with msg object from device data node

TIP: If you do not see a string of messages, check that your app is running in the Intel IDE and in the DEVICES tab of the IoT Dashboard that data is being received from the Edison.

13. At this point we just have a big dump of data and not much is being done by the rest of the flow even though this flow is supposed to handle temperature data. Let's see what the problem is by double-clicking the pink **temp** function node.

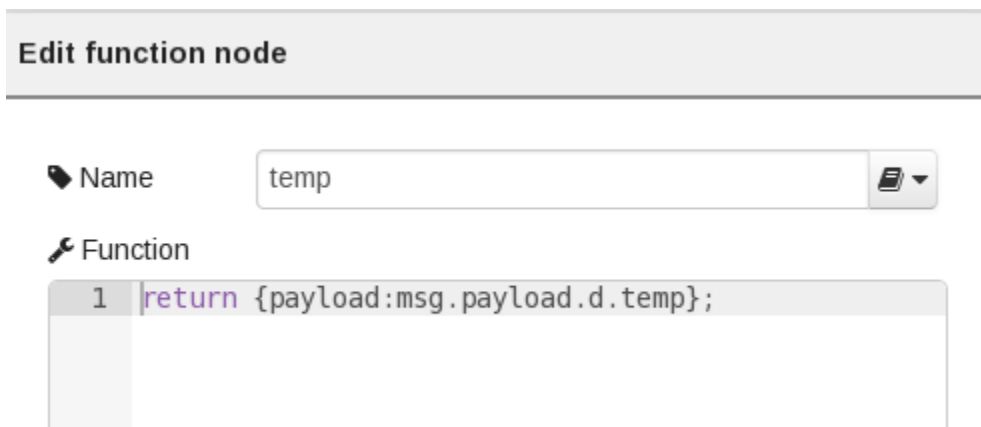


Figure 2-10: Edit function node window

As we can see the function is trying to parse out a temp item, but our Edison is sending Celsius and Fahrenheit. We need to change the return to pull the correct data. Modify the Function code as follows:

```
return [{c:msg.payload.d.celsius, f:msg.payload.d.fahrenheit}];
```

14. Now we double-click the yellow **temp thresh** node and modify it to evaluate the Celsius value.

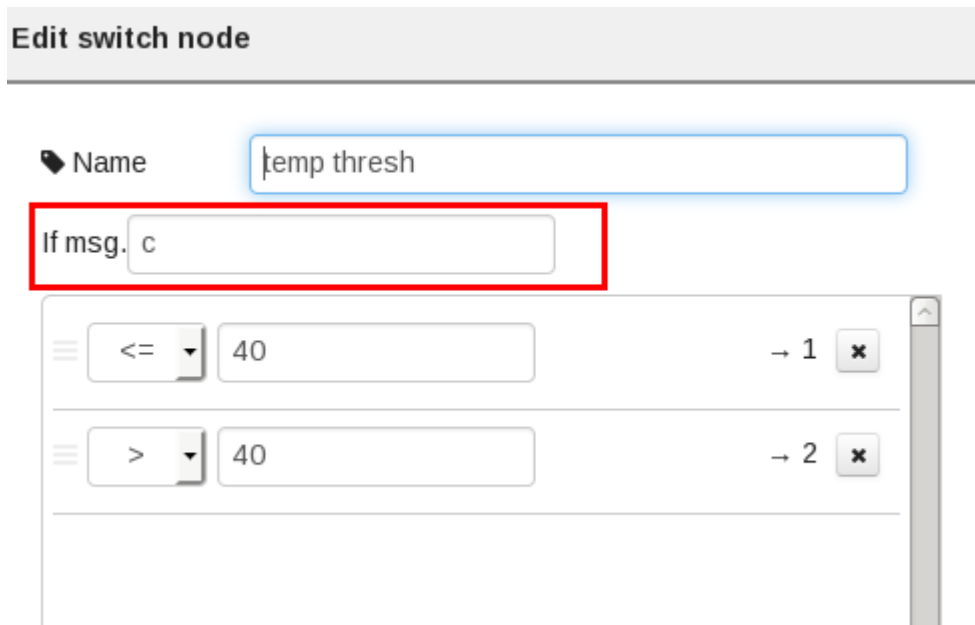


Figure 2-11: Edit switch node window with *msg.c* instead of *msg.payload*

15. The following two orange nodes are template nodes used to format a message. Right now we have a chunk of raw data, but using these nodes we can make a cleaner, more readable output. Open the **safe** and **danger** nodes and update their values as follows:

Safe Node:

Temperature {{c}}C/{{f}}F within safe limits

Danger Node:

Temperature {{c}}C/{{f}}F critical

16. Last, we need to see our output so we open the green **cpu status** node set the “to” property to the debug tab.

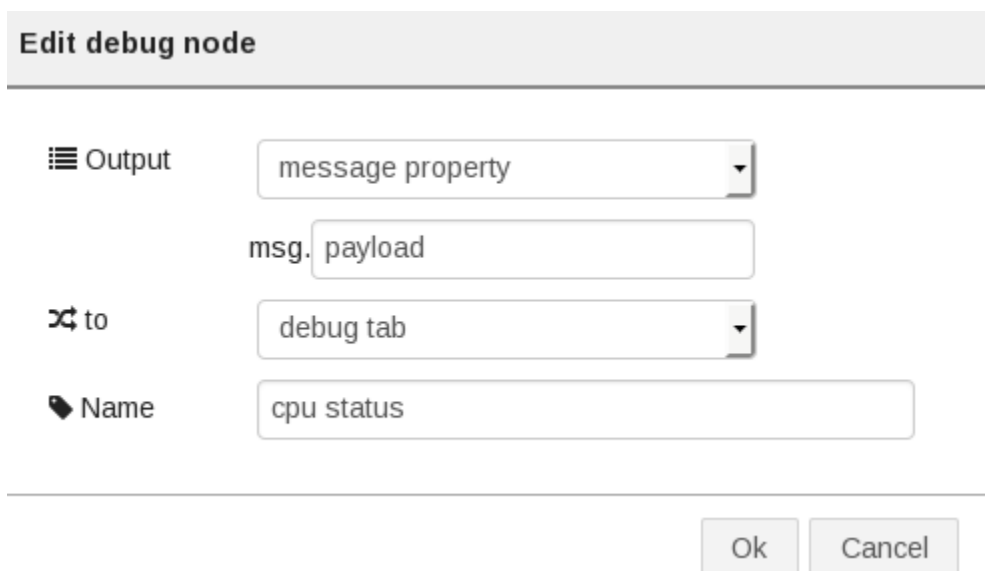


Figure 2-12: Edit debug node window

17. Now we Deploy the flow and watch the debug tab for messages.

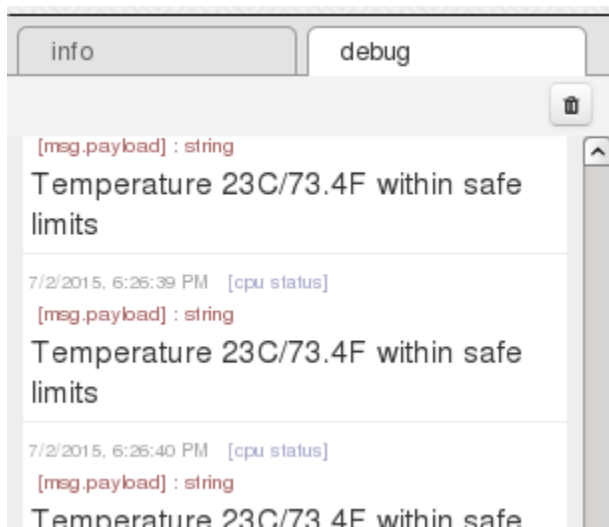


Figure 2-13: Debug tab with new formatted messages

Success!

We have data in Node-RED! Now to do something with it.

Part Three: Output Messages to Grove LCD RGB Backlight

This section walks through a simple way to send commands back to a device using the IBM IoT Out node. Get out your Grove LCD RGB Backlight and connect it to the I2C port next to the A0 port the temperature sensor is plugged in to. See the picture at the end of this section if you are not sure which port to use.

1. The IBM IoT Out node requires a specifically formatted payload. To get this format we are going to use two function nodes, one for safe and one for danger. In the node toolbox on the left, drag two function nodes over to the workspace.
2. To connect nodes, select the gray square on the right of the safe/danger node and drag the orange line that appears to the square on the left of a function node.

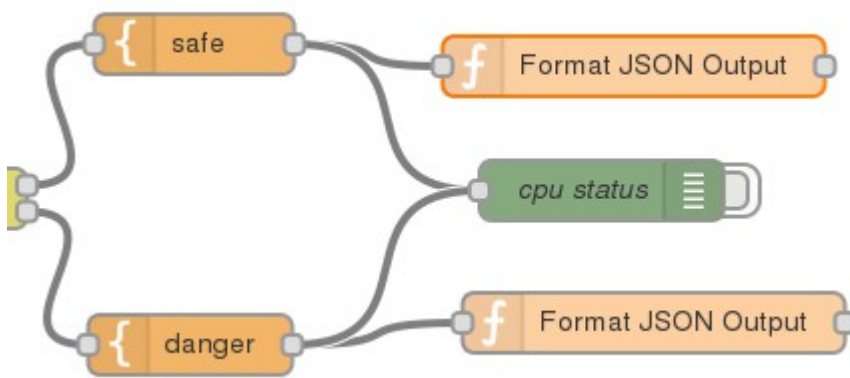


Figure 3-1: Connected function nodes in the flow

3. We edit each of the function nodes with the following logic:

safe function node:

```
var text = msg.c+' C('+msg.f+' F)';

var m = '{"d":{"text":"","text":"","color":"green"}}';
var msg = {};

msg.payload=m;
return msg;
```

danger function node:

```
var text = msg.c+' C('+msg.f+' F)';

var m = '{"d":{"text":"","text":"","color":"red"}}';
var msg = {};

msg.payload=m;
return msg;
```

4. Now to add an IBM IoT out node to send data to our Edison.

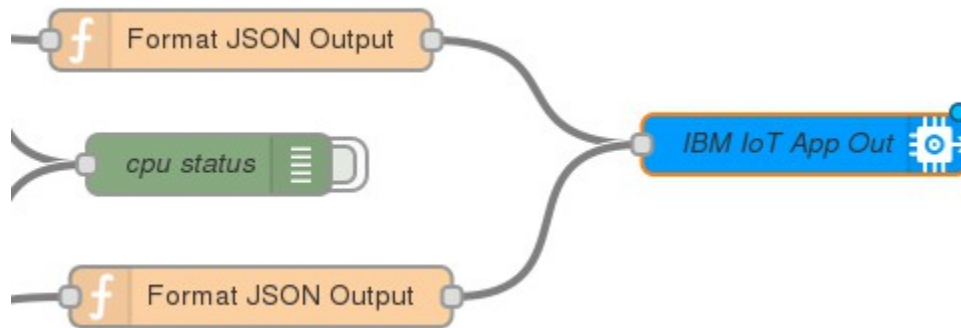


Figure 3-2: IBM IoT App Out node connected to the two function nodes

5. Edit the IBM IoT out node with the following credentials.
 - Authentication: Bluemix Service
 - Output Type: Device Command
 - Command Type: display
 - Format: json
 - Data: {"d":{""}}

Edit ibmiot out node

Authentication	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Bluemix Service</div>
Output Type	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Device Command</div>
Device Type	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">Edison</div>
Device Id	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">[Placeholder]</div>
Command Type	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">display</div>
Format	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">json</div>
Data	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">{"d":{""}}</div>
Name	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">IBM IoT App Out</div>

Note: If there is a property in the message that corresponds to any of the values entered above, then the property in the message takes precedence. See the Info tab for more details.

Example JSON device event: {"d":{"myName":"Arduino Uno", "temperature":989}}

Ok

Cancel

Figure 3-3: Edit ibmiout out node window

6. Deploy the Node-RED flow and navigate back to the Intel IDE.
7. Our code has a number of commented out lines for the LCD display functionality. Now we'll remove the `//` and `/* */` comment indicators.

- Under the header **'//Load Grove and MQTT modules'** remove the comment characters next to:

```
var LCD = require('jsupm_i2clcd');
```

- Under the header **'// Set up variables for LCD display'** remove the comment characters next to

```
var text = " ";  
var myLCD = new LCD.Jhd1313m1(6, 0x3E, 0x62);
```

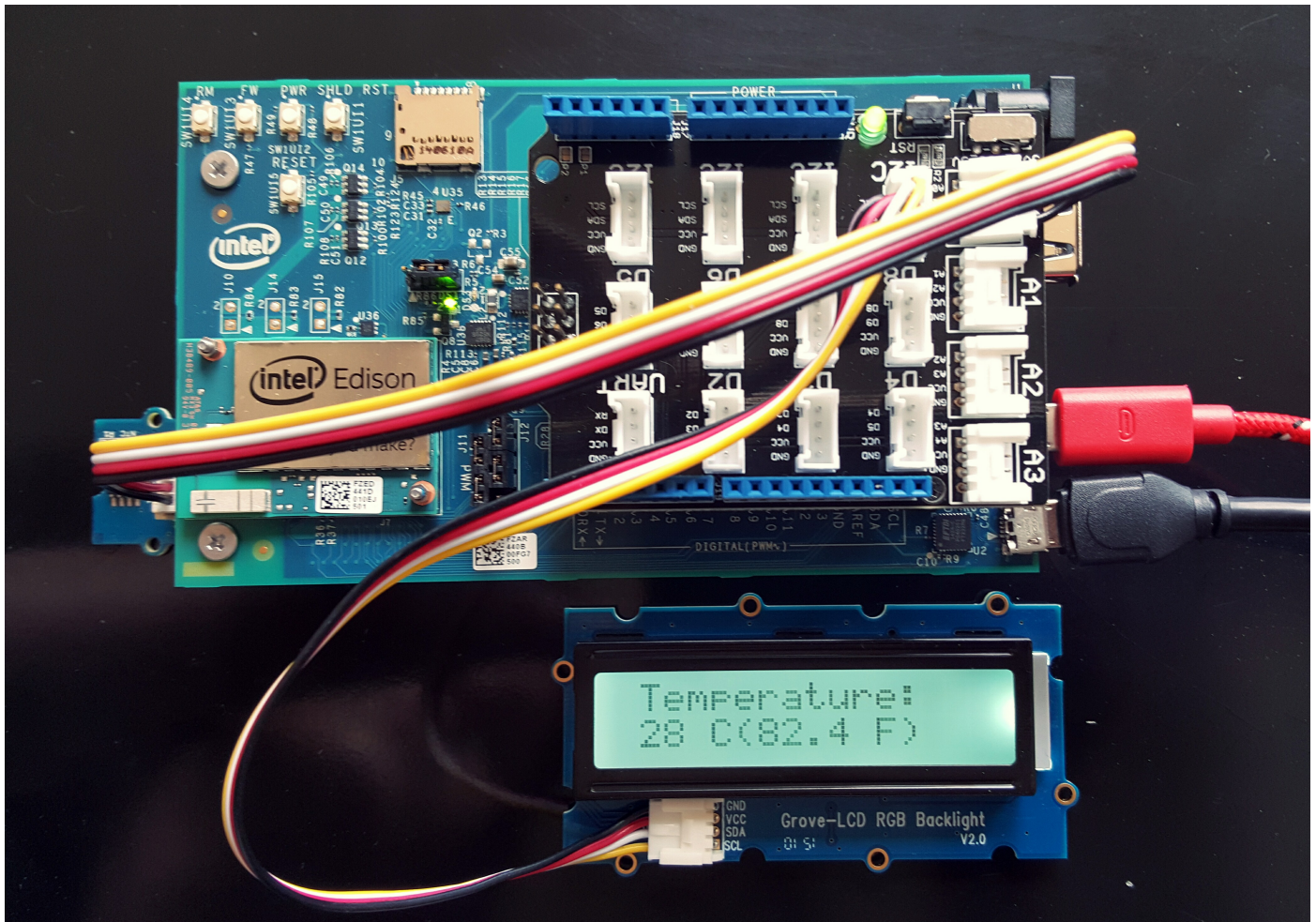
- In the **client.on('message')** function, remove the comments next to the call to `lcdDisplay`:

```
lcdDisplay(msg);
```

- Last, we remove the comment characters (`/*` and `*/`) around the **lcdDisplay function** at the bottom of `main.js`:

```
function lcdDisplay(data) {  
  var text = data.d.text;  
  var color = data.d.color;  
  myLCD.clear();  
  if (color === "green") {  
    myLCD.setColor(110, 215, 089);  
  }  
  if (color === "red") {  
    myLCD.setColor(210, 058, 053);  
  }  
  myLCD.setCursor(0, 1);  
  myLCD.write("Temperature:");  
  myLCD.setCursor(1, 1);  
  myLCD.write(text);  
}
```


8. Now we build, upload, and run the application.



This concludes the lab!

Continue to Part Four to work with Twilio, Twitter, or Cloudant nodes.

Part Four: Fun with Other Nodes

This is a bonus step in the lab if you have extra time or want to play with something other than the LCD. There are three nodes discussed in this section:

- Cloudant
- Twitter
- Twilio

Skip to the section for the node that interests you or do all three!

The Nodes: Cloudant Node

The boilerplate we selected already has a Cloudant service bound to it, so for this exercise we are going to set up a node to store our temperature payload from the IBM IoT in node.

1. To add Cloudant we scroll through the node toolbox on the left until we get to the storage section. We want to strictly output data from our device to the Cloudant database, so we'll need the node with only an output. We drag the node over to our flow and set it somewhere.

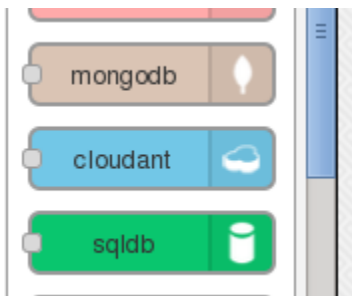


Figure 4-1: Cloudant output node

2. Since we want the raw data to be stored, we need to click on the grey box on the output side of the IBM IoT App In, it will turn orange on hover, and drag the line over to the input on the cloudant node.

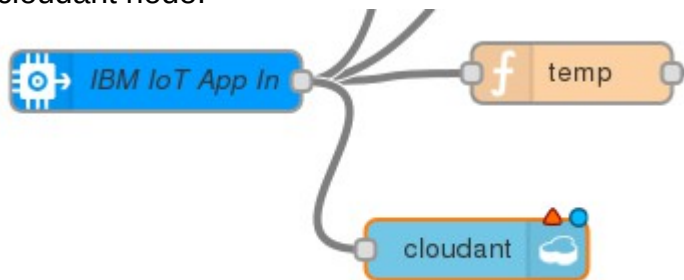


Figure 4-2: Cloudant node in Node-RED flow

3. Opening up the dialog for the Cloudant db we see that there is no service assigned to this node. We now select the Cloudant service bound to our application as our Service. Next we give the database a name and check the box to only store the msg.payload object.

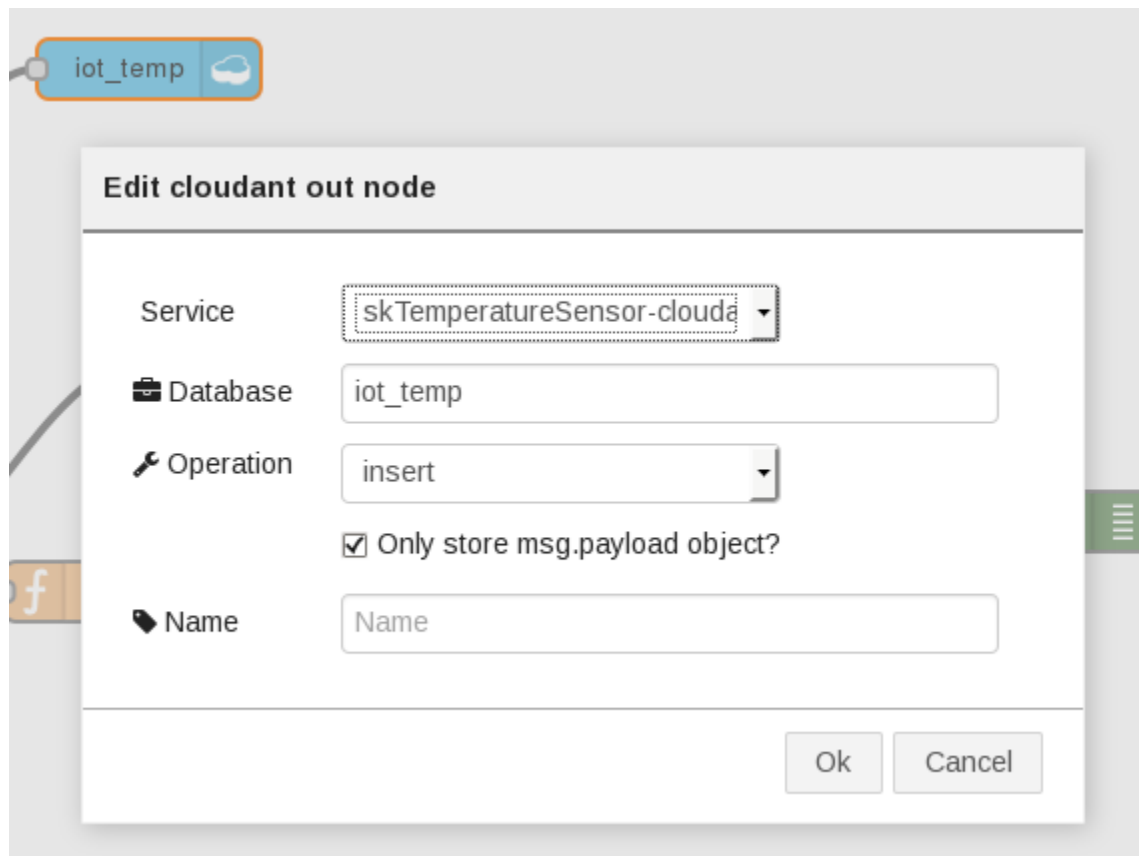


Figure 4-3: Edit Cloudant out node dialog

4. We redeploy to commit the changes and navigate over to Bluemix to check if data is coming to our database. To view the database, we select the Cloudant NoSQL DB service from our application's Overview and select then LAUNCH.



Figure 4-4: Launch button for Cloudant

5. We now select our database, `iot_temp` in this case.

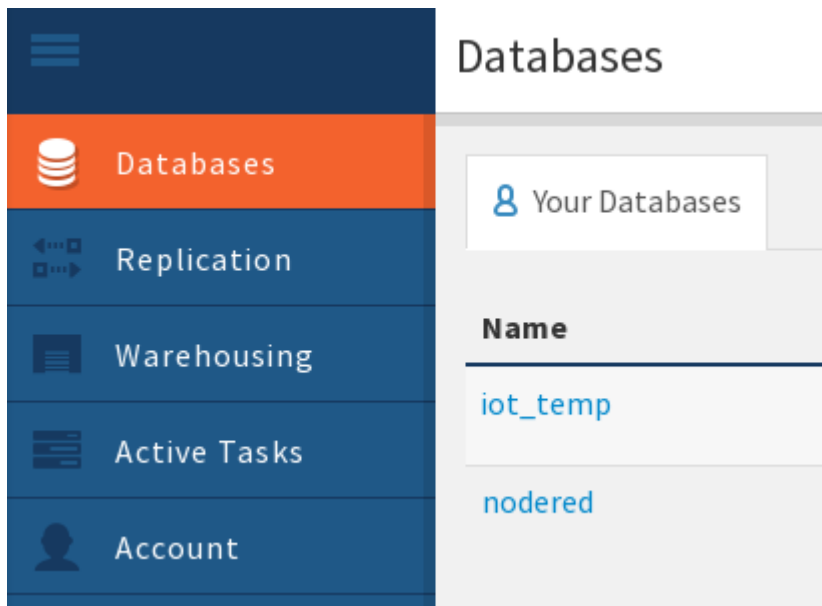


Figure 4-5: Databases view

6. In the right pane there are several documents that look like a bunch of senseless data and numbers. Clicking on the pencil icon in the upper right corner reveals a little more about these json payloads.

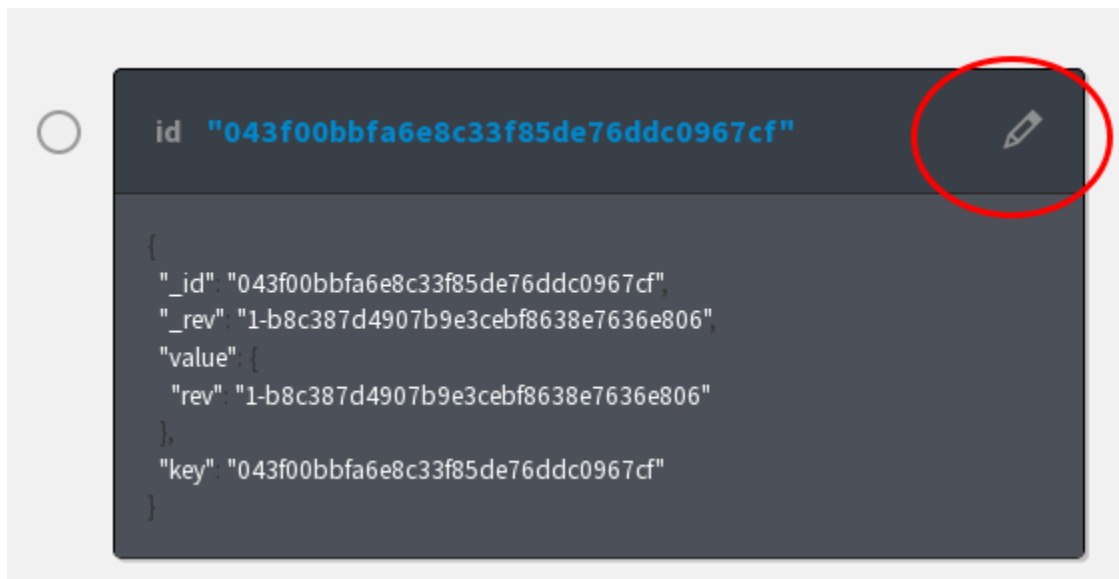


Figure 4-6: Edit icon for document

7. The data contains the temperature data our Node-RED flow is handling as well as some humidity data.

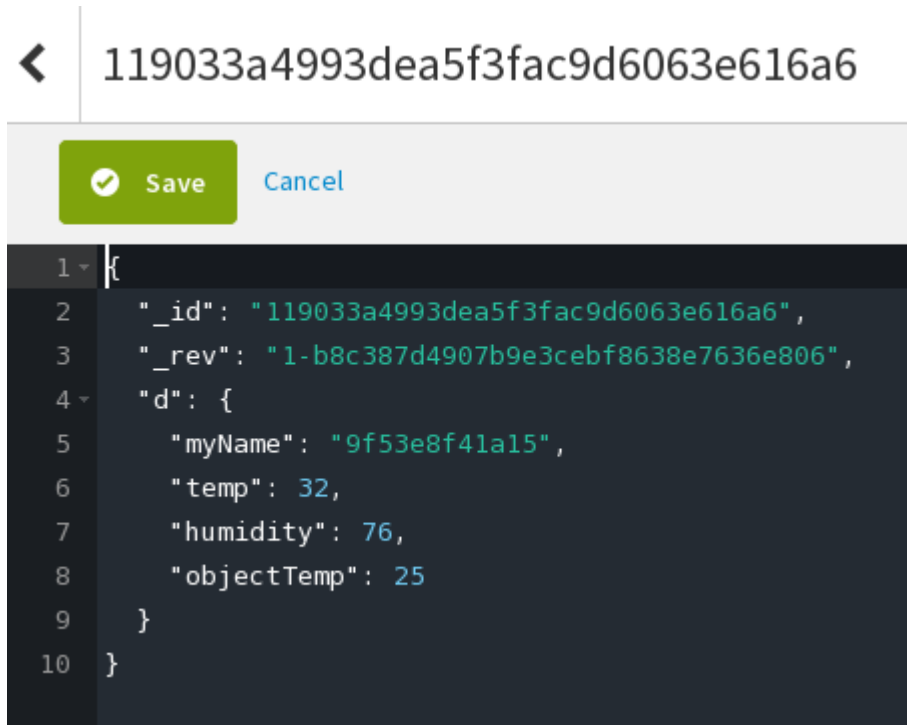


Figure 4-7: Contents of the document

The Nodes: Twitter Node

To use this node you need a Twitter account. If you do not have one, you can bribe your neighbor to let you use their credentials.

1. Having our data readily available and archived is interesting, but maybe we want to have real-time notifications in a more easily accessible output. Scrolling up and down the node toolbox we can see dozens and dozens of nodes linking to services such as Watson and Twilio. We drag the twitter output node over to our flow to get started.

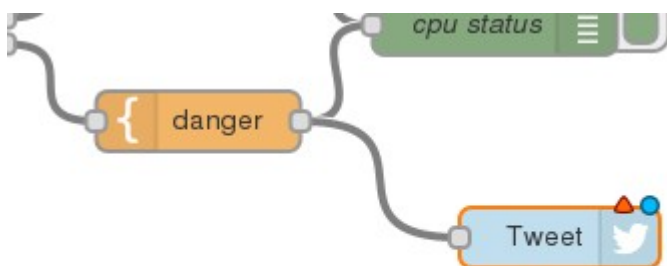


Figure 4-8: Connected Twitter node

2. In this case we only want to tweet when the temperature is out of the safe limits and we'll add a delay node to keep from spamming our Twitter timeline. To do this we drag the delay node over from the function section of the toolbar. The node is going to receive input from the danger node and output to the Tweet node. We'll set a delay of one minute on messages.

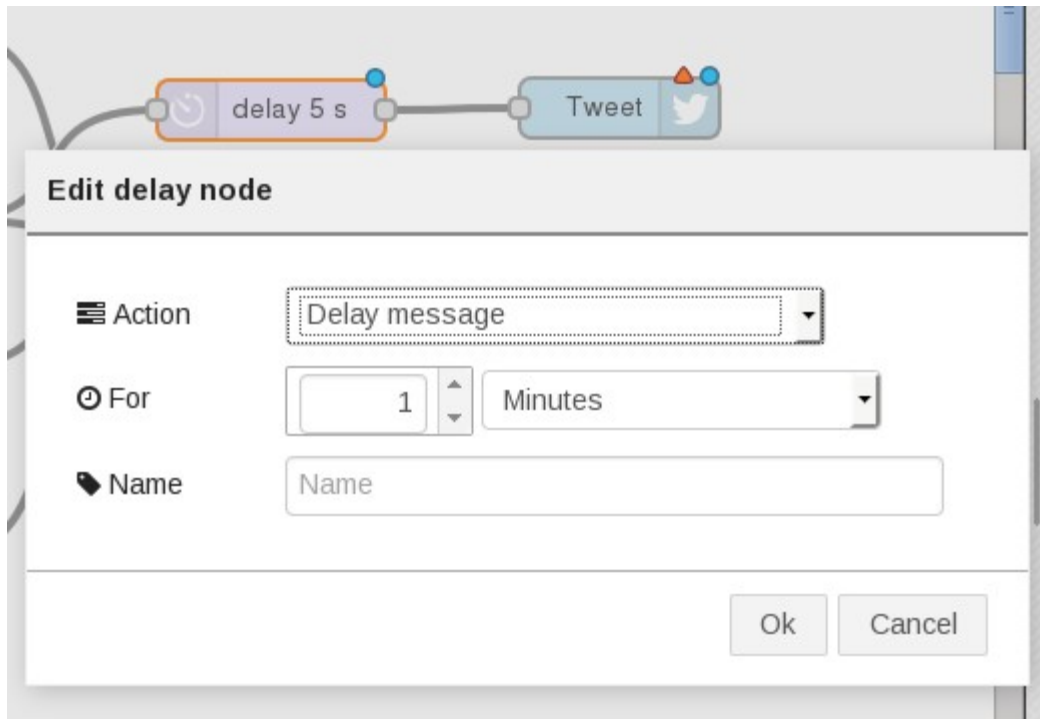


Figure 4-9: Add a delay node

3. Now we open the dialog box for Tweet and select the pencil icon to the right of the Twitter field. Node-RED will now walk us through authenticating with Twitter.

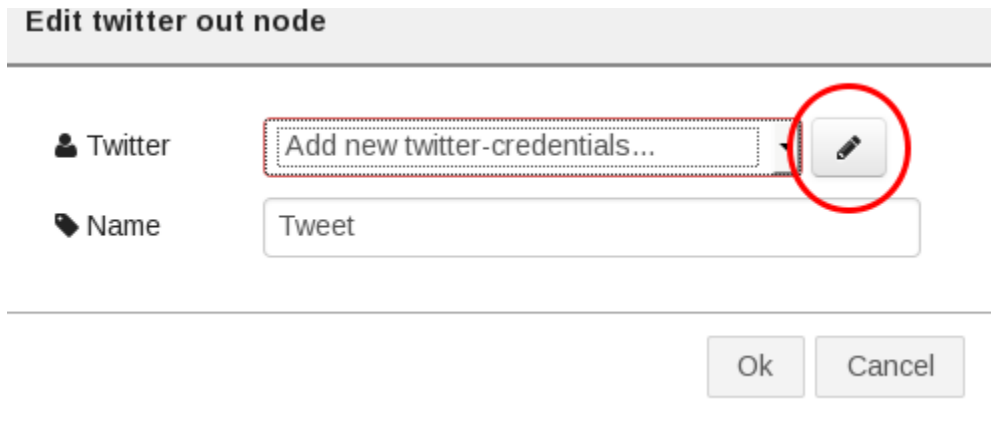
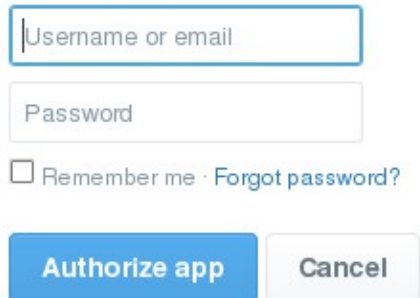


Figure 4-10: Edit twitter out node dialog

4. We'll log in using our Twitter credentials to authorize Node-RED to access our Twitter account.

Authorize Node RED to use your account?



A Twitter authorization dialog box. It has a title "Authorize Node RED to use your account?". Below the title are two input fields: "Username or email" and "Password". Below the password field is a checkbox labeled "Remember me" and a link "Forgot password?". At the bottom are two buttons: "Authorize app" (blue) and "Cancel" (grey).



Figure 4-11: Authorize Node RED to Twitter

5. Upon successful authentication we should see the following message. Back in Node-RED we click Add to add the new user and Ok to close out the dialog.

Authorised - you can close this window and return to Node-RED

Figure 4-12: Twitter connect success

6. We now redeploy the application to pick up our changes and navigate either to the temp node or the IoT simulated sensor and alter the threshold to trip the danger node.

The Nodes: Twilio Node

Like Twitter, using the Twilio node requires that you have a Twilio account. They have a free level, so go to <https://www.twilio.com/try-twilio> and sign up.

1. To start, we'll need the Twilio out node, so drag the red node from the toolbox to the workspace. Like with Twitter we only want to know when the temperature hits critical so we'll connect our node to the danger node.

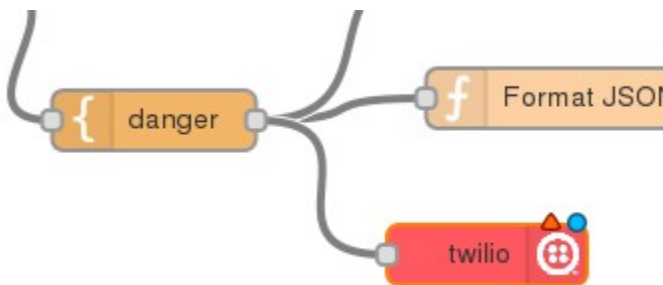


Figure 4-13: Twilio node in the workspace

2. Double-click the node to open the edit window and select **External Service** as the Service from the drop-down. Now we need to add our Twilio credentials so we select the pencil icon next to 'Add new twilio-api...'

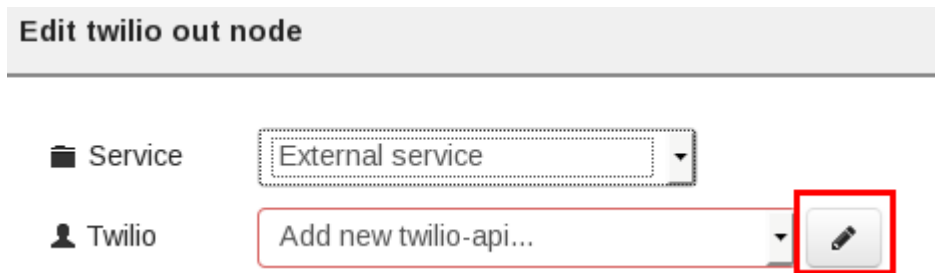
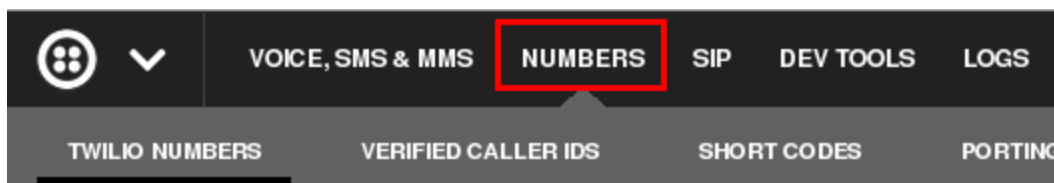


Figure 4-14: Edit twilio out node window

3. The information we need is stored in our account details on twilio.com. Opening a browser to the twilio website, we select **Numbers** from the top navigation menu.



Manage Numbers

Figure 4-15: The NUMBERS tab in twilio.com account

4. We select a number from the list (likely only one number is displayed) and copy the relevant data from Twilio into Node-RED.
5. The dialog also asks for a Token which is actually the API key. To find our API key, on Twilio.com we select Voice, SMS, & MMS from the top navigation. From there select Show API Credentials on the right side of the page.

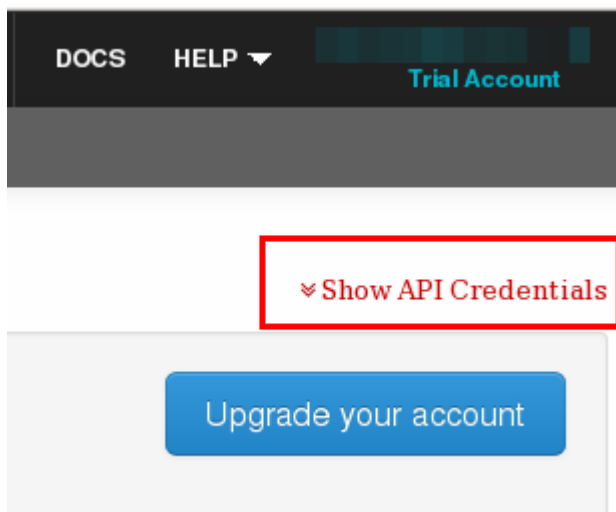
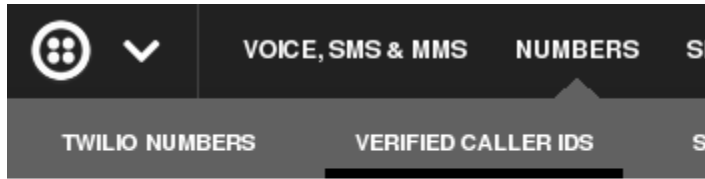


Figure 4-16: Show API Credentials link

Select **Add** to exit the dialog when done.

6. For the SMS to enter your cell phone number. This number also needs to be in our Verified Caller Ids on Twilio.



Manage Caller IDs

Figure 4-16: VERIFIED CALLER IDS for adding approved numbers

7. Save the node and deploy the flow. Trip the danger threshold for the temperature and watch the text messages start flying in!