Develop IoT with Samsung ARTIK platform

Samsung Strategic and Innovation Center

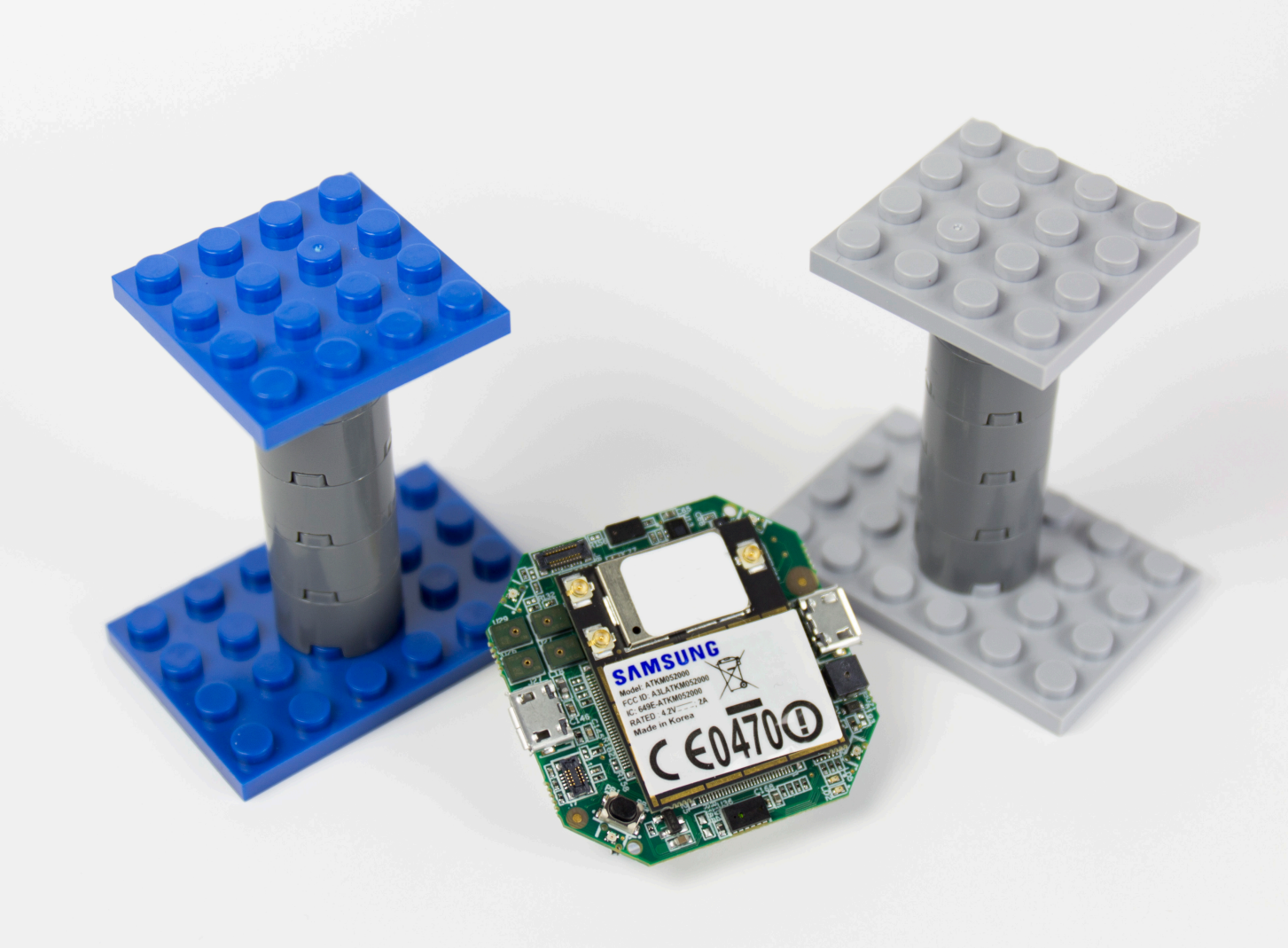


Table of Contents

Introduction

Welcome to the Samsung ARTIK workshop! This workshop is your introduction to developing IoT with the Samsung ARTIK platform. In this session, you will learn how to:

- Access your target board from the serial console
- Build program flows using Node-RED
- Collect sensor data
- Use MongoDB for storing and retrieving your information
- Enable MQTT so that your ARTIK boards can communicate with each other
- Use ARTIK SDK to set up a ZigBee network
- Stream data to ARTIK Cloud
- Use the ARTIK Cloud Rules Engine to trigger cross-device actions

Before you begin

1. Bring your own laptop. For Windows users, pre-install PuTTY.

   PuTTY (http://www.putty.org/) – SSH and Telnet client, for serial console access

2. Have a Gmail account: ARTIK will send emails to your Gmail account. In order to do this, we need to turn on "Access for less secure apps" setting in your Gmail.

   If you are using a corporate Gmail account or if you want to keep your project emails separate from your personal emails, please create a new account for the workshop.

3. Establish an ARTIK Cloud user portal account: We will stream data to ARTIK Cloud service. Create an account at ARTIK Cloud user portal (https://artik.cloud/).

Setup

1. Access Kitra from your serial console. Instructions can be found here:

   https://github.com/rushup/kitra520/wiki/Getting-started-with-Kitra520

   To power up Kitra, press the power button until you hear 2 beeps, then connect its serial USB to your host machine. Follow the instruction in the link below to access the board from your serial console.

   https://developer.artik.io/documentation/getting-started-beta/communicating-pc.html

2. Connect Kitra to the network.

   Your Kitra board should automatically obtain an IP address after it boots up. Do a 'ping' test from your console and take note of your board IP address. If you are not able to obtain an IP address, please go to step 3.

```
[root@localhost ~]# ping www.google.com
PING www.google.com (74.125.28.103) 56(84) bytes of data.
64 bytes from nuq04s18-in-f4.1e100.net (74.125.28.103): icmp_seq=1 ttl=52 time=8.83 ms
64 bytes from nuq04s18-in-f4.1e100.net (74.125.28.103): icmp_seq=2 ttl=52 time=59.9 ms
64 bytes from nuq04s18-in-f4.1e100.net (74.125.28.103): icmp_seq=3 ttl=52 time=17.2 ms
```

   (Ctrl-C to terminate)

```
[root@localhost ~]# ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.0.2  netmask 255.255.255.0  broadcast 10.0.0.255
    inet6 2601:647:4e01:7b45:489d:21ff:fe85:6c27  prefixlen 64  scopeid 0x0<global>
    inet6 fe80::489d:21ff:fe85:6c27  prefixlen 64  scopeid 0x20<link>
    ether 4a:9d:21:85:6c:27  txqueuelen 1000  (Ethernet)
    RX packets 14769  bytes 20317291 (19.3 MiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 311790 (304.4 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. In case you are not able to obtain an IP address, please restart dhclient by following the steps below, then repeat step 2.

```
[root@localhost ~]# dhclient wlan0 –r
Killed old client process
[root@localhost ~]# dhclient wlan0
```

Exercise 1: Monitor sensor data and set up real time sensor dashboard (Node-RED)

In this exercise, we can utilize sensors on Kitra to monitor temperature of each pole, detect if a pole is falling and if there is an earthquake. At the end of the exercise, we will set up a real time sensor dashboard.

1. Start Node-RED in the background from your serial console.

```
[root@localhost ~]# node-red &
…
Welcome to Node-RED
===================

29 Jan 13:53:43 - [info] Node-RED version: v0.15.2
29 Jan 13:53:43 - [info] Node.js  version: v4.6.1
…
29 Jan 13:53:48 - [info] Server now running at http://127.0.0.1:1880/
```

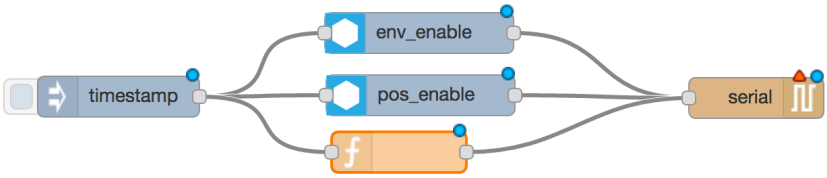
2. Open a browser on your laptop, and launch http://<your_board_ip_address>:1880/ . <your_board_ip_address> should be replaced by the IP address you obtained in Setup portionearlier.

3. Design your first project flow.

3.1 Create an ENABLE flow on the Node-RED canvas.

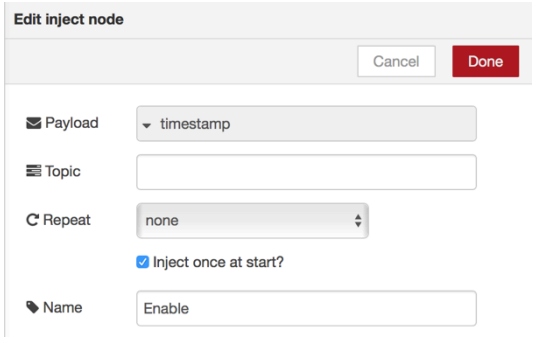
- Drag an “inject” input node from the node palette to the canvas (it initially shows as “timestamp”).
- Drag an “env_enable” node(*kitra_output* category)to the right of the “inject(timestamp)” node. Drag a “pos_enable” node(*kitra_output* category)and place it under “env_enable” node. Drag a “function” node(*function* category) and place it under “pos_enable” node.
- Drag a “serial” **out** node 

to the rightmost position
- Connect these nodes by dragging “wires” between them.

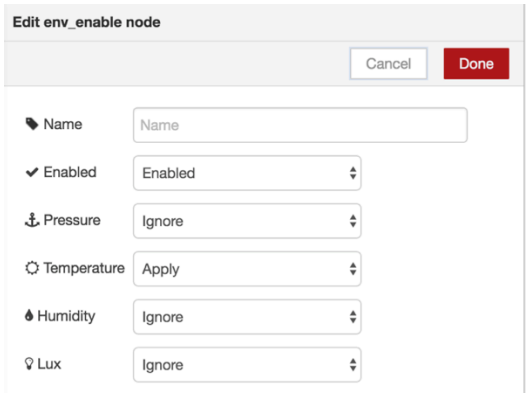


3.2 Configure each node by double-clicking.

- “inject” node(shows as “timestamp”): Enable “Inject once at start?” checkbox. Rename the node to “Enable”.



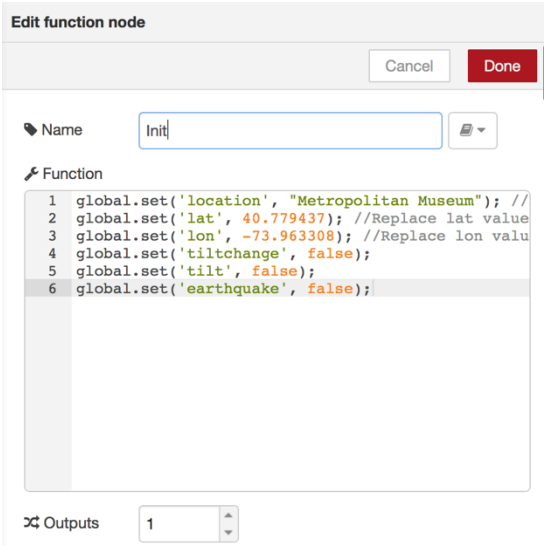
- “env_enable” node: In this node, we only enable Temperature, and ignore other sensor values. Set Temperature to “Apply”, and change Pressure, Humidity and Lux to “Ignore”.



- We keep the default setting in “pos_enable” node.
- In the “function” node, we initialize global variables to define our location, GPS coordinates etc. Please replace

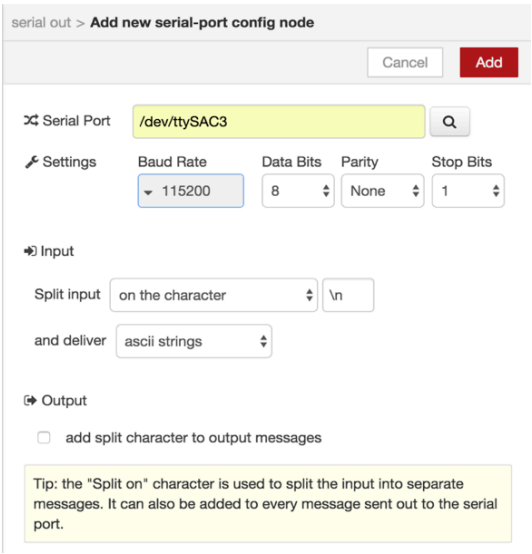
```
global.set('location', "Metropolitan Museum"); //Replace "Metropolitan Museum" with your own landmark location
global.set('lat', 40.779437); //Replace lat value with your own latitude
global.set('lon', -73.963308); //Replace lon value with your own longitude
global.set('tilt', false);
global.set('tiltchange', false);
global.set('earthquake', false);
```

the highlighted parameters with your own landmark location and GPS coordinates. Rename the node to “Init”.

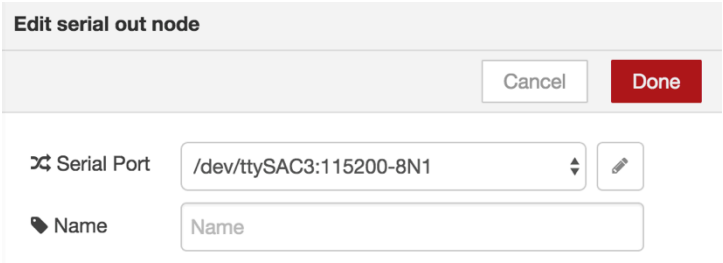


- "serial" node: Click the "Edit" button  next to Serial Port to add new serial port config node.

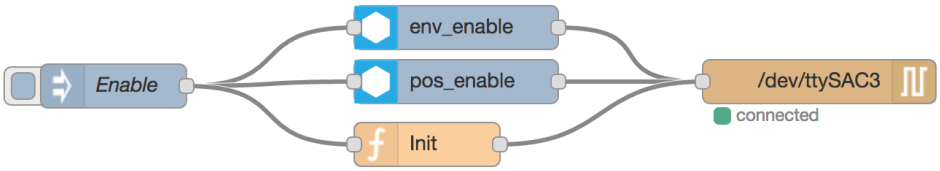
  In the "Add new serial-port config node" window, set the Serial Port to "/dev/ttySAC3", set the Baud rate to "115200", and click the "Add" button.



Click "Done" in "Edit serial out node" window



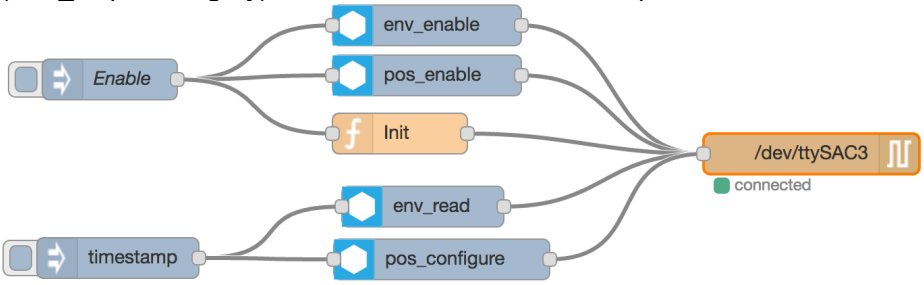
3.3 Click  at the upper right corner to make your application live. If everything is configured properly, you should see a "connected" status show up under serial out node "/dev/ttySAC3" now.



4. Add a READ flow

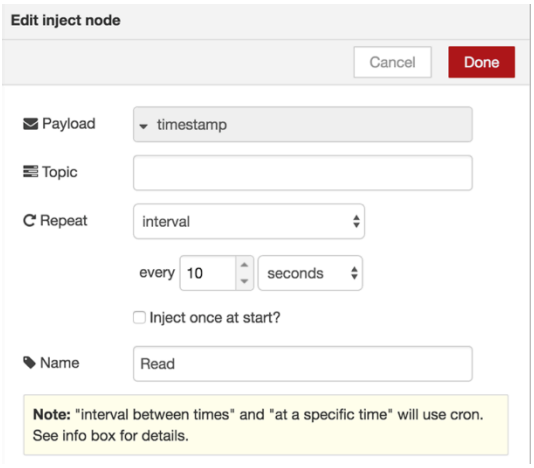
4.1 Add a READ flow to the canvas. In the READ flow, we define the sensor reading interval and configure the sensors to be monitored.

Drag an “inject” node (shows as “timestamp”), “env_read” node(*kitra_output* category) and “pos_configure” node(*kitra_output* category) to the canvas, and wire them up as below.

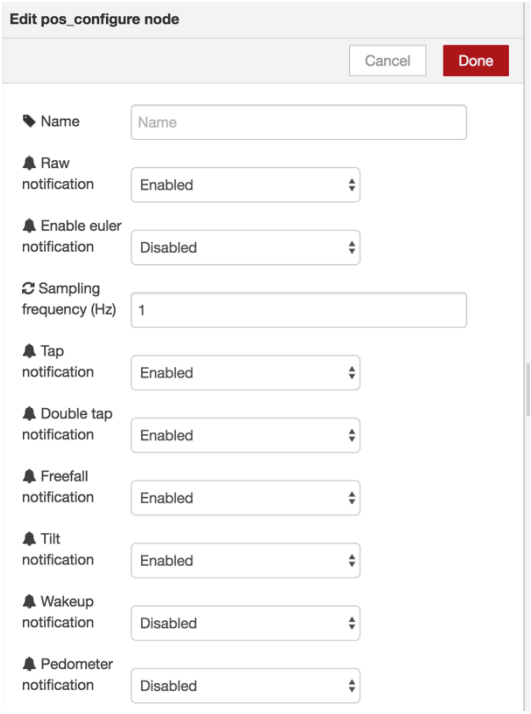


4.2 Double click each node to configure.

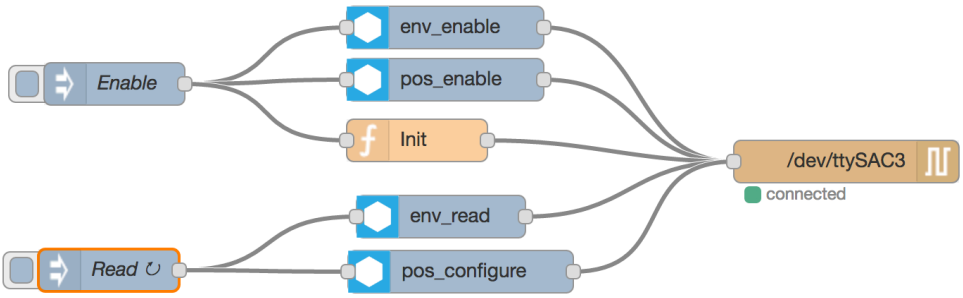
- “inject” node(shows as “timestamp”): Rename the node to “Read”. Change “Repeat” interval to 10 secs so that sensor data is collected every 10 secs.



- “pos_configure” node: In “pos_configure” node, we will enable the following notifications: “Raw notification” (used to calculate the inclination angle), “Tap notification”, “Double tap notification”, “Freefall notification” and “Tilt notification”.



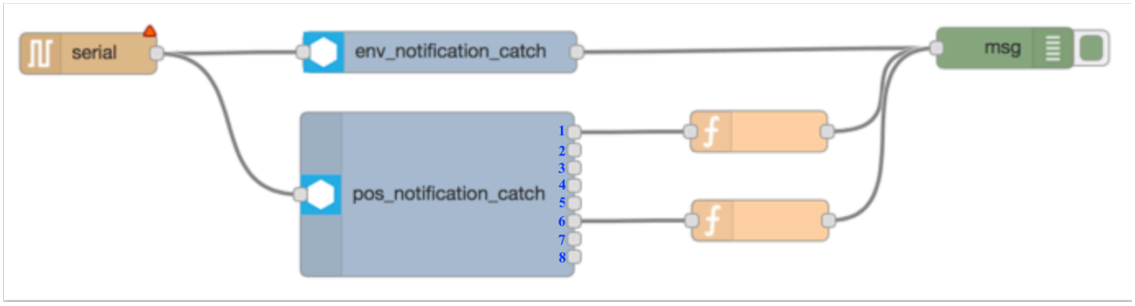
- Click “Deploy” to save your changes.



5. Add a CATCH flow to catch the sensor values and notifications.

5.1 Drag a “serial” **in** node 

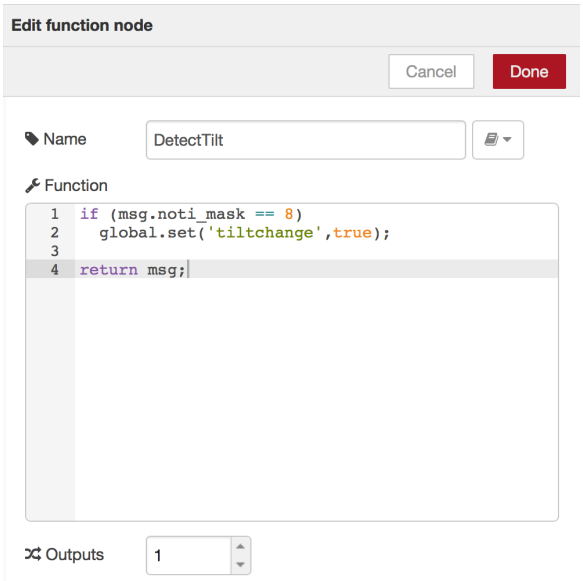
, an “env_notification_catch” node(*kitra_input* category) and a “pos_notification_catch” node(*kitra_input* category) to the canvas. Connect a function node to the 1st and 6th outputs of “pos_notification_catch” node respectively. Add a “debug” node as the output node.



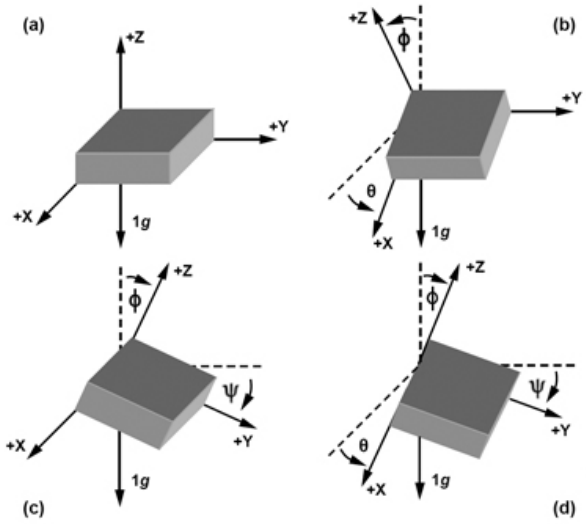
5.2 Double click each node to configure.

- Double click the “serial” **in** node, and make sure your Serial Port shows as “/dev/ttySAC3:115200-8N1”.
- The function node that connects to the 6th output (3rd from the bottom) of “pos_notification_catch” is triggered every time the inclination angle of your Kitra changes. In this function, we set “tiltchange” global variable to be true. Rename the function as “DetectTilt”.

```
if (msg.noti_mask == 8)
  global.set('tiltchange',true);

return msg;
```



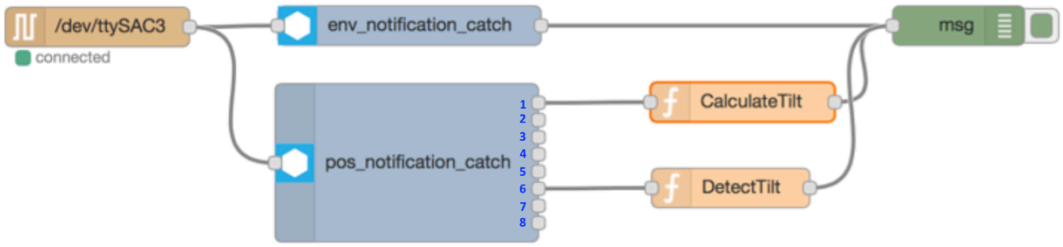
- In the function node that connects to the 1st output of “pos_notification_catch”, we calculate the angle between the surface of your Kitra board and the world XY plane (we call it inclination angle), and decide if the utility pole is tilted. If the inclination angle is greater than 155 degrees, we will consider the pole to be in a flat position, otherwise, the pole is in a non-flat position.



Rename the function node to “CalculateTilt”.

```
//Calculate the angle of the inclination
if (global.get('tiltchange') === true) {
    var acc_x = msg.acc_x;
    var acc_y = msg.acc_y;
    var acc_z = msg.acc_z;

    //Normalize the accelerometer vector
    var normal = Math.sqrt(acc_x * acc_x + acc_y * acc_y + acc_z * acc_z);
    acc_x = acc_x / normal;
    acc_y = acc_y / normal;
    acc_z = acc_z / normal;

    global.set('tiltchange', false);

    //Calculate the inclination
    var inclination = Math.round(Math.acos(acc_z) * (180 / Math.PI));
    console.log("inclination", inclination);

    if (inclination > 155) {
      global.set('tilt', false);
      return msg;
    } else {
      global.set('tilt', true);
      return msg;
    }
}
```

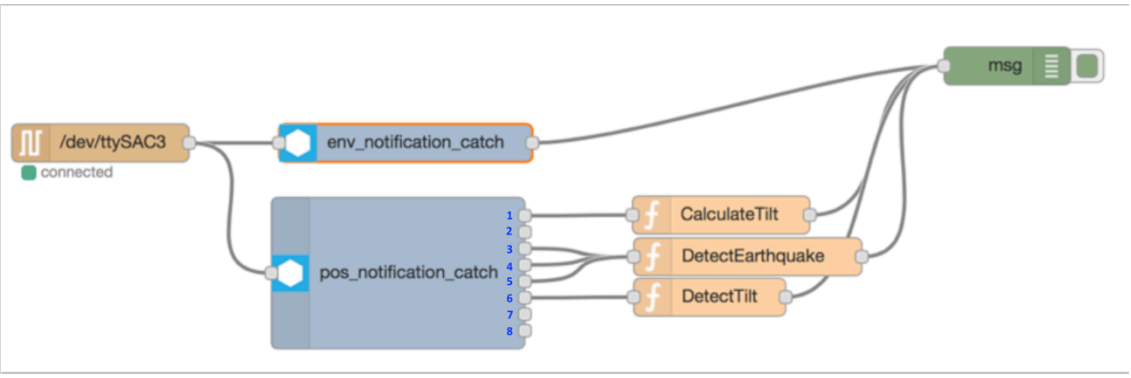
- Deploy the flow.



Now, in the debug panel, you will be able to see the environment sensor data every 10 secs. When you change the incline of your pole, the accelerator data will be displayed on the debug panel as well.

5.3 Add a function “DetectEarthquake” to capture “tap notification”, “double tap notification” and “free fall notification”. We will use these notifications to simulate an earthquake.

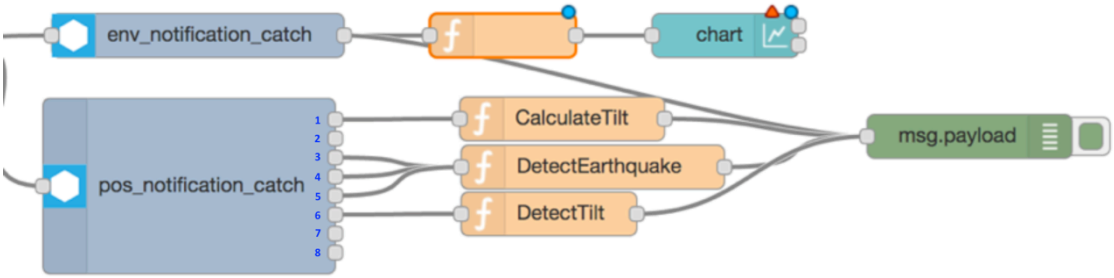
```
if (msg.noti_mask == 2 || msg.noti_msk == 3 || msg.noti_mask == 4) {
  global.set('earthquake', true);
}
return msg;
```



6. Set up a real time sensor dashboard. We will use the widgets under the “dashboard” category from the Node-RED palette for dashboard.

6.1

- Add a “function” node and a “chart” node to the output of “env_notification_catch” node. Wire them up



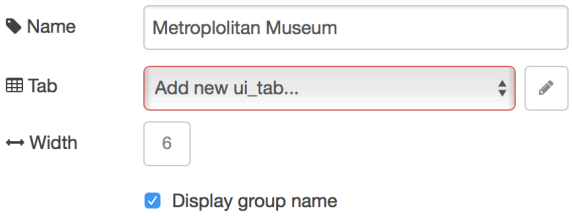
- Dashboard widgets use *msg.payload* as its data input. In the function node, we simply define our message payload

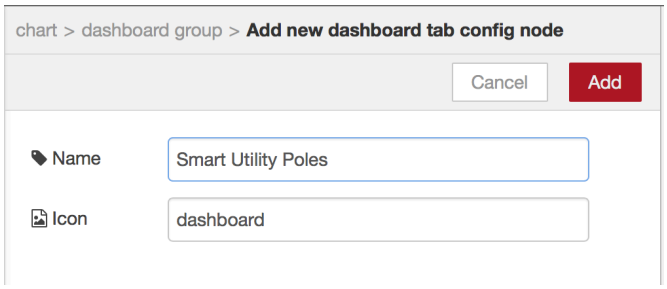
```
var temperature = msg.temperature;
var dbMsg = {payload: temperature};
return dbMsg;
```

as msg.temperature. Rename the node to “CatchTemp”.

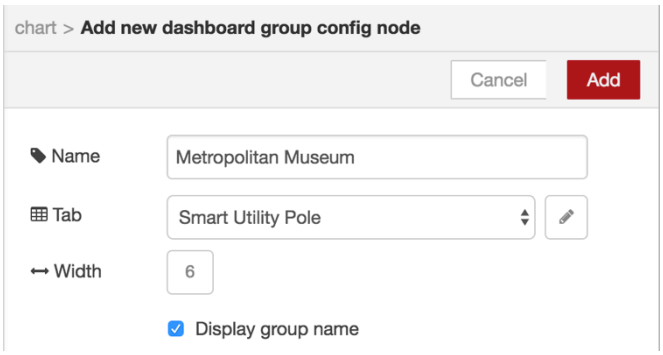
- In the “chart” node, from Group dropdown list, select “Add New ui_group”, then click the  button next to it to add a new UI group. Here, I use “Metropolitan Museum” as the name(Please use your own landmark location name to replace this).

Click the  button next to “Add new ui_tab” field, and define “Smart Utility Poles” as the dashboard tab name.

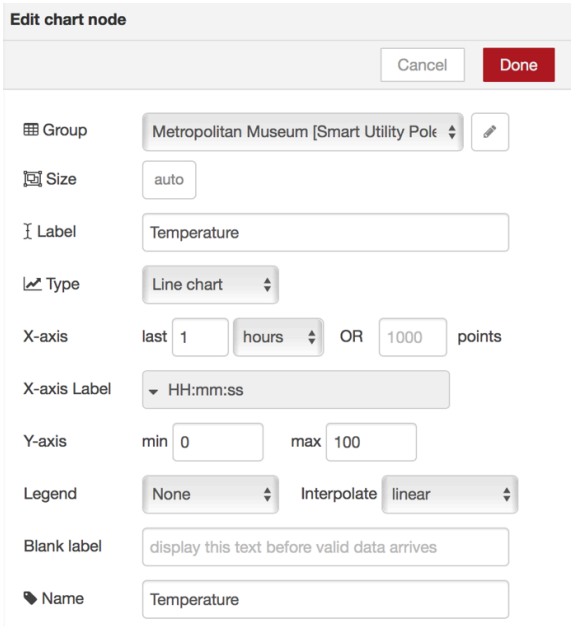




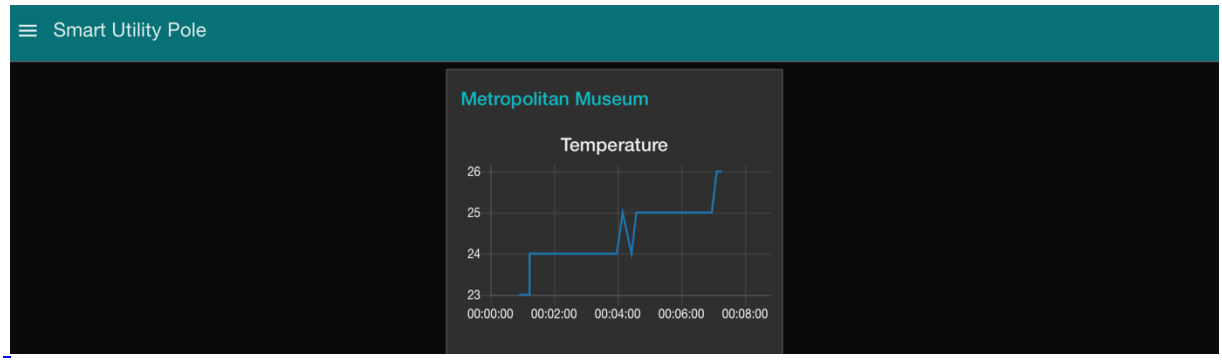
Click “Add” and this is what our final configuration looks like;



- In the Chart node Edit dialog, Change the Label to “Temperature”. Rename the node to “Temperature”.



In your browser, open http://<your_board_ip_address>:1880/ui. You can see streamed temperature values.



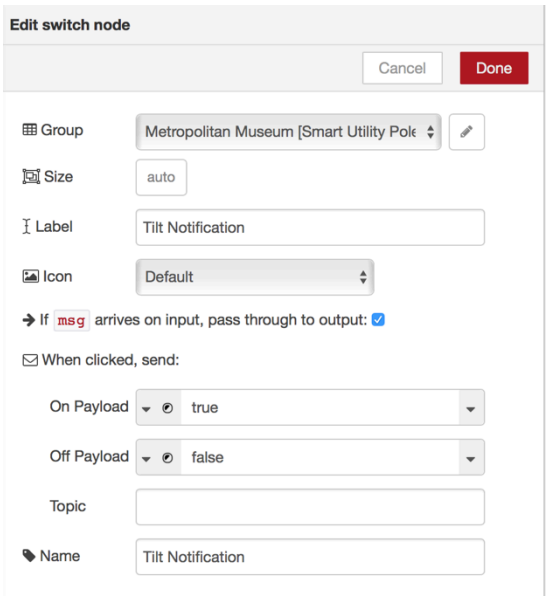
6.2 Display Tilt Notification on dashboard.

- Extend “CalculateTilt” function to support dashboard. Define *msg.payload* to true or false as a tilt indicator. The lines in blue are what have been added to the function.

```
//Calculate the angle of the inclination
if (global.get('tiltchange') === true) {
    var acc_x = msg.acc_x;
    var acc_y = msg.acc_y;
    var acc_z = msg.acc_z;

    //Normalize the accelerometer vector
    var normal = Math.sqrt(acc_x * acc_x + acc_y * acc_y + acc_z * acc_z);
    acc_x = acc_x / normal;
    acc_y = acc_y / normal;
    acc_z = acc_z / normal;

    global.set('tiltchange', false);

    //Calculate the inclination
    var inclination = Math.round(Math.acos(acc_z) * (180 / Math.PI));
    console.log("inclination", inclination);

    if (inclination > 155) {
      msg.payload = false;
      global.set('tilt', false);
      return msg;
    } else {
      msg.payload = true;
      global.set('tilt', true);
      return msg;
    }
}
```

- Drag a “switch” UI node 

as the output node of “CalculateTilt” function. Change “Label” and “Name” to “Tilt Notification”.

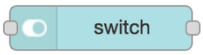


6.3 Display Earthquake Notification on dashboard

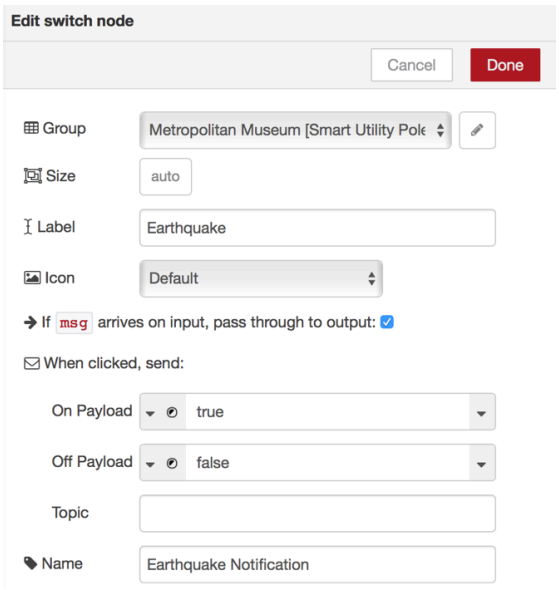
- Extend “DetectEarthquake” function to support dashboard. Define *msg.payload* to true as an earthquake

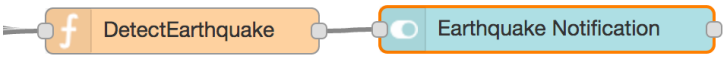
```
if (msg.noti_mask == 2 || msg.noti_mask == 3 || msg.noti_mask == 4) {
  global.set('earthquake',true);
  msg.payload = true;
}
return msg;
```

indicator. The lines in blue are what have been added to the function.

- Append a “switch” UI node 

as the output of “DetectEarthquake” function node. Rename it to “Earthquake Notification”.



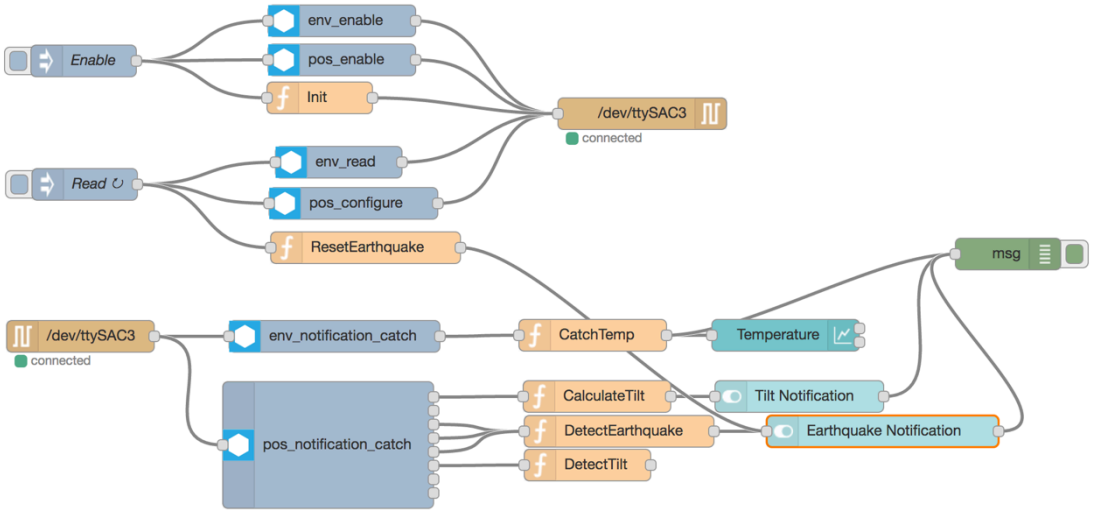


- Correspondingly, we need a function to reset earthquake alert. Drag a function node to the READ flow, and place it under “pos_configure” node. Rename the node to “ResetEarthquake”. The following code resets the earthquake status every time then a sensor reading is triggered.

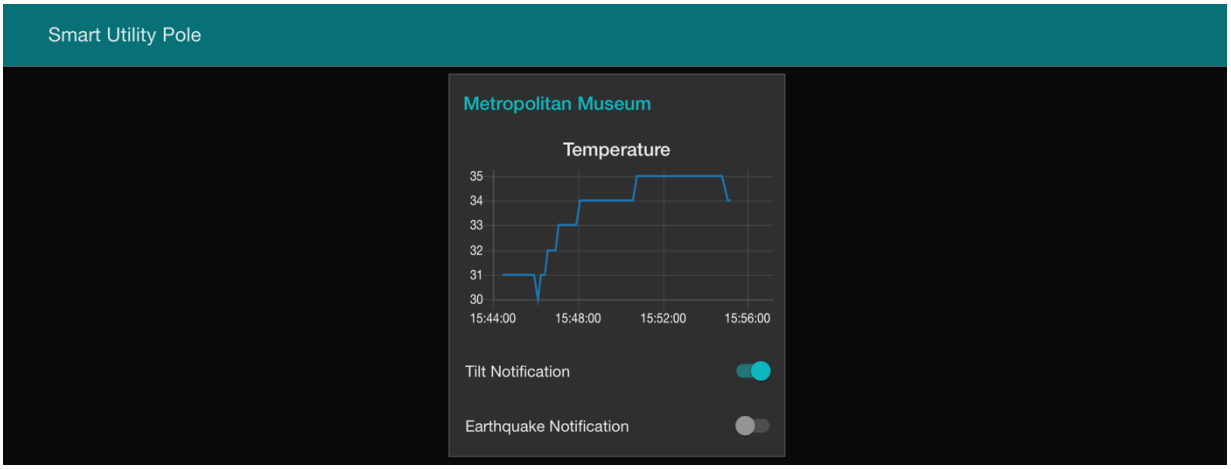
  Wire the output of “Read” node to “ResetEarthquake” node, and output of “ResetEarthquake” node to “Earthquake Notification”, so the UI widget can be updated accordingly.

```
global.set('earthquake', false);
msg.payload = false;
return msg;
```

Here is our final flow:



Our real-time sensor dashboard looks like below:

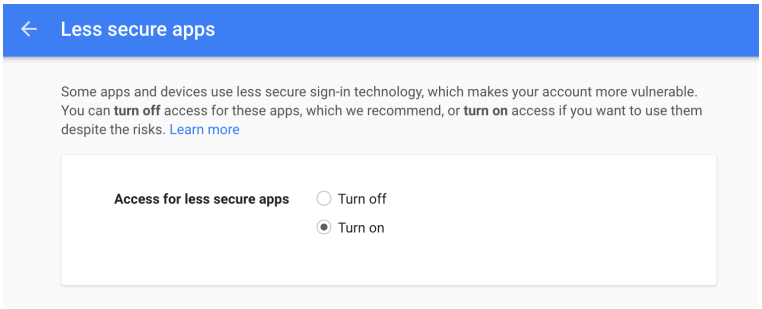


Exercise 2: Get email notification and show aggregate data on a map (Node-RED, MongoDB)

In exercise 2, we will trigger an email alert when the utility pole is tilted. We will also stream our sensor data to MongoDB, and show aggregated real-time data on a New York map.

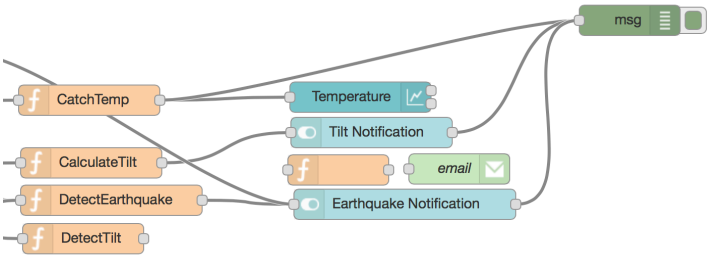
1. We will be using Gmail's SMTP service. For this workshop, turn on the "Access for less secure apps" setting within Gmail.

   Log into your Gmail account, and go to https://www.google.com/settings/security/lesssecureapps to enable the setting.

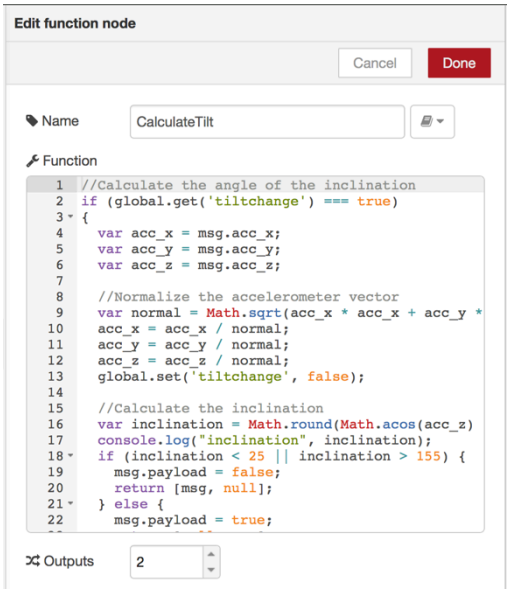


2. Trigger an email notification when the utility pole is tilted.

   2.1 Drag a "function" node and an "email" **out** node, and place them between "Tilt notification" node and "Earthquake notification" node.

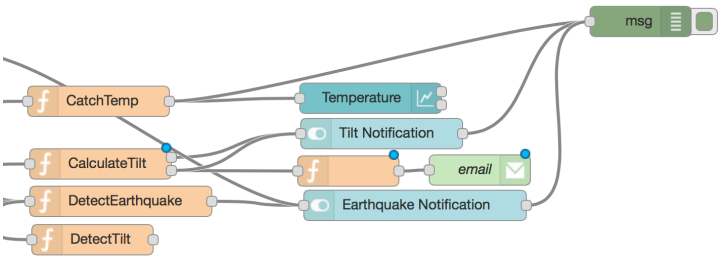


   2.2 In the "CalculateTilt" function node, change the return values from one parameter to two parameters. Also, update the number of outputs from 1 to 2. The lines in blue are what we have changed in the function.



```
//Calculate the angle of the inclination
if (global.get('tiltchange') === true)
{
  var acc_x = msg.acc_x;
  var acc_y = msg.acc_y;
  var acc_z = msg.acc_z;

  //Normalize the accelerometer vector
  var normal = Math.sqrt(acc_x * acc_x + acc_y * acc_y + acc_z * acc_z);
  acc_x = acc_x / normal;
  acc_y = acc_y / normal;
  acc_z = acc_z / normal;
  global.set('tiltchange', false);

  //Calculate the inclination
  var inclination = Math.round(Math.acos(acc_z) * (180 / Math.PI));
  console.log("inclination", inclination);
  if (inclination > 155) {
    msg.payload = false;
    global.set('tilt', false);
    return [msg, null];
  } else {
    msg.payload = true;
    global.set('tilt', true);
    return [null, msg];
}
```

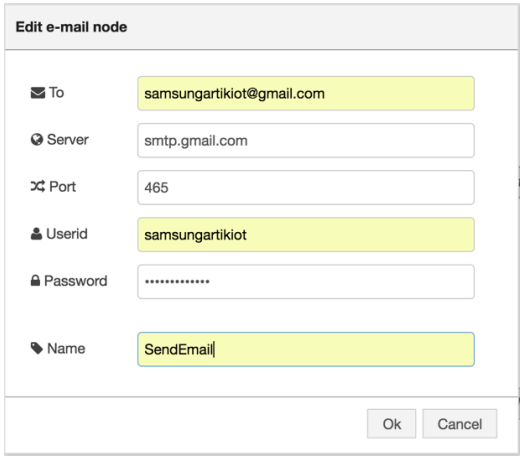
- Connect the 2nd output of “CalculateTilt” function to “Tilt Notification” node for UI update, also connect it to the newly added function node. Connect the output of the function node to the input of email node.



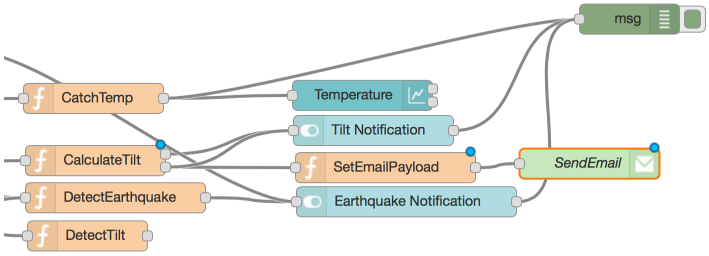
- In the “function” node, we define the email message in msg.payload, and Rename the node to “SetEmailPayload”.

```
msg.payload = 'Utility pole at ' + global.get('location') + ' is tilted';
return msg;
```

- In the “email” node: Enter your email address in the “To” field, email address user id in the “UserId” field, and email password in the “Password” field. Rename the node to “SendEmail”.



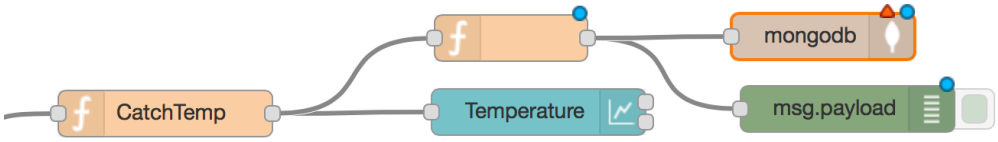
Here is what the final flow looks like:



2.3 Deploy your change, and you should be able to get an email alert when your utility pole is tilted.

3. Stream sensor data to MongoDB

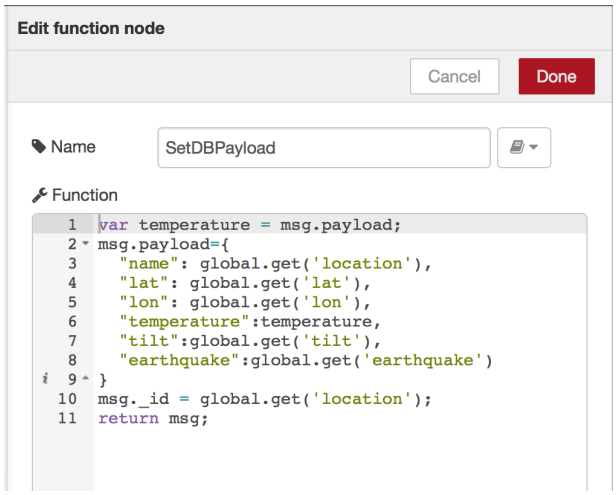
3.1 Place a “function” node and a “mongodb” **out** node above the “temperature” chart node. Connect the nodes as below.



3.2

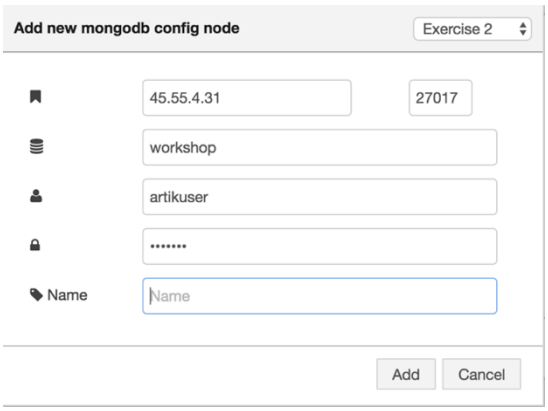
- In the function between “CatchTemp” and “mongodb” nodes, we define a payload that will be sent to MongoDB. Rename to “SetDBPayload”. Here is the code:

```
var temperature = msg.payload;

msg.payload={
    "name": global.get('location'),
    "lat": global.get('lat'),
    "lon": global.get('lon'),
    "temperature":temperature,
    "tilt":global.get('tilt'),
    "earthquake":global.get('earthquake')
}
msg._id = global.get('location');

return msg;
```

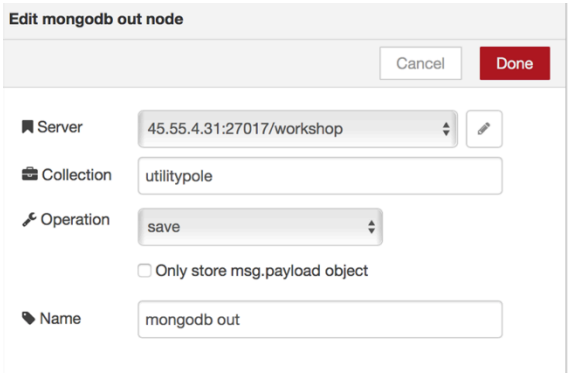


- "MongoDB" output node: Here, we need to configure our remote MongoDB host.

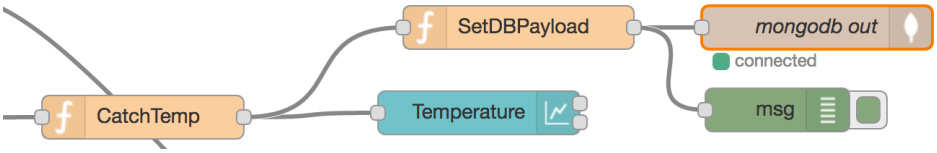
Click  in the "Edit mongodb out node" dialog, "Add new mongodb config node" opens up. Enter "45.55.4.31" as the host IP address, "27017" as the port number, and "workshop" as the database name. Use "artikuser" and "iot2017" as your username and password.



Click “Add” (or “Update” if changing existing parameters) to return to the “Edit mongodb out node” dialog. Enter “utilitypole” as the collection name. Rename to “mongodb out”.



This is what the flow looks like after deployment. Now, your real-time sensor data is being streamed to MongoDB.



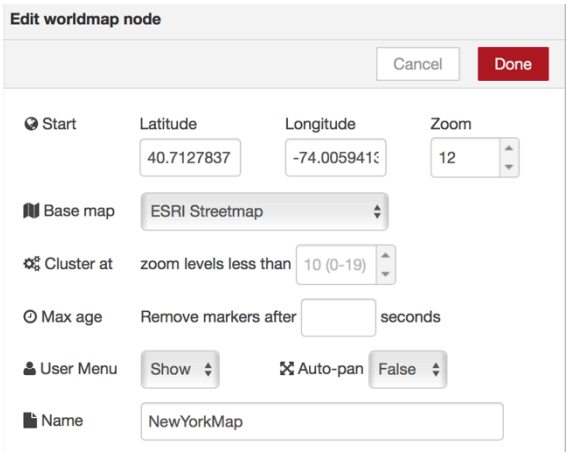
4. Display real-time aggregate data on a New York map.

3.1 Drag an “inject” node, a “function” node and a “world map” **out** node 

(“location” category) to the canvas. Wire them up. We will use this flow to initialize our New York map.

3.2 Configure each node by double clicking.

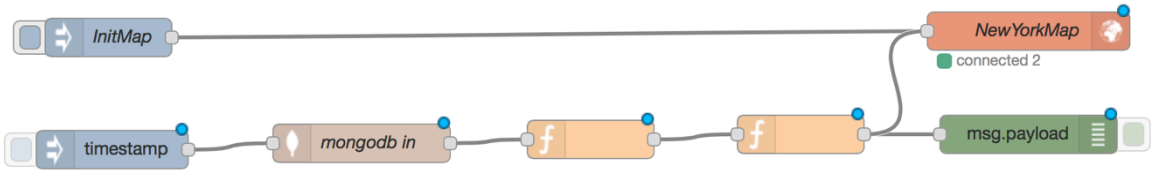
- In “timestamp” node, enable “Inject once at start” setting, and Rename the node to “InitMap”.
- In “world map” node, configure Latitude as 40.7127837, Longitude as -74.00594130000002, Zoom level as 12. Base map as “ESRI Streetmap”. Rename the node to “NewYorkMap”.



3.3 Deploy the flow. Launch http://<your_board_ip_address>:1880/worldmap, You will see a map of New York on your browser.

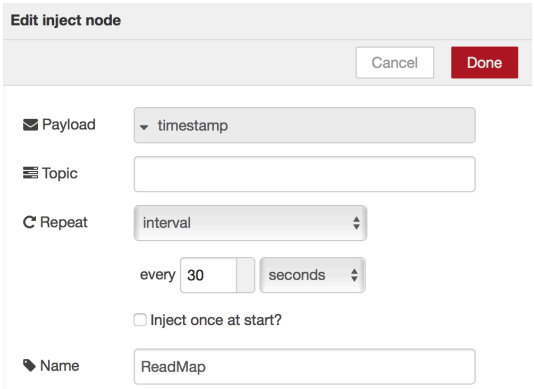
3.4 Drag an “inject” node, a “Mongodb” **in** node 

, two function nodes and a debug node to the canvas. Wire them up as below.

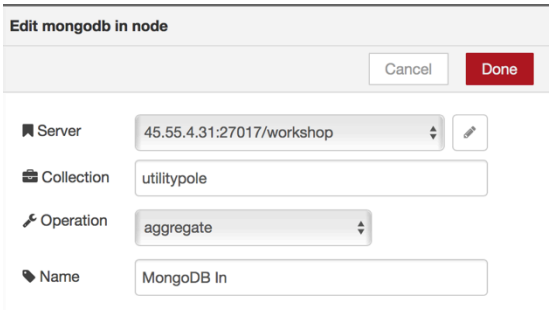


3.5 Configure each node by double clicking

- Rename the “inject(timestamp)” node to “ReadMap”. Set the interval to 30 seconds.



- In the “Mongodb” **in** node , it should have picked up server settings from the MongoDB out node configuration above. Add “utilitypole” as collection name, change the Operation to “aggregated”. Rename the node to “Mongodb In”.



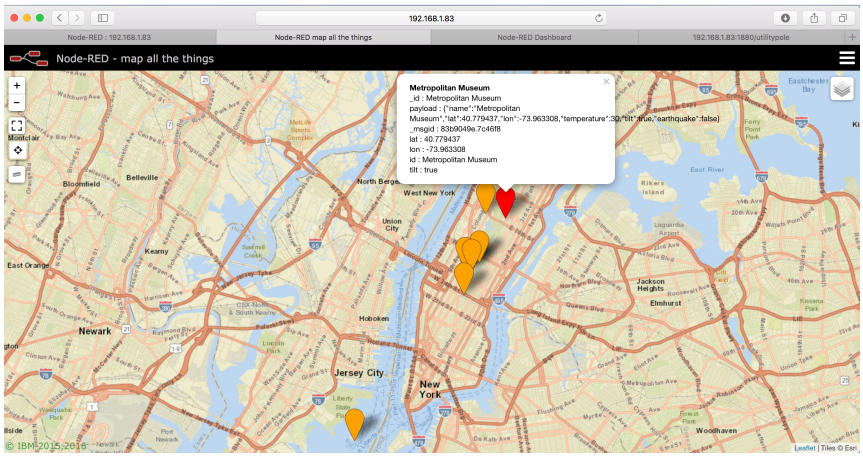
- In the first “function” node, we iterate through MongoDB. Here is the code you need to copy over. Rename to “IterateDB”.

```
var mapResult = {
 countdown: msg.payload.length,
 result: msg.payload.map(function(){})
};
return [msg.payload.map(function(element, index) {
 return {
  payload: element,
  index: index,
  mapResult: function() {
   return mapResult;
  }
 };
})];
```

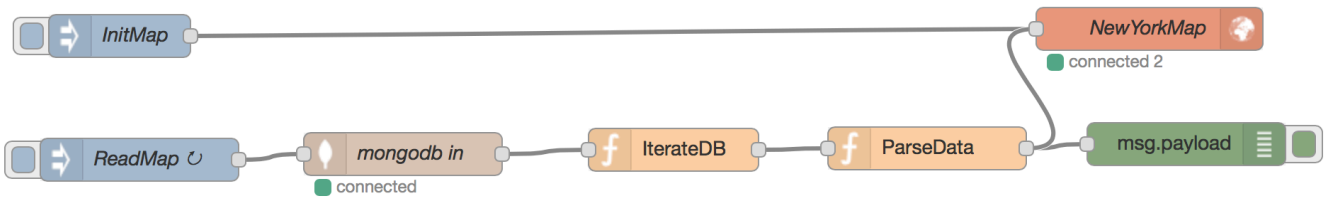
- In the second “function” node, we define payload of map markers. Rename the node to “ParseData”.

```
msg.payload.lat = msg.payload.payload.lat;
msg.payload.lon = msg.payload.payload.lon;
msg.payload.name = msg.payload.payload.name;
msg.payload.id = msg.payload.payload.name;
msg.payload.tilt = msg.payload.payload.tilt;
msg.payload.icon = "globle";
if (msg.payload.payload.tilt === true)
  msg.payload.iconColor = "Red";
else
  msg.payload.iconColor = "Orange";
return msg;
```

Now, deploy your changes. Launch <your_ARTIK_IP_address>:1880/worldmap and you will be able to see the real-time sensor data from all of our utility poles. If a pole is tilted, its marker will turn red.



Here is what our final flow looks like:



Exercise 3: Build a high availability utility pole network (MQTT & Node-RED)

In this exercise, we are going to use MQTT to monitor the heartbeats of utility poles. Each MQTT client sends heartbeat information (its location) to the MQTT broker. If the MQTT broker doesn't receive a heartbeat from a MQTT client for a pre-defined period of time, it publishes a MQTT message indicating that a utility pole is inactive.

1. Select one device as MQTT broker in each group. Launch MQTT broker 'mosquitto' on the board, and let your group members know your board's IP address.

```
#mosquitto –d
```

2. Extend "Init" function in ENABLE flow. The lines in blue are what have been added to the function.

```
global.set('location', "Metropolitan Museum"); //Replace "Metropolitan Museum" with your own landmark location
global.set('lat', 40.779437); //Replace lat value with your own latitude
global.set('lon', -73.963308); //Replace lon value with your own longitude
global.set('tilt', false);
global.set('tiltchange', false);
global.set('earthquake', false);
global.set('map', context.global.d3.map());
context.global.d3.map().clear();
```

3. MQTT Clients, please follow the instructions below.
MQTT Brokers, please skip to step 4.

2.1 Publish MQTT heartbeat messages to MQTT brokers.

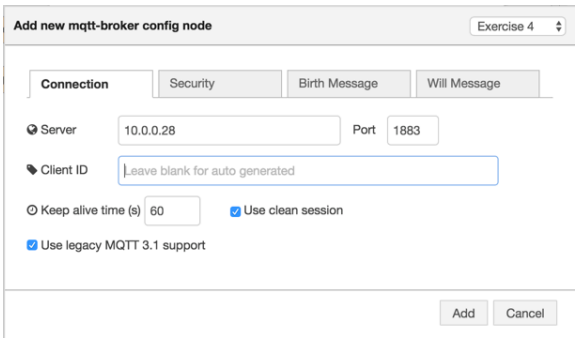
- Drag an "inject" node, a "function" node and an mqtt **out** node 

to the canvas. Wire them up.

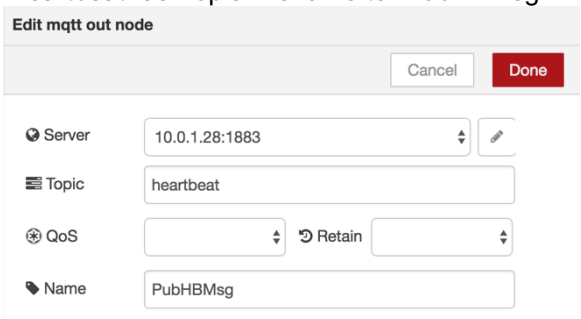
- In the "timestamp" node, configure Repeat Interval to 10 seconds so that your MQTT client will send heartbeat messages to the MQTT broker every 10 seconds. Rename to "Heartbeat".
- In the "function" node, we simply define our *msg.payload* as current locaion. Rename to "SetHBMsg".

```
msg.payload = global.get('location');
return msg;
```

- In the "mqtt" out node, click on the  button in the "Edit mqtt out node" dialog, "Add new mqtt-broker config node" opens up. Enter your broker's IP address and use default Port number 1883.

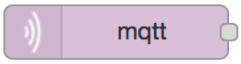


Click “Add” (or “Update” if changing existing parameters) to return to the “Edit mqtt out node” dialog. Enter your “heartbeat” as Topic. Rename to “PubHBMsg”.

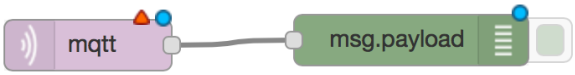


This is how MQTT’s client looks like right now:

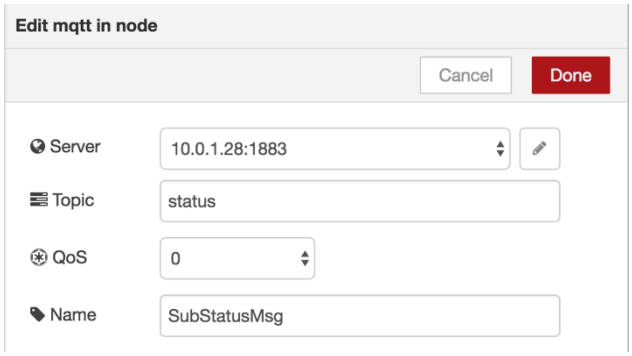
2.2 Subscribe to MQTT status message:

- Drag an mqtt **in** node 

and a debug node to the canvas. Wire them up.



- In the“mqtt” node, enter “status” as topic name and change the QoS level to 0. Rename to “SubStatusMsg”.



2.3 To unregister a MQTT client from MQTT broker, we need to create a flow similar to the one in 2.1.

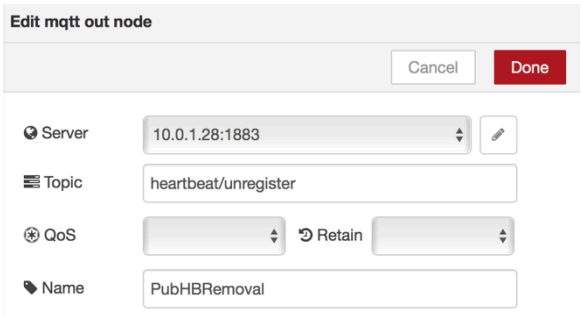
- Drag an “inject” node, a “function” node and an mqtt **out** node 

to the canvas. Wire them up.

- We use the default settings in the “timestamp” node,
- In the “function” node, we simply define our *msg.payload* as current locaion. Rename to “RemoveHBMsg”.

```
msg.payload = global.get('location');
return msg;
```

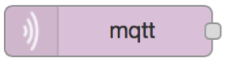
- In the “mqtt” **out** node, enter “heartbeat/unregister” as Topic. Rename to “PubHBRemoval”.



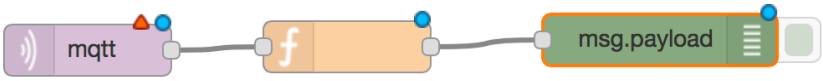
This is how MQTT’s client looks like right now:

4. MQTT Brokers, please follow the instructions below.
MQTT Clients, please STOP here.

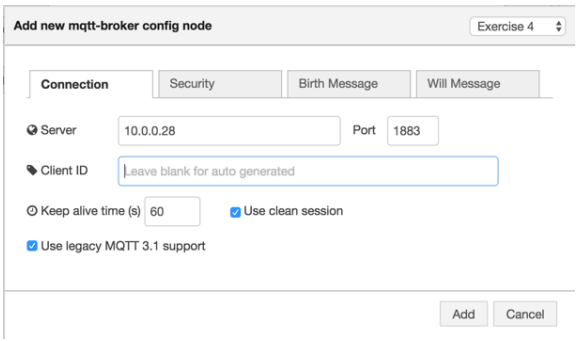
4.1 Listen to heartbeat messages from MQTT clients

- Drag an mqtt **in** node 

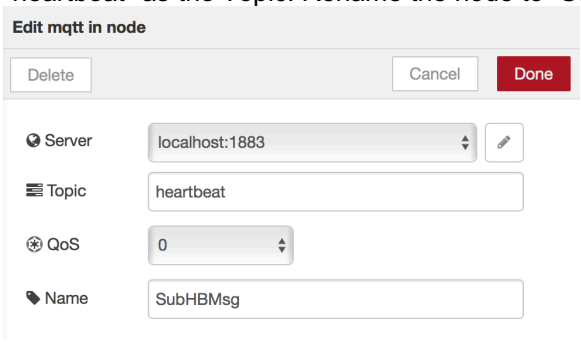
, a “function” node and a “debug” node to the canvas. Wire them up.



- Double click mqtt **in** node, click  button in the “Edit mqtt out node” dialog, “Add new mqtt-broker config node” opens up. Enter your local IP address (or use “localhost”) and use default Port number 1883.



Click “Add” (or “Update” if changing existing parameters) to return to the “Edit mqtt out node” dialog. Enter your “heartbeat” as the Topic. Rename the node to “SubHBMsg”.



- In the “function” node, we use key/value pairs to track if the broker receives heartbeats from clients.

  When a heartbeat is received from a client, a key/value pair is created. The key is the location of the client and the value is the number of heartbeats the broker has received. When we receive a heartbeat message from a new location, we initialize its heartbeat value as 1. Every heartbeat message received thereafter increases the value by 1.

  A timer callback function is invoked every 20 seconds. In the callback function, we iterate through the key/value pairs to see if any entry has a heartbeat value of zero. If so, this means that no heartbeat messages were received from that location within the past 20 seconds. We consider such locations as inactive.

  Rename the node to “MonitorHB”.

```
var mqttClient = msg.payload;
var mqttBroker = global.get('map');
var timerId = context.get('timerId') || 0;
var str = context.get('str') || '';

//Create  or update key/value pairs of mqtt clients
//key - location of the mqtt client
//value - number of heartbeats received
if (mqttBroker.has(mqttClient) === true) {
  var count = mqttBroker.get(mqttClient) + 1;
  mqttBroker.set(mqttClient, count);
} else {
  mqttBroker.set(mqttClient, 1);
}

//timer callback, invoked every 20 seconds
//Iterate through mqtt client key/value pairs, and look for entries with 0
heartbeats
function timer() {
  var size = mqttBroker.size();

  for (var i = 0; i < mqttBroker.size(); i++) {
    key = mqttBroker.entries()[i].key;
    value = mqttBroker.entries()[i].value;
    if (value === 0)
      str = str + key + " is inactive! ";
    mqttBroker.set(key, 0);
  }
  global.set('pubMsg', str);
  str = '';
  context.set('str', '');

  timerId = setTimeout(timer, 20000);
  context.set('timerId', timerId);
}

//Set up timer the first time when the function is invoked
if (timerId === 0) {
  timerId = setTimeout(timer, 20000);
  context.set('timerId', timerId);
}

return msg;
```

Here is the flow:

4.2 Publish Status messages to clients

- Drag an “inject” node, a “function” node and an mqtt **out** node 

to the canvas. Wire them up.

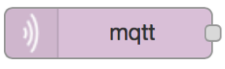
- In the “timestamp” node, configure Repeat Interval to 10 seconds so that your MQTT broker publishs status messages to MQTT clients every 10 seconds. Rename to “Status”.
- In the “function” node, we define our MQTT status message for publication. Rename to “SetStatusMsg”.

```
var pubMsg = global.get('pubMsg') || "";
var mqttBroker = global.get('map');

if (typeof(pubMsg) !== 'undefined' && pubMsg !== '')
  msg = {payload: pubMsg};
else if (mqttBroker.size() === 0) {
  global.set('pubMsg', "");
  msg = {payload: "No registered poles yet"};
} else {
  global.set('pubMsg', "");
  msg = {payload: "All poles are alive"};
}
```

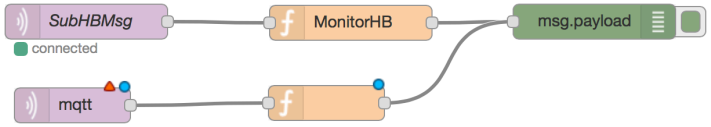
- In the “mqtt” out node, click on the  button in the “Edit mqtt out node” dialog, Enter “status” for the Topic. Rename to “PubStatusMsg”.

Here is an example of status message the broker publishes:

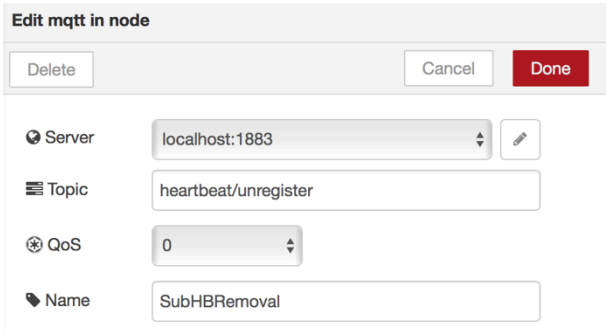
4.3 To listen to heartbeat unregister message, brokers need to add a flow similar to the one in 4.1

- Drag an mqtt **in** node 

, and a “function” node to the canvas. Wire them up and connect the output of “function” node to the Debug node in the SubHBMsg flow.



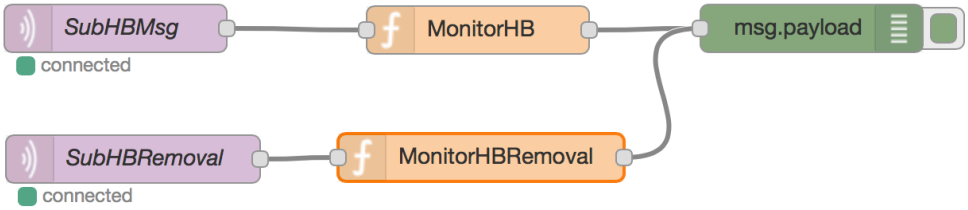
- In mqtt **in** node, Enter “heartbeat/unregister” as the Topic. Rename the node to “SubHBRemoval”.



- In the “function” node, we remove the key/value pair for this location. Remove the node to “MonitorHBRemoval”.

```
var mqttClient = msg.payload;
var mqttBroker = global.get('map');

if (mqttBroker.has(mqttClient) === true)
   mqttBroker.remove(mqttClient);

return msg;
```

Here is the flow:



Exercise 4: Use ARTIK SDK to establish a ZigBee network (ARTIK SDK)

In this exercise, we are going to use ARTIK SDK to establish a ZigBee network.One device on each table acts as a ZigBee coordinators(at different channels) and other devices will act as ZigBee routers.

1 . Add ZigBee devices

1.1 ZigBee coordinators, please follow the instructions in 1.1.
ZigBee routers, please skip to step 1.2.

```
[root@localhost ~]# cd /root
[root@localhost ~]# gcc artik_zigbee_coordinator.c artik_zigbee_test_common.c -o
artik_zigbee_coordinator -I/usr/include/artik/base -I/usr/include/artik/zigbee –
L/usr/lib –lartik-sdk-zigbee -lartik-sdk-base
```

artik_zigbee_coordinator is generated at this point. Launch it from command line,

```
[root@localhost ~]# ./artik_zigbee_coordinator
======================
1: On/Off Switch
2: Level Control Switch
3: On/Off Light
4: Dimmable Light
5: Light Sensor
6: Remote Control
0: ADD DEVICE DONE
Add device: 1
Set endpoint id (1):  (Press <Enter> to use the default value )
Added device "On/Off Switch" with ep(1)

Add device: 0
```

You will see a menu as shown above. Enter **1** to add an On/Off Switch, press <Enter> to accept the default endpoint id. Once “On/Off Switch” device is added, enter **0** to finish adding devices.

1.2 **ZigBee coordinators, please stop here.**

```
[root@localhost ~]# cd /root
[root@localhost ~]# gcc artik_zigbee_router.c artik_zigbee_test_common.c -o
artik_zigbee_router -I/usr/include/artik/base -I/usr/include/artik/zigbee –
L/usr/lib –lartik-sdk-zigbee -lartik-sdk-base
```

Launch *artik_zigbee_router* from command line,

```
[root@localhost ~]# ./artik_zigbee_router
======================
1: On/Off Light
0: ADD DEVICE DONE
Add device: 1
Set endpoint id (19): (Press <Enter> to use the default value)
Added device "On/Off Light" with ep(19)

Add device: 0
```

Enter 1 to add an On/Off Light device, press <Enter> to accept the default endpoint id. Once “On/Off Switch” device is added, enter 0 to finish adding devices.

2. Manually form a ZigBee network **(Only ZigBee coordinators need to do this. Routers please wait coordinators’ permission to join a network, then go to step 3.2)**
From 1.1 above, when coordinators hit <Enter> after adding device, coordinator will see the menu below. Initial state shows ZIGBEE_NO_NETWORK. Enter 2 to manually form a ZigBee network.

```
Add device: 0
State: ZIGBEE_NO_NETWORK
Network: Non Exist
======================
1: Form network
2: Form network (advance)
3: Join network
4: Join network (advance)
5: Leave network
6: Get network state
7: Discover device
8: Start testing without network
9: Network permit join
10: Network find and join
11: Network stop scan
e: Exit (Quit with calling clean and reset device)
q: Exit
Please select operation:    2
Preferred channel (11, 14, 15, 19, 20, 24, 25)
Set channel(25): 11  (Please use the channel number that is assigned to your table)
Preferred TX (-9 ~ 8)
Set TX(2):  (Press <Enter> to use the default TX value)
Set PAN ID(0x1234):  (Press <Enter> to use the default PAN ID value)
Form network channel(11) TX(2) PAN ID(0x1234):
Manually form network success
Done
```

In case coordinators see error message like below, please press 5 to leave the network, and press 2 to manually form the network again.

```
03-24 16:16:53.651  2558  2558 E <network.c:61> NULL or Wrong response type
03-24 16:16:53.652  2558  2558 E <artik_zigbee_coordinator.c:1540> Manually form network failed: No zigbee
message
```

3. Join ZigBee network
3.1 **ZigBee coordinators, please follow the instructions in 3.1.**
ZigBee routers, please skip to step 3.2.

Coordinators can press <Enter> at anytime to show the menu.

```
======================
State: ZIGBEE_JOINED_NETWORK
Type: ZIGBEE_COORDINATOR
======================
1: Form network
2: Form network (advance)
3: Join network
4: Join network (advance)
5: Leave network
6: Get network state
7: Discover device
8: Start testing
9: Network permit join
10: Network find and join
11: Network stop scan
e: Exit (Quit with calling clean and reset device)
q: Exit
Please select operation: 9
Permit join for 60 seconds
Done
```

Enter **9** to permit join a network, after this, all routers have up to **60 seconds** to join the newly formed ZigBee networks.

3.2 **ZigBee routers only**

Routers can press <Enter> anytime to show the menu.

Enter **1** to join a network, then follow the instruction below. In case the network is not formed successfully, immediately press **2** to leave the network, and press **1** to re-join. You only have **60 seconds** to join the network unless coordinators grant permissions again.

```
State: ZIGBEE_NO_NETWORK
Network: Non Exist
======================
1: Join network (advance)
2: Leave network
3: Get network state
e: Exit (Quit with calling clean and reset device)
q: Exit
Please select operation: 1
 Set channel(25): 11    (Please use the channel number that is assigned to your table)
Preferred TX (-9 ~ 8)
Set TX(2): (Press <Enter> to use the default TX value)
Set PAN ID(0x1234): (Press <Enter> to use the default PAN ID value)
Join network channel(11) TX(2) PAN ID(0x1234):
Device discovering ...
03-12 20:13:10.149  2699  2699 D <artik_zigbee_router.c:1849> In callback, response type : 3403
03-12 20:13:10.150  2699  2699 D <artik_zigbee_router.c:2256> [testzigbee] callback end

03-12 20:13:10.150  2699  2699 D <artik_zigbee_router.c:1849> In callback, response type : 3309
Device discovery start
03-12 20:13:10.150  2699  2699 D <artik_zigbee_router.c:2256> [testzigbee] callback end

03-12 20:13:11.875  2699  2699 D <artik_zigbee_router.c:1849> In callback, response type : 3403
03-12 20:13:11.876  2699  2699 D <artik_zigbee_router.c:2256> [testzigbee] callback end

03-12 20:13:11.878  2699  2699 D <artik_zigbee_router.c:1849> In callback, response type : 3309
Device discovery found:
Device id:0x0000 eui:0xB05C220B006F0D00
Endpoint 1
Cluster id 0x0000, SERVER
Cluster id 0x0003, SERVER
Cluster id 0x0003, CLIENT
Cluster id 0x0006, CLIENT
03-12 20:13:11.880  2699  2699 D <artik_zigbee_router.c:2256> [testzigbee] callback end

03-12 20:13:11.888  2699  2699 D <artik_zigbee_router.c:1849> In callback, response type : 3309
Done
03-12 20:13:11.889  2699  2699 D <artik_zigbee_router.c:2256> [testzigbee] callback end
```

3.3 (ZigBee coordinators only) If the network is successfully formed, coordinators will see callback messages below.

```
03-12 20:13:10.125  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3307
NETWORK_NOTIFICATION: JOIN
Device discovering ...
03-12 20:13:10.138  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee] callback end

03-12 20:13:10.147  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3403
03-12 20:13:10.147  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee] callback end

03-12 20:13:10.148  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3309
Device discovery start
03-12 20:13:10.148  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee] callback end

03-12 20:13:10.295  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3403
03-12 20:13:10.296  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee]   callback end

03-12 20:13:10.297  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3309
Device discovery found:
Device id:0x87B5 eui:0x1F4A220B006F0D00
Endpoint 19
Cluster id 0x0000, SERVER
Cluster id 0x0003, SERVER
Cluster id 0x0004, SERVER
Cluster id 0x0005, SERVER
Cluster id 0x0006, SERVER
03-12 20:13:10.299  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee] callback end

03-12 20:13:10.301  3244  3244 D <~rtik_zigbee_coordinator.c:1886> In callback, response type : 3309
Done
03-12 20:13:10.305  3244  3244 D <~rtik_zigbee_coordinator.c:2293> [testzigbee] callback end
```

4. Test ZigBee network.**(ZigBee Coordinators Only)**
After the ZigBee network is formed, **coordinators** choose 8 to test the network. Then select 4 to turn and off ZigBee

```
======================
State: ZIGBEE_JOINED_NETWORK
Type: ZIGBEE_COORDINATOR
======================
1: Form network
2: Form network (advance)
3: Join network
4: Join network (advance)
5: Leave network
6: Get network state
7: Discover device
8: Start testing
9: Network permit join
10: Network find and join
11: Network stop scan
e: Exit (Quit with calling clean and reset device)
q: Exit
Please select operation: 8
======================
1: Get device information
2: Auto test
3: Identify
4: On/off
5: Level control
6: Illuminance measurement
7: Remote Control
8: Commissioning target start
9: Commissioning target stop
10: Commissioning initiator start
11: Commissioning initiator stop
0: Setup network
e: Exit (Quit with calling clean and reset device)
q: Exit
Please select operation: 4
```

devices.

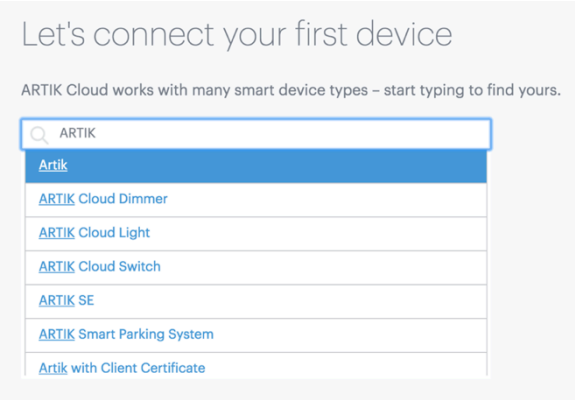
At this point, routers should see the LEDs on their Kitra520 turned on by coordinators. Press 4 again will turn off the LEDs on all routers.

Exercise 5: Connect to ARTIK Cloud and trigger cross-device action by using Rules Engine (ARTIK Cloud, Node-RED)

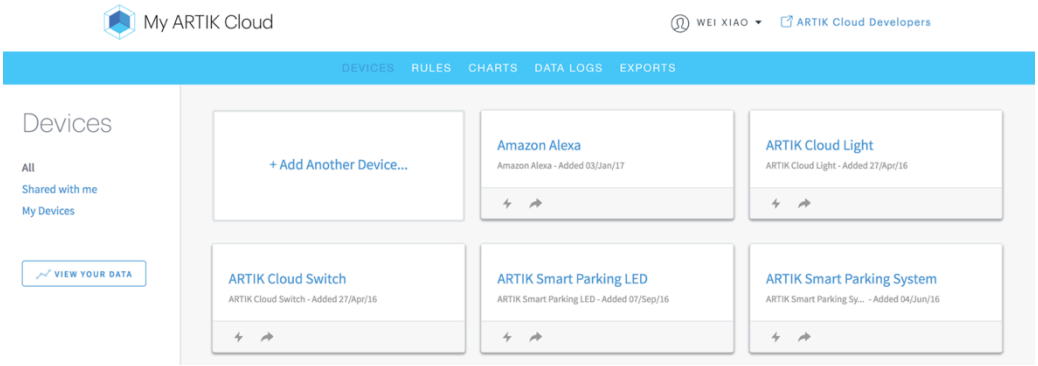
In this exercise, we are going to stream utility pole sensor data to ARTIK Cloud.

1. Log onto the ARTIK Cloud user portal https://artik.cloud.

1.1 If this is your first device, you will be re-directed to the screen below:

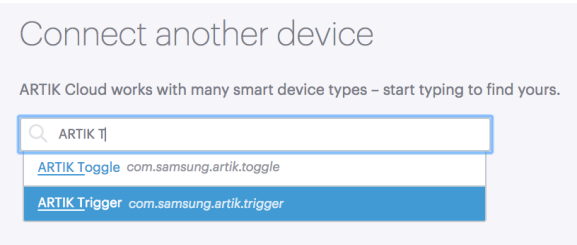


Otherwise, under “MY ARTIK CLOUD”/”DEVICES”, you will see all of your connected devices.

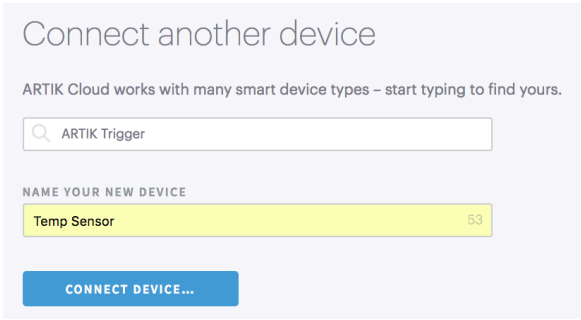


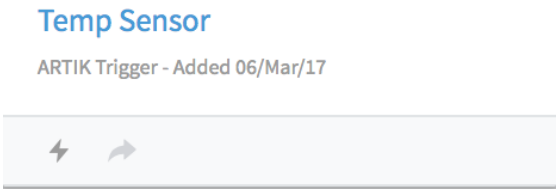
Click on the “Add Another Device…” button.

1.2 Search for “ARTIK Trigger” and select it. “ARTIK Trigger” is a public device type, which includes “temperature” attribute.

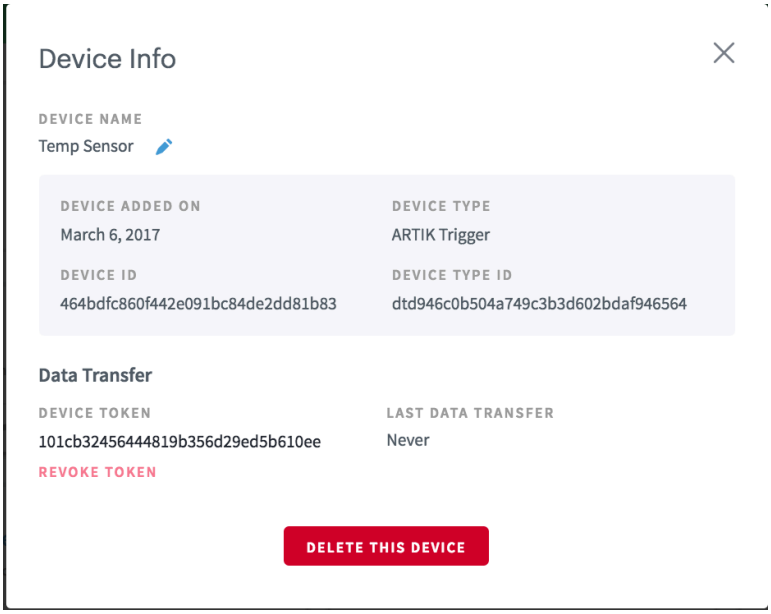


1.3 Change the device name to “Temp Sensor”, then click the “CONNECT DEVICE…” button. A new device will be created.

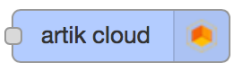




1.4 Click the “Temp Sensor” device name. A Device Info modal will popup which contains your Device Type, Device ID, Device name, and additional details. Click the “GENERATE DEVICE TOKEN…” link to generate a device token.



We need to use the **Device ID** (not “**DEVICE TYPE ID**”) and **Device Token** to associate your temperature sensor with ARTIK Cloud. Make a copy or leave the window open.

2. Drag an ARTIK Cloud **out** node 

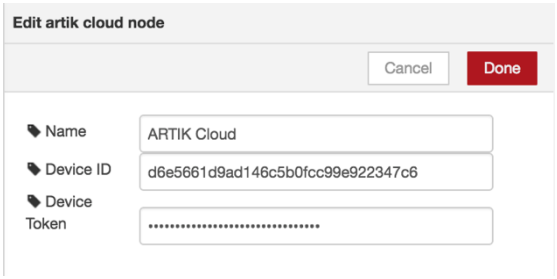
to the canvas, and place it between “SetDBPayload” and “Temperature” nodes.

2.1 Update the “CatchTemp” function to define a message payload for ARTIK Cloud.

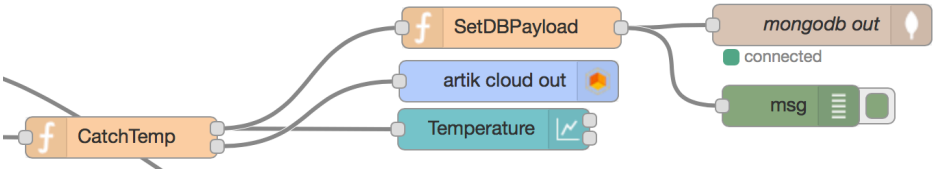
```
var temperature = msg.temperature;
var dbMsg = {payload: temperature};
var cloudMsg = {payload: {"temperature":temperature}};
return [dbMsg, cloudMsg];
```

Increase “CatchTemp” function’s number of outputs to 2.

2.2 Double click the ARTIK Cloud **out** node and enter the Device ID and Token in the Edit Window, Rename to “ARTIK Cloud”.

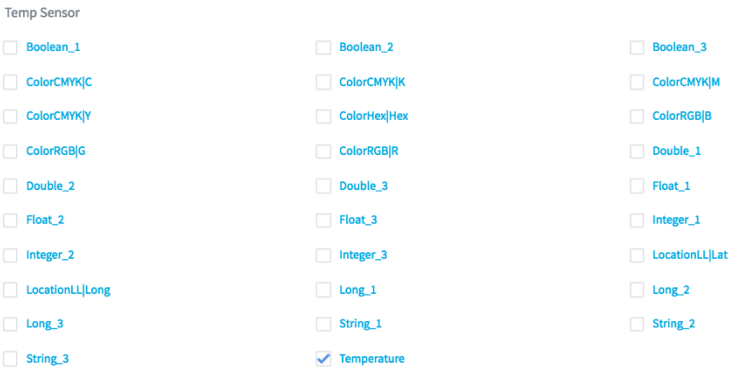


2.3 Connect the 2nd output of ARTIK Cloud **out** node to ARTIK Cloud node.
This is what your final flow should look like:

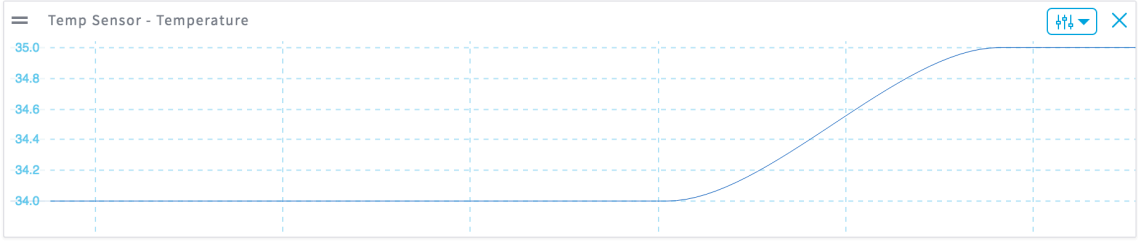


3. Go to the ARTIK Cloud user portal, click 

button, then click the “+/- CHARTS” button on the upper left side. Click on the “Temperature” checkbox to view the utility pole’s temperature.



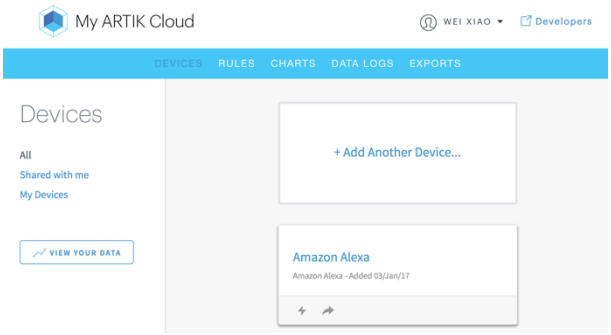
You will see the streamed temperature in your portal:



Now, let’s explore an advanced feature within ARTIK Cloud: the Rules Engine. We will define a rule in ARTIK Cloud that monitors your temperature sensor data. If your temperature is higher than a threshold value, the rules engine will turn on the LED on your Kitra board.

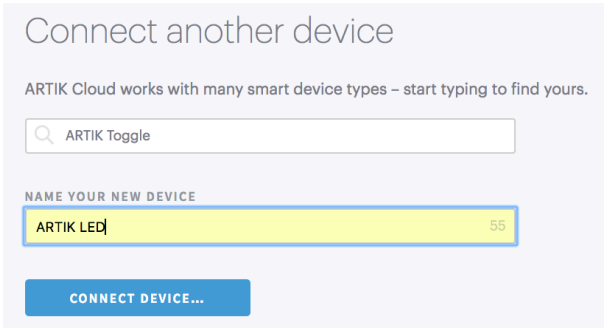
1. Create an LED device in the ARTIK Cloud user portal.

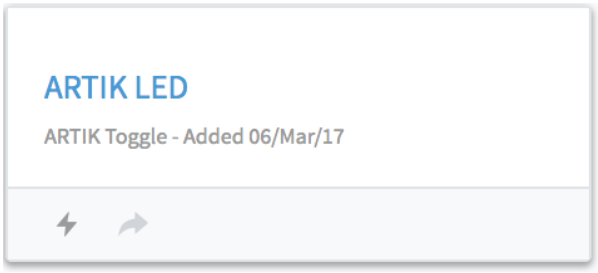
   1.1 From ARTIK Cloud user portal https://artik.cloud, go to “MY ARTIK CLOUD”/”DEVICES”, click “Add Another Device…” link.



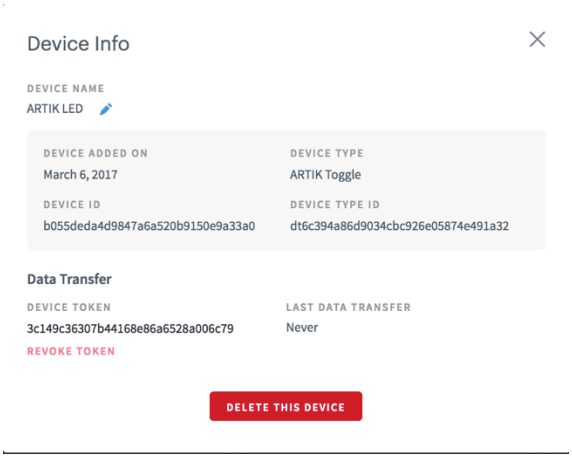
   1.2 Search for “ARTIK Toggle” and select it. This is the public device type that we created for this workshop. The device type includes one attribute “toggle”, which is a Boolean type. Its valid value is either true or false, to represent if the LED is ON or OFF.

   1.3 Change the device name to “ARTIK LED”, then click the “CONNECT DEVICE…” button. A new device will be created.





   1.4 Click on the device name, you will see the Device Info popup, which shows your Device Type, Device ID, Device name, and additional details. Click the “GENERATE DEVICE TOKEN…” link to generate a device token.

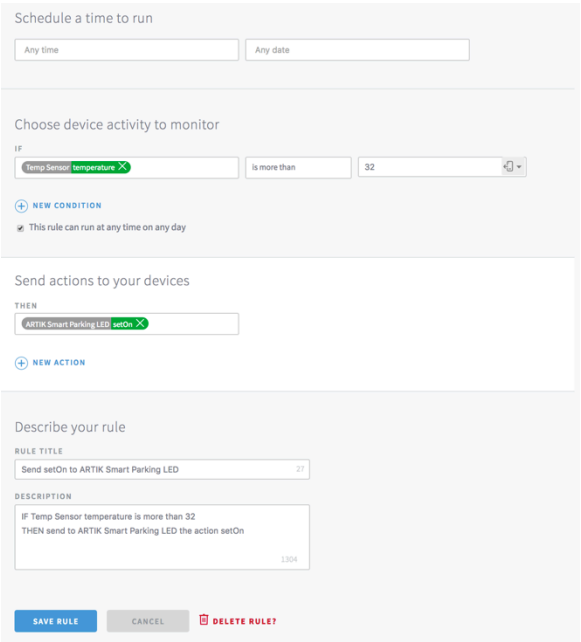


2. Add Rules to our Rules Engine. By doing this, we will be able to connect our “Temp Sensor” device and “ARTIK LED” device, and trigger cross-device actions.

2.1 In the ARTIK user portal, go to “RULES”, and click the “CREATE A NEW RULE” button. The first rule we will add is to turn on the LED when temperature is higher than 32 degrees. You can create the rule by using plain English.

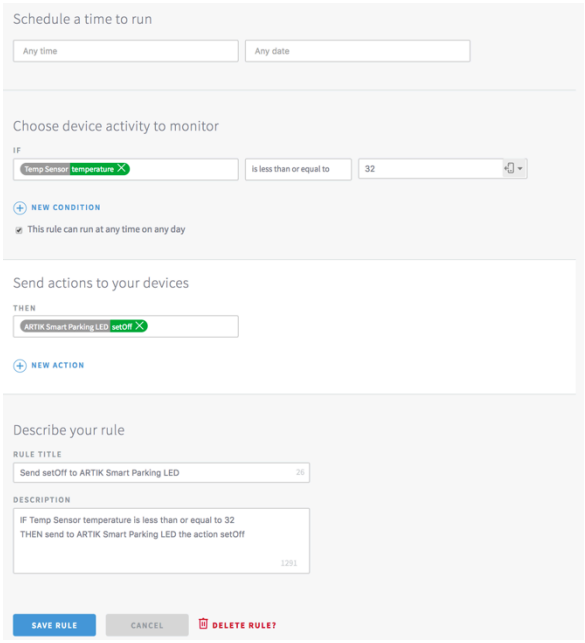
IF
“Temp Sensor” temperature is more than **32**
THEN
“ARTIK LED” setOn

Click the “SAVE RULE” button.



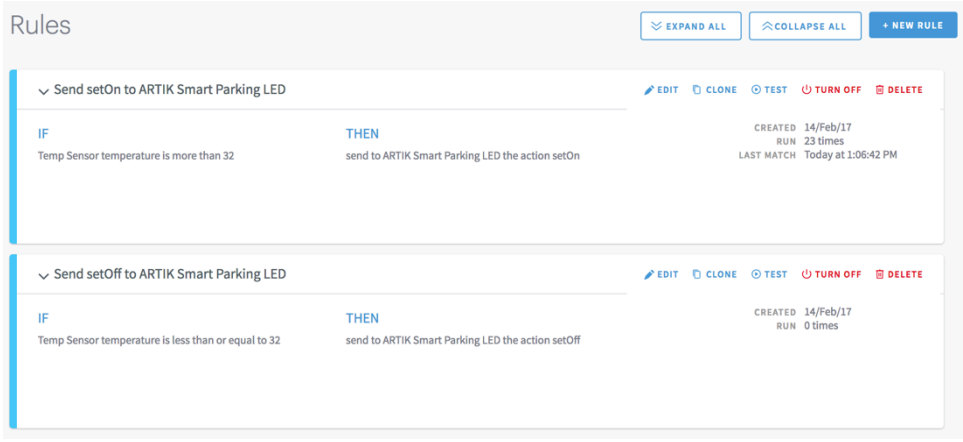
2.2 Repeat the above step, and create a second rule to turn off the LED when the reservation is cancelled or expired.

IF
“Temp Sensor” temperature is less than or equal to **32**
THEN
“ARTIK LED” setOff

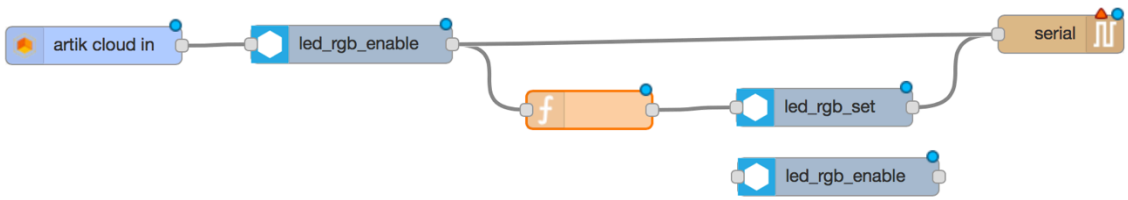


Click the “SAVE RULE” button.

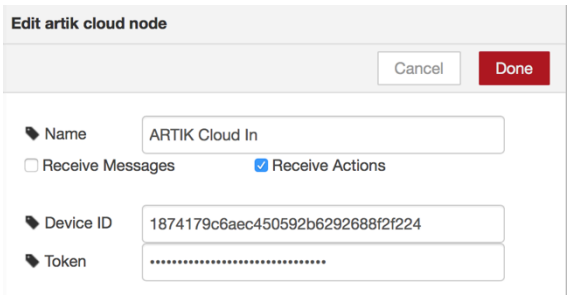
2.3 Now, we have 2 rules defined:



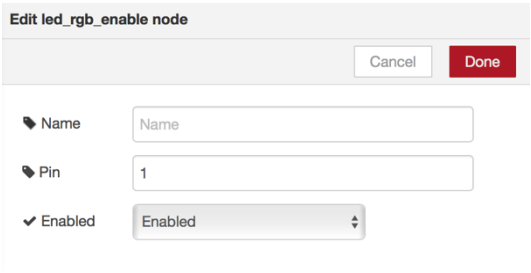
3. Listen for the triggered action.
   - Drag an “artik cloud” **in** node, a “led_rgb_enable” node (kitra_output category), a “function” node, a “led_rgb_set ” node(kitra_output category), a “led_rgb_enable” node and a serial **out** node to the canvas. Wire them up as below.



   - In “artik cloud” in node, select “Enable Actions”, enter your ARTIK LED device’s device ID/Token. Rename the node as “ARTIK Cloud In”.



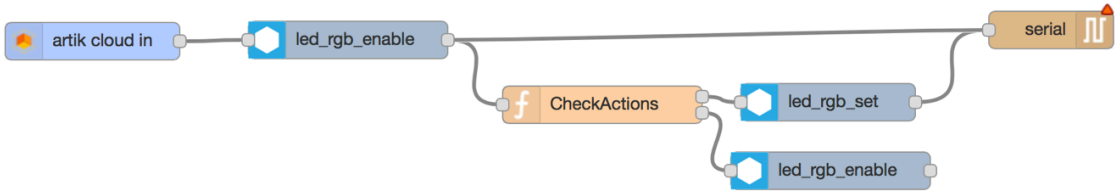
- Double click the first “led_rgb_enable” node to enable the LED.You can use the default setting or set another Pin#. There are 4 LEDs on Kitra board. Acceptable Pin numbers are 1 to 4.



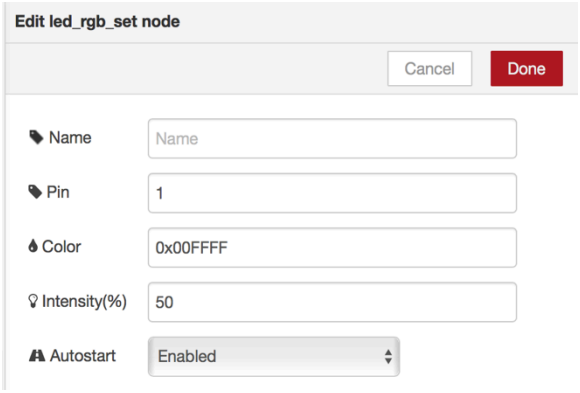
- In the “function” node, increase the number of outputs to 2. We listen for incoming actions: When the incoming action is “setOn”, we go to the 1st output to turn on an LED. If the action is “setOff”, we go to the 2nd output to disable the LED. Rename the node to “CheckActions”.

```
var actions = msg.actions;
var action = actions[0].name;

if (action === 'setOn') {
    return [msg, null];
} else if (action === 'setOff') {
    return [null, msg];
}
```

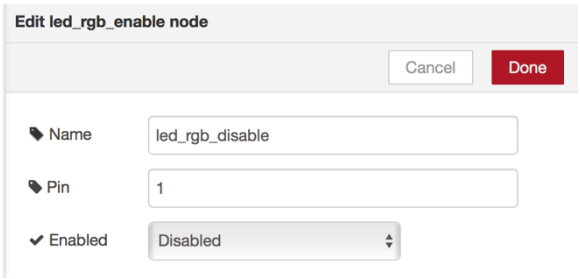
Connect the 2nd output of “CheckActions” node to the 2nd “led_rgb_enable” node(the one under “led_rgb_set”).



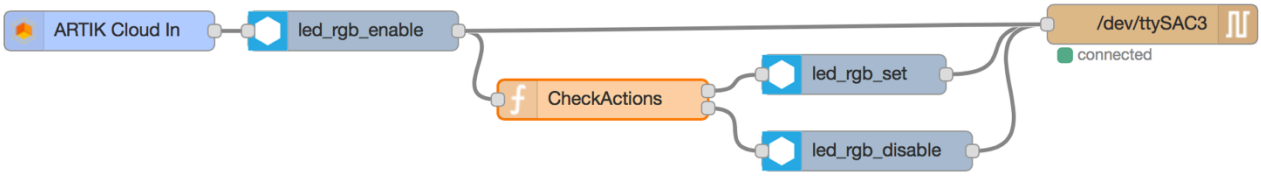
- In “led_rgb_set” node, configure the Pin # to be the same pin # you enabled in the step above, and use a 24bit hex color in the Color field. Here I define Color as “0x00FFFF”, which is blue.



- In the 2nd "led_rgb_enable" node (the one below "led_rgb_set"), disable the LED by setting "Enabled" to "Disabled". Rename the node to "led_rgb_disable".



Connect the "led_rgb_disable" node to "serial" node. Click the "serial" node and assign it to the serial port you are using. E.g, "/dev/ttySAC3" in our exercise.



Now, when your temperature rises above 32 degrees, the LED on your board will be turned on by rules engine. When the temperature is less than or equal to 32 degrees, the LED will turn off.