

SPRINT DELIVERY – 2

TEAM ID	PNT2022TMID12914
Project Name	Smart Farmer - IOT Enabled Smart Farming Application

Building Project

Connecting IoT Simulator to IBM Watson IoT Platform

Open link provided in above section 4.3

Give the credentials of your device in IBM Watson IoT Platform

Click on connect

My credentials given to simulator

are:

OrgID: **jztdcw**

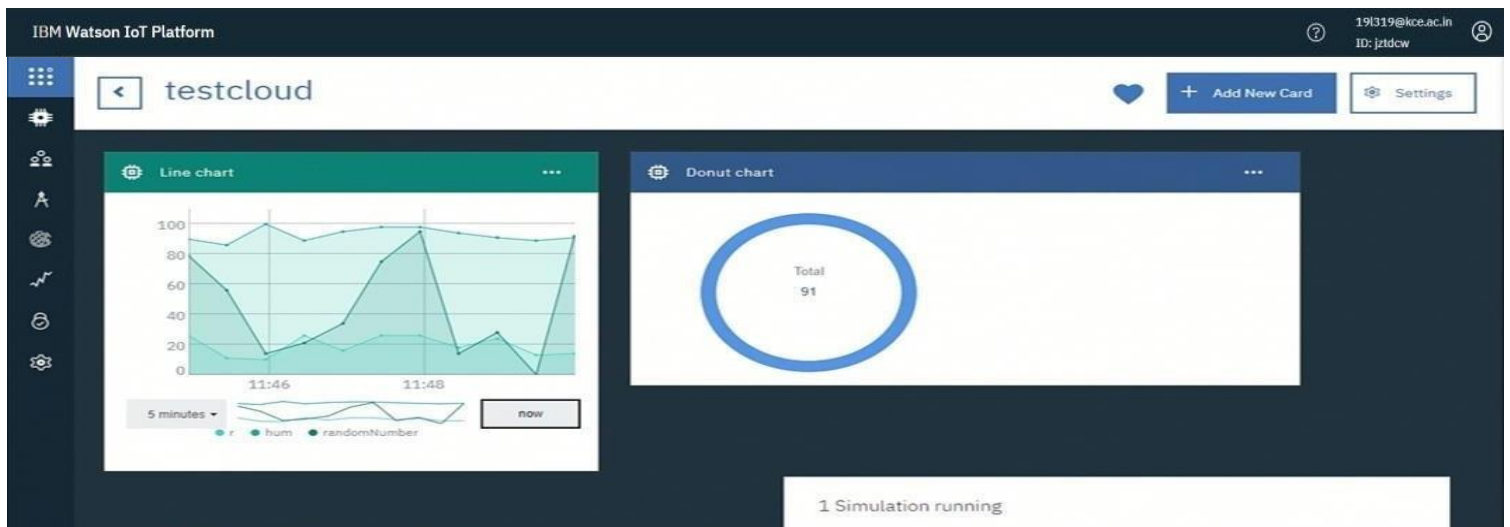
api: **a-157uf3f5rg4qxpd3**

Device type: **abcd**

token: **6ogMaaQHWNWFegOD8R?**

Device ID : **7654321**

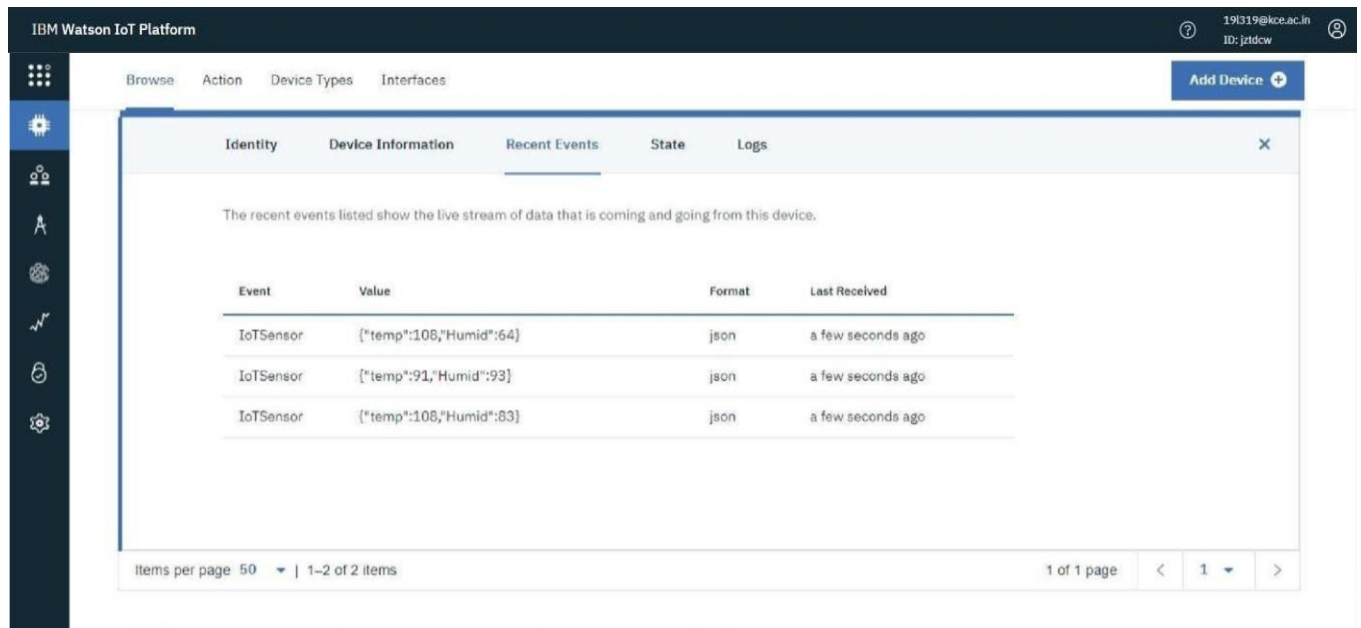
Device Token : **987654321**



You can see the received data in graphs by creating cards in Boards tab

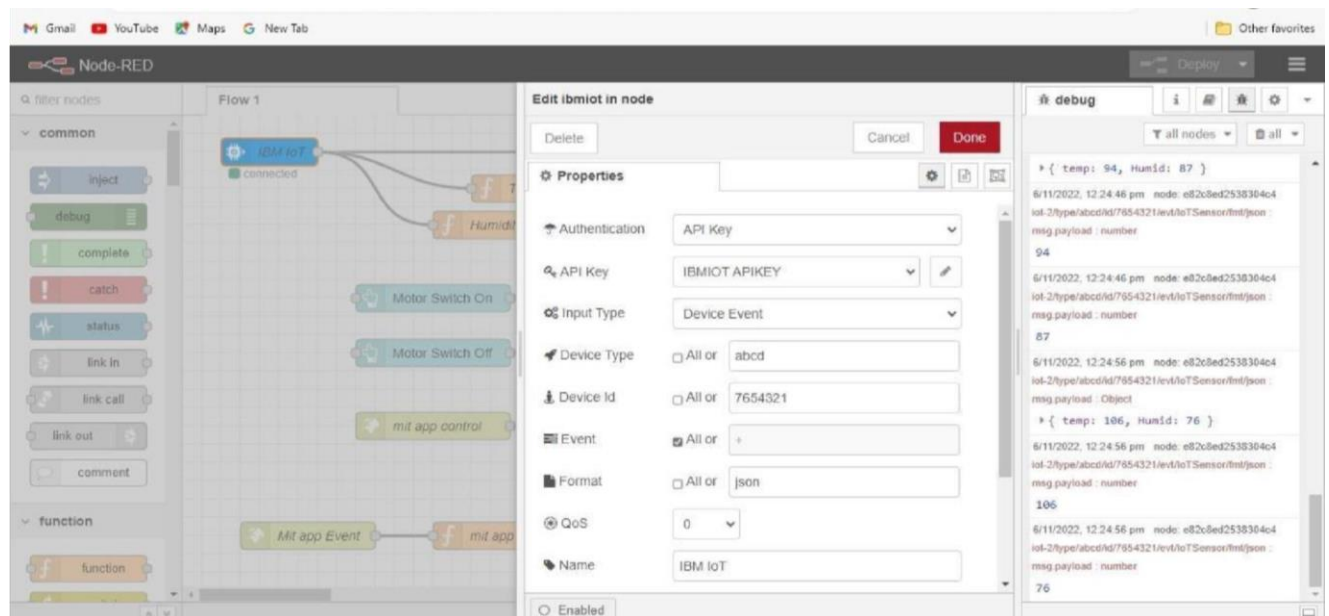
- You will receive the simulator data in cloud
- You can see the received data in Recent Events under your device
- Data received in this format(json)

```
{  
  "d": {  
    "name": "abcd",  
    "temperature": 17,  
    "humidity": 76,  
    "Moisture ": 25  
  }  
}
```



Configuration of Node-Red to collect IBM cloud data

The node IBM IoT App In is added to Node-Red workflow. Then the appropriate device credentials obtained earlier are entered into the node to connect and fetch device telemetry to Node-Red.



Once it is connected Node-Red receives data from the device

Display the data using debug node for verification

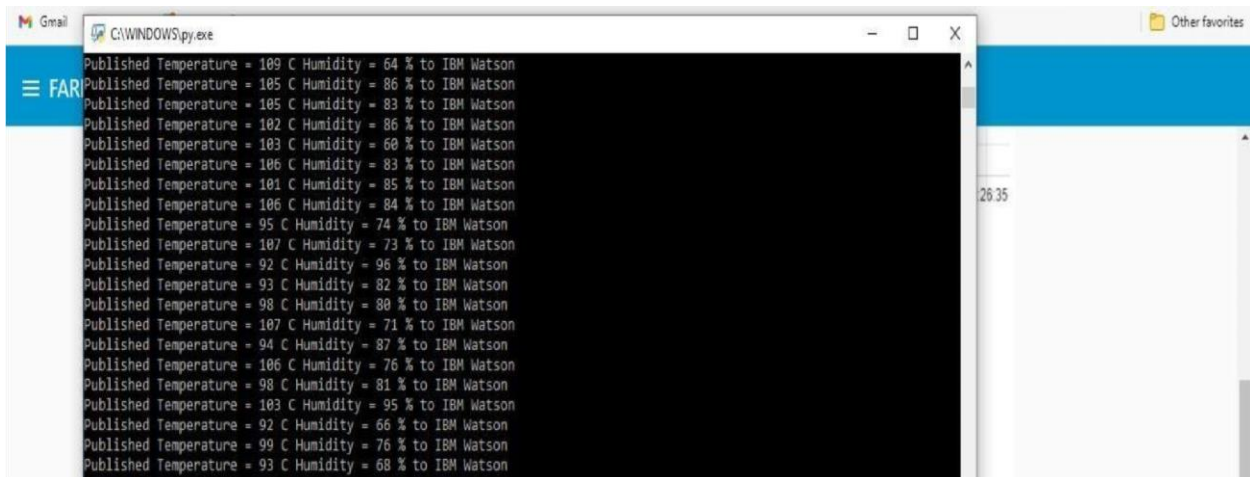
Connect function node and write the Java script code to get each reading separately.

The Java script code for the function node is:

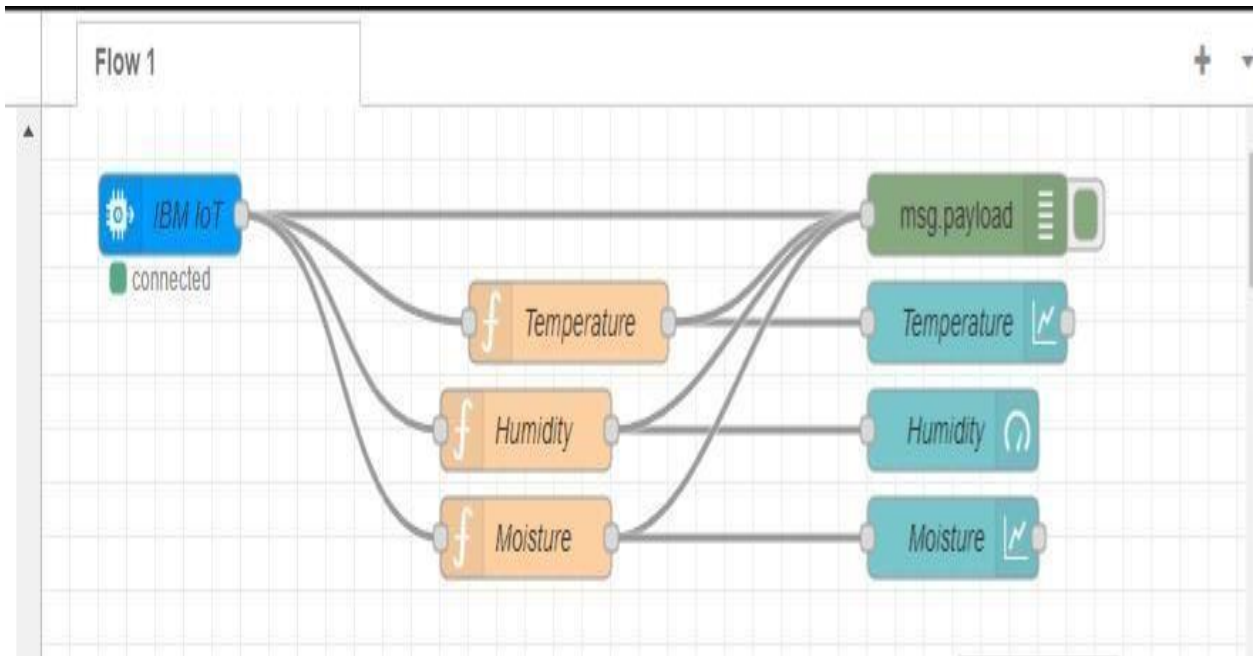
```
msg.payload=msg.payload.d.temperature
```

```
return msg;
```

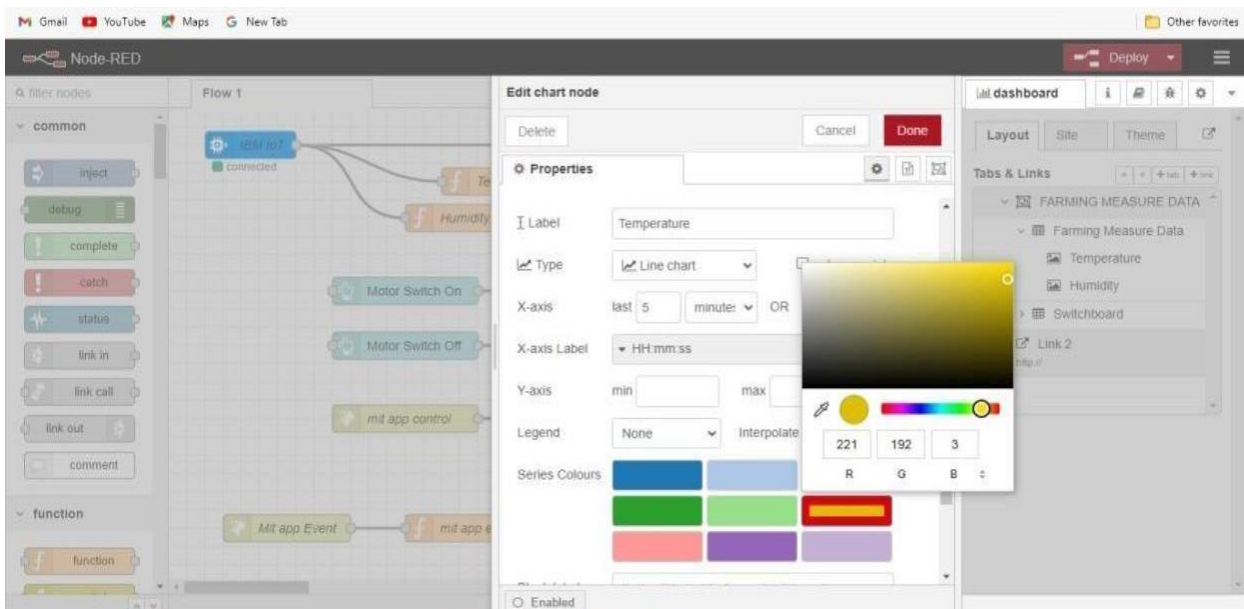
Finally connect Gauge nodes from dashboard to see the data in UI



Data received from the cloud in Node-Red console



Nodes connected in following manner to get each reading separately



This is the Java

script code I written for the function node to get Temperature separately.

Configuration of Node-Red to collect data from OpenWeather

The Node-Red also receive data from the OpenWeather API by HTTP GET request. An inject trigger is added to perform HTTP request for every certain interval. HTTP request node is configured with URL we saved before in section 4.4 The data we receive from OpenWeather after request is in below JSON

```
format:{"coord":{"lon":79.85,"lat":14.13},"weather":[{"id":803,"main":"Clouds",
"
description":"brokenclouds","icon":"04n"}],"base":"stations","main":{"temp":3
07
59,"feels_like":305.5,"temp_min":307.59,"temp_max":307.59,"pressure":1002,
"h
umidity":35,"sea_level":1002,"grnd_level":1000},"wind":{"speed":6.23,"deg":1
70}
,"clouds":{"all":68},"dt":1589991979,"sys":{"country":"IN","sunrise":158993355
3,
"sunset":1589979720},"timezone":19800,"id":1270791,"name":"Gūdūr","cod":
20
0}
```

In order to parse the JSON string we use Java script functions and get each parameters

```
var temperature = msg.payload.main.temp;
temperature = temperature-273.15;
return {payload : temperature.toFixed(2)};
```

In the above Java script code we take temperature parameter into a new variable and convert it from kelvin to Celsius

Then we add Gauge and text nodes to represent data visually in UI

The screenshot displays the Node-RED web interface. On the left, the 'common' node palette is visible, containing nodes like inject, debug, complete, catch, status, link in, link call, link out, and comment. The 'function' node palette is also visible, containing a 'function' node. In the center, a flow is being edited. A 'function' node is selected, and its configuration panel is open. The 'Name' field is set to 'Temperature'. The 'On Message' tab is active, showing the following JavaScript code:

```
1 msg.payload=msg.payload.temp
2 global.set("t",msg.payload)
3 return msg;
```

On the right, the 'debug' console is open, showing a list of messages. The messages are JSON objects containing temperature and humidity data:

```
{ temp: 107, Humid: 73 }
```

The messages are timestamped and include node IDs and IoT sensor information. The temperature values shown are 107, 73, 92, and 96.