

# **IOT ENABLED SMART FARMING APPLICATION**

## **SPRINT DELIVERY – 2**

TEAM ID: **PNT2022TMID47731**

TEAM MEMBERS:

**YUVASRI**

**MOOFIKA BEGAM**

**SIVABALA**

**DIVYA BHARATHI**

## 5, Building Project

### 5.1 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:

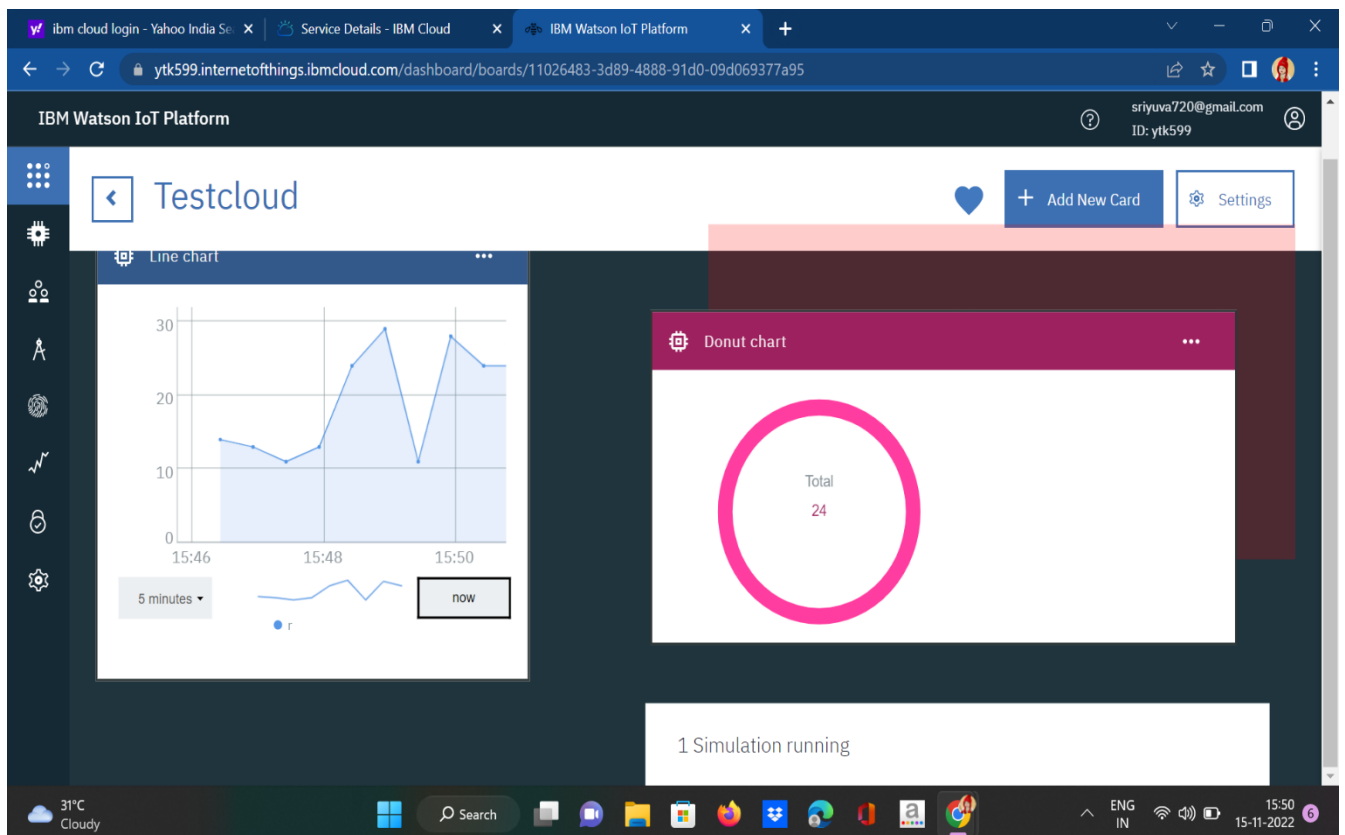
OrgID: **ytk599**

api: **a-ytk599-ix5bz4q7ej**

Device type: **wxyz**

Token: **V&tXP(r(N8SbPDqJz5)**

Device ID : **27**



Device Token : **27012701**

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)

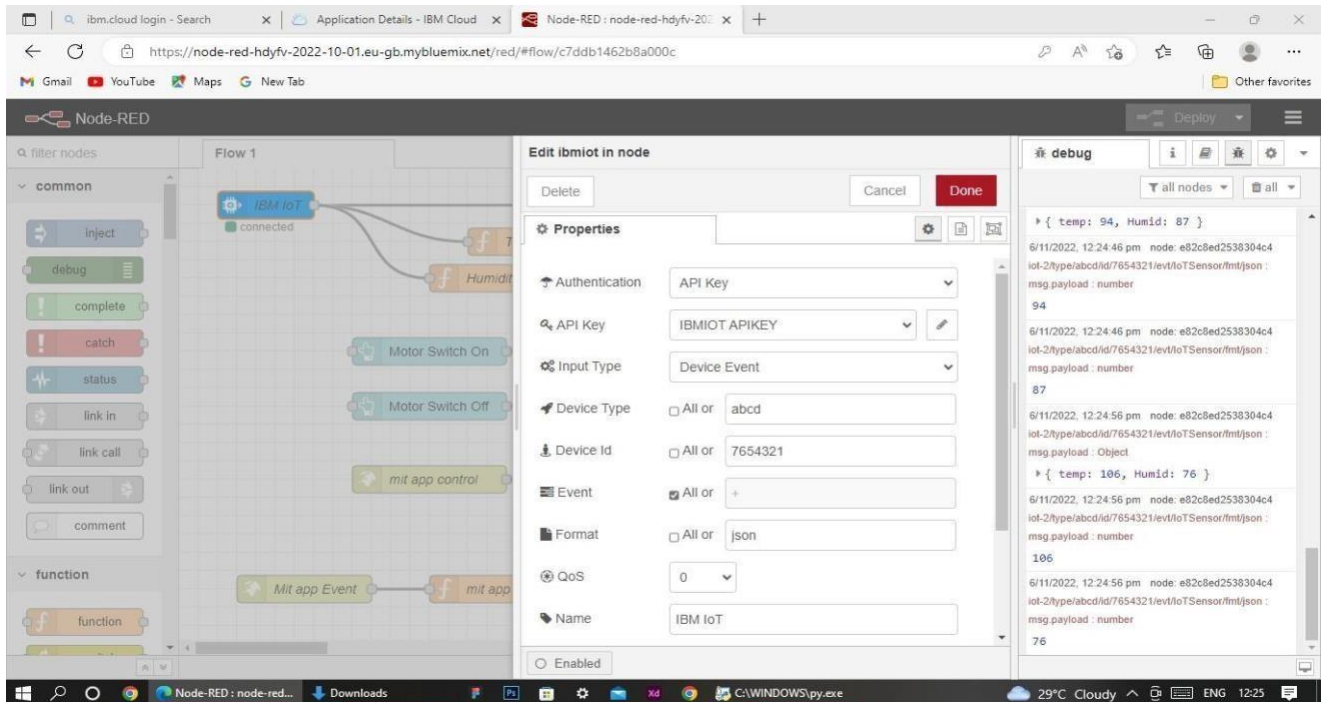
```
{
  "d": {
    "name": "wxyz",
    "temperature": 17,
    "humidity": 76,
    "Moisture ": 25
  }
}
```

The screenshot displays the IBM Watson IoT Platform dashboard. The top navigation bar includes tabs for Browse, Action, Device Types, and Interfaces. A sidebar on the left contains icons for various IoT functions. The main content area shows a modal window for a specific device, with tabs for Identity, Device Information, Recent Events, State, and Logs. The 'Recent Events' tab is active, displaying a table of live data streams. The table has four columns: Event, Value, Format, and Last Received. It lists four recent events from an 'IoTSensor' device, each providing temperature and humidity data in JSON format.

Event	Value	Format	Last Received
IoTSensor	{"temp":99,"Humid":96}	json	a few seconds ago
IoTSensor	{"temp":109,"Humid":84}	json	a few seconds ago
IoTSensor	{"temp":107,"Humid":95}	json	a few seconds ago
IoTSensor	{"temp":92,"Humid":82}	json	a few seconds ago

## 5.2 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:

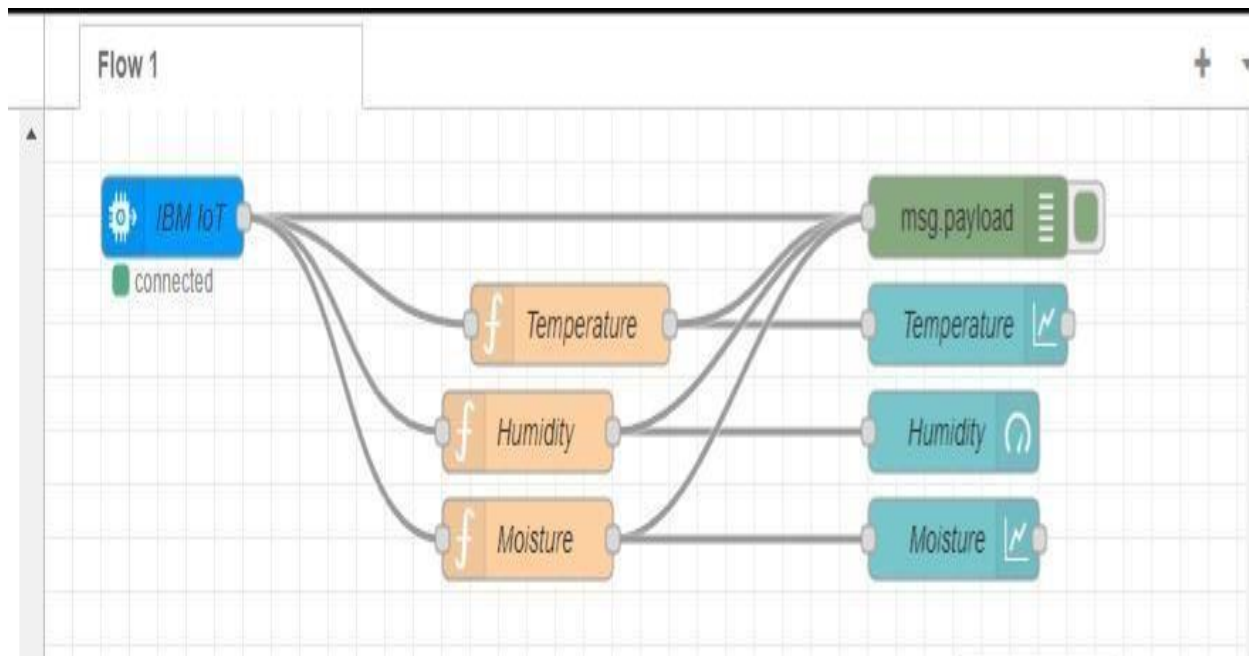
```
msg.payload=msg.payload.d.temperature return
```

```
msg;
```

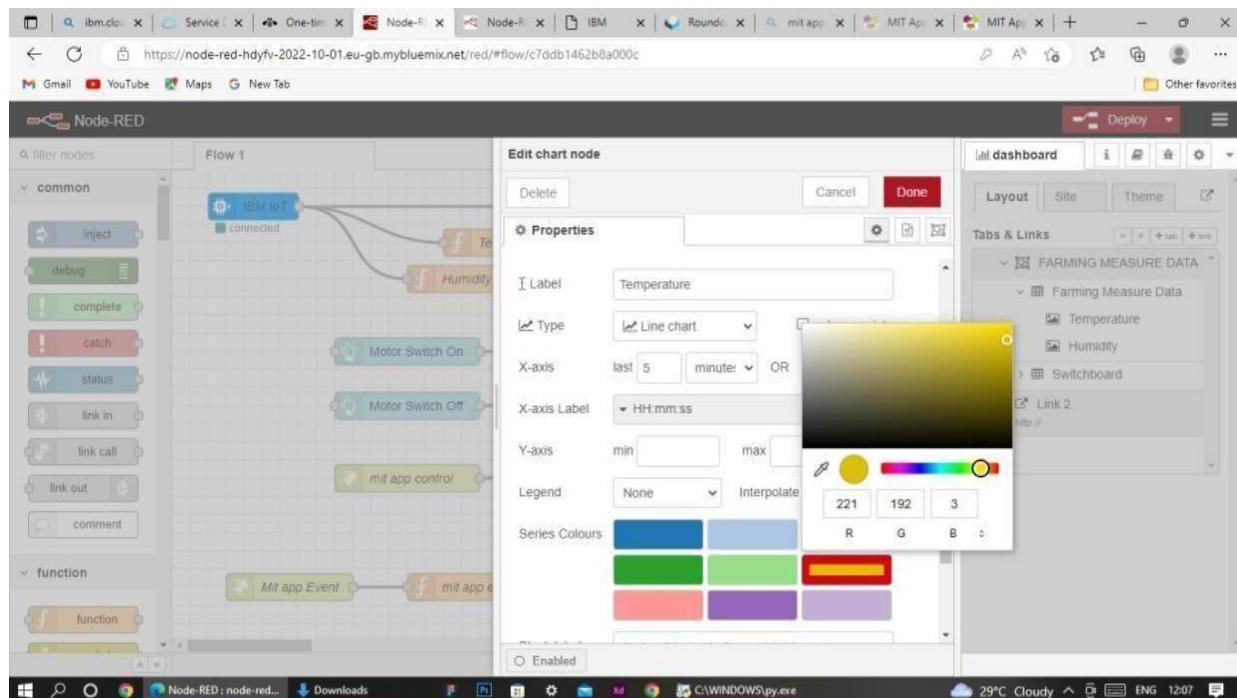
Finally connect Gauge nodes from dashboard to see the data in UI

```
Published Temperature = 109 C Humidity = 64 % to IBM Watson
Published Temperature = 105 C Humidity = 86 % to IBM Watson
Published Temperature = 105 C Humidity = 83 % to IBM Watson
Published Temperature = 102 C Humidity = 86 % to IBM Watson
Published Temperature = 103 C Humidity = 60 % to IBM Watson
Published Temperature = 106 C Humidity = 83 % to IBM Watson
Published Temperature = 101 C Humidity = 85 % to IBM Watson
Published Temperature = 106 C Humidity = 84 % to IBM Watson
Published Temperature = 95 C Humidity = 74 % to IBM Watson
Published Temperature = 107 C Humidity = 73 % to IBM Watson
Published Temperature = 92 C Humidity = 96 % to IBM Watson
Published Temperature = 93 C Humidity = 82 % to IBM Watson
Published Temperature = 98 C Humidity = 80 % to IBM Watson
Published Temperature = 107 C Humidity = 71 % to IBM Watson
Published Temperature = 94 C Humidity = 87 % to IBM Watson
Published Temperature = 106 C Humidity = 76 % to IBM Watson
Published Temperature = 98 C Humidity = 81 % to IBM Watson
Published Temperature = 103 C Humidity = 95 % to IBM Watson
Published Temperature = 92 C Humidity = 66 % to IBM Watson
Published Temperature = 99 C Humidity = 76 % to IBM Watson
Published Temperature = 93 C Humidity = 68 % to IBM Watson
```

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.

### 5.3 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": 78.0528,
    "lat": 9.918
  },
  "weather": [
    {
      "id": 804,
      "main": "Clouds",
      "description": "overcast",
      "icon": "04n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 298.21,
    "feels_like": 298.83,
    "temp_min": 298.21,
    "temp_max": 298.21,
    "pressure": 1012,
    "humidity": 79,
    "sea_level": 1012,
    "grnd_level": 995,
    "visibility": 10000,
    "wind": {
      "speed": 2.78,
      "deg": 50,
      "gust": 4.59
    },
    "clouds": {
      "all": 94
    },
    "dt": 1668520212,
    "sys": {
      "country": "IN",
      "sunrise": 1668472933,
      "sunset": 1668514958
    },
    "timezone": 19800,
    "id": 1254356,
    "name": "Tirupparangunram",
    "cod": 200
  }
}
```

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

