

MUTHAYAMMAL ENGINEERING COLEGE

Department of Electronics and Communication Engineering

Smart Farmer-IOT Enabled Smart Farming Application SPRINT DELIVERY – 2

TITLE	Smart Farmer-IOT Enabled Smart Farming Application
DOMAIN NAME	INTERNET OF THINGS
TEAM ID	PNT2022TMID19105
LEADER NAME	B. BALA KRISHNA
TEAM MEMBER NAME	I.RAVI KIRAN G.SUMANTH REDDY J.BHUVANESWAR
MENTOR NAME	SUBHASUNDHARI

Building Project

Connecting IoT Simulator to IBM Watson IoT Platform

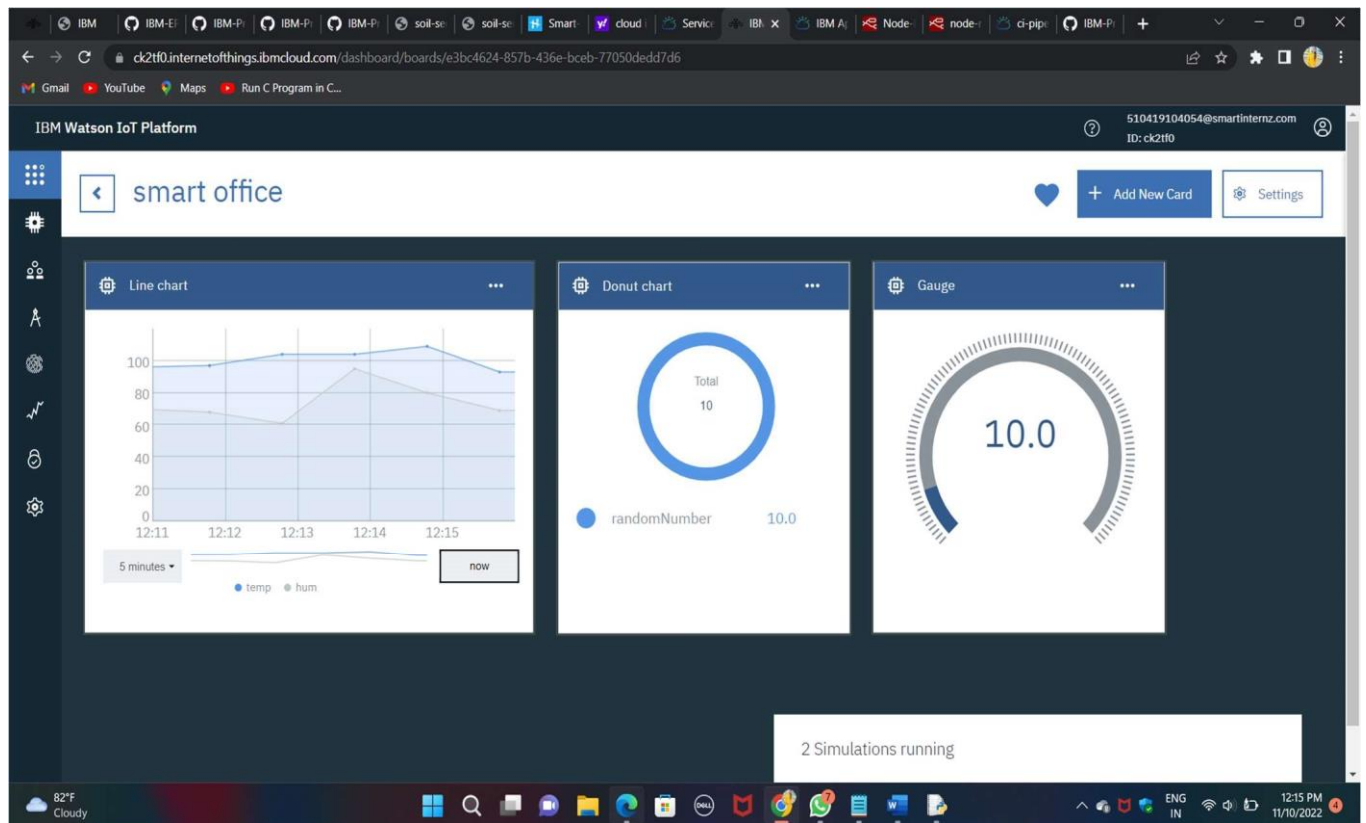
- ☐ Open link provided in below image
- ☐ Give the credentials of your device in IBM Watson

□ Platform Click on connect My credentials given to simulator are:

api: a-ck2tf0-yutwjanphx

Device type: 1234

Token: MaZKMomT_thHLD54ml



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

```

}
}

```

The screenshot displays the IBM Watson IoT Platform interface. The main dashboard shows a list of devices, including two NodeMCU devices with IDs 1234 and 12345, both in a 'Disconnected' state. Below this, the 'Recent Events' section for device 12345 is visible, showing a table of events with columns for Event, Value, Format, and Last Received. The events are of type 'eventflow' and contain JSON payloads with random values for 'randomNumber', 'temp', and 'hum'.

An overlay window titled 'Device Type: NodeMCU' is open, showing the configuration for an event. The 'Event type name' is set to 'eventflow'. The 'Schedule' is configured to trigger 'Every Minute'. The 'Payload' section shows a JSON structure with three fields: 'randomNumber' (random(0, 100)), 'temp' (random(90, 110)), and 'hum' (random(60, 100)). The window includes 'Send', 'Upload a CSV file', 'Cancel', and 'Save' buttons.

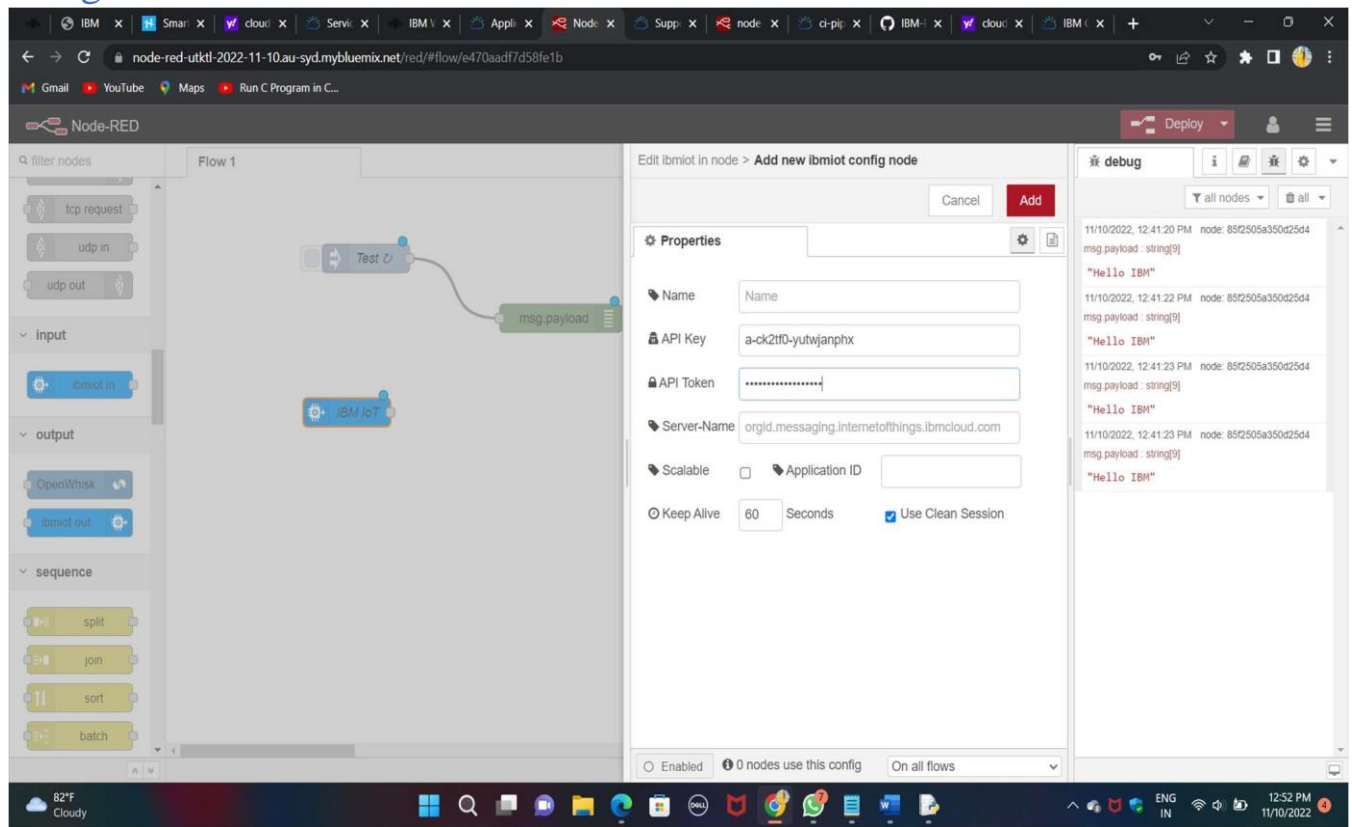
Event	Value	Format	Last Received
eventflow	{"randomNumber":17,"temp":103,"hum":91}	json	a few seconds ago
eventflow	{"randomNumber":9,"temp":109,"hum":66}	json	a few seconds ago
eventflow	{"randomNumber":77,"temp":101,"hum":98}	json	a few seconds ago

```

{
  "randomNumber": random(0, 100),
  "temp": random(90, 110),
  "hum": random(60, 100)
}

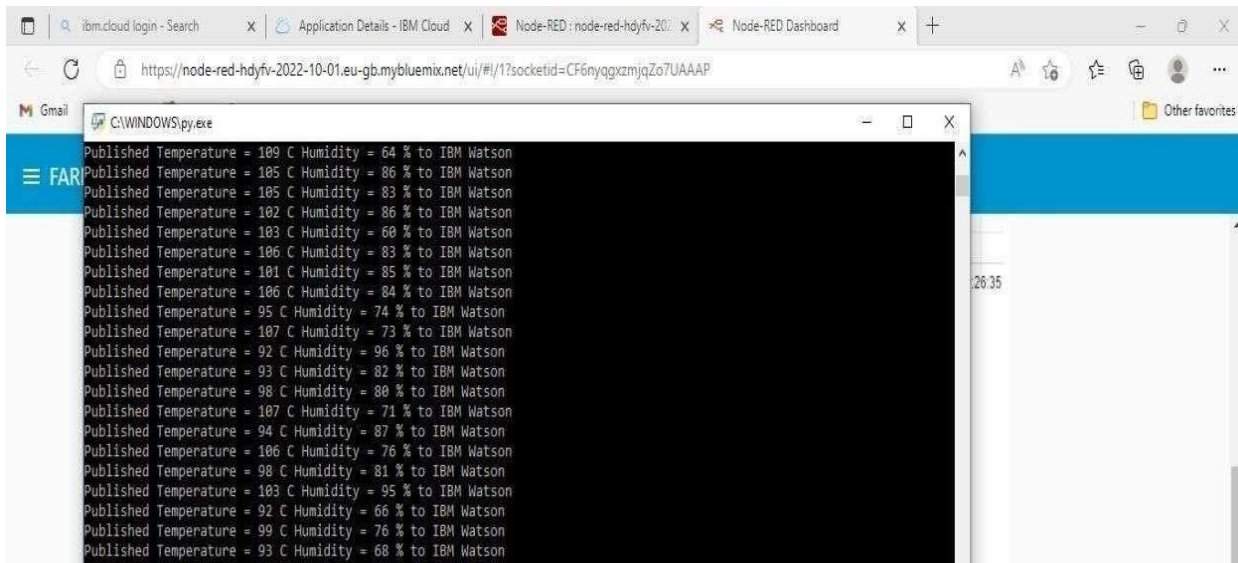
```

Configuration of Node-Red to collect IBM cloud data

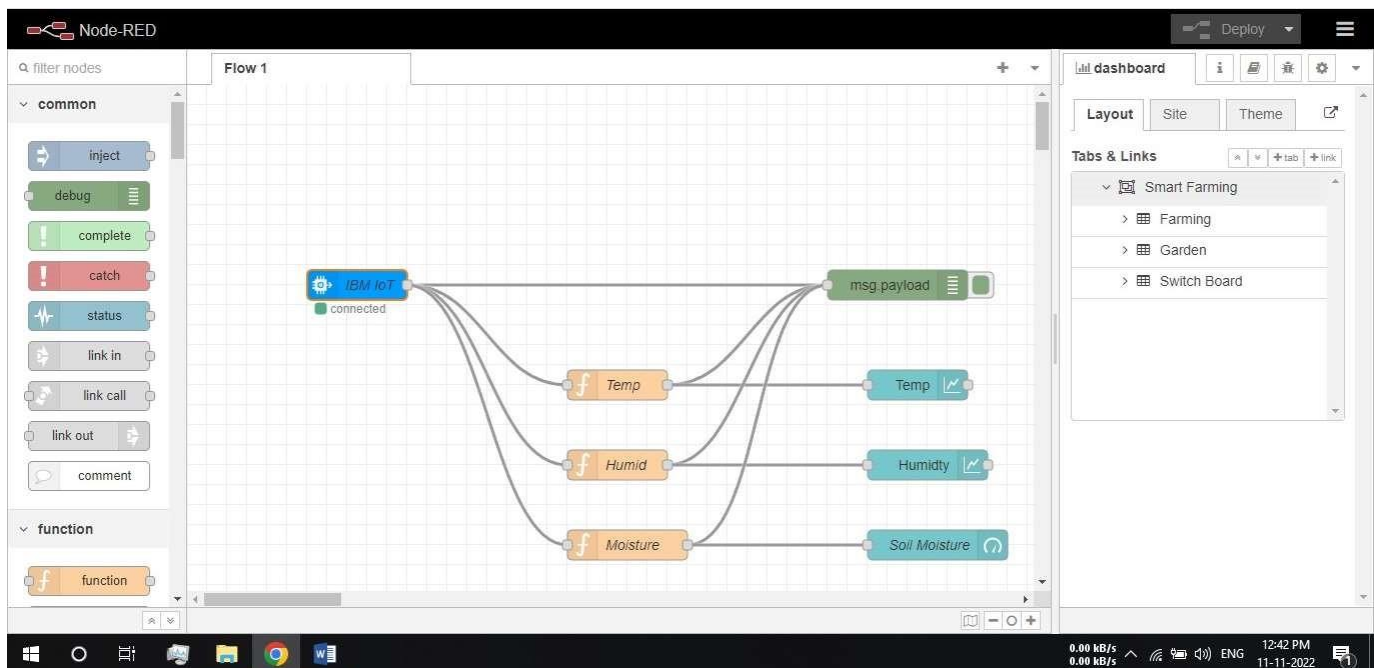


- ☐ The node IBM IoT App In is added to Node-Red workflow. Then the appropriate device credentials obtained earlier are entered into the node to connect and fetch device telemetry to Node-Red.
- ☐ Once it is connected Node-Red receives data from the device
Display the data using debug node for verification
- ☐ Connect function node and write the Java script code to get each reading separately.
- ☐ The Java script code for the function node is:

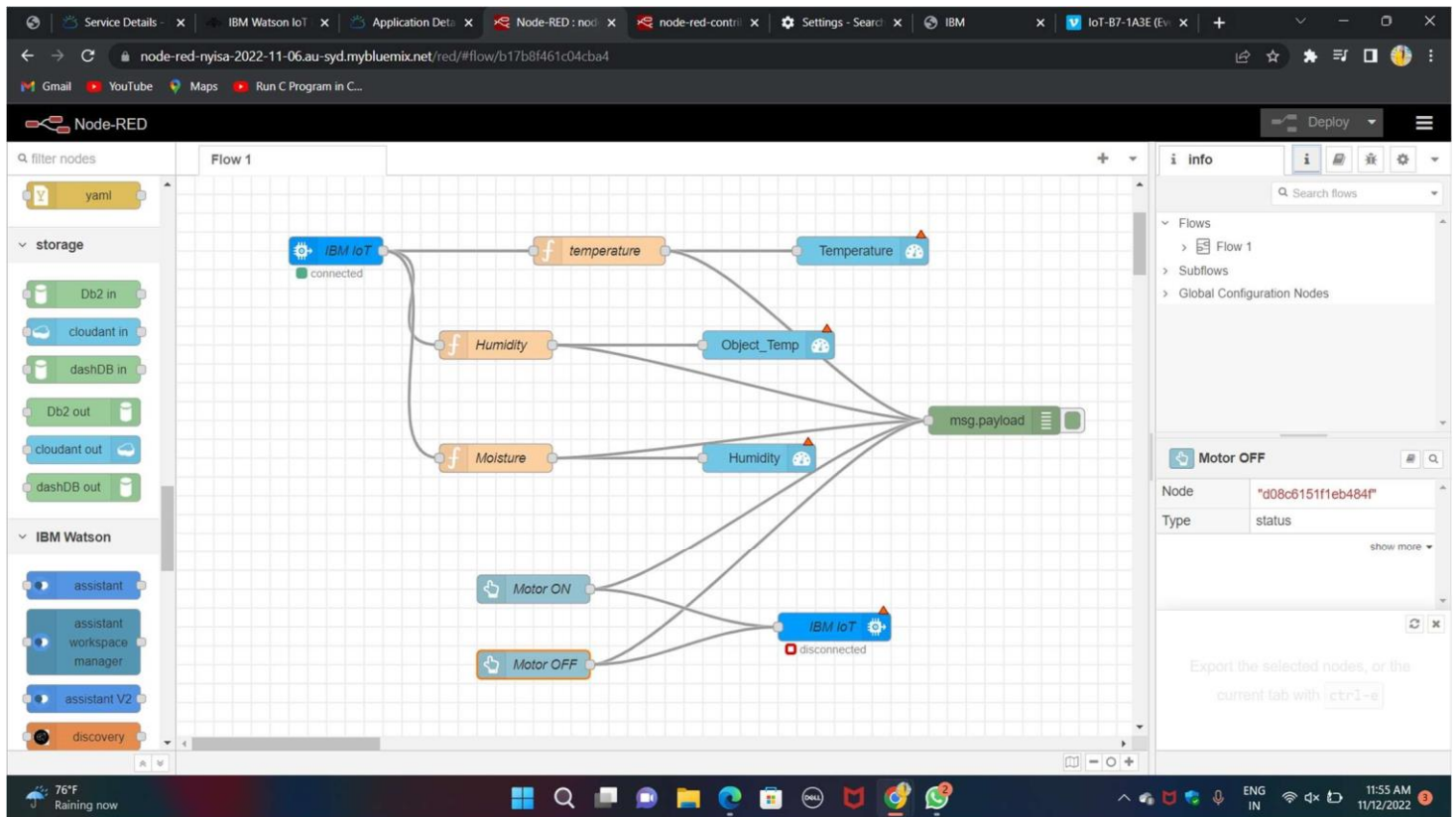
```
msg.payload=msg.payload.d.temperature  
return msg;
```
- ☐ Finally connect Gauge nodes from dashboard to see the data in UI



□ 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":307
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":170
}}
```

```
, "clouds": { "all": 68 }, "dt": 1589991979, "sys": { "country": "IN", "sunrise": 1589933553, "sunset": 1589979720 }, "timezone": 19800, "id": 1270791, "name": "Gūdūr", "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

