

Smart Agricultural System Based On IoT

Team Id	PNT2022TMID25689
Project title	SmartFarmer - IoT Enabled Smart Farming Application
Team members	S. Tharunkumar K. Yuvaraj S. Rubanraj P. Suresh

CONTENTS

1. Introduction
2. Problem Statement
3. Proposed Solution
4. Theoretical Analysis
 - 4.1 Block Diagram
 - 4.2 Required Software installation
 - 4.2.A Node-Red
 - 4.2.B IBM Watson IoT Platform
 - 4.2.C Python IDE
 - 4.3 IoT simulator
 - 4.4 OpenWeather API
5. Building Project
 - 5.1 Connecting IoT Simulator to IBM Watson IoT Platform
 - 5.2 Configuration of Node-Red to collect IBM cloud data
 - 5.3 Configuration of Node-Red to collect data from OpenWeather
 - 5.4 Configuration of Node-Red to send commands to IBM cloud
 - 5.5 Adjusting User Interface
 - 5.5 Receiving commands from IBM cloud using Python program
6. Flow chart
7. Observations & Results

8. Advantages & Disadvantages

9. Conclusion

10. Bibliography

1. Introduction

The main aim of this project is to help farmers automate their farms by providing them with a Web App through which they can monitor the parameters of the field like Temperature, soil moisture, humidity and etc and control the equipment like water motor and other devices remotely via internet without their actual presence in the field.

2. Problem Statement

Farmers are to be present at farm for its maintenance irrespective of the weather conditions. They have to ensure that the crops are well watered and the farm status is monitored by them physically. Farmer have to stay most of the time in field in order to get a good yield. In difficult times like in the presence of pandemic also they have to work hard in their fields risking their lives to provide food for the country.

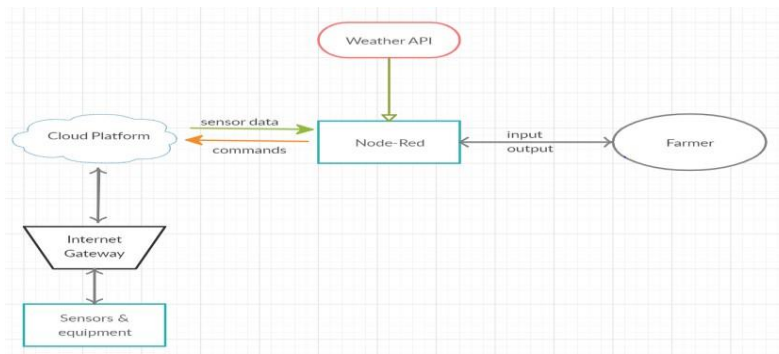
3. Proposed Solution

In order to improve the farmer's working conditions and make them easier, we introduce IoT services to him in which we use cloud services and internet to enable farmer to continue his work remotely via internet. He can monitor the field parameters and control the devices in farm.

4. Theoretical Analysis

4.1 Block Diagram

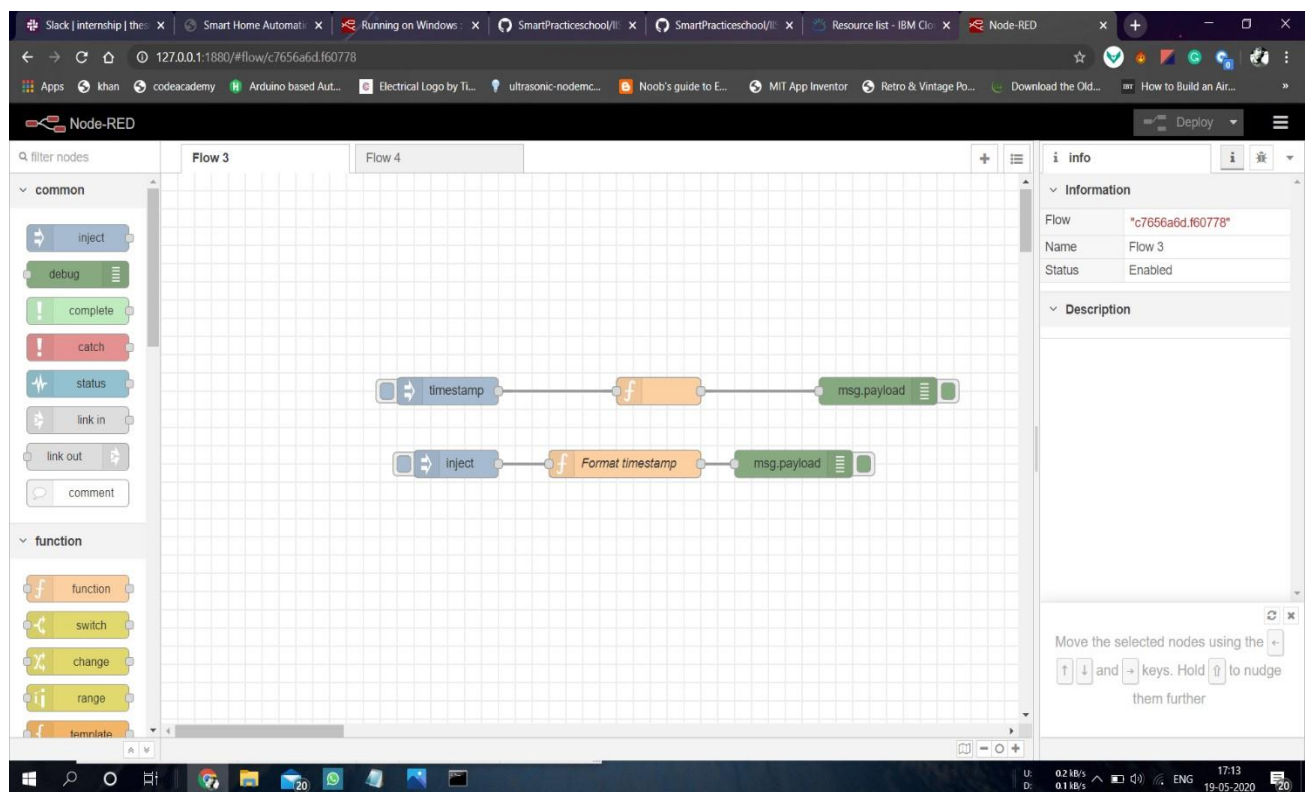
In order to implement the solution , the following approach as shown in the block diagram is used



4.2 Required Software Installation

4.2.A Node-Red

Node-RED is a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions.



Installation :

- First install npm/node.js
- Open cmd prompt
- Type => `npm install node-red` **To run the application :**
- Open cmd prompt
- Type=> `node-red`
- Then open <http://localhost:1880/> in browser

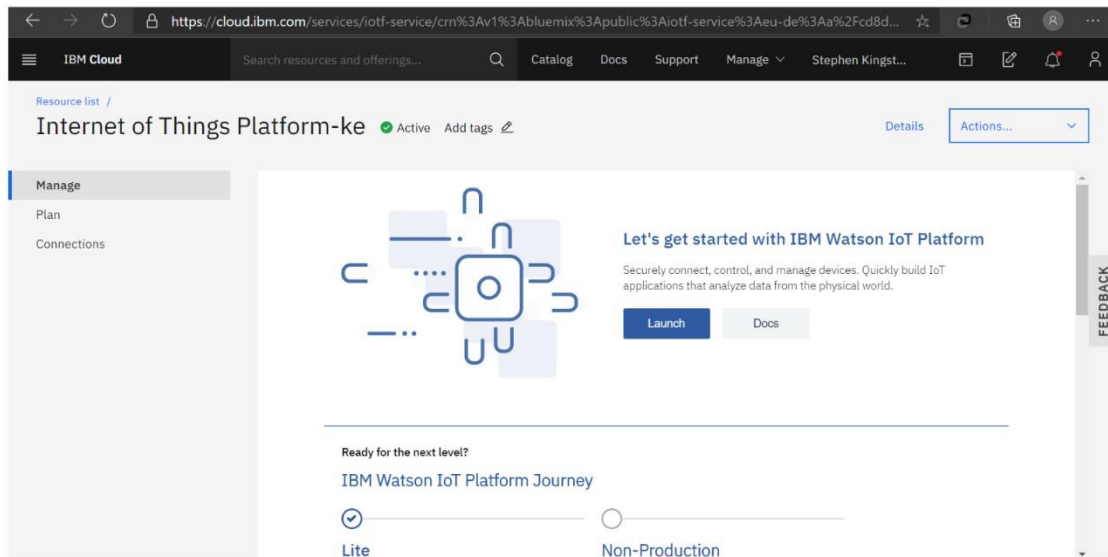
Installation of IBM IoT and Dashboard nodes for Node-Red

In order to connect to IBM Watson IoT platform and create the Web App UI these nodes are required

1. IBM IoT node
2. Dashboard node

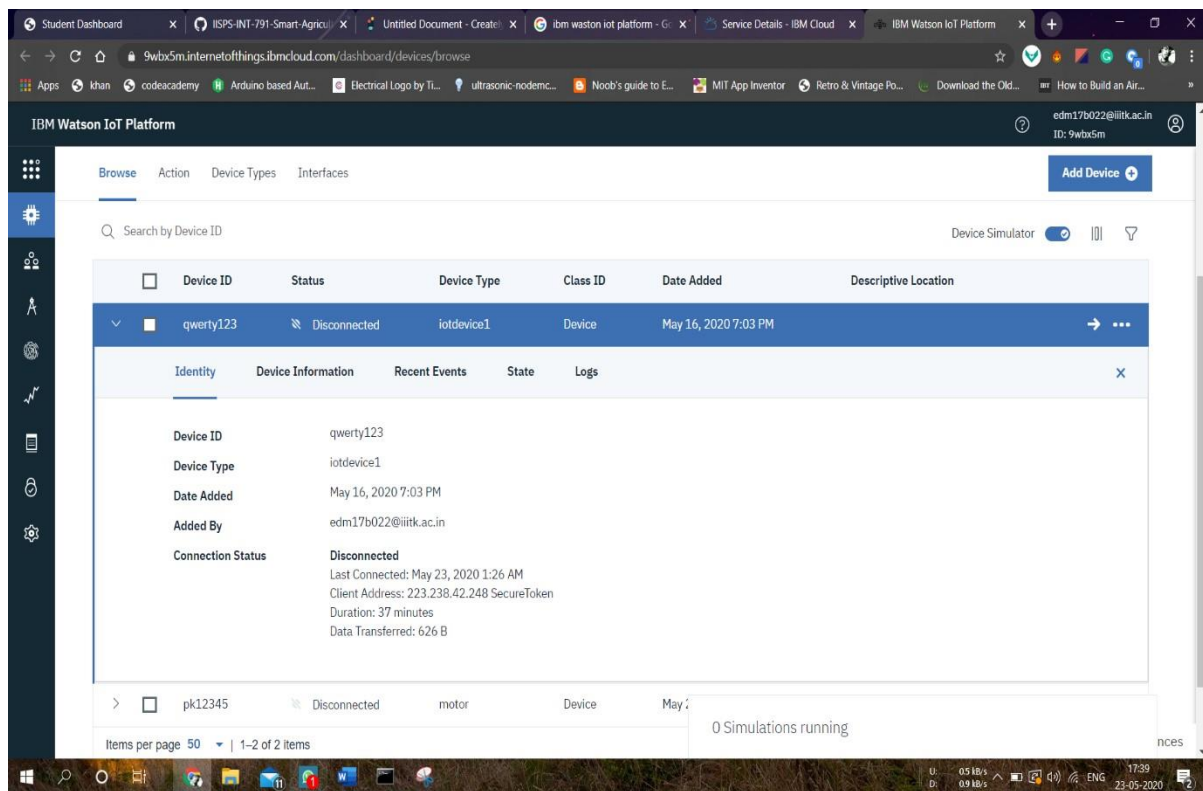
4.2.B IBM Watson IoT Platform

A fully managed, cloud-hosted service with capabilities for device registration, connectivity, control, rapid visualization and data storage. IBM Watson IoT Platform is a managed, cloud-hosted service designed to make it simple to derive value from your IoT devices.



Steps to configure:

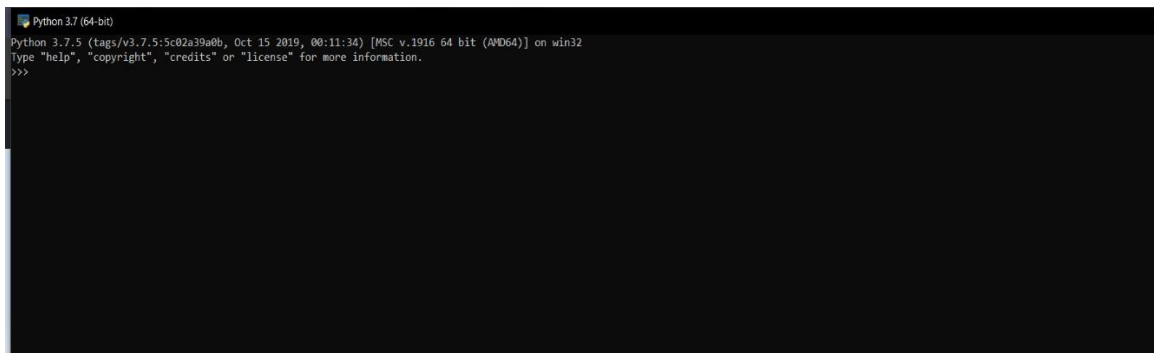
- Create an account in IBM cloud using your email ID
- Create IBM Watson Platform in services in your IBM cloud account
- Launch the IBM Watson IoT Platform
- Create a new device
- Give credentials like device type, device ID, Auth. Token • Create API key and store API key and token elsewhere.



4.2.C Python IDE

Install Python3 compiler

Install any python IDE to execute python scripts, in my case I used Spyder to execute the code.



4.3 IoT Simulator

In our project in the place of sensors we are going to use IoT sensor simulator which give random readings to the connected cloud.

The link to simulator: <https://watson-iot-sensor-simulator.mybluemix.net/>

We need to give the credentials of the created device in IBM Watson IoT Platform to connect cloud to simulator.

4.4 OpenWeather API

OpenWeatherMap is an online service that provides weather data. It provides current weather data, forecasts and historical data to more than 2 million customer.

Website link: <https://openweathermap.org/guide>

Steps to configure:

○ Create account in OpenWeather
○ Find the name of your city by searching
○ Create API key to your account
○ Replace “city name” and “your api key” with your city and API key in below red text **api.openweathermap.org/data/2.5/weather?q={city name}&appid={your api key}** Link I used in my project:

<http://api.openweathermap.org/data/2.5/weather?q=Gudur,in&appid=62354068e45f41ffa6a5b164714145fe>

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: 9wbx5

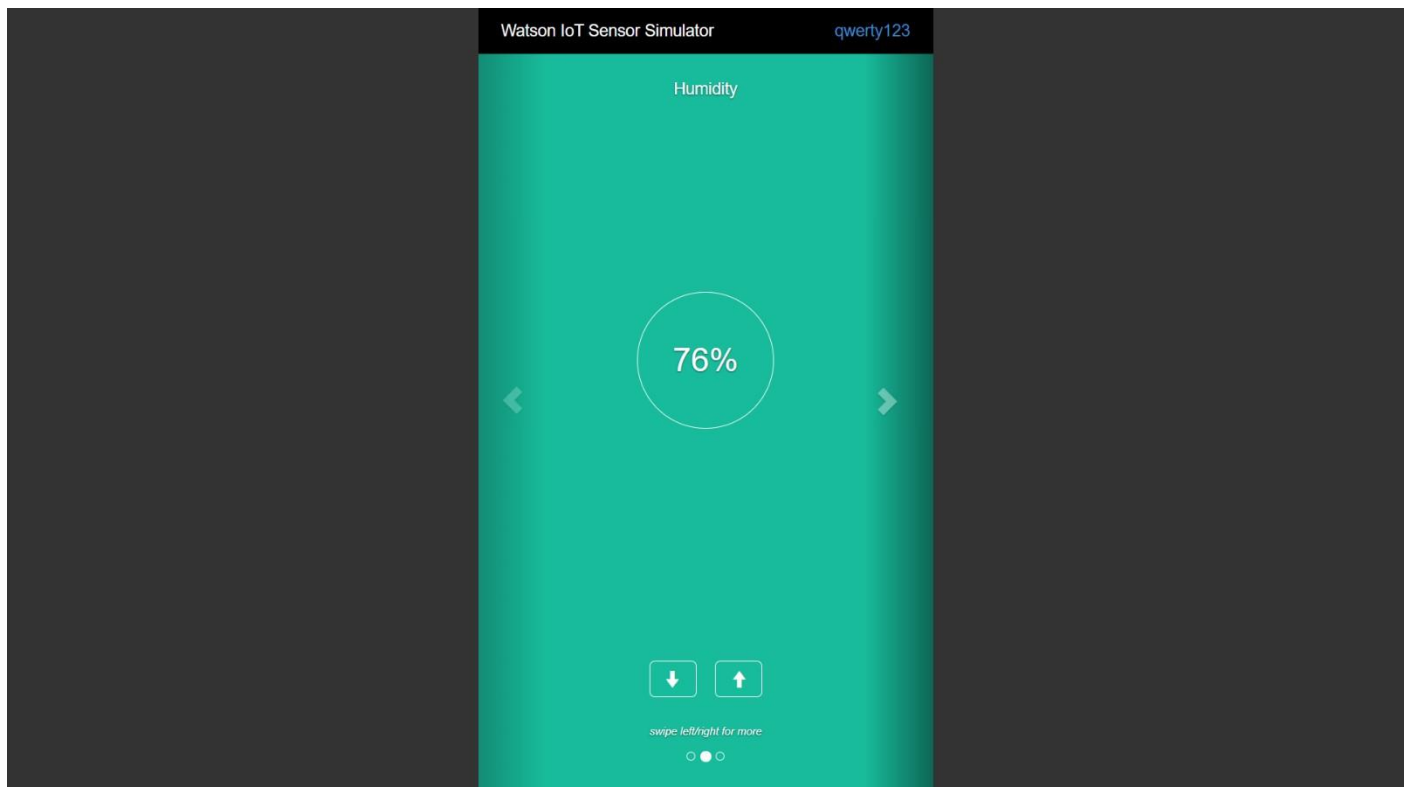
api: a-9wbx5m-1qfklrf7jl

Device type: iotdevice1

token: JcU(4(9Z37PdL!Rmz(

Device ID : qwerty123

Device Token : johnyjohnnyespapa



- 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": "qwerty123",
```

```

✦ "temperature": 17,
✦ "humidity": 76,
✦ "objectTemp": 25 } }

```



▼

qwerty123

Connected

iotdevice1

Device

May 16, 2020 7:03 PM

Identity

Device Information

Recent Events

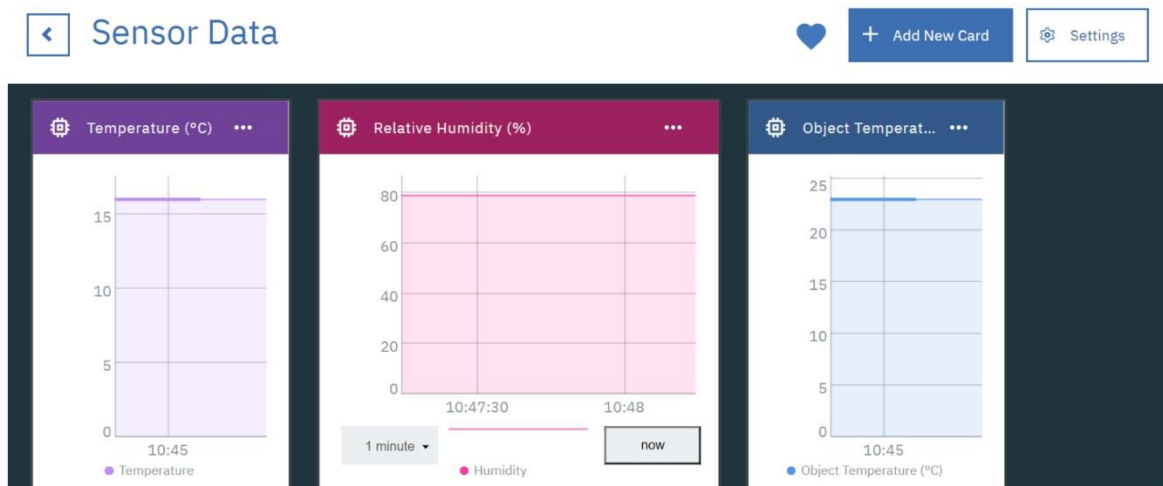
State

Logs

The recent events listed show the live stream of data that is coming and going from this device.

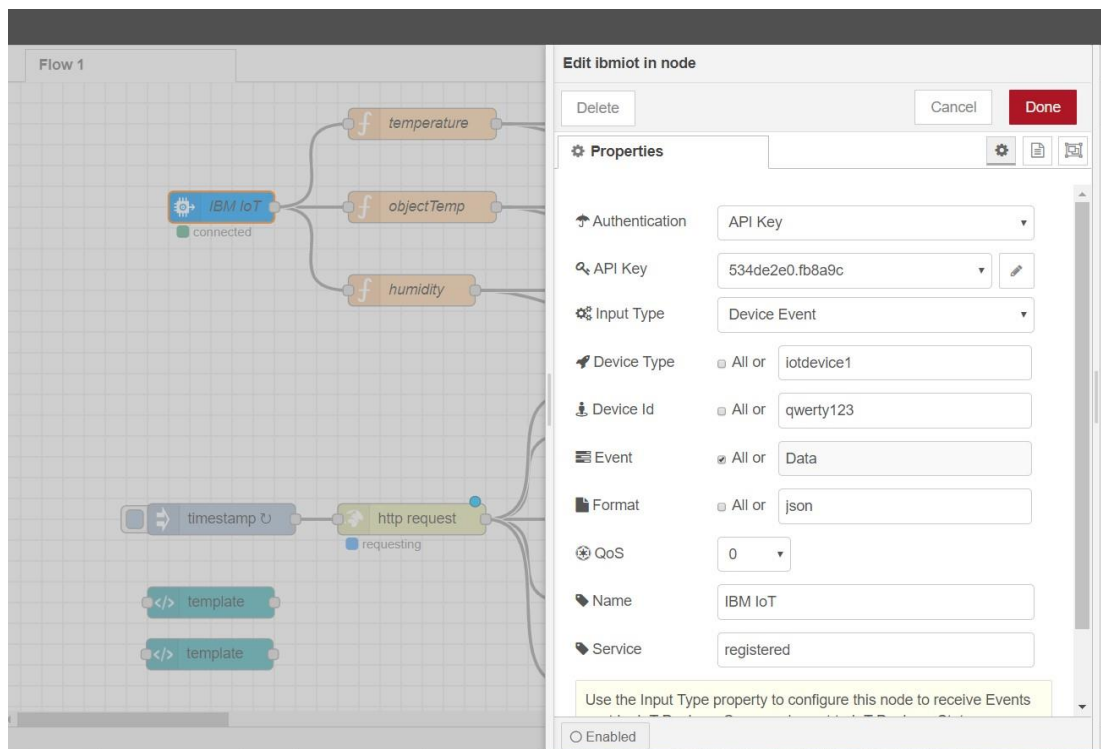
Event	Value	Format	Last Received
iotsensor	{"d":{"name":"qwerty123","temperature":17,"hu...	json	a few seconds ago
iotsensor	{"d":{"name":"qwerty123","temperature":17,"hu...	json	a few seconds ago
iotsensor	{"d":{"name":"qwerty123","temperature":17,"hu...	json	a few seconds ago
iotsensor	{"d":{"name":"qwerty123","temperature":17,"hu...	json	a few seconds ago
iotsensor	{"d":{"name":"qwerty123","temperature":17,"hu...	json	a few seconds ago

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



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

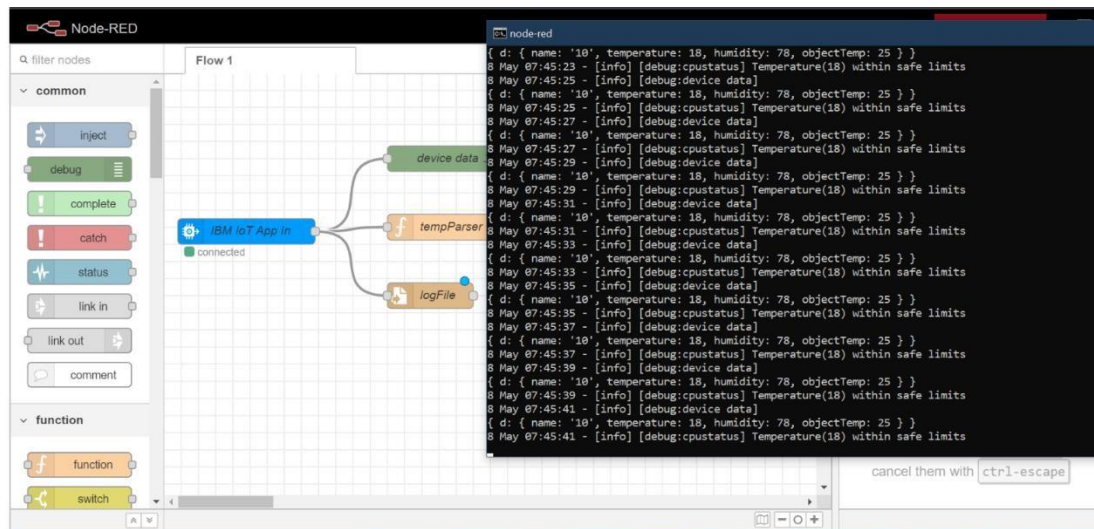
Connect function node and write the Java script code to get each reading separately.

The Java script code for the function node is:

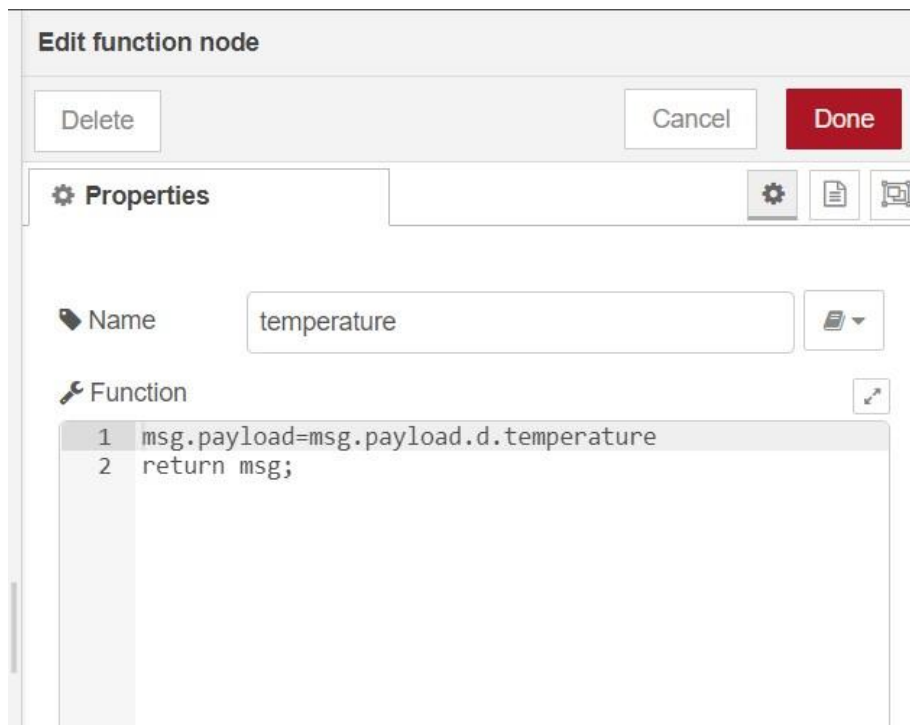
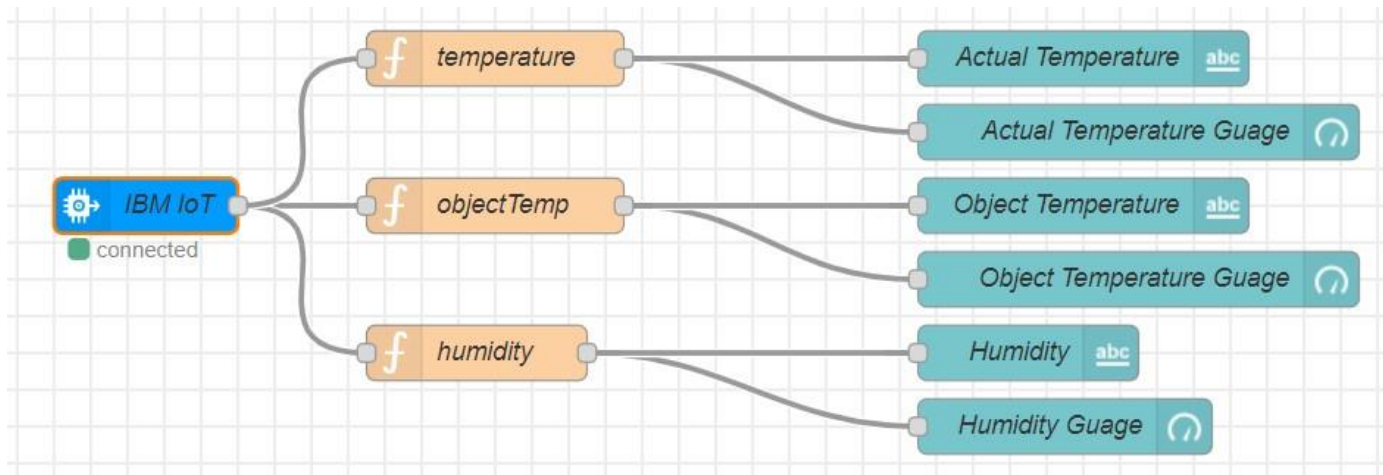
```
msg.payload=msg.payload.d.temperature  
return msg;
```

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

Nodes connected in following manner to get each reading separately



Data received from the cloud in Node-RED console



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": 79.85,
    "lat": 14.13
  },
  "weather": [
    {
      "id": 803,
      "main": "Clouds",
      "description": "broken clouds",
      "icon": "04n"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 307.59,
    "feels_like": 305.5,
    "temp_min": 307.59,
    "temp_max": 307.59,
    "pressure": 1002,
    "humidity": 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
}
```

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

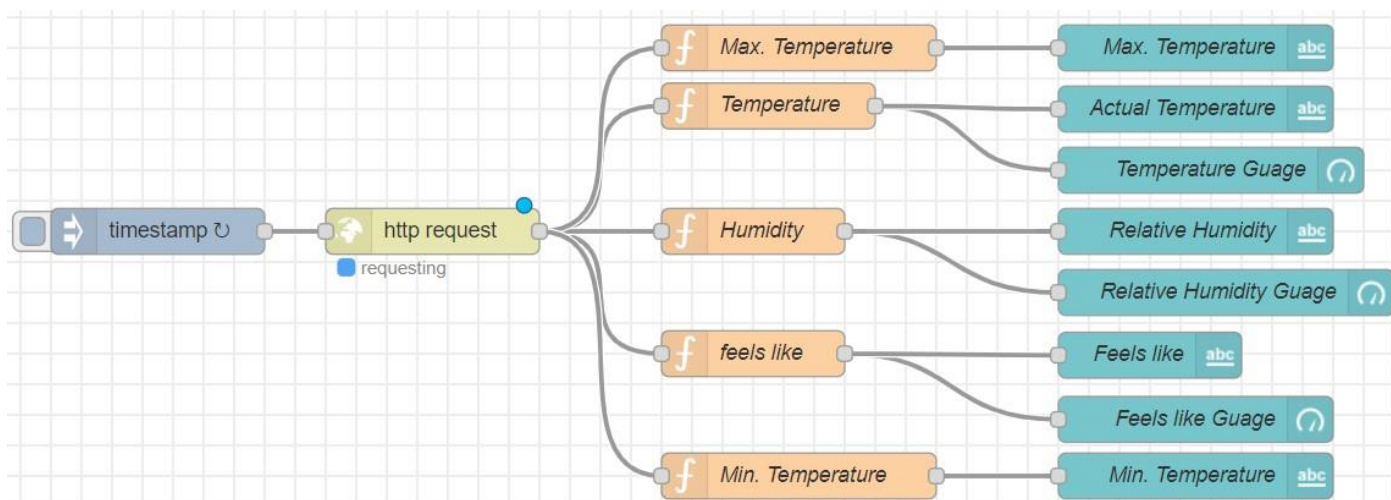
```
var temperature = msg.payload.main.temp;

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 above image has the program flow for receiving data from OpenWeather

Edit http request node

Delete Cancel Done

Properties

Method: GET

URL: http://api.openweathermap.org/data/2.5/weather?i

☐ Append msg.payload as query string parameters

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy

Return: a parsed JSON object

Name: Name

Tip: If the JSON parse fails the fetched string is returned as-is.

Edit function node

Delete Cancel Done

Properties

Name: Temperature

Function

```

1 var temperature = msg.payload.main.temp;
2 temperature = temperature-273.15;
3 return {payload : temperature.toFixed(2)};
4

```

The above two images contain http request and function node data that needs to be filled.

5.4 Configuration of Node-Red to send commands to IBM cloud

ibmiot out node I used to send data from Node-Red to IBM Watson device. So, after adding it to the flow we need to configure it with credentials of our Watson device.

Edit ibmiot out node

Delete Cancel Done

Properties

Authentication: API Key

API Key: 534de2e0.fb8a9c

Output Type: Device Command

Device Type: iotdevice1

Device Id: qwerty123

Command Type: motor_on_off

Format: json

Data: ON

QoS: 0

Name: IBM IoT

Service: registered

☒ Enabled

Here we add three buttons in UI which each sends a number 0,1 and 2.

0 -> for motor off

1 -> for motor on

2 -> for running motor continuously 30 minutes

We used a function node to analyse the data received and assign command to each number.

The Java script code for the analyser is:

```

if(msg.payload===1)
  msg.payload={"command":"ON"}; else
  if(msg.payload===0)
    msg.payload={"command":"OFF"};
  else
    msg.payload={"command":"runfor30minutes"};
return msg;

```

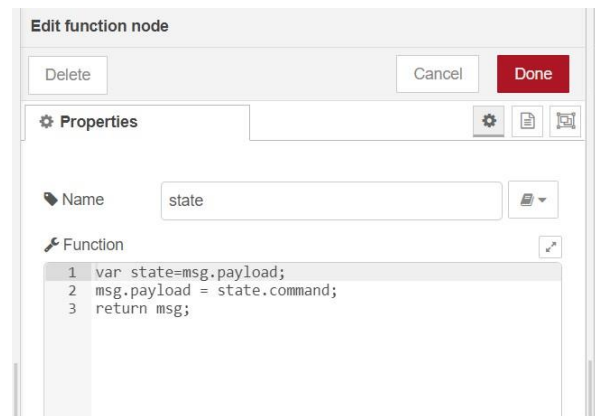
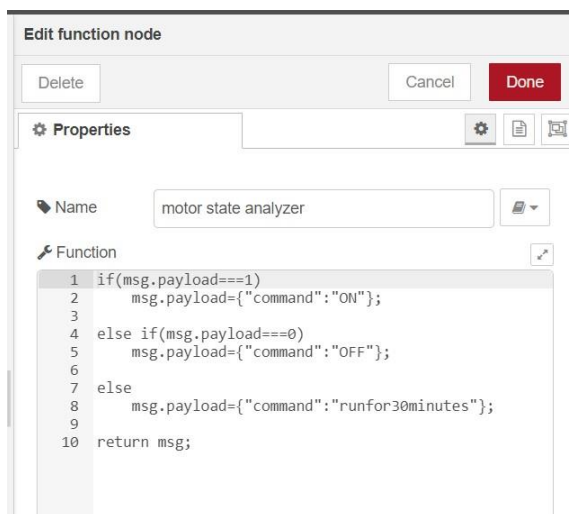
Then we use another function node to parse the data and get the command and represent it visually with text node.

The Java script code for that function node is:

```

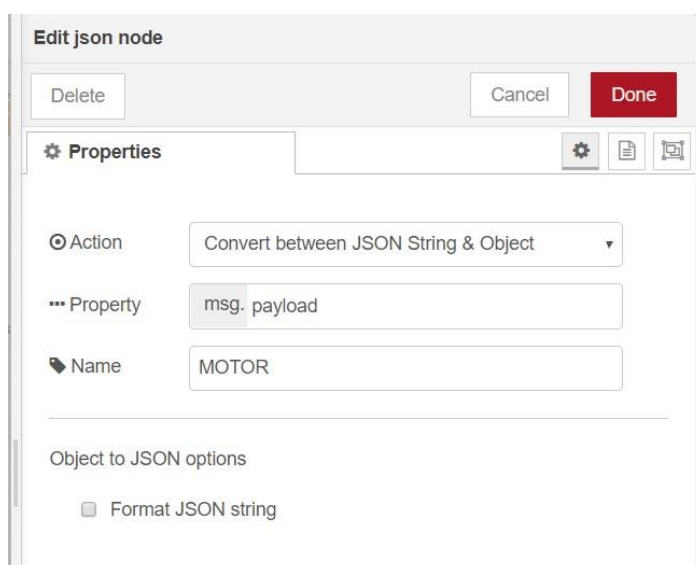
var state=msg.payload;
msg.payload = state.command;
return msg;

```

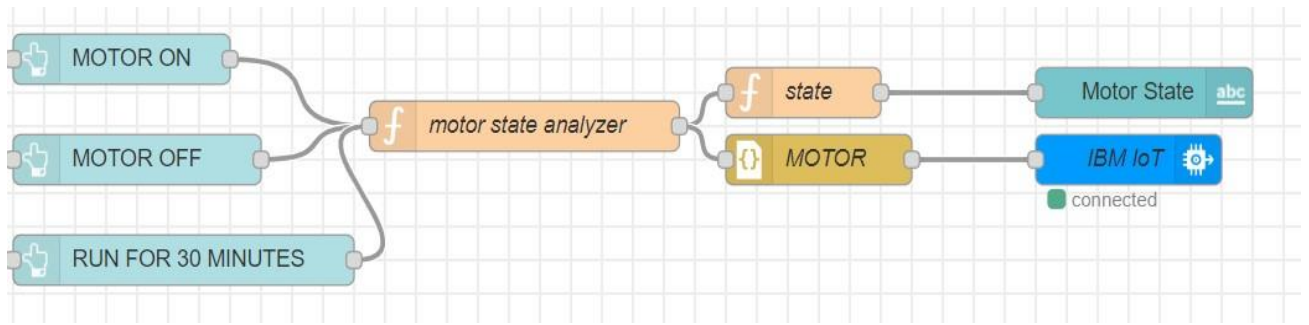


The above images show the java script codes of analyser and state function nodes.

Then we add edit Json node to the conversion between JSON string & object and finally connect it to IBM IoT Out.



Edit JSON node needs to be configured like this



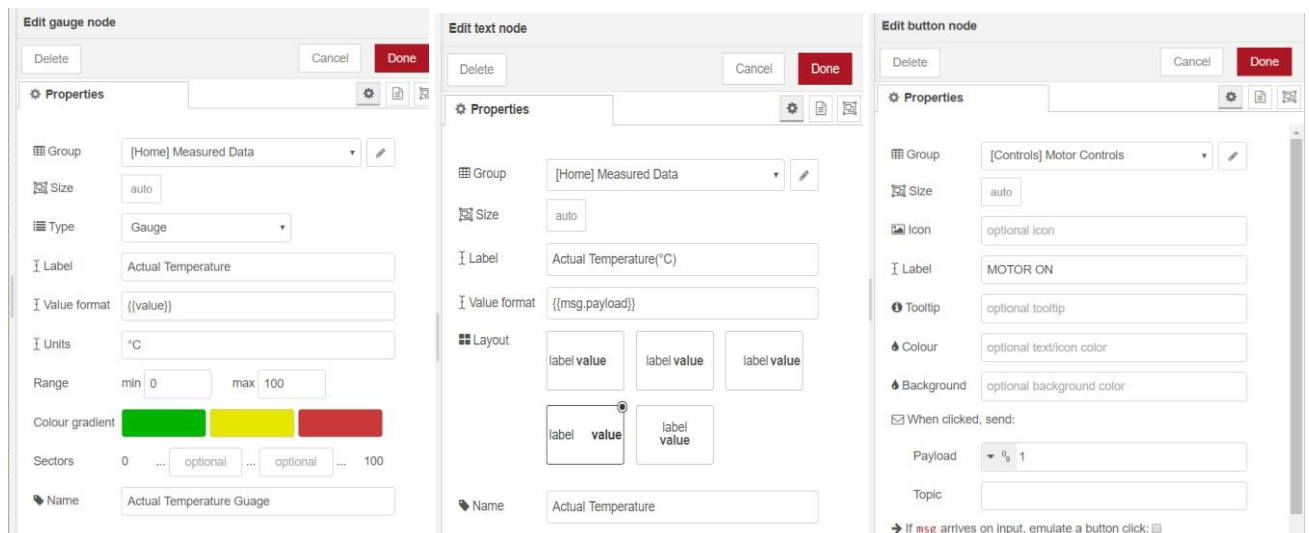
This is the program flow for sending commands to IBM cloud.

5.5 Adjusting User Interface

In order to display the parsed JSON data a Node-Red dashboard is created

Here we are using Gauges, text and button nodes to display in the UI and helps to monitor the parameters and control the farm equipment.

Below images are the Gauge, text and button node configurations



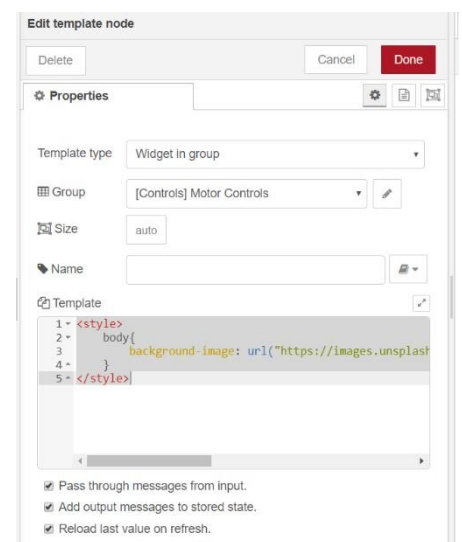
Adding Background image to the UI

To add the background image we are going to add template node and configure it with below HTML code

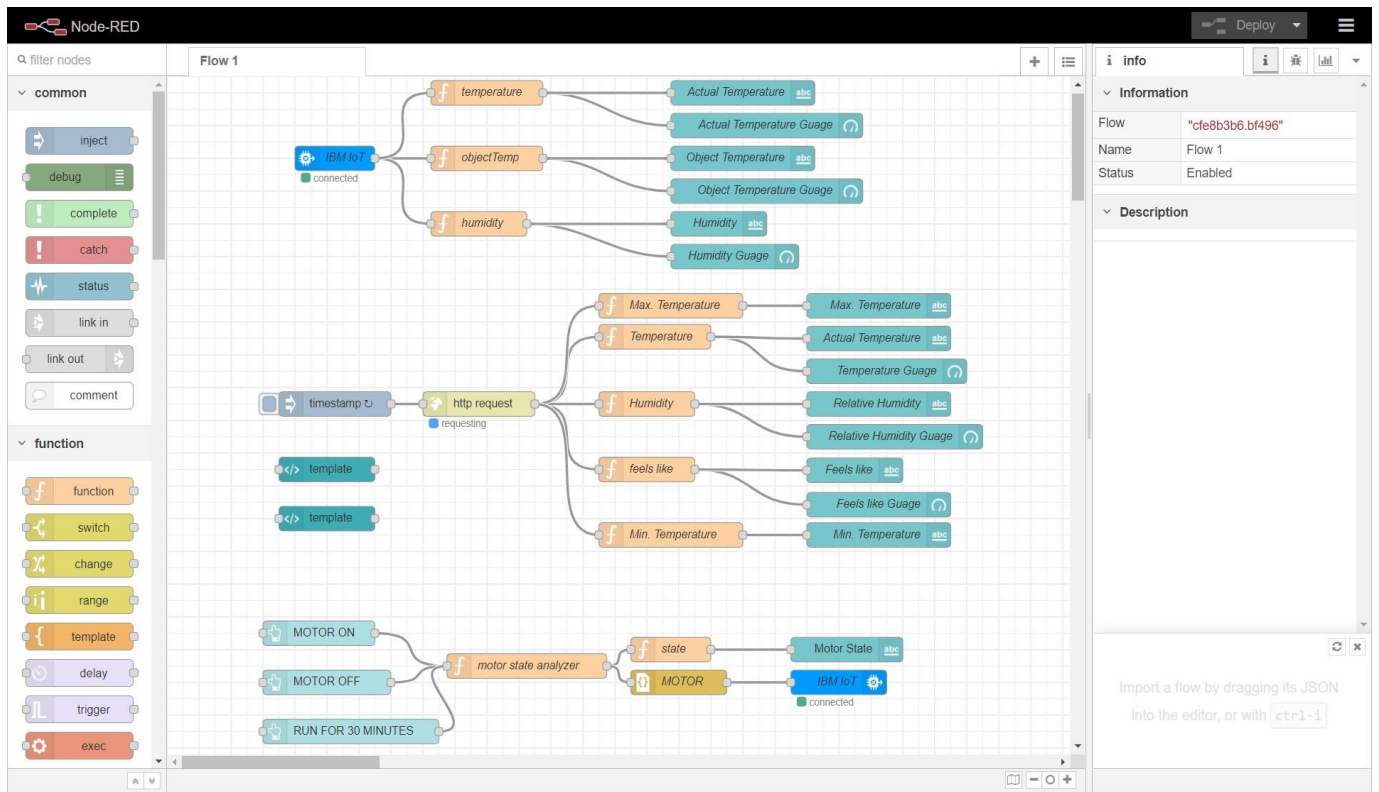
```
<style> body{ background-image: url("https://images.
  unsplash.com/photo-1563514227147-
  6d2ff665 a6a0?ixlib=rb-
  1.2.1&auto=format&fit=crop&w
  =1951&q=80");
}
```

We add URL of image that need to be displayed

Configuration is shown in the beside image

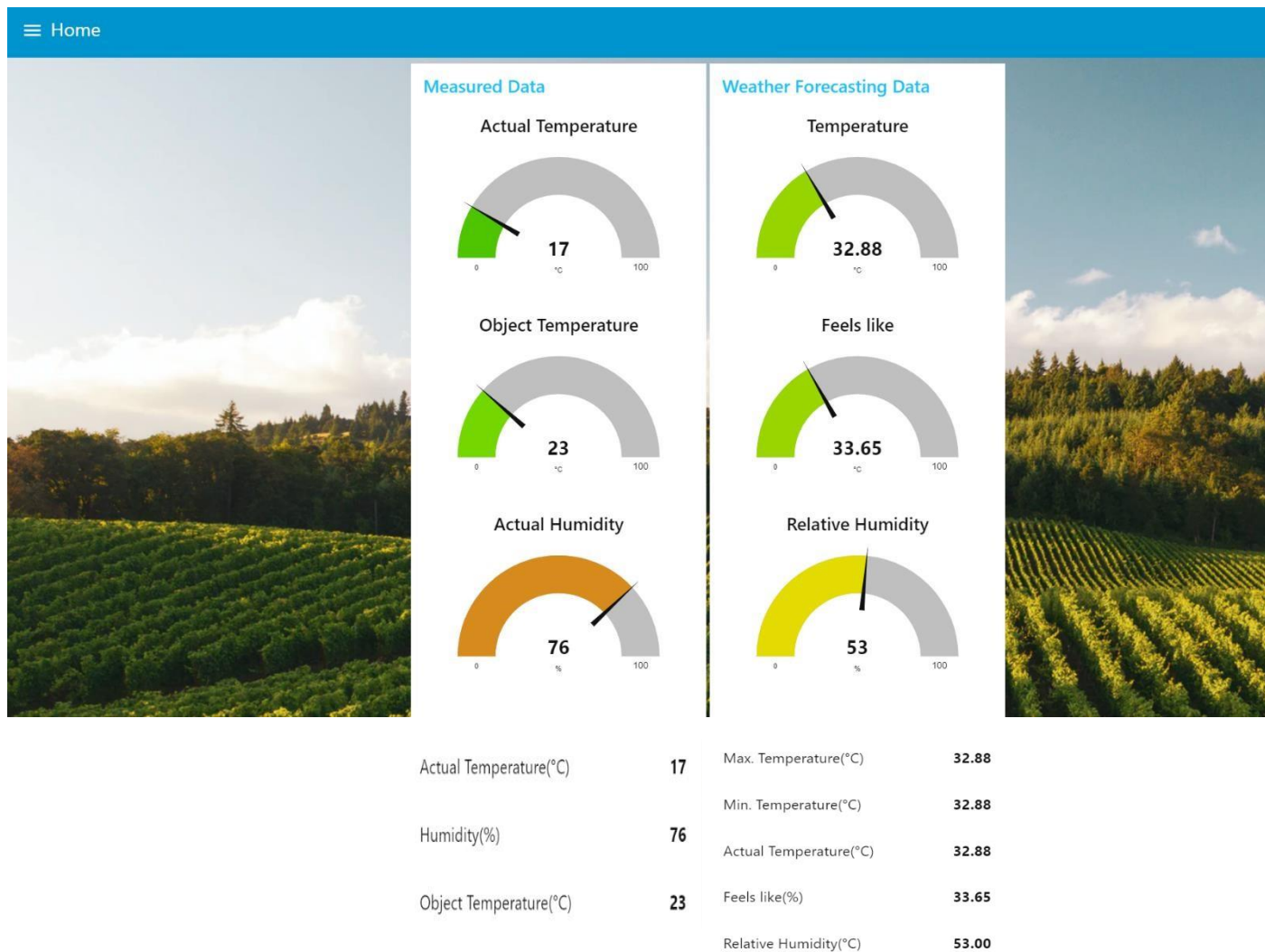


Complete Program Flow

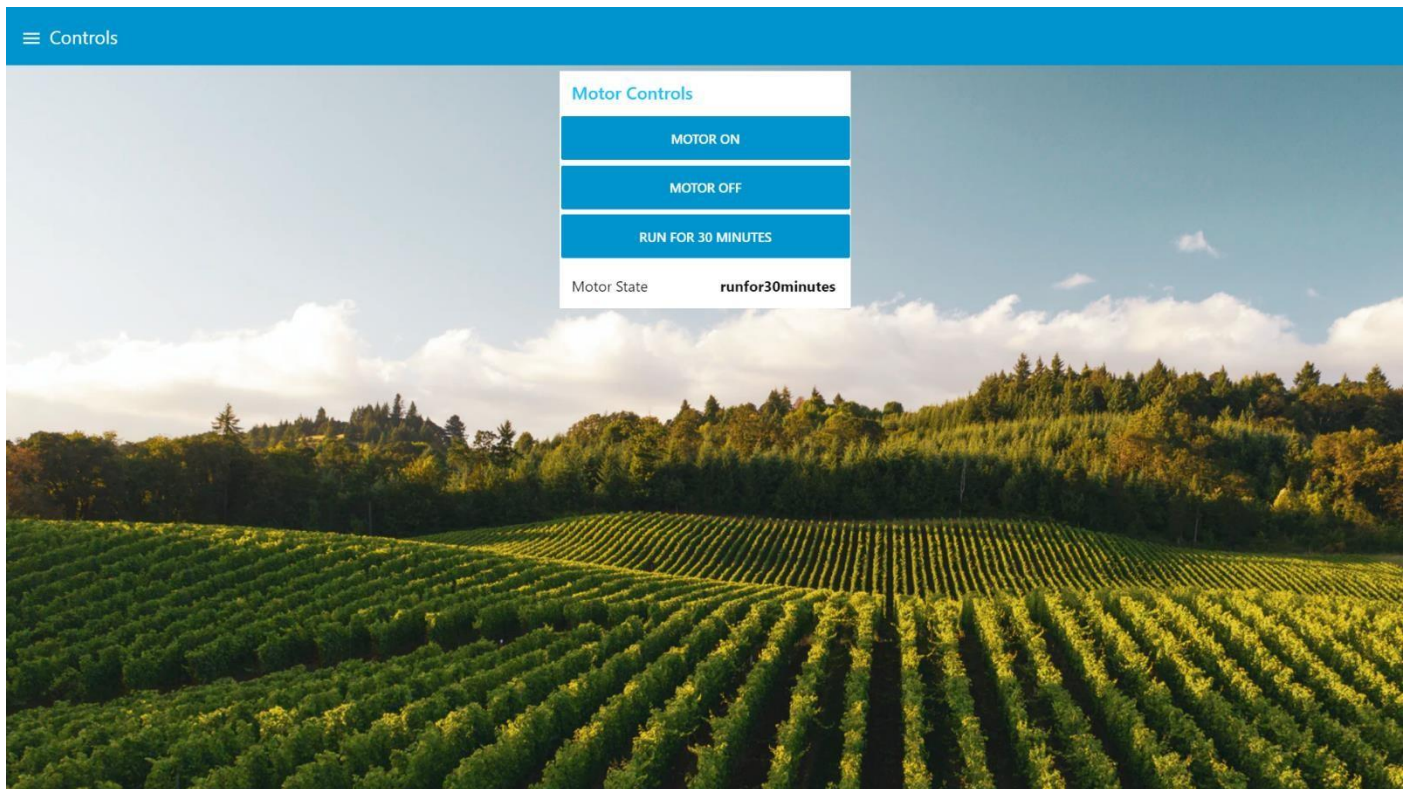


Web APP UI

Home Tab



Controls Tab



5.5 Receiving commands from IBM cloud using Python program

This is the Python code to receive commands from cloud to any device like Raspberry Pi in the farm

```
• import time import sys import
  ibmiotf.application import
  ibmiotf.device organization =
  "9wbx5m" deviceType =
  "iotdevice1" deviceId =
  "qwerty123" authMethod =
  "token" authToken =
  "johnyjohnnyespapa"
def myCommandCallback(cmd):
    print("Command received: %s" % cmd.data) if
    cmd.data['command']=='ON': print("MOTOR
    ON IS RECEIVED")
    time.sleep(1)
    print("MOTOR  STARTED") elif
    cmd.data['command']=='OFF':
    print("MOTOR OFF IS RECEIVED")
    time.sleep(1)
```



```
print("MOTOR STOPPED") elif
```

```
cmd.data['command']=='runfor30minutes':
```

```
print("MOTOR RUNS FOR 30 MINUTES") print("MOTOR  
STARTED")
```

```
for i in range(1,31):
```

```
print("%d minutes to stop"%(30-i)) # use time.sleep(60) for delay of one minute
```

```
print("MOTOR STOPPED")
```

```
try:
```

```
deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method":  
authMethod, "auth-token": authToken}
```

```
deviceCli = ibmiotf.device.Client(deviceOptions)
```

```
except Exception as e:
```

```
print("Caught exception connecting device: %s" % str(e))
```

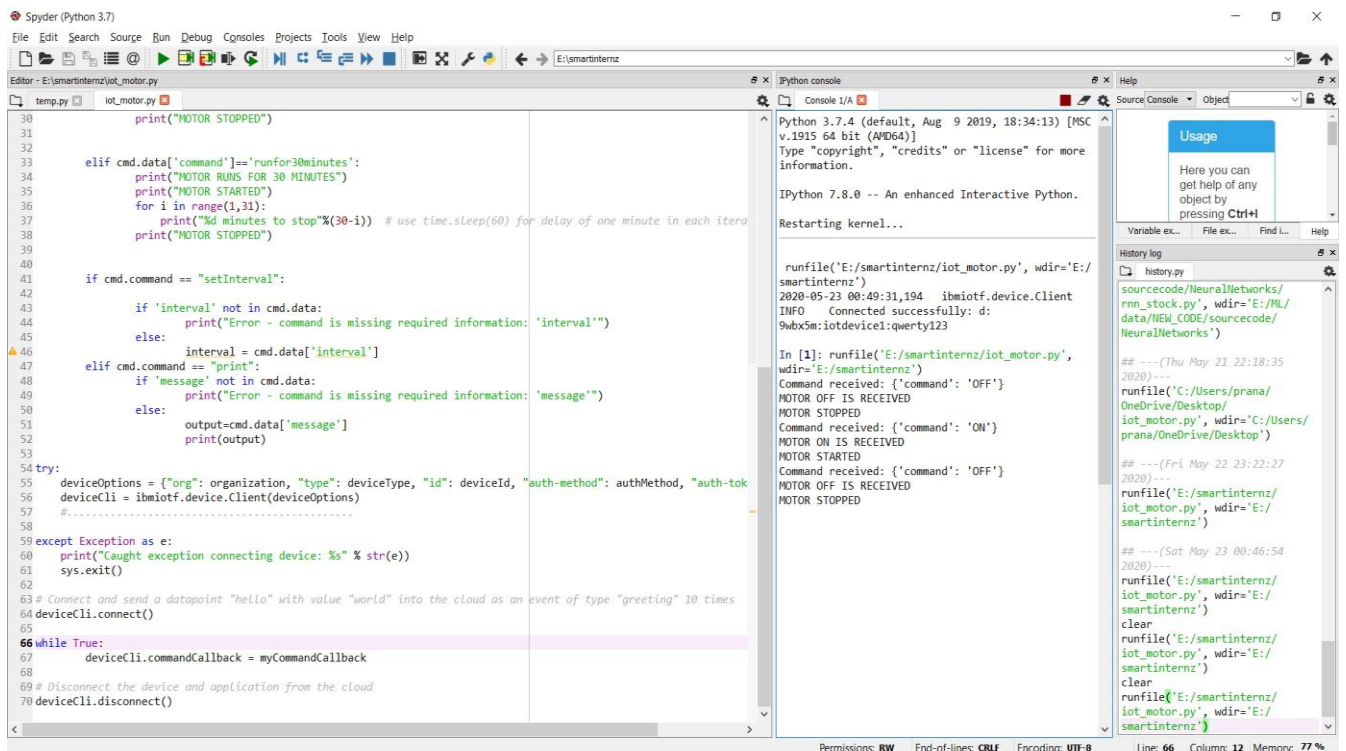
```
sys.exit()
```

```
deviceCli.connect()
```

```
while True:
```

```
deviceCli.commandCallback = myCommandCallback •
```

```
deviceCli.disconnect()
```



The screenshot shows the Spyder Python IDE interface. The left pane displays a Python script named `iot_motor.py` with the following code:

```
30 print("MOTOR STOPPED")
31
32 elif cmd.data['command']=='runfor30minutes':
33     print("MOTOR RUNS FOR 30 MINUTES")
34     print("MOTOR STARTED")
35     for i in range(1,31):
36         print("%d minutes to stop"%(30-i)) # use time.sleep(60) for delay of one minute in each itera
37         print("MOTOR STOPPED")
38
39
40
41 if cmd.command == "setInterval":
42
43     if 'interval' not in cmd.data:
44         print("Error - command is missing required information: 'interval'")
45     else:
46         interval = cmd.data['interval']
47 elif cmd.command == "print":
48     if 'message' not in cmd.data:
49         print("Error - command is missing required information: 'message'")
50     else:
51         output=cmd.data['message']
52         print(output)
53
54 try:
55     deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-tok
56     deviceCli = ibmiotf.device.Client(deviceOptions)
57     #.....
58
59 except Exception as e:
60     print("Caught exception connecting device: %s" % str(e))
61     sys.exit()
62
63 # Connect and send a datapoint "hello" with value "world" into the cloud as an event of type "greeting" 10 times
64 deviceCli.connect()
65
66 while True:
67     deviceCli.commandCallback = myCommandCallback
68
69 # Disconnect the device and application from the cloud
70 deviceCli.disconnect()
```

The right pane shows the Python console output, which includes the following text:

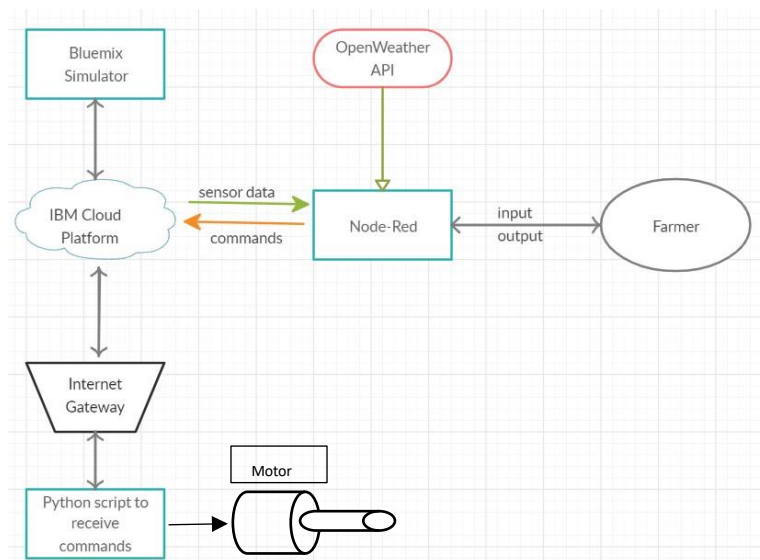
```
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC
v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more
information.
IPython 7.8.0 -- An enhanced Interactive Python.
Restarting kernel...

runfile('E:/smartinternz/iot_motor.py', wdir='E:/
smartinternz')
2020-05-23 00:49:31,194 ibmiotf.device.Client
INFO Connected successfully: d:
9wbx5m:iotdevice1:querty123

In [1]: runfile('E:/smartinternz/iot_motor.py',
wdir='E:/smartinternz')
Command received: {'command': 'OFF'}
MOTOR OFF IS RECEIVED
MOTOR STOPPED
Command received: {'command': 'ON'}
MOTOR ON IS RECEIVED
MOTOR STARTED
Command received: {'command': 'OFF'}
MOTOR OFF IS RECEIVED
MOTOR STOPPED
```

The bottom status bar indicates the file permissions are `RW`, end-of-lines are `CRLF`, encoding is `UTF-8`, and the current line is 66, column is 12, and memory usage is 77%.

6.Flow Chart



7.Observations & Results

```
IPython console
Console 1/A

In [3]: runfile('E:/smartinternz/iot_motor.py', wdir='E:/
smartinternz')
2020-05-22 23:25:53,710 ibmiotf.device.Client INFO
Connected successfully: d:9wbx5m:iotdevice1:qwerty123
Command received: {'command': 'ON'}
MOTOR ON IS RECEIVED
MOTOR STARTED
Command received: {'command': 'OFF'}
MOTOR OFF IS RECEIVED
MOTOR STOPPED
Command received: {'command': 'runfor30minutes'}
MOTOR RUNS FOR 30 MINUTES
MOTOR STARTED
29 minutes to stop
28 minutes to stop
27 minutes to stop
26 minutes to stop
25 minutes to stop
24 minutes to stop
23 minutes to stop
22 minutes to stop
21 minutes to stop
20 minutes to stop
19 minutes to stop
18 minutes to stop
17 minutes to stop
16 minutes to stop
15 minutes to stop
14 minutes to stop
13 minutes to stop
12 minutes to stop
11 minutes to stop
10 minutes to stop
9 minutes to stop
8 minutes to stop
7 minutes to stop
6 minutes to stop
5 minutes to stop
4 minutes to stop
3 minutes to stop
2 minutes to stop
1 minutes to stop
0 minutes to stop
MOTOR STOPPED
```

8. Advantages & Disadvantages

Advantages:

- Farms can be monitored and controlled remotely.
- Increase in convenience to farmers.
- Less labour cost.
- Better standards of living.

Disadvantages:

- Lack of internet/connectivity issues.
- Added cost of internet and internet gateway infrastructure.
- Farmers wanted to adapt the use of WebApp.

9. Conclusion

Thus the objective of the project to implement an IoT system in order to help farmers to control and monitor their farms has been implemented successfully.

10. Bibliography

IBM cloud reference: <https://cloud.ibm.com/>

Python code reference: <https://github.com/rachuriharish23/ibmsubscribe>

IoT simulator : <https://watson-iot-sensor-simulator.mybluemix.net/>

OpenWeather : <https://openweathermap.org/>