

# **SMART FARMER - IoT BASED SMART FARMING APPLICATION**

## **PROJECT REPORT**

**TEAM ID :** PNT2022TMID31713

**TEAM LEAD :** SAMRITHA S

**TEAM MEMBERS :** 1. AAKASH J  
2. DHEVAKI V  
3. JANANI S  
4. GOWTHAM S

# CONTENTS

1. Introduction
2. Problem Statement
3. Proposed Solution
4. Theoretical Analysis
  - 4.1 Required Software installation
    - 4.1.A Node-Red
    - 4.1.B IBM Watson IoT Platform
    - 4.1.C Python IDE
  - 4.2 IoT simulator
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 send message to mobile
  - 5.4 Configuring of Node -Red to send commands to IBM Cloud
  - 5.5 Adjusting User Interface
  - 5.6 Receiving Commands From Python Code
6. Observations & Results
7. Advantages & Disadvantages
9. Conclusion
10. Bibliography

## 1. Introduction

The main goal of this project is to assist farmers in automating their farms by granting them access to a Web App that allows them to remotely control equipment like water motors as well as other gadgets without even being physically present in the field while also monitoring field parameters like temperature, soil moisture, and humidity.

## 2. Problem Statement

Regardless of the weather, farmers must remain on site to maintain their farms. They need to personally check on the farm's condition and make sure the crops are getting enough water. To obtain a decent harvest, the farmer must spend the majority of his time on the field. They must toil in their fields and risk their lives to provide food for the nation during tough times, such as when a pandemic is present.

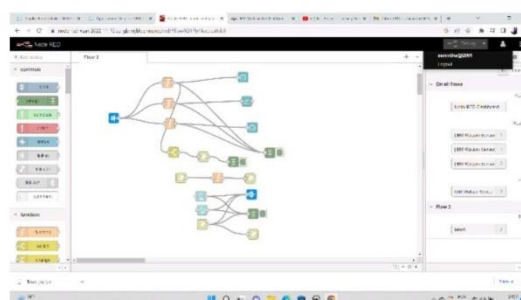
## 3. Proposed Solution

We bring IoT services to the farmer in order to enhance and facilitate his working circumstances. These services make it possible for the farmer to operate remotely using the internet and cloud services. Through a smartphone app, user may monitor the field's characteristics and manage the farm's equipment.

## 4. Theoretical Analysis

### 4.1 Block Diagram

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

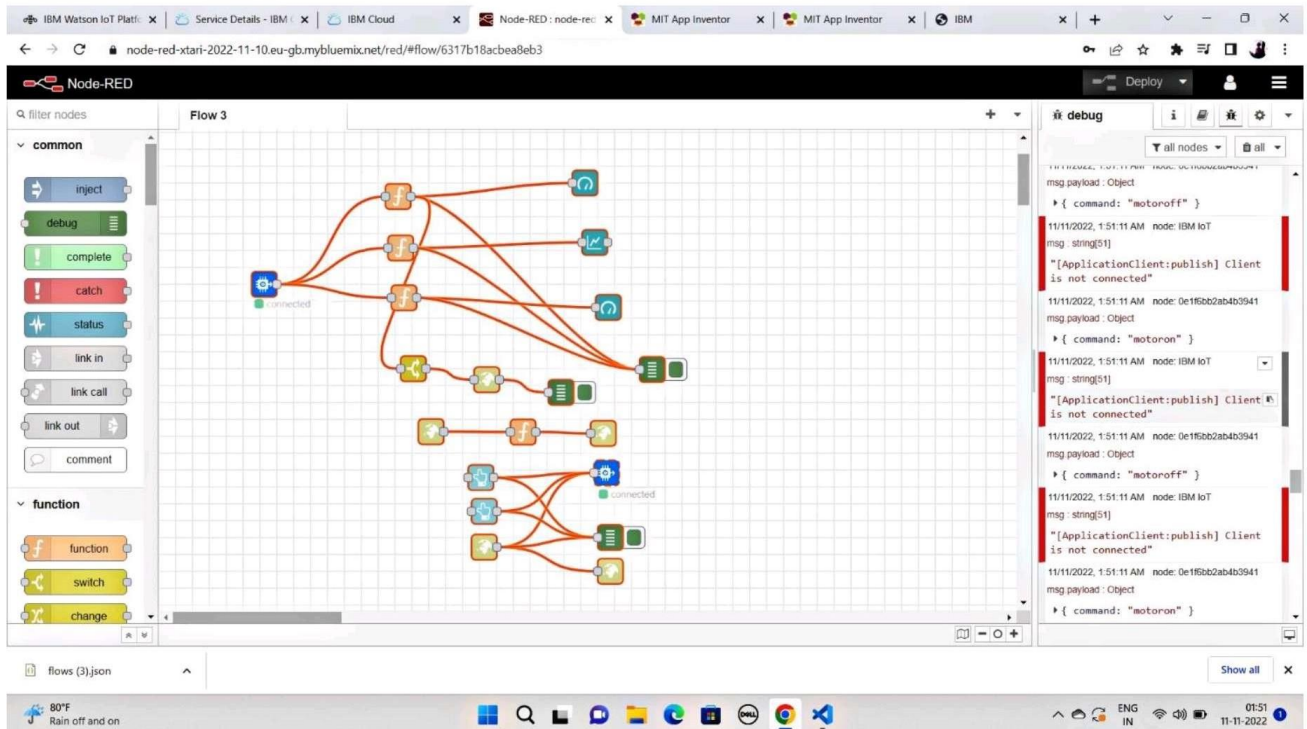


## 4.2 Required Software Installation

### 4.2.A Node-Red

Node-RED is a flow-based visual programming tool that was initially created by IBM for the Internet of Things to connect hardware components, APIs, and web services.

An online flow editor powered by Node-RED is available for developing JavaScript functions.



#### Installation:

- In the IBM Cloud Create a Node Red Service
- Also Create Internet of Things Service

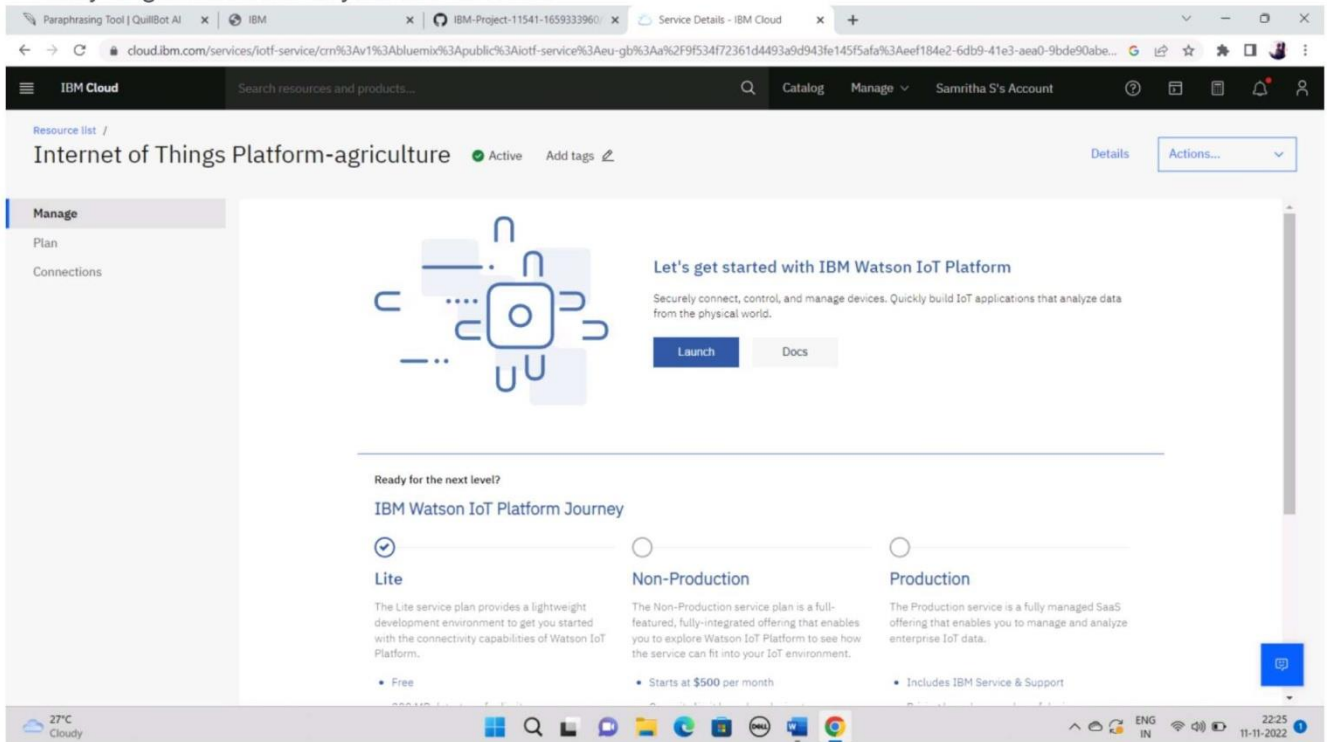
#### Installation of IBM IoT and Dashboard nodes for Node-Red

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

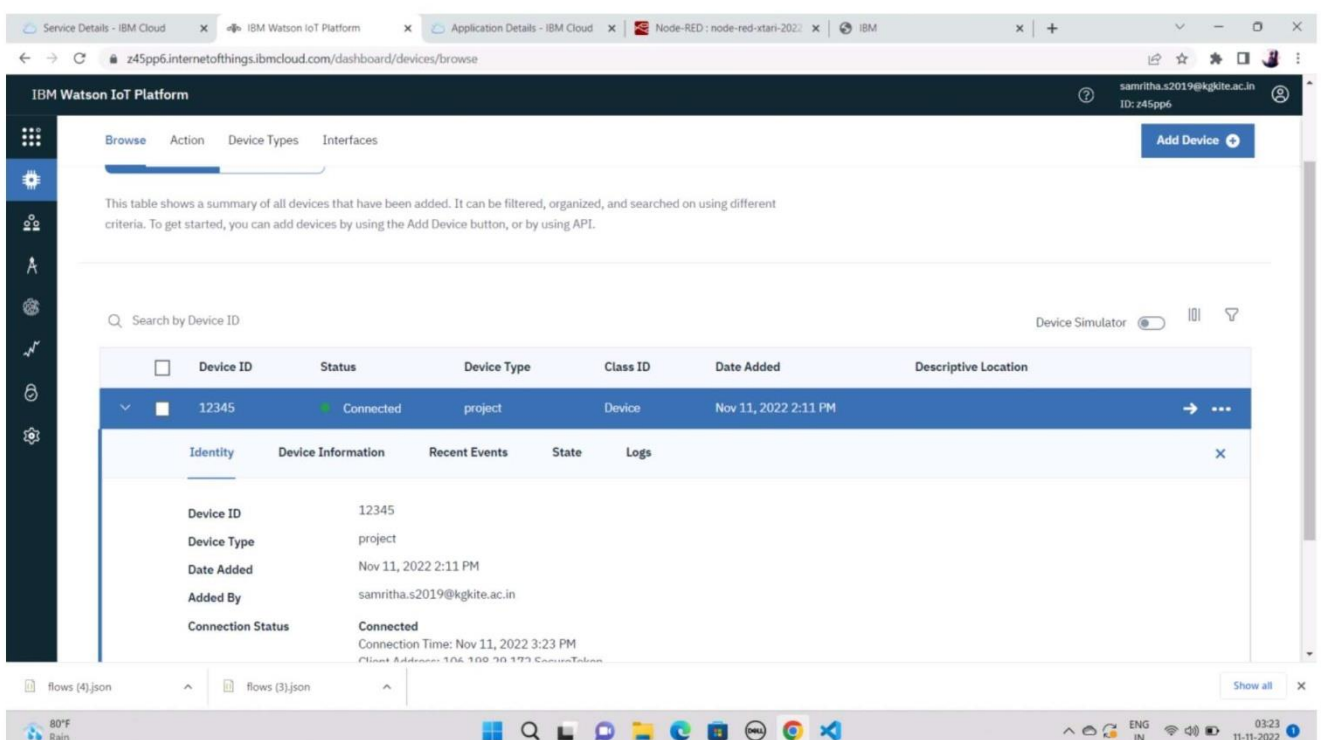
## 4.2.B IBM Watson IoT Platform

A fully managed, cloud-hosted solution that has data storage, communication, control, and device registration features. The managed, cloud-hosted IBM Watson IoT Platform is a solution that makes it easy to get value out of your IoT devices.



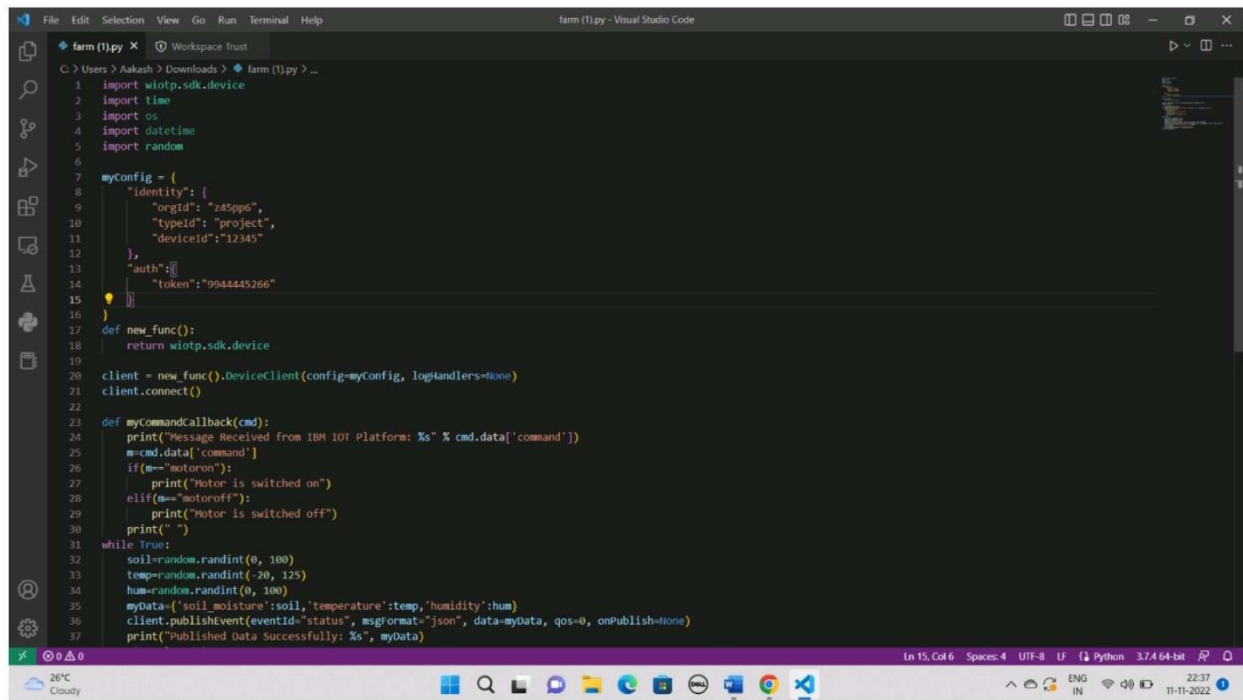
### Steps to configure:

- 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 VS Code



```
1 import wiotp.sdk.device
2 import time
3 import os
4 import datetime
5 import random
6
7 myConfig = {
8     "identity": {
9         "orgid": "z45pp6",
10        "typeid": "project",
11        "deviceId": "12345"
12    },
13    "auth": {
14        "token": "9944445266"
15    }
16 }
17
18 def new_func():
19     return wiotp.sdk.device
20
21 client = new_func().DeviceClient(config=myConfig, loghandlers=None)
22 client.connect()
23
24 def myCommandCallback(cmd):
25     print("Message Received from IBM IoT Platform: %s" % cmd.data['command'])
26     m=cmd.data['command']
27     if(m=="motoron"):
28         print("Motor is switched on")
29     elif(m=="motoreff"):
30         print("Motor is switched off")
31     print(" ")
32
33 while True:
34     soil=random.randint(0, 100)
35     temp=random.randint(-20, 125)
36     hum=random.randint(0, 100)
37     myData={ 'soil moisture':soil, 'temperature':temp, 'humidity':hum}
38     client.publish(eventid="status", msgformat="json", data=myData, qos=0, onPublish=None)
39     print("Published Data Successfully: %s" % myData)
```

## 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://node-red-xtari-2022-11-10.eu-gb.mybluemix.net/>

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

## 5. Building Project

### 5.1 Connecting IoT Simulator to IBM Watson IoT Platform

Give the credentials of your device in IBM Watson IoT Platform

Click on connect

My credentials given to simulator are:

OrgID: z45pp6	api: a-2tpzx1-kg8gomgycx
Device type: project	token:2GBxXdoL6bLYO58svl
Device ID : 12345	
Device Token : 9944445266	

- 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": "project",
    "soil_moisture": 32,
    "temperature": 11,
    "humidity": 25
  }
}

```

The screenshot displays the IBM Watson IoT Platform dashboard. The top navigation bar includes tabs for 'Service Details - IBM Cloud', 'IBM Watson IoT Platform', 'Application Details - IBM Cloud', 'Node-RED : node-red-xtari-202...', and 'IBM'. The main header shows the user 'samritha.s2019@kgkite.ac.in' with ID 'z45pp6'. The left sidebar contains icons for various platform features.

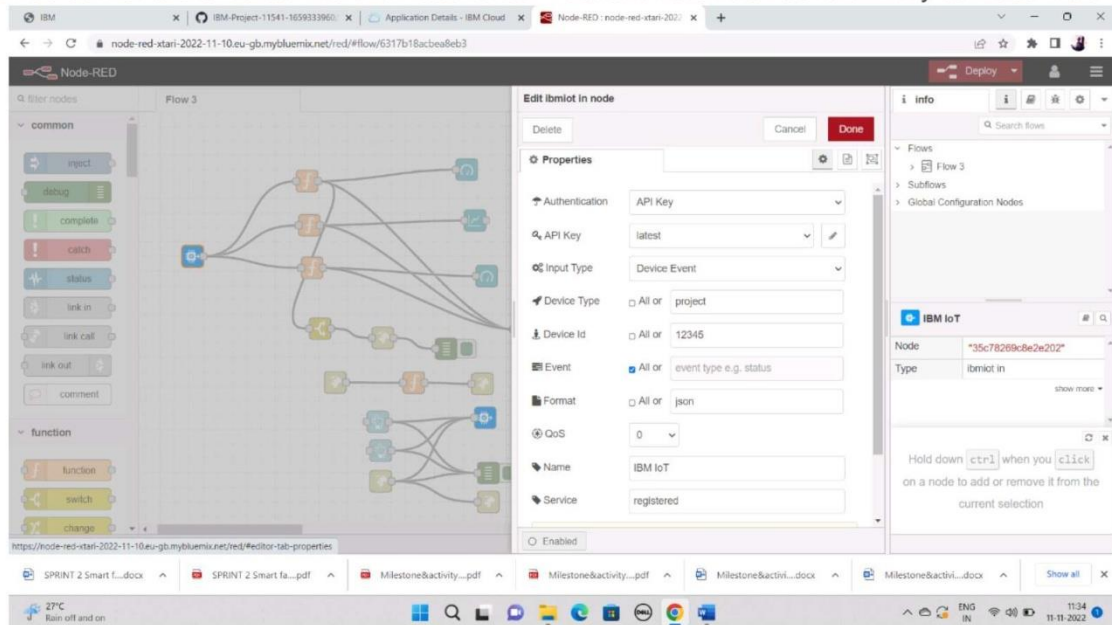
The main content area is titled 'Browse' and includes a search bar 'Search by Device ID'. A table lists devices, with the first device having ID '12345', status 'Connected', device type 'project', class ID 'Device', and date added 'Nov 11, 2022 2:11 PM'. Below the table, the 'Recent Events' tab is selected, showing a live stream of data. The events table has columns for Event, Value, Format, and Last Received.

Event	Value	Format	Last Received
status	{"soil_moisture":32,"temperature":11,"humidity":...	json	a few seconds ago
status	{"soil_moisture":16,"temperature":50,"humidity":...	json	a few seconds ago
status	{"soil_moisture":93,"temperature":14,"humidity":...	json	a few seconds ago

The bottom of the screen shows a Windows taskbar with the date '03:24 11-11-2022' and system status icons.

## 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:

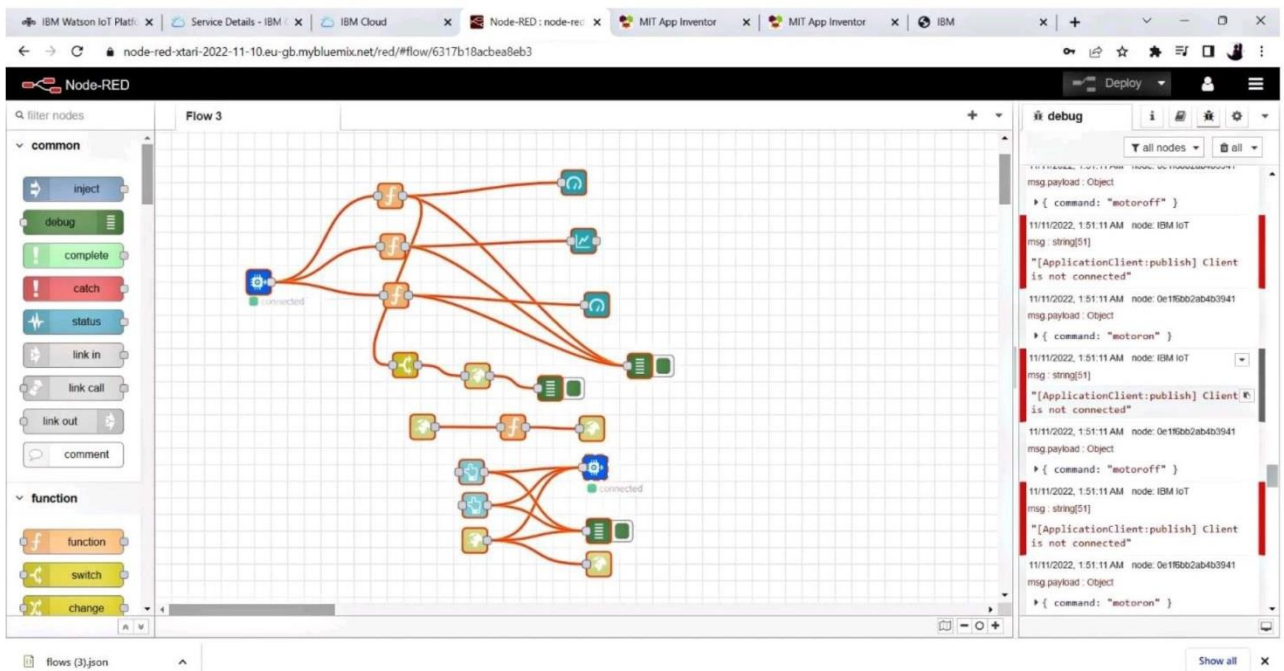
```
global.set('moist',msg.payload,soil_moisture)
```

```
msg.payload=msg.payload.soil_moisture
```

```
return msg;
```

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





Nodes connected in following manner to get each reading separately

**Edit function node**

Delete

Cancel

Done

Properties

Name

temperature

Function

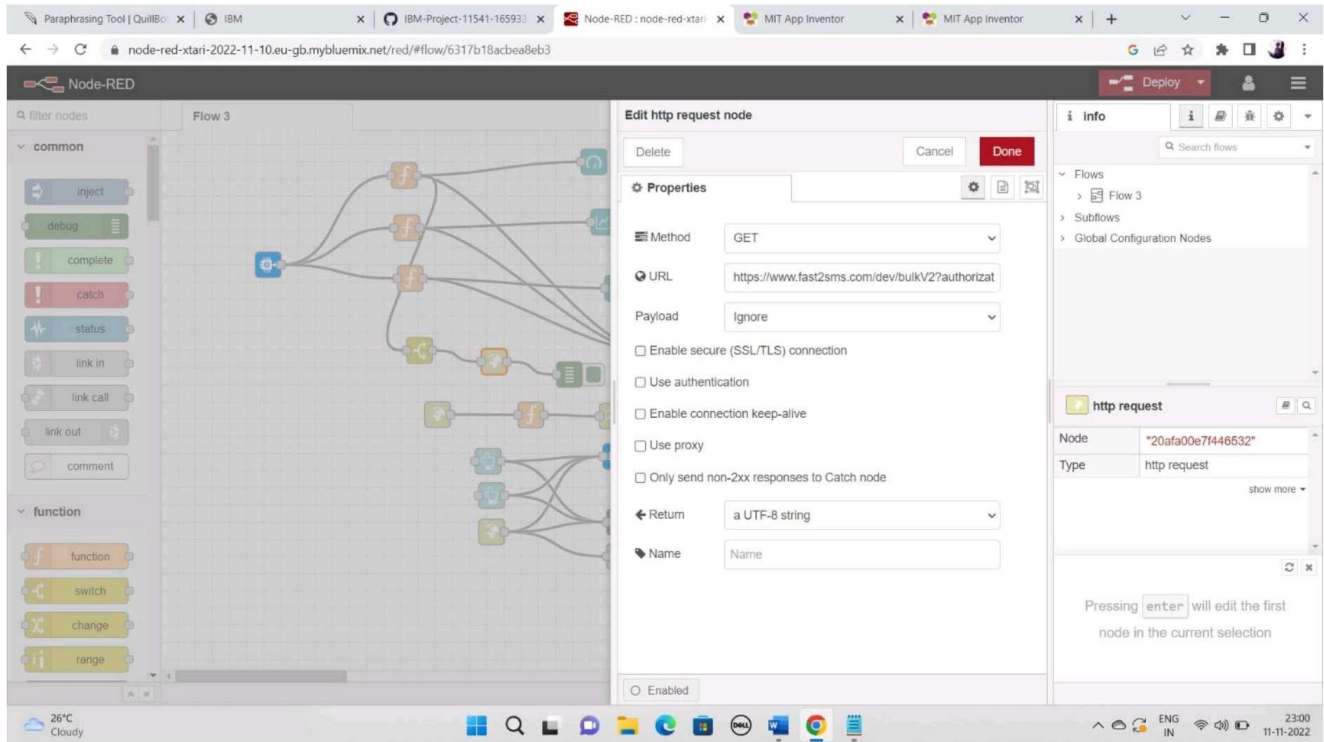
```
1 msg.payload=msg.payload.d.temperature  
2 return msg;
```

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

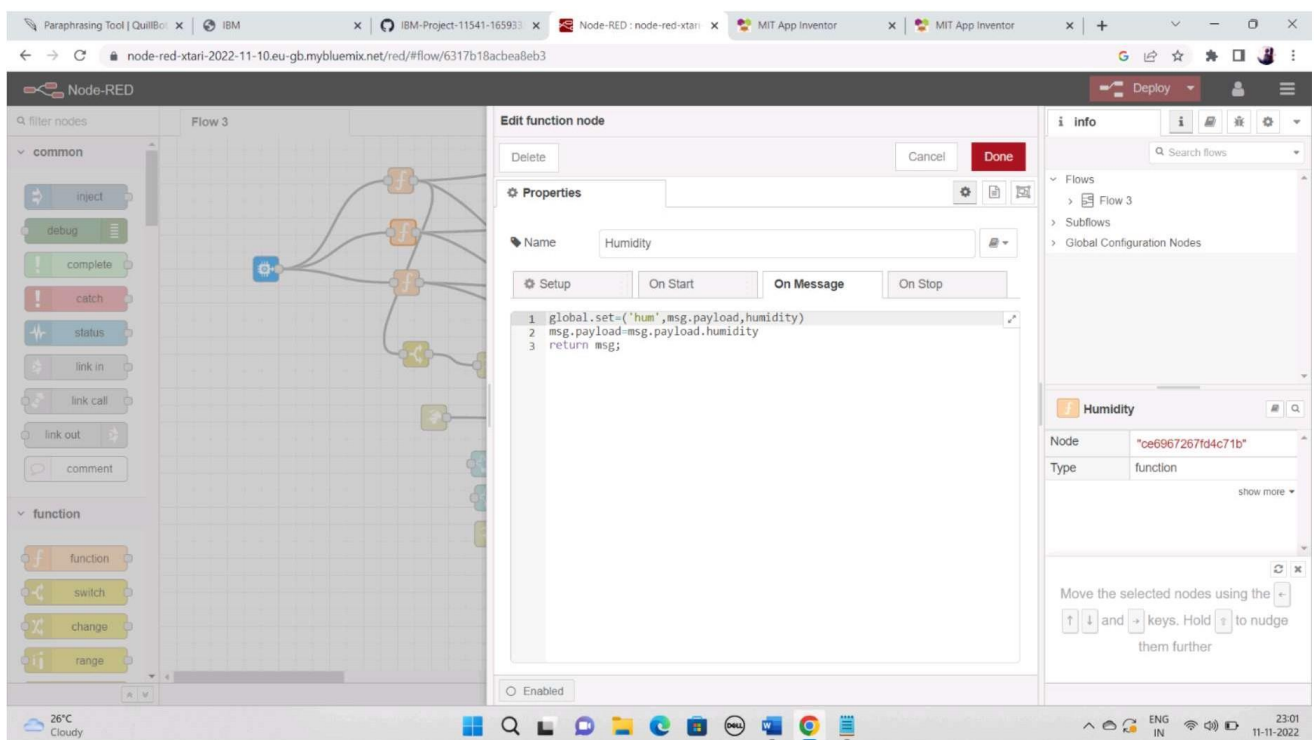
## 5.3 Configuration of Node-Red to send message to mobile

The Node-Red also receive data from the Fast To SMS 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

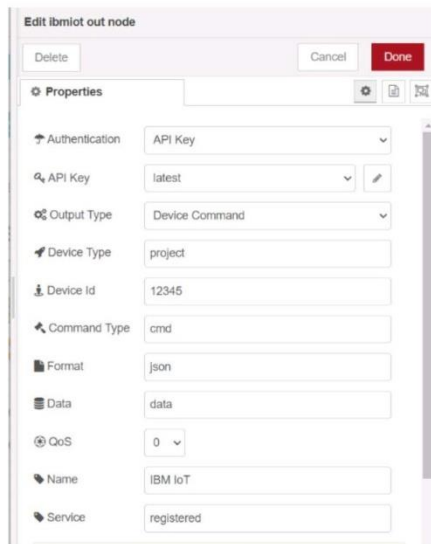


## Configuring The Node



## 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.



The screenshot shows the 'Edit ibmiot out node' configuration window. At the top, there are buttons for 'Delete', 'Cancel', and 'Done'. Below this is a 'Properties' tab with a settings icon. The configuration fields are as follows:

- Authentication: API Key (dropdown)
- API Key: latest (dropdown with edit icon)
- Output Type: Device Command (dropdown)
- Device Type: project (text input)
- Device Id: 12345 (text input)
- Command Type: cmd (text input)
- Format: json (text input)
- Data: data (text input)
- QoS: 0 (dropdown)
- Name: IBM IoT (text input)
- Service: registered (text input)

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

0 -> for motor off

1 -> for motor on

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

```
{"command":"motoron"}
```



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.

Edit gauge node

Delete
Cancel
Done

Properties

Group
[Home] Measured Data
Size
auto
Type
Gauge
Label
Actual Temperature
Value format
{{value}}
Units
°C
Range
min 0 max 100
Colour gradient
Sectors
0 optional optional 100
Name
Actual Temperature Gauge

Edit text node

Delete
Cancel
Done

Properties

Group
[Home] Measured Data
Size
auto
Label
Actual Temperature(°C)
Value format
{{msg.payload}}
Layout

label value

label value

label value

label value

label value

Name
Actual Temperature

Edit button node

Delete
Cancel
Done

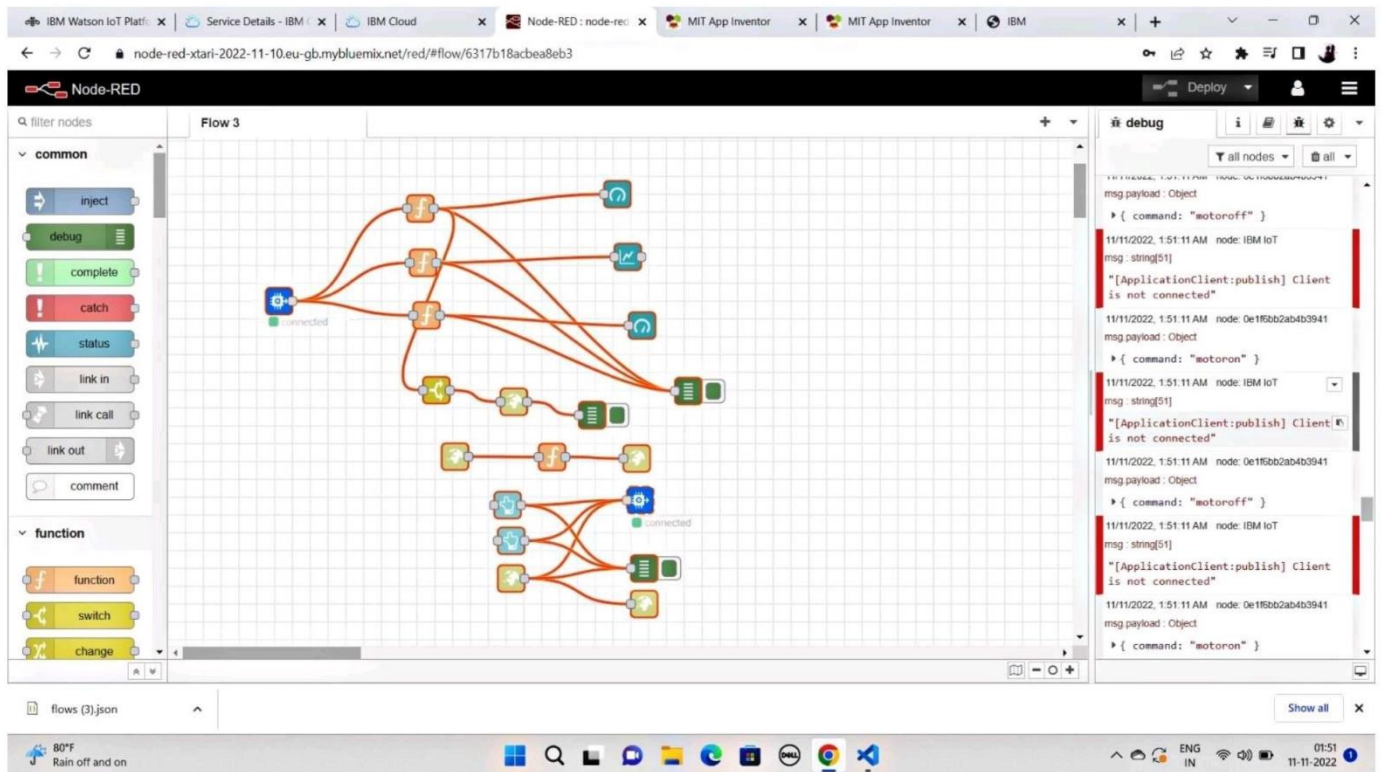
Properties

Group
[Controls] Motor Controls
Size
auto
Icon
optional icon
Label
MOTOR ON
Tooltip
optional tooltip
Colour
optional text/icon color
Background
optional background color
When clicked, send:

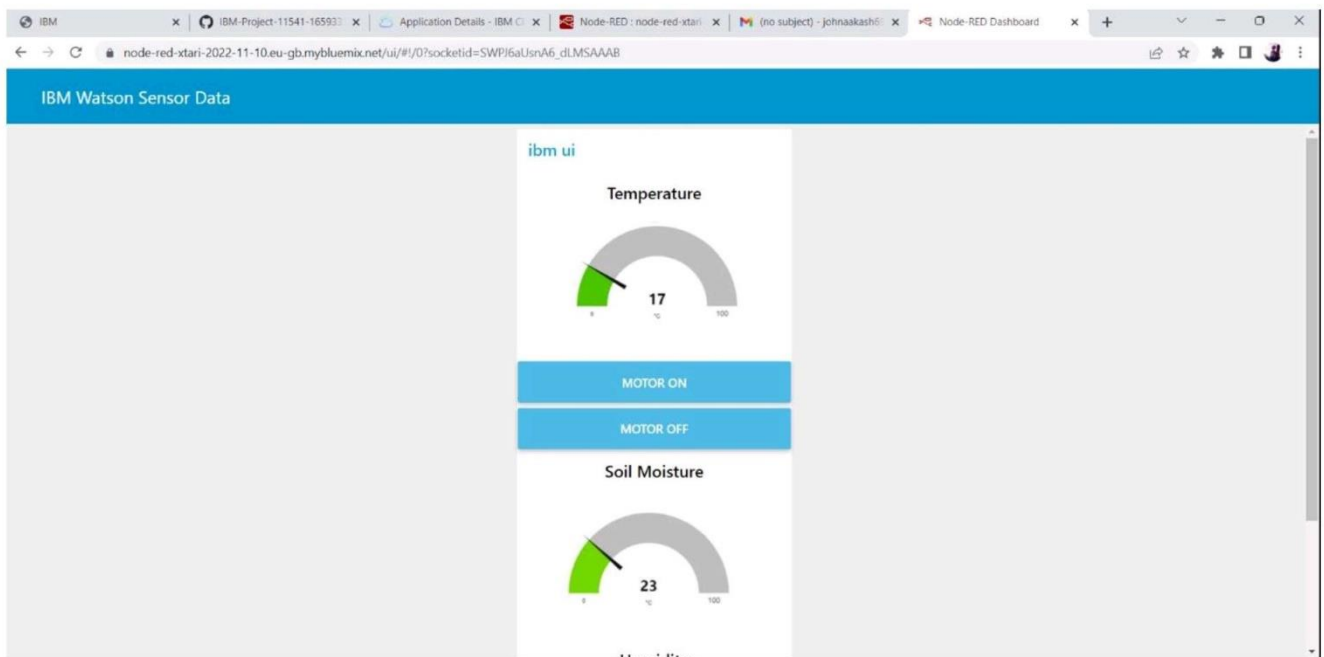
Payload
1
Topic

If msg arrives on input, emulate a button click:

## Complete Program Flow



## Web APP UI



## Control Tab





## 5.6 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 wiotp.sdk.device
import time
import os
import datetime
import random

myConfig = {
    "identity": {
        "orgId": "z45pp6",
        "typeId": "project",
        "deviceId": "12345"
    },
    "auth": {
        "token": "99444445266"
    }
}

def new_func():
    return wiotp.sdk.device

client = new_func().DeviceClient(config=myConfig, logHandlers=None)
client.connect()

def myCommandCallback(cmd):
    print("Message Received from IBM IOT Platform: %s" % cmd.data['command'])
    m=cmd.data['command']
    if(m=="motoron"):
        print("Motor is switched on")
    elif(m=="motoroff"):
        print("Motor is switched off")
    print(" ")
while True:
    soil=random.randint(0, 100)
    temp=random.randint(-20, 125)
    hum=random.randint(0, 100)
    myData={'soil_moisture':soil,'temperature':temp,'humidity':hum}
    client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
    print("Published Data Successfully: %s", myData)
    time.sleep(2)
    client.myCommandCallback = myCommandCallback
client.disconnect()
```



```
1 import wiotp.sdk.device
2 import time
3 import os
4 import datetime
5 import random
6
7 myConfig = {
8     "identity": {
9         "orgId": "z45pp6",
10        "typeId": "project",
11        "deviceId": "12345"
12    },
13    "auth": {
14        "token": "99444445266"
15    }
16 }
17 client = wiotp.sdk.device.DeviceClient(config=myConfig, loghandlers=None)
18 client.connect()
19
20 def myCommandCallback(cmd):
21     print("Message Received from IBM IoT Platform: %s" % cmd.data['command'])
22     m=cmd.data['command']
23     if(m=="motoron"):
24         print("Motor is switched on")
25     elif(m=="motoroff"):
26         print("Motor is switched off")
27     print(" ")
28 while True:
29     soil=random.randint(0, 100)
30     temp=random.randint(-20, 125)
31     hum=random.randint(0, 100)
32     myData={'soil_moisture':soil,'temperature':temp,'humidity':hum}
33     client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
34     print("Published Data Successfully: %s", myData)
35     time.sleep(2)
36     client.myCommandCallback = myCommandCallback
37 client.disconnect()
```

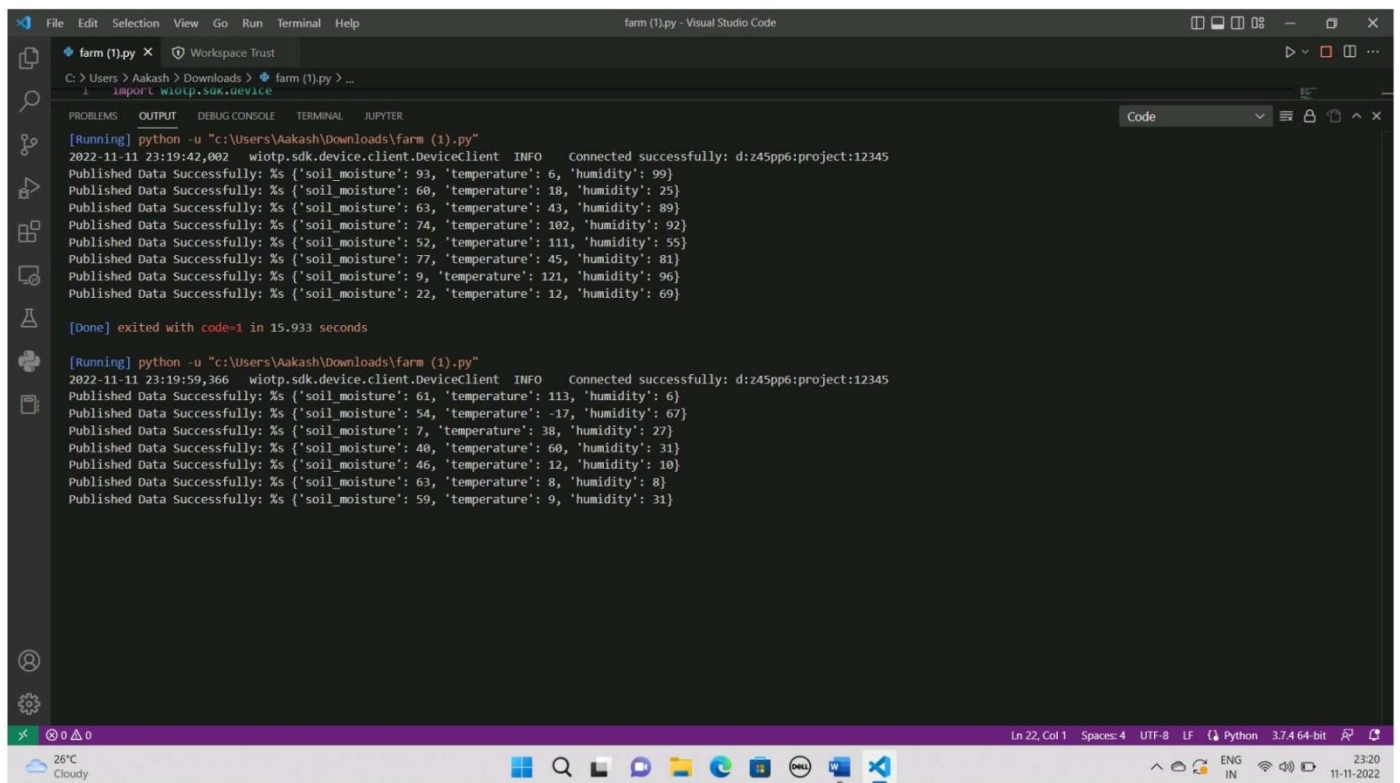
Out Put:

```
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Aakash> & C:/python/python37/python.exe "c:/Users/Aakash/Downloads/farm (1).py"
2022-11-11 02:38:48,112 wiotp.sdk.device.client.DeviceClient INFO Connected successfully: d:z45pp6:project:12345
Published Data Successfully: %s {\'soil_moisture\': 43, \'temperature\': 79, \'humidity\': 90}
```

## 6.Observations & Results



```
farm (1).py X Workspace Trust
C:\Users\Aakash\Downloads> farm (1).py > ...
  1 import wiotp.sdk.device

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Code
[Running] python -u "c:\Users\Aakash\Downloads\farm (1).py"
2022-11-11 23:19:42,002 wiotp.sdk.device.client.DeviceClient INFO Connected successfully: d:z45pp6:project:12345
Published Data Successfully: %s {'soil_moisture': 93, 'temperature': 6, 'humidity': 99}
Published Data Successfully: %s {'soil_moisture': 60, 'temperature': 18, 'humidity': 25}
Published Data Successfully: %s {'soil_moisture': 63, 'temperature': 43, 'humidity': 89}
Published Data Successfully: %s {'soil_moisture': 74, 'temperature': 102, 'humidity': 92}
Published Data Successfully: %s {'soil_moisture': 52, 'temperature': 111, 'humidity': 55}
Published Data Successfully: %s {'soil_moisture': 77, 'temperature': 45, 'humidity': 81}
Published Data Successfully: %s {'soil_moisture': 9, 'temperature': 121, 'humidity': 96}
Published Data Successfully: %s {'soil_moisture': 22, 'temperature': 12, 'humidity': 69}

[Done] exited with code=1 in 15.933 seconds

[Running] python -u "c:\Users\Aakash\Downloads\farm (1).py"
2022-11-11 23:19:59,366 wiotp.sdk.device.client.DeviceClient INFO Connected successfully: d:z45pp6:project:12345
Published Data Successfully: %s {'soil_moisture': 61, 'temperature': 113, 'humidity': 6}
Published Data Successfully: %s {'soil_moisture': 54, 'temperature': -17, 'humidity': 67}
Published Data Successfully: %s {'soil_moisture': 7, 'temperature': 38, 'humidity': 27}
Published Data Successfully: %s {'soil_moisture': 40, 'temperature': 60, 'humidity': 31}
Published Data Successfully: %s {'soil_moisture': 46, 'temperature': 12, 'humidity': 10}
Published Data Successfully: %s {'soil_moisture': 63, 'temperature': 8, 'humidity': 8}
Published Data Successfully: %s {'soil_moisture': 59, 'temperature': 9, 'humidity': 31}
```

Ln 22, Col 1 Spaces: 4 UTF-8 LF Python 3.7.4 64-bit

26°C Cloudy 23:20 11-11-2022

## 7. Advantages & Disadvantages

### Advantages:

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

### Disadvantages:

- Lack of internet/connectivity issues.
- Added cost of internet and internet gateway infrastructure.
- Farmers wanted to adapt the technology.
- Less Accuracy

## 8. Conclusion

Thus, the project's goal of putting in place an IoT system to assist farmers in managing and monitoring their fields has been accomplished.

## 9. 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/>

Fast To SMS: <https://fast2sms.com/dashboard/dev-api>