Project Report

Since 1962

IOT based Weather Monitoring System using MQTT protocol

Submitted by

Saurav Bhosale (Seat No: IN6A005)
Abhijeet Gaonkar (Seat No: IN6A011)
Prabhat Jaiswar (Seat No: IN6A015)
Manali Ganacharya (Seat No: IN6A057)

Class and div: D13-INST

Submitted to
Prof. Deepti Khimani

Vivekanand Education Society's Institute of Technology
Chembur, Mumbai – 400 074

May 2021

Declaration

We Saurav Bhosale (IN6A005), Abhijeet Gaonkar (IN6A011), Prabhat Jaiswar (IN6A015), Manali Ganacharya (IN6A057) students of Semester VI, third year Engineering from Department of Instrumentation, V.E.S. Institute of Technology, hereby declare that the project titled IOT based Weather Monitoring System using MQTT protocol, submitted to the University of Mumbai for the academic year 2020-2021, is a record of an original work done by us under the guidance of Prof. Deepti Khimani, Department of Instrumentation , V.E.S. Institute of Technology, Mumbai -400074.

This project report is submitted as a theoretical background for the fulfillment of the requirement of the Instrumentation Mini Project in Semester VI.

Names: Saurav Bhosale, Abhijeet Gaonkar, Prabhat Jaiswar, Manali Ganacharya
Signatures:
Date: 28/05/2021
Place: VESIT, Chembur

CERTIFICATE

This is to certify that the report entitled

IOT based Weather Monitoring System using MQTT Protocol

that is being submitted by Saurav Bhosale (IN6A005), Abhijeet Gaonkar (IN6A011), Prabhat Jaiswar (IN6A015), Manali Ganacharya (IN6A057) from D13-Instrumentation, V.E.S Institute of Technology,

In fulfillment of the requirement of the Instrumentation Mini Project in Semester VI to the University of Mumbai is a record of bonafide work carried out by them under my guidance and supervision.

Prof. Deepti Khimani

Department of Instrumentation

V.E.S. Institute of Technology

Mumbai -74.

INDEX:

ABSTRACT:

In this project we have made use of hardware components along with software. Hardware components consist of DHT11 sensor module(Temperature and humidity sensor), NodeMCU ESP8266 module(Microcontroller), breadboard, jumper wires, USB 2.0 cable(For power connection). Software used are Arduino IDE(to program ESP8266 module), AWS ubuntu instance with node-red and mosquitto broker and MQTT box. We collected Temperature and humidity data from 2 DHT11 hardware setup which was set on 2 distinct locations. This data from DHT11 was sent to the AWS MQTT server by the nodeMCU ESP8266 module using the MQTT protocol. This data was then visualized on a node-red dashboard in an aesthetic way. This data was also displayed on the MQTT box for ensuring proper MQTT connection. The alert was sent to email using the IFTTT service.

INTRODUCTION:

Conventional Weather Monitoring System:

The significance of weather monitoring exists in numerous perspectives. Weather affects a wide range of man's activities, including agriculture, transportation, industries, and leisure time. Man wants to stay updated about the latest weather conditions of any place like a college campus or any other particular building. Since the world is changing so fast there should be easily ascible weather stations.Existing weather monitoring generally uses weather stations that use multiple instruments such as thermometers, barometers, wind vanes, rain gauge, etc. to measure weather and climate changes. Most of these instruments use simple analog technology which is later physically recorded and stored in a database. This information is later sent to news reporting stations and radio stations where the weather report is given.

Limitations of the existing system are:

1. Existing weather monitoring systems that are used in the field generally consist of unconventional and heavy machinery.
2. Power requirements are one of many major constraints these instruments are generally cited far from the main power supply.
3. Data that is collected by the instruments needs to be manually transferred from the logger to a laptop or computer via a cable.

Smart Weather Monitoring System:
Alternative for this conventional weather monitoring system is Smart Weather Monitoring System which makes use of IoT. There are multiple sensors commercially available that can work in the 'Smart Weather Monitoring and Real Time Alert system using IoT' application. Existing IoT based systems utilize sensors such as DHT11 for temperature and humidity measurement, anemometer for wind speed measurement, LDR for light intensity monitoring, UV radiation measurement using GY8511 solar sensor, Carbon monoxide level monitoring using MQ7, soil moisture measurement using Hygrometer, ultrasonic sensor for level measurement of rain water, raindrop sensor for detecting rainfall/snowfall and Arduino microcontroller based cyber-physical interface. The combination of these components creates a robust alert system (for dealing with emergency scenarios) and data logging mechanism which ensures that the alerts are not lost over time when left unattended. Smart weather monitoring system allow a significant advantage over their conventional counterparts in terms of portability, ease of installation and monitoring, allowing complex weather evaluations/predictions, size, compactness, decreased power requirements, ease of interfacing with APIs using Raspberry Pi, creation of an app that links with the physical system and allows linking the system to a web page that logs data allowing its graphical assessment.

Internet of Things(IoT):
A term first coined by Kevin Ashton in 1999, IoT (a shorthand for Internet of Things) is an interconnected network of devices that allows for the development of a communication channel between physical devices that are placed a million miles apart. The concept of IoT uses physical devices, embedded systems, communication protocols like MQTT, softwares such as Node-RED and AWS to provide a global IT infrastructure.
To implement IoT we need a sender physical device that continuously transmits signals, an application programming interface like Node-Red and cloud software like Microsoft Azure or AWS. The cloud software and application programming interface form a communication channel and the receiver device(a remotely placed system of computers that form the cloud).

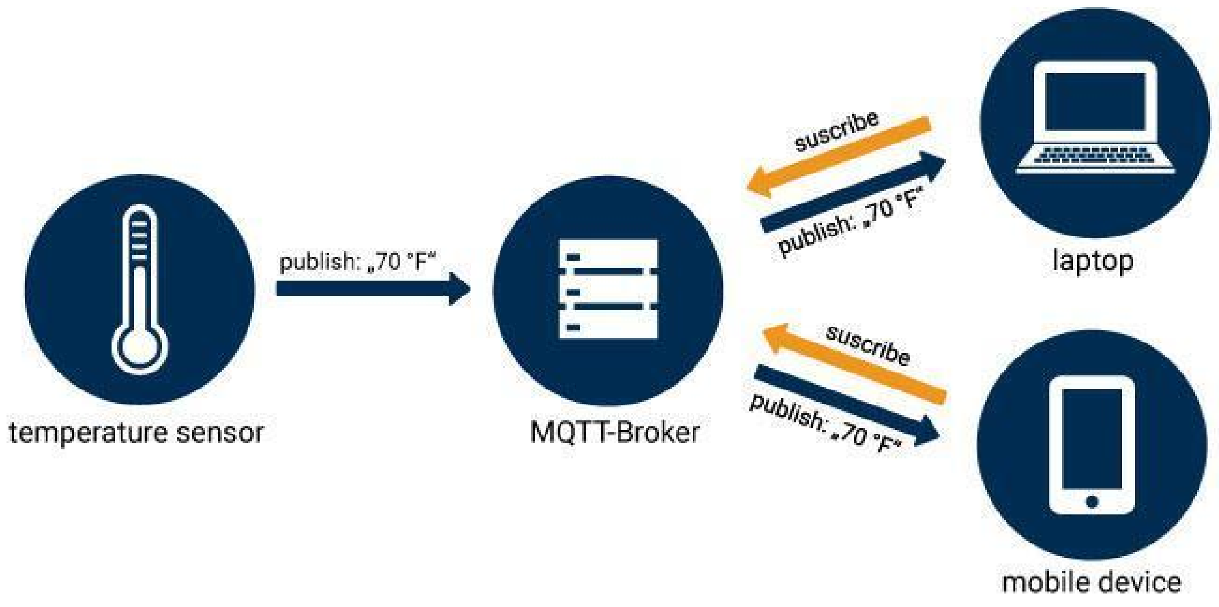
A plethora of communication protocols in IoT:

The defining features of IoT is the interconnection of and communication between everyday machines and devices. These features are made possible by the use of various radio technologies such as RFID, WLAN(802.11),WPAN(IEEE 802.15) and WMAN(IEEE 802.16) for communications at the base level. The radio technologies in tandem with the physical machines and devices form the Machine-to-Machine (M2M) network. However, the M2M cannot be fully established without the end devices making their data available to the industrial Internet (industrial Internet is a group of sensors and actuators, aka physical devices used for industrial purposes that generate data independently). This availability is made possible by the use of communication protocols that suit the M2M application at hand. There are many communication protocols available for open usage, however, the most popularly used ones are MQTT, HTTP, CoAP, AMQP. Let us go through the utility of each of these communication protocols and then

briefly cover the reason behind our choice of MQTT for our smart weather monitoring/real time alerting application.
CoAP(Constrained Application Protocol) and HTTP(HyperText Transfer Protocol) are both URI based protocols wherein data is published onto a Uniform Resource Identifier(URI) and a subscriber subscribes to a particular topic on the URI to receive real time data updates of the same. CoAP is a binary protocol with a fixed header size(4 bytes) and small message payload dependent up to a maximum size decided by the capability of the web server and/or the programming technology. HTTP, on the other hand, does not define a header size and message payload because of its inherent existence as a text-based protocol (consequently, it is dependent on the web server or programming technology). CoAP's preferred transport protocol is UDP whilst HTTP uses TCP as its transport protocol; CoAP uses DTSL for encryption while HTTP uses TLS/SSL for the same. The server-client interface of CoAP is connectionless and diagram-oriented ensuring greater simplicity but at the cost of reliability; on the other hand, HTTP has a server-client interface that is connection-oriented. QoS levels is a staple of confirmation of message reception in CoAP whilst HTTP supports QoS but does not explicitly define it. In CoAP, the QoS levels are defined as "confirmable" or "non-confirmable" with the former being confirmed only by the generation of ACK packets at the receiver end. Both CoAP and HTTP make use of the RESTful Web environment with its request/response architecture, in fact, CoAP is built to be used in tandem with HTTP. In comparison to CoAP, HTTP is a globally recognized web protocol that offers several features like consistent connections, request pipelining and chunked transfer encoding.
AMQP(Advanced Message Queuing Protocol) is a lightweight, M2M, corporate protocol whose primary goal is to offer reliability, security, provisioning and interoperability. AMQP finds usability in request/response and publish/subscribe architecture with this versatility allowing it to offer a wide range of features such as reliable queuing, topical publish-and-subscribe communication, flexible routing and transactions. The premise of AMQP communication revolves around the creation and sharing of an "exchange" to participants of the protocol. Publishers and end users make use of the name of this exchange to discover and interact with each other. An end user creates a "queue" which is attached to the exchange. Messages received via the exchange have to be matched to the queue using a method called "binding". AMQP's communication methodology can be header (or topic) based or a variant of the fanout form. A fixed header size of 8 bytes and small message payloads with size reliant on the broker/server or programming infrastructure are all defining features of AMQP's binary nature. AMQP's preferred security protocols are TLS/SSL and SASL whilst its default transport protocol is TCP. A connection-oriented communication interface between client and broker is used by AMQP. A core tenet of AMQP is its reliability with its use of two preliminary QoS levels for message delivery: Unsettle format (aka message is not reliable) and Settle format (aka message is reliable).

MQTT Protocol:
MQTT(Message Queuing Telemetry Transport Protocol) is one of the earliest M2M communication protocols with its introduction in 1999. Developed by IBM's Andy Stanford-Clark and Arcom Control Systems Ltd (Eurotech)'s Arlen Nipper, it is a lightweight M2M publish/subscribe communication protocol that is designed for constrained network applications. It is a binary protocol with a header size of 2 bytes and maximum allowable message payload is 256 MB. The MQTT client transmits messages to a MQTT broker topic, end-users/clients that are subscribed to the topic receive these updates or they can be stored for future subscription. With a default TCP transport protocol and TLS/SSL security protocol combined with multi-topic subscription utility makes it a connection-oriented protocol. MQTT offers three QoS levels for reliable message transmission. We have chosen MQTT as our choice of IoT communication protocol because of its suitability for large networks of small devices(temperature sensor in our smart weather monitoring application) that requires monitoring on a back-end Internet server implemented using Node Red servers.

The publish-subscribe model used in MQTT is different from the traditional client-server model. It separates the client (publisher) that sends the message from the client (subscriber) that receives the message. The publisher and the subscriber do not need to establish direct contact. We can either let multiple publishers publish messages to one subscriber, or let multiple subscribers receive messages from one publisher at the same time. The essence of it is that an intermediate role called a broker is responsible for all message routing and distribution. The traditional client-server model can achieve similar results, but it cannot be as simple and elegant as the publish subscribe model.



SOFTWARE USED:

Arduino IDE: We used arduino ide to program nodemcu esp8266. The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of third-party cores, other vendor development boards. This software can be used with any Arduino board. The source code for the IDE is released under the GNU General Public License, version 2. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution.The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.By default, avrdude is used as the uploading tool to flash the user code onto official Arduino boards.

Amazon Web Services (AWS) : AWS is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow. Amazon Web Service is a Platform as a Service type of cloud. Here, service refers to the development environment and packages/subsystems that form a part of it. Users are involved in the distribution, development and testing of their own applications but the infrastructure is provided by the service provider which is Amazon Web Service in this case.
In simple words AWS allows you to do :

1. Running web and application servers in the cloud to host dynamic websites.
2. Securely store all your files on the cloud so you can access them from anywhere.
3. Using managed databases like MySQL, Oracle or SQL Server to store information.
4. Deliver static and dynamic files quickly around the world using a Content Delivery Network (CDN).
5. Send bulk email to your customers.

Node-Red Services: Flow editing and visual programming is a key aspect of IoT. It helps in hosting the entire IoT network on a vast variety of platforms ensuring ease of access for future development. The concept of flow editing involves using Visual data flow programming Languages(VDFPLs) that define all the server components: input, output and functions as a series of nodes interconnected by arcs to form a graphical network for the visual representation of data exchange. The VDFPLs help a non-programmer to gain an intuitive understanding of the IoT placed on the cloud. To implement flow editing, an open-source web-based/browser-based tool called Node-RED is available. Node-RED works hand in hand with JavaScript, using it in its event model, client editor and server along with providing native support for it. The data-flow structures in Node-RED are called ‘flows’ composed of wire-connected ‘nodes’. The basic interface consists of an easy to access flow editor with flow templates provided on the left hand side. These flow templates can be dragged and dropped onto the flow canvas, with all users managing a single Node-RED flow. The creation of a flow is followed by its deployment, which implies the saving and then re-execution of the flow on the Node-RED server. Nodes are members of two JavaScript classes: Node base class and EventEmitter subclass (in Node.js event API). Upon initialization , the nodes may subscribe to external services and receive data on ports or begin processing HTTP requests. Upon completion of data processing by a node, either from an external service or upon reception from an upstream node, base class Node send() is called with a JavaScript object. The send method works in tandem with the EventEmitter.emit() method to send named events to downstream node instances that process data and get involved in generation of additional events, or allow for communication with outside services or OS. The community Node-RED support involves IBM and a plethora of users that contribute new nodes and flows. The implementation of new nodes can be done in JavaScript and their addition can be done via the addition of an HTML file to implement browser UI, and a JavaScript file for server-based data processing and integration. Text-based flow representations can be exchanged between instances. If a node used in one implementation of Node-RED is not usable anywhere else, a temporary node is placed to ensure the user implements the missing node before flow deployment in the UI.

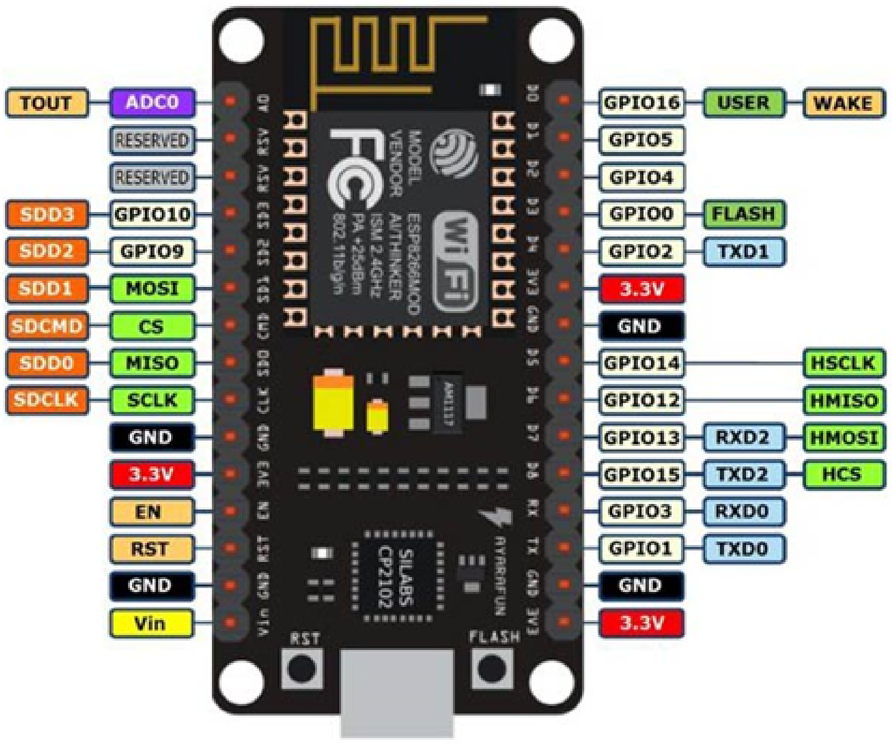
MQTT (Message Queuing Telemetry Transport) Box: MQTT Box is a helper program to develop and load test MQTT based clients, brokers, devices, cloud and apps. Every aspect of MQTT Box is specifically designed to maximize development and testing productivity. Together, with MQTT clients and load testing tools integrated, it's powerful enough to supercharge your MQTT workflow. MQTT Box provides support on Windows, Linux, Chrome, Mac and Web. It also provides ‘subscribe/publish’ functionality along with support for important protocols like 6LoWPAN WSN. Combined with an aesthetically pleasing and simple interface, MQTT Box is the primary broker software for any IoT application.

IFTTT: IFTTT derives its name from the programming conditional statement “if this, then that” and it helps all your apps and devices work better together. On IFTTT we can create applets.
What are Applets: Applets bring your services together to create remarkable, new experiences.
What are Services: Services are the apps and devices you see every day. There are countless useful ways to connect services with Applets.
In our project this applet will connect to our node red terminal and send us email whenever the temperature limit exceeds a certain value.

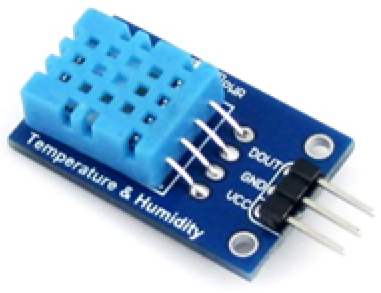
HARDWARE USED:

- DHT 11 Module (Temperature and Humidity sensor)
- NodeMCU ESP8266 Module (Microcontroller)
- Breadboard
- Connecting wires (F-F)

NodeMCU ESP8266 Module: The NodeMCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi and Deep Sleep Operating features make it ideal for IoT projects.
NodeMCU can be powered using Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface. NodeMCU has a built-in-wifi module of 2.4GHz that supports 802.11 b/g/n , 1 ADC(Analog to Digital) Pin of 10 bit resolution with 3.0 to 3.6 V operating voltage and 80mA of operating current. Operating temperature range varies from -40 to 125 degree Celsius.

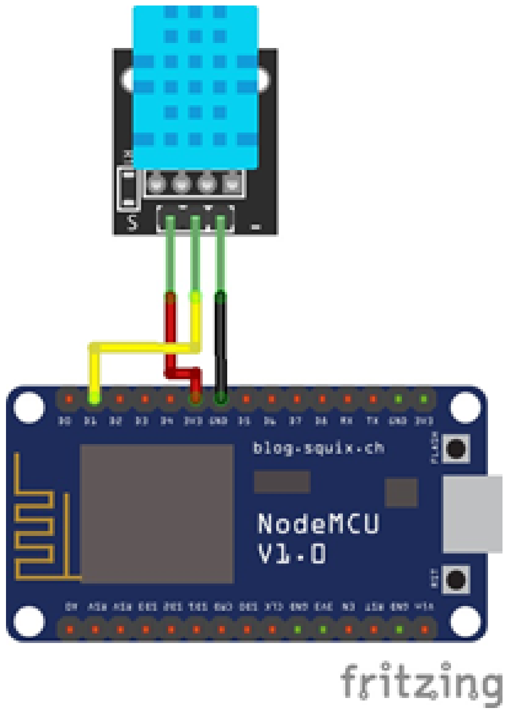


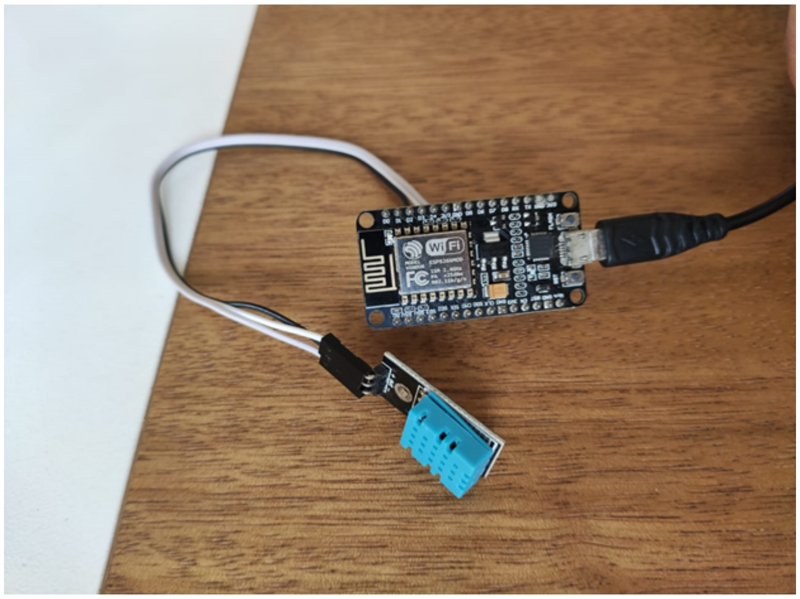
DHT11 MODULE: The DHT11 is a commonly used Temperature and humidity sensor. It measures the Temperature using thermistor and Humidity using Capacitor. The temperature ranges from 0 to 50 degree Celsius and Humidity ranges from 20 to 80%. The operating voltage and current is about 3 to 5 V or 3.5 to 5.5 V and 2.5 to 3.5 mA respectively. DHT11 Module has 3 pins i.e. VCC, GND and Data Pin. Both the output Temperature and Humidity sensed by the module is given through the Data pin.



The only difference between the DHT11 sensor and module is that the module will have a filtering capacitor and pull-up resistor inbuilt, and for the sensor, you have to use them externally if required.

Circuit Connection:

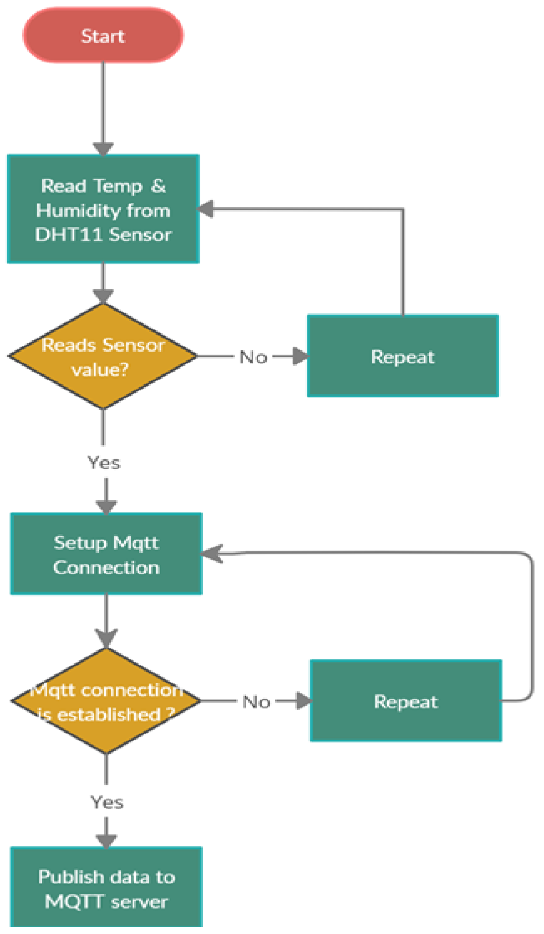


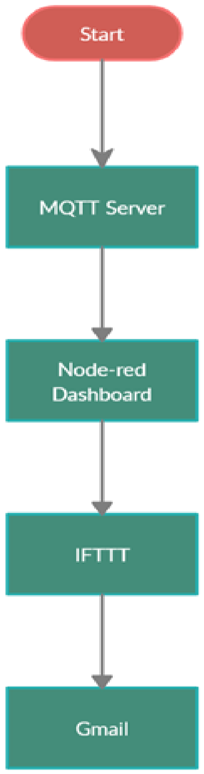


VCC Pin of DHT11 Module is connected to 3V Pin of NodeMCU similarly the GND pin of DHT11 Module is connected to GND pin of NodeMCU and through the data pin the overall information is sent to any one of the digital pin of the NodeMCU (Digital pin D1 in our case shown in figure) and then data is sent to the cloud.

FLOW OF PROJECT:

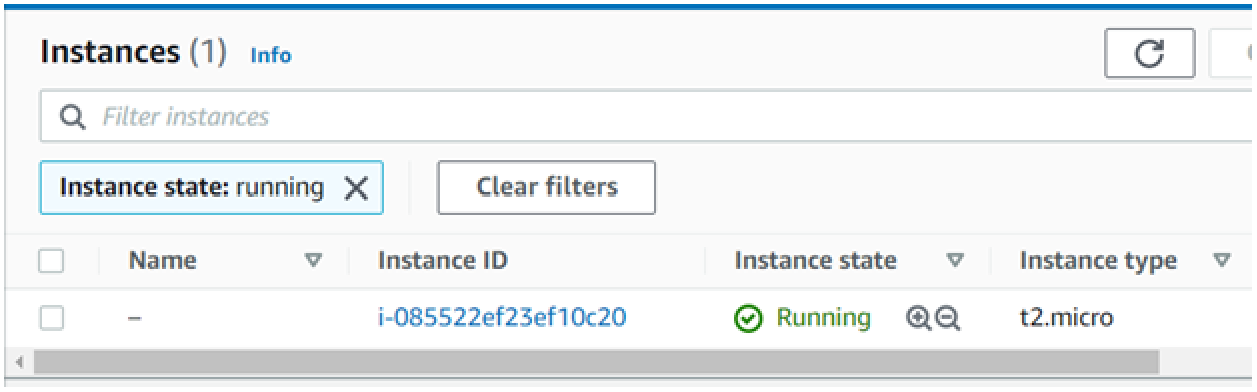
In this part we will be discussing the complete structure and flow of our project. We have divided our project into 2 main parts, in the first part we will be interfacing the DHT11 temperature and humidity sensor with NodeMCU ESP8266. After reading the value from the sensor once it reads the value, we will check if our nodemcu is connected to our mqtt server and once it is connected, we will publish data on our mqtt server. In the second part we will be building our flow of nodes on the node-red terminal so that node-red will receive the data from our NodeMCU. We have also made a simple, decent and user friendly interface for visualizing data on a node-red dashboard. Next service which we are using is IFTTT which is actually taking output from node red and if the value of temperature exceeds a certain threshold value then an alert email will be sent to the user. One more and the most important thing is that we have used AWS to setup our server and by using mosquitto library in ubuntu we have set up a MQTT broker, which is the middle man between our sensor and other devices. The flowchart is shown below:


Start
Read Temp & Humidity from DHT11 Sensor
Reads Sensor value?
No
Repeat
Yes
Setup Mqtt Connection
Mqtt connection is established ?
No
Repeat
Yes
Publish data to MQTT server



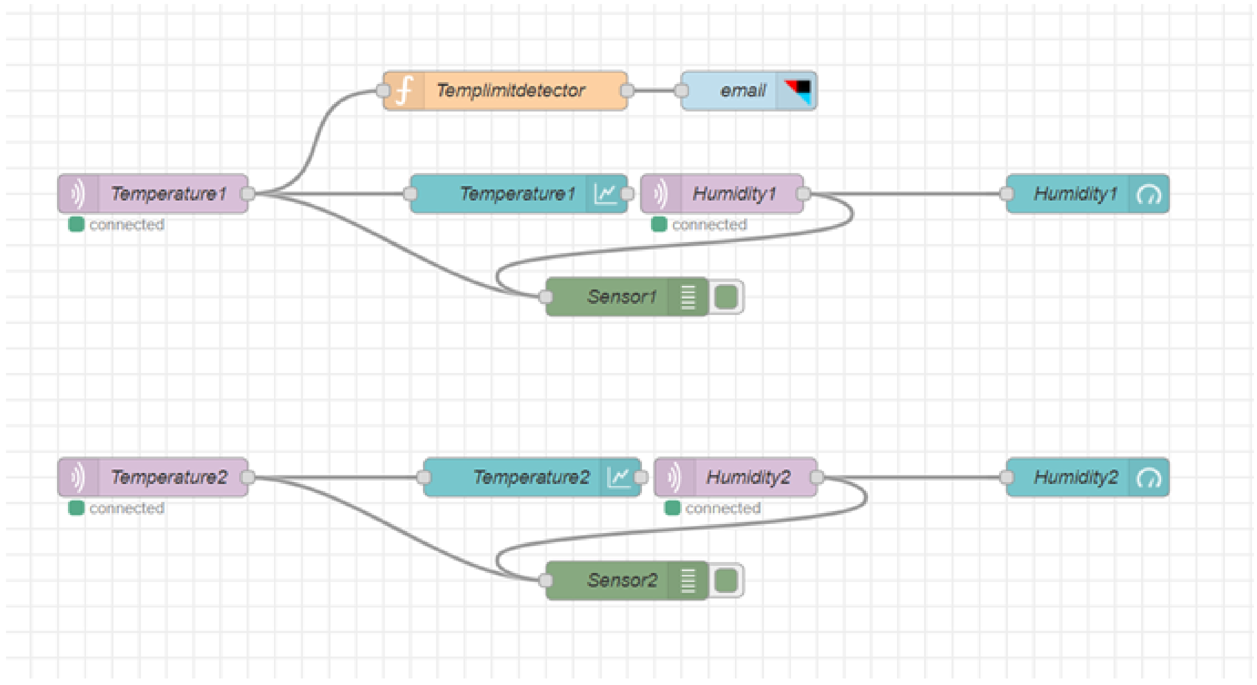
IMPLEMENTATION:

Setting Up AWS Server:
We have to create a Ubuntu instance using Amazon Web Services. This is basically a virtual pc. The terminal of this virtual server is opened using putty software. In this terminal we have to install node-red and the mosquitto broker.This server acts as a virtual Raspberry Pi. The AWS instance installed is shown below:

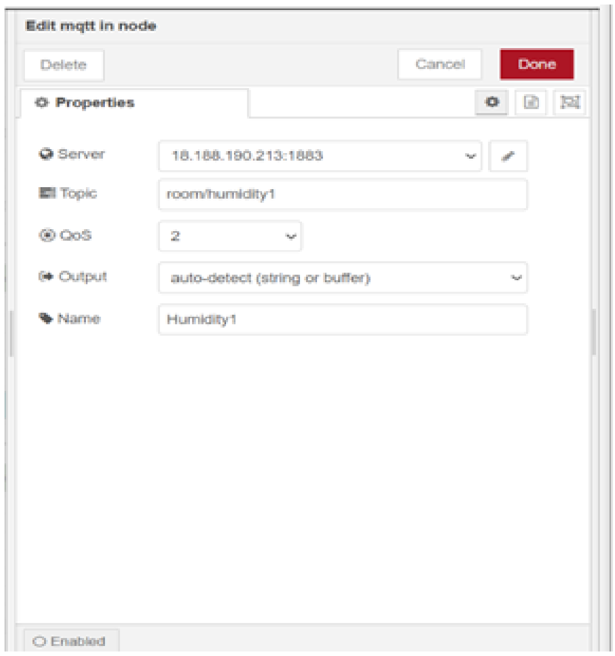


Name	Instance ID	Instance state	Instance type
–	i-085522ef23ef10c20	Running	t2.micro

Node-Red Flow Editing:



In the above diagram we can see that there are various nodes used. The first node which is used is the "MQTT in" node which is named Temperature1. This node subscribes to the topic that we publish to from the Arduino IDE and sends the data to the dashboard. In the MQTT node we have to specify the ip address and the topic that we want to subscribe to. Similarly we give different topic names for humidity which is shown below in the fig.



Edit mqtt in node

Delete Cancel Done

Properties

Server 18.188.190.213:1883

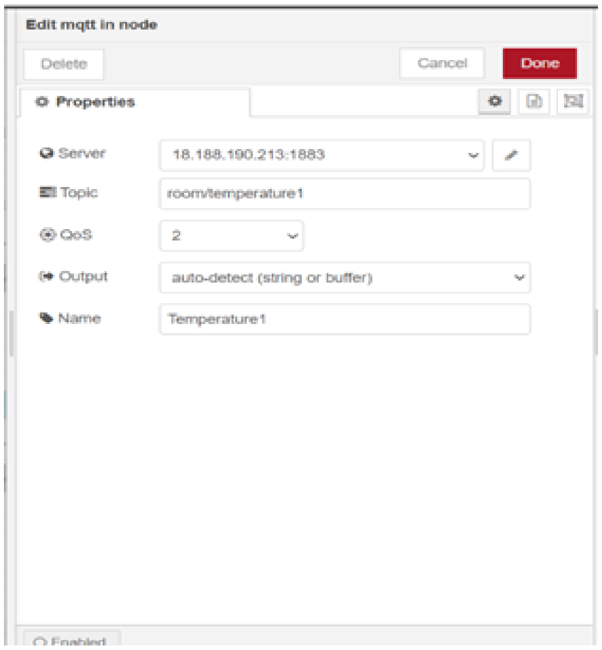
Topic room/humidity1

QoS 2

Output auto-detect (string or buffer)

Name Humidity1

Enabled



Edit mqtt in node

Delete Cancel Done

Properties

Server 18.188.190.213:1883

Topic room/temperature1

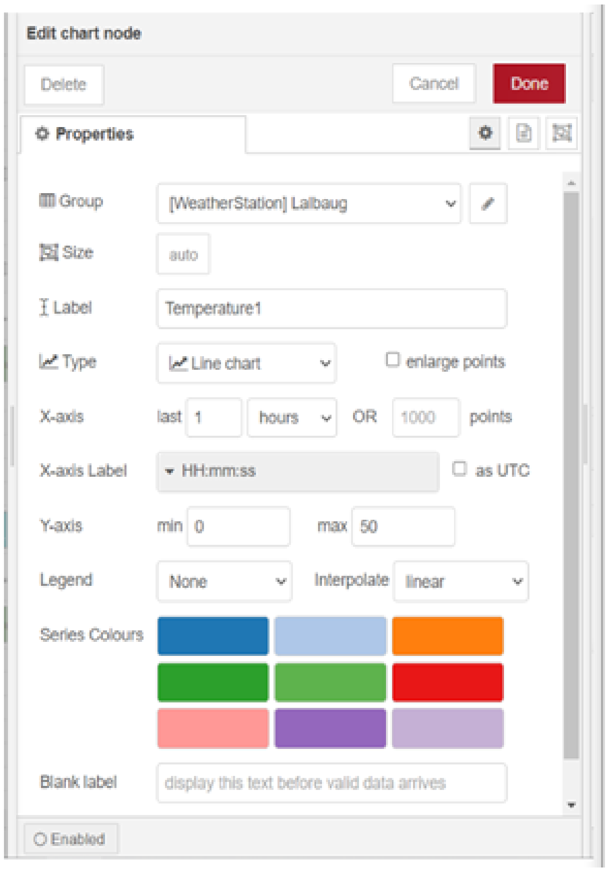
QoS 2

Output auto-detect (string or buffer)

Name Temperature1

Enabled

The other node which we used is a dashboard node in which we decide to give name and type of dashboard i.e Line chart and Gauge for Temperature and Humidity respectively. Here we can customize the colour of the chart or gauge. The setting of this node is shown in the figure below.

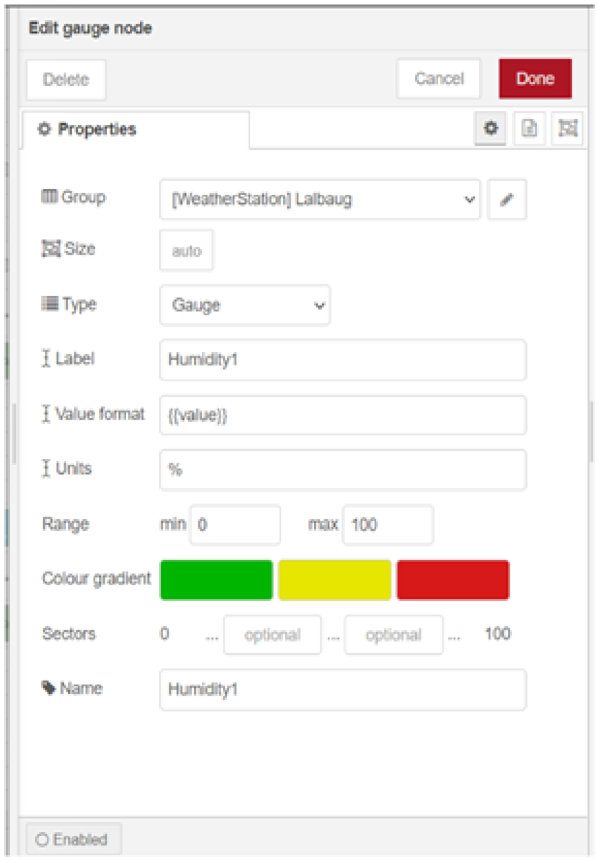

Edit chart node

Delete | Cancel | Done

Properties

Group	[WeatherStation] Lalbaug
Size	auto
Label	Temperature1
Type	Line chart ☐ enlarge points
X-axis	last 1 hours OR 1000 points
X-axis Label	HH:mm:ss ☐ as UTC
Y-axis	min 0 max 50
Legend	None Interpolate linear
Series Colours	🟦 🟦 🟧 🟩 🟩 🟥 🟥 🟪 🟪
Blank label	display this text before valid data arrives

Enabled


Edit gauge node

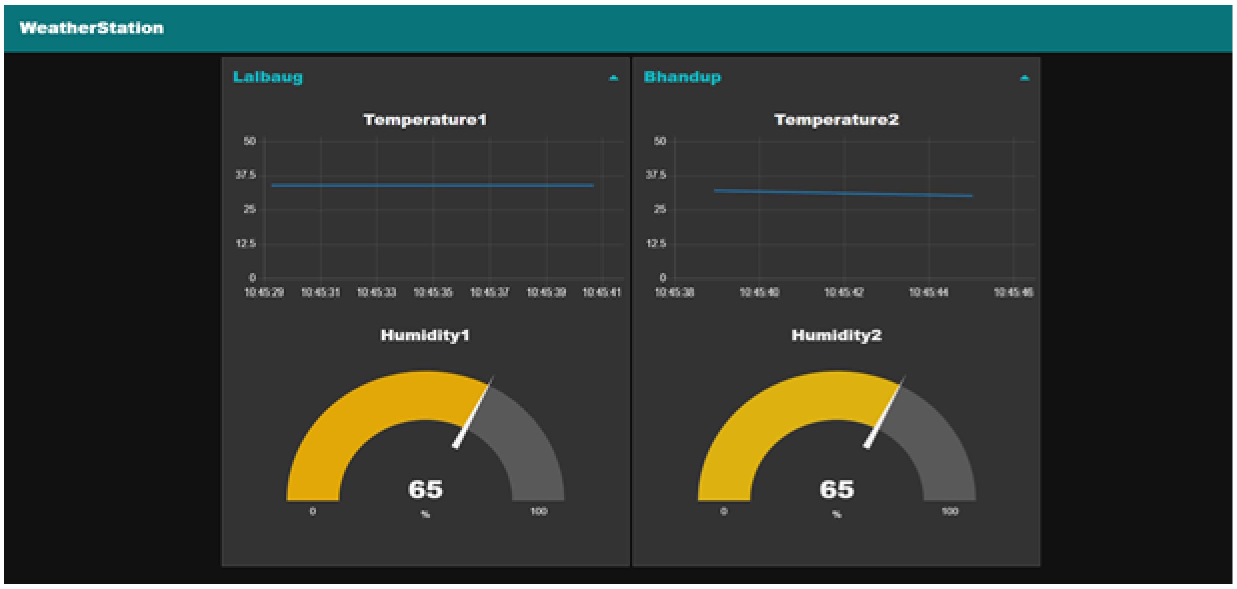
Delete | Cancel | Done

Properties

Group	[WeatherStation] Lalbaug
Size	auto
Type	Gauge
Label	Humidity1
Value format	{{value}}
Units	%
Range	min 0 max 100
Colour gradient	🟩 🟨 🟥
Sectors	0 ... optional ... optional ... 100
Name	Humidity1

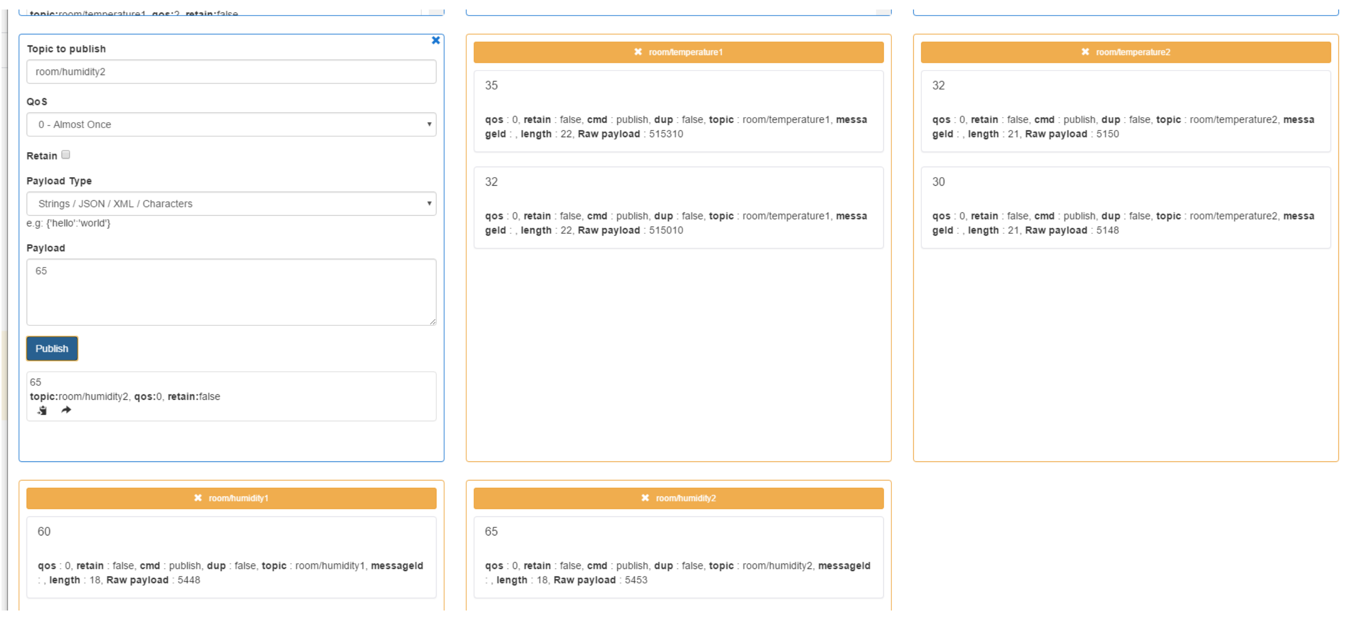
Enabled

The Dashboard can be seen on the UI page as shown below:



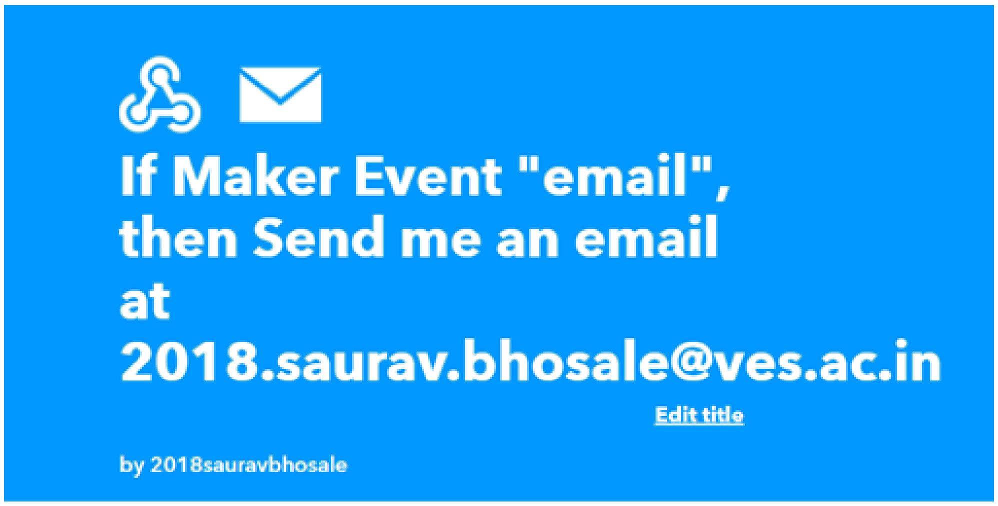
The Data coming from the DHT11 sensor can be visualized on the UI page which is opened by

writing ‘your_ipaddress:1880/ui’. The debug node which is named as sensor1 is used to see the data on the node-red terminal. This data can be seen on the debug window in node-red. Similar process is followed for the data received from other locations.
The data being received can be seen on mqtt box as follows:



```
topic:room/temperature1, qos:2, retain:false

Topic to publish
room/humidity2
QoS
0 - Almost Once
Retain
Payload Type
Strings / JSON / XML / Characters
e.g: {'hello':'world'}
Payload
65
Publish
65
topic:room/humidity2, qos:0, retain:false

room/temperature1
35
qos : 0, retain : false, cmd : publish, dup : false, topic : room/temperature1, messa
geld : , length : 22, Raw payload : 515310
32
qos : 0, retain : false, cmd : publish, dup : false, topic : room/temperature1, messa
geld : , length : 22, Raw payload : 515010

room/temperature2
32
qos : 0, retain : false, cmd : publish, dup : false, topic : room/temperature2, messa
geld : , length : 21, Raw payload : 5150
30
qos : 0, retain : false, cmd : publish, dup : false, topic : room/temperature2, messa
geld : , length : 21, Raw payload : 5148

room/humidity1
60
qos : 0, retain : false, cmd : publish, dup : false, topic : room/humidity1, messageld
: , length : 18, Raw payload : 5448

room/humidity2
65
qos : 0, retain : false, cmd : publish, dup : false, topic : room/humidity2, messageld
: , length : 18, Raw payload : 5453
```

Alert on email using IFTTT:
We created an applet on IFTTT where we receive a web request using Webhooks service which is given by IFTTT and send an email whenever the web request is received.

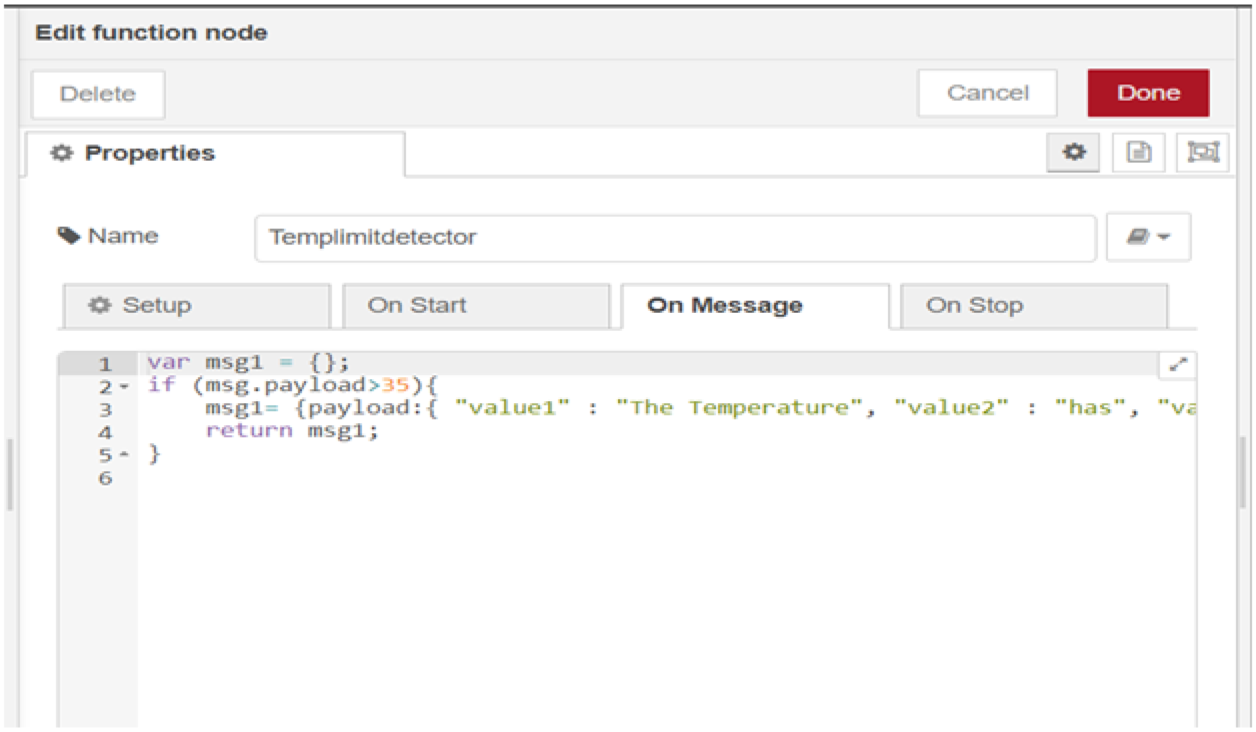


If Maker Event "email", then Send me an email at 2018.saurav.bhosale@ves.ac.in

Edit title

by 2018sauravbhosale

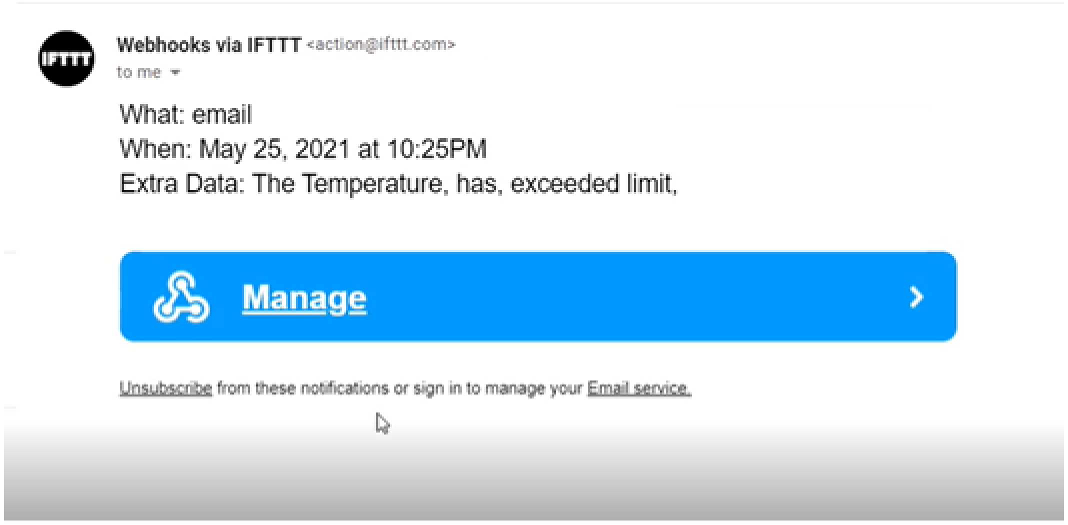
The web request is received whenever msg.payload i.e. the temperature1 is greater than 35. This logic is executed using a simple javascript program written in the function node. This program will only send values greater than 35 in the IFTTT out node. In this node we specify a key and write the event name. Whenever this event is triggered the webhooks will receive a webrequest and subsequently a mail in the specified email address.
The javascript code in the function node is shown below:



Name: Templimitdetector

```
var msg1 = {};
if (msg.payload>35){
    msg1= {payload:{ "value1" : "The Temperature", "value2" : "has", "va
    return msg1;
}
```

The mail received is as follows:



Webhooks via IFTTT <action@ifttt.com>
to me

What: email
When: May 25, 2021 at 10:25PM
Extra Data: The Temperature, has, exceeded limit,

Unsubscribe from these notifications or sign in to manage your Email service.

CONCLUSION:

In this way we have implemented a Smart Weather Monitoring System using MQTT protocol and node-red. We learned about IoT, how IoT systems work and different protocols used in it. We also implemented one of the IoT protocols which is MQTT Protocol. This project is not only limited to this system but also it can be used in process applications like a level control loop or a temperature control loop. Such type of weather monitoring can also be used in predictive analysis of weather where we can log the data in a csv file and use it for further data analytics.

REFERENCES:

1. M. Lekić and G. Gardašević, "IoT sensor integration to Node-RED platform," 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, Bosnia and Herzegovina, 2018, pp. 1-5, doi: 10.1109/INFOTEH.2018.8345544.
2. N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 2017, pp. 1-7, doi: 10.1109/SysEng.2017.8088251.
3. Internet of Things (IOT) Based Weather Monitoring system . AUTHOR : Bulipe Srinivas Rao, Prof. Dr. K. Srinivasa Rao, Mr. N. Ome.
4. Srivastava, D., Kesarwani, A., & Dubey, S. (2018). Measurement of Temperature and Humidity by using Arduino Tool and DHT11. International Research Journal of Engineering and Technology (IRJET), 5(12), 876-878.
5. Rahut, Y., Afreen, R., Kamini, D., & Gnanamalar, S. S. (2018). Smart weather monitoring and real time alert system using IoT. International Research Journal of Engineering and Technology.
6. https://www.mobileappdaily.com/iot-in-weather-technologies
7. http://www.pollutionequipmentnews.com/the-importance-of-weather-monitoring
8. http://www.ijser.in/archives/v6i7/IJSER172702