An MQTT-based IoT Cloud Platform with Flow Design by Node-RED

Choopan Rattanapoka
College of Industrial Technology
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
choopanr@kmutnb.ac.th

Somphop Chanthakit
College of Industrial Technology
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
s5903053816027@email.kmutnb.ac.th

Apatsaraporn Chimchai
College of Industrial Technology
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
s6003052422111@email.kmutnb.ac.th

Amorntip Sookkeaw
College of Industrial Technology
King Mongkut's University of Technology North Bangkok
Bangkok, Thailand
s6003052422129@email.kmutnb.ac.th

***Abstract*— Nowadays, Internet of Things or IoT is gaining popularity. However, the process to set up the IoT platform is complex and time consuming for some IoT developers. Thus, this paper presents the design and development of an MQTT-based IoT cloud platform with flow design by Node-RED to help developers save time to set up the IoT platform by themselves. The user interface of this platform is in the form of a web application developed with the MERN stack. The platform provides three main services: (1) MQTT broker for data transmission (2) Node-RED for designing the flow of data processing and displaying data in graphical form via Dashboard and (3) InfluxDB and MongoDB databases for storing data. These services operate as containers running on the Docker. The web application connects to the Docker via Docker API using the Dockerode library to start, stop and get the status of the containers. Using the test by making an IoT system for monitoring and displaying temperature values, we found that our platform can support and help users in developing the IoT system very well. The developers who want to develop IoT systems do not need to set up an IoT platform by themselves.**

Keywords—IoT, Cloud Platform, Node-RED, Docker

I. Introduction

Currently, Internet of Things or IoT is gaining popularity because of the concept that the devices can connect and communicate with each other. In [1], they categorize the IoT applications into five major areas including transportation and logistics, healthcare, smart environments, personal and social, and futuristic. There are many kinds of research and innovation in various fields based on IoT such as the research in [2] presented an IoT based activity recognition system for smart home environment based on wearable sensors which gather the user's activity data in real-time with the accuracy of 72%, [3] presented a framework for smart transportation using big data analytics to provide citizens with better life and services, and [4] presented a comprehensive survey of IoT architecture and system design for healthcare systems.

For implementing the IoT systems, the developers need to set up the IoT platform such as computer servers, related softwares, and related services by themselves. Therefore, the big companies such as Microsoft Azure, Amazon AWS, and Google Cloud IoT offer the IoT platform on cloud that helps the developers to save their time for setting up the IoT platform. However, these platforms are located overseas and may cause the problem with the speed of real-time receiving and sending of data.

This paper presents the design and development of an IoT platform on cloud in the form of web application developed with the MERN stack. The goals of our platform are to save costs from using public IoT cloud platforms and to maximize the usage of the domestics Internet bandwidth. Our platform consists of three parts: (1) MQTT Broker for sending and receiving data (2) Node-RED for designing the flow of data processing and displaying data in graphical form via Dashboard and (3) InfluxDB and MongoDB database systems for storing data.

II. Backgrounds and Related Works

A. MERN stack

The MERN stack is a framework to help develops designing and developing web applications. The MERN stack consists of four parts: MongoDB, Express.js, ReactJS, and Node.js.

MongoDB is a document-oriented database system classified as a NoSQL database system. The data are stored in a document format with a structure similar to JSON documents. The documents in MongoDB are stored as key and value insides the collection.

Express.js or Express is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js

ReactJS or React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. It supports the JSX (JavaScript Syntax Extension). When implementing the web application, each part of the website is divided into components. The data in each component are called state, and the data passed from the top layer to the lower layer is called props.

Node.js is a cross-platform runtime environment for server-side written in JavaScript language. NodeJS uses Event-driven and Non-blocking I/O to make the system has not only efficiency and faster processing but also uses fewer resources.

978-1-7281-4100-8/19/$31.00 ©2019 IEEE

B. Docker

Docker [5] is a tool designed to make it easier to create, deploy, and run applications by using containers. Docker works quite similar to a virtual machine, however, instead of creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they are running on and only requires applications be shipped with things not already running on the host computer. This concept and implementation give a significant performance boost and reduces the size of the application.

C. Node-RED

Node-RED [6] is an open-source software tool for developers to connect hardware devices with APIs and online services. Hence, developers do not need to write the server-side APIs themselves. Also, Node-RED provides developers with a browser-based flow editor that developers can use a web browser to put nodes and link them together for designing and implementing a flow-based program. There are many types of nodes in Node-RED that developers can use such as node for receiving, sending, processing, transforming, and storing data. Thus, it makes developers faster to develop IoT programs.

D. MQTT

MQTT [7] is a publish-subscribe-based protocol that works on top of the TCP/IP protocol suite. It is a lightweight protocol designed to maximize network traffic and quickly and simply bring the reliability of messaging applications written for these environments. MQTT can be used for mobile applications and also for telemetry and sensor machine-to-machine applications. The public-subscribe messaging pattern requires a message broker. Thus, in MQTT, the MQTT broker acts as an intermediary to forward the message received from the publishers to the subscribers according to the topic of the message.

E. InfluxDB

InfluxDB [8] is an open-source time-series database. It is written in Go and optimized for fast, high availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, IoT sensor data, and real-time analytics. Moreover, InfluxDB can take care of dropping data after a certain time, with a concept called retention policy to reduce the size of the database and make database operations faster.

F. Related works

Besides Microsoft Azure, Amazon AWS and Google Cloud IoT that provides a total solution for IoT applications, CloudMQTT and FRED platform are providing the minimal set of tools that users need for an IoT platform.

CloudMQTT [9] is a platform hosting message brokers for the IoT. Users can create MQTT broker instances and manage devices connecting to that broker via the MQTT library or Websocket. However, CloudMQTT aims to provide only the message broker part. There are no built-in database or data processing.

FRED (Front End for Node-RED) [10] is the closest platform to our platform. FRED allows users to create MQTT, Node-RED and InfluxDB instances. However, as the FRED servers are located in the United States, the connection latency may affect the speed for exchanging IoT data for the IoT applications hosted in Thailand.

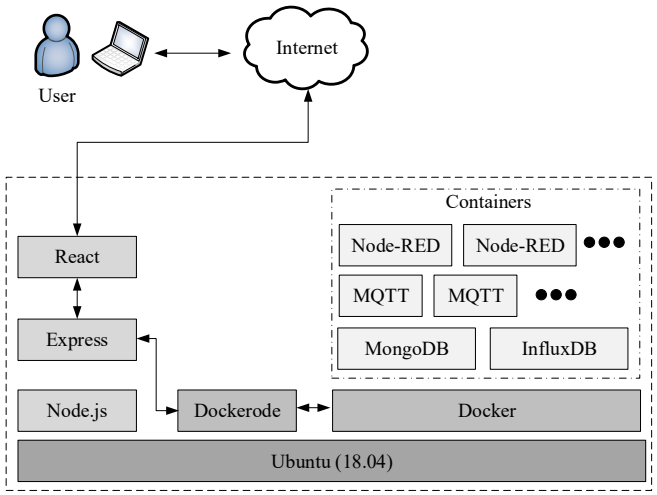

Fig. 1. The architecture of the proposed system.

III. Research Methodology

The architecture of our MQTT-based IoT cloud platform shows in Fig. 1. Our system consists of (1) the web application implemented on MERN stack which allows users to access our system on their web browser via the Internet (2) Docker that manages the containers of Node-RED, MQTT broker, MongoDB and InfluxDB, and (3) the Dockerode [11] library that allows our web application to connect and manage Docker. All of these components in this research are installed and tested on Ubuntu version 18.04.

To use our system, the users need to register and login to the web application. Then, each user can create and manage MQTT broker, Node-RED, MongoDB and InfluxDB instances independently. For MQTT broker and Node-RED instances, one instance is running on one container, meanwhile, a MongoDB container and an InfluxDB container are run on the system. The reason that we design to run only one container per database type and share among the users is to reduce the resources used in the system. Thus, each user can create databases, but the name of the database must not be the same as other users. Moreover, for security reasons, users need to specify the username and password to access the created databases.

The steps to implement our system in this research can be divided into four main steps as follows.

A. Preparing docker images

After installing the Docker on the Ubuntu, we download four docker images from the dockerhub by using the "docker pull" command to provide the services to users, The four docker images consisting of the images of Node-RED, MQTT broker (mosquito), MongoDB, and InfluxDB. The docker images of the MQTT broker, MongoDB, and InfluxDB are ready to use without any additional settings. However, for the Node-RED docker image, we require some additional settings by adding the module "dashboard" for displaying data and the modules for connecting the MongoDB and InfluxDB databases.

To add modules in the Node-RED image, we need to start a Node-RED container with the "docker run" command. Then, we open the Node-RED management page via the web browser, select the "Manage Palette" menu, search and install the Dashboard, InfluxDB, and MongoDB palettes. When the installation of palettes is completed, we use the "docker commit" command to save the customized Node-RED docker image to the new docker image for using in our platform.

```
var Docker = require('dockerode');
var dockerIP = "192.168.56.2"
var dockerPort = 2375

var docker = new Docker({host: dockerIP, port: dockerPort});

docker.createContainer({
    name: 'containername',
    Image: 'toke/mosquitto',
    AttachStdin: false,
    AttachStdout: false,
    AttachStderr: false,
    Tty: false,
    Cmd: [],
    OpenStdin: false,
    StdinOnce: false,
    HostConfig: {
        PortBindings: {
            "1883/tcp": [
                {
                    HostIp: "0.0.0.0",
                    HostPort: "0"
                }
            ]
        }
    }
}).then(function(container) {
    return container.start();
}).catch(function(err) {
    console.log(err);
});
```

Fig. 2 A partial code for creating a docker container using Dockerode.

B. *Web application and docker connection*

The main function of the web application is to facilitate users in managing all of the four services (Node-RED, MQTT broker, MongoDB, and InfluxDB). Users must create service instances, start services, stop services, monitor the status of services, and delete the service instances by themselves. Thus, the web application needs to connect to the Docker and can control and manage it. In our case, the web application connects and communicates with the Docker via Docker API and we use the Dockerode library to make the development process more convenient.

Fig. 2 shows a partial code for communicating with the Docker API via Dockerode to create a docker container from the docker image "mosquito" that acts as an MQTT broker. For creating a container of MQTT broker, we need to specify the HostConfig parameter to bind the container port 1883/TCP which is an MQTT broker service port to a random physical machine port (HostPort: "0"). If the container is successfully created, the docker will send the feedback data in JSON format as follows: [{"IP":"0.0.0.0", "PrivatePort": 1883, "Public Port":32773, "Type": "tcp"}]. This feedback data means that the port 1883 of the container is bound to the port 32773 of the physical machine.

C. *Node-RED security configuration*

By the default settings, The Node-RED does not turn on the security protection. Thus, anyone with a link can access the Node-RED management page. Fortunately, Node-RED provides basic security by using a /data/settings.js file. The structure of the /data/settings.js file is shown in Fig. 3. When this file is used, users need to type the username and password on the login page of the Node-RED web correctly as be set on the adminAuth, httpNodeAuth, and httpStaticAuth parameters to access the Node-RED management page.

However, each Node-RED container should have a different username and password to access the management page. Moreover, users should customize the username and password by themselves. Thus, we solve this problem by preparing a file "template.js" which is a copy of the setttings.js file and overwriting the username and password values of the adminAuth, httpNodeAuth and httpStaticAuth parameters to {{username}} and {{password}}.

Then, when a user creates a Node-RED instance, our system takes the username and password from user input, then replaces the {{username}} and {{password}} in the template.js with the user input, and finally saves the modified template.js file to a new file to use it later when user run the Node-RED instance. Fig. 4 shows a partial code to create a settings.js from the template.js file where *userred* and *passred* are the values of the username and password from the user input. The password value needs to be encrypted with a hash function.

```
module.exports = {
    uiPort: process.env.PORT || 1880,
    mqttReconnectTime: 15000,
    serialReconnectTime: 15000,
    debugMaxLength: 1000,

    adminAuth: {
        type: "credentials",
        users: [{
            username: "{{username}}",
            password: "{{password}}",
            permissions: "*"
        }]
    },

    httpNodeAuth: {user:"{{username}}",pass:"{{password}}"},
    httpStaticAuth: {user:"{{username}}",pass:"{{password}}"},

    logging: {
        console: {
            metrics: false,
            audit: false
        }
    },
    editorTheme: {
        projects: {
            enabled: false
        }
    }
}
```

Fig. 3 A structure of /data/settings.js in a Node-RED container.

```
var fs = require('fs')

const hash = bcrypt.hashSync(passred, config.SALT_ROUNDS);
fs.readFile('template.js', "utf8", function (err,data) {
    if (err) {
        return console.log(err);
    }
var result = data.replace(/{{username}}/g, userred);
var result2 = result.replace(/{{password}}/g, hash);
fs.writeFile('usepassnodered/'+username+namered+'.js',
                result2, 'utf8',
    function (err) {
        if (err) return console.log(err);
    });
});
```

Fig. 4 A partial code for creating a settings.js for each Node-RED container.

```
const Influx = requre('influx');
const influx = new Influx.InfluxDB(influxDBURL);
influx.createDatabase(databaseName);
influx.createUser(dbUser, dbPassword);
influx.grantPrivilege(dbUser, 'ALL', databaseName);
```

Fig. 5 A partial code for creating an InfluxDB database.

D. *Creating a database user account*

Because the databases in our system are shared among users, hence the name of the database must be unique. When users create a database instance, our web application takes three values from users which are database name, database user, and database password (databaseName, dbUser, and dbPassword respectively). Then, our web application connects to the database via database system API to create the database, to create the username and password to access the database, and to grant the privilege to that created user to the created database. The example code to do these operations on the InfluxDB is shown in Fig. 5.

IV. Experiments

After logging in to our IoT platform, the landing page is shown to users (shown in Fig. 6). On the left side of the page shows the menu to manage all of our three services: NodeRED to manage Node-RED instances, MQTT to manage MQTT broker instances, and Datastore to manage InfluxDB and MongoDB instances.

To evaluate the functions of our platform, we will use our implemented platform to build a basic IoT system for measuring and displaying the temperature values. The design of this system shows in Fig. 7. The device for measuring the temperature is a NodeMCU with a temperature sensor which we have built it in [12]. The device also acts as a publisher to public MQTT messages that contain the temperature value to the MQTT broker. On another side, Node-RED is used to act as a subscriber to receive MQTT messages from the MQTT broker. Then, Node-RED extracts the temperature value from MQTT messages and displays it in real-time on a gauge chart and also stores that temperature value into the InfluxDB. We also implemented Node-RED flow to take the temperature of the last 24 hours from the InfluxDB to display on a line chart.

The steps for developing this IoT system are as follows.

A. *Creating an MQTT broker instance*

To create an MQTT broker instance, users should select the MQTT menu on the left side of the web application. Then, click on the "create" button and users will be informed to input the name of MQTT instance. After the successful creation of the MQTT instance, the web application displays the IP address and the port number of the MQTT broker along with the MQTT broker status. Moreover, on this page, users can start, stop and delete the MQTT instances. In Fig. 8 shows the information on the MQTT management page after we created an MQTT instance named "tempBroker" which is running on the host IP address 192.168.56.2 and port 32774.

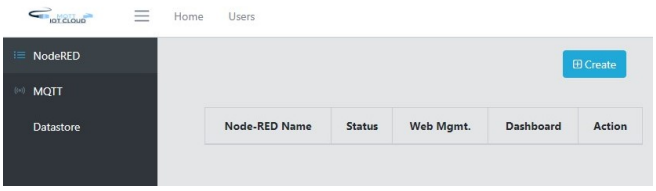

Fig. 6 The user's landing page of our platform.

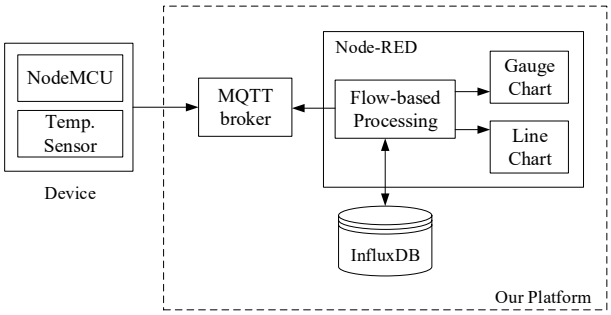

Fig. 7 The design of a basic IoT system for temperature monitoring.

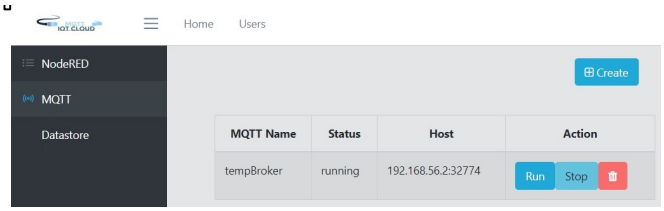

Fig. 8 The MQTT management page.

B. *Setting up the device*

In this work, we use a NodeMCU and a temperature sensor. Thus, we write a program in the NodeMCU to take the temperature from the sensor and then publish that temperature value to the MQTT broker at IP address 192.168.56.2 on the port 32774. The temperature data published to the MQTT broker is under the topic named "temp".

C. *Creating an InfluxDB instance*

Due to the design of the testing IoT system, we need to store temperature data to the InfluxDB database. Thus, we must create an InfluxDB database. To do this, users should select the Datastore menu on the left side of the web application. Then, click on the "create" button and there will be a dialog displaying to users to select the database type and to input the database name, database username, and database password as shown in Fig. 9.

In this work, we choose the InfluxDB database and create the database name "databasetest" with the database username "choopan". After the successful creation of the database instance, the web application displays the database system "InfluxDB", the database name "databasetest", the database username "choopan", the dataIP address 192.168.56.2 of the database server and port 8086 of the database server as shown in Fig. 10.

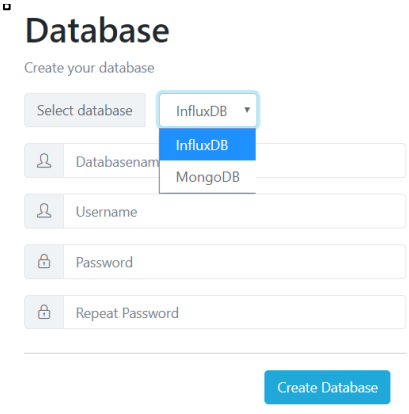

Fig. 9 The dialog to create a database.

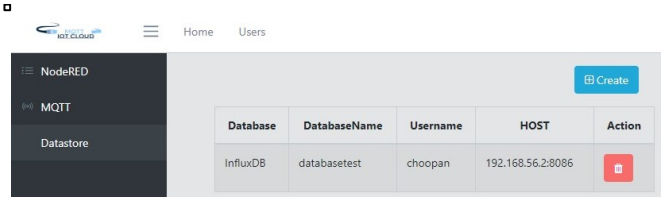

Fig. 10 The database management page.

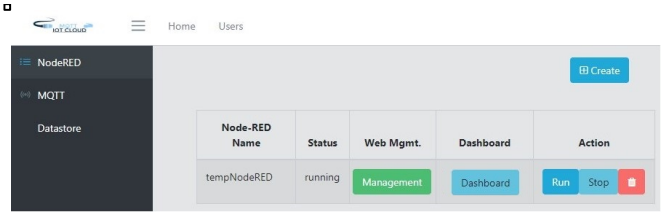

Fig. 11 The Node-RED management page.

D. *Creating a Node-RED instance*

Node-RED is used to make data flow processing and display a dashboard. Users should select the NodeRED menu on the left side of the web application. Then, click on the "create" button and there will be a dialog displaying to users to input the Node-RED instance name, username, and password to access the Node-RED web. After the successful creation of Node-RED instance, the web application displays the Node-RED instance name (we created the Node-RED instance name "tempNodeRED"), instance status, a button to access Node-RED management web page, a button to access Node-RED dashboard web page, and buttons to manage the instance as shown in Fig. 11.

E. *Design a flow processing on Node-RED*

After the Node-RED instance is activated, users can access the Node-RED web page via a button on our Node-RED management page. Users need to input the username and password specified in the previous step to login to the Node-RED web page.

We designed the flow processing as shown in Fig 12. There are two flows consisting of: (1) The upper flow is a flow to receive data from MQTT broker and insert that data into the InfluxDB database and at the same time send that data to display on the Node-RED dashboard in the form of a gauge chart and (2) The bottom flow is a flow that query the temperature data of the last one day from the InfluxDB database and display them on the Node-RED dashboard of a line chart.

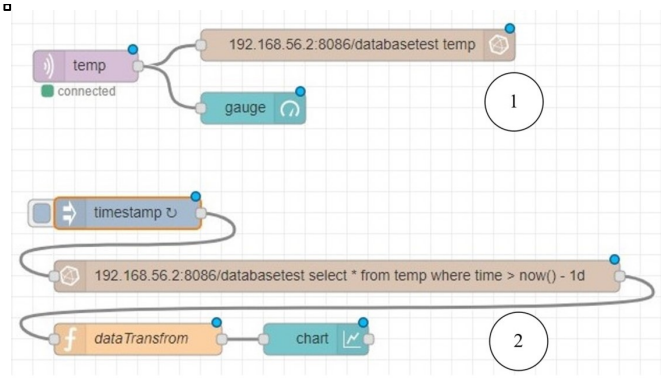

Fig. 12 The Node-RED management page.

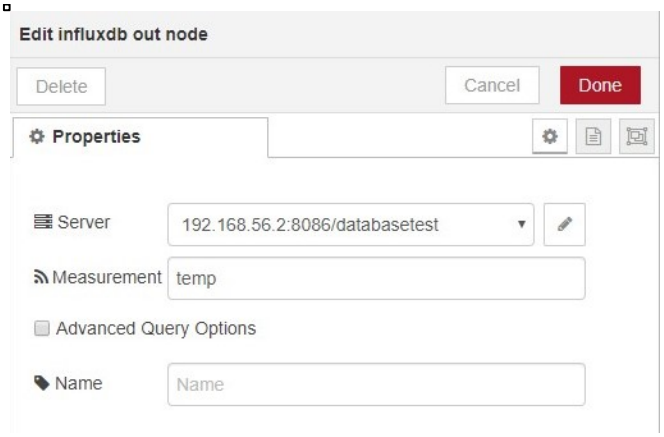

Fig. 13 The configuration of the influxdb out node.

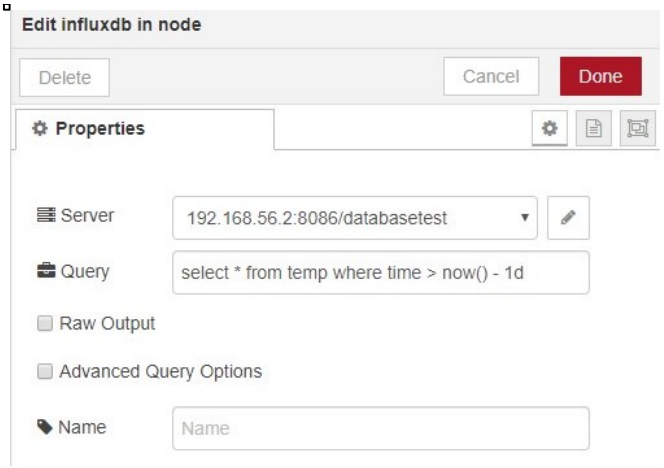

Fig. 14 The configuration of the influxdb in node.

The upper flow consists of three nodes.

- "mqtt in" node named "temp" which subscribes to the topic name "temp" of the MQTT broker at IP address 192.168.56.2 on the port 32774. This MQTT broker is the tempBroker that we created in the previous section.
- "influxdb out" node which connects to the InfluxDB database server at IP address 192.168.56.2 on the port 8086 and inserts the data received from the "mqtt in" node to the "temp" table in the database "databasetest". The configuration of this node is shown in Fig.13.
- "ui_gauge" node is a node that receives data from the "mqtt in" node and displays it in the form of a gauge chart on the Node-RED dashboard page.

The bottom flow consists of four nodes.

- "inject" node which is set to send a timestamp every 1 second. The timestamp sent by this node is not important, we use this node because we want the "influxdb" node is be activated every 1 second.
- "influxdb in" node which connects to the InfluxDB database server and queries the values of the temperature in the last 24 hours from the table "temp" in the database "databasetest". The query command is "selected * from temp where time > now () - 1d". The configuration of this node is shown in Fig.14.

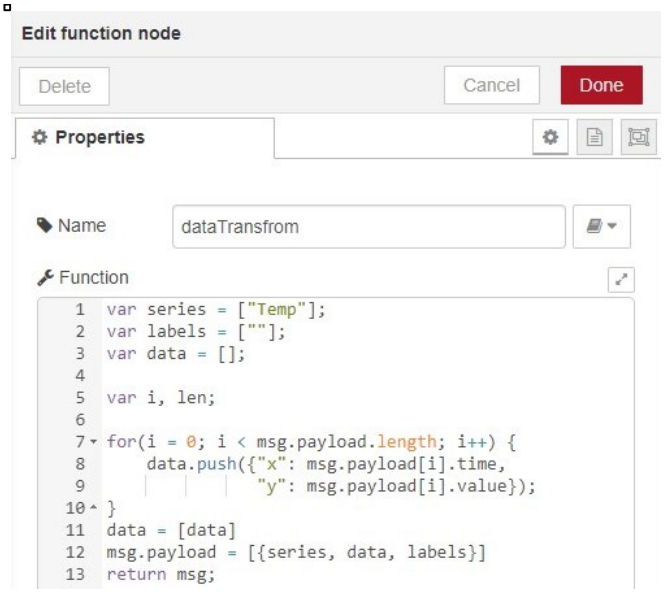

Fig. 15 The function node that transforms temperature data from the database to the ui-chart format.

- "function" node named "dataTransform" is used because the data from the database is in the object format and cannot display directly on the Node-RED dashboard. Thus, we need this node to convert to data in an object format to an appropriate JSON format. We can do this by writing code in Javascript in this "function" node as shown in Fig.15.
- "ui-chart" node is a node that receives data from the "function" node and displays it in the form of a line chart on the Node-RED dashboard page.

After finishing creating the Node-RED flows, users can deploy the flows and monitor the temperature data on the Node-RED dashboard as shown in Fig. 16.

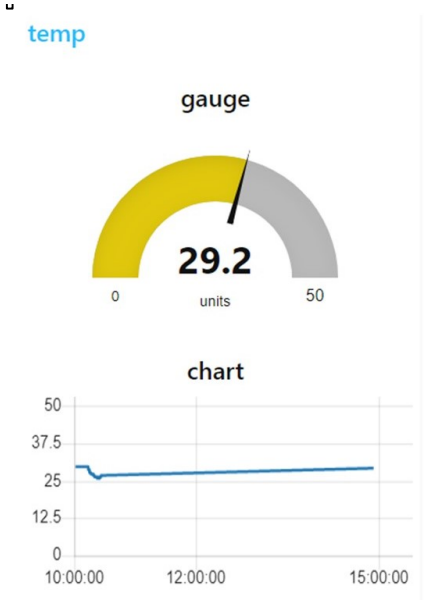

Fig. 16 The dashboard monitoring page for the testing IoT system.

V. Conclusion

We present an MQTT-based IoT cloud platform with flow design by Node-RED. The infrastructure of this platform is running on Docker. The platform is in the form of the web application developed on the MERN stack and the platform allows users creating and managing the MQTT broker instances, the Node-RED instances, and the database instances (MongoDB and InfluxDB databases).

From the test, we implemented a basic temperature IoT monitoring system, we found that the platform can support and assist well in the development of IoT applications. The users who want to develop the IoT systems or the IoT applications do not need to set up the servers and services by themselves.

References

[1] P. Datta and B. Sharma, "A survey on IoT architectures, protocols, security and smart city based applications", 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Delhi, 2017, pp. 1-5.

[2] T. Perumal, Y. L. Chui, M. A. B. Ahmadon and S. Yamaguchi, "IoT based activity recognition among smart home residents", 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), Nagoya, 2017, pp. 1-2.

[3] S. Shukla, Balachandran K and Sumitha V S, "A framework for smart transportation using Big Data", 2016 International Conference on ICT in Business Industry & Government (ICTBIG), Indore, 2016, pp. 1-3.

[4] N. Kumar, "IoT architecture and system design for healthcare systems", 2017 International Conference On Smart Technologies for Smart Nation (SmartTechCon), Bangalore, 2017, pp. 1118-1123.

[5] Merkel Dirk, "Docker: Lightweight linux containers for consistent development and deployment", Linux J., vol. 2014, no. 239, March 2014.

[6] "Node-RED Programming Guide", 2019. [Online]. Available: http://noderedguide.com. [Accessed: 16-Sep-2019].

[7] "MQTT", 2019. [Online]. Available: http://mqtt.org. [Accessed: 16-Sep-2019]

[8] "InfluxDB 1.7 documentation", 2019. [Online]. Available: https://docs.influxdata .com /influxdb/v1.7. [Accessed: 16-Sep-2019].

[9] "Documentation | CloudMQTT", 2019. [Online]. Available: http://cloudmqtt.com/docs/index.html. [Accessed: 16-Sep-2019].

[10] "FRED Docs: Intro", 2019. [Online]. Available: http://docs.sensetecnic.com/fred. [Accessed: 16-Sep-2019].

[11] "Dockerode", 2019. [Online]. Available: https://github.com/apocas/dockerode. [Accessed: 16-Sep-2019].

[12] S. Chanthakit and C. Rattanapoka, "MQTT based air quality monitoring system using node MCU and Node-RED", 2018 Seventh ICT International Student Project Conference (ICT-ISPC), Nakhonpathom, 2018, pp. 1-5.