ISSN 1816-6075 (Print), 1818-0523 (Online)
Journal of System and Management Sciences
Vol.12 (2022) No.1, pp.158-174
DOI:10.33168/JSMS.2022.0112

Implementation of Edge Computing Platform for Smart Farm using MQTT and Kafka

Seokjin Shin[1], Seonyong Eom[1] and Min Choi[1]

[1]Chungbuk National University, Cheong-ju, Republic of Korea

mchoi7812@gmail.com

Abstract. Smart farms are attracting attention as a solution to rural problems facing sustainability crises, such as slowing income, export, and growth rates due to the aging of the agricultural and livestock industries. In addition, if the concentration of atmospheric constituent gases changes due to global climate change, the amount of sunlight required for global warming and photosynthesis is not maintained, so a sufficient growth environment is not created, and quality degradation due to growth inhibition is expected, promoting aging and inappropriateness. In this study, considering these issues, an edge computing platform structure that can be quickly applied to smart farms was implemented. A smart farm consists of IoT systems. A smart farm consists of IoT systems. Basically, IoT systems have limited communication bandwidth and small code space. Typical IoT systems utilize MQTT, a low-overhead protocol, and edge computing concepts that place edge nodes near improve performance issues caused by limited communication bandwidth. Despite this adoption of MQTT and edge computing, challenges still exist. The problem with MQTT is that data order is not guaranteed. To solve this problem, the Kafka protocol generated to process continuously occurring sensor data was integrated with MQTT and applied to the Edge Computing platform to solve the problem based on the improved performance of the existing solution. So, we configure the edge node for each farm to be able to manage it. Sensor modules for smart farms are manufactured and the edge is configured to manage the data of each sensor module using the MQTT protocol. A distributed coordination system is implemented using Kafka for data interworking between each edge node.

Keywords: Edge computing, smart farm, MQTT, Kafka

1. Introduction

The population trend continues to grow and by 2050 there will be 9.6 billion people. The Food and Agriculture Organization (FAO) estimates that to provide food for all populations, at least 70% of existing food production will need to be increased [1]. To solve those challenges, the agricultural sector must adopt smart farms. The Alliance for the Internet of Things Innovation (AIOTI) defines smart farms as applications of data collection, processing, analysis, and automation technologies across the entire value chain. Smart farms are growing rapidly with the development of the Internet of Things (IoT) and cloud computing [2]. So most smart farms incorporate the Internet-of-Things (IoT) technology to transmit and receive real-time data from sensors and manage them. Accurately measuring field variables such as air, soil, etc. as well as variables such as temperature, humidity and shock encountered by vehicles during product transportation can influence and aid in plant or animal evaluation [3]. Coping with these factors and implementing an optimized farming strategy requires the patience and expertise of the farmer. However, it is difficult for all farmers to acquire these specialized skills in a short period of time [4]. Therefore, the application of IoT in the agricultural sector can control the efficiency of resource use and significantly improve production capacity. However, several challenges need to be addressed to adopt IoT One of them is processing and analyzing vast amounts of data coming from heterogeneous devices [5]. Smart farm tools are defined as helping to continuously reduce and maintain these impacts or minimize environmental constraints and reduce production costs in agricultural activities. [6]. The edge computing platform implemented in this study can be used as an actual smart farm tool according to the definition. Furthermore, processing all this collected data directly to a central server is inefficient and sometimes impractical due to available storage, limited computer-to-computer communication, unpredictable latency and energy costs. To address these challenges, we introduce the concept of an edge platform, where data processing tasks are pushed to the edge of the network. There are two major elements of the edge platform implemented in this study. The Message Queuing Telemetry (MQTT) broker and Kafka server. The MQTT broker is responsible for exchanging messages between various sensors and actuators in the smart farm, and Kafka reliably sends large amounts of data generated in the smart farm to the consumers.

2. Background

2.1. MQTT

The MQTT is a server to publish/subscribe messaging transport protocol designed to be open, simple, and easy for clients to implement. These characteristics are used in many contexts, including limited environments such as Machine to Machine (M2M) communications and the Internet of Things (IoT) [7]. Today, MQTT is used in a

variety of industries including telecommunications, gas, manufacturing, oil and automotive.

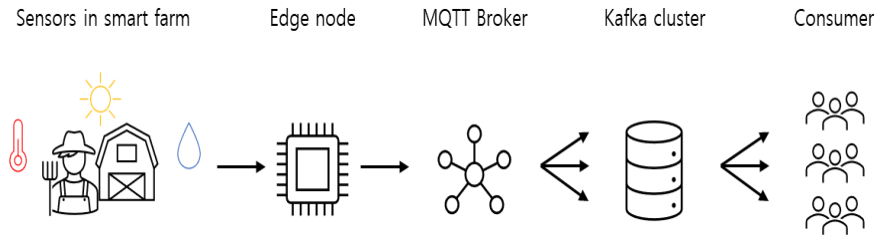

Fig. 1: Smart farm data collection analysis platform flowchart

Reliability of message delivery is important for many IoT use cases. Therefore, MQTT has three defined quality of service (QoS) levels: 0 - at most once, 1- at least once, 2 - exactly once. The QoS refers to a level that guarantees the quality of service. An appropriate QoS level should be selected according to the type of service. In this study, the QoS level was set to 0 because speed is prioritized over the reliability of data generated by sensors.

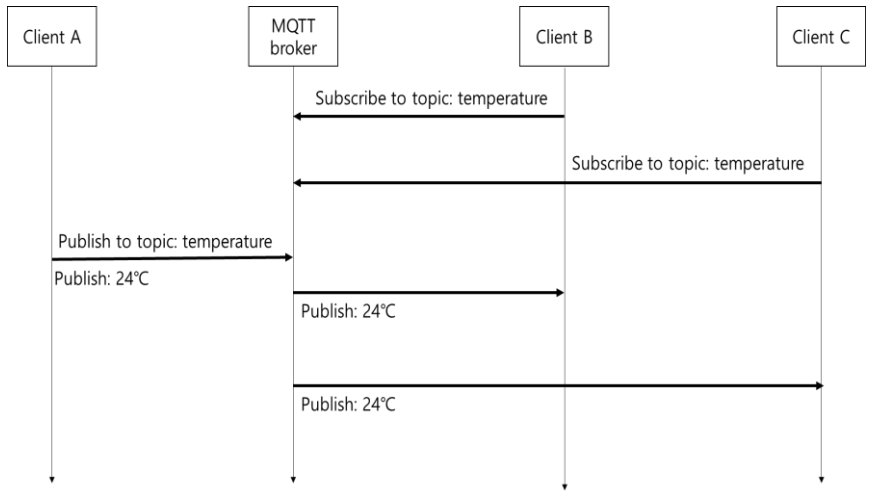

Fig. 2: MQTT broker pattern

After collecting the sensor data generated from the sensor module located in the smart farm through the edge platform, it is processed to be classified as an MQTT component by TOPIC (classified by temperature, sunlight, rainfall, etc.).

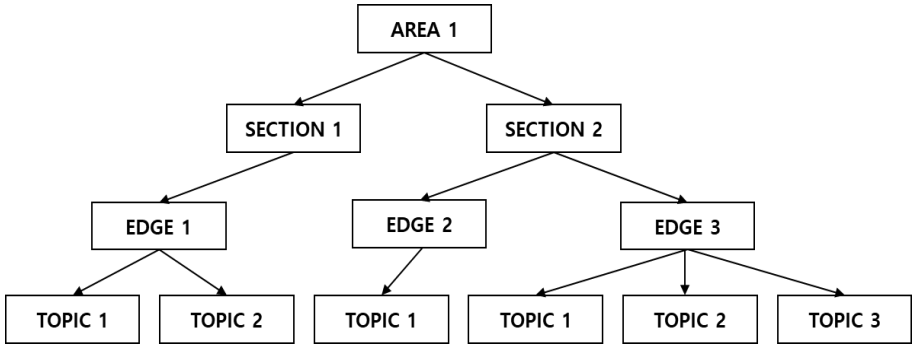

Fig. 3: Sensor data MQTT Topic classification

2.2. Kafka

Apache Kafka is a distributed, event-based, open-source streaming platform that provides a pipeline for streaming data and provides additional high-performance features for data integration and analysis [8]. It provides a publish/subscribe messaging model for data production and consumption and supports the ability to access data in real-time for real-time stream processing by allowing long-term storage of data. Kafka was designed from the ground up to provide long-term data storage and data replay. Apache Kafka has a unique approach to data persistence, fault tolerance, and replay. Therefore, this can be seen in how it handles scalability by allowing data access using cross-partition data sharing, topics/partitions, data offsets, and consumer group names for data replication persistence in clusters, increased data volume, and load.

Apache Kafka is also well suited for real-time stream processing applications because it is designed to act as a communication layer for real-time log processing. This makes Apache Kafka suitable for applications running on communications infrastructure that process large amounts of data in real-time.

Every partition has one server that acts as the leader for all read/write operations within the server, and the other server acts as a follower of this leader. If a leader goes down or fails, by default one of the followers on the other server is chosen as the new leader. Producers can generate specific messages coming to selected partitions within a topic. Consumers can consume published messages based on topics. Messages are delivered to consumer instances within the subscribing consumer group.

In Figure 4, the TOPIC-partitioned data is transmitted to the server (KAFKA Cluster) and replicated to each broker in the cluster (each partition of the server).

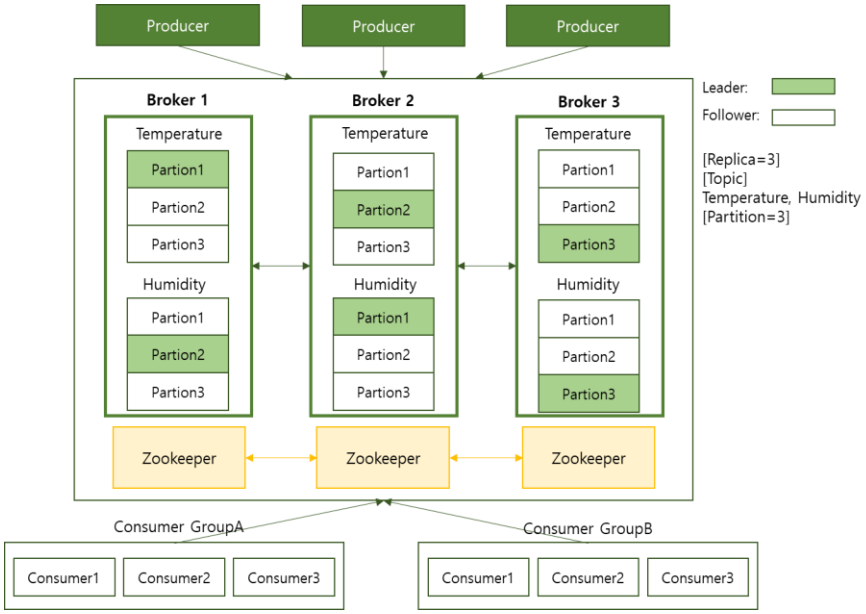

Fig. 4: Kafka cluster process

In Figure 5, the TOPIC-partitioned data is transmitted to the server (KAFKA Cluster) and replicated to each broker in the cluster (each partition of the server). Afterward, at the request of the consumer group, each broker in the KAFKA cluster designed a system capable of distributing data and transmitting large amounts of data.

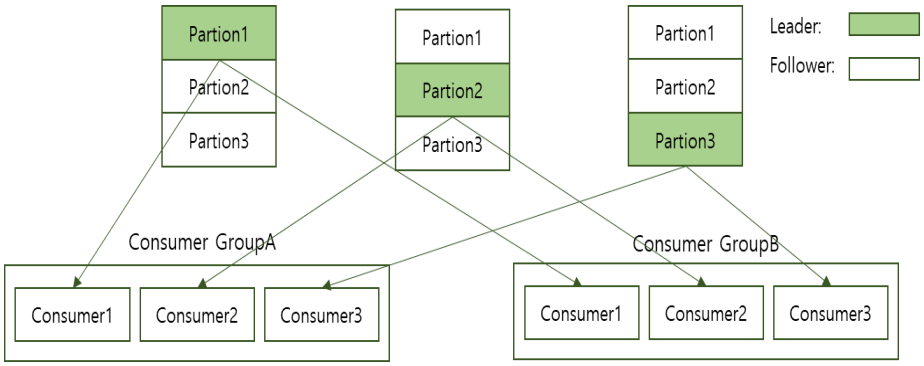

Fig. 5: KAFKA large data streaming method

2.3. Edge computing

Data is increasingly produced at the edge of the network. Therefore, a more efficient approach is to process the data even at the edge of the network. In this way, edge computing that supports the operation of stream data generated in the cloud and IoT respectively is called edge computing.

Edge computing represents a paradigm shift from centralized to decentralized. It improves the QoE by using compute resources closer to the client. In this study, an environment supporting edge computing will be referred to as an edge platform. Edge platform eliminates bottlenecks and potential points of failure and enables rapid recovery from failures. For this reason, edge platforms aim to reduce response time or latency by caching content [7]. The edge platform can be used wherever edge computing is used such as location-based, Internet of Things (IoT), data caching, big data, and sensor monitoring activity spaces, mobile cloud, and others.

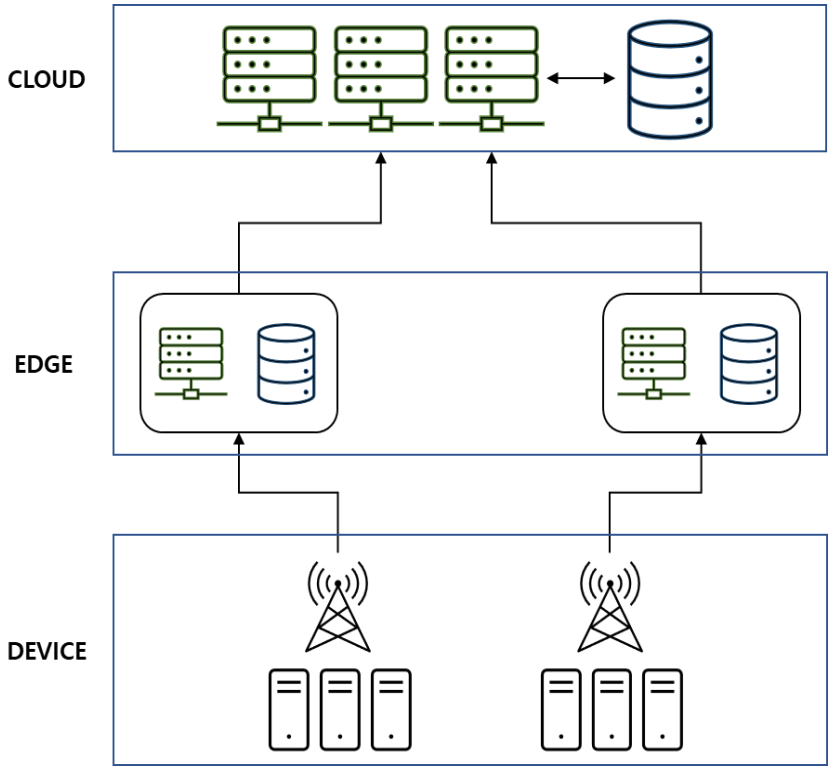

Fig. 6: Edge computing concept

3. Implementation

3.1. Hardware architecture

In a real smart farm, various sensors and actuators are required. In this study, we try to simulate an edge platform implemented using some smart farm elements without including all elements.

Fig: 7 Raspberry pi zero w

The Raspberry pi w is a tiny computer with all the peripherals found in a regular PC. Most IoT applications use the Raspberry Pi series because of its cheapness and powerful performance. So, here we use Raspberry Pi w to configure the edge node.

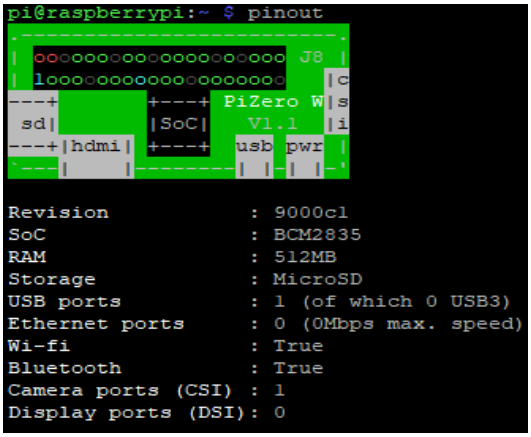

Fig: 8 Raspberry pi zero w specs

It is designed to lower the barrier to entry when applied in real agriculture using the Raspberry Pi Zero w with minimal resources, specifically 1 GHz, single-core CPU, and 512 MB RAM.

For non-contact infrared temperature measurement, the MLX90614 thermometer integrates a signal conditioning ASIC, thermopile detector chip, into a TO-39 can. It also integrates a DSP unit and a 17-bit ADC to show high resolution and accuracy. The flare lowing addresses in Table. I are used to access RAM and read-only information. Ta is the ambient temperature of the object and T_OBJ1 and T_OBJ2 are the temperature of the object.

Fig. 9 MLX90614 temperature sensor

The result has a resolution of 0.02C and is available in RAM. To is derived from RAM as:

$$\mathrm{To[K]} = \mathrm{Toreg} * 0.02, \mathrm{or}\ 0.02\ \mathrm{K}/LSB \tag{1}$$

The temperature information obtained from the MLX90614 accumulates the information in a database and is communicated to multiple users of that temperature through an edge platform. Real-time temperature information is displayed to the web service user, and the actuator operates according to the temperature.

Table 1: MLX90614 temperature sensor RAM addresses

RAM (32x16)		
Name	Address	Read access
Melexis reserved	0x00	Yes
Melexis reserved	0x03	Yes
Raw data IR channel 1	0x04	
Raw data IR channel 2	0x05	
T_A	0x06	Yes
T_{OBJ1}	0x07	Yes
T_{OBJ2}	0x08	Yes
Melexis reserved	0x09	Yes
…	…	…
Melexis reserved	0x1f	Yes

In practice, several types of actuators are used. In this study, the role of this actuator is assumed to be an SG90 servo motor. The ultra-compact, high-power SG90 servo motor can rotate 90 degrees in each direction and is smaller than other products. Servo motors provide feedback on whether the data obtained from the sensor is being processed properly.

Fig. 10: SG90 servo motor

Each hardware introduced above operates on an independent IoT device and communicates with edge nodes using the MQTT protocol, which is relatively lightweight compared to HTTP. Each hardware interacts with the edge platform and exchanges large amounts of data. Each sensor is not interconnected and operates through an edge platform. This means that data can be processed efficiently without unnecessary communication.

3.2. Software architecture

The overall structure mainly consists of five elements. First, the sensor installed on the actual smart farm side and the edge node responsible for it are considered as one element. Second, it is an MQTT broker that brokers MQTT data. Based on the pub/sub model, an MQTT broker remembers multiple subscribers subscribed to a particular topic and forwards the data as it is received. The third is the edge platform. The edge platform in this study serves to connect MQTT and Kafka protocol. It also plays a role in reducing the burden on IoT devices in charge of sensors by placing the device serving as the corresponding edge platform physically close to the smart farm. The fourth is the Kafka cluster. Kafka clusters communicate using the Kafka protocol. Like MQTT, it uses a pub/sub model and serves to stream large amounts of continuous data from sensors based on events. Finally, it uses a client that uses Kafka data. In this study, a web client was configured using Thingsboard, an open-source IoT dashboard platform, and designed to store data by linking MongoDB and Kafka client.

The whole system consists of an edge platform with multiple MQTT clients and multiple edge nodes. Each client connects to the edge platform to send and receive data. The edge node is mainly composed of two components, the MQTT component, and the Kafka server. The MQTT component is a broker and subscriber to the MQTT protocol. This allows for immediate data transfer as well as other operations after receiving the data. In this study, specific data is sent to Kafka after receiving data from a component for reliable data storage and transmission.

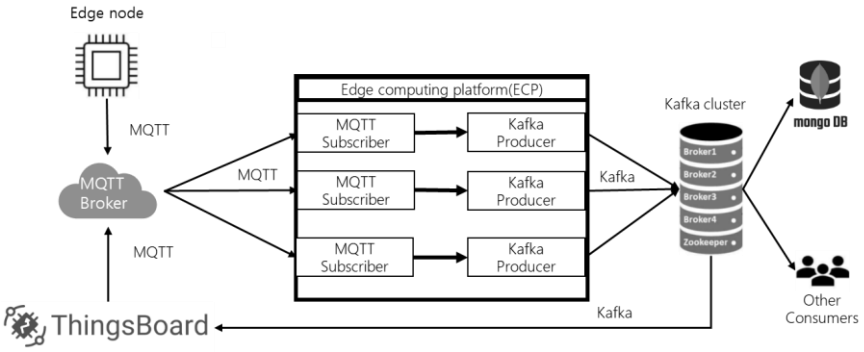

Fig. 11: System architecture

The data generated by the sensor is processed by Node-MCU and sent to the MQTT broker. The broker then sends it to an edge platform with multiple subscribers. The edge platform internally switches from the MQTT protocol to the Kafka protocol and sends it to the Kafka cluster. After receiving data from the Kafka cluster, it forwards the received data to multiple consumers.

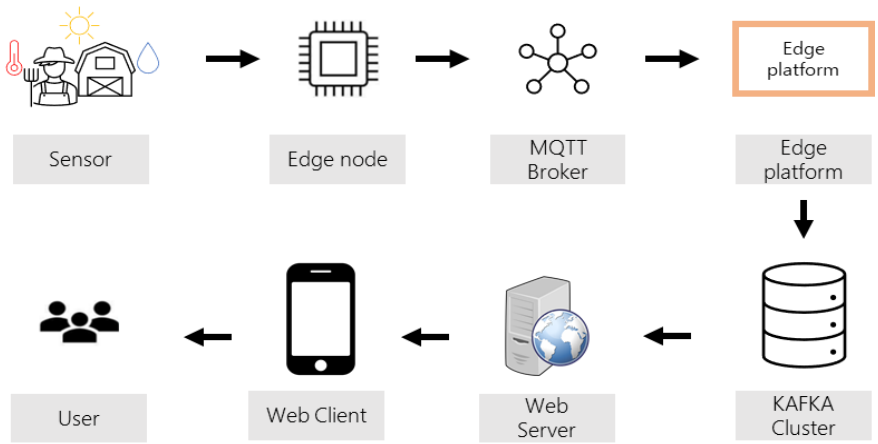

Fig. 12: System data flow chart with the web client

We can easily create IoT dashboards for real-time data visualization and remote device control using WebSocket-based services provided by Thingsboard. With over 30 custom widgets, it can build end-user custom dashboards for most IoT use cases. It collects and stores telemetry data in a scalable and fault-tolerant way. It visualizes data with built-in or custom widgets and flexible dashboards. It also can define data processing rule chains. it transforms and normalizes your device data, and raises alarms on incoming telemetry events, attribute updates, device inactivity, and user actions.

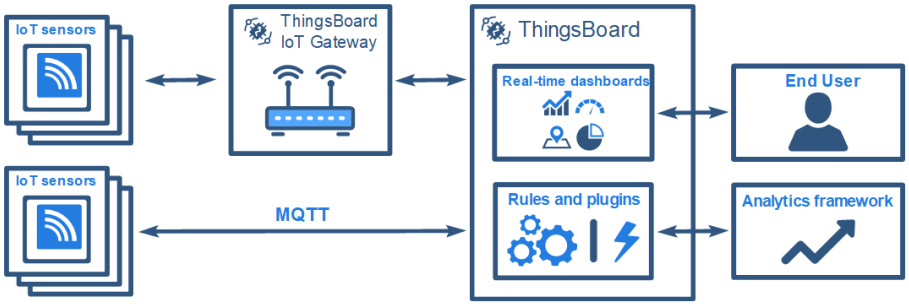

Fig. 13: Thingsboard system architecture

Table. II shows the modules used in this study. The runtime in which the server runs is composed of Node.js based on the JavaScript language, and a package suitable for Node.js is configured so that the server can work well.

Table 2: Using modules

	modules	version
Run time	Node.js	v14.17.3
Webserver	Thingsboard	v3.3.1
MQTT client	mqtt.js	v0.46.1
MQTT broker	aedes.js	v4.2.8
Database	MongoDB	v4.4.6
Database client	mongodb.js	v4.1.1
Kafka server	Apache Kafka	v2.8.0
Kafka client	kafka.js	v1.15.0

When a sensor publishes data on a specific topic, the MQTT component receives it and classifies it into direct processing data and data processing through Kafka. After an MQTT client establishes a connection to an MQTT broker, it is set up to send sensor data connected to that edge node every 100ms. By maintaining the established connection between the client and the broker, the burden on the expensive part of the network connection is reduced. Many sensor data take constant values except under special circumstances where it exhibits unusual values. In this case, it is important to reliably transfer the desired data between successive sets of data. Kafka within the edge platform does this. In Kafka, the received data is shared on the edge platform and data is shared with multiple consumers who consume the data. The server processing data sends to different services. When data is sent through Kafka, consumers of such as databases and web servers consume the data immediately and proceed as follows:

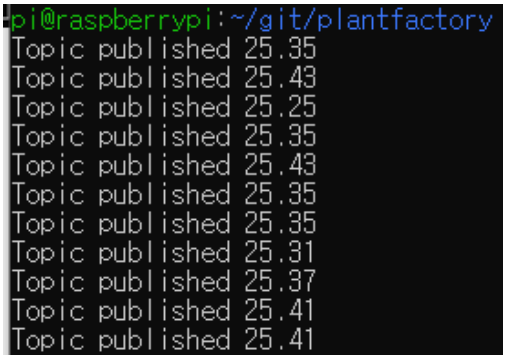

Fig. 14: Temperature published by Node-MCU

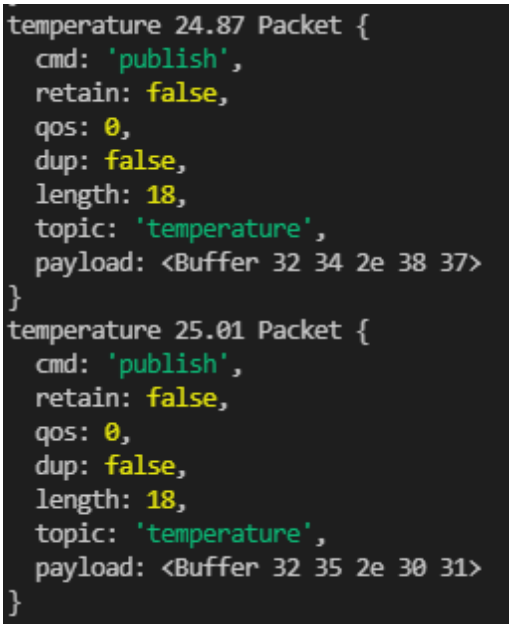

Fig. 15: Temperature topic subscriber

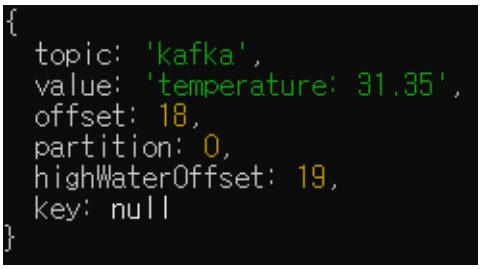

Fig. 16 Kafka consumer subscribed to temperature topic

After receiving data from the Kafka cluster, data is accumulated through the MongoDB connector. The query result in MongoDB that processes the data received from the Kafka cluster is as follows.

MongoDB, a type of NoSQL database, is a document-oriented database system. For cross-platform support, avoid traditional relational database structures and use dynamic schema-typed documents such as JSON. This makes data integration for specific kinds of applications easier and faster. Since the runtime of this study is node.js, communication with JSON-based DB made development more efficient. It is judged that it will be easy to store image data of smart farms in the future by utilizing these document-oriented JSON-based characteristics.

```
> db.temperature.find({})
{ "_id" : ObjectId("613971936f5b64ad6b9ee464"), "1631154579442" : 25.69 }
{ "_id" : ObjectId("613971936f5b64ad6b9ee465"), "1631154579556" : 25.67 }
{ "_id" : ObjectId("613971946f5b64ad6b9ee466"), "1631154580529" : 25.75 }
{ "_id" : ObjectId("613971956f5b64ad6b9ee467"), "1631154581550" : 25.73 }
{ "_id" : ObjectId("613971966f5b64ad6b9ee468"), "1631154582529" : 25.73 }
{ "_id" : ObjectId("613971976f5b64ad6b9ee469"), "1631154583584" : 25.73 }
{ "_id" : ObjectId("613971986f5b64ad6b9ee46a"), "1631154584566" : 25.67 }
{ "_id" : ObjectId("613971996f5b64ad6b9ee46b"), "1631154585527" : 25.79 }
{ "_id" : ObjectId("6139719a6f5b64ad6b9ee46c"), "1631154586560" : 25.79 }
{ "_id" : ObjectId("6139719b6f5b64ad6b9ee46d"), "1631154587521" : 25.75 }
{ "_id" : ObjectId("6139719c6f5b64ad6b9ee46e"), "1631154588528" : 25.81 }
{ "_id" : ObjectId("613971d36f5b64ad6b9ee46f"), "1631154643989" : 25.69 }
{ "_id" : ObjectId("613971d46f5b64ad6b9ee470"), "1631154644121" : 25.79 }
{ "_id" : ObjectId("613971d56f5b64ad6b9ee471"), "1631154645058" : 25.75 }
{ "_id" : ObjectId("613971d66f5b64ad6b9ee472"), "1631154646065" : 25.73 }
{ "_id" : ObjectId("613971d76f5b64ad6b9ee473"), "1631154647088" : 25.75 }
{ "_id" : ObjectId("613971d86f5b64ad6b9ee474"), "1631154648072" : 25.81 }
{ "_id" : ObjectId("613971d96f5b64ad6b9ee475"), "1631154649077" : 25.73 }
{ "_id" : ObjectId("613971da6f5b64ad6b9ee476"), "1631154650072" : 25.69 }
{ "_id" : ObjectId("613971db6f5b64ad6b9ee477"), "1631154651095" : 25.79 }
```

Fig. 17: Temperature Sensor data in MongoDB

The web dashboard utilizes the sensor data delivered to the DB through the edge platform and visualizes it through the graph tool on the web using Thingsboard. It communicates with the edge platform through WebSocket, where the status of the smart farm can be checked in real-time. Figure 18 is the dashboard implemented in this study. It shows a real-time time-series graph according to the access time. Location information can also be managed as longitude and latitude values and displayed on a map based on these values. The criteria for the alarm function can be set by the user, so if the criteria are out of range, an alarm is automatically displayed on the dashboard. It can also operate connected actuators via the RPC API provided by Thingsboard.

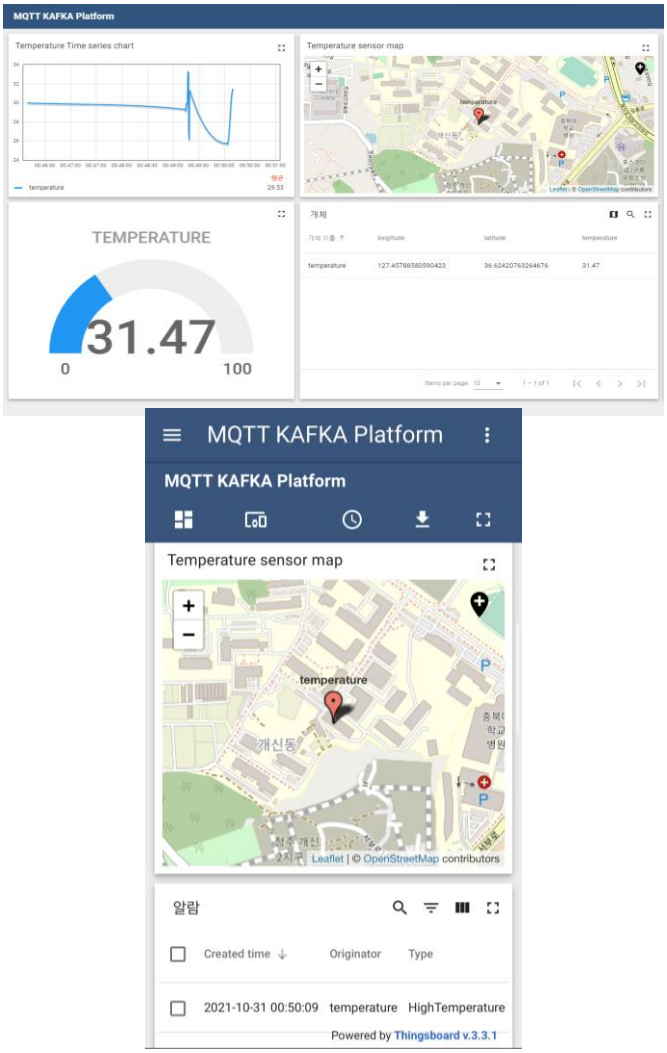

Fig. 18: Thingsboard IoT web dashboard

4. Performance Evaluation

In this paper, we use Apache JMeter For the performance evaluation of the designed MQTT Kafka-based edge computing. Apache JMeter is an open-source software, a Java application designed to load functional behavior and measure performance. It provides extended functionality from its original purpose of testing web applications to other testing capabilities. Plug-ins supporting various protocols are additionally configured to use MQTT Kafka and edge computing platform. In the case of the Kafka client, the consumer creation function was insufficient, so it was additionally configured using the JSR223 script.

Apache JMeter version 5.4.1 used as a client-server structure was used, and a virtual MQTT and Kafka client were created on the test server to perform integrated analysis in a mixed environment of the two protocols. Assuming one IoT client, 30 iteration evaluations per thread were performed. We did one MQTT client connection and termination for each thread, 50 MQTT publishes, and one consumer creation and termination. Plugins and extensions are required to handle MQTT and Kafka clients in JMeter. For MQTT, connect, terminate, and publish are provided independently. However, for Kafka, it comes with a producer and consumer pair with integrated connection and termination capabilities. In the case of the consumer, to use the necessary functions, a script must be created using JSR223 created for script support in Java. Therefore, considering this environment, in the case of MQTT, connect, publish, and terminate are recognized as one work process and compared with Kafka consumer. Table III shows that the average response time of MQTT clients is 3.9 times faster than the average response time of Kafka clients.

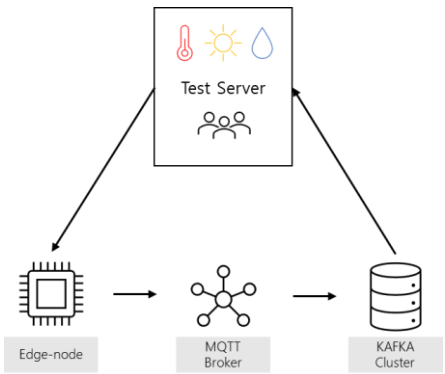

Fig. 19: Performance evaluation environment configuration

We can view the overall information and graphs of the performance evaluation performed through the listener provided by JMeter. The listener provides the number of responses, average value, minimum value, maximum value, standard deviation, error rate, bandwidth, received data size, transmitted data size, and average data size.

Figure 20 is a graph showing the table comparing the average response time described above according to the execution time. We can see that the response times for MQTT disconnects and publishes are less than 100ms. Since MQTT and Kafka are both TCP-based protocols, the initial response time is long due to the initial connection setup of socket communication. There is a difference of about 100ms between the Kafka client and the MQTT client, and after that, the difference of about 400ms continuously occurs during data transmission and reception. Through this, the MQTT Kafka platform implemented in this study is effective for use in environments where network bandwidth is limited, or a large amount of data is continuously transmitted and received.

Table 3: Performance evaluation results for response time

Label	Samples	Average	Std. Dev.	Throughput
MQTT Connect	30	76	180.83	2.24316
MQTT Publish	1500	0	0.44	121.8225
MQTT Disconnect	30	14	1.9	2.44021
Kafka Consumer	30	352	155.63	2.3855
TOTAL	1590	8	58.79	115.9991

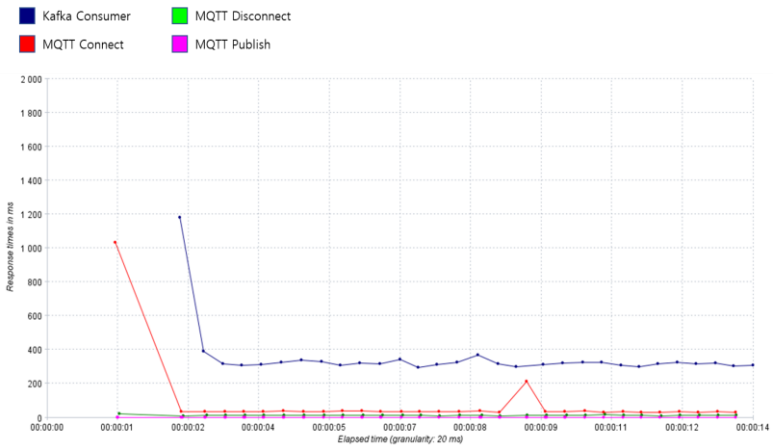

Fig. 20: Performance evaluation result graph

02:42:37.693	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.693	MQTT Kafka Grou...	MQTT Publish	1
02:42:37.694	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.694	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.695	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.695	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.695	MQTT Kafka Grou...	MQTT Publish	0
02:42:37.696	MQTT Kafka Grou...	MQTT Disconnect	23
02:42:37.725	MQTT Kafka Grou...	Kafka Consumer	1184
02:42:38.912	MQTT Kafka Grou...	MQTT Connect	36
02:42:38.948	MQTT Kafka Grou...	MQTT Publish	1

Fig. 21: Performance evaluation result graph

5. Conclusions

Smart farms can be built easily and quickly through the edge platform using MQTT and Kafka implemented in this study. In smart farms, efficiency can be increased based on massive sensor data, and sales can be expected to increase as production increases. In addition, as the time and technology required to build a smart farm decrease, the burden of continuous change and the introduction of technology is expected to decrease. Through the smart farm edge platform technology using MQTT and KAFKA developed in this study, we can expect to increase the sales of the farm and contribute to the continuous development of big data-based services in the future.

In addition, the edge computing platform is expected to reduce the cost of sensor replacement by contributing to compatibility in real farms where sensors need to be constantly replaced.

References

Muhammad Rusyadi Ramli, Philip Tobianto Daely, Dong-Seong Kim, Jae MinLee (2020). IoT-based adaptive network mechanism for the reliable smart farm system. *Computers and Electronics in Agriculture*, 170, No.105287.

H. Sundmaeker, C.N. Verdouw, J. Wolfert, Luis Perez Freire (2016). Internet of food and farm 2020. *Digitising the industry*, 49, 129-150.

Jirapond Muangprathub, Nathaphon Boonnam, Siriwan Kajornkasirat, Narongsak Lekbangpong, Apirat Wanichsombat, Pichetwut Nillaor (2019). IoT and agriculture data analysis for smart farm, *Computers, and Electronics in Agriculture*, 156, 467-474.

Jesús María Domínguez-Niño, Jordi Oliver-Manera, Joan Girona, Jaume Casadesús (2020). Differential irrigation scheduling by an automated algorithm of water balance tuned by capacitance-type soil moisture sensors. *Agricultural Water Management*, 228, No.105880.

Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu (2016). Edge Computing: Vision and Challenges, *IEEE Internet of Things Journal*, 3(5), 637-646.

S. Fountas, G. Carli, C. G. Sørensen, Z. Tsiropoulos, C. Cavalaris, A. Vatsanidou, B. Liakos, M. Canavari, J. Wiebensohn, B. Tisserye (2015). Farm management information systems: Current situation and future perspectives. *Computers and Electronics in Agriculture*, 115, 40-50

Biswajeeban Mishra, Attila Kertesz (2020). The use of MQTT in M2M and IoT systems: A survey. *IEEE Access*, 8, 201071-201086

Bunrong Leang, Sokchomrern Ean, Ga-Ae Ryu and Kwan-Hee Yoo (2019) Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment. *Sensors*, 19(1), 134.

Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, Hannu Flinck (2017). Mobile Edge Computing Potential in Making Cities Smarter. *IEEE Communications Magazine*, 55(3), 38-43.