# Performance Evaluation of Containerized Systems before and after using Kubernetes for Smart Farm Visualization Platform based on LoRaWAN

1st Sungjin Park
*Computer Science and Engineering*
*Chungbuk National University*
Cheongju, South Korea
huitseize@chungbuk.ac.kr

2nd Haegyeong Im
*AI Convergence*
*Soongsil University*
Seoul, South Korea
fine632@soongsil.ac.kr

3rd Gayoung Yeom
*Computer and Electronic Systems Engineering*
*Hankuk University of Foreign Studies*
Gyeonggi, South Korea
gayoung@hufs.ac.kr

4th Dayeon Won
*Information of Convergence*
*Kwangwoon University*
Seoul, South Korea
aakk9350@kw.ac.kr

5th Minji Kim
*Computing Engineering*
*Jeju National University*
Jeju, south Korea
minzyk0729@jejunu.ac.kr

6th Xavier Lopez
*Cybersecurity and Network Engineering*
*Purdue University*
West Lafayette, United States
lopezx@purdue.edu

7th Smith Anthony
*Computer and Information Technology*
*Purdue University*
West Lafayette, United States
ahsmith@purdue.edu

***Abstract*—To meet the demand for food following the increase in food production, smart farms have emerged with IoT, and most smart farms gather data using Low Power Wide Area Network protocols such as LoRa. In this paper, we designed a platform that farmers can visually check real-time information of the smart farms based on LoRaWAN, and intend to confirm the increase in efficiency by applying Kubernetes in this service. After building a Web Server, the performance evaluation is shown through experiments that compare CPU usage, response time, and throughput before and after applying Kubernetes.**

***Index Terms*—Kubernetes, distributed computing, LoRaWAN, Docker**

## I. Introduction

According to the United Nations (UN), the world population is expected to be over 9 billion people by 2050 [1]. This population growth requires an increase in food production by about 70 percent, following the Food and Agriculture Organization of the United Nations [1]. To meet the growing demand for food, smart agriculture systems based on the Internet of Things (IoT) have been presented [2]. It makes agriculture easy and cost-effective by reducing labor costs and improving crop yield. IoT in agriculture collects data from wide areas. Hence, Low Power Wide Area Network (LPWAN) protocols are needed in order to accumulate the data with reliable communication [3]. LoRaWAN is suggested as the most acceptable communication network for LPWAN of smart agricultural IoT [4]. This collected data provides information about various environmental factors [5], [6].

A data visualization platform for farmers is designed in this study. Farm data from LoRaWAN is transmitted to the SENET and it is provided to farmers through the web interface. Moreover, Kubernetes is introduced in this platform to improve efficiency. It is an open-source platform that has some advantages such as monitoring, auto-scaling, and self-healing. Kubernetes configures servers optimally by monitoring the status of resources in real-time. If servers are down, it recovers itself automatically. This platform was built as containerized applications for fast and easy deployment. Docker was applied to build and host containers.

Nevertheless, Kubernetes can have a high learning curve and opportunity cost to put in existing services. However, the application of Kubernetes is expected to be highly effective in terms of its performance and resource. Therefore, in this paper, the efficiency of performance is investigated by comparing before and after using Kubernetes.

The rest of this paper is organized as follows. Section 2 explains the concept of LoRaWAN, Docker, and Kubernetes used in data collection and processing. Section 3 deals with the overall system architecture of this study. Section 4 evaluates the performance of the containerized system before and after using Kubernetes through experiments. Section 5 finishes papers while showing the extension of the system.

## II. Related Works

### A. LoRaWAN

IoT enables farmers to manage overall farms meticulously. There are many IoT network systems for agriculture. Among them, especially LoRaWAN is implemented in farming. [7] It is suitable for IoT which should be able to cover a wide range and get many sensing data. According to this advantage, many nations use the LoRaWAN network protocol on farms. LoRaWaN can cover 2-5km in urban, and 15km in rural. [7]

In addition, battery usage is an important thing in IoT. As LoRaWAN uses asynchronization system, it is advantageous for running a smart farm due to its good battery efficiency. [8]

### B. Docker

Virtualization is widely used in IT services. There are many virtualization technologies. Docker is one of them, which has many benefits compared with other virtualization methods. Among them, docker containers and virtual machines are mainly compared.

Rad et al, concluded that Docker provides some advantages, which are helpful for developers and administrators. It is an open platform that can be made use of building, distributing, and running applications in a portable, lightweight runtime, and packaging tool, known as Docker Engine. [9]

Potdar et al, conducted performance evaluation on virtual machine and Docker container-based hosts in terms of CPU Performance, Memory throughput, Disk I/O, Load test, and operation speed measurement. It is observed that Docker containers perform better over VM in every test, as the presence of Quick EMUlator (QEMU) layer in the virtual machine makes it less efficient than Docker containers. [10]

### C. Kubernetes

As containerized applications have gained much importance in recent years, much attention has been also drawn to tools for supervising containers. There are many orchestration tools that manage the life cycle of containers and services. One of them, Kubernetes has become a de facto standard due to outstanding features and popularity. It provides functions for load balancing, auto-scaling, and self-healing which help for maintainability, scalability, and reliability. [11]

Based on these references, the performance of containerized servers using Docker and Kubernetes for smart farm data visualization platform based on LoRaWAN was evaluated.

## III. Methodology

This system is a weather data visualization platform designed for farmers who have smart farms. Fig. 1 shows the overall system architecture from sensors to users focused on functions.

### A. Network

Real-time weather data from weather API is processed. The data are stacked in the ChirpStack server and transmitted to an Application Server at regular intervals. The process takes place over the LoRa network.

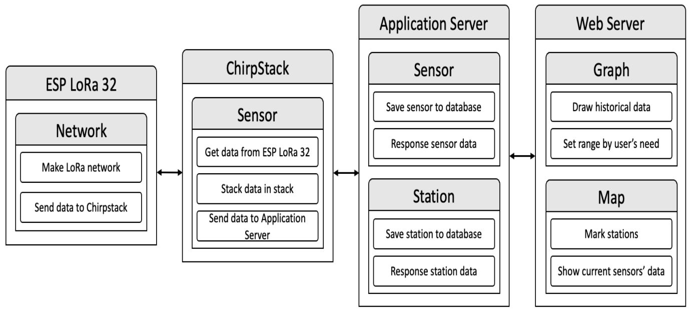

Fig. 1. the overall system architecture

### B. Application Server

The Database stores information about sensors and stations, and the Application Server saves or provides data in the Database at the request of the client. The clients of the Application Server are the ChirpStack server and the Web Server. The ChirpStack server requests to save sensor data, and put it in the Database. If the Web Server requests the sensor data, the Database returns it.

### C. Web Server

The Web Server configures the main page consisting of graphs and a map based on the user's current location. Users could view the historical data through graphs, and find the locations of other stations and the data corresponding to this on the map.

### D. Kubernetes

Kubernetes manage effectively hardware resources controlling the number of pods according to the real-time load of running containers related to the Web Server and Application Server. When making the pods increase, the node with the least load is selected and uses the resource evenly. Kubernetes also keep the optimal state by removing the node with the least throughput when decreasing the pods.

In addition, pods are distributed across multiple nodes, allowing the service to operate normally if they have just one node alive, even if some nodes fail or are down. At this point, the pods having an error are replaced with other pods or restarted so that they can work normally again.

### E. Physical Architecture

Fig. 2 represents the physical architecture which is a data flow from farms to a device of farmers. This system has four real-time weather data of farms such as temperature, humidity, pressure, and wind speed. The types of sensors are as follows.

The weather data is gathered from the OpenWeather API server with ESP32 LoRa which is the Micro Controller Unit(MCU) using WiFi. They flow from the LoRa gateway to ChirpStack using LoRa in ESP32 LoRa. The ChripStack server processes received data in an array format, and push it to the Application Server using the HTTP protocol.

The Application Server was developed by a framework, Spring Boot, based on Java. It is linked to the Database as

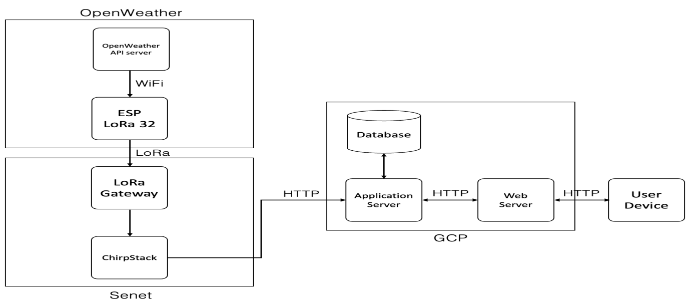

Fig. 2. the overall physical architecture

a Spring Data Java Persistence API(JPA) and the Database was built in Mysql. The Application Server parses the weather data from the ChirpStack server to the array class and put it in the Database. When the Web Server requests the data, the Application Server selects it and returns it to the Web Server. The Web Server gets the data and displays it to Interface.

The Web Server was developed by a javascript-based library, React. The UI is configured by styled-components, and the data was brought using APIs of the Application Server. A State of a weather station is managed by react-redux. The initial status is based on the user's current location, and thereafter, according to the weather stations requested by the user, the corresponding sensor data are visualized through graphs and a map on the main page. The graphs and map are used react-apexchars and react-map-gl library respectively.

The Application Server and Web Server were containerized by Docker and run on the Compute Engine of Google Cloud Platform(GCP). The Database was also built on Cloud SQL in GCP.

### F. Web Site

When a user accesses the website, farm data is displayed based on the current location. Fig. 3 is the main page showing real-time data of the present farm via graphs and a map. The users can monitor historical data in graphs, and current data on the map.

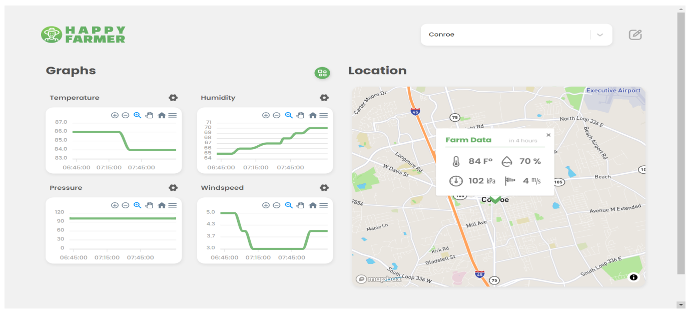

Fig. 3. the main page

Fig. 4 is a page in which a user sets a range appearing in each graph, and the user can directly set the high and low values in every single sensor. Fig. 5 is a page after setting the range of sensors. It makes it possible for farmers to identify visual trends as they can see a meaningful range of data for themselves. It means that they can perceive whether the crops are in an optimized state to grow well by identifying environmental factors for one's farm.

Moreover, the farmers can check real-time data from other farms as well as their own through the search function.

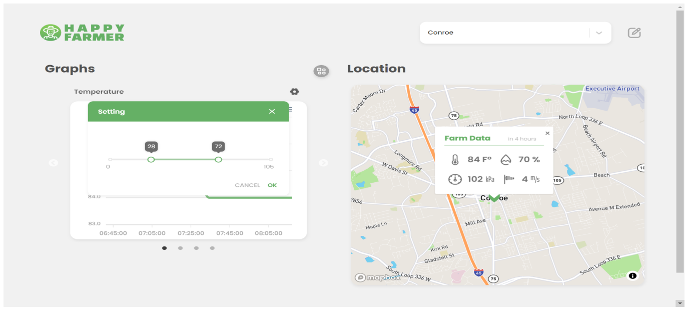

Fig. 4. before setting a range of graph

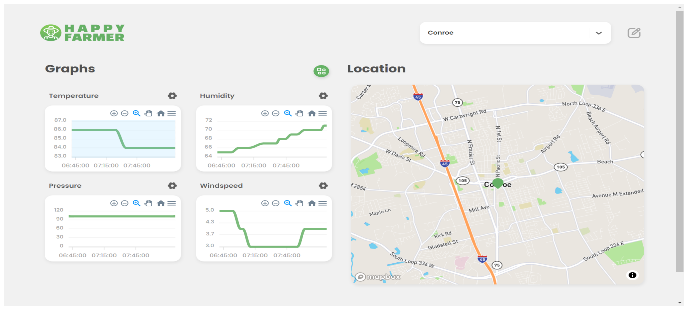

Fig. 5. after setting a range of graph

### G. the structure of Kubernetes

The web side of this system is managed by Kubernetes. Its structure looks like Fig. 6. The Kubernetes engine is set up as a cluster by using Google Kubernetes Engine(GKE). The master nodes operate for each Application Server and Web Server in the cluster. Application Server and Web Server are dockerized as images, each running inside of containers. These containers are in the pods and the pods are in the worker nodes.

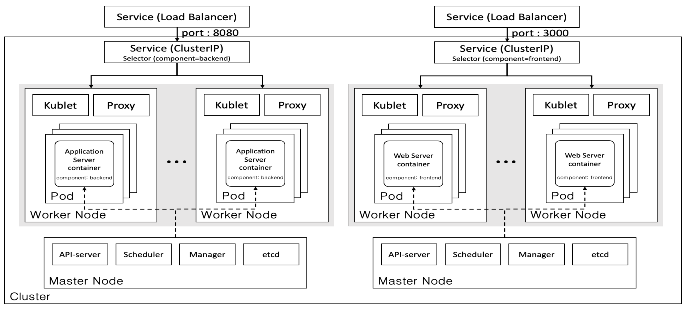

Fig. 6. the structure of Kubernetes

The worker nodes are controlled by the master node and the process is as follows: Each worker node's kublet monitors the status of the node and sends it to the master node. The master node stores the state of the node in etcd. The controller watches the status of nodes and makes the status of the cluster optimally by controlling the number of nodes and pods.

The pods in each field are grouped by an abstract unit which is called Service. The pods have their own IP addresses that can be accessed internally, which must be grouped into services for exposure on the Internet. The service is exposed again as connecting with TCP/UDP Load Balancer. Therefore, the services are determined by the Selector. Each set of the back-end and front-end pods is targeted by each service. The LoadBalancer type services are placed for connecting with external traffic as Internal TCP/UDP Load Balancer in GKE. After the external traffic come in through the TCP/UDP Load Balancer, the Service which is set with the type LoadBalancer distributes the traffic to worker nodes using its own IP address in a cluster.

### H. *Testing environment - Kubernetes*

The test was carried out using Locust which is a distributed load testing tool based on python. The load testing is conducted by giving a user request traffic to the Web Server. At this point, as large-scale traffic is needed, the distributed load testing is carried out in GKE, not in a single host.

The structure of the testing server is like Fig. 7. There is a cluster and a master node controlling the several worker nodes. The number of worker nodes is two and the pods are five. They carry out the test. Depending on the amount of load to be tested, the number of pods and worker nodes in the web server changes, and each test container is built as a Locust.

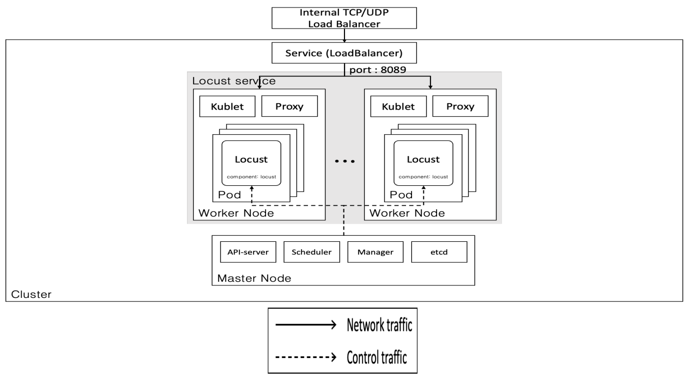

Fig. 7. the structure of load testing server

## IV. Experiments

We report on two models: Single System Model(SSM) and Distributed System Model with Kubernetes(DSMK). SSM consists of an Application Server and a Web Server on Compute Engine. DSMK has any number of Application Servers and Web Servers.

The hardware specifications of SSM are equal to the total hardware specifications of pods in DSMK. DSMK extends pods according to CPU Usage. When the maximum number of arbitrary pods is reached, the pods no longer increase and the server uses the maximum resources. Allocated resources of DSMK increase or decrease as CPU Usage. On the other side, the allocated resources of SSM are fixed, so SSM always uses maximum resources.

The hardware specification is defined in table 1 and 2. The number of CPU cores is increased when the usage exceeds 80 percent in DSMK. The pods which belong to the same service have the same hardware specification.

TABLE I
THE HARDWARE SPECIFICATION OF DSMK

| | Application Server | Web Sever |
|---|---|---|
| The number of CPU Core | 0.25 | 1 |
| Memory (MB) | 512 | 2048 |
| The maximum number of pods | 4 | 4 |

TABLE II
THE HARDWARE SPECIFICATION OF SSM

| | Application Server | Web Sever |
|---|---|---|
| The number of CPU Core | 1 | 4 |
| Memory (MB) | 2 | 8 |
| The maximum number of pods | 1 | 1 |

Distributed load testing is run with Locust on Kubernetes. The test was carried out by gradually increasing the number of user requests. The changes in CPU Utilization, Total Requests Per Second (RPS), and Response Times are monitored.

The target host to request is a Web Server. As one request to the Web Server makes a chain of transactions to the Application Server, a virtual user requests to the Web Server's home page.

The test process is shown in Table 3. The test turn is divided into three sections: The load test operates in the first and third sections. The second section is paused without loading.

TABLE III
THE METHOD OF TEST

| | 1st trial | Pause | 2nd trial |
|---|---|---|---|
| The number of users | 250 | 0 | 500 |
| Spawn rate | 10 | 0 | 20 |
| Time (MM:SS) | 4:00 | 00:30 | 06:30 |

### A. *Testing CPU Usage*

The below graphs show the CPU utilization of each model. CPU utilization on DSMK is higher than SSM. Even CPU utilization can exeeds CPU allocation in DSMK (a).

However, this suggests that efficient usage of the available CPU core We take this as evidence that Kubernetes helps ensure that the application always has the right amount of capacity to handle the current traffic demand, as the number of pods is changed by Auto Scaling.

### B. *Testing Web Server*

The below table data shows that DSMK's average RPS is higher than SSM's. It seems like that DSMK distributes traffic to each pod by load balancing, so the bottleneck phenomenon is not caused. The variance of DSMK is lower than SSM's. Thus DSMK is more stable.

TABLE IV
RPS STATISTICS

| Load Test | DSMK | | SSM | |
|---|---|---|---|---|
| | RPS | Variance | RPS | Variance |
| 1st trial | 1124.0 | 30983.4 | 1374.5 | 226118.0 |
| 2nd trial | 3052.6 | 190261.3 | 1044.9 | 58170.7 |

The below response times of DSMK on the first trial are slower than SSM's as depicted in table IV. However, the response times of DSMK on the second trial are faster than SSM's. The gap between the 50th percentile and 90th percentile of DSMK decreases in the second trial unlike the gap of SSM. As seen in fig. 2, CPU allocation is increased after starting the second trial. We take this as evidence that as the number of pods increased, the servers on DSMK response faster than the servers on SSM without bottleneck.

TABLE V
RESPONSE TIMES STATISTICS

| Load Test | DSMK | | | SSM | | |
|---|---|---|---|---|---|---|
| | 50% | 95% | Gap | Variance | RPS | Gap |
| 1st trial | 180 | 630 | 450 | 110 | 280 | 170 |
| 2nd trial | 140 | 250 | 110 | 200 | 960 | 760 |

## V. Conclusion

A web server that shows real-time data is run on Kubernetes. The CUP usages are higher on average than a single server when the web server deploys with Kubernetes. However, it uses the resources efficiently as it controls the CPU resources following the traffic with auto-scaling. The Requests per Second of the website are higher than that of a single server by a server with Kubernetes. This is due to the load-balancing function of Kubernetes, which distributes traffic. In addition, on the server with Kubernetes, the Response time of user requests can be handled consistently as traffic increases, it stabilizes. Through these experiments, the server to that Kubernetes was applied has some strengths in resource processing efficiency, high throughput, and response stability compared to a single server. In the future, if there are a lot of users coming into this service, it will be expected the advantages of Kubernetes will exert more effect.

## References

[1] Gupta, M., et al., ”Security and privacy in smart farming: Challenges and opportunities,” IEEE Access vol. 8, 2020, pp. 34564-34584.

[2] Bu, F., and X. Wang, ”A smart agriculture IoT system based on deep reinforcement learning,” Future Generation Computer Syst. vol. 99, 2019, pp. 500-507.

[3] Gutiérrez, S., et al., “Smart mobile LoRa agriculture system based on internet of things,” 2019 IEEE 39th Central America and Panama Conv. (CONCAPAN XXXIX), pp. 1-6.

[4] Miles, B., et al., “A study of LoRaWAN protocol performance for IoT applications in smart agriculture,” Computer Commun. vol. 164, 2020, pp. 148-157.

[5] Gondchawar, N., et al., “IoT based smart agriculture,” Int. Journal of advanced research in Computer and Commun. Eng., 2016, pp. 838-842.

[6] Ji, M., et al., “Lora-based visual monitoring scheme for agriculture IoT,” 2019 IEEE sensors appl. symp. (SAS), pp. 1-6.

[7] D. Davcev, K. Mitreski, S. Trajkovic, V. Nikolovski and N. Koteli, ”IoT agriculture system based on LoRaWAN,” 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), 2018, pp. 1-4, doi: 10.1109/WFCS.2018.8402368.

[8] LoRa Allience, What is LoRaWAN Specification, https://lora-alliance.org/about-lorawan/, [Online; accessed 24-May-2022]

[9] Rad, Babak Bashari, Harrison John Bhatti, and Mohammad Ahmadi. ”An introduction to docker and analysis of its performance.” International Journal of Computer Science and Network Security (IJCSNS) 17.3 (2017): 228.APA

[10] Potdar, Amit M., et al. ”Performance evaluation of docker container and virtual machine.” Procedia Computer Science 171 (2020): 1419-1428.APA

[11] Kubernetes, Kubernetes, https://kubernetes.io/, [Online; accessed 20-May-2022]