IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 14, 2021, accepted July 28, 2021, date of publication August 5, 2021, date of current version August 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3102595

Enhanced Service Framework Based on Microservice Management and Client Support Provider for Efficient User Experiment in Edge Computing Environment

RONGXU XU[1], WENQUAN JIN[2], AND DOHYEUN KIM[1]
[1]Department of Computer Engineering, Jeju National University, Jeju 63243, South Korea
[2]Big Data Research Center, Jeju National University, Jeju 63243, South Korea

Corresponding author: Dohyeun Kim (kimdh@jejunu.ac.kr)

This work was supported by the 2021 Scientific Promotion Program funded by Jeju National University.

ABSTRACT Leveraging the edge computing paradigm, computing resources are deployed in the network edge to provide heterogeneous services. Edge computing delivers sensing and actuating services to the Internet from the constrained Internet of Things (IoT) devices. Meanwhile, management of various elements is provided by offloading sufficient computing and storage to the edge of the networks for the IoT environments such as home, factory, and private spaces without cloud servers. In this paper, we propose an enhanced service framework based on microservice management and client support provider for efficient user experiments in the edge computing environment. For providing the edge computing service and management in the network edge, this paper presents an edge-computing architecture that provides various functions through microservice modules on the edge platform engine. Through the microservices, the interfaces are provided to the client to access the device, data, and additional services. Using Docker, the microservice modules are deployed in the edge platform to provide the services. However, the services and management functions need to be presented to the clients based on the friendly user interfaces. For providing the user interfaces of the services and Docker engine to the clients, the client support service provider is developed and deployed in the network edge based on the edge platform. Therefore, the proposed edge platform provides the services and management to the users for accessing the resources and functions through visualized interfaces in the IoT environment based on edge computing. The performance of our proposed system can be checked through the test result screen and delay time. Compared to controlling edge computing by using a command-line tool for users, we made it easy for general users who are not computer savvy to access edge services through a graphic user interface. And by measuring the delay time and comparing the execution time, it can be seen that the proposed system operates faster.

INDEX TERMS Container, edge computing, EdgeX, Internet of Things, microservice management.

I. INTRODUCTION

The recent global IoT market currently has a potential value due to non-face-to-face or un-contact social and economic activities. The increased devices are used for sensing the environment to present the physical world to us which enables most people moves the work and entertainment to the living area. For example, home electronic devices are being armed with network access and sensors to provide autonomous abilities to their users [1]. In particular, attention is focused on edge computing technology and intelligent services that change the existing vertical IoT service structure to a horizontally distributed ecosystem [2]. Developing edge computing technology to distribute and process data generated from various IoT devices is important in IoT environments [3] and [4]. However, cloud computing can be used for applications that require high availability, computing process, and rich storage space [5]. IoT Cloud integration facilitates in terms of provisioning large-scale data storage and feasible network scaling [6]. Nevertheless, a massive amount of data is generated

The associate editor coordinating the review of this manuscript and approving it for publication was Zhipeng Cai.

by the increased number of IoT devices which derives high pressure to cloud computing to reduce the latency in the interactions between the cloud and IoT environments.

Deploying the computing ability to the IoT environment for sensing, storage, analysis, and decision making, that enables less latency for real-time applications and services [7]–[9]. For the real-time applications in the IoT environment, less delay is most important to maintain the quality of service with the IoT devices [10]. To improve the capabilities of IoT devices, offloading tasks to the nearby node equipped with efficient computing power and storage resources is a solution [11]. Edge computing enables sufficient computing and storage to the constrained IoT environment to provide various functions such as web services and management. Instead of the cloud, deploying the service and management providers to the network edge reduces the network latency, and also prevents data to be exposed for private spaces [12]–[14]. In the entry of the IoT network, the edge platform can be deployed to provide services and management close to the environments where the IoT devices are deployed to provide services of sensing and actuating with limited resources. The edge platform performs as a gateway to provides interfaces to IoT devices and cloud servers in the entry [15]–[17]. Therefore, various integrated services and solutions are enabled to be provided in the edge of networks based on the edge platform for proxy, intelligence, autonomous and management.

The management of IoT devices provides good interoperability for the visibility and accessibility of IoT services from the Internet [18]. For presenting the heterogeneity of the IoT devices to the client using the consistent interface, the device management represents and integrates the physical resources through cyber information [19]. The edge platform can be deployed in the entry of the IoT network to provide management for registering, discovering, and monitoring devices and environment, also bridging heterogeneous device protocols [20]–[23]. For providing a unified data format and consistent interface, standard frameworks can be adopted to develop the edge platform. The EdgeX framework is a lightweight edge computing implementation that is adopted to develop the proposed edge platform to provide various functions based on the microservices [24]. The microservices provide Representational State Transfer (REST) Application Programming Interface (API) REST APIs that are exposed by the microservice server modules that are deployed in the edge platform based on Docker. Docker runs the microservice server modules without considering the underlying operating system [25]. Therefore, using the microservices and docker to integrate the service providers and management to the network edge enables comprehensive IoT services to be performed close to the environment.

In this paper, we propose edge support service and management based on the Docker engine that provides management of microservice server modules. The microservices are developed for providing services in edge computing that enables the edge support service and management to be provided in the network edge based on the EdgeX framework. The EdgeX-based edge platform is deployed in the entry of the IoT network where the sensing and actuating functions are provided through the IoT devices with the sensors and actuators. The edge platform brings the computing ability to the IoT environment with constrained resources. The proposed edge platform includes a device, data, and additional service management based on the EdgeX framework. For providing the User Interface (UI) to access the microservice of the edge platform, the client support service provider is included to provide the UIs of the microservice modules. The microservice modules are deployed in the edge platform through the Docker that orchestrates the microservices to provide various functions at the edge of the network. Based on the Docker engine, the microservices are installed, started, and stopped in the edge computing environment. For providing the client console of the Docker engine based on UIs, the client support service provider also provides interfaces to access the Docker engine. Furthermore, the proposed edge computing architecture enables the users to access the services and management through visualized interfaces based on the edge platform in the network edge for interacting with IoT devices. Our proposed system made the following contributions:

1) Reduced installation and execution time of edge computing services.
2) We reduce the barriers to exploring IoT services for ordinary users who are not familiar with computers by providing a graphic user interface, based on microservice management and client support provider services.
3) we improve the efficiency of user experiments through visualized interfaces based on the edge platform.

The rest of this paper is organized as follows. Section II introduces the related works for existing solutions of edge computing regarding the management functions. Section III introduces the proposed edge computing architecture with the Docker-based microservice management and functions of the edge platform including the microservice modules, client support service provider, and docker engine. Section IV presents edge computing service and management sequences for the scenarios in the network edge with the proposed edge computing architecture. Section V presents the implementation details of the edge computing service and management. Section VI evaluates the performance of the edge platform based on the Docker engine. Finally, we conclude this paper and introduce our future directions in Section VII.

II. RELATED WORK

Microservice-based web application development provides advantages in maintenance and functional extension through the characteristics of flexibility, lightweight, and loose coupling in development and deployment [26]–[28]. For developing constrained applications in the IoT networks, a set of small services can be provided through microservices that run on a standalone process [29]. EdgeX foundry proposed a standard IoT edge computing framework to services based on the microservices in the edge of networks using constrained

devices such as Raspberry Pi [30]. The EdgeX framework supports the connections to heterogeneous device protocols and provides various management functions for devices, data, and edge computing environments. Using the framework, the edge gateway includes several microservice server modules to enable the functions to be scale up and down.

For providing management in the network edge, multiple functions are required such as configurations of application, network, and system, monitoring, and interfaces [31]. The edge platform is deployed in the entry of the network to interact with IoT devices and cloud servers. The management functions can be provided through the microservices based on the edge platform in the network edge. Various standards provide management in the IoT and edge computing environment. The Open Mobile Alliance (OMA) provides device management and security workflow for the IoT devices based on the APIs of device configuration, connectivity monitoring/statistics, security, firmware update, and server provisioning [32]. The LwM2M is the implementation of OMA that is developed for constrained IoT devices using a construed protocol such as CoAP [33]. The LwM2M can be used for the edge gateway with a constrained machine to be deployed in the network edge. The oneM2M is a scalable and interoperable IoT standard that is too heavy to deploy in the network edge [34]. The Open Connectivity Foundation (OCF) standard proposed the IoTivity that is an implementation of the OCF core specification [35], [36]. The main protocol is the CoAP that is used for implementing the role of server and client. The edge platform can be the OCF server to receive the data from the IoT device.

Many standard frameworks are available for implementing IoT edge computing and deploying various industrial domains. Many initiatives proposed solutions that include management, transmission data formats, interaction interfaces, communication protocols. The European Telecommunications Standards Institute (ETSI) proposed Mobile edge computing (MEC) for providing the data and cloud computing service at the edge of networks through offloading the computation to the mobile environment [3]. The Industrial Internet Consortium (IIC) proposed a reference architecture that considers interactions with the enterprise layer from edge computing to process heavy computing tasks [37]. The Openfog architecture proposed three layers including communication, service, and application security to provide proxy and services to cloud servers as well as IoT devices [38]. The cloudlet is proposed for deploying a small-scale cloud data center to the edge of networks [39]. The cloudlet-based architectures bring powerful computing machines to the environment where the data is generated. However, most edge computing requires constrained environments with limited resources and spaces. Therefore, deploying high-performance computing machines to the network edge is difficult.

Docker provides a development and running environment for applications without considering the underlying system. Through containers, Docker deploys applications for distributing the computation and storage. Alam *et al.* [40] proposed a modular and scalable edge computing architecture based on lightweight virtualization using Docker and microservices. Al-Rakhami *et al.* [41] proposed an edge computing framework based on Docker to enable the performing of data processing close to the environment. Yuzhou Huang *et al.* [42] proposed intelligent edge computing through training the model in the cloud and offloading it to the edge based on the Docker that enables the prediction model to be operated in the edge platform. Smet *et al.* [43] proposed a mechanism to deploy specific functionalities to the layers in edge computing using Docker. Ha *et al.* [44] proposed a mechanism of deploying the web services to the edge platform based on Docker for managing service in the smart factory.

III. PROPOSED SERVICE FRAMEWORK WITH MICROSERVICE MANAGEMENT AND CLIENT SUPPORT PROVIDER BASED ON EDGE COMPUTING PLATFORM

A. CONCEPTUAL ARCHITECTURE OF PROPOSED EDGE COMPUTING SERVICE FRAMEWORK

To enhance the efficiency of user experiments in edge computing we propose an architecture of edge computing service framework based on microservice management provider and client support provider. Figure 1, presents the overall conceptual architecture of the proposed solution. The edge computing service framework includes a microservice management module and client support provider module to provide web user interface-based monitoring and management for edge computing services.

The client support provider consists of service monitoring, service management, and service repository modules. The service monitoring module provides default information of edge computing services. The service management enables a client to use services with a simplified user interface to enhance the user experience. The service repository includes overall information of services managed by this proposed solution such as name, image, dependencies, etc.

The microservice management provider composes platform monitoring, service operation, and platform repository components. The platform repository stores the information of the platform that serves as an executing environment for edge computing services. With the service operation, a component client could control the executions of services of edge computing through the docker engine. Platform monitoring component allows a client to get information on the running environment of edge computing services. The microservice management provider releases the user from the swamp of command-line scripts to transparency the management of the execution of edge computing services.

The edge computing platform is the environment for running edge computing services with container-based applications. The services comprised in each container, communicate with each other via REST-APIs to provides functionalities to users. The docker engine manages docker

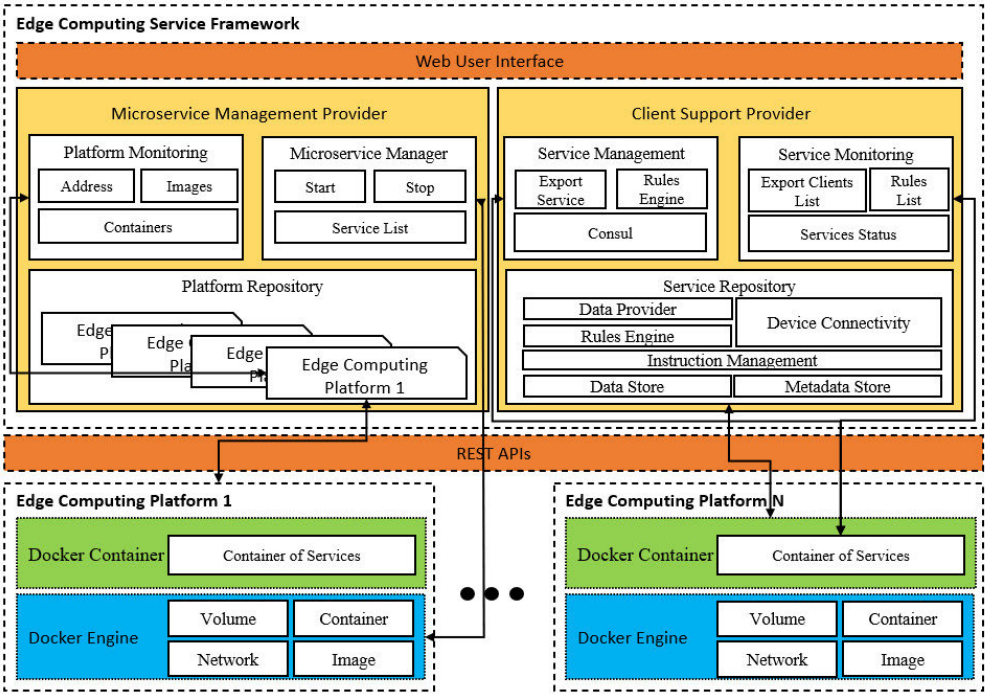

FIGURE 1. Proposed edge computing service framework based on microservice management and client support provider.

objects, such as images, containers, networks, and volumes to support containerized applications running properly.

B. FUNCTIONALITIES OF OUR PROPOSED EDGE COMPUTING ARCHITECTURE

The proposed edge platform provides device, data, consul, rules engine, microservice, and additional service management functionalities based on the EdgeX framework. Figure 2, depicts the overall functionalities of our proposed edge computing architecture. The client support services consist of several support services to provide the User Interface (UI) to the clients that are used to access the microservice of the edge platform, the client support services are implemented to provide the UIs of the microservice modules. The microservice modules of the EdgeX framework are deployed and executed in the edge platform through the Docker engine. The Docker engine use image of the application to build a containerized service in the platform. It also generates related volumes and networks to connect the services in the platform. Based on the Docker engine, the microservices of the EdgeX framework are containerized as a container, and started and stopped in the edge computing environment. For providing the management functionalities of the Docker engine to clients based on UIs, the microservice client support service provides interfaces to access the Docker engine. Furthermore, through the developed edge computing architecture the users enable to access the services and manage them.

Figure 3, shows a functional architecture of microservice management for presenting platforms and services information through interacting with the Docker engine. The Docker engine provides services to manage container-related

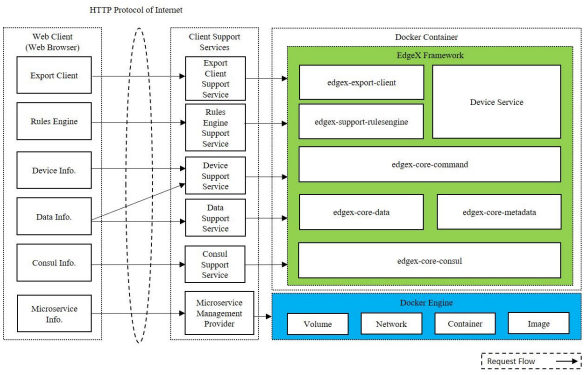

FIGURE 2. The microservice and additional service management functionalities of edge computing service framework.

functionalities. The container of each service is generated by the Docker engine based on the image. The image contains the container's filesystem, it must contain everything needed to run an application - all dependencies, configuration, scripts, binaries, etc. The Docker engine also provides volumes and networks to manage container-related data and connectivity them. With the web client, users can retrieve information about managed platforms and services. The platform list represents the physical devices that serve edge computing services. Through the web page, users can get default information of the device such as the number of the images, number of the containers, number of running containers, and the address of the device. With the service list, users can retrieve information on managed services of each platform. The service list represents the microservices of edge computing. Through the web page, users can get default information

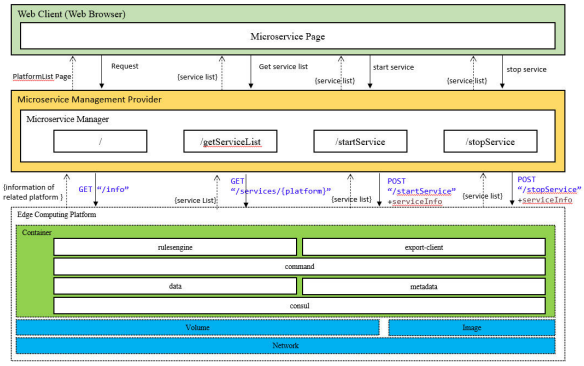

FIGURE 3. Microservice management functional architecture of edge computing platform based on docker containers.

of the service such as service name, status, container name, and start/stop specific service on the web page.

Figure 4 shows a functional architecture of consul management for presenting service information, node information, and Key/Value information through interacting with the core Consul of the EdgeX framework.

(a), the web client requests service /consul to get the page and accesses /getServiceList to get the service list. Once a service item is selected, the web client accesses service /getService to get the detailed information of the service. For providing the information of Consul, services /catalog/datacenters, /internal/ui/services, and /health/service are used.

(b), the web client requests service /consul node to get the page and accesses /getNodeList to get the node list. Once a node item is selected, the web client presents detailed information about the node. The node detail information is included in the node list retrieved by accessing service /getNodeList. The service /catalog/datacenters provides a list of data centers that are used in the Consul Node Page. The service /internal/ui/nodes provides a list of nodes that are used for presenting the registered nodes in Consul.

(c), the registered keys and values in Consul are provided by the resource /kv. With the keys and values, the Consul Kv Page presents list keys, and once the last level of a key is clicked, then the web client presents the value of the key. Web client requests service /consulkv to get the page and accesses /getKvList to get the key list. Once a key item is selected, the web client presents sub key items of the selected key. If a selected key has value, then the value will be presented.

Figure 5 shows a functional architecture of export client management for retrieving registered export clients in the export service index page and detail information of a selected export client, and creating new export client information to the export client in the EdgeX framework. Web client requests service /export to get the export service index page with an export client list. Through the delete function of each item in the export client list, the web client requests the service /exportdelete to delete a selected export client. The export client is a server for providing micro-services in the EdgeX framework. For the export service index page,

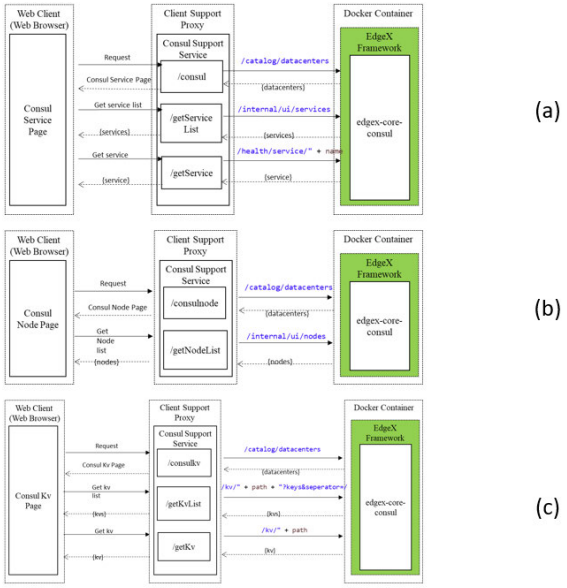

FIGURE 4. Consul management functional architecture – (a) Service page, (b) Node page, (c) Key/Value page.

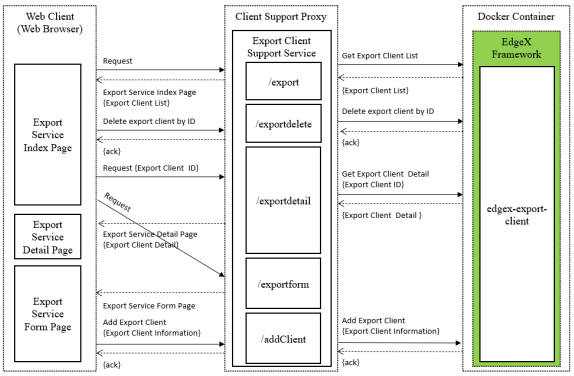

FIGURE 5. Export client management functional architecture.

services /export and /exportdelete are used and access the export client.

To retrieve detailed information of a selected export client, the web client requests service /exportdetail to get the export service's detail page with export client detailed information. The export client detailed information is retrieved by the selected export client ID that is included in the export client list on the export service index page. The service /exportdetail accesses the export client to get the detailed information of the selected client by ID and returns the information with the user interface.

In order to create a new export client information to the export client, the web client requests service /exportform to get the export service form page and presents it to the user. The user inputs the information of required fields and submits it to the service /addClient for creating new export client information. Using the submitted form data, the service /addClient generates JSON format data for the new export client and sends it to the export client of the EdgeX framework.

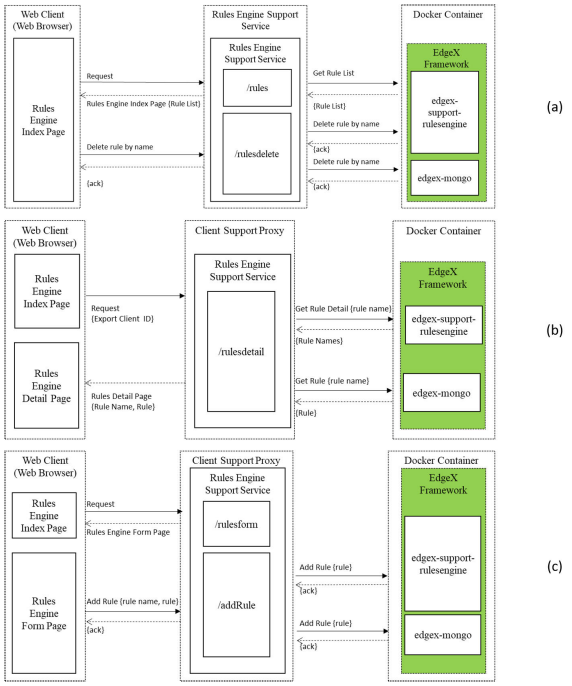

FIGURE 6. Rules engine management functional architecture.

Figure 6 shows a functional architecture of rules engine management for retrieving registered rules in the rules engine index page and detail information of a selected rule, creating new rule information to the rules engine and Mongo DB in the EdgeX framework.

(a), Rules Engine is a server for providing micro-services in the EdgeX framework. For the rules engine index page, services /rules and /rulesdelete are used to access the rules engine. Web client requests service /rules to get the rules engine index page with rules list. Through the delete function of each item in the rule list, the web client requests the service /rulesdelete to delete a selected rule.

(b), web client requests service /rulesdetail to get the rule detail page with rule detail information. The rule detail information is retrieved by the selected rule ID that is included in the rule list on the rules engine index page. The rules engine in the EdgeX framework only saves the rule name of a created rule. Therefore, the Mongo DB is used for saving the rule profile.

(c), web client requests service /rulesform to get the rules engine form page and presents it to the user. The user inputs the information of required fields and submits it to the service /addRule for creating new export client information. The service /addRule sends the rule profile to the rules engine and Mongo DB in the EdgeX framework.

IV. PROPOSED SERVICE FRAMEWORK BASED ON EDGE COMPUTING PLATFORM

EdgeX Foundry that is introduced by Linux Foundation and Dell, adopting microservice-based architecture to achieve the edge computing paradigm. In EdgeX Foundry, all services are generally developed as a dockerized lightweight container which is isolating services and providing simplified maintainability and scalability for the EdgeX Foundry framework. To simplify the deployment and operation of services in the platform, the specifications of the services are explained in the configuration file. The proposed microservice support service in the startup read the configuration file to preparing the operating environment. In order to shorten the time on initialization, we implement each initializer separately including environment initializer, network initializer, volume initializer, and service initializer. Each initializer operates asynchronously to save related configurations in the repository and create related instances in the edge computing platform, exclude the environment initializer it only saves the configurations in the repository. Before creating related instances, the initializer should inspect the instances, if there are instances just quit, else POST requests to the Docker engine to create related instances such as networks, volumes, containers. Figure 7, illustrates a detailed procedure.

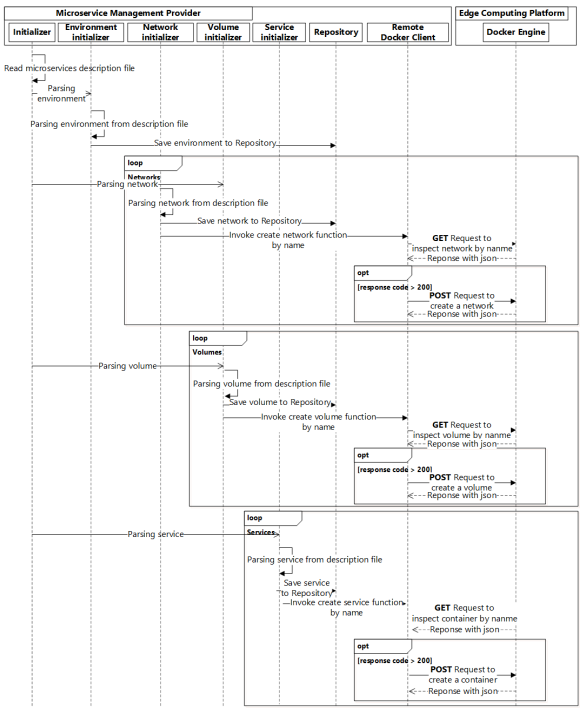

FIGURE 7. Sequence diagram of microservice management provider initialization.

The EdgeX framework is a microservice-based architecture, which means the services are modeled as isolated units that manage a reduced set of issues. However, to deal with problems the fully functional systems rely on the cooperation and integration of its parts which is the dependency between the different parts of the application, and the EdgeX framework is not an exception. When a client request to start a service of EdgeX framework, the controller will retrieve information of platform by id which is a parameter of the request. To start service the controller calls start service

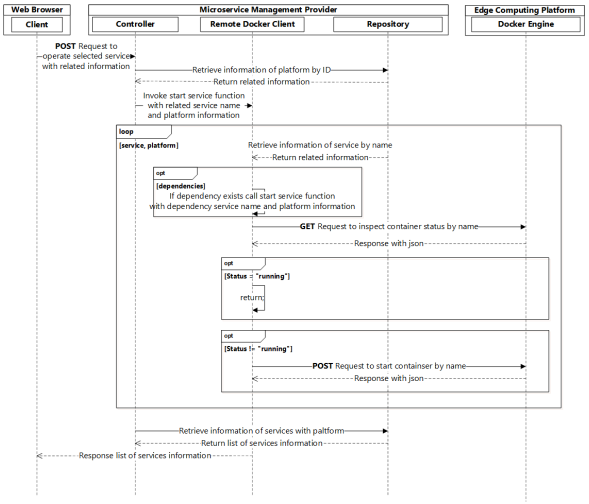

FIGURE 8. Sequence diagram of service operation.

function with a related service name and retrieved platform information as parameters of this function. To make sure the dependent services of the selected service are running, the function is implemented as a recursive function. First, the function retrieves information of service by name from a repository, then check if the dependencies are existing. If true the function will call itself again with dependency service name and platform information. In seconds, the function requests the Docker engine to inspect the service status, if the service is running just quit, otherwise request the docker engine to start service by name. Finally, the controller invokes the repository to get current information of services then return it to a client. Detailed progress is depicted in Figure 8.

V. IMPLEMENTATION AND TEST RESULTS

The Implemented edge platform provides device, data, consul, rules engine, microservice, and additional service management functionalities based on the EdgeX framework. Figure 8, depicts the overall architecture of our Implemented edge computing architecture. The client support services consist of several support services to provide the User Interface (UI) to the clients that are used to access the microservice of the edge platform, the client support services are implemented to provide the UIs of the microservice modules. The microservice modules of the EdgeX framework are deployed and executed in the edge platform through the Docker engine. The Docker engine use image of the application to build a containerized service in the platform. It also generates related volumes and networks to connect the services in the platform. Based on the Docker engine, the microservices of the EdgeX framework are containerized as a container, and started and stopped in the edge computing environment. For providing the management functionalities of the Docker engine to clients based on UIs, the microservice client support service provides interfaces to access the Docker engine. Furthermore, through the developed edge computing architecture the users enable to access the services and manage them.

1) DEVELOP SPECIFICATIONS

To implement our proposed solution, the windows 10 operating system with 500 GB hard disk, 64GB memory, and Intel® core i5-8500 CPU desktop is exploited as the development environment. In order to simplify the development of the web application, we are used the Spring boot framework and Eclipse development tool. Table 1 shows the detailed specification of the development environment. The microservices of EdgeX framework running on the raspberry pi 3 model B is a single-board computer with ubuntu 20.04 operating system, Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB memory, and 16GB MicroSD card. In the executing environment, the docker engine is installed to serve our developed solution. Detailed information refers to Table 2.

TABLE 1. Development environment specification for client support services.

Category	Specification	Description
Desktop	OS	Windows 10
	CPU	Intel® core™ i5-8500
	Memory	64GB
	Hard Disk	500GB
Library	JAVA	An open-source programming language
Application	Eclipse	A Integrate Development Tool
Framework	Spring boot	A comprehensive programming and configuration model for Java-based applications.

TABLE 2. Operating environment for microservices of EdgeX framework.

Category	Specification	Description
Raspberry Pi 3	OS	Ubuntu 20.04 64bit
	CPU	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
	Memory	1GB
	MicroSD Card	16GB
Application	Docker Docker Compose	A platform enables developers to develop, ship, and run applications.

2) TEST RESULTS

Figure 9 shows the implementation result of consul management for presenting service information through interacting with the Core Consul of EdgeX framework. The services are presented as a list that provides clickable items to display the details of a selected service. In the detailed information of a selected service, the items link to the nodes. For each item in the service list, the service name and passing count are presented. For the detailed information of service, tags and nodes of the service are presented.

Figure 10 shows the implementation result of consul management for presenting node information through interacting with the core consul of the EdgeX framework. On this page, the nodes are presented as a list that includes nodes as items to be clicked. Once a node item is clicked, then the detailed information of the clicked node is presented. In the presented node information, the items link to the service page to present detailed information about the service.

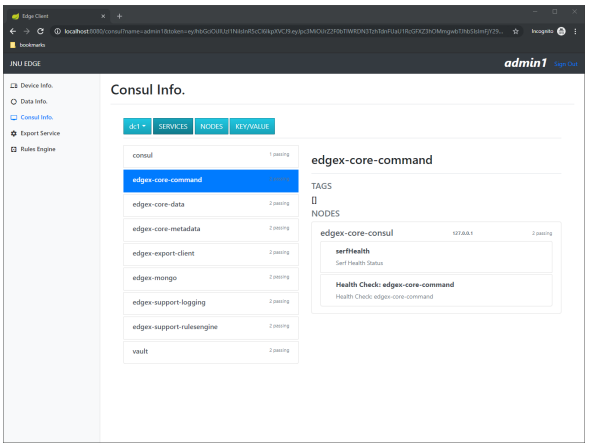

FIGURE 9. Development result of service page for consul management in edge computing services.

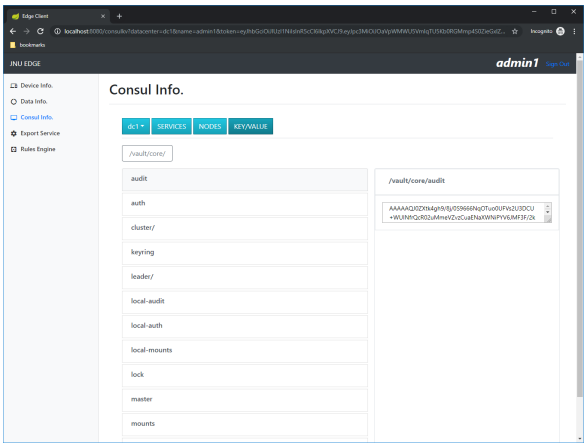

FIGURE 11. Development result of key/value page for consul management in edge computing services.

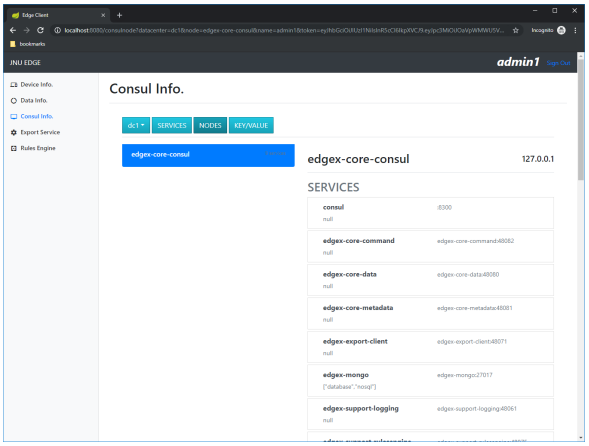

FIGURE 10. Development result of node page for consul management in edge computing services.

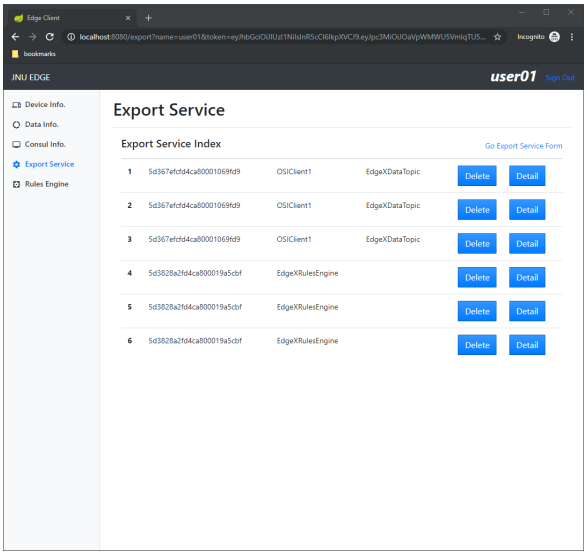

FIGURE 12. Development result of export client list page for export client in edge computing services.

Figure 11 shows the implementation results of consul management for presenting Key/Value information through interacting with the core consul of the EdgeX framework. In this user interface, keys are presented as a list. The items of the key list are used for into next level of selected key items. The top button is used to go to the previous level of keys, and the last level of keys is clicked to present the value of a key. For example, the first item does not have a child-level key, therefore, the symbol / is not attached at last. Once the item is clicked, then the value of the key is presented on the left side.

Figure 12 shows an implementation result of the export client list to present export client id, name, and topic in each item. Each item of the list includes a delete button and detail button to support deleting selected export client and linking to a detail page for the selected export client respectively. The presented export client list is delivered from the client support proxy using JSON format data. For creating a new export client, the link Go Export Service Form redirects to a form page.

Figure 13 shows an implementation result of export client detail for presenting export client's detailed information to the user using a user interface of the web client. The user interface displays the export client's detailed page that includes export client detail information including export client's ID, name, and information of addressable, filter, and encryption. The information of a selected export client is retrieved from the export client of the ExgeX framework by the ID of the export client. The delivered data from the EdgeX is JSON format data that is parsed and well printed by the web client.

Figure 14 shows an implementation result of a form page for creating a new export client using a user interface of a web client. The user interface displays the form page that includes fields for filling properties of the export client that is used for creating a new export client profile. Once the fields are filled, commit the Submit button to send the export client information to the EdgeX framework. The submitted values of properties are sent to the client support proxy and included in JSON format data. Then client support proxy sends the JSON data to the export client of the EdgeX framework.

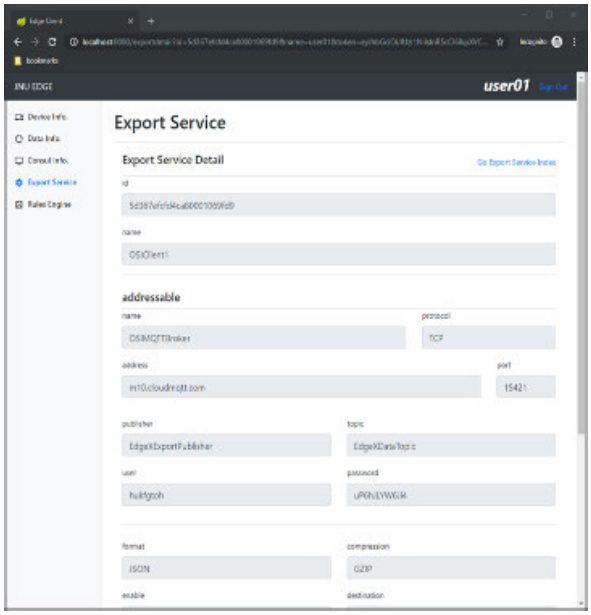

FIGURE 13. Development result of export client list page for export client in edge computing services.

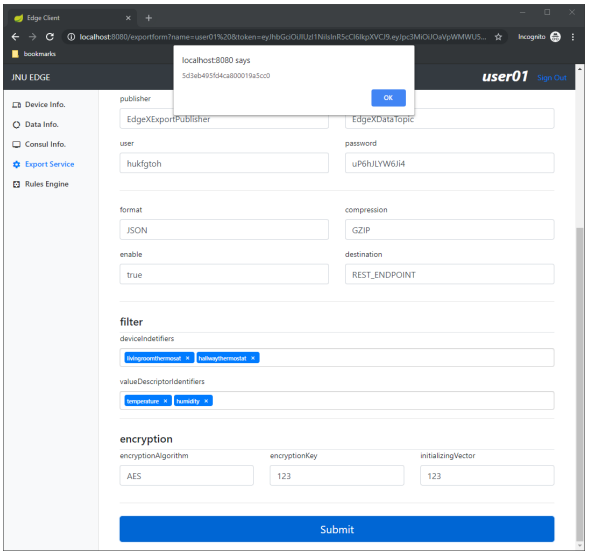

FIGURE 14. Development result of creating new export client for export client in edge computing services.

Figure 15 shows an implementation result of the rule list to present the rule name in each item. For presenting rule information in this list, the data is delivered from the rules engine of the EdgeX framework. The delivered data only includes the uploaded rule's name. Therefore, only the names are presented in the rules list. Each item of the list includes a delete button and detail button to support deleting the selected rule and linking to the detail page for the selected rule. For creating a new rule, the link ''Go Rules Engine'' form redirects to a form page.

Figure 16 shows an implementation result of rule details for presenting a rule's detailed information to the user using a user interface of the web client. The user interface displays

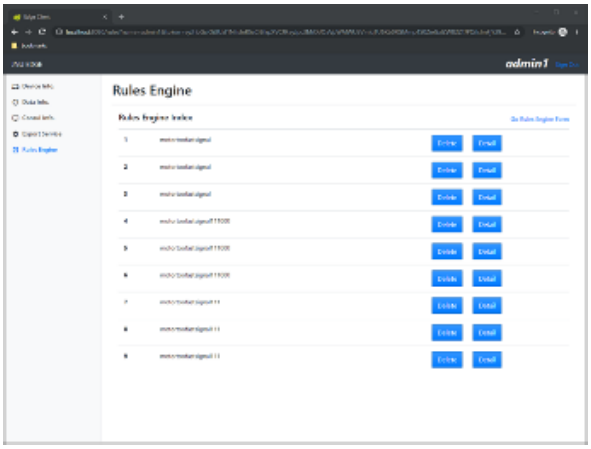

FIGURE 15. Development result of rule list page for rules engine in edge computing services.

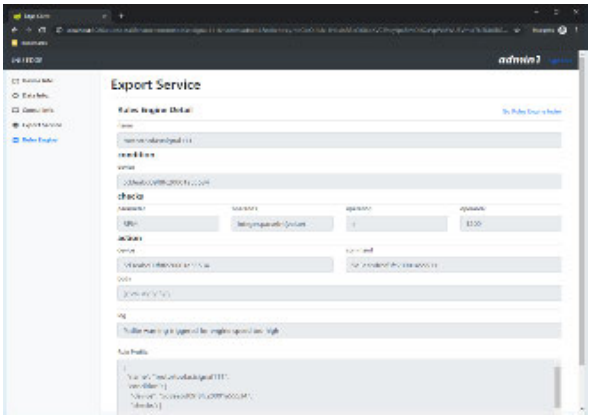

FIGURE 16. Development result of rule detail page for rules engine in edge computing services.

the rule's detail page that includes the rule's detailed information including the rule's name and other properties for the rule profile that is used for creating the Drool profile in the EdgeX framework. The information of a rule profile is JSON format data that is delivered from Mongo DB of EdgeX framework and presented in the fields by the web client. For each property of the rule profile, the values are filled in the field as well as the whole profile is also presented.

Figure 17 shows an implementation result of a form page for creating new rules using a user interface of web clients. The user interface displays the form page that includes fields for filling properties of rules that are used for creating a new rule profile. Once the fields are filled, commit the submit button to send the rule to the EdgeX framework. In this form, the drop boxes for the device, in the condition and action areas, show a device list through requesting metadata.

Figure 18 shows the developed result of the platform list and service list. (a), presents default information of the device such as the number of the images, number of the containers, number of running containers, and the address of the device. With the view of the platform list, the user would get rough information of the platform that is managed by the users.

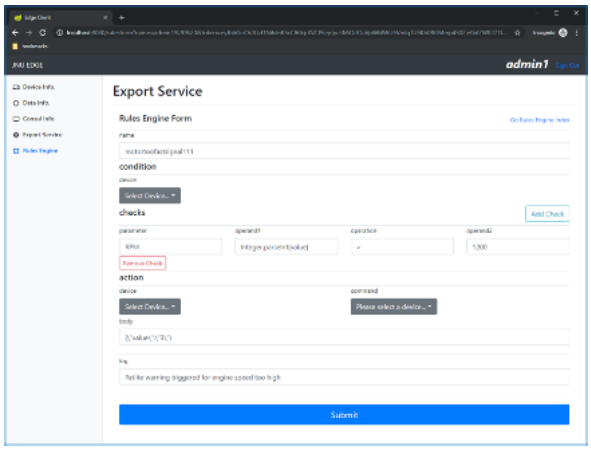

FIGURE 17. Development result of creating new rule for rules engine in edge computing services.

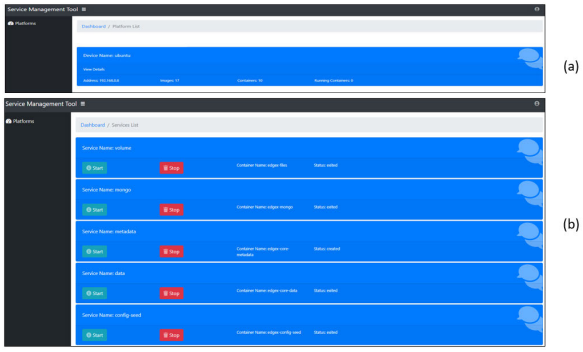

FIGURE 18. Development result of microservice management provider. (a) Platform list, (b) Service list.

When you click the view details button, the result of the service list shows up.

(b), exposes the service list that is provided by the EdgeX framework. With the view of the service list, the client enables to know the status of the service. Each service has the ability to start and stop service by the buttons. Figure 18, shows the result of the start service called ''data''. With one click the dependent services are started properly.

VI. PERFORMANCE EVALUATION AND ANALYSIS

To evaluate the performance of the proposed system, we measure the execution time of services that start with the proposed system and command-line script. As you can see from Figure 19, the time it takes to start the service when using the proposed system is faster than that using the command line script. Especially, when metadata is executed, the startup with the proposed system is faster than the command-line script about 24 seconds. Among them, the smallest difference is about 16 seconds when logging starts. In addition, it takes less than 10 seconds to run services such as volume, consul, and mongo.

Figure 20, measures the time it takes to stop the service. When using the command line script, the maximum time is about 48 seconds and the minimum time is about 16 seconds.

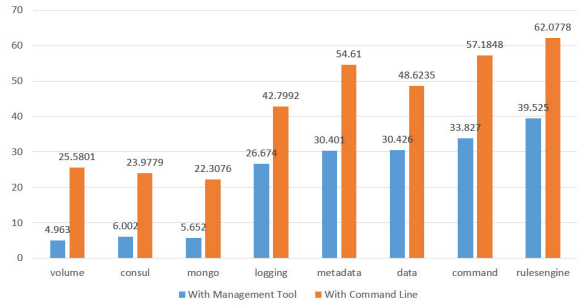

FIGURE 19. Comparison of the latency taken to start services for edge computing services.

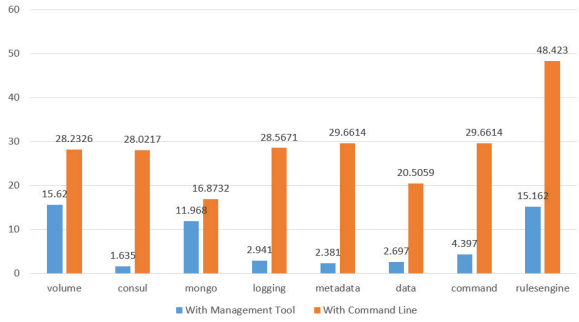

FIGURE 20. Comparison of the latency taken to stop services for edge computing services.

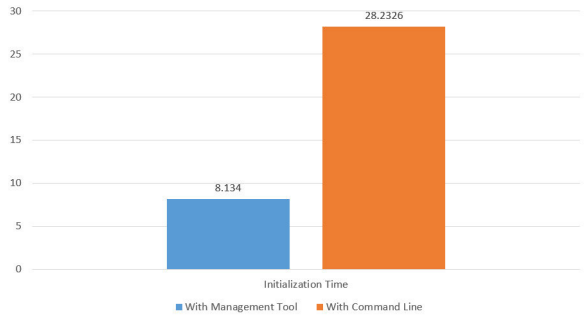

FIGURE 21. Comparison of the latency taken to initialize services for edge computing services.

In contrast, stopping the service with the proposed system takes less than 5 seconds except for the volume, mongo, and rules engine that takes more than 10 seconds.

Figure 21, measures the time used in the initialization phase before starting the service. All services go through the process of being dockerized before being controlled by the docker engine, and the time used to do this is as shown in the figure. The command-line script takes about 30 seconds, while the proposed system takes about 9 seconds.

VII. CONCLUSION AND FUTURE DIRECTIONS

For providing the edge computing service and management in the network edge with IoT devices, this paper proposed the development of an edge platform to provide various functions through microservice modules based on the Docker engine.

The microservices enables the clients to access device, data, and additional services through the REST APIs. The Docker engine deploys microservice modules in the edge platform that is comprised of multiple microservice modules to provide various IoT and edge computing services. The management edge computing is developed based on the EdgeX framework, and deployed in the entry of the IoT network to deliver the sensing data to the clients and forward the control commands to the actuators. With the REST APIs of the EdgeX-based microservice modules, the client support service provider is developed and deployed on the edge platform to provide the user interfaces of the services and Docker engine. Therefore, the proposed edge platform integrates various functions to the edge computing device using microservice modules for providing the services in the network edge such as private spaces with constrained resources. Moreover, the visualized interfaces are provided to the users for accessing the IoT environment and managing the edge computing elements based on the EdgeX and Docker. When we saw the test results of our proposed system, instead of asking users to control edge computing using a command-line tool, we made it possible for general users who are not computer savvy to easily access edge services through a graphic user interface. As can be seen from the latency comparison results, it can be seen that our proposed system operates faster.

In the future, we will develop intelligent approaches based on deep learning algorithms to provide intelligent and autonomous services at the edge of the networks. For offloading the intelligent approaches to the network edge, the intelligent service module can be developed based on the microservice and deployed by the Docker engine. However, offloading multiple intelligent services to edge computing can be a challenge that requires optimized resource allocation. In addition, for a more complete performance evaluation, we would perform tests on specific IoT cases, such as access some Industrial IoT applications or Machine Communications devices in robotic surgery through our proposed system in the future.

REFERENCES

[1] M. S. Khan, M. F. Abrar, D. Kim, F. Tila, I. A. Khan, J. Shuja, and A. N. Khan, "Resource-based direct manipulation: A user-centric visual interface for operational customization of future smart appliances," *Telecommun. Syst.*, vol. 75, no. 3, pp. 291–306, Nov. 2020.

[2] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2016, pp. 20–26.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[4] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[5] V. Gezer, J. Um, and M. Ruskowski, "An extensible edge computing architecture: Definition, requirements and enablers," in *Proc. UBICOMM*, 2017, pp. 1–5.

[6] R. M. A. Haseeb-ur-rehman, M. Liaqat, A. H. M. Aman, S. H. A. Hamid, R. L. Ali, J. Shuja, and M. K. Khan, "Sensor cloud frameworks: State-of-the-art, taxonomy, and research issues," *IEEE Sensors J.*, early access, Jun. 21, 2021, doi: 10.1109/JSEN.2021.3090967.

[7] W. Jin, R. Xu, S. Lim, D.-H. Park, C. Park, and D. Kim, "Dynamic inference approach based on rules engine in intelligent edge computing for building environment control," *Sensors*, vol. 21, no. 2, p. 630, Jan. 2021.

[8] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.

[9] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *J. Syst. Archit.*, vol. 98, pp. 289–330, Sep. 2019.

[10] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the Internet of Things," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 17–24.

[11] S. K. U. Zaman, A. I. Jehangiri, T. Maqsood, Z. Ahmad, A. I. Umar, J. Shuja, E. Alanazi, and W. Alasmary, "Mobility-aware computational offloading in mobile edge networks: A survey," *Cluster Comput.*, pp. 1–22, Apr. 2021.

[12] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2495–2508, Sep. 2018.

[13] W. Jin, R. Xu, T. You, Y.-G. Hong, and D. Kim, "Secure edge computing management based on independent microservices providers for gateway-centric IoT networks," *IEEE Access*, vol. 8, pp. 187975–187990, 2020.

[14] S. Naveen and M. R. Kounte, "Key technologies and challenges in IoT edge computing," in *Proc. 3rd Int. Conf. I-SMAC (IoT Social, Mobile, Analytics Cloud) (I-SMAC)*, Dec. 2019, pp. 61–65.

[15] R. Morabito, R. Petrolo, V. Loscri, and N. Mitton, "LEGIoT: A lightweight edge gateway for the Internet of Things," *Future Gener. Comput. Syst.*, vol. 81, pp. 1–15, Apr. 2018.

[16] C.-H. Chen, M.-Y. Lin, and C.-C. Liu, "Edge computing gateway of the industrial Internet of Things using multiple collaborative microcontrollers," *IEEE Netw.*, vol. 32, no. 1, pp. 24–32, Jan./Feb. 2018.

[17] R. Morabito, R. Petrolo, V. Loscrí, and N. Mitton, "Enabling a lightweight edge gateway-as-a-service for the Internet of Things," in *Proc. 7th Int. Conf. Netw. Future (NOF)*, Nov. 2016, pp. 1–5.

[18] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung, "Recent advances in industrial wireless sensor networks toward efficient management in IoT," *IEEE Access*, vol. 3, pp. 622–637, 2015.

[19] J. D. C. Silva, J. J. P. C. Rodrigues, J. Al-Muhtadi, R. A. L. Rabêlo, and V. Furtado, "Management platforms and protocols for Internet of Things: A survey," *Sensors*, vol. 19, no. 3, p. 676, 2019.

[20] W. Jin and D.-H. Kim, "IoT device management architecture based on proxy," in *Proc. 6th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Oct. 2017, pp. 84–87.

[21] W. Jin and D. Kim, "Resource management based on OCF for device self-registration and status detection in IoT networks," *Electronics*, vol. 8, no. 3, p. 311, 2019.

[22] W. Jin and D. Kim, "Development of virtual resource based IoT proxy for bridging heterogeneous web services in IoT networks," *Sensors*, vol. 18, no. 6, p. 1721, 2018.

[23] W. Jin and D. Kim, "Improved resource directory based on DNS name self-registration for device transparent access in heterogeneous IoT networks," *IEEE Access*, vol. 7, pp. 112859–112869, 2019.

[24] *EdgeX Foundry*. (2021). [Online]. Available: https://www.edgexfoundry.org/

[25] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2015, pp. 171–172.

[26] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Berlin, Germany: Springer, 2017, pp. 195–216.

[27] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2018, pp. 29–2909.

[28] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. Sebastopol, CA, USA: O'Reilly Media, 2015.

[29] C. Santana, B. Alencar, and C. Prazeres, "Microservices: A mapping study for Internet of Things solutions," in *Proc. IEEE 17th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2018, pp. 1–4.

[30] F. Liu, G. Tang, Y. Li, Z. Cai, X. Zhang, and T. Zhou, "A survey on edge computing systems and tools," *Proc. IEEE*, vol. 107, no. 8, pp. 1537–1562, Aug. 2019.

[31] A. Musaddiq, Y. Bin Zikria, O. Hahm, H. Yu, A. K. Bashir, and S. W. Kim, "A survey on resource management in IoT operating systems," *IEEE Access*, vol. 6, pp. 8459–8482, 2018.
[32] O. M. Alliance, "Lightweight machine to machine technical specification," Approved Version, Open Mobile Alliance, San Diego, CA, USA, Tech. Rep. OMA-TS-LightweightM2M_Core-V1_1-20180710-A, 2017, vol. 1, no. 1.
[33] S. Rao, D. Chendanda, C. Deshpande, and V. Lakkundi, "Implementing LWM2M in constrained IoT devices," in *Proc. IEEE Conf. Wireless Sensors (ICWiSe)*, Aug. 2015, pp. 52–57.
[34] H. Park, H. Kim, H. Joo, and J. Song, "Recent advancements in the Internet-of-Things related standards: A oneM2M perspective," *ICT Exp.*, vol. 2, no. 3, pp. 126–129, Sep. 2016.
[35] Open Connectivity Foundation. (2021). *OCF*. [Online]. Available: https://openconnectivity.org/
[36] *IoTivity*. (2021). [Online]. Available: https://iotivity.org
[37] M. Tseng, T. Canaran, and L. Canaran, "Introduction to edge computing in IIoT," Ind. Internet Consortium, Needham, MA, USA, Tech. Rep., 2018, pp. 1–19.
[38] *OpenFog Reference Architecture for Fog Computing OPFRA001.020817*, OpenFog Consortium Architecture Working Group, Fremont, CA, USA, 2017, p. 162.
[39] V. Prokhorenko and M. A. Babar, "Architectural resilience in cloud, fog and edge systems: A survey," *IEEE Access*, vol. 8, pp. 28078–28095, 2020.
[40] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using Docker and edge computing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 118–123, Sep. 2018.
[41] M. Al-Rakhami, M. Alsahli, M. M. Hassan, A. Alamri, A. Guerrieri, and G. Fortino, "Cost efficient edge intelligence framework using Docker containers," in *Proc. IEEE 16th Int. Conf Dependable, Autonomic Secure Comput., 16th Int. Conf Pervasive Intell. Comput., 4th Int. Conf Big Data Intell. Comput. Cyber Sci. Technol. Cong. (DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 800–807.
[42] Y. Huang, K. Cai, R. Zong, and Y. Mao, "Design and implementation of an edge computing platform architecture using Docker and Kubernetes for machine learning," in *Proc. 3rd Int. Conf. High Perform. Compilation, Comput. Commun.*, Mar. 2019, pp. 29–32.
[43] P. Smet, B. Dhoedt, and P. Simoens, "Docker layer placement for on-demand provisioning of services on edge clouds," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 3, pp. 1161–1174, Sep. 2018.
[44] J. Ha, J. Kim, H. Park, J. Lee, H. Jo, H. Kim, and J. Jang, "A web-based service deployment method to edge devices in smart factory exploiting Docker," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2017, pp. 708–710.