Article

EdgeX over Kubernetes: Enabling Container Orchestration in EdgeX

Seunghwan Lee ¹, Linh-An Phan ¹ , Dae-Heon Park ² , Sehan Kim ² and Taehong Kim ^{1,*} ¹ School of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, Korea; seunghwanlee@cbnu.ac.kr (S.L.); linhan@cbnu.ac.kr (L.-A.P.)² Electronics and Telecommunications Research Institute, Daejeon 34129, Korea; dhpark82@etri.re.kr (D.-H.P.); shkim72@etri.re.kr (S.K.)

* Correspondence: taehongkim@cbnu.ac.kr

Abstract: With the exponential growth of the Internet of Things (IoT), edge computing is in the limelight for its ability to quickly and efficiently process numerous data generated by IoT devices. EdgeX Foundry is a representative open-source-based IoT gateway platform, providing various IoT protocol services and interoperability between them. However, due to the absence of container orchestration technology, such as automated deployment and dynamic resource management for application services, EdgeX Foundry has fundamental limitations of a potential edge computing platform. In this paper, we propose EdgeX over Kubernetes, which enables remote service deployment and autoscaling to application services by running EdgeX Foundry over Kubernetes, which is a product-grade container orchestration tool. Experimental evaluation results prove that the proposed platform increases manageability through the remote deployment of application services and improves the throughput of the system and service quality with real-time monitoring and autoscaling.

Keywords: EdgeX; edge computing; container orchestration; Kubernetes; remote deployment; autoscaling; Internet of Things

Citation: Lee, S.; Phan, L.-A.; Park, D.-H.; Kim, S.; Kim, T. EdgeX over Kubernetes: Enabling Container Orchestration in EdgeX. *Appl. Sci.* **2022**, *12*, 140. <https://doi.org/10.3390/app12010140>

Academic Editor: Agostino Forestiero

Received: 16 November 2021

Accepted: 21 December 2021

Published: 23 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the exponential growth of the Internet of Things (IoT), the number of connected devices is constantly increasing as various application services such as augmented reality (AR), virtual reality (VR), smart cars, smart cities, and healthcare, have developed in recent years. As a result, there is increasing demand for quickly processing the huge amount of data generated from these IoT devices, especially for time-sensitive applications such as autonomous vehicles, AR/VR gaming, and smart factories. Recently, edge computing has emerged as a means of overcoming the limitations of cloud computing by placing computational resources (i.e., edge servers) close to end devices to meet the stringent requirements on latency that these applications have. In the edge computing architecture, application services can be deployed in edge servers to directly process data and provide immediate responses. As a result, edge computing allows reduced network bandwidth and latency and improved Quality of Service (QoS) in IoT applications [1].

The IoT gateway is a representative example of edge servers in IoT systems. As well as collecting and aggregating data from IoT devices, an IoT gateway processes data with its computational resources and can connect to remote cloud servers to perform further processing and analytics if necessary. Moreover, an IoT gateway has the ability to control IoT devices and provide application services to users. Among various open-source-based edge computing platforms, such as CORD [2], EdgeX Foundry [3], Apache Edgent [4], Azure IoT Edge [5], Akraio Edge Stack [6], OpenEdge, and KubeEdge, Apache Edgent and EdgeX Foundry were developed to target deploying IoT applications. Both support various protocols—such as HTTP, ModBus, and MQTT—to provide connectivity with IoT devices in IoT application environments (e.g., smart homes, environmental monitoring, and remote

IoT management). However, Apache Edgent is not an appropriate platform for modern IoT environments owing to its monolithic characteristic of using Java virtual machine, which affords less scalability and changeability than a container. In contrast, EdgeX Foundry focuses on communicating with various protocols and facilitates interoperability between devices. Moreover, EdgeX Foundry offers a container-based application service; thus, it offers flexibility in the addition or removal of application services without interrupting other existing application services [7].

Although EdgeX Foundry has many advantages, it lacks container orchestration abilities, such as the dynamic deployment of microservices and dynamic resource management, leading to the following challenges in building and operating edge computing infrastructure with multiple edge nodes.

- EdgeX Foundry does not support remote deployment and management of services across multiple edge nodes because it adopts Docker Compose [8] to deploy and manage container-based application services. Docker Compose was developed considering service deployment on a single host. Thus, EdgeX Foundry offers low application services manageability in the edge computing infrastructure.
- EdgeX Foundry is unable to perform dynamic adjustments to computational re-sources based on the demand (workload) of the service. In edge computing infrastructure, the demand for the application service can vary depending on the user distribution and number of user requests. Therefore, allocating appropriate resources to each service based on its real-time demand allows improvements to QoS and efficient utilization of hardware resources of edge nodes.

Kubernetes is a product-grade container orchestration tool that provides useful functions for use in an edge computing environment, such as automated deployment, resource management, load balancing, and autoscaling [9]. For example, users can deploy application services remotely without any deep knowledge of the internal system and manual configuration. Moreover, an autoscaling function allocates more resources to an application service when it exceeds the available resource threshold. Therefore, the autoscaling function provides adaptiveness of application resources according to the dynamic demands on the application services.

To address the limitations of EdgeX Foundry, we propose *EdgeX over Kubernetes* (EoK) in this study. EoK combines EdgeX Foundry and the representative container orchestration platform Kubernetes. EoK supports remote deployment of multiple services to edge nodes in a short duration. Moreover, by utilizing the Horizontal Pod Autoscaler (HPA) feature of Kubernetes, EoK can dynamically increase the number of replicas for each service based on its real-time demand. The objective of EoK is to enable container orchestration on EdgeX Foundry to facilitate the use of EdgeX Foundry in practical edge computing infrastructure. The main contributions of this study are summarized as follows.

- We propose EoK as a practical implementation of EdgeX Foundry to improve manageability through the remote deployment of services. Moreover, EoK provides resource autoscaling capabilities, which are crucial in edge computing systems. The implementation of EoK is described in detail to ensure the reproducibility of this work.
- We performed experimental evaluations to demonstrate the feasibility and advantages of the EoK system. The results obtained demonstrate that services of EdgeX Foundry can be easily deployed from the Kubernetes master node. Moreover, the HPA feature in Kubernetes can result in improved throughput and latency of services in the EoK system compared with the original EdgeX Foundry platform.

The remainder of this paper is organized as follows. Section 2. surveys related work. Section 3 gives preliminary explanations of Kubernetes and EdgeX Foundry. In Section 4, we describe the implementation of EdgeX over Kubernetes. Performance evaluations are presented in Section 5. Finally, we conclude the paper in Section 6.

2. Related Work

In this section, we discuss existing research related to building an IoT edge computing infrastructure. In the traditional IoT environment, IoT gateways play a key role in building the IoT infrastructure; Hao Chen [10] defined IoT gateways by dividing them into three layers: sensing, network, and application layers. Shang Guoqiang et al. [11] and Lin Wu et al. [12] highlighted the lack of compatibility between heterogeneous protocols as the fundamental problem—i.e., different types of gateways are required for each protocol—and suggested IoT gateway structures to solve this problem. The authors of [11] proposed a smart IoT gateway that can customize user cards according to the protocol environment. Meanwhile, the authors of [12] proposed a plug-configuration-play service-oriented gateway structure for fast and easy connection with various protocols.

Moreover, with the development of container technologies, such as Docker, IoT gateways have evolved to distribute and operate container-based microservices. According to [13,14], microservices are better than monolithic architecture in aspects such as scalability, changeability, and upgrades. For example, microservice architecture can facilitate simple system migration because the demanded microservice only needs to be replicated at a new location. Furthermore, microservices can be changed or upgraded easily when a problem occurs or an update is needed. These microservice characteristics can accelerate the spread of IoT gateway-based services. EdgeX Foundry [3] is a promising microservice-based IoT edge gateway platform that supports various IoT connectivity protocols and provides interoperability between heterogeneous devices. EdgeX Foundry offers multiple strategic advantages for gateways, as follows. First, it provides reference implementations for IoT protocols and software development kits so that users can add new ones. Second, it provides flexible connectivity to various enterprise and IoT environments. Third, EdgeX Foundry communicates with devices that use legacy protocols and can translate data for comprehension by modern devices. Finally, EdgeX Foundry offers a microservice-based platform that enables plug-and-play of components.

To increase service quality, the authors of [15–17] proposed methods to improve IoT gateway service quality through machine learning. In [15], the authors proposed an automatic temperature adjustment method to improve user convenience by applying an optimization engine and fuzzy control to the IoT gateway. The authors of [16] placed an intelligent deep-learning-based function in the rule's engine of the IoT gateway to process user requests quickly. In [17], the authors proposed a way to improve service quality by containerizing and distributing the deep learning model and running it at edge nodes. They also further improved the service quality by distributing the computational load on IoT gateways to the cloud servers. The authors of [18] proposed an intelligent service management technique that can process large amounts of data generated by many devices in real-time while solving various problems, such as connectivity and security, in an industrial IoT environment. However, these approaches focus on optimization at a single IoT gateway; therefore, it is difficult to apply them to edge computing infrastructures composed of multiple IoT gateways.

Nonetheless, the authors of [19,20] addressed the challenges involved when the gateways are geographically dispersed on a large scale. In [19], the authors proposed a hierarchical structure, wherein the upper layer supervises the workload at data centers in the lower layer and triggers the migration of applications when necessary. However, this approach evaluated the gateway based on a virtual machine and could face limitations in term of scalability and resource utilization compared with microservice architectures. In [20], the authors observed that service composition is a critical issue for efficient utilization of the available services that are widely dispersed across the Internet of Service paradigm and proposed a framework that deploys services close to frequently-used-together IoT devices through service migration.

In summary, existing approaches have not considered performing service deployment and resource management in edge computing environments containing multiple IoT gateways. In this paper, based on the idea that EdgeX Foundry is a promising platform for

building an IoT edge computing infrastructure, we propose EoK, which utilizes EdgeX Foundry as is but solves its limitations, such as inconvenient service deployment and absence of dynamic resource management, using Kubernetes. Specifically, the proposed method can provide dynamic service deployment and resource management in an IoT gateway environment by enabling EdgeX Foundry to run on Kubernetes, a representative container orchestration platform. Moreover, the proposed method is verified experimentally.

3. Preliminary Research

3.1. EdgeX Foundry

EdgeX Foundry is an open-source-based IoT edge computing platform located between cloud and IoT devices (device/sensor) to interact with IoT devices [3] and play the role of edge nodes [21] in an edge computing environment. In other words, EdgeX Foundry can collect, store, analyze, and convert data from IoT devices and transfer the data to a designated endpoint. Moreover, the user can directly control the operation of IoT devices and monitor the device data through EdgeX Foundry.

EdgeX Foundry contains four main layers: *Device Services*, *Core Services*, *Supporting Services*, and *Export Services*. The *Device Services* layer interacts with IoT devices and abstracts their connectivity protocols. Microservices in EdgeX Foundry can request data collected from IoT devices or transmit the data to other microservices, such as the *Core Services* layer. The *Core Services* layer is an intermediary between the upper layers and *Device Services* layer. The *Core Services* layer includes initial information and sensor data about IoT devices connected to edge nodes, which are stored in a local database until they are sent to the upper layers and cloud systems. The *Core Services* layer manages information on microservices, such as hostname and port number, and provides a connection between microservices. When one microservice wants to connect to another microservice, it calls the *Core Services* layer to obtain the microservice information necessary to connect the microservice. The *Supporting Services* layer performs common software functionalities, such as scheduling, notifications, and alerts; it allows each microservice to execute a designated task at a specified time or under given rules. In addition, the *Supporting Services* layer collects logs inside EdgeX Foundry, records them to log files or databases, and notifies or alerts external systems or users connected to EdgeX Foundry. The *Application services* layer contains a set of functions that process messages in a specific order based on the function pipeline concept. Furthermore, the *Applications Services* layer prepares (e.g., transforms, filters, etc.), formats (e.g., reformats, compresses, encrypts), and exports data to an external service designated as an endpoint.

3.2. Kubernetes

Kubernetes is an open-source container orchestration platform that manages and deploys container-based applications [22]. A Kubernetes cluster includes master nodes and at least one worker node.

A pod is the smallest deployable unit that can be created and managed by Kubernetes. A pod can contain one or more containers that share storage and networks, it has a specification of how each container runs, and each pod is assigned a unique internal IP address. Moreover, one specification can create several pods, referred to as replicas [23], to provide scalability and availability for applications.

Note that a pod is ephemeral, and it can be recreated or terminated at any time. Further, when a pod restarts, its IP address changes. To allow access to pods, Kubernetes defines *Service* of three kinds: *ClusterIP*, *NodePort*, and *LoadBalancer* [22]. A *Service* abstracts multiple pods as a group and exposes the group to the network. By default, a group is assigned an invariable IP address by *ClusterIP*. *ClusterIP* Service works as a frontend for grouped pods, and grouped pods work as a backend. When the frontend (i.e., *ClusterIP* Service) receives traffic, it redirects requests to the backend (i.e., grouped pods) following Kubernetes' rules. However, *ClusterIP* is used only for connection between cluster components, so it cannot

receive the traffic from outside. For communicating outside, the *NodePort* and *LoadBalancer* Services are used. The *NodePort* Service enables access from outside of the cluster by reserving and exposing port numbers on external IP addresses of each worker node; then, traffic requests can access to backend pods using an external IP address and reserved port. The *LoadBalancer* Service is provided by a cloud vendor such as Google Cloud Platform, Azure [5], and Amazon Web Services. The cloud vendor provides an external IP address and URL to the *LoadBalancer* Service for access from outside.

In Kubernetes, the master node oversees the management and control of the Kubernetes cluster and consists of *etcd*, *kube-scheduler*, *kube-controller-manager*, and *kube-apiserver*. The *etcd* is backend storage that stores all cluster data in a key-value structure. The *kube-scheduler* is in charge of assigning a newly created pod to a node according to scheduler rules, such as resources optimization, policy constraints, and node-affinity. The *kube-controller-manager* ensures that the current node or pod operates in the desired state. For example, when using autoscaling, if resource usage is found to be above the threshold of the pod specification through periodic monitoring, the *kube-controller-manager* determines whether to deploy a replica or scale up the pod resources according to the specifications. The *kube-apiserver* processes all requests from the cluster and interacts with the worker node through *kubelet*.

Kubernetes provides three kinds of autoscalers to provide optimal resource management on demand: Cloud Autoscaler (CA), Vertical Pod Autoscaler (VPA), and Horizontal Pod Autoscaler (HPA) [23]. The CA works with cloud services to adjust the number of nodes. If a node has insufficient resources to create a pod, the CA adds a node and creates a pod on the newly added node. The VPA configures a pod with more resources when pod resources are insufficient, but the pod must restart to apply the configuration. The HPA increases the number of pods to scale resources. Specifically, the HPA stores the resource metric of a pod in the metric server and scales by increasing the number of pods when the metric exceeds a specified threshold. Therefore, multiple replicas can provide one application service. When a user request reaches a specific application service, Kubernetes forwards the request to a pod according to the *kube-proxy* load balancing policy, such as round robin or random approach. The Kubernetes HPA can adjust pod resources dynamically according to resource status without interruption of existing application services. For this reason, the HPA is an essential technology for IoT edge computing infrastructure, where user requests can vary over location and time.

4. EdgeX over Kubernetes

In this section, we present EoK to overcome the fundamental limitations of EdgeX Foundry when building an IoT edge computing infrastructure. First, the problems of EdgeX Foundry can be defined as follows:

(1) EdgeX Foundry offers poor application service manageability. As EdgeX Foundry uses Docker Compose to deploy and manage microservices on a single node, it does not provide the functionality to remotely deploy and manage microservices to edge nodes. In other words, as Figure 1a shows, each service of EdgeX Foundry is deployed and operated locally in the form of a microservice using Docker Compose. As Docker Compose is installed on a single host, an administrator must enter commands directly at the edge node when they want to update or deploy new services to edge nodes. Accordingly, when the service requested by the user is not deployed at the corresponding edge node, it is impossible to deploy and use the service in real-time.

Figure 1. Schematic illustrations of the concepts of (a) EdgeX Foundry and (b) EdgeX over Kubernetes.

(2) EdgeX Foundry does not allocate adequate computational resources based on resource status. In edge computing infrastructure, the demands on the application service can vary depending on the user distribution and their requests. For example, if the allocated computational resources are insufficient to process the requests, the QoS, such as in terms of response time and throughput, must be lowered. Therefore, allocating appropriate resources to services based on real-time demands and current resource status presents a cost-efficient approach toward improving QoS. However, EdgeX Foundry lacks features such as real-time resource monitoring and dynamic resource management for application services; therefore, the service cannot be scaled up even when resource usage by the Virtual Pod (device service) increases, as shown in Figure 1a.

To address the problems concerning low manageability and service quality discussed above, we propose EoK. EoK is based on EdgeX Foundry; however, it can increase manageability by efficiently managing edge nodes and services with help from Kubernetes container orchestration. As shown in Figure 1b, EdgeX Foundry's edge nodes become worker nodes of a Kubernetes cluster. In this cluster, EdgeX Foundry's edge nodes can deploy and manage microservices remotely through Kubernetes, not Docker Compose, providing increased manageability. Moreover, service resource management and service quality can be improved by enabling autoscaling of microservices through the HPA—one of the features of Kubernetes. For example, if the resource usage of the Virtual Pod is above a given threshold, EoK can adapt its resource usage by dynamically deploying replicas of the Virtual Pod, as shown in Figure 1b. To summarize, the proposed EoK provides functions such as remote service deployment and autoscaling with the help of Kubernetes to improve the manageability and QoS of EdgeX-based IoT edge gateways.

In the following subsection, we describe in detail how Kubernetes functions can be used to design the EoK system. Then, we describe two common-use cases that benefit from the EoK system.

4.1. Implementation of EdgeX over Kubernetes

The implementation of EoK is discussed in detail below.

First, microservices in EdgeX Foundry use the Docker Compose YAML file structure, and its specification structure is different from that of Kubernetes, although both use the same YAML extension. For example, in Docker Compose YAML, a service can be classified

according to whether the *volumes* or *depends_on* field is used. The *volumes* field configures storage, so it can specify a key (volume name)–value (actual path) pair in the Docker Compose YAML structure. The *depends_on* field expresses dependencies between services, i.e., it is a start/stop order between services. For example, if service Y and service Z declare the *depends_on* field with service A, services Y and Z must start before service A starts; service A cannot start before services Y and Z. Moreover, service A will terminate before services Y and Z.

It is important to note that the YAML structures of Docker Compose and Kubernetes have incompatible fields; Docker Compose fields must be converted to Kubernetes fields to make EdgeX Foundry work similarly on Kubernetes in EoK. The *volumes* field in Docker Compose can be easily replaced with the *Volume* field of Kubernetes as they work the same. However, because Kubernetes does not define the *depends_on* field, we add *readinessProbe* and *livenessProbe* to the Kubernetes YAML file. The *readinessProbe* and *livenessProbe* fields are container diagnostic tools that *kubelet* periodically executes. The *readinessProbe* investigates whether a pod is ready to process the request or not. If the probe's diagnosis returns success, i.e., if the pod is prepared to process the requests, Kubernetes allocates an IP to the pod and enables communication. However, if the pod is not ready, the pod IP is removed from the endpoints of all other services. The *livenessProbe* checks the pod's operating status; if the pod does not operate properly, the *livenessProbe* handles the pod according to the *RestartPolicy* (*Always*, *OnFailure*, *Never*) written in the YAML file (i.e., *PodSpec*). To this end, the *livenessProbe* restarts a pod after a certain period if the pod starts before necessary pods start and the pod fails to start, and the *readinessProbe* checks if the pod can handle requests. If so, the pod successfully starts, and Kubernetes allocates an IP to the pod. Therefore, the settings of the two fields (*readinessProbe* and *livenessProbe*) operate similarly to those of the *depends_on* field and make all EdgeX-Foundry-related pods start smoothly on Kubernetes.

Moreover, EdgeX node must set the Kubernetes *Service* on several microservices. For example, the *Command* pod in the *Core Services* layer handles user requests from outside the cluster. In this case, the *Command* pod should be a *NodePort* Service for communicating with the outside. Then, users can send requests to EdgeX nodes. Other pods, such as pods in *Supporting Services* layer, can be set as *ClusterIP* Service as they communicate with other pods in the same cluster.

Figure 2 shows the proposed EoK architecture, denoting the worker node that has deployed EdgeX Foundry as an EdgeX node. The EdgeX Foundry microservices are deployed successfully in the EdgeX node through Kubernetes. It is interesting to note that not only is EdgeX Foundry deployed, but *kube-proxy* and *kubelet* are also deployed successfully in an EdgeX node. EdgeX node acts as a worker node in the Kubernetes cluster by combining *kubelet* and *kube-proxy* components. EdgeX node maintains and manages the microservices by reporting the operational status to the master node through *kubelet*. In addition, it can communicate with pods in other EdgeX nodes in the same cluster through *kube-proxy*.

Through *etcd*, we can check the data of an EdgeX node in a cluster, and *kube-scheduler* assigns new device services or device services that must be updated to the desired EdgeX node. The *kube-controller-manager* ensures that the specified number of pod replicas operate in the cluster. Moreover, an EdgeX node can create, execute, and manage pods from the master node, and it can create a Kubernetes *Service* through *kube-apiserver*.

In this manner, EoK can dynamically distribute the necessary services to various EdgeX nodes through Kubernetes, and the distributed services can maintain high QoS by periodically managing resource status through *kubelet* and providing autoscaling through the HPA.

Figure 2. EdgeX over Kubernetes architecture.

4.2. Remote Device Service Deployment

In this section, we discuss the process of deploying a new device service to an EdgeX node using the container orchestration of Kubernetes. A new service can be implemented as a service of pod units, as described earlier, and is deployed to the *Device services* layer of EdgeX Foundry.

As shown in Figure 3, the master node commands to deploy (Kind: Deployment) the device service using *kubectl*. The *kube-controller-manager* creates as many pods as specified in the corresponding *PodSpec* and *kube-scheduler* selects an EdgeX node to deploy a pod according to *nodeAffinity*, where *nodeAffinity* defines the priority and preference for selecting the node to deploy the pod.

Figure 3. EdgeX over Kubernetes service deployment procedure.

Once the pod is deployed to the selected EdgeX node, *kubelet* starts the pod and transports the status of all device services in the pod to the API server. The *Core Services*

layer in the EdgeX node uses *Metadata* to check whether a device service is already registered or not. If the newly deployed device service does not exist in the EdgeX node, the *Device Services* layer requests a service registration to *Metadata*, and *Metadata* adds the device service to the database.

In summary, in contrast to the existing method of distributing new services locally in EdgeX Foundry, the proposed EoK platform enables remote distribution of new services to multiple EdgeX nodes, increasing the ease of management of application services.

4.3. HPA

In this section, we explain how the autoscaling of EdgeX services works in EoK. To enable service autoscaling through the HPA feature of Kubernetes, the minimum/maximum number of running pods in the cluster must be set through *minReplicas*, *maxReplicas*, and *threshold* in the deployment file of the service. *cAdvisor* monitors the pod resources aggregated at *Metrics-Server*; *kube-controller-manager* compares the collected metrics with the threshold specified in *PodSpec* and calculates the desired number of pods every 15 s period by default.

When the Central Processing Unit (CPU) resources of the pod exceed the given threshold, *kube-controller-manager* calculates the desired number of pods for the overloaded service and sends them to *kube-apiserver*. After that, *kube-apiserver* deploys the additional number of pod replicas to reach the desired number of pods in the cluster. On the contrary, if the desired number of pods is less than the current number of pods, *kube-apiserver* terminates pod replicas for service downscaling.

For example, in Figure 4, the *Virtual Device* service sets *minReplicas* as 2, *maxReplicas* as 5, and threshold as 50% of the CPU resources. The *cAdvisor* collects each CPU usage and sends it to *Metrics-Server*. *Metrics-Server* averages the CPU usages and sends it to *Metrics Aggregator* in *kube-apiserver*. The HPA of *kube-controller-manager* compares the current CPU usage and threshold, which is 50%, to determine whether the CPU usage exceeds the threshold or not; if the current CPU usage is higher than the threshold, it deploys additional replicas of the *Virtual Device* service through *Deployment*. At this time, these are registered additionally to an EdgeX node by following the service deployment procedure shown in Figure 3.

Figure 4. HPA architecture of EoK.

Therefore, we can conclude that EoK improves the QoS as well as throughput in an edge computing environment by applying the HPA of Kubernetes to dynamically increase the number of replicas in response to an increase in user requests or computational load.

5. Performance Evaluations

This section evaluates the performance of EoK in terms of service deployment time and service autoscaling through the HPA. The experimental evaluation environment was set up as follows. The master node had 4 CPU cores and 8 GB of RAM, and the worker node had 2 CPU cores and 4 GB of RAM. All nodes in the Kubernetes cluster had Kubernetes version 1.17.0 and Docker version 19.03.6 installed and running on the Ubuntu 18.04.4 LTS operating system. We deployed EdgeX Foundry in the worker node, and the services in EoK could be accessed from outside the cluster using the *NodePort* Service of Kubernetes.

We used the *Virtual Device* service, which can receive HTTP requests, to evaluate the throughput of the service. We set the CPU limitation for the HPA as 20 m, minimum number of replicas as 1, and maximum number of replicas as 4. Moreover, the cycle for data scraping of the CPU and number of pods was set to 1 min, which is the same as a HPA cycle in Kubernetes. An Apache HTTP server benchmarking tool (ab) was used for traffic generation, and each experiment was repeated 10 times to ensure the evaluation result accuracy.

5.1. Remote Device Service Deployment

In this section, we evaluate the dynamical deployment the *Device Service* to EdgeX nodes in a Kubernetes cluster and the time required to remotely deploy a service in EoK. The service deployment time was measured from the point at which the deployment command was executed in the Kubernetes controller until the service ran in the EdgeX node. Note that the service image was not downloaded to the EdgeX node in advance in this evaluation, and the service status was checked every second to verify the time of deployment completion.

Figure 5 shows the total deployment time and time of the first to the fourth replicas deployed to the EdgeX node, expressed by a cumulative distribution function, where deployment time includes downloading from the repository and running the pod replica in each node as per the master node's *kubectl* command. As shown in Figure 5, we can observe that 75% of the first replica deployment time was approximately under 20 s to start, regardless of how many replicas had to be deployed. Moreover, the time after the first replica deployment until the remaining replicas were deployed gradually decreased, as shown in Figure 5b,c, because the first replica had to download the image for operation while the subsequent replicas did not. Thus, the total deployment time increased with the number of replicas, but not linearly. Although the total deployment time increases with the number of replicas, this result proves that service deployment through EoK can be completed within several tens of seconds via the deployment command at the master node located remotely. Therefore, we can conclude that EoK improves the manageability by allowing geographically remote deployment of services at edges.

Figure 5. Total time for remote deployment and setup time for (a) 1 replica, (b) 2 replicas, and (c) 4 replicas.

5.2. Performance Change in EoK Using HPA

This section verifies the performance improvement of EoK through the HPA of Kubernetes by evaluating CPU usage, processing time, throughput, and latency according to the number of pod replicas.

Figure 6 shows the changes in the average CPU usage and number of pods during 3000 requests from outside the cluster when the maximum number of replicas was set to 1, 2, and 4. As shown in Figure 6, we can observe that CPU usage increased with increasing traffic requests, and the number of replicas increased up to the maximum as well. For example, the number of replicas in Figure 6a could not further increase because the maximum number of replicas was limited to 1, whereas the number of pods in Figure 6b increased at 120 s, and that in Figure 6c increased at both 230 s and 300 s. According to the increase in replicas, the average CPU usage tended to decrease, as did the completion time, while processing 3000 requests. In Figure 6, we can observe that the processing was completed at 840 s with 1 replica, 720 s with 2 replicas, and 660 s with 4 replicas. On the contrary, EdgeX Foundry does not support the HPA function, so it showed the same performance as when limiting the maximum number of replicas to 1 in Figure 6a. Therefore, we can conclude that it is difficult to improve the throughput with only EdgeX Foundry.

Figure 6. Average CPU usage and number of replicas for EoK with HPA (a) up to 1 replica, (b) up to 2 replicas, and (c) up to 4 replicas.

Figure 7 evaluates the average CPU usage according to the number of pods, throughput, and latency for different numbers of replicas. It is important to note that Figure 6 measures the performance according to the change in the number of replicas over time starting with 1 replica, while Figure 7 evaluates the performance using a fixed number of replicas (1, 2, or 4) in EoK to focus on the effect of the number of replicas as opposed to the original. In these evaluations, we include the evaluation of EdgeX Foundry to compare the difference in the performances of EoK and EdgeX Foundry. However, the evaluation result of the original is expected to resemble that of EoK with one replica, because EoK with one replica is architecturally equivalent to EdgeX Foundry except for the presence of a Kubernetes cluster. As a result, the process completion time in Figure 7a reduced to 840 s for 1 replica, 600 s for 2 replicas, and 480 s for 4 replicas compared with the 840, 720, and 660 s shown in Figure 6a–c, respectively. Moreover, the process completion time of EdgeX Foundry is the same as that of EoK with one replica.

Figure 7b shows the throughput of EdgeX Foundry and EoK according to the number of concurrent requests. First, as expected, when one replica was used in EoK, the throughput observed was the same as that of EdgeX Foundry because both cases have equal resources for the service. Moreover, we can see that the throughput could not be increased due to the resource limit, even as the number of concurrent requests increased. On the contrary, the throughput of EoK with 2 and 4 replicas increased to approximately 17% and 71%, respectively, as the number of concurrent requests increased. More specifically, the throughput with 2 replicas increased from 6 to 7 requests/s as the number of concurrent requests increased from 1 to 5, while that with 4 replicas increased from 7 to 12 requests/s under the same conditions. It is interesting to note that Kubernetes provides load balancing

by forwarding requests to multiple replicas; thus, we can expect the throughput to be improved by increasing the number of pod replicas.

Figure 7. Performance of EoK with HPA: (a) average CPU usage, (b) throughput, and (c) latency of each replica set.

However, in contrast to the increase in throughput according to the number of replicas, the increase in throughput according to the number of concurrent requests varied depending on the maximum number of replicas. This can be seen as a result of the difference between the required number of resources to handle the actual traffic and the maximum available resources. For example, the performance improvements with 3 and 5 concurrent requests for EoK with 2 replicas were insignificant. This indicates that when the number of concurrent requests was 3, all available CPU resources were already used. Therefore, even when we increased the number of concurrent requests to 5, there was no further improvement in throughput. In contrast, when increasing the number of concurrent requests from 1 to 3 for EoK with 4 replicas, we can see that the throughput improved significantly by fully utilizing the CPU resources for the increasing requests.

Figure 7c illustrates the latency of the requests measured, which are shown in Figure 7a,b. The results show that the latency tended to increase as the number of concurrent requests increased. For example, the average latency for EoK with 1 replica for each of 1, 3, and 5 concurrent requests was 250, 730, and 1270 ms, respectively. This is because the more requests are received, the longer the waiting time required for processing. However, we can also see that the waiting time decreased as the number of replicas increased. For example, the average latency for EoK with 2 replicas under 1, 3, and 5 concurrent requests was reduced to 163, 425, and 710 ms, respectively. In particular, the average latency for EoK with 4 replicas under 1, 3, and 5 concurrent requests was significantly reduced to 134, 254, and 414 ms, respectively. However, in this evaluation, it was observed that one replica with EoK yielded a marginally lower delay than that of EdgeX Foundry, and the difference in the delay of the two platforms grew with an increase in the number of concurrent requests. This is because EdgeX Foundry does not have predefined proxy settings; however, by default, Kubernetes uses iptables, which exhibit low system overhead while processing traffic. Moreover, as the autoscaling and load balancing features are absent in EdgeX Foundry, a single application service must handle all requests. Consequently, EoK outperforms EdgeX Foundry considerably. Therefore, this evaluation proves that EoK improves the latency for processing requests with load balancing among multiple replicas in Kubernetes, while EdgeX cannot.

Therefore, we can conclude from our experimental evaluations that EoK can improve the service manageability and resource management through dynamic service deployment and horizontal pod autoscaling while using the existing EdgeX Foundry system as is within the Kubernetes.

6. Conclusions

As a representative open-source-based edge computing platform, EdgeX Foundry has been considered a key enabling technology for building IoT edge computing infrastruc-

ture. However, due to the absence of container orchestration functions, there are several limitations to its use in an IoT edge computing environment, where edge nodes are geographically dispersed. To overcome this problem, we proposed EoK, which runs EdgeX Foundry over a representative container orchestration platform, Kubernetes. Through performance evaluation, we verified that the throughput and QoS can be significantly improved through resource autoscaling, while the manageability of the system is enhanced with the remote service deployment. In future studies, we will investigate the deployment of connectivity services to support various protocols through heterogeneous IoT devices. Moreover, we will study the load balancing mechanism among edge nodes that can address the imbalance of user demands in IoT edge computing infrastructure. We believe the proposed EoK platform contributes to the acceleration of the realization of industrial and dispersed IoT edge computing environments by offering flexibility pertaining to various protocols and in managing nodes and services.

Author Contributions: Conceptualization, S.L., L.-A.P. and T.K.; experiment, S.L. and L.-A.P.; writing—original draft preparation, S.L.; review & editing, L.-A.P. and T.K.; supervision, D.-H.P. and T.K.; funding acquisition, D.-H.P. and S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00387, Development of ICT based Intelligent Smart Welfare Housing System for the Prevention and Control of Livestock Disease).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Ha, K.; Chen, Z.; Hu, W.; Richter, W.; Pillai, P.; Satyanarayanan, M. Towards wearable cognitive assistance. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, Bretton Woods, NH, USA, 16–19 June 2014; pp. 68–81.
- CORD. Available online: <https://opennetworking.org/cord> (accessed on 6 August 2021).
- EdgeX Foundry. Available online: <https://www.edgexfoundry.org> (accessed on 6 August 2021).
- Apache Edgent. Available online: <https://incubator.apache.org/projects/edgent> (accessed on 6 August 2021).
- Microsoft Azure. Available online: <https://azure.microsoft.com> (accessed on 6 August 2021).
- Akraino. Available online: <https://www.lfedge.org/projects/akraino> (accessed on 6 August 2021).
- Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An updated performance comparison of virtual machines and Linux containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 171–172.
- Docker Compose. Available online: <https://docs.docker.com/compose> (accessed on 6 August 2021).
- Nguyen, N.D.; Phan, L.-A.; Park, D.-H.; Kim, S.; Kim, T. ElasticFog: Elastic Resource Provisioning in Container-Based Fog Computing. *IEEE Access* **2020**, *8*, 183879–183890. [CrossRef]
- Chen, H.; Jia, X.; Li, H. A brief introduction to IoT gateway. In Proceedings of the IET International Conference on Communication Technology and Application (ICCTA 2011), Beijing, China, 14–16 October 2011; pp. 610–613.
- Guoqiang, S.; Yanming, C.; Chao, Z.; Yanxu, Z. Design and Implementation of a Smart IoT gateway. In Proceedings of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Beijing, China, 20–23 August 2013; pp. 720–723.
- Wu, L.; Xu, Y.; Xu, C.; Wang, F. Plug-configure-play service-oriented gateway: For fast and easy sensor network application development. In Proceedings of the 2nd the International Conference on Sensor Networks, Barcelona, Spain, 19–21 February 2013; pp. 53–58.
- Butzin, B.; Golatowski, F.; Timmermann, D. Microservices approach for the internet of things. In Proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), Berlin, Germany, 6–9 September 2016; pp. 1–6.
- Krivic, P.; Skocir, P.; Kusek, M.; Jezic, G. Microservices as Agents in IoT Systems. In Proceedings of the 11th KES International Conference, KES-AMSTA 2017, Vilamoura, Algarve, Portugal, 21–23 June 2017; Volume 74, pp. 22–31.

15. Xu, R.; Jin, W.; Hong, Y.; Kim, D.-H. Intelligent Optimization Mechanism Based on an Objective Function for Efficient Home Appliances Control in an Embedded Edge Platform. *Electronics* **2021**, *10*, 1460. [[CrossRef](#)]
16. Jin, W.; Xu, R.; Lim, S.; Park, D.-H.; Park, C.; Kim, D. Dynamic Inference Approach Based on Rules Engine in Intelligent Edge Computing for Building Environment Control. *Sensors* **2021**, *21*, 630. [[CrossRef](#)] [[PubMed](#)]
17. Gupta, N.; Anantharaj, K.; Subramani, K. Containerized Architecture for Edge Computing in Smart Home: A consistent architecture for model deployment. In Proceedings of the 2020 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 22–24 January 2020; pp. 1–8.
18. Choi, H.; Song, J.; Yi, K. Brightics-IoT: Towards Effective Industrial IoT Platforms for Connected Smart Factories. In Proceedings of the 2018 IEEE International Conference on Industrial Internet (ICII), Seattle, WA, USA, 21–23 October 2018; pp. 146–152.
19. Forestiero, A.; Mastroianni, C.; Meo, M.; Papuzzo, G.; Sheikhalishahi, M. Hierarchical approach for green workload management in distributed data centers. In Proceedings of the European Conference on Parallel Processing, Porto, Portugal, 25–29 August 2014; pp. 323–334.
20. Forestiero, A.; Mastroianni, C.; Papuzzo, G.; Spezzano, G. A proximity-based self-organizing framework for service composition and discovery. In Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, VIC, Australia, 17–20 May 2010; pp. 428–437.
21. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [[CrossRef](#)]
22. Kubernetes. Available online: <https://kubernetes.io> (accessed on 6 August 2021).
23. Nguyen, T.-T.; Yeom, Y.-J.; Kim, T.; Park, D.-H.; Kim, S. Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration. *Sensors* **2020**, *20*, 4621. [[CrossRef](#)] [[PubMed](#)]