논문 21-46-09-20 The Journal of Korean Institute of Communications and Information Sciences '21-09 Vol.46 No.09
https://doi.org/10.7840/kics.2021.46.9.1507

Edge 컴퓨팅 환경을 위한 경량 KubeEdge 툴

레 반끄엉*, 유 명 식°

Lightweight KubeEdge Tool for Edge Computing Environments

Van-Cuong Le*, Myungsik Yoo°

요 약

지난 10년간 모바일 기기가 급증하면서 짧은 지연시간 내에 처리해야 하는 데이터가 급증했다. 필요한 위치에 가깝게 설계된 에지 컴퓨팅이 이러한 문제를 해결할 것으로 기대된다. 일반적으로 에지 노드는 클라우드 노드보다 적은 자원을 가지고 있다. 따라서 컨테이너 기술은 시스템에 상당한 오버헤드를 주는 가상화 기술보다 더 경쟁력 있는 솔루션으로 간주된다. 컨테이너 관리 및 조정 툴로 각광받는 Kubernetes는 에지 컴퓨팅을 완전히 지원하지 않는다. KubeEdge는 Kubernetes를 기반으로 에지 컴퓨팅에서 컨테이너 기반 애플리케이션 관리가 가능하도록 확장하였다. KubeEdge 시스템 구축에는 네트워킹 및 운영 체제에 대한 복잡한 지식이 필요하다. 본 논문에서는 KubeEdge 시스템의 구성에 필요한 절차들에 대한 복잡성을 최소화할 수 있는 경량 KubeEdge 툴 제공을 목표로 한다. 구현 및 평가를 통해 제안된 툴이 KubeEdge 시스템을 빠르고 효과적으로 구축할 수 있음을 보여주었다.

Key Words : NFV, Container, Edge Computing, cgroup, KubeEdge

ABSTRACT

Booming mobile devices in the past decade eventually led to massive data that needed to be processed at low latency. We expect edge computing placed closer to the required location will satisfy the challenges. Each edge node typically has resources lower than a cloud node. Hence, containerization is considered a more competitive solution than virtualization, which has a significant overhead on systems. Kubernetes, leading as a containerized management and orchestration tool, do not fully support edge computing. KubeEdge extends native containerized application orchestration capability into edge computing based on Kubernetes. Deploying a KubeEdge system requires complex knowledge about networking and operating system. This study aims to provide a lightweight KubeEdge tool to minimize complexity of configuration of KubeEdge system. By implementing and evaluating, we show that the proposed tool can deploy KubeEdge in a fast and effective way.

※ This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2017-0-01633) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

◆ First Author : Van-Cuong Le, Department of Information Communication Convergence, Soongsil University, Seoul, South Korea, cuonglv@ssu.ac.kr, 학생회원

° Corresponding Author : Myungsik Yoo, School of Electronic Engineering, Soongsil University, Seoul, South Korea, myoo@ssu.ac.kr, 종신회원

논문번호 : 202106-130-D-RN, Received June 10, 2021; Revised June 17, 2021; Accepted June 17, 2021

Ⅰ. Introduction

As the number of mobile devices rapidly increases, there is an increasing tendency for service providers to apply edge computing to adapt to massive data along with a low latency requirement[1] when rolling out their services. Application developers, along with hyper-scale cloud providers (Microsoft Azure, Amazon Web Service, etc.), are betting on Kubernetes at the edge with their open-source projects like KubeEdge[2], Akri[3], and Google Anthos[4]. When companies want to have a consistent approach to managing workloads at the edge with Kubernetes[5], different challenges appear[2-4].

KubeEdge allows operators to deploy and manage their services as containerizations at the edge side through a centralization management interface. To deploy a service in production on the KubeEdge system, it has to be designed and evaluated on a KubeEdge system. Therefore, we need a quick solution for the configuration of a KubeEdge system to accelerate service development. KubeEdge is extended from Kubernetes that is a leading containerized orchestration but still hard to deploy. Besides, KubeEdge is also work in progress[6]. Hence, it requires a complex configuration process along with prerequisite details that are not mentioned in the official document[2]. This paper aims to provide a tool to automatically configure a KubeEdge system for an edge computing environment.

The rest of this paper is organized as follows. Section II describes the background on containerized technologies and KubeEdge. A difficulty when deploying a KubeEdge system and flowchart details of the proposed tool are explained in Section III. Section IV describes the detail of the experiment. Finally, section V concludes the paper.

Ⅱ. Background

2.1 Container Runtime Interface

Container Runtime Interface (CRI) is a plugin interface. It gives kubelet the ability to use a variety of container runtimes. Besides, Kubernetes plugins execute and observe over containers using CRI. CRI includes gRPC API, protocol buffers, and libraries[7].

Open Container Initiative (OCI)[8] runtime is "low-level" runtimes. It focuses on managing the container lifecycle and is not required to do much else. Low-level runtimes create and run "the container".

Containerd is the default CRI of the Docker engine. Because containerd is popular, containerd is an industry-standard. This runtime container uses runc, the reference implementation of the OCI runtime specification[8]. It provides the minimum set of functionality to run containers and manages images and snapshots on a node. It runs and terminates containers by delegating execution tasks to the OCI runtime. Containerd has to the following actions to run a new container:

- Create a new container from a given OCI image;
- Create a new task in the container context;
- Start the task

CRI-O is a container runtime built to provide a link between the high-level Kubernetes CRI and OCI runtimes[9]. It is an alternative solution for containerd. RedHat has developed and maintained it. The typical life cycle of the interaction with CRI-O is almost the same as containerd over the CRI endpoint[10].

Because containerd, which is reliable and popular with Docker, is an industry-standard container runtime, this study uses containerd as a default container runtime.

2.2 Container Orchestration Tools

Containers can be created and managed using container runtime. However, we need tools that manage and orchestrate these containers across a cluster system. They are required because usually a single service comprises many containers working together. Thus, container orchestration includes the management of a life cycle of container.

There are various container orchestration tools available such as Kubernetes, Apache Mesos[11], Docker Swarm[12], etc. They provide different

frameworks for managing and orchestrating containers at scale. Containers are deployed on hosts, usually in replicated groups called pods. Container orchestration tools schedule the deployment. The most appropriate host is chosen to place the container based on predefined constraints such as idle compute unit, available memory, network bandwidth, etc. While the container runs on the host, the container orchestration tool manages the container's life cycle. Container orchestration tools can be used in any environment that can run containers, from personal computers to private cloud instances running on Amazon Web Services (AWS), DigitalOcean, etc. Currently, Kubernetes is one of the leading container orchestration tools.

All the above container orchestration tools do not fully support edge computing. KubeEdge fully supports edge computing, but it needs some effort to make it easier to use.

2.3 Docker

Docker engine is an open-source containerization platform based on Linux containers for building and containerizing the application[13]. It enlarges current container technology by providing lightweight containers and API for managing container images of multi-container applications. Docker can help us to deploy an application regardless of environment settings from one environment to another one. Besides, it can create any network functions in service function chains as a container.

Docker container relies on the kernel of the host running the docker engine[13]. It only separates the user space at the operating system level and consumes resources when running an application. Basically, a container consists of processes isolated from the rest by namespaces. The containers running in Docker share the host OS kernel. It uses Linux Kernel features such as storage, networking, and control groups to build containers on top of an operating system. Docker encloses an application's software into a package with everything it needs to run like OS, application code, Run-time, system tools, libraries, etc. Hence, an application can run on any other Linux machine regardless of any customization configurations. Additionally, it adds a read-write file system over the read-only file system of the Docker image to create a Docker container.

The proposed tool does not install a completed Docker engine. It just installs containerd that is developed by Docker and now by Cloud Native Computing Foundation (CNCF).

2.4 KubeEdge

KubeEdge is an open-source edge computing platform designed to extend Kubernetes containerized orchestration capabilities to hosts at edge[2]. It supports networking application deployment and metadata synchronization between cloud and edge devices. Compared with Kubernetes, orchestration capabilities are separated from edge nodes to the cloudcore. Cloudcore is the extended Kubernetes controller that controls edge nodes and pods' metadata that helps data being sent to identified edge nodes. Meanwhile, workloads are handled by the edge node.

As shown in Figure 1, KubeEdge consists of two parts.

Part1-At the cloud side:

- EdgeController is responsible for connections between kube-apiserver and edge nodes. It has the ability to register the edge node with the kube-apiserver.
- DeviceController manages the device by describing device metadata/status and synchronizing these device updates between edge and cloud.
- Cloud Hub is one part of cloudcore and provides communication ability between EdgeController and Edge nodes. It supports both QUIC protocol and web-socket protocol.

Part 2-At the edge side:

- Edge Hub interacts with the cloud-side for edge computing. It comprises reporting edge-side status to the cloud-side, synchronizing resource updates between them.
- EdgeD or edge node is used to operate Pods on the KubeEdge system. Every node must have a

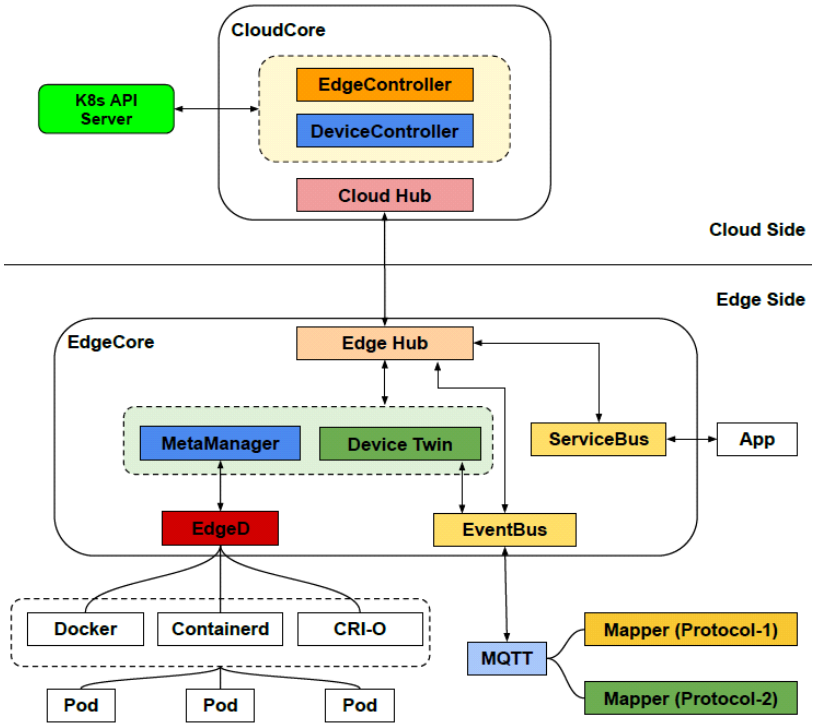

Fig. 1. KubeEdge architecture

Kubelet that is an agent to operate containers. It has the ability to register the node with the apiserver via EdgeHub and CloudHub. Kubelet receives tasks to be performed at its node like start, stop, and terminate an application container by Kubernetes master (API Server). Besides, it also sends back node and container status to the Kubernetes master.

- EventBus is an MQTT client that interacts with MQTT servers, offering pub/sub capabilities to other components.
- DeviceTwin is responsible for storing the device states.
- MetaManager is the message processor between the Edged and Edge hub.
- Servicebus acts as sending/receiving messages from applications.

Ⅲ. Practical Implementation

3.1 Cgroup driver

Cgroups are a mechanism for controlling certain subsystems in the kernel, which include devices, CPU, RAM, network access, and so on. Container runtimes on Linux also rely on cgroups. Cgroup allocates a set of specific resources to an application. It also allows container runtime such as containerd and cri-o to distribute available hardware resources to containers and optionally execute limitations and constraints[14].

The hosts running on Linux are designated with the systemd as a cgroup driver. On the other hand, the container runtime and the kubelet use cgroupfs as its cgroup driver. When using systemd and cgroupfs, different cgroup drivers will manage the same resources on a host together. In industry, people have reported that the host uses the systemd, and container runtime uses the cgroupfs to manage the host processes' resources, leading to unstable under resource pressure[14]. Figure 2 shows how to unify the cgroup driver on one system by setting the cgroup driver used by container runtime to the systemd. It needs to perform on a node before it joins a KuberEdge system[14].

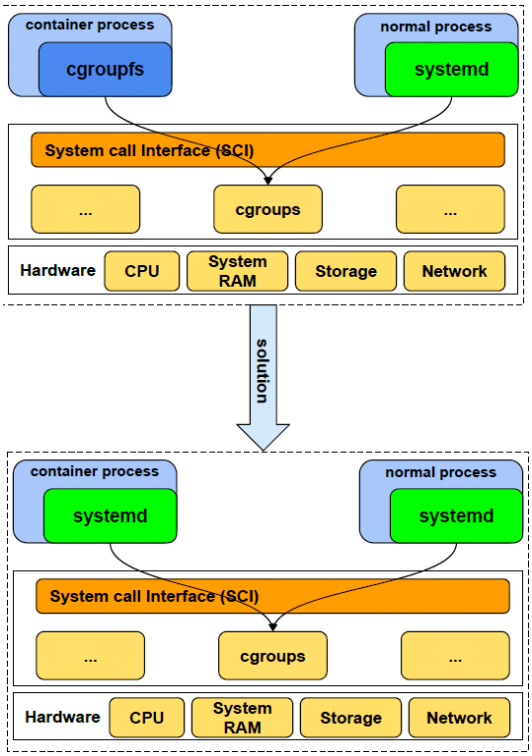

Fig. 2. The problem of cgroup drivers and solution

3.2 Implementation

All steps for setting a KubeEdge system are integrated into a proposed script tool. Additionally, the tool covers some of the prerequisites not mentioned in the KubeEdge official documents. Appropriate functions will execute on the corresponding part of a KubeEdge system. We describe step by step to deploy a KubeEdge system by the proposed tool as shown in Figure 3:

For preparing:

- The proposed tool uses the Kernel-based Virtual Machine (KVM)[15] tool built-in Linux kernel that provides a virtualization package for the kernel to behave as a hypervisor to create Virtual Machines (VM) (1).
- After creating VMs, the proposed tool sets up at least two LANs inside the KVM. One LAN is for nodes shaping a Kubernetes (k8s) cluster at the cloud side of a KuberEdge system (2). The other LAN connects the Kubernetes cluster (cloudcore) at the cloud side and the edge node of the KubeEdge system. The tool then adds network interfaces to respective VMs. On the cloud side, the k8s master node of the k8s cluster is added the number of network interfaces respective to the number of the edge sides. The tool also opens access on two ports, 10000 and 10002, in the cloud side for edge node connections.
- Kubernetes requires to disable swap memory on any cluster nodes to prevent the Kube-scheduler from assigning a Pod to a node that has run out of CPU/memory or reached its designated CPU/memory limit. Since KubeEdge is based on Kubernetes, the proposed tool disables swap memory for all nodes (3).
- All nodes in the system install containerd as the

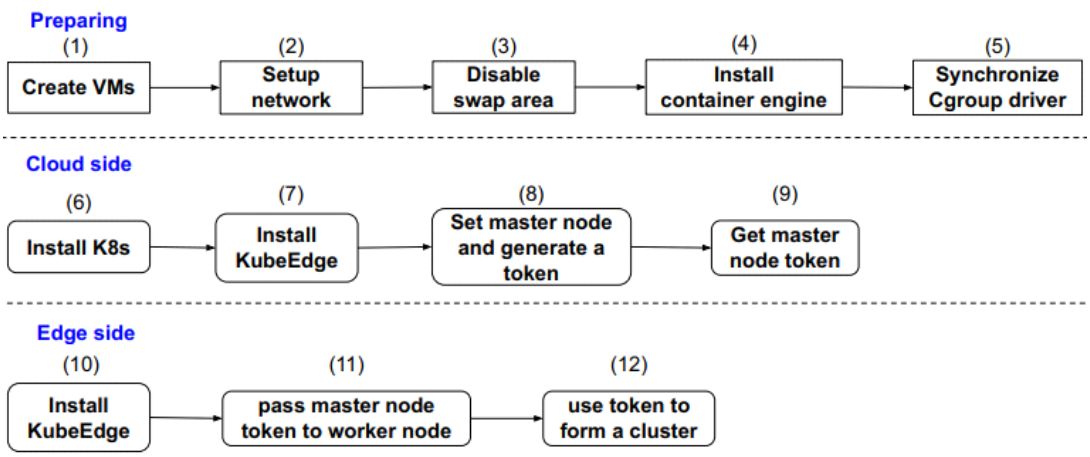

Fig. 3. The flowchart of the proposed tool

default container runtime by the proposed tool (4). Users may change to use their favorite container runtime in the proposed tool.

- As mentioned in the 3.1 section, each of the nodes in the system needs to set its container runtime to use the systemd as a cgroup driver by the proposed tool (5).

For cloud side:

- The proposed tool installs K8s (6).
- Then the proposed tool installs KubeEdge (7).
- After that, the proposed tool sends the command to set the master node and generate a token (8).
- Finally, the proposed tool gets the master node token (9).

For edge side:

- The proposed tool installs KubeEdge on the node (10).
- Then the proposed tool sends the master node token to the worker node (11).
- Finally, the worker node uses the token to join master node to form a cluster (12).

Ⅳ. Experiment

The experiment was performed on a server that has the system configuration as shown in Table 1.

Figure 4 describes nodes in the edge environment built by the proposed tool. On the cloud side, a Kubernetes cluster (cloudcore) runs on top of two virtual machines (VMs), one plays Kubernetes master node, and the other is Kubernetes worker node. The connections between nodes on the cloud side are provided by a LAN (192.168.23.0/24). On

Table 1. Specifications of the Experiment

Entity	Details
Server hardware	Intel(R) Xeon(R) CPU E3-1240 V3 @ 3.40GHz, 8 cores
Operation System	Ubuntu 18.04 LTS, 64 bit
Software	Linux KVM
VM1, VM2, edge nodes	2 cores, 4 GB memory, Ubuntu 16.04

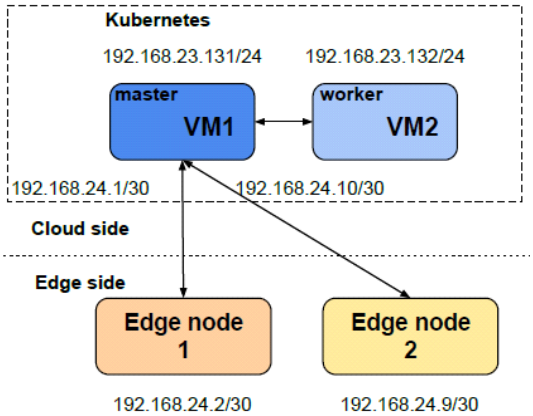

Fig. 4. Network design for experiment

the edge side, two VMs play as edge Kubernetes worker nodes, which run on their own LAN. The connection between the cloudcore and edge nodes uses different LANs (192.168.24.0/30 and 192.168.24.8/30). All LANs are created using the KVM. Figure 5 shows the result of the experiment after the tool executed.

To prove that the tool helps reduce the complexity of deployment and the time to deploy KubeEdge, we mainly focus on measuring the configuration time and installation time of the whole KubeEdge cluster. Configuration time is the delay to configure the requirements of KubeEdge. It consists of (1), (2), (3), (4), (5) in Figure 3. Installation time has different meanings in the two cases. For cloud side, installation time is the delay to install K8s, KubeEdge and generate a master node token. It consists of (6), (7), (8), (9) in Figure 3. For edge side, installation time is the delay to install

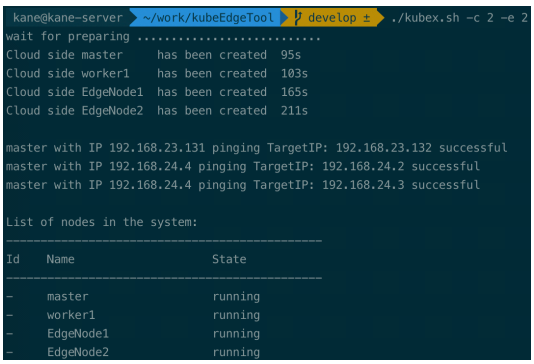

Fig. 5. Experiment results of KubeEdge deployment

KubeEdge and use the master node token to form a cluster. It consists of (10), (11), (12) in Figure 3. Table 2 shows the configuration time and the installation time for each node. Without the proposed tool, if users want to deploy a KubeEdge system, they must manually perform all setting steps, which leads to misconfiguration and causing wastes of time and resources. The proposed tool makes the deployment of a KubeEdge system easier and faster.

Table 2. Implementation Results

No.	Entity	Configuration time	Installation time
1	Kubernetes cluster	242s	157s
2	Edge Node 1	165s	122s
3	Edge Node 2	211s	118s

V. Conclusion

In this study, we presented KubeEdge architecture and difficulties when deploying a KubeEdge system in the laboratory environment. It showed that Kubernetes could manage remote edge nodes and orchestrate edge applications into the edge with only one API. KubeEdge offers significant command-line operations using kubectl that is configured to work with KubeEdge. One can deploy a KubeEdge in a fast and efficient way with the proposed tool, as shown in the experiment.

References

[1] Y.-Y. Shih, H.-P. Lin, A.-C. Pang, C.-C. Chuang, and C.-T. Chou, “An NFV-based service framework for IoT applications in edge computing environments,” *IEEE Trans. Netw. and Serv. Manag.*, vol. 16, no. 4, Dec. 2019.

[2] *KubeEdge*, [Online] Available: https://kubeedge.io.

[3] *Akri*, [Online] Available: https://github.com/deislabs/akri.

[4] *Google Anthos*, [Online] Available: https://cloud.google.com/anthos.

[5] *Kubernetes*, [Online] Available: https://kubernetes.

[6] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, “Extend cloud to edge with KubeEdge,” *2018 IEEE/ACM SEC*, pp. 373-377, Seattle, WA, USA, 2018, doi: 10.1109/SEC.2018.00048.

[7] L. Espe, A. Jindal, V. Podolskiy, and M. Gerndt, “Performance evaluation of container runtimes,” in *Proc. 10th Int. Conf. Cloud Comput. and Serv. Sci.* (CLOSER 2020), pp. 273-281, pp. 273-281, 2020.

[8] S. Fu, J. Liu, X. Chu, and Y. Hu, “Toward a standard interface for cloud providers: The container as the narrow waist,” *IEEE Internet Computing*, vol. 20, no. 2, pp. 66-71, Mar.-Apr. Feb. 2016.

[9] *CRI-O*, [Online] Available: https://cri-o.io.

[10] *Introduction crun* [Online] Available: https://www.redhat.com/sysadmin/introduction-crun.

[11] *Apache Mesos* [Online] Available: http://mesos.apache.org.

[12] *Docker Swarm* [Online] Available: https://docs.docker.com/engine/swarm.

[13] *Docker*, [Online] Available: https://www.docker.com.

[14] *Cgroup Driver*, [Online] Available: https://kubernetes.io/docs/setup/production-environment/container-runtimes.

[15] *Kernel Virtual Machine*, [Online] Available: https://www.linux-kvm.org.

레 반끄엉 (Van-Cuong Le)

Van-Cuong Le received the B.Eng. degree in software engineering from the University of Science & Technology, University of Da Nang, Da Nang City, Vietnam, in 2018. He is currently pursuing a master's degree with Soongsil University. His research interests include Software-defined Network/Network Function Virtualization.

유 명 식 (Myungsik Yoo)

Myungsik Yoo received his B.S. and M.S. degrees in electrical engineering from Korea University, Seoul, Republic of Korea, in 1989 and 1991, and his Ph.D. in electrical engineering from State University of New York at Buffalo, New York, USA in 2000. He was a senior research engineer at Nokia Research Center, Burlington, Massachusetts. He is currently a professor in the school of electronic engineering, Soongsil University, Seoul, Republic of Korea. His research interests include visible light communications, sensor networks, Internet protocols, control, and management issues.

[ORCID:0000-0002-5578-6931]