논문 21-46-11-22 The Journal of Korean Institute of Communications and Information Sciences '21-11 Vol.46 No.11
https://doi.org/10.7840/kics.2021.46.11.1958

사물인터넷 환경을 위한 경량 K3S 도구

레 반끄엉*, 유 명 식°

Lightweight K3S Tool for Internet of Things Environments

Van-Cuong Le*, Myungsik Yoo°

요 약

사물 인터넷(IoT)은 스마트홈, 환경 모니터링 및 의료 등 다양한 분야에 빠르게 적용되고 있다. 그 결과 점점 더 많은 IoT 디바이스가 다양한 사용 사례에 구축되고 있다. 일반적으로 IoT(Internet of Things) 장치들은 포설된 지역의 Rasberry Pi 및 NVIDIA Jetson Nano와 같은 소형 컴퓨터로 관리된다. 최근 들어 컨테이너 기술은 컨테이너 관리 툴인 Kubernetes(K8s)와 함께 애플리케이션을 효율적으로 설치하고 관리하기 위한 가상화 기술로 부상하고 있다. 하지만 K8s는 자원이 제한된 IoT 기기에 설치되어 운영되기 어렵다. 따라서 경량 Kubernetes인 K3s는 자원이 제한된 장치용으로 특별히 설계되었다. IoT 디바이스는 네트워크 구성 및 OS(운영 체제) 설정과 같은 K3s 클러스터의 초기화 및 가입에 대한 K3s 요구 사항을 충족해야 한다. 그러나 K3s 공식 문서에서는 이러한 요구 사항 구성을 지원하고 있지 않다. 본 논문에서는 요구사항 구성 및 K3s 클러스터 구현을 자동화할 수 있는 툴을 제안한다. 구현 및 평가를 통하여 제안된 툴이 빠르고 효과적으로 K3s를 설치할 수 있음을 보여주고자 한다.

Key Words : container, internet of things, iptables, kubernetes, k3s

ABSTRACT

The Internet of Things (IoT) is rapidly applying to many different domains, including smart homes, environmental monitoring and health care. As a result, more and more IoT devices are being deployed in a variety of use cases. Typically, fleets of Internet of Things (IoT) devices need to be managed by a small computer such as Raspberry Pi and NVIDIA Jetson Nano in the deployed region. In recent years, the containerized technology is emerging as an efficient virtualized technology for launching and managing applications along with Kubernetes (K8s), the leading containerized orchestration tool. But native K8s is heavy for the IoT devices with limited resources. Hence, K3s, a lightweight Kubernetes, is specially designed for resource-limited devices by Rancher. An IoT device has to satisfy the requirements of K3s for initialing and joining to a K3s cluster, such as network configurations and settings on the operating system (OS). However, the K3s official documents do not support configuring these requirements. We propose a tool to automate configuring requirements and deploying a K3s cluster. By implementing and evaluating, we show that the proposed tool can deploy K3s 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 : Department of Information Communication Convergence, Soongsil University, Seoul, South Korea, cuonglv@soongsil.ac.kr
° Corresponding Author : School of Electronic Engineering, Soongsil University, Seoul, South Korea, myoo@ssu.ac.kr, 정회원

논문번호 : 202107-177-D-RN, Received July 21, 2021; Revised July 31, 2021; Accepted July 31, 2021

Ⅰ. Introduction

Nowadays, more and more sensors, actuators, gadgets, called IoT Devices, have been used to support human activities in life. In addition, more and more applications have been developed to process the massive data generated by the millions of IoT devices[1,2]. Typically, the applications are designed to process raw data at the local before sending the data to the cloud for processing to get the final result[3]. To manage and orchestrate hundreds to thousands of containerized applications in a cluster, Kubernetes (K8s) is utilized[4]. But Kubernetes is so heavy for the IoT devices with limited resources[5]. Rancher K3s[6], which is based on K8s, was proposed for the devices with limited resources in the IoT environment to provide the ability similar to K8s. A node needs to satisfy the K3s configuration requirements such as enabling iptables for network configuration, enabling cgroups for resources management, disabling swap area on the node, and configuring container runtime for executing containers[7]. By default, users have to configure and install K3s manually. It will cost users a lot of time and effort, and cause errors in configurations. This paper proposes a tool to automate configuring requirements such as iptables, cgroups, installing K3s on IoT devices, and clustering IoT devices.

The rest of this paper is structured as follows. Section II describes the background on containerized technologies and K3s. 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

A container runtime is an application that manages container images and executes containers on a node. There are many container runtimes such as containerd[8], cri-o[9]. Containerd is the default Container Runtime of the Docker. It uses runc, the reference implementation of the Open Container Initiative (OCI) specification[10]. 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 is an industry-standard container runtime and also is the default container runtime in K3s because containerd uses less resources than other container runtimes.

2.2 Kubernetes

Kubernetes streamlines the process of implementing multi-container applications. Using Kubernetes, operators can adjust each container of an application exactly how much resource is allocated for each container and combine different containers within a single Pod. Kubernetes then handles the process of rolling them out, maintaining them, and ensuring that all the components remain in synchronization[4].

A Kubernetes cluster has at least one master and multiple worker nodes. A set of master nodes called a control plane is the management layer of Kubernetes. Basically, the control plane contains a Kubernetes API server providing the Kubernetes API. Every communication between Kubernetes components is through an API server. The API server communicates with Etcd, a distributed key-value database via gRPC protocol that is an open-source remote procedure proposed by Google. The API server stores metadata of the API objects in Etcd. All resources in a Kubernetes cluster are presented as API objects. The other part of the control plane is the Kube-controller-manager, which consists of many sub-controllers. Each sub-controller monitors different types of resource objects such as Pod, service, etc. Then it tries to synchronize the current state of the resource object closer to the desired state. In K8s, state is defined as a temporal property of a resource object. Finally, the control plane contains a scheduler, which is responsible for scheduling the application pods to the appropriate nodes.

In an IoT environment, the devices have limited resources, so native K8s cannot be installed on the devices.

2.3 K3s

Rancher K3s[6] is a highly available, certified lightweight Kubernetes distribution designed for applications in unattended, resource-constrained, remote locations or inside IoT appliances. It fully adheres to K8s principles, contains all essential components by default, and targets a fast, simple, and efficient way to provide a highly available and fault-tolerant cluster to a set of nodes. The architecture of K3s is shown in Figure 1[11].

For the master node:

- The API server validates and manages the components of the cluster, such as pods, replication controllers, and services.
- The controller manager is responsible for watching and sharing the state of the cluster and adjusting the state in a cluster to meet the desired state.
- The scheduler finds the created and unscheduled pods to be assigned to nodes based on the constraints and available resources.
- K3s support many databases such as etcd3[12], MySQL[13], Postgres[14], and SQLite3[15] instead of etcd, making K3s different from other lightweight Kubernetes distributions[16]. It is responsible for storing the state of applications deploying in a K3s cluster. Especially, K3s also provides an embedded SQLite database, which is a lower memory footprint and is simple to operate than etcd.
- The Tunnel Proxy is responsible for communication inside a cluster. Commands from the K3s master are sent to the K3s worker and the state of the K3s worker node also is reported to the K3s master through the Tunnel Proxy.

For the worker node:

- The kube-proxy is a network proxy and maintains network rules on nodes. The kube-proxy's role is load balancing.
- Flannel is a very simple overlay network supporting container-to-container, Pod-to-Pod, Pod-to-Service, and External-to-Service communications in a cluster.
- Tunnel Proxy is responsible for communicating with the K3s master node.
- kubelet is an agent that runs on each node of the cluster. It is directed by the API server of master, and manages the containers.

Ⅲ. The Proposed Tool

The proposed tool runs on a machine that is different from the master node or the worker node. The proposed tool needs to be provided IP addresses and security keys of all the nodes. The proposed tool connects to all nodes via a local area network (LAN), and sends commands through ssh protocol,

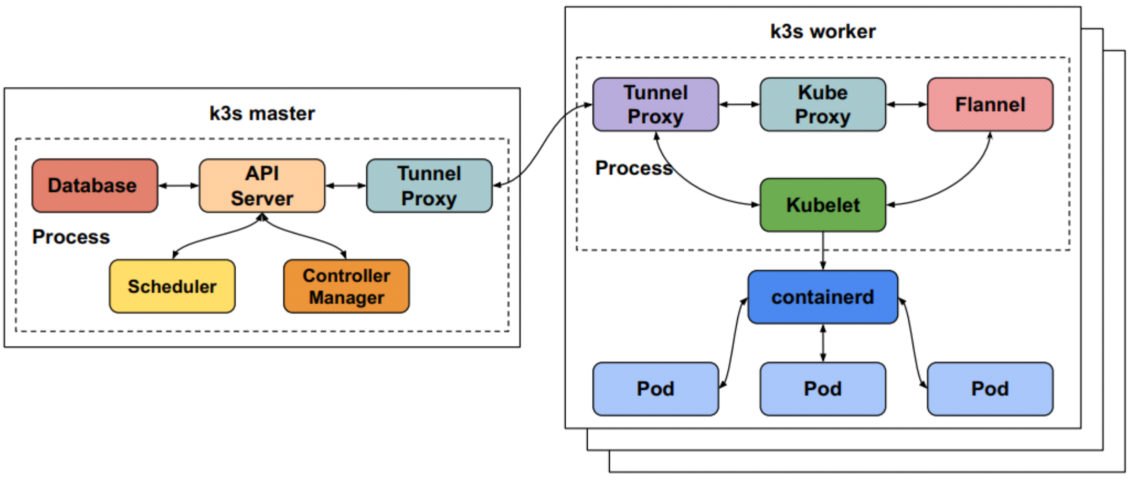


Fig. 1. The K3s architecture

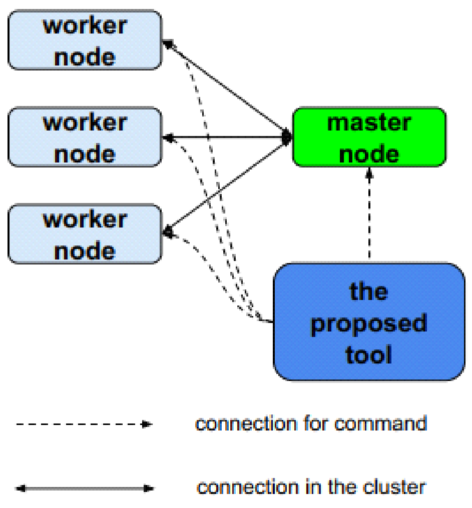


Fig. 2. The connections between the proposed tool and all of the nodes in a cluster

as shown in Figure 2.

All steps for setting a K3s cluster are integrated into the proposed tool. It covers the K3s configuration requirements such as enabling iptables for network configuration, enabling cgroups for resources management, disabling swap area on the node, and configuring container runtime for executing containers[7]. These things are not supported directly by the K3s official documents[19]. Appropriate functions will be executed on the corresponding node. We describe steps to configure requirements and install K3s on IoT devices by the proposed tool as shown in Figure 3.

Table 1. List of access ports

Protocol	Port	Source	Entity
tcp	10250	master and worker	kubelet metric
udp	8470	master and worker	require only for Flannel VXLAN
tcp	6443	worker	api server

The proposed tool for creating a bare-metal cluster is based on Raspberry Pi. It also assumes that all Raspberry Pi must set up Raspbian Buster as Operating System.

For all nodes:

- First of all, the proposed tool enables legacy iptables on Raspbian instead of nftables that Raspian Buster uses by default (1). It also opens access to the list of ports in the master node for worker node connections, as shown in Table 1.
- 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 K3s is based on Kubernetes, the proposed tool disables swap memory for all nodes (2).
- The proposed tool enables cgroups for each node

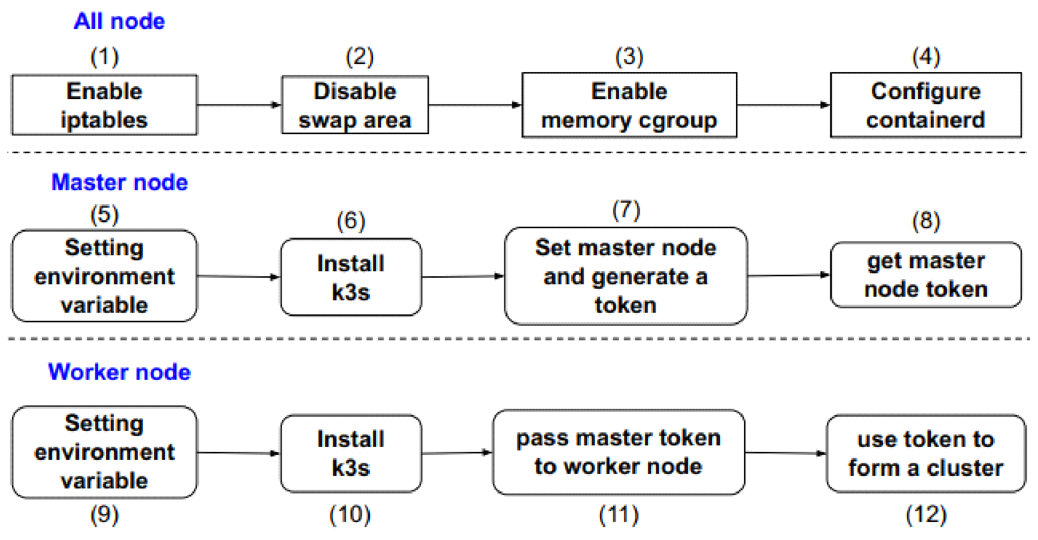

All node
(1) Enable iptables
(2) Disable swap area
(3) Enable memory cgroup
(4) Configure containerd
Master node
(5) Setting environment variable
(6) Install k3s
(7) Set master node and generate a token
(8) get master node token
Worker node
Setting environment variable (9)
Install k3s (10)
pass master token to worker node (11)
use token to form a cluster (12)

Fig. 3. The workflow of the proposed tool

(3). Typically, operating systems on IoT devices do not enable cgroup by default.
- The proposed tool installs containerd as the default container runtime for all nodes (4).

For the master node:
- The proposed tool sets up some environment variables for master node for customization (5).
- Then the proposed tool installs K3s using the official K3s installation script (6).
- After that, the proposed tool sends the command to set the master node and generate a token (7).
- Finally, the proposed tool gets the master node token (8).

For each worker node:
- The proposed tool adds the URL of K3s master node to the worker node, which consists of the master node IP address, respective port, and the master node token as the environment variables for joining the K3s cluster (9).
- Then the proposed tool then installs K3s on the node (10).
- After that, 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 devices that have the system configuration, as shown in Table 2 and Table 3.

Figure 4 shows nodes in the K3s cluster in the IoT environment built by the proposed tool for this experiment. A virtual machine (VM) is created using Kernel-based Virtual Machine (KVM), installed Raspberry Pi Desktop (OS) in the cloud part of the K3s cluster. The VM role is the master node. Two Raspberry Pi devices install Raspberry Pi OS as two worker nodes in the local part of the K3s cluster. The connections between nodes in the K3s cluster are provided by a LAN (192.168.24.0/24). We use bridged network mode for connecting the virtual network adapter of the VM to the physical network adapter. The bridged network mode helps network packets to be sent and received directly from/to without the virtual network adapter routing. Therefore, the master node can directly communicate with two worker nodes in the same LAN. Figure 5 shows the result of the experiment after the tool was executed.

To prove that the tool helps reduce the complexity of deployment and the time to deploy a

Table 2. Specifications of each Raspberry Pi in the experiment

Entity	Details
Raspberry Pi	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM
Operating System (OS) on the Raspberry Pi	Raspbian Buster ARMhf

Table 3. Specification of the Virtual Machine in the experiment

Entity	Details
Server hardware	Intel(R) Xeon(R) CPU E3-1240 V3 @ 3.40GHz, 8 cores
Server OS	Ubuntu 18.04 LTS, 64 bit
Virtualization solution	Linux KVM
Resource of the VM	2 cpu cores and 2 GB of RAM

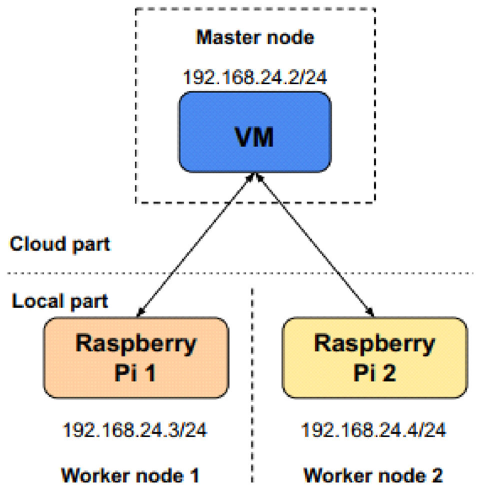
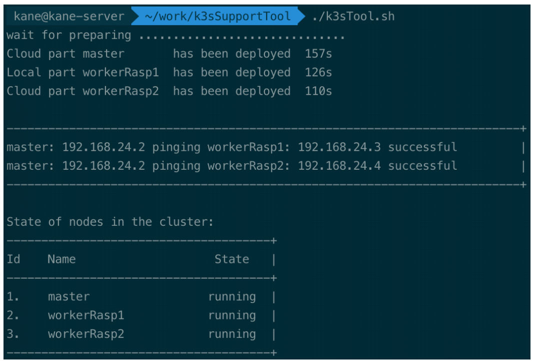


Fig. 4. Network design for experiment



```
kane@kane-server  ~/work/k3sSupportTool  ./k3sTool.sh
wait for preparing ..............................
Cloud part master       has been deployed  157s
Local part workerRasp1  has been deployed  126s
Cloud part workerRasp2  has been deployed  110s


-------------------------------------------------------------------------+
master: 192.168.24.2 pinging workerRasp1: 192.168.24.3 successful        |
master: 192.168.24.2 pinging workerRasp2: 192.168.24.4 successful        |
-------------------------------------------------------------------------+


State of nodes in the cluster:
--------------------------------------+
Id    Name                    State   |
--------------------------------------+
1.    master                 running  |
2.    workerRasp1            running  |
3.    workerRasp2            running  |
--------------------------------------+
```

Fig. 5. Experiment results of K3s cluster deployment

K3s cluster, we mainly focus on measuring the configuration time and installation time of the whole K3s cluster. Configuration time is the delay to configure the requirements of K3s. It consists of (1), (2), (3), (4) in Figure 3. Installation time has different meanings in the two cases. For master node, installation time is the delay to install K3s and generate a master node token. It consists of (5), (6), (7), (8) in Figure 3. For worker node, installation time is the delay to install K3s and use the master node token to form a cluster. It consists of (9), (10), (11), (12) in Figure 3. Table 4 shows the configuration time and the installation time for each node. Without the proposed tool, if users want to deploy a K3s cluster, they have to manually do all setting steps, which leads to misconfiguration and causing wastes of time and resources. The proposed tool makes the deployment of a K3s cluster easier and faster.

Table 4. Implementation Results

No.	Entity	Configuration time	Installation time
1	Master node	157s	96s
2	Raspberry Pi	126s	101s

V. Conclusion

In IoT environment, a million applications need to be managed and orchestrated using Kubernetes. A native K8s is heavy for IoT devices with limited resources. Therefore, K3s is proposed for the IoT environment. K3s requires complex node configuration and cluster deployment. This paper proposed a tool that helps to configure and install K3s on IoT devices, and interconnect them to form a K3s cluster without manual steps. The experiment shows that the proposed tool makes the deployment of a K3s cluster easier and faster.

References

[1] Y. Han, S. Shen, X. Wang, S. Wang, and Victor C. M. Leung, "Tailored learning based scheduling for kubernetes-oriented edge-cloud system," *IEEE INFOCOM*, 2021.

[2] M. C. Ogbuachi, A. Reale, P. Suskovics, and B. Kovacs, "Context-Aware kubernetes scheduler for edge-native applications on 5G," *J. Commun. Softw. Syst.*, vol. 16, no. 1, 2020.

[3] D. Lennick, A. Azim, and R. Liscano, "Container-based internet-of-things architecture pattern: Kill switch," *ISORC*, Nashville, TN, USA, May 2020.

[4] *Kubernetes*, [Online] Available: https://kubernetes.io/docs/home/

[5] T. Goethals, F. DeTurck, and B. Vockaert, "Extending kubernetes cluster to low-resource edge devices using virtual kuberlets," *IEEE Trans. Cloud Computing*, 2020.

[6] *K3s* [Online] Available: https://rancher.com/docs/k3s/latest/en/

[7] *K3s Requirements* [Online] Available: https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/

[8] *Containerd*, [Online] Available: https://containerd.io/

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

[10] *Open Cotnainer Initiative*, [Online] Available: https://opencontainers.org/

[11] *K3s Architecture* [Online] Available: https://rancher.com/docs/k3s/latest/en/architecture/

[12] *Etcd3* [Online] Available: https://etcd.io/docs/v3.3/learning/api/

[13] *MySQL* [Online] Available: https://www.mysql.com/

[14] *Postgres* [Online] Available: https://www.post

gresql.org/
[15] *SQLite3* [Online] Available: https://www.sqlite.org/index.html
[16] S. Bohm and G. Wirtz, "Profiling lightweight container platforms: MicroK8s and K3s in comparison to kubernetes," *13th ZEUS Wkhps.*, pp. 65-73, Germany, Feb. 2021.
[17] *K3s Official Script*, [Online] Available: https://get.k3s.io/

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]