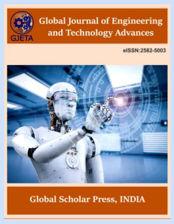
Global Journal of Engineering and Technology Advances

eISSN: 2582-5003

Cross Ref DOI: 10.30574/gjeta

Journal homepage: https://gjeta.com/



(Research Article)

Check for updates

The implementation of Container as a Service (CaaS) cloud using openSUSE kubic

I Putu Agus Eka Pratama *

Department of Information Technology, Faculty of Engineering, Udayana University, Bali, Indonesia.

Global Journal of Engineering and Technology Advances, 2021, 06(01), 001–009

Publication history**:** Received on 20 December 2020; revised on 30 December 2020; accepted on 02 January 2021

Article DOI: https://doi.org/10.30574/gjeta.2021.6.1.0123

Abstract

Currently, software development based on Development and Operations (DevOps) generally uses containers and Cloud Computing to support portability, reliability, scalability, and security. One combination of these two technologies is Container as a Service (CaaS) Cloud. From a number of previous researches, no one has studied the technical implementation and deployment of CaaS using openSUSE Kubic based on the Linux kernel and Linux operating system, as well as testing in it using Pods from the service and replication side. The test results prove that openSUSE Kubic is very easy and reliable to use for CaaS implementation and deployment, so it is very helpful for software developers in realizing software development more scalable and faster.

Keywords: Container; CaaS; Deployment; DevOps; OpenSUSE Kubic

1. Introduction

Currently, software development uses the Development and Operations (DevOps) methodology and approach, which emphasizes the process of communication, collaboration, and integration between software developers and IT professionals [1]. By using DevOps, it allows for good cooperation between the development department and the operational division, thereby reducing development costs and time and increasing performance [2]. The cost of releasing software also decreased due to software as a service, where organizations can release software early and with a high frequency, so that they have a higher capability to compete in the market.

To realize software as a service, Cloud Computing is implemented to complement DevOps. Cloud Computing is a computing model that provides convenience, comfort, and on-demand access to access and configure computing resources (network, servers, storage, applications, and services) that can be quickly released without much interaction with providers. Service [3, 4]. Software as a Service Cloud is one of three types of services provided by Cloud Computing to users according to their needs, where the other two services are Platform as a Service Cloud and Infrastructure as a Service Cloud [5].

Regarding DevOps in software development, the need for compatibility and portability in different software environments makes a container used in it. In the context of the Linux environment commonly used in software development, a container is a tool in the form of software that runs at the operating system level (especially on the Linux operating system), which can be used to create an isolated system, which can run on a single kernel-based host. Linux [6, 7]. Compared to a Virtual Machine (VM), a Container has advantages in terms of flexibility, scalability, resource savings in IT Management, and shortens software development and testing times on various platforms [8]. Two open-source containers that are widely used today in DevOps are Docker and Kubernetes [9, 10, 11].

* Corresponding author: I Putu Agus Eka Pratama
Departement of Information Technology, Faculty of Engineering, Udayana University, Bali, Indonesia.

There has been several of previous researches covering containers, Container as a Service (CaaS) Cloud, docker, and Kubernetes on DevOps. Literature study regarding container security and solutions is discussed by [12], using four use cases, namely: protecting a container from applications inside it, inter-container protection, protecting the host from containers, and protecting containers from a malicious host. The results of this research are container security requirements and possible vulnerabilities and attacks. The research about Container as a Service (CaaS) has been conducted by [13] to shows a novel architecture for task selection and scheduling at the edge of the network using Container as a Service (CaaS) at Nano Data Center at Fog Computing, with the results shown that the proposed scheme reduces the energy consumption. Research about Container as a Service (CaaS) also conducted by [14] with the purpose to improve resource utilization of CPU cores, memory size of both VMs and PMs, and minimize the number of instantiated VMs and active PMs in a cloud environment, using Best Fit (BF) and Max Fit (MF) method.

The implementation of Docker to make it easier for computer users to use the application without the need for configuration along with data transfer speed testing on the LVM and RAID used in it, has been carried out by [15] with a case study at the Department of Information Engineering of Petra Christian University. The test results show that the implementation of Docker can make it easier for users to use the computer without the need for further configuration, as well as better data transfer speeds in RAID compared to LVM. Other research on docker to support queries related to performance monitoring and logging services for a single container, performance monitoring, and log service schemes on a distributed platform, was carried out by [16] accompanied by technical and analysis. The results of this study were in the form of significance of guidelines for integration and perfection of Docker-based services.

Research on Kubernetes on microservice was conducted by [17] using the Availability Management Framework (AMF) as a solution for middleware services to handle high-availability for some applications. The results of this research shown that the service outage for some of the applications managed with Kubernetes is significantly high. Another research on Kubernetes was carried out by [18] through a proposed algorithm from the results of studying the scheduling-modules in the K8s source code, to reduce the total resource cost and the maximum load of the node, and makes the task assignment more balanced on Kubernetes. The results of this study can show that the proposed algorithm is better than the original scheduling algorithm in terms of handling load balanced on Kubernetes.

From these previous researches, there has been no specific discussion regarding the technicalities of creating a Container as a Service (CaaS) Cloud using both Linux kernel and Linux operating system. For this reason, this research raises the technical implementation and deployment of CaaS using openSUSE Kubic. OpenSUSE Kubic is a project from openSUSE that is based on Linux kernel and Linux operating system (especially openSUSE Tumbleweed), devoted to implementing containers and handling microservices [19].

In this research, two points form the problem in the form of research questions, namely: 1.) How to implement and deploy Container as a Service (CaaS) using openSUSE Kubic? 2.) How to do test the implemented and deployed CaaS using Pods?

The purpose of this research is to prove the ease of implementation and deployment of Container as a Service (CaaS) based on openSUSE Kubic along with testing in it using Pods, so that it can help software developers with the DevOps method in developing and testing software better and faster based on containers.

2. Material and methods

To support the implementation and testing of this research, several supporting hardware and software were used. The hardware used is a Dell Latitude E6440 Notebook (Intel Core i7, 16GB RAM, 500GB hard drive), internet modem, and mouse. The software used is the openSUSE Kubic (MicroOS) 64 bit with Docker, Kubernetes, and Pod inside it. The research was conducted in the period from the beginning of May 2020 to mid of December 2020 by online during the Covid19 pandemic.

This research uses a methodology in the form of a Systematic Literature Review (SLR) [20] utilizing some references from various published papers, books, and website URLs on the internet, related to containers, dockers, Kubernetes, Container as a Service, along with several related studies that have been done previously. as a state of the art.

In this study, assessment indicators are used, namely: 1.) The ability of openSUSE Kubic to assist the implementation and deployment process of Container as a Service (CaaS) through the available modules, where every step of implementation and testing is observed and recorded, and 2.) Testing the implementation and deployment of CaaS based on openSUSE Kubic using Pods available in it to test service and replication processes.

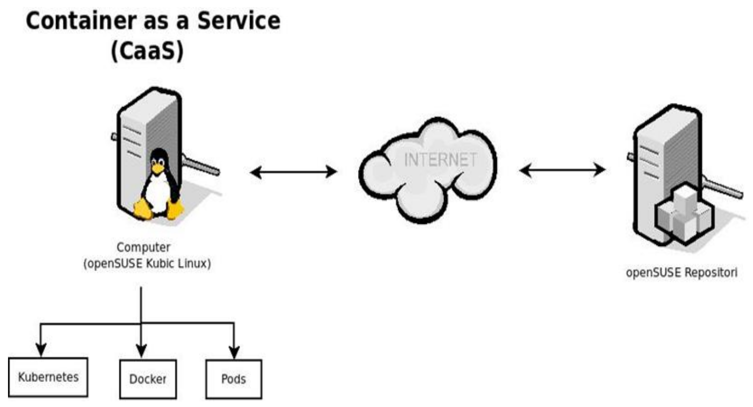


Figure 1 Test Scenario

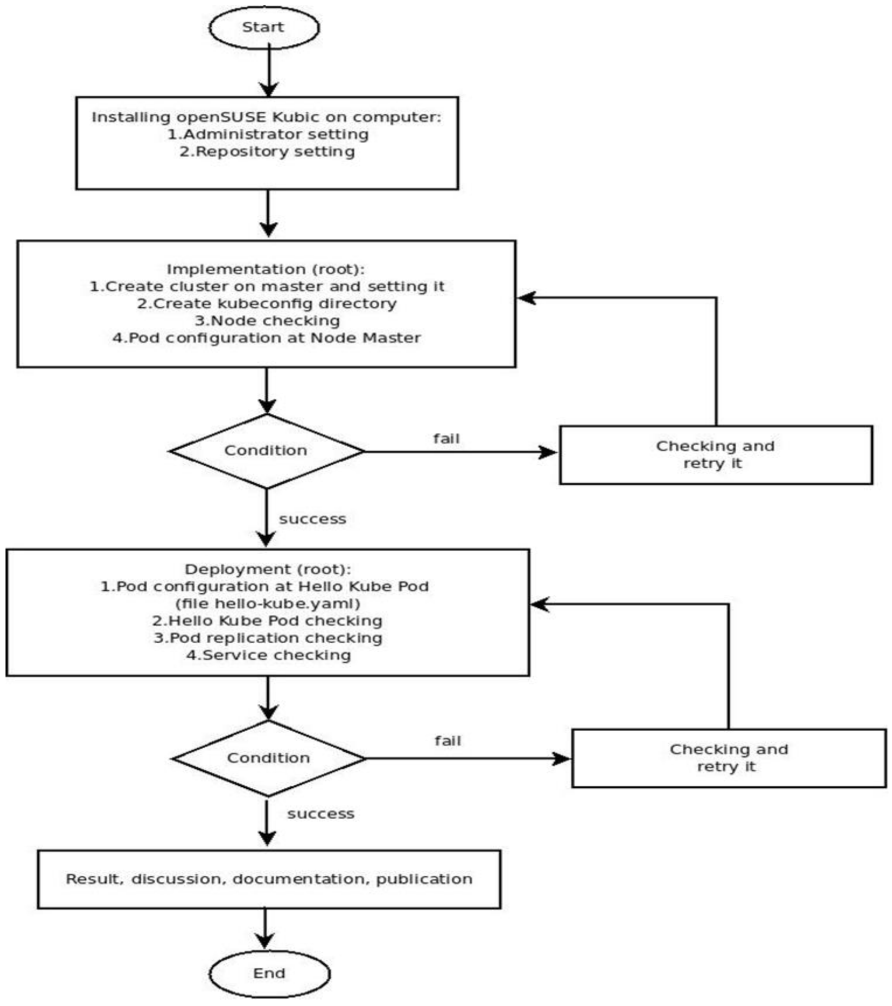

Start
Installing openSUSE Kubic on computer:
1.Administrator setting
2.Repository setting
Implementation (root):
1.Create cluster on master and setting it
2.Create kubeconfig directory
3.Node checking
4.Pod configuration at Node Master
Condition
fail
Checking and retry it
success
Deployment (root):
1.Pod configuration at Hello Kube Pod
(file hello-kube.yaml)
2.Hello Kube Pod checking
3.Pod replication checking
4.Service checking
Condition
fail
Checking and retry it
success
Result, discussion, documentation, publication
End

Figure 2 Research Flowchart

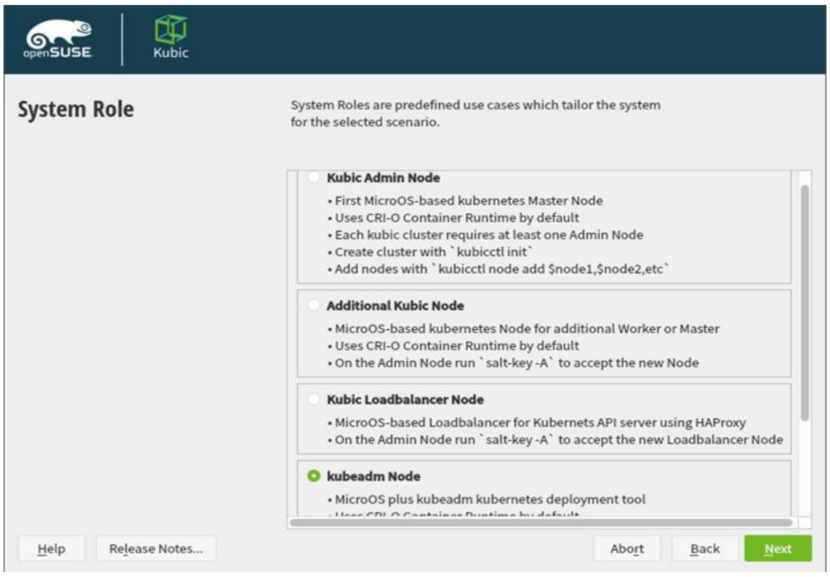
In this study, observations were made, saved screenshots, recorded, analyzed, and discussed the test results, to be evaluated and obtained conclusions that answered the research questions. To support the implementation, deployment, and testing carried out in this study, the openSUSE Linux online repository is used. The openSUSE online repository is a digital repository for storing thousands of additional applications and libraries that can be quickly and easily added to the online operating system if required by the user, providing by SUSE [21] and replicated locally in Indonesia (mirror) [22].

The implementation and testing steps in this research were carried out based on the design of the test scenario and the research flowchart. The test scenario involves a computer to then install and configure openSUSE Kubic to implement, deploy, and testing Container as a Service (CaaS) utilizing Pods. The computer also including with Kubernetes, Docker, and Pods inside it. While the research flowchart describes the sequence of steps and processes carried out during this research. The testing scenario and research flowchart are shown in Figure 1 and Figure 2, respectively:

3. Implementation and Testing Phase

3.1. Installing openSUSE Kubic with Administration and Repository Setting

The first step is to install openSUSE Kubic on the computer. Installation using the OpenSUSE MicroOS .iso file according to the instructions on the screen, choosing a system role in the form of Kubeadm Node with the kubeadm init command, configuring the root password, and configuring the online repository. In this research, the system role is using kubeadm Node, as shown in Figure 3 below:



System Role

System Roles are predefined use cases which tailor the system for the selected scenario.

Kubic Admin Node
- First MicroOS-based kubernetes Master Node
- Uses CRI-O Container Runtime by default
- Each kubic cluster requires at least one Admin Node
- Create cluster with `kubicctl init`
- Add nodes with `kubicctl node add $node1,$node2,etc`

Additional Kubic Node
- MicroOS-based kubernetes Node for additional Worker or Master
- Uses CRI-O Container Runtime by default
- On the Admin Node run `salt-key -A` to accept the new Node

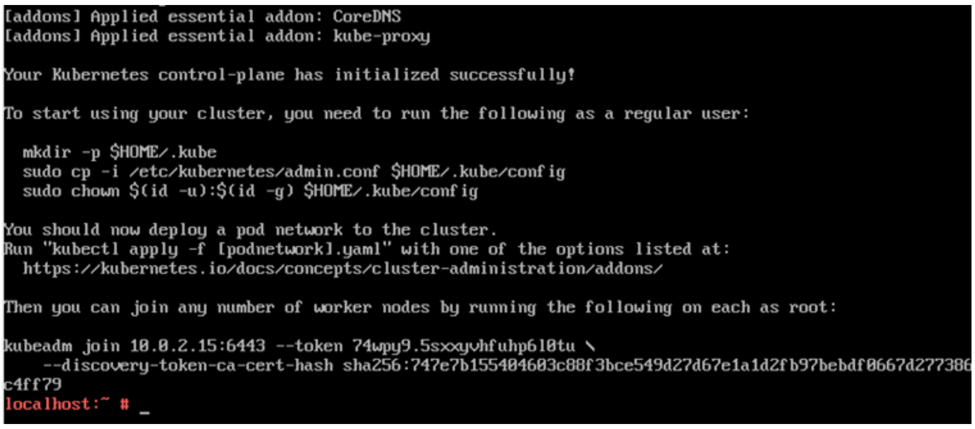
Kubic Loadbalancer Node
- MicroOS-based Loadbalancer for Kubernets API server using HAProxy
- On the Admin Node run `salt-key -A` to accept the new Loadbalancer Node

[✓ **kubeadm Node**]
- MicroOS plus kubeadm kubernetes deployment tool
- Uses CRI-O Container Runtime by default

Figure 3 Open SUSE System Roles

3.2. Implementation

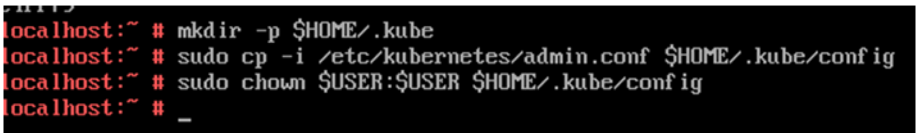
After installation, the login process is carried out as root into the system. Then the master cluster is created using the kubeadm init command. After the cluster is formed and initialized, the kubeadm join command is executed so that workers can be merged into the cluster with the token, as shown in Figure 4 below:



```
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.0.2.15:6443 --token 74wpy9.5sxxyvhfuhp610tu \
    --discovery-token-ca-cert-hash sha256:747e7b155404603c88f3bce549d27d67e1a1d2fb97bebdf0667d277386
c4ff79
localhost:~ # _
```

Figure 4 Merger the Worker into the Cluster

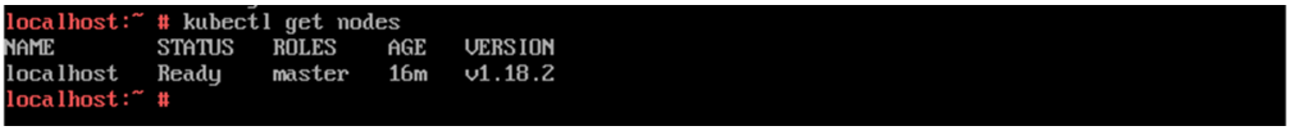
After the workers are merged into the cluster, a kubeconfig directory is created in Kubernetes, located at $HOME/.kube/config using the mkdir command. Then the admin.conf file is copied from /etc/ kubernetes/ into the $HOME/.kube/config directory using the cp -i command, ending by changing the ownership of the $HOME/.kube/config directory to root. This process shown at Figure 5 below:



```
localhost:~ # mkdir -p $HOME/.kube
localhost:~ # sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
localhost:~ # sudo chown $USER:$USER $HOME/.kube/config
localhost:~ # _
```

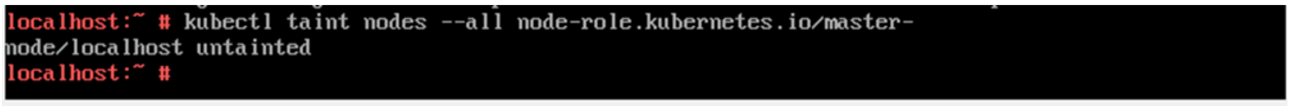
Figure 5 Create kubeconfig directory and copy the files

The final step of implementation is to checking the node status using the kubectl get nodes command. The output displayed is that there is one node connected to the cluster, namely localhost with the master roles. Then do the master node configuration on a single node using the command kubectl taint nodes --all node.role.kubernetes.io/master-node/localhost untainted to remove restrictions from pods running on the master node (in this research it is a single node). The process of checking the node status and master node configuration, shown at Figure 6 and Figure 7, respectively:



```
localhost:~ # kubectl get nodes
NAME        STATUS    ROLES     AGE    VERSION
localhost   Ready     master    16m    v1.18.2
localhost:~ #
```

Figure 6 Checking the Node Status and Master Node Configuration



```
localhost:~ # kubectl taint nodes --all node-role.kubernetes.io/master-
node/localhost untainted
localhost:~ #
```

Figure 7 Checking the Node Status and Master Node Configuration

3.3. Deployment

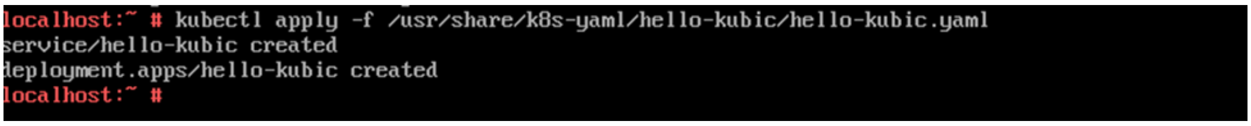
After the initial implementation of Container as a Service (CaaS) using openSUSE Kubic was successfully carried out, then continued with the deployment stage. For the deployment of this research, the hello-kubic.yaml [23] file is used. The hello-kubic.yaml file is located in /usr/share/k8s-yaml/hello-kubic and is used as a Pod.

The Hello-kubic Pod is configured to perform 3 replications, which are managed via the replicas: 3 line in hello-kubic.yaml. The contents of the hello-kubic.yaml file are as follows:

```
apiVersion: v1
kind: Service
metadata:
  name: hello-kubic
spec:
  type: LoadBalancer
  # loadBalancerIP: 192.168.111.40
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: hello-kubic
---
apiVersion: apps/v1 6
kind: Deployment
metadata:
name: hello-kubic
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-kubic
  template:
    metadata:
      labels:
        app: hello-kubic
    spec:
      containers:
      - name: hello-kubic
        image: registry.opensuse.org/kubic/hello-kubic:latest
        ports:
        - containerPort: 8080
        imagePullPolicy: Always
        env:
        # - name: MESSAGE
        #   value: I haven't specified a message yet
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: POD_IP
          valueFrom:
```

```
        fieldRef:
          fieldPath: status.podIP
    - name: POD_SERVICE_ACCOUNT
      valueFrom:
        fieldRef:
          fieldPath: spec.serviceAccountName
```

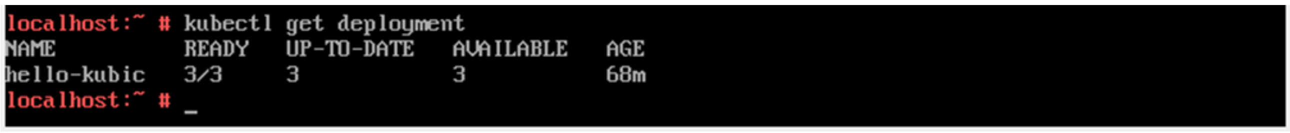
The deployment process uses hello-kubic was done using the command kubectl apply –f /usr/share/k8s-yaml/hello-kubic/hello-kubic.yaml as shown in Figure 8 below:



```
localhost:~ # kubectl apply -f /usr/share/k8s-yaml/hello-kubic/hello-kubic.yaml
service/hello-kubic created
deployment.apps/hello-kubic created
localhost:~ #
```

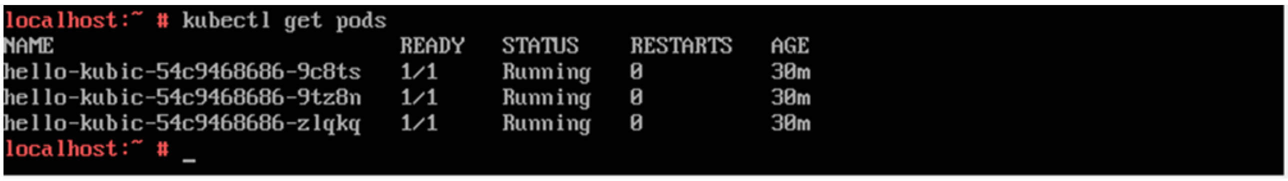
Figure 8 Deployment Process

After the deployment using hello-kubic is complete, then checking the Pod status again using the kubectl get deployment command. From the inspection using kubectl get pods command, it was found that the hello-kubic Pod made three replicates successfully, where all three replicates were available, ready, and running. The process of checking the Pod status after deployment using hello-kubic and all of three replicates were available and running, shown in Figure 9 and Figure 10 below, respectively:



```
localhost:~ # kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
hello-kubic   3/3     3            3           68m
localhost:~ # _
```

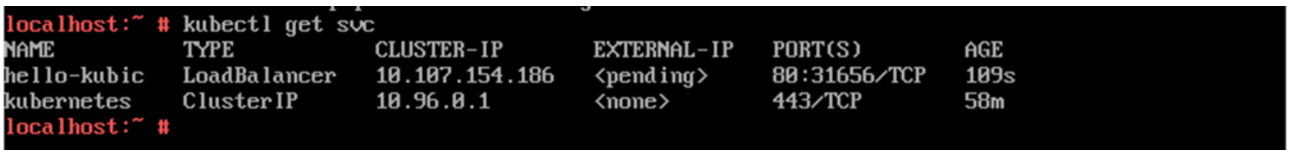
Figure 9 Checking the Pod Status After Deployment Using hello-kubic



```
localhost:~ # kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-kubic-54c9468686-9c8ts   1/1     Running   0          30m
hello-kubic-54c9468686-9tz8n   1/1     Running   0          30m
hello-kubic-54c9468686-zlqkq   1/1     Running   0          30m
localhost:~ # _
```

Figure 10 Checking all Three Replicates Running

Finally, checking the services running on CaaS that have been implemented and deployed. Checks running services using the kubectl get svc command. Figure 11 below shown the output of this command:



```
localhost:~ # kubectl get svc
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
hello-kubic   LoadBalancer   10.107.154.186   <pending>     80:31656/TCP   109s
kubernetes    ClusterIP      10.96.0.1        <none>        443/TCP        58m
localhost:~ #
```

Figure 11 Checking the Service Running

4. Results and discussion

Containers are basically an extension of virtualization using the concept where services and resources run isolated in different libraries, drivers, and binaries, through resource sharing on the operating system. Thus, the containers can run on the same machine and can share the operating system kernel, because each container runs as an isolated process. Containers isolate only libraries to run so they can be more effective and lightweight when running. From the Container concept, Container as a Service (CaaS) runs as a container-based service in the Cloud Computing network that summarizes all running services. This makes software development faster, scalable, secure, efficient, and supports high portability.

From the output shown in Figure 11, it can be observed and discussed that there are two running services, namely kubernetes service and hello-kubic service. The first service is hello-kubic service. This service is Load Balancer type, with CLUSTER-IP address of 10.107.154.186 and runs on port number 80 and port node number 31656. It is running on Network Layer using Transmission Control Protocol (TCP). The second service is kubernetes service. It has a ClusterIP type with a ClusterIP address 10.96.0.1 and runs on port 443. Same as the first service, kubernetes service also running on Network Layer using Transmission Control Protocol (TCP). With the concept of container and Container as a Service (CaaS), the running of two services indicates that CaaS deployment using openSUSE Kubic has been successfully implemented.

5. Conclusion

From the testing that has been done, it can be concluded that openSUSE Kubic can be relied on to implement and deploy Container as a Service (CaaS) quickly and easily in handling several services and applications in the form of containers, with Kubernetes and Pod support. Based on openSUSE Tumbleweed and open-source Linux kernel, this makes it easier for software developers and organizations to implement and deploy CaaS as needed.

Compliance with ethical standards

Acknowledgments

Gratitude to the Indonesian Linux and FOSS community, Indonesian openSUSE community (OpenSUSE ID), Udayana University, and my family for the support during this research.

Disclosure of conflict of interest

There are no conflict of interest at this manuscript.

References

[1] Jabbari ER, Ali N, Petersen K, Tanveer B. What is DevOps? A Systematic Mapping Study on Definitions and Practices. Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh Scotland UK May. 2016.

[2] Erich FMA, Amrit C, Danevaa M. A Qualitative Study of DevOps Usage in Practice. Journal of Software: Evolution and Process. 2017; 02(10).

[3] Mell P, Grance T. The NIST Definition of Cloud Computing, Recommendations of the National Institute of Standards and Technology. NIST Publication. 2011; 800-145.

[4] Jackson KL, Goessling S. Architecting Cloud Computing Solutions: Build Cloud Strategies That Align Technology and Economics While Effectively Managing Risk. Birmingham; Packt Publishing Ltd. 2018.

[5] MP Vaishnnave, KS Devi, P Srinivasan. A Survey on Cloud Computing and Hybrid Cloud, International Journal of Applied Engineering Research. 2019; 14(2): 429-434.

[6] Rajdeep D, et al. Virtualization vs Containerization to Support PaaS. IEEE International Conference on Cloud Engineering (IC2E), Boston USA. March 2014; 610-614.

[7] Canonical, Ltd. What's LXC? [internet]. USA: Canonical; ©2020 [cited: 2020 Dec 10].

[8] Seo K, Hwang H, Moon I, Kwon O, Kim B. Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud. International Conference on Computing, Networking and Communications (ICNC 2014), Honolulu Hawaii, USA, February 2014.

[9] Docker Inc. Get Started with Docker [internet]. USA: Docker, Inc.; ©2020 [cited: 2020 Nov 24].

[10] Linux Foundation. What is Kubernetes? [internet] USA: Linux Foundation; ©2020 [cited: 2020 Oct 15].

[11] Khare N. Docker Cookbook: 80 Hands On Recipes to Efficiently Work With the Docker Environment on Linux. USA: Packt Publishing. 2015.

[12] Sultan S, Ahmad I, Dimitriou T. Container Security: Issues, Challenges, and the Road Ahead. IEEE Access. 2019; 7(3): 52976-52996.

[13] K Kaur, T Dhand, N Kumar, S Zeadally. Container as a Service at the Edge: Trade off Between Energy Efficiency and Service Availability at Fog Nano Data Centers. IEEE Wireless Communications. 2017; 24(3): 48-56.

[14] Hussein MK, Mousa MH, Alqarni MA. A Placement Architecture for a Container as a Service (CaaS) in a Cloud Environment. Journal of Cloud Computing: Advances, Systems and Applications. 2019; 8(7).

[15] Adi K, Henry NP, Justinus A. Eksplorasi Pemanfaatan Docker untuk Mempermudah Pengelolaan Instalasi Komputerdi Laboratorium Komputer Teknik Informatika Universitas Kristen Petra [Master thesis]. Surabaya, Indonesia: Universitas Kristen Petra. 2020.

[16] Lijuan L. Research and Implementation of Docker Performance Service in Distributed Platform. International Journal Of Engineering And Computer Science (IJECS). 2017; 6(11).

[17] Leila AV. Kubernetes as an Availability Manager for Microservice Applications [Master thesis]. Montreal, Canada: Concordia University. 2019.

[18] Zhang WG, Ma XL, Zhang JZ. Research on Kubernetes Resource Scheduling Scheme. Proceedings of the 8th International Conference on Communication and Network Security (ICCNS). 2018; 144–148.

[19] OpenSUSE. Portal Kubic Welcome to Kubic Portal [internet]. Germany: OpenSUSE; ©2020 [cited: 2020 May 25].

[20] Guillaume L. Systematic Literature Reviews: An Introduction. Proceedings of the Design Society: International Conference on Engineering Design. 2019; 1(1): 1633 – 1642.

[21] SUSE, LLC. OpenSUSE Repositories [internet]. Germany: OpenSUSE; ©2020 [cited: 2020 Oct 15].

[22] LinuxSec. Daftar Repository Lokal Indonesia OpenSUSE Leap 42.1 [internet]. Indonesia: LinuxSec; ©2015 [cited: 2020 July 18].

[23] Thkukuk. Hello Kubic Hello World POD for openSUSE Kubic [internet]. USA: Github; ©2019 [cited: 2020 Sept 30].