Performance analysis of video streaming service migration using container orchestration

F R Poetra¹*, S Prabowo¹, S A Karimah¹, and R D Prayogo²

¹ Telkom University, Bandung, Indonesia
² Universitas Muhammadiyah Bandung, Indonesia

*fauzanrambang@student.telkomuniversity.ac.id

Abstract. This paper proposes an implementation of container orchestration to accelerate the migration process from one cloud to another. The use of cloud computing as the infrastructure of a system is in high demand. Cloud computing has been adopted in the use of various applications that are very useful for organizations in establishing a system to reduce effort, cost, and time. The system that runs on the same cloud provider has disadvantages. If the cloud provider goes down, then all of the running services will be affected. Therefore, it is needed the solution to migrate the services to some containers with one controller. In this research, two types of architecture are compared, i.e. container orchestration architecture using Kubernetes and simple container architecture using Docker. We use Kubernetes since it is one of the most popular container orchestrators widely used in big company’s productions. The results show that the container creating time average in the container orchestration architecture using Kubernetes is 26 times faster and the percentage average of resource usage on CPU utilization is 3.6 times smaller compared to simple container architecture using Docker.

1. Introduction

The use of cloud computing as a system infrastructure is increasingly becoming popular since it has been adopted to run several applications based on the idea of online computational resources [1]. Cloud computing service providers work by connecting multiple servers and utilizing virtual applications that can produce large-scale computing and storage. It is very influential to reduce the cost when making the system since it is not necessary to spend the budget for the service, software, or infrastructure [2]. Users are provided by the flexibility and reliability of resources with high-performance [3].

Virtualization is a technology using hardware resources that allows users to share computing resources under any conditions. Two types of technology used in hardware virtualization, i.e. container-based virtualization and hypervisor-based virtualization [4]. A container is an evolutionary form of hypervisor-based virtualization that can share operating system access without virtual machines. The container-based virtualization architecture is shown in Figure 1.

Some studies implement hypervisor-based virtualization as the main architecture and the solution to improve hypervisor-based virtualization is to implement container-based virtualization. As in Li, hypervisor-based virtualization is compared to container-based virtualization [5]. The results show that
container-based virtualization is lighter than hypervisor-based virtualization since the container-based virtualization does not require other peripherals to run applications and has a well-isolated system.

![Figure 1. Container-based virtualization architecture.](image)

The study of various types of virtualization and hypervisor techniques is observed in Singh and Srivastava [6,7], the results show the comparison of hypervisors, i.e. type 1 (Bare-metal hypervisor) and type 2 (hosted hypervisor). As in Sabahi, a new security architecture is proposed in hypervisor-based virtualization to secure a cloud environment [8]. It is concluded that the distributed security system shows a very good way to reduce the workload from hypervisor-based virtualization.

The next problem is the cloud computing architecture since all services work in one system. When the system is down, all services in the system will be down. To overcome this problem, a service migration process is carried out from one cloud system to another. To speed up the process, a scheme called a container operation is implemented [9]. Container orchestration is an open-source system as a scheme that has a main system to control several containers that contain services to facilitate the process of scaling, repairing, and migrating [10].

In the study Xie, the performance of the container without container orchestration is compared to the container using container orchestration by observing the performance of CPU and disk I/O [11]. The results show that the performance of the container using container orchestration is more efficient than that of the container without container orchestration. Reference Vayghan observes the ability of Kubernetes to restore the lost nodes by comparing the container creating time when pods containing containers are deleted manually and using a traffic generator [12]. Kubernetes is an open-source platform that serves as a controller of containers by creating cluster blocks that can contain various pod containers [13]. In the implementation of Kubernetes, it is used two nodes, i.e. master and worker. In this research, we propose a comparison of two architectures, i.e. container orchestration architecture using Kubernetes and simple container architecture using Docker by observing the parameter of the container creating time and CPU utilization.

2. Method

2.1. Kubernetes components

Kubernetes consists of some mandatory and optional parts [14]. The master node consists of API, scheduler, and controller. API server as a connector between the master node and the worker node. Scheduler serves for the container installation process on the cluster. The controller is the head of the master node that serves for decision making from the state sent by a worker node. The worker node consists of Kubelet, Kub-proxy, pod container. Kubelet sends the state to the master node. Kub-proxy connects Kubelet and container engine. Pod-container is the instance in the Kubernetes cluster that contains a container image. Kubernetes architecture is shown in Figure 2.
2.2. Pod lifecycle
Pod Lifecycle is one of the features provided by Kubernetes that allows the pod to repair itself using the controller manager located in the master node [14]. The process is to insert the container image into the pod, the container image information will be stored in the storage class in the pod. When the pod is damaged, the storage class will not be damaged. The controller manager requests the previous container image information through the Persistent Volume Claim (PVC) to be forwarded to the storage class and continues to create a new pod. The process of retrieving container image information is shown in Figure 3.

![Figure 3. The entity function of Kubernetes.](image)

2.3. System topology
This study has two main architectural systems, i.e. Kubernetes architecture and simple Docker architecture. Kubernetes cluster is installed in Kubernetes architecture, i.e. master node in system host 1 and worker node in system host 2. It continues to install the Docker container engine followed by installing the Nginx video streaming server container image into the container that was created in the worker node as shown in Figure 4. The container engine is installed in simple Docker architecture on the host system, followed by installing the container image in the container engine as shown in Figure 5.

![Figure 4. Proposed Kubernetes architecture.](image)
2.4. Testing scenario
The parameters measured in this study are container creating time and CPU utilization. By performing a container removal scenario manually, a comparison of the time needed to rebuild the lost container is done. By using the bash script image migrate and load image command to run the migrating process of containers on a simple Docker architecture and using the delete pod Kubernetes command on the container architecture orchestration. In CPU utilization, a comparison of the percentage of resource usage when migrating on the host is performed.

![Figure 5. Proposed simple Docker architecture.](image)

3. Result and discussion

3.1. Container creating time
The container creating time is obtained by performing 30 times of testing on both architectures. In the container orchestration scheme is performed the command kubectl delete pod namespace and run a bash script on the Docker scheme that contains a collection of docker stop, commit, save, scp filename host destination, i.e. _le destination, docker load, and docker run.

The comparison of the container creating time results in Kubernetes architecture and simple Docker architecture is shown in Figure 6 and Figure 7, respectively. The average of the container creating time in Kubernetes architecture is 7.67 seconds, while in simple Docker architecture is 198.73 seconds. These results are due to Kubernetes architecture has a pod lifecycle feature. The feature allows pod containing container able to create a new instance by duplicating the Docker image information inside the container at the previous instance stored in the storage class. In the simple Docker architecture, the running Docker container must be stopped rst. The container will be saved as a Docker image _le in the form of .tar. It is followed by running secured copy files (SCP) command through ssh that are connected between two different computer hosts and done manually.

3.2. CPU utilization
The percentage of CPU utilization is obtained by performing 10 times of testing on both architectures. The CPU utilization is calculated by running the TOP command provided by the Ubuntu operating system. After recording the resource usage, a sum of the percentages of both hosts in the two architectures (master node and worker node in Kubernetes and Host 1 and Host 2 in Docker) is performed.

The percentage comparison of the CPU utilization results in Kubernetes architecture and simple Docker architecture is shown in Figure 8. The percentage of the CPU utilization in Kubernetes architecture is 15.3%, while in simple Docker architecture is 55.4%. These results are due to the resource usage on the host in Kubernetes architecture takes a little time, while in the Docker architecture takes a long time for the entire migration process. This causes the resource usage still running until the migration process is complete. The more time needed in the migration process, the more resources used in rebuilding container.
4. Conclusion
The comparison of container orchestration architecture using Kubernetes and simple container architecture using Docker has been implemented in video streaming service migration. The parameters compared in this study are obtained by measuring the container creating time and CPU utilization percentage. The results show that the process of container migration using Kubernetes architecture by rebuilding pod that contains container takes the container creating time average 26 times faster than container migration using Docker architecture. The resource usage of CPU utilization in the container orchestration architecture using Kubernetes is 3.6 times smaller than a simple container using Docker. It can be concluded that the use of container orchestration can accelerate the migration process from one cloud to another with smaller resource usage.

Figure 6. Container creating time results of Kubernetes architecture(s).

Figure 7. Container creating time results of Docker architecture(s).

Figure 8. The percentage comparison of the CPU utilization in Kubernetes and Docker.

Figure 9. The percentage comparison of the CPU utilization in Kubernetes and Docker.
Acknowledgments
Authors would like to thank Telkom University and Universitas Muhammadiyah Bandung for supporting this research.

References
[1] Shah J L 2014 Cloud computing: The technology for next generation *Int. Journal of Advances in Computer Science and Technology* 3(3) pp 152-155
[2] Aljabre A 2012 Cloud computing for increased business value *Int. Journal of Business and Social Science* 3(1) pp 234-239
[3] Dar A R and Ravindran D 2018 A comprehensive study on cloud computing paradigm *Int. Journal of Advance Research in Science and Engineering* 7(4) pp 235-242
[4] Silva V G, Kirikova M and Alksnis G 2018 Containers for virtualization: An overview *Applied Computer Systems* 23(1) pp 21-27
[5] Li Z, Kih M, Chen Y and Zhang H 2018 Two-stage performance engineering of container-based virtualization *Advances in Science, Technology and Engineering Systems Journal* 3(1) 521-536
[6] Singh B and Singh G 2018 A study on virtualization and hypervisor in cloud computing *Int. Journal of Computer Science and Mobile Applications* 6(1) 17-22
[7] Srivastava S and Singh S P 2016 A survey on virtualization and hypervisor-based technology in cloud computing environment *Int. Journal of Advanced Research in Computer Engineering and Technology* 5(2) 430-435
[8] Sabahi F 2012 Secure virtualization for cloud environment using hypervisor-based technology *Int. Journal of Machine Learning and Computing* 2(1) 39-45
[9] Park J, Yun D, Kim U and Yeom K 2017 Approach for cloud recommendation and integration to construct user-centric hybrid cloud *IEEE Int. Conf. on Smart Cloud (SmartCloud)* pp 24-32
[10] Kim D, Muhammad H, Kim E, Helal S and Lee C 2019 TOSCA-Based and federation-aware cloud orchestration for Kubernetes container platform *Appl. Sci.* 9 191 1-13
[11] Xie X, Wang P and Wang Q 2017 The performance analysis of Docker and rkt based on Kubernetes 13th *Int. Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* 2137-41
[12] Vayghan, L A, Saied, M A, Toeroe, M and Khendek, F 2018 Deploying microservice based applications with Kubernetes: Experiments and lessons learned *IEEE 11th Int. Conf. on Cloud Computing (CLOUD)* pp 970-973
[13] Gupta T and Dwivedi A 2017 Data storage and load balancing in cloud computing using container clustering *Int. Journal of Engineering Sciences and Research Technology* 6(9) pp 656-666
[14] Thota S 2018 Docker and google kubernetes *Int. J. Adv. Res.* 6(6) pp 984-998