Study on network management systems by using Docker Kubernetes

E Rohadi\textsuperscript{1,2,*}, C Rahmad\textsuperscript{1,2}, F Chrissandy\textsuperscript{2} and A Amalia\textsuperscript{2}

\textsuperscript{1} Department of Information Technology, State Polytechnic of Malang, Jalan Soekarno Hatta 9, Malang, Indonesia
\textsuperscript{2} Department of Electrical Engineering, State Polytechnic of Malang, Jalan Soekarno Hatta 9, Malang, Indonesia

*erfanr@polinema.ac.id

Abstract. The development of web based applications is growing rapidly along with technological advances recently. Therefore the user of web and mobile applications are also increase. So applications can be accessed using a variety of devices such as computer, mobile platform, notebook, that’s why ease in the process of deployment of web applications to the server is needed. The previous monolithic technology, the deployment process requires a large amount of resources, scalability, system availability and computational service capabilities. Recently, we present container technology is the solution to dealing large computing services with high traffic, container technology system is created one with libraries, application and configuration files and everything needed. The container can be managed with kubernetes to makes it easier for administrator to control management data efficiently. In this study the retrieval of data is taken from QOS parameters consisting of: throughput, response time. Data from the test will determine the reliability of docker kubernetes based container technology. The suggested result is to compare the QOS multi container process with a single container or virtual private server, deployment and provide an alternative solution to make it easier for administrators to manage containers using the Kubernetes more easily and efficiently.

1. Introduction

Containerization technology as a virtualization implementation in OS level has gained popularity in recent years replacing several virtual machine roles, for run the application server in implementing the latest concepts in software development, such as DevOps and microservices architecture. In its implementation, Such server applications generally involve a number of container at once. In this condition, a system is needed for scheduling, management and provisioning of each container which runs in a cluster, or better known as container orchestration system [1]. But the container scheduler available on orchestration systems like Kubernetes generally only focuses on the availability of system resources and general threshold. By combining the scalability process in technology Kubernetes, the container automation process can be implemented according to the number of users accessing [2]. The container will automatically add if the user exceeds certain limitation limits. This container scalability process can be applied to Kubernetes with several parameters which supports. It is expected that the application of scalability will improve performance and server response time against the user without reducing the ability of the server utility.
The scheduler is created to run on the Kubernetes and do the optimization process of container allocation in the application as shown Figure 1.

- Cluster Efficiency manipulate container allocation on the cluster for minimize the number of nodes that are actively running containers from the microservices application with microservices architecture.
- Balanced Throughput Hosted Application scaling the number of containers that run each microservice from the microservices application based on the number HTTP request received at each endpoint and target request.

![Image](image.png)

**Figure 1.** Virtual private server vs container.

Virtual machines and containers differ in several ways, but the primary difference is that containers provide a way to virtualize an OS so that multiple workloads can run on a single OS instance. With VMs, the hardware is being virtualized to run multiple OS instances. Containers’ speed, agility, and portability make them yet another tool to help streamline software development.

One technology that is the current trend is Kubernetes. Kubernetes is an open source system function to automate the distribution and scaling container [3]. With this technology can help resolve the issue of distributed servers. Where all server, application, database will be accommodated in a container contained in Kubernetes. Kubernetes will managing all existing containers as needed.

Kubernetes is divided into master nodes and worker node has a different function. Master node on Kubernetes functions as a master controls the entire unit and cluster, managing workload and communication between systems [4]. While the worker node or known as a worker or minion, a machine where containers (workloads) used. The master node in Kubernetes functions as a master control the entire unit of the cluster, manage workload and communication between systems.

This research focuses on applying scalability to Kubernetes. Single container compare multi container which will answer the effect of implementation scalability of performance or container performance Kubernetes, comparison of response time, throughput, cpu usage and concurrent users from the implementation of scalability using regional code API for test data.

2. **Design system and research methods**

2.1. **System architecture Of Docker**

The Docker System Installation and Configuration will be implemented on the Rack Server. The Configuration of These RackServer System is Ram 8Gb, CPU Intel Xeon with virtualization IntelVTx. The base Operating System is Latest version Debian 10 and container will be run on both machine.

Docker is an Open Source tool designed to make it easier to create, deploy, and run application using containers. Containers allow developer to package All of the parts it needs, such as libraries and dependencies, and deploy it as one package. the developer can be sure that the managed application will run on any other Linux machine.
In the system design above the docker kubernetes will be installed on the Server Rack, in this docker kubernetes installed 3 container(multi container) containing the Debian OS, Nginx Web Server, Mysql and PHP as shown Figure 2. In this Docker Testing Client will request / run applications that are on the docker kubernetes virtual server.

2.2. Design system architechture container with Kubernetes

![Diagram of System Architecture with Kubernetes](image)

Figure 2. System architecture Container Kubernetes.

Figure 2 shows the system used in this Docker system on each server containing the debian 10. Topology explain have 3 container for server worker nginx 1, nginx 2 and database mysql. Client will request run applications to API Server have kube proxy. Kube proxy scaling on container on kubelet. Testing request API Server analysis using Postman.

2.3. Research methods

This research has been carried out in accordance with the Fishbone Diagram flow shown in Figure 3 below.

![Fishbone Diagram](image)

Figure 3. Research method implementation.
3. Results and discussions
This Section Discusses about the result of single container compare multi container. Testing performed on local network to view API Data From Region Code Government. To Access API Data, the client must access IP Server 10.10.10.2:8080 which then forwarded to virtual server in docker. Figure 4 shows a web Rest API data when testing is done to prove that virtual server is running properly.

![Figure 4. Kubernetes UI management container.](image)

Testing the Kubernetes and Docker system we use the webserver stress tool. web server stress tool is an application that is able to generate clients virtually, the number of clients and requests can be arranged as needed. QOS parameters used in this study are response time, request loss and throughput.

### 3.1. Single server container
Testing by generating several requests sent from the client to a single container without a docker container. After that a single container will send a response to the client. In this test, the author generates 50, 100, 150, 200, 250 clients by sending 1000 requests continuously as shown by Figure 5.

![Figure 5. Single server container.](image)

| The Number Of Client | The Number Of Request | Response Time (ms) | Request Loss (%) | Throughput (Kbps) | CPU(status %) |
|----------------------|-----------------------|--------------------|------------------|------------------|---------------|
| 50                   | 1000                  | 4.163              | 0                | 573.646          | 34.0          |
| 100                  | 1000                  | 5.075              | 0                | 495.1874         | 38.0          |
| 150                  | 1000                  | 5.933              | 0                | 482.0543         | 42.0          |
| 200                  | 1000                  | 5.133              | 0                | 43.435           | 53.2          |
| 250                  | 1000                  | 6.475              | 0                | 456.264          | 78.10         |
3.2. Multi server container

Testing by generating several requests sent from the client to a multi container without kubernetes some Pod. After that a multi container will send a response to the client. In this test, the author generates 50, 100, 150, 200, 250 clients by sending 1000 requests continuously as shown by Figure 6. Table 2 show the calculation results of the Virtualized Single Container and Virtualized Multi Container. From the results, Virtualized Multi Container doesn’t affect the performances on Responds Time and Throughputs. On the other hand, the CPU status become increasing performance 50%.

![Multi server container](image)

**Figure 6.** Multi server container.

**Table 2.** Virtualized multi container.

| The Number Of Client | The Number Of Request | Response Time (ms) | Request Loss (%) | Throughput (Kbps) | CPU(status %) |
|----------------------|-----------------------|--------------------|------------------|------------------|---------------|
| 50                   | 1000                  | 4.101              | 0                | 500.120          | 30.2          |
| 100                  | 1000                  | 4.575              | 0                | 503.8734         | 35.7          |
| 150                  | 1000                  | 4.800              | 0                | 550.1103         | 39.2          |
| 200                  | 1000                  | 5.125              | 0                | 570.3211         | 43.5          |
| 250                  | 1000                  | 5.312              | 0                | 589.3102         | 49.1          |

4. Comparison with relate works

**Table 3.** Comparison with relate works.

| Relate Works                          | Limitation                                                                                                                                  | Our System                                                                                           |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Implementation Container Kubernetes to Support Scalability | Non scalability database and server minimum, no delay occurs during the testing process take place.                                             | Tests on Our System Using API Data Codes and Information on Indonesian Government Administration Areas different pod database. |
| Docker Containers Versus Virtual Machine-Based Virtualization | System configuration of these systems is RAM 4GB, CPU Intel Xeon Quad core. The base operating system is Ubuntu 12.04 and containers will run on both machines. | Our System Uses Server Rack. The configuration of these systems is Intel Xeon with 8GB ram. The base operating system is Latest version Debian 10 Buster. |
5. Conclusions

Study on Network Management Systems by Using Docker Kubernetes has been presented. Single Container compare Multi container was done. Web service on each containers works well so that all web applications accessible to the client. With the implementation of scalability of containers, there is saving cpu pod usage when scalability is applied. Due to the distribution of containers scattered on each worker. From the calculation results of the Virtualized Single Container and Virtualized Multi Container, show that the Virtualized Multi Container doesn’t affect the performances on Responds Time and Throughputs. On the other hand, the CPU status become lower about 50%. Comparison of response time between single and multiple servers the server takes longer for multiple servers due to the time delay from making the container which is scalability. CPU ratio between single container and multi container, CPU usage is smaller multi container because master Kubernetes serves many containers in the pod.

Acknowledgments

The authors would like to thank State Polytechnic of Malang for supporting this research.

References
[1] Adyandana A, Pramukantoro E S and Bakhtiar F A 2017 Pengembangan Lightweight Data Analytic Pada Perangkat Edge *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer* e-ISSN 2548 964X
[2] Scheepers M J 2014 Virtualization and containerization of application infrastructure: A comparison *21st twente student conference on IT* vol 21 (Enschede, The Netherlands: University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science)
[3] Kubernetes 2018 “Kubernetes,” [Online] Retrieved from: https://kubernetes.io/
[4] Rice L and Hausenblas M 2018 *Kubernetes Security. How to Build and Operate Applications Securely on Kubernetes* (US: O'Reilly Media, Inc.)