

High Availability Server Using Raspberry Pi 4 Cluster and Docker Swarm

T. Yudi Hadiwandra¹, Feri Candra²

Department of Informatics Engineering, Universitas Riau^{1,2}

tyudihw@lecturer.unri.ac.id¹, feric@eng.unri.ac.id²

Article Info

Article history:

Received Nov 11, 2020

Revised Dec 1, 2020

Accepted July 6, 2021

Keyword:

Cluster Computing

Docker Swarm

High Availability

Raspberry Pi

Web Server

ABSTRACT

In the Industrial 4.0 era, almost all activities and transactions are carried out via the internet, which basically uses web technology. For this reason, it is absolutely necessary to have a high-performance web server infrastructure capable of serving all the activities and transactions required by users without any constraints. This research aims to design a high-performance (high availability) web server infrastructure with low cost (low cost) and energy efficiency. low power) using Cluster Computing technology on the Raspberry Pi Single Board Computing and Docker Container technology. The cluster system is built using five raspberry Pi type 4B modules as cluster nodes, and the Web server system is built using docker container virtualization technology. Meanwhile, cluster management uses Docker Swarm technology. Performance testing (Quality of Service) of the cluster system is done by simulating a number of loads (requests) and measuring the response of the system based on the parameters of Throughput and Delay (latency). The test results show that the Raspberry Pi Cluster system using Docker Swarm can be used to build a High Availability Server system that is able to handle very high requests that reach Throughput = 161,812,298 requests / sec with an Error rate = 0%.

© This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Corresponding Author:

T. Yudi Hadiwandra,

Department of Informatics Engineering

Universitas Riau,

Kampus Bina Widya Km. 12,5 Simpang Baru, Pekanbaru, Indonesia, 28293

Email: tyudihw@lecturer.unri.ac.id

1. INTRODUCTION

In the Industrial 4.0 era, almost all activities and transactions are carried out via the internet, which basically uses web technology. The high activity and transaction of requests for services from users can cause the web server to fail to serve the user needs. Therefore, it is absolutely necessary to have a high-performance web server infrastructure that capable to serving all the activities and transactions required by the user without any constraints. In the era of cloud computing, a distributed computing system is needed that can abstract the capabilities of the hardware in carrying out a computation process called Virtualization [1]. With virtualization, hardware resources that exist in a cloud computing service can share and run a variety of different application environments. There are two types of technology commonly used in hardware virtualization, namely Hypervisor and Container [2]. Efficient use of hardware resources, replication of processes for high availability, and

the demand for systems that are more tolerant of system errors drive the development of this virtualization system. In a study it was reported that the use of virtualization technology in a data center can reduce carbon emissions by about 30% compared to using physical infrastructure [3].

Virtualization technology is also growing following the abstraction needs of the computational process. Traditional virtualization techniques are considered to consume too much hardware resources to run computational processes, so the lighter container-based virtualization techniques can be an attractive option. The giant technology companies such as Microsoft, Google and Facebook also use container technology in their datacentre services [4]. Compared to the traditional approach that uses virtual machines as the basis for the development and deployment of applications running on a cloud-based infrastructure, container technology provides a higher level of portability and availability that enables developers to build and deploy their applications more widely in efficient and flexibility manner. [5].

Container technology is a modern virtualization technology that is gaining popularity. One of the most adopted container-based virtualization technologies is Docker [6]. Docker is an open source project which is an implementation of a very light weight operating system level virtualization technology. Docker was introduced in 2013, and is the industry standard for container technology. Containers are standard units of software that allowed developers to isolate their applications from their environment. Today, Docker is the de facto standard for building and sharing applications in containers ranging from the desktop to the cloud [7].

This study aims to design and build a web server infrastructure with high performance (high availability) at low cost (low cost) and energy efficient (low power) using the Raspberry Pi Single Board Computing and Docker Container technology. In this study, several tests were also conducted on the Quality of Service (QoS) of the system built to determine the system's ability to handle service requests from users.

2. RESEARCH METHOD

In this study, the cluster system was built using five nodes where each node will be implemented using the Raspberry Pi type 4B Single Board Computer (SBC) module. The Web server system in this study was built using virtualization technology which will be implemented using Docker containerization technology. This container system virtualization technology is different from the virtualization technology that uses a virtual machine. A virtual machine technology performs virtualization by emulating at the machine level while container technology emulates at the operating system level. Each node in the cluster will be integrated into a single system using Docker Swarm technology [8]. Docker Swarm acts as a manager in this cluster system and also has an internal load balance system which is used to manage the distribution of workloads to each worker node. With this load balance, the utilization of computational resources can be done efficiently and maximally. Load balance can also minimize system failure in serving user requests.

In this study also tested the performance of the cluster system based on the parameters of throughput and delay/latency. Throughput is a measure of the average number of successful deliveries in a measure of time. In general, the maximum throughput can indicate the network capacity (bandwidth) of the system. Delay or latency is the time it takes a data packet to travel from origin to destination. In implementation, the delay value is the length of time the packet takes from the original application to the destination application. The amount of delay can be caused by many things. One of the factors that influence is the determination of packet priority from the scheduling algorithm used.

2.1. Cluster Architecture Design

The architectural design of the cluster that is built can be seen as in Figure 1. The architecture of this cluster system uses five Raspberry Pi type 4B modules with CPU specifications using Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, 4GB RAM and a Gigabit ethernet port. For the switch we using the TP-Link TL-SG1008MP Gigabit 8-port. This design uses the standard

architecture recommendation of Docker Swarm [8] with 3 nodes as Managers and 2 nodes as Workers as shown in Table 1.

Each node is made into a Docker Machine by installing the Docker Engine application version 19.03.12 which runs on the 32-bit version of the Raspberry Pi OS Lite version of the operating system. To merge all nodes into one cluster, run the Docker Swarm application on all nodes and are given the appropriate roles, namely 3 nodes as Master and 2 nodes as Workers.

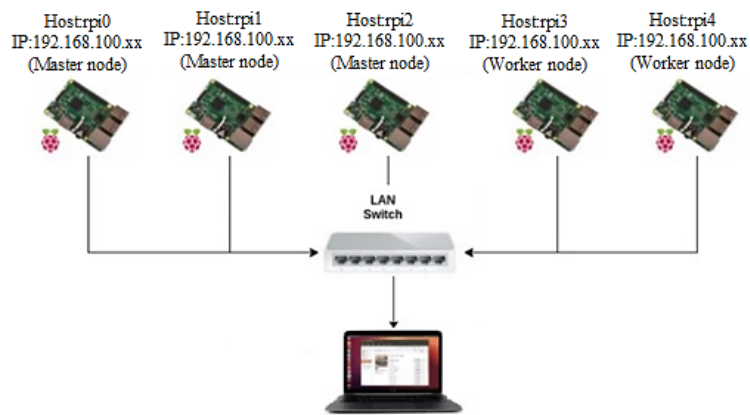


Fig. 1 Cluster architectural design

Table 1. Swarm manager fault tolerance

Swarm Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3
9	5	4

(source: www.docker.com)

2.2. System Performance

Performance of the system is measured based on throughput and delay/latency parameters of the response given by the system after being given load testing. The request load is simulated using Apache Jmeter application [9] version 5.3 which is run from the client system in the form of a desktop PC with intel pentium G640 processor with 2.8GHz, 6GB RAM and Ms.Windows 8.1 operating system version 64bit. For easy observation of cluster behavior is visually used Visualizer image container [10] which is run on the manager node of the cluster system as a web service that can be accessed through the browser in the client machine. Some of the test scenarios performed are as follows:

- Scenario 1: testing the reliability of system in handling system failures due to resource unavailability (failover). This test aims to observe and measure the availability level of the system by providing a condition where one or more machines are no longer available to be able to provide the services (single point of failure). And then try to access the Visualizer application service at the same address to see if the cluster system is still capable of running the Visualizer application service [10].
- Scenario 2: testing the reliability of the system in handling the process of adding and reducing the number of server nodes in anticipation of adjusting machine capacity needs (scalability). This

test aims to observe and measure the availability level of the system by looking at the scalability capabilities of the system by trying to duplicate system services. In this scenario, it will be observed how the system behaves when scaling both scale-up and scale-down. In this test we created a static website using Hypriot image container [11] as a web service that will be in scaling and viewed cluster behavior using Visualizer.

- c) Scenario 3: testing the reliability of system in handling overload with load balancing capabilities. This test aims to observe and measure the availability level of the system by providing a condition where the system will be burdened with a number of service requests as shown in Table 2. In this test is used Apache Jmeter application that will simulate a number of service request loads (requests) to the system and simultaneously measure the performance of throughput and delay/latency of the system.

Table 2. List of test parameters in Scenario 3

#Tes	#Container	#Request	#Tes	#Container	#Request
1	1	1000	11	5	1000
2	1	2500	12	5	2500
3	1	5000	13	5	5000
4	1	10000	14	5	10000
5	1	20000	15	5	20000
6	2	1000	16	10	1000
7	2	2500	17	10	2500
8	2	5000	18	10	5000
9	2	10000	19	10	10000
10	2	20000	20	10	20000

2.3. Docker

Docker is a software platform that packages the applications into a standard unit called a container that has everything the applications needs to work including libraries, system tools, code, and processing time. Container is one of the virtualization techniques at the operating system level where each process or application running each container will have the same kernel while virtualization at the machine level such as virtual machine requires a different operating system kernel per application running [12]. Docker has two licensing models: open source Docker Community Edition (CE) and subscription-based Docker Enterprise Edition (EE).

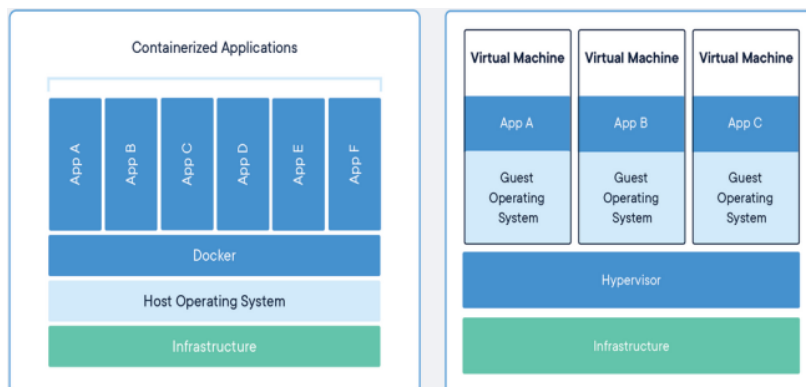


Fig. 2 Container and Virtual Machine Architecture [13]

Docker is an application based on open source technology that allows developers or anyone else to create, run, test and launch applications in a container. Docker quickly packages the applications with their components in an isolated container, allowing them to run on-premises

infrastructure without configuring containers[14]. Docker uses a client-server architecture. The Docker client contacts the Docker daemon, which performs the running job, and distributes the Docker container. Both docker client and daemon can run on the same system. Docker client and daemon communicate via sockets or via API provided by Docker[13]

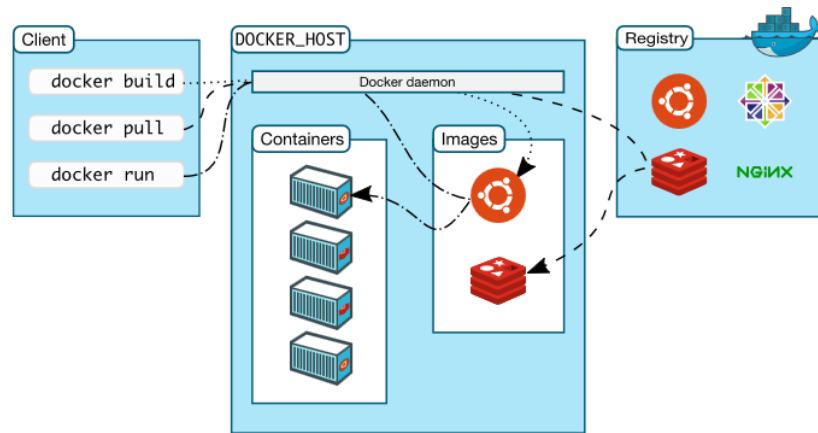


Fig. 3 Docker System Architecture Scheme [13]

2.4. Docker Swarm

Docker Swarm is a tool for managing clusters and containers that are already integrated in the Docker Engine, also known as Swarm-kits. Swarm-kit is a separate set of projects from the opensource community of developers. Cluster swarms are a collection of multiple Docker hosts that run swarm mode and there are acting as managers (who manage cluster members and manage delegate tasks) and some act as workers (who implement or process services (swarm services)). Any Docker host can act as a manager, worker, or both [15].

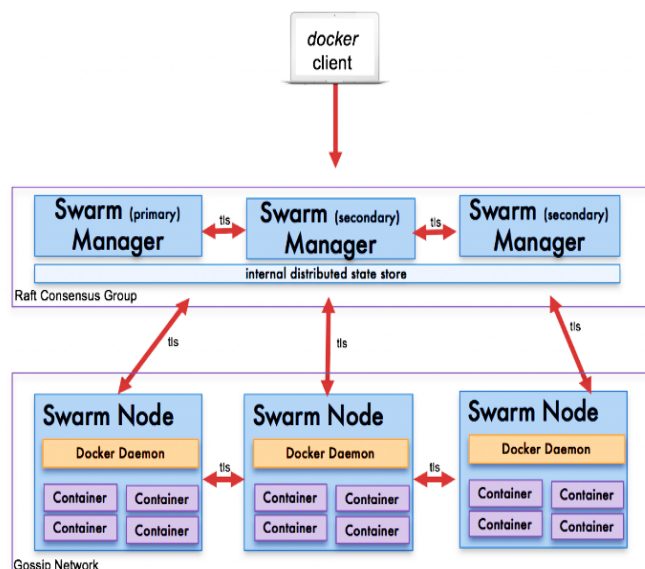


Fig. 4 Docker Swarm Architecture [16]

3. RESULTS AND ANALYSIS

In a test experiment with Scenario 1, the initial condition of the Visualizer service was run by the rpi2 server machine. Then we reboot the rpi2 server and try to access the Visualiver application

service at the same address. As seen in Figure 5. Visualizer application service can still run and this service has been moved automatically to rpi1 server. Thus the ability of fail over on this cluster system is able to run very quickly and without downtime. This capability is absolutely necessary in a high availability system

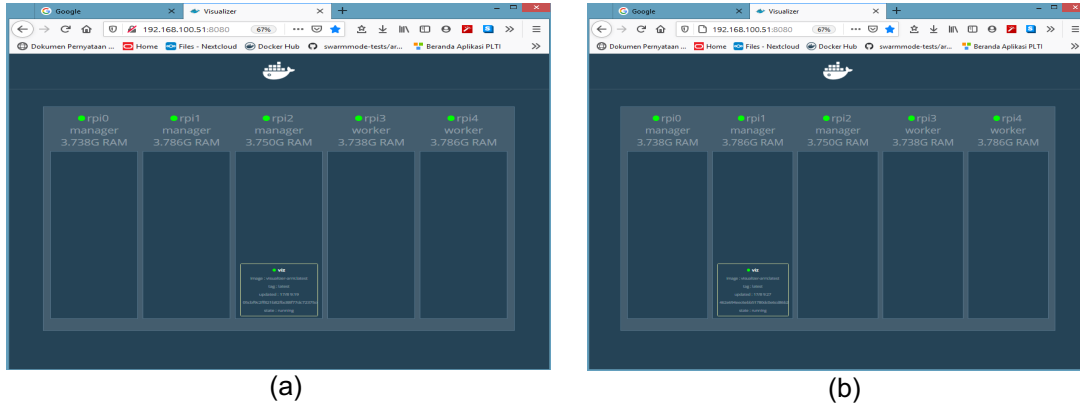


Fig. 5 Visualization cluster (a) initial condition (b) reboot condition rpi2

In test scenario 2, we observed and measured system availability level based on scalability capability of the system by trying to scale-up and scale-down the Hypriot web sever as a service running

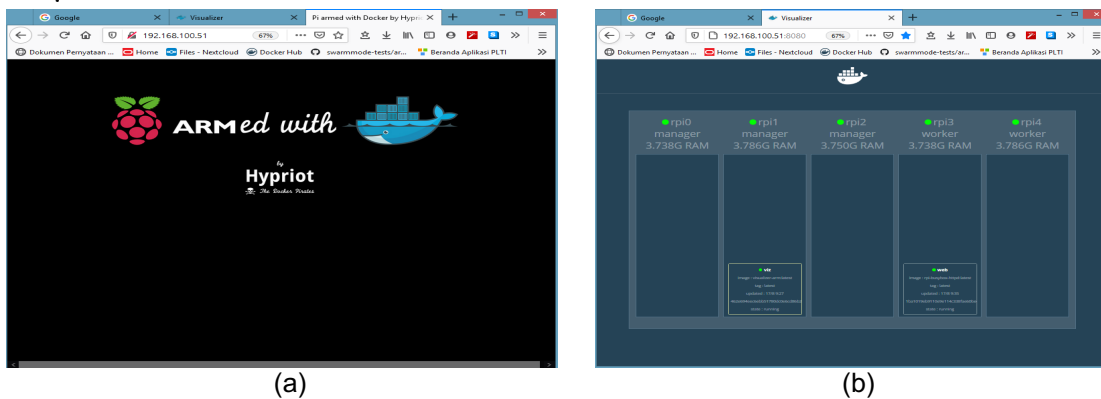


Fig. 6 (a) Hypriot web server (b) Hypriot container visualization at rpi3

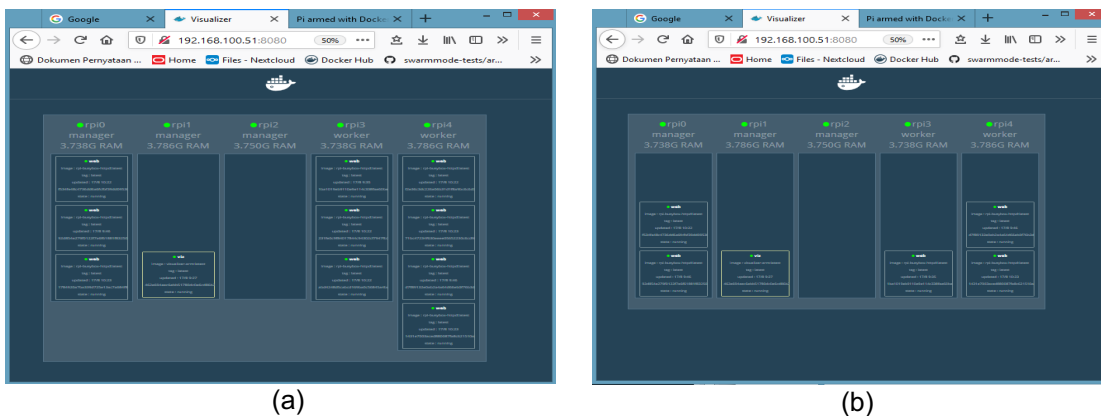


Fig. 7 Scaling container Hypriot (a) Scale-up to 10 (b) Scale-down to 5

From the test results, it can be seen that the system capability in the scale-up and scale-down process can run well and without downtime. This capability is absolutely necessary in a high

availability system when it comes to serving an instant increase in the number of high requests so that the system is not overloaded.

In scenario 3 testing, we observed and measured system Availability level by providing a number of users (samples) who perform service requests using the Apache Jmeter application that will simulate a certain amount of service request load to the system and simultaneously measure system performance

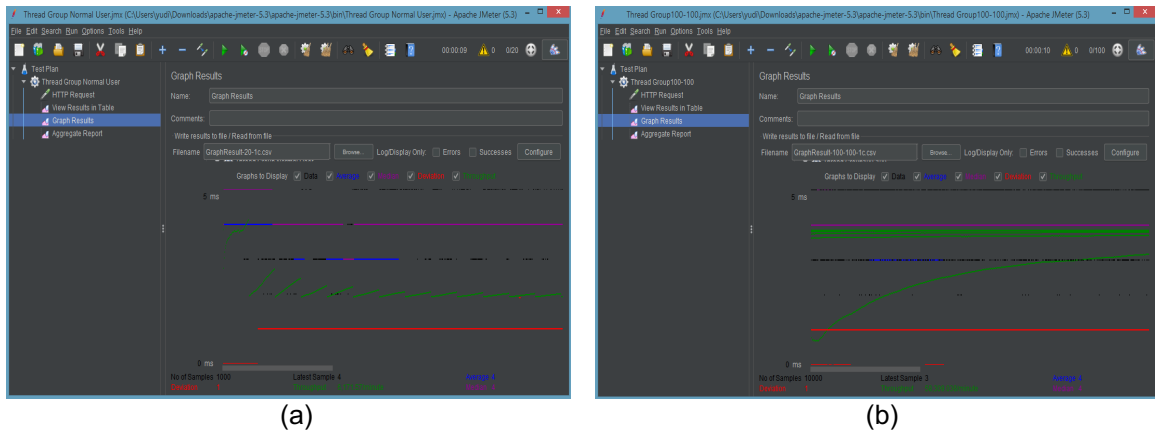


Fig. 8 JMeter Graph Result with 1 Container (a) 1,000 Samples (b) 10,000 Samples

Table 4. Throughput Test Results

# Samples	1 Container (request/ms)	2 Container (request/ms)	5 Container (request/ms)	10 Container (request/ms)
1000	10.285.949	10.233.320	10.267.995	10.466.820
2500	24.982.512	25.118.055	25.030.036	25.092.844
5000	49.677.099	49.426.651	49.407.115	49.436.425
10000	97.181.730	97.560.976	96.786.682	96.983.804
20000	137.014.455	161.812.298	150.060.024	146.487.951

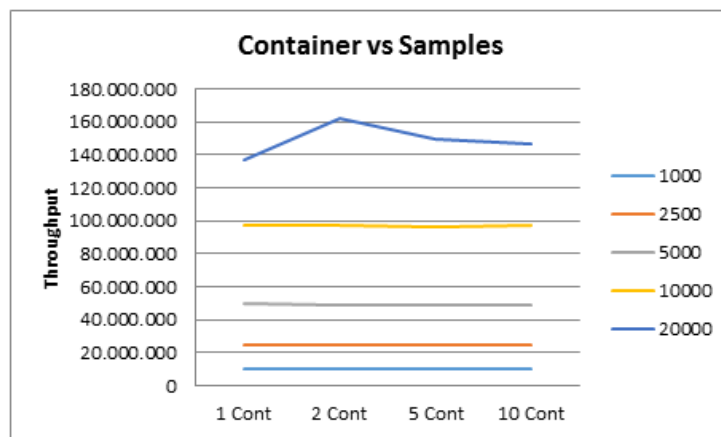


Fig. 9 Throughput Comparison by number of containers and samples

Table 5. Latency Test Results

# Samples	1 Container (ms/request)	2 Container (ms/request)	5 Container (ms/request)	10 Container (ms/request)
1000	4	4	4	4
2500	3	4	4	4
5000	3	4	4	5

10000	4	4	4	5
20000	18	13	17	21

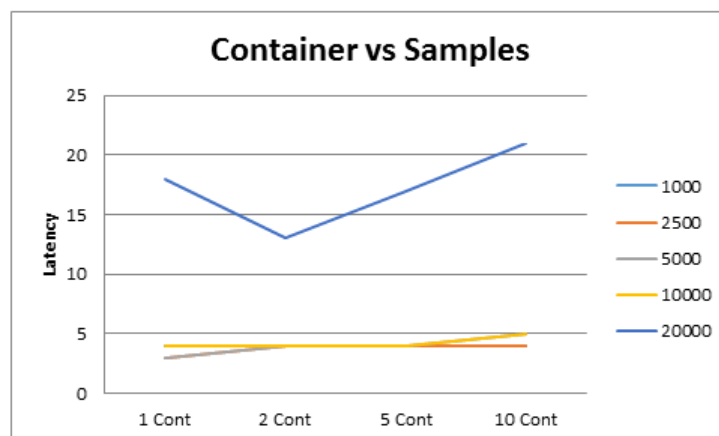


Fig. 10 Latency comparison based on number of containers and samples

4. CONCLUSION

From the results of this experiment it can be concluded that raspberry devices are able to run virtualization technology well using docker. Docker virtualization technology can also be used to create a computer cluster that supports high availability servers. Based on the test result data it appears that raspberry cluster devices using docker are able to handle requests without any constraints with error rate = 0%. The designed device is also capable of handling a very high number of requests until it reaches throughput = 161,812,298 requests/sec.

From the test data obtained, it is recommended to choose the maximum number of containers possible. Too many containers do not guarantee system reliability will improve. The more containers will consume more CPU and Memory resources

ACKNOWLEDGEMENTS



The author thanked the Institute of Research and Community Service of Riau University for providing financial support to this research.

REFERENCES

- [1] Wikipedia, "Virtualization" [Online]. Available: <https://en.wikipedia.org/wiki/Virtualization>. [Accessed: 11-Oct-2020].
- [2] V. G. da Silva, M. Kirikova, and G. Alksnis, "Containers for Virtualization: An Overview," *Appl. Comput. Syst.*, vol. 23, no. 1, pp. 21–27, 2018.
- [3] M. Pretorius, M. Ghassemian, and C. Ierotheou, "An investigation into energy efficiency of data centre virtualisation," in *Proceedings - International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2010*, 2010, pp. 157–163.
- [4] T. Gupta and A. Dwivedi, "Data storage & load balancing in cloud computing using container clustering," *Int. J. Eng. Sci. Res. Technol.*, vol. 6, no. 9, pp. 656–666, 2017.
- [5] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015, pp. 171–172.
- [6] H.-E. Yu and W. Huang, "Building a Virtual HPC Cluster with Auto Scaling by the Docker," Sep. 2015.
- [7] Docker, "Why Docker? | Docker," 2017. [Online]. Available: <https://www.docker.com/why->

- docker. [Accessed: 19-Oct-2020].
- [8] T. M. Mark Church, Marlon Ruiz, Andrew Seifert, “Docker - Docker Swarm Reference Architecture: Exploring Scalable, Portable Docker Container Networks,” 2019. [Online]. Available: <https://success.docker.com/article/networking#whatyouwilllearn>. [Accessed: 03-Aug-2020].
- [9] Apache, “Apache JMeter - Apache JMeter™,” *Apache Jm.*, p. 2019, 2014.
- [10] Miranda, et. al., “GitHub - dockersamples/docker-swarm-visualizer: A visualizer for Docker Swarm Mode using the Docker Remote API, Node.JS, and D3.” [Online]. Available: <https://github.com/dockersamples/docker-swarm-visualizer>. [Accessed: 29-Oct-2020].
- [11] Docker Team, “Getting started with Docker on your Raspberry Pi · Docker Pirates ARMed with explosive stuff.” [Online]. Available: <https://blog.hypriot.com/getting-started-with-docker-on-your-arm-device/>. [Accessed: 29-Oct-2020].
- [12] T. P. Kusuma, R. Munadi, and D. D. Sanjoyo, “Implementasi dan Analisis Computer Clustering System dengan Menggunakan Virtualisasi Docker,” *e-Proceeding Eng.*, vol. 4, no. 3, pp. 1–6, 2017.
- [13] Docker, “Docker overview | Docker Documentation,” *Docker.Com*, 2018. [Online]. Available: <https://docs.docker.com/get-started/overview/>. [Accessed: 07-Nov-2020].
- [14] S. Dwiyatno, E. Rakhmat, and O. Gustiawan, “Implementasi Virtualisasi Server Berbasis Docker Container,” *Prosisko*, vol. 7, no. 2, pp. 165–175, 2020.
- [15] Docker, “Swarm mode overview | Docker Documentation,” *Docker*, 2020. [Online]. Available: <https://docs.docker.com/engine/swarm/>. [Accessed: 07-Nov-2020].
- [16] IT Solution Architects, “Containers 102: Continuing the Journey from OS Virtualization to Workload Virtualization,” *medium.com*, 2017. [Online]. Available: <https://medium.com/@ITsolutions/containers-102-continuing-the-journey-from-os-virtualization-to-workload-virtualization-54fe5576969d>. [Accessed: 07-Nov-2020].

BIOGRAPHY OF AUTHORS

	<p>T. Yudi Hadiwandura obtained Bachelor Degree in Computer Engineering from Universitas Gunadarma in 1998, obtained Master Degree in Computer Science from Universitas Gadjah Mada in 2004. He has been a Lecturer with the Department of Informatics Engineering Universitas Riau Indonesia since 2018. His current research interests include artificial intelligent, internet of thing, machine learning and data science.</p>
	<p>Feri Candra obtained Bachelor Degree in Electrical Engineering from Institute Sains dan Teknologi Nasional in 1999, obtained Master Degree in Electrical Engineering from Universitas Indonesia in 2002, and obtained Doctoral of Electrical Engineering from Universiti Teknologi Malaysia in 2017. he has been a Lecturer with the Department of Informatics Engineering, Universitas Riau Indonesia since 2002. His current research interests include signal processing, artificial intelligent, machine learning and data science.</p>