Implementation of Dynamic Web Server Based on Operating System-Level Virtualization using Docker Stack

Heru Nurwarsito
Faculty of Computer Science
University of Brawijaya
Malang, Indonesia
heru@ub.ac.id

Verio Brika Sejahtera
Faculty of Computer Science
University of Brawijaya
Malang, Indonesia
vrobias@student.ub.ac.id

Abstract*—A webserver is a provider that receives and provides data-based services that are sent through a web browser, then given a response in the form of web pages and HTML documents. The main problem that often occurs on a web server is the level of availability, demand affects the performance of the webserver which gets a lot of loads so that the webserver experiences overload, it will cause the web to go down. The Docker stack allows multiple services to connect to each other and run on multiple connected machines. The solution to this is by using operating system-level virtualization that utilizes the cluster system provided by the docker swarm, the docker stack is used to run multi-container images and web server services that will be distributed to other virtual servers. The results of this research show that the operating system-level virtualization supports the failover and load balancing mechanisms in the docker swarm area and the docker stack plays a role in distributing web server services from node manager to node worker as a backup webserver service, testing the best average response time parameters on failover obtained on a node with the time obtained 18.90 milliseconds while in availability through the parameter request test, at 200 requests the system successfully received 80% requests and indicated that the system could not meet the nines criteria at levels of availability.

***Keywords*—Web server, operating system-level virtualization, docker stack, failover, availability**

I. Introduction

The rapid development of technology today creates and fosters many innovations, especially in the use of the internet network in the present. The growing times and technological advances of many large companies offer convenience for internet users to access information. It has been observed that the level of use of the internet as a medium for providing the information is increasing, causing many companies to allocate funds for website development and to provide strong and reliable web server services in managing data. The infrastructure used in managing these data is a server [1]. The more people who access a website, the more workload on a service provider called a web server will be less optimal [2].

The main problem that occurs on a web server is the availability side. If the webserver only provides a few static web pages such as simple information, the static web can be applied, if necessary, to improve security systems or interact with users in a real-time, dynamic web can be applied in this condition [3]. With the increasing number of requests from users for information in a short time and at the same time, there will also be a spike in requests for access to the website on the webserver. The request affects the performance of the webserver which is getting a lot of loads so that it cannot handle the request when the webserver is overloaded it will cause the web to go down. Disruptions that occur on the web server will cause the website to be accessed cannot be displayed causing some services to also experience downtime, events like this will be detrimental to companies that provide web server services on the availability side.

To support the webserver to work optimally with real-time usage and a high level of availability, it is necessary to have a load balancing mechanism and a web server failover in the cluster swarm area. load balancing is a technique for distributing traffic loads on two or more connection lines in a balanced manner so that traffic can run optimally, maximize throughput, minimize response time and avoid overloads on one of the connection lines [4]. Failover is a backup operational mode in which a service such as software or hardware from a node fails and is then diverted to another node in the system that is still available. This mechanism can run automatically and replace services that experience failures as before [5].

The failover mechanism is carried out in the operating system-level virtualization in the swarm cluster area using the docker swarm. Operating system-level virtualization is a server-virtualization method that shares operating system kernels to share and provide user workspace instances which can be said to be containers provided by Docker. Because this unique feature of container virtualization is often referred to as operating system-level virtualization, the host operating system in a container apart from being able to share the kernel can also eliminate overhead and also provide isolation between applications, the container feature allows to send small containers containing the full operating system that encapsulates and run only the necessary files [6]. The implementation of operating-system-level virtualization can migrate directly but it is also able to balance dynamic container loads between nodes in the cluster [7]. In the cluster swarm service area, the webserver will be deployed by the docker stack and distributed to worker nodes that have been registered in the cluster system. The Docker swarm will do its job by running a load balancing mechanism to distribute the load that is executed to several worker nodes that have been managed by the node manager.

The Docker stack itself is a shared and interconnected service that can be managed and scaled together. The use of the stack can define and coordinate the functionality of the entire system, as well as applications, although some very complex applications make it possible to use multiple stacks. Stack files are written into scripts using the YAML format, the format used in docker-compose, which defines one or more executable services. When novice docker users want to build the services they write YAML files, compose templates, design their service components and use them in docker but barely describe their services. Some group items are not clearly defined but are complicated and difficult to understand [8]. In making the docker-compose script, all processes that have been written and services to be run will be processed using the docker stack deploy. With docker stack deploy there is only one stack service running on a single host. The purpose of implementing the docker stack is to allow services to connect and run on multiple connected machines using only one executable command, namely the docker stack deploy.

Research on load balancing on a web server is examining traffic and balancing the load of requests on a web server that implements a load balancing algorithm in a docker swarm. The method tested by the researcher is to compare the performance of the load balancing algorithm that uses the least connection and also the round-robin to find the throughput value. The researcher also tests the load balancer to find out which algorithm has a good performance. The test results show that the throughput value of the least connection algorithm is better than the round-robin algorithm, while the results of the load balancer testing of the two algorithm methods used to support high availability and can run well when the failover mechanism is executed [9].

Research on autoscaling and failover mechanisms on a server cluster to run a web server, namely researchers testing the failover process that focuses on node failure and service failure, while the autoscaling process focuses on the scale-up and scale down, the goal of the test is to get an average time. -the average of both processes. The results of the tests conducted by researchers on failover testing showed that service failure was faster than node failure from the average time obtained, while in autoscaling the scale-up process was faster than the scale down process [10].

Based on the description of the problem and the description of the method or technology to be used that has been presented in the previous paragraphs. The researcher plans to implement a dynamic webserver service using the open-source Apache as the webserver and MySQL as a database created for the virtual server. Implementation is done using a virtual server created on a virtual machine to support the use of operating-system-level virtualization, which will take advantage of the cluster system provided by the docker swarm, while the docker stack implementation is used to run multi-container images and web server services that will be deployed. headed to another virtual server. In this research using 4 virtual servers made for one node manager and three worker nodes, the node manager will be the center in implementing the docker service image creation from the web server and deploying services to the worker node, the worker node is tasked with running the webserver service provided. node manager. The test parameters used to measure the performance of the webserver service include failure, downtime, and request.

The implementation of the docker stack that utilizes operating system-level virtualization in the swarm cluster area can distribute and manage containerized dynamic web server services to have better performance in distributing web server services to each node and the performance of the cluster system supports the use. Failover and availability mechanisms to increase the availability or success of the system is running the webserver service.

II. Basic Theory

A. *Web Server*

Web server is server software or it can be hardware dedicated to running the software, which can fulfill client requests. In general, a web server can contain one or more websites.

The Web Server has the main function of storing and processing files and providing web page requests to clients through a communication protocol between the client and the server that occurs using the Hypertext Transfer Protocol (HTTP), the page submitted is a Hypertext Markup Language (HTML) document. Several web servers are used and one of the most common is the Apache webserver which is the most widely used web server because Apache can be said to be an HTTP web server and has support for access control, PHP, and SSL. According to Abdullah, Web servers are usually referred to as HTTP servers, because their bases use the HTTP protocol [11].

B. *Operating System-Level Virtualization*

Operating system-level virtualization is a server-virtualization method in which the kernel of an operating system allows for multiple instances of isolated user-space which are sometimes called software containers or containers. Operating system-level virtualization is generally used in virtual hosting environments, where it is useful to isolate limited hardware resources from large numbers of distrustful users. System administrators can also use it, to a lesser extent, to consolidate server hardware by moving services on separate hosts to containers on a single server. The implementation of operating-system-level virtualization can migrate directly, but it is also able to balance the dynamic load of containers between nodes in the cluster [7].

C. *Docker Stack*

A stack is a collection of services that make up the application in a particular environment. Stack files use the YAML format, similar to docker-compose.yml, which defines one or more services. Stack has automatic support for building in a docker container, using volume mount, and switching user IDs. A stack is an easy way to use multiple services that connect to each other automatically, without needing to define each service separately. The stack file defines the environment variables, the deployment tag, the number of containers, and the specifically associated environment configurations. Therefore, it must use separate stack files for development, staging, production, and other environments.

A service can have a set or replica of images running in a container and jobs running on the container. Compose launches all the containers and runs the task and then forgets about it. A stack can track the state of containers even after they have been launched. This means if a container that correlates with the service goes down it will launch another one in its place at its discretion. Applications that run will be guarded by a swarm. Built with High Availability and load balancing [12].

III. Proposed Method

The system design describes the relationship and communication flow between the five components, namely

Node1 as a manager, Node2, Node3, and Node4 as workers, and docker volume as a place for sharing data. The communication flow between components, namely from Node1, which acts as a Manager, will run the docker swarm init command, which is to provide a token to the worker, then node2, node3, and node4 which act as workers will run the docker swarm join command so that node2, node3, node4 can be connected to it. node1. The following is the system design flowchart in Fig 1:

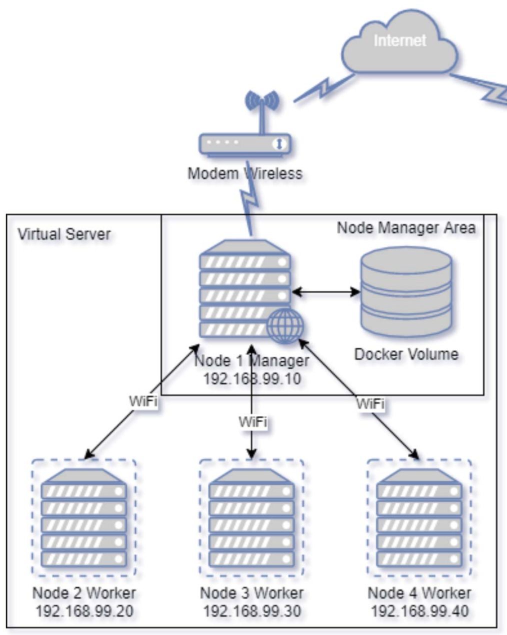

Fig. 1. Design System

Fig 1 shows the system design concept in this research. The process is carried out with several components, namely Node1 which acts as a manager, Node2, Node3, and Node3 which acts as a worker. The supporting component in this system is the docker volume which is outside the service swarm area which acts as a file-sharing or directory sharing a place to the desired docker container. This sharing method is used to share disks by first mounting the disk on the Docker server. For each component on the system to run well, it requires a stable internet connection network so that each component can be properly connected. To be able to run a web server, it is necessary to create an image that is built using the Dockerfile. The Dockerfile contains commands to install the tools needed to support running the webserver. Some of the tools needed support are Apache2, MySql, and PHP (Hypertext Processor) in the form of web HTML source code.

A. *Docker Image Design*

The docker image design describes the process flow in creating the docker image which is used as the parent of several containers that have the required tools or applications installed. This Docker image refers to the contents of the FROM directive in the Dockerfile. The following is the flowchart for the docker image design in Fig 2.

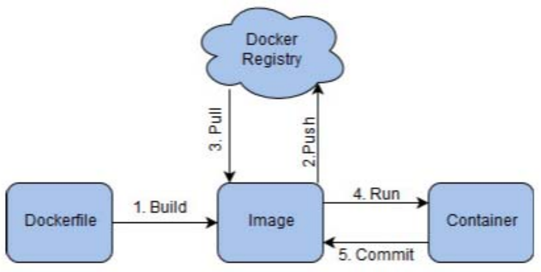

Fig. 2. Docker Image Design

Fig 2 shows the docker image design concept in this research. The following is a detailed explanation.

1. The initial process is to create a dockerfile which contains source code or instructions for creating a docker image, the dockerfile can contain what tools will be installed. After the dockerfile is created, the next step is to build, this process changes the contents of the dockerfile into an image containing tools or applications and is stored in the container to be used.
2. After the image is created, the next process is to push the image into the Docker registry, by doing a push, the image that has been created is stored in the Docker registry.
3. The pull process is used to retrieve the image that has been uploaded to the docker registry so that it can be used.
4. The Run process is used to enter the image that will be run and also functions to activate the container in the image.
5. The commit process is carried out if there is a change or update on the container side, starting from adding tools, updating tools, and so on.

B. *Docker Stack Design*

The docker stack design describes the process flow that occurs in the docker stack, namely by running services and images that have been created in a multi-container manner. The following design will be applied in this research as shown in Fig 3:

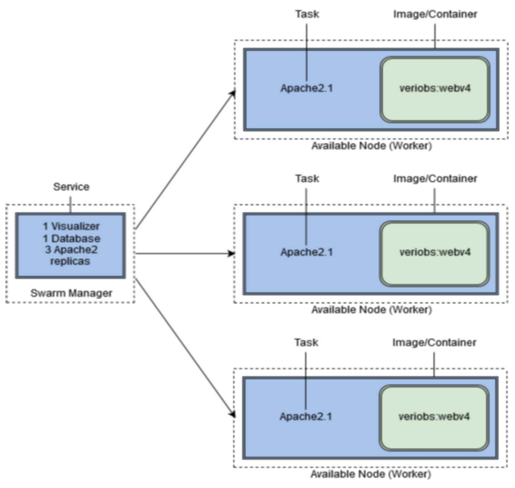

Fig. 3. Docker Stack Design

Fig. 3 shows the docker stack design concept in this research. The following is a detailed explanation:

1. The initial process carried out on the node manager side will determine what services and images will be created and run later. All configuration and implementation to support the running and working of the docker stack are done in the node manager. The image that was created in the previous process will be applied to docker-compose, in docker-compose, what services will run and determine where the service is placed.
2. After determining what contents in the docker-compose service to run, the node manager will execute the command to run the service and image in a multi-container manner. The service that is run will be distributed to all worker nodes, the worker node will run the task, and the image or container that has been specified in docker-compose.

IV. Result and Discussion

Functional testing is carried out to find out whether the webserver is functioning properly and as it should. There will be 2 functional tests carried out, namely web server functionality and functional failure, by checking each function based on a predetermined scenario and finding the appropriate results and writing the results in the validity statement table.

Another test that is performed is the failover testing, this test is done to measure and find out what the lowest average response time is on the webserver service by using load testing. This failover testing scenario uses the help of web server stress tools. This webserver stress tool will provide a test in the form of load testing where the test will calculate the response time of the webserver service when a load balancing mechanism is applied to the cluster system area to share the traffic load and a failover mechanism for the backup process if the network or web server conditions fail.

Furthermore, the test carried out is the availability test. This availability test will be tested to measure the level of availability and the percentage of success of a web server service system that is built in the swarm cluster area that can handle a high number of load requests. Testing will be carried out using Apache JMeter, in this test a scenario will be carried out where the node manager will be given several load requests as many as 50 requests, 100 requests, 150 requests, and 200 requests.

Failover testing is carried out to measure performance and find out how much time from the response time parameter test is needed to perform a web server failover service mechanism on each node in 5 attempts. In addition, it is also to determine the comparison of the time obtained between nodes. The process is carried out with supporting applications, namely webserver stress tools and selecting the time for constant load based on a specific time. Run the test with 1 minute, the number of users is 3 and the delay interval is 12 seconds.

A. Results and Analysis of Failover Testing

Based on the whole test trial of response time parameters that have been carried out in the load testing failover test in the cluster area. Tests carried out as many as 5 trials, 10 trials, 15 trials, and 20 trials, all the response times obtained will be made into a table as a reference for each trial carried out. To make it easier to make decisions to draw the results of the analysis of the whole trial, from all trials the average response time of the trials that have been carried out will be taken, the average time of each trial will be as in Table II.

TABLE I. The Results of Failover Testing

Trials	Node2 (ms)	Node3 (ms)	Node4 (ms)
5	27,40	25,20	23,80
10	24,90	28,20	22,80
15	25,07	24,40	27,73
20	18,90	18,25	18,05

Based on Table I, which shows the results of the average response time from failover testing in the webserver cluster area of each worker node. The results of the test get the average time of every 5 trials, 10 trials, 15 trials, and 20 trials, all the results of the average time will be implemented into a graph as in Fig 4.

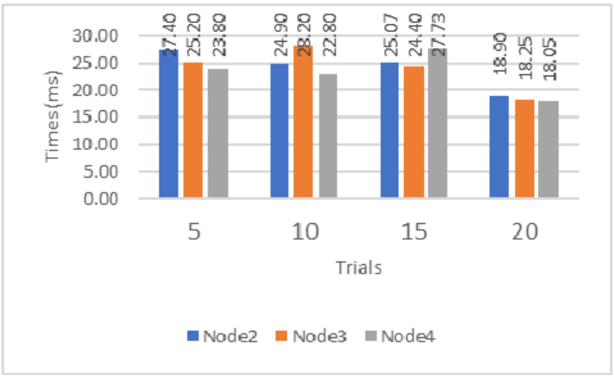

Fig. 4. Graph of Failover Test Results

The graph in Fig 4 shows the results of the average response time of all the trials that have been carried out. In the results of 5 trials the lowest average response time was obtained at node4 with a time of 23.80 milliseconds, the results of 10 trials the lowest average time obtained was at node4 with a time of 22.80 milliseconds, on 15 trials the lowest average time was obtained at node3 with a time of 24.40 milliseconds and at 20 trials the lowest average time was obtained at node4 with a time of 18.05 milliseconds, but this time cannot be said to be valid because of the 20 trials that have been carried out, at the time of testing simulation node3 and node4 cannot complete the last test because time has run out while processing the test, then the valid value obtained in 20 trials is obtained at node2 with a time of 18.90 milliseconds. The time results obtained are very varied, there is a difference in the difference in the average time at each node because of the influence of the load testing process that occurs, the factor that affects the difference in the average time difference is the delay time interval given in each number of trials, besides that during testing. Internet traffic used has increased during the load testing process so that it affects the results obtained in each trial.

In failover testing, it is known that the average response time parameter test of all trials that have been carried out, the lowest average response time parameter test was obtained at node2 with the time obtained 18.90 milliseconds, the time gain is influenced by the delay interval that has been determined during running load testing webserver service for 1 minute until the process is stopped by the system.

B. Results and Analysis of Availability Testing

Systems that run web server services will be given 50, 100, 150, and 200 HTTP requests. Testing will be carried out on the node manager which controls all services at the worker node, by accessing the node manager's IP address, namely 192.168.99.10 and PATH / QQ which is the initial barrier to the webserver service that will be invaded by the user.

TABLE II. THE RESULTS OF AVAILABILITY TESTING

Number of Requests	**Request Successful (%)**	**Request Error (%)**	**Throughput (s)**
50 *requests*	100%	0%	23.6
100 *requests*	100%	0%	32.9
150 *requests*	98.67%	1.33%	35.7
200 *requests*	80.00%	20.00%	40.4

Based on Table II which shows the results of load requests with HTTP request requests from availability testing in all trials to test the webserver service, finally, the data is obtained in the form of a percentage of successful requests, request errors and throughput values obtained, then these numbers will be implemented into a graph such as in Fig 5 and Fig 6.

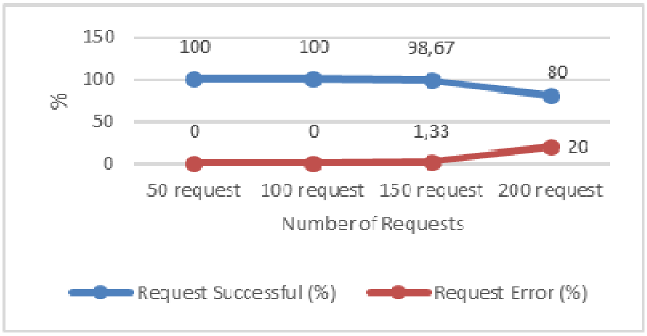

Fig. 5. Graph of Availability Testing Request Results

Referring to the graph in Fig 5, the results obtained from testing the request parameters successfully explain that in the trial by receiving a load request of 50 requests and 100 requests, no errors were found, which indicates that the system is at 100% success rate and is still able to handle the service. The web server was invaded by 50 requests and 100 requests, on the 150 request trial the success rate was found to be 98.67% and an error occurred of 1.33% due to the high number of requests and the system began to be a little overwhelmed, while in the 200 request the system began to be overloaded and experienced a failure in the process of receiving requests, the success rate obtained was 80% and the error value obtained was 20%, the percentage obtained from the 200 request trial was in the sufficient success category in terms of system availability.

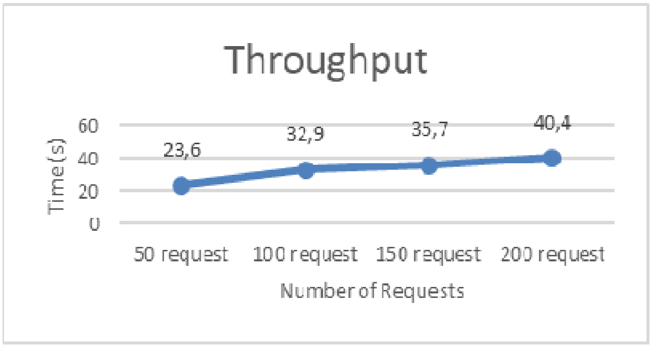

Fig. 6. Graph of Availability Testing Throughput Results

In Fig 6, a graph shows the increase in the throughput value obtained from each trial with the number of requests given. In 50 requests the throughput value gets 23.6 seconds, at 100 requests the throughput value increases to 32.9 seconds, the next trial with 150 requests the throughput value increases to 35.7 seconds and at 200 requests the throughput value also increases. In the range of 40.4 seconds, the increase in throughput value is due to the increased load on the server during the loan request process, so the consumption and internet traffic will also increase. The throughput value is calculated from the beginning of the sample until the last sample is sent, several factors make the throughput increase apart from the number of requests given, namely from the type of data sent and also the computer specifications used, but in that sense the greater the throughput value, the better. Even though the availability test with 200 requests has a fairly high error rate, on the other hand, the throughput value has a high time. Factors that influence the results of this availability test are the large number of requests requested. Besides, network quality and internet traffic conditions also affect the results obtained.

In the availability test, the request parameter test was successful at 200 requests, the system successfully received a request by 80% and indicated that the system could not meet the criteria for the nines levels of availability, but on the side of the throughput value of the 200 requests trial got a high throughput value at 40.4 seconds. Referring to the level of availability, the term nines refers to a high level of availability in the system. The success rate of the system can be said to have high readiness when the success rate is at 90% and it is included in the Nines criteria at levels of availability.

V. CONCLUSION

Based on research on the Implementation of a Dynamic Web Server Based on Operating System-Level Virtualization Using the Docker Stack, several conclusions as follows:

1. Implementation of a dynamic web server with a docker stack in the scope of testing using a virtual server as a media operating system-level virtualization, containerized management between servers-virtualization. The image web server can run the Apache open source like a web server base and process the PHP source code to become an HTML in the form of a dynamic web page that is integrated with an image database for services in data storage, grouping data and identifying data. Operating system-level virtualization supports failover and load balancing mechanisms in the

docker swarm area and the docker stack plays a role in distributing web server services from the node manager to the worker node as a backup web server service if one of the nodes fails.

2. Dynamic web server performance based on predefined test parameters. In the failover test, the best average response time parameter is obtained on a node with the time obtained 18.90 milliseconds, the time gain is influenced by the predetermined delay interval during load testing of the webserver service for 1 minute until the process is stopped by the system. In the availability test, the request parameter test was successful, at 200 requests the system successfully received requests by 80% and indicated that the system could not meet the Nines criteria at the levels of availability, but on the side of the throughput value of the 200 requests trial got a good throughput value at 40, 4 seconds. The success rate of the system can be said to have high readiness when the success rate is at 90% and it is included in the Nines criteria at levels of availability.

REFERENCES

[1] Megan. G. H., 2010. *Design and Implementation of Server Virtualization in Thiess Contractors Indonesia.*

[2] Rosalia, M., 2016. *Implementation of High Availability Server Using Load Balancing and Failover Method on Virtual Web Server Cluster.* Vol.3, No.3 December 2016

[3] Yakun. L., & Xiaodong. C., 2010. *Design and implementation of embedded Web server based on arm and Linux.*

[4] Dewobroto, P., 2009. *Load Balance menggunakan Metode PCC.*

[5] Jayaswal, K*., 2005. Administering Data Centers: Servers, Storage, And Voice Over Ip.*

[6] Srinath, R, M., 2018. *Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications*. St. Cloud State University.

[7] Teimouri, D., 2017. *Operating-System-Level Virtualization.*

[8] Klinbua, K., & Vatanawood, W., 2017. *Translating TOSCA into docker-compose YAML file using ANTLR.*

[9] Afis, D., Data, M., & Yahya, W., 2019. *Load Balancing Server Web Berdasarkan Jumlah Koneksi Klien Pada Docker Swarm*. vol. 3, no. 1, p. 925-930. ISSN 2548-964X.

[10] Yosua, S., Data, M., & Siregar, R., 2018. *Implementasi Failover Dan Autoscaling Kontainer Web Server Nginx Pada Docker Menggunakan Kubernetes.* vol. 2, no. 12, p. 6849-6854. ISSN 2548-964X.

[11] Abdullah, A., Simamora, S. N. M. P. & Andrian, H. R., 2010. *Implementasi dan Analisa Load-Balancing pada suatu Web-Server Lokal*

[12] Jeeva, S., Vinod S and Pethuru R., 2017. *Learning Docker Second Edition*.