Jurnal Ilmiah Teknik Elektro Komputer dan Informatika (JITEKI)
Vol. 9, No. 4, December 2023, pp. 1024-1034
ISSN: 2338-3070, DOI: 10.26555/jiteki.v9i4.27220

Comparative Analysis of Apache 2 Performance in Docker Containers vs Native Environment

Rizky Bintang Saputra, Subektiningsih
Informatics Department, Faculty of Computer Science, Universitas Amikom Yogyakarta, Yogyakarta, 55283, Indonesia

ARTICLE INFO

Article history:

Received September 15. 2023
Revised October 31, 2021
Published November 07, 2023

Keywords:

Performance;
Web server;
Docker containers;
Native environment;
Linux

ABSTRACT

Web servers have become crucial to facilitate access to and distribute such content on the internet. In this case, Docker containerization technology offers a solution. Docker allows developers to package applications and dependencies in one container, making deploying web servers faster and easier. But with these features, is there any performance that must be sacrificed if we choose to use docker in our web server deployment process. We will look at how much performance will be sacrificed. However, we must thoroughly analyze how Apache2 performs when running in a Docker container compared to running natively. That's why we're conducting a study to compare the performance of Apache2 in a Docker container versus a native environment using experimental methods. For this study, we'll use the Apache bench tool to test Apache2's performance in both environments. By experimenting, it should become clear how the performance of Docker containers compares to native environments when developing web servers. The research shows that Apache2 performance on native hosts is about 5-10% better than in a docker environment in handling small request loads. The better performance here refers to the parameters we tested: total time results, requests per second, and transfer speed. The request load variation can differ depending on the server specification itself. Although Docker offers features in terms of application isolation and scalability, our results show that running Apache2 natively is more efficient without changing its default configuration. The additional overhead Docker can be required to run the docker system in isolating the application; in this case, the virtualization layer is required to run Apache2 inside a Docker container. This can affect application performance and cause a slight performance degradation compared to using the host operating system directly. This research aims to inform developers about the performance difference between apache2 in Docker and the native environment. It will help them make informed decisions about deployment environments. Docker offers appealing features, but its performance may need to improve. Test results show that the native host performs better, although its feature set is not as extensive as that of Docker.

Corresponding Author:

Subektiningsih, Universitas Amikom Yogyakarta, Yogyakarta, 55283, Indonesia
Email: subektiningsih@amikom.ac.id

1. INTRODUCTION

In recent decades, the Internet has advanced significantly [1], [2]. The rapid expansion of web services is one of the most obvious tendencies in this technological progress. To support increasingly complex web server requirements, a new technology solution is needed to simplify the creation of web servers. Almost every sector and industry has benefited from developments in this field, from commercial applications to consumer services [3]–[5]. Web services have become a crucial component in linking, integrating, and optimizing many elements of corporate activities and services in a time when information and communication technologies are constantly growing [6]–[9]. Because of this, we need to enhance end-user experiences, such as regarding performance,

latency, and bandwidth throughput. Cloud-based web applications must frequently be intelligently deployed to some geographic areas [10]–[12]. Due to the increasing demand, these programs have a multi-tier architecture, making it challenging to create, debug, deploy, and upgrade [13], [14].

To meet increasingly complex web server requirements, a new technology solution is needed to simplify the creation of web servers [15], [16]. Web content, such as web pages, media, databases and other resources, is stored, processed, and sent to users by a web server, a software program, or a hardware device [17]–[19]. The most relevant answer is containerization, because using system resources is more manageable for its use in cloud computing and to avoid incompatibility issues that often occur when using the virtualization [20]–[22]. A Docker container is a lightweight, standalone, and executable software package that encapsulates an application and its dependencies, libraries, and configuration files [23]. A container within a virtualized environment provides a consistent and isolated runtime environment for applications to run seamlessly across different computing environments. Docker was selected as the containerization application in this study to stand in for several different containerization platforms due to its simplicity of use, largest forum, and current documentation [24]–[27].

This approach facilitates portability, scalability, and efficient resource utilization, making Docker containers a pivotal tool in modern software development and deployment workflows [28], [29]. Developers can deploy web server programs and their dependencies in a container normally operated in various environments by using the solution provided by the Docker containerization [30], [31]. Docker provides functionality like scheduling containers, monitoring health checks, failover, volume container/volume binding, etc [31]. However, it is still necessary to conduct an in-depth analysis to compare the performance between using a web server in a Docker container and a native environment. Native environment refers to the original or inherent setting, context, or habitat in which a particular organism, system, or entity naturally exists and operates [32]–[34]. Understanding the native environment is essential to understanding the nature and interaction of the subject within the scope of work [35], [36]. We need to assess whether the advantages of docker are equivalent to the performance generated compared to the native environment. In this test, we need a benchmark tool to capture data to compare the performance of the two systems. ApacheBench, commonly known as "ab," is a command-line tool developed by the Apache Software Foundation. It is designed to perform performance testing and benchmark web servers or web applications [37]. Apachebench will perform tests by generating load on the web server; these tests will assess the server's ability to handle heavy traffic and determine its performance limits [38]. The data results are crucial feedback for evaluating and enhancing web-based system performance [39], [40].

Previous research on web server performance by comparing LXD containers with Docker containers resulted in an experiment that Docker is superior in receiving large packets while LXD excels in transfer rate and time per request; both have advantages and disadvantages [41]. It can be seen in this research that both container systems have their own advantages, which must be underlined that both systems require overhead space to containerize. This makes the author interested in comparing systems that require overhead (here author chose Docker) compared to native hosts. Research [20], the author compared the performance of LXC and Docker and weighed the advantages and disadvantages of each; in the study, there was a comparison experiment regarding network latency whose results were superior to LXC. Research [42], the author conducted a performance evaluation between Native, Docker, LXC, and LXD on multiple services. This research attempts to evaluate the overall system performance, including file system management, Central Processing Unit (CPU) performance, and Random Acces Memory (RAM) speed in containers, through a series of benchmarks. The tested services ranged from web servers, FTP servers, mail servers, etc. Research [43], the author describes the workings and application of multiple containers on a machine. In [44], the author explains the concept of load balance on a web server and collects related research that discusses it. The author created a table to present the research details he collected. Research [45], the author analyzes and compares the performance of applications between hypervisor-based virtual machines and Docker containers. The results of this study show that docker is far superior in resource usage and performance than virtual machines. Research [46], the authors investigate the performance of using cloud-native frameworks, specifically Docker and Kubernetes, from a resource management perspective. This research aims to monitor worker node load on docker and Kubernetes platforms.

Based on the description above, this research compares the performance of Apache2 from two environments, namely Docker containers and native environments, with experimental methods. The author chose Docker over other containers because Docker is rising in trend, is easy to share images, has high compatibility (can be run on Windows, Linux, etc.), and is lightweight compared to other container systems [47]. By comparing it between native and docker, we can know the extent of the performance difference with the features offered by docker. This research will be carried out on 3 Virtual Machines (VM) with similar specs and operating systems (OS), each representing the Docker container system, native host, and system for the

testing tool (apachebench). The operating system used in this test is Debian. Debian was chosen because it is a common Linux distro with many derivatives. Debian is also one of the most widely used Linux OS in servers [48]. Testing is done in stages, starting from the docker container VM, followed by the Native host VM. The range of requests tested in this study is 1000-50,000, and each test result will be averaged and then calculated into data. The parameters taken from the test are total time results, requests per second, and transfer rates. The author selected total time outcomes, requests per second, and transfer rates as test parameters because they represent the overall performance of the two systems, providing valuable insights into how applications react to various workloads [20]. The results of this research are expected to provide a better view of the performance comparison of the two types of systems to determine which system is effective for deploying web services or other services. In addition, the research also hopes that this research can be an initial reference for building servers for the long term; by determining which environment is chosen from the start, expenses can be saved. And finally, readers can focus on where to configure after seeing the data from this research.

2. METHODS

In this study, the authors used an experimental methodology in conducting their tests, which referred to the research [41]. In this section, the evaluation performance of apache2 using Docker container and native environment is performed using benchmarking tools. The following benchmarking tools used for the performance evaluation is apachebench [37]. The steps of this procedure are a review of the literature, design, simulation, and data analysis. Fig. 1 depicts these steps.

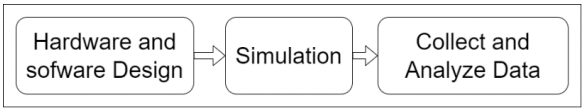

Fig. 1. The Flow of The Experimental Method of Research

2.1. Hardware and software design

2.1.1. Hardware

The methodology was to create 3 virtual machines with the same specifications to represent identical hardware configurations. The Apache2 server was run natively on the first system, while a Docker container was used to run the server on the second machine. Using the Apachebench benchmark program, machine 3 is utilized to test both machines. Apachebench was chosen because this tool is open-source, free, and is commonly used for debugging or testing the performance of a web server. Tests using machine 3 are carried out alternately, starting from the first machine. Stress tests are carried out by sending 1000 – 50.000 data requests. Each shipment is sent 100 data packages. The parameters tested are the total time result, request per second, and transfer rates. This study wants to determine which system is better from the parameters taken. The base hardware specifications used to run the virtual machine in this study are listed in Table 1. The scenario of using 3 virtual machines can be seen in Fig. 2. The specifications of each VM can be seen in Table 2. We chose this spec configuration because we took the reference that this specification is the initial spec in buying a Virtual Private Server (VPS), with this specifications when we rent it to vendors, the price can still be said to be affordable. Another reason for choosing this specifications is because of the limitations of the author's computer specifications in performing the test.

Table 1. Hardware Host Specifications

Processor	RAM	SSD	VGA
Core I5 7500 4 core 3,8Ghz	16GB DDR4 2400Mhz	512 GB	RX 580 4GB

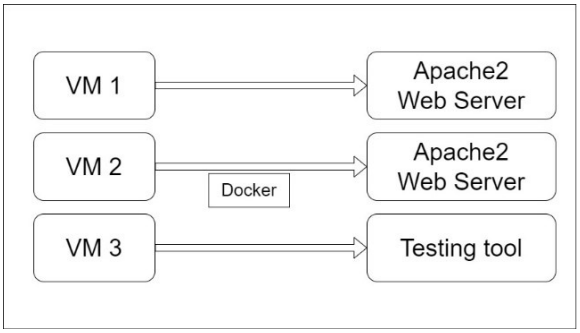

Fig. 2. Scenario VM Deployment

Table 2. Virtual Machine Specifications

Name	Processor	RAM	Storage(Virtual)	OS
VM 1 (Docker)	2 Core	2048 MB	20 GB	Debian
VM 2 (Native)	2 Core	2048 MB	20 GB	Debian
VM 3 (Tool)	2 Core	2048 MB	20 GB	Debian

2.1.2. Software

After the VMs have all been created, the next steps to prepare the program in this test are first, all VMs are installed with Linux Debian 11. This version of Debian was chosen to avoid bugs or errors if you choose a newer version of Debian and make it easier when troubleshooting. The reason for choosing Debian is because the Linux OS is often used in servers, the Linux product with the most users, and the most documentation on the internet [48].

A different approach was taken in the case of VM 1, which represents the original host environment. Apache2, an open-source web server software, was directly installed on the original host without modifying the default settings. This decision aims to mimic conventional deployment scenarios often encountered in the native environment and compare performance without any changes from the initial installation. By maintaining the default configuration, we sought to emphasize the comparative analysis between Dockerized deployments and the original host, emphasizing out-of-the-box performance.

For VM 2, a Docker environment, selecting Docker images is important in shaping the software design. To achieve consistency and build a controlled environment, Docker images are sourced from the official Docker Hub repository for Apache HTTP Server (httpd) with the latest version (at the time of testing version 2.4.58-bookworm). This ensures that the configuration and software settings in the Docker container align with established best practices for deploying Apache HTTP Server instances. For VM 3, this machine only installed the benchmark tool (apachebench). Which will only be used to benchmark VM 1 and VM 2. I hope that a clean system will not interfere with benchmarking results.

2.2. Simulation

All configurations were kept at their default settings for the entire testing process, ensuring a standardized device performance evaluation. The testing procedure included sequentially sending packets of various sizes, ranging from 1000 to 50,000 requests per test, with each test transmitting 100 individual data packets. This range was selected to comprehensively assess the device's response to varying workloads; each request represents the number of users [42]. Tests are alternated between VM 1 and VM 2 to allow the Docker container environment system and the native host to perform under different package parameter scenarios. Total time results, requests per second, and transfer rates, essential performance metrics, are recorded for both VMs during each testing iteration. By consistently utilizing default configurations, we aimed to remove bias and guarantee an impartial comparison between Docker-based and native host environments. This method allows for an objective evaluation of software deployment methods and performance differences. The gathered metrics were subsequently analyzed to uncover any variations in performance between the two environments across various packet parameter scenarios. Details of the simulation flow can be seen in Fig. 3.

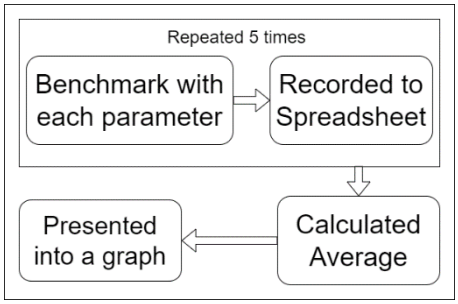

Fig. 3. The Flow of Performing Simulation

2.3. Collect and analyze data

2.3.1. Collect

The following process is the process of collecting data. The sequence of steps is presented as follows:

1. Turn on VM 1 (Docker) and VM 3 (Apachebench).
2. Commands VM 3 to benchmark VM 1 (starting from 1000 requests).
3. Benchmark is successful (the result is as shown in Fig. 4)
4. We take the data we will observe: the total time result, requests per second, and transfer rate.

5. We take the data and enter it into Excel
6. Repeat steps 2-5 up to 5 times.
7. After the process is done 5 times, continue by changing the number of requests in step 2 (starting from 1000,5000,10000,25000,40000,50000 requests)
8. Perform steps 2-7
9. Do this until all requests are completed
10. Perform the same steps for VM 2 by turning off VM 1 and turning on VM 2.
11. Repeat steps 2-9

2.3.2. Analyze Data

After each iteration of the test with certain parameters generated has been carefully documented in an Excel spreadsheet. In this test, 5 repetitions were conducted to make the data more consistent and reduce the error of external factors. After all the data is collected in the Excel table, the average is calculated using the average formula. The data that has been averaged is then made into a visual graph. After the visual graph is visible, we can observe and analyze how the performance trends and variations are observed between the Docker-based environment (VM 1) and the native host (VM 2) under different package parameter scenarios. The visual representation helps to get a quick and intuitive understanding of the comparative performance results. The analyzed results are continued in the results and discussion section.

3. RESULTS AND DISCUSSION

This section presents and discusses the experiments' results, focusing on three key performance metrics: total time results, requests per second, and transfer rates. Each metric is addressed individually to provide a comprehensive understanding of the observed differences between the Docker-based (VM 1) and native host environment(VM 2) deployment environments. Each test ranges from 1000 to 50,000 requests for static content and the protocol used Http, with each test covering the transmission of 100 individual data packets. Since there are many factors involved in this test, and to avoid inconsistent results, each test for each parameter is run 5 times, and then the average is calculated and considered representative of the test results. The results graph is displayed in a line format to illustrate the results in general better.

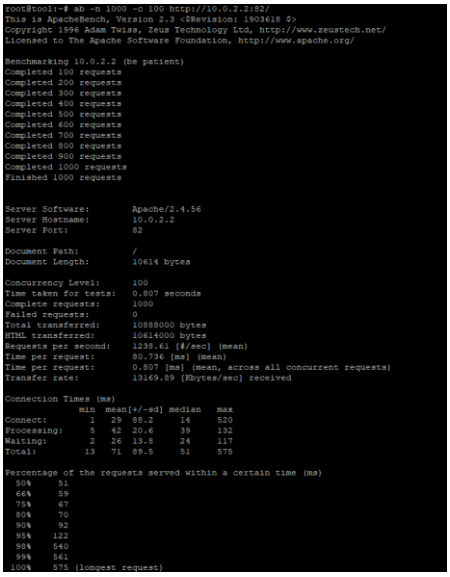

Fig. 4. Example of Benchmark Results

The Fig. 4 shows Apache Bench Tool's benchmark results, which give valuable information about the time necessary for each stage of the request-response cycle. Every request is measured by Apache Bench to evaluate the minimum, average, standard deviation (std), median, and maximum values for each stage, all in milliseconds. The initial step is called "Connect" and checks the time Apache Bench takes to set up a TCP connection with the target server before sending a request through that connection (ctime). The second stage, called "Processing," measures the duration the connection remains open post its establishment (time - ctime). Lastly, "Waiting" measures the time interval during which Apache Bench waits after sending a request before beginning to read the responses from the connection (wait-time). Finally, "Total" measures how long it takes

to establish and close a connection during the Apache Bench test. This metric offers a clearer insight into the overall latency of the request-response cycle, unlike "Time taken for tests" and "Time per request" which are only based on individual object data. This metric offers a clearer insight into the overall latency of the request-response cycle, unlike "Time taken for tests" and "Time per request" which are only based on individual object data. Furthermore, "Total" reveals which stages are responsible for the latency. This metric offers a clearer insight into the overall latency of the request-response cycle, unlike "Time taken for tests" and "Time per request" which are only based on individual object data.

The example above shows that the "Connect" stage takes the longest time on average in the cycle, both in mean and median values, although it has the highest standard deviation. We can investigate further to determine which side is responsible for this variation because the "Connect" metric relies on both client- and server-side latency. Additionally, we can focus our optimization efforts on enhancing how our backend handles new TCP connections. The most recent report from Apache Bench includes a chart that breaks down request wait times by percent. This better explains how wait times are spread out compared to the standard deviation in the Connection Times chart. The chart highlights the distribution of request wait times for different percentages, with "certain time" referring to the time property value [49].

3.1. Total Time Result

Total time results are the cumulative time to process requests or tasks in a specific test scenario. In our study, we use this metric to gain insight into request processing efficiency within the Docker-based (VM 1) and native host (VM 2) deployment environments. A lower total time indicates quicker processing, whereas a higher total time suggests potential delays in request execution. By comparing the total time results for different request parameter scenarios across the two environments, we can understand how each environment manages workloads and executes requests. As seen from the data in Fig. 5, the total time difference between the Docker-based environment and the original host increases as the request parameters increase. However, it can be seen that the increase in total time result increases most drastically when more than 10000 requests are executed, this can be due to the resources starting to approach their limits making the resources overloaded which results in a significant increase. This can be different when the hardware tested has higher specifications. Other things that can affect this are packet queues, throttling, or performance degradation that prevent overloading. However, this difference remains relatively small between Docker and the native host as the difference when averaged is only 7% and also considering the overhead of the Docker system. This suggests that for this specific range of workloads, the choice of deployment environment does not significantly impact the total time results.

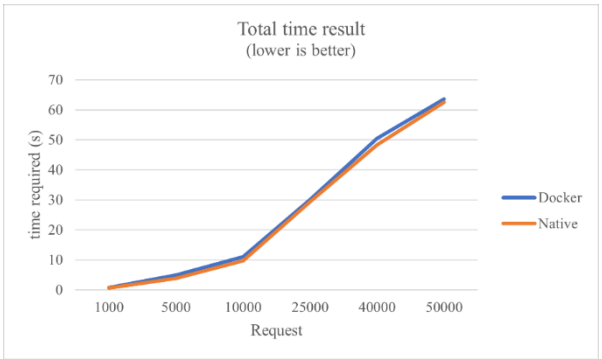

Fig. 5. Total Time Result

3.2. Requests Per Second

Requests per second (RPS) represent the rate at which a system or server can handle incoming requests in a given time frame. In our analysis, RPS reflects the responsiveness and throughput capabilities of the Docker-based (VM 1) and native host (VM 2) environments. A higher RPS value indicates better responsiveness and the ability to handle a larger volume of requests concurrently. Based on the data in Fig. 6, it is clear that the native host environment performs better than the Docker-based environment when the packet load is lower because it has less overhead and more direct access to hardware resources, which makes it more efficient. Specifically, it can be seen that the result of requests per second has started to decrease when it starts to touch 10000 requests; this happens when the resource is approaching its limit, then at 25000-50000, it is seen that the results are practically consistent which indicates that the resource has reached its capacity. This happens because the more requests that must be processed, the more resources must be shared between those requests. This includes CPU time, memory allocation, network bandwidth, and other resources. When resources are divided too thinly, there may be queuing of requests, increased response times, or even failures

in request fulfillment. So, the selection becomes smaller as the requirements increase. This may indicate that deployment decisions significantly affect applications with smaller requirements.

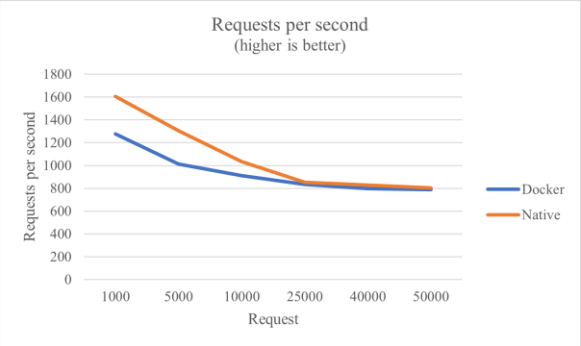

Fig. 6. Requests Per Second

3.3. Transfer Rate

Transfer rates refer to the speed at which data is transmitted between servers or clients. This metric provides insights into data transfer efficiency within the Docker-based (VM 1) and native host (VM 2) environments. A higher transfer rate signifies faster data transmission, while a lower rate indicates potential bottlenecks in data movement. The data in Fig. 7 illustrates that when handling a smaller number of requests, the performance difference between the Docker-based environment and the native host is considerable. Just like the previous parameter, it can be seen that the decline in transfer rates performance only starts to decline when it starts to touch 10000 requests; this happens when the resource is close to its capabilities, then at 25000-50000, the results are practically flat which indicates that the resource can no longer handle requests. When it reaches the resource limit, the performance tends to be the same. As the request load increases, the performance gap decreases, indicating that both systems equally prioritize success in handling incoming requests, resulting in response delays that result in lower transfer rates. This certainly makes the choice to deploy significantly impact the application when the transfer load is still low. Test data results from each parameter shown in Table 3.

Fig. 7. Transfer Rates

Table 3. Test Data Results From Each Parameter

Request	Parameter					
	Total time result (ms)		Request per second (#/s)		Transfer Rates (Kbytes/s)	
	Docker	Native	Docker	Native	Docker	Native
1000	0.7866	0.623	1276.6	1604.6	13545.2	17062
5000	4.9492	3.8468	1013.2	1304.2	10752.4	13872.2
10000	11.0168	9.6862	909.8	1033	9650.8	10983.4
25000	29.9692	29.3346	834.4	853	8851	9067.6
40000	50.4146	48.2412	796.8	829.4	8452.8	8819
50000	63.5478	62.538	786.8	802.4	8347.2	8530.6

The results of the 3 parameter data above reveal an interesting relationship in each scenario. Looking at the total time result parameter, it can be seen that both have the same result trend. This means that the difference in the system does not affect the overall request completion. Moving on to the requests per second and transfer

rates parameters, the authors found that the difference between Docker and the native host becomes more prominent when the workload is relatively small. This indicates that the native host has an advantage in handling a low number of requests because the native host does not need to pass through a process again (Docker engine), so the process is processed faster, which can be seen in Fig. 8. Our results show that these three performance parameters are interrelated and impact the overall performance of the application. Understanding the data results of these three parameters can help make better decisions in selecting a deployment environment for a particular application, especially in the face of different workload requirements.

In summary, the number of requests (request packets) affects the entire performance dynamics of the system. The higher the number of requests, the more resources are required, and this can affect the total time, requests per second, and transfer rate. Knowing how these parameters are related is important for planning and optimizing system performance, especially when facing fluctuating workloads or high request rates. For example, to improve performance, it may be necessary to expand system resources such as CPU or memory or increase network capacity to support higher requests per second.

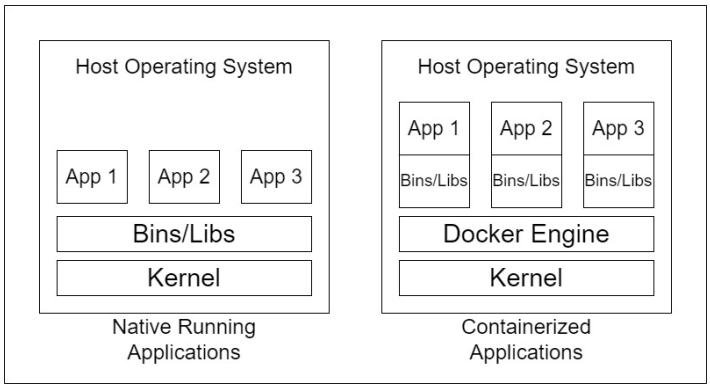

Fig. 8. Comparison Native Environment vs Docker Container

Fig. 8 compares the application architecture arrangement on the native host with the docker container. In a native environment, applications run directly on the host operating system, with full access to physical resources like CPU, RAM, and other hardware. This describes the traditional approach where applications are installed and run directly on the physical machine [32], [50]. As for Docker systems, the application runs in a Docker engine that contains all its dependencies. This container is isolated from the host operating system, but kernel performance and physical resources are shared with the host [25]. Docker allows for strong application isolation and portability across different environments because it is built into the engine. This figure illustrates how a modern application approach is more efficient and manageable. From this, the application on the native host environment can run faster because fewer processes are passed. Future research could focus on understanding the benefits and challenges of using containers in a production environment. To continually improve performance and efficiency, research into the automation of container management will also be an important topic. By understanding and addressing these issues, we can ensure that container technology continues to be a more powerful and efficient tool to support future applications and services.

4. CONCLUSION

It can be concluded that tests conducted using three virtual machines with various container architectures (Docker, native host, and testing machine) show different performance patterns depending on the workload level. At light workload levels, the native host performs better than Docker. However, as the workload increases, both Docker and the native host show similar performance, with the difference getting smaller. The results of this study provide a better understanding of how the choice of deployment architecture, i.e., Docker or native host, should strongly consider the workload characteristics and performance needs. Depending on the specific use case, one architecture may offer better advantages. While these results provide interesting insights, this research has limitations in testing, such as the limited number of requests and parameters. Therefore, future research can investigate the factors that affect the performance difference between Docker and the native host at various workload levels. In addition, additional configuration and optimization methods should be considered to identify ways to maximize performance in both environments. Future research could also focus on specific types of applications or workloads that may exhibit more significant performance differences between Docker and the native host. With a deeper understanding of these dynamics, we can make better decisions when choosing the most appropriate application architecture for the unique needs of an application or system. The authors hope that future research can further investigate the factors that affect the performance

differences between Docker and native hosts at different workload levels and with other benchmark parameters. In addition, it is also necessary to consider additional optimization and configuration methods to identify ways to maximize performance in both environments. Future research could also focus on specific types of applications or workloads that may show more significant performance differences between Docker and native hosts.

REFERENCES

[1] C. Wu, Y. Zhao, and S. Wan, "Design and Implementation of a Distributed Container-Deployed Web Business Traffic Generator," in *the 3rd International Conference on Computing and Big Data*, pp. 13–18, 2020, https://doi.org/10.1145/3418688.3418691.

[2] R. R. Zebari, S. R. M. Zeebaree, and K. Jacksi, "Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers," in *International Conference on Advanced Science and Engineering (ICOASE)*, pp. 156–161, 2018, https://doi.org/10.1109/ICOASE.2018.8548783.

[3] R. Zese and E. Bellodi, "A web application for reasoning on probabilistic description logics knowledge bases," *Softw Pract Exp*, vol. 53, no. 9, pp. 1741–1762, 2023, https://doi.org/10.1002/spe.3212.

[4] A. Huf and F. Siqueira, "Composition of heterogeneous web services: A systematic review," *Journal of Network and Computer Applications*, vol. 143, pp. 89–110, 2019, https://doi.org/10.1016/j.jnca.2019.06.008.

[5] Y.-C. Lu *et al.*, "Service deployment and scheduling for improving performance of composite cloud services," *Computers & Electrical Engineering*, vol. 74, pp. 616–634, 2019, https://doi.org/10.1016/j.compeleceng.2018.07.018.

[6] X. Zhao, R. Li, and X. Zuo, "Advances on QoS-aware web service selection and composition with nature-inspired computing," *CAAI Transactions on Intelligence Technology*, vol. 4, no. 3, pp. 159–174, 2019, https://doi.org/10.1049/trit.2019.0018.

[7] R. K. Ibrahim, S. R. M. Zeebaree, and K. F. S. Jacksi, "Survey on Semantic Similarity Based on Document Clustering," *Adv. sci. technol. eng. syst. j.*, vol. 4, no. 5, pp. 115–122, 2019, https://doi.org/10.25046/aj040515.

[8] K. Jacksi, S. R., and N. Dimililer, "LOD Explorer: Presenting the Web of Data," *Ijacsa*, vol. 9, no. 1, 2018, https://doi.org/10.14569/IJACSA.2018.090107.

[9] F. Filipe, I. M. Pires, and A. J. Gouveia, "Why Web Accessibility Is Important for Your Institution," *Procedia Computer Science*, vol. 219, pp. 20–27, 2023, https://doi.org/10.1016/j.procs.2023.01.259.

[10] Y. Aldwyan, R. O. Sinnott, and G. T. Jayaputera, "Elastic deployment of container clusters across geographically distributed cloud data centers for web applications," *Concurrency Computat Pract Exper*, vol. 33, no. 21, 2021, https://doi.org/10.1002/cpe.6436.

[11] H. Nurwarsito and M. Fadhil, "Implementation of Dynamic Web Server Cluster Based on Operating System-Level Virtualization using Docker Swarm," in *10th Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS)*, pp. 217–221, 2020, https://doi.org/10.1109/EECCIS49483.2020.9263428.

[12] H. Nurwarsito and V. B. Sejahtera, "Implementation of Dynamic Web Server Based on Operating System-Level Virtualization using Docker Stack," in *12th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp. 33–38, 2020, https://doi.org/10.1109/ICITEE49829.2020.9271710.

[13] M. Curiel and A. Pont, "Workload Generators for Web-Based Systems: Characteristics, Current Status, and Challenges," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 2, pp. 1526–1546, 2018, https://doi.org/10.1109/COMST.2018.2798641.

[14] A. M. Mejías, R. Bellas, J. E. Pardo, and E. Paz, "Traceability management systems and capacity building as new approaches for improving sustainability in the fashion multi-tier supply chain," *International Journal of Production Economics*, vol. 217, pp. 143–158, 2019, https://doi.org/10.1016/j.ijpe.2019.03.022.

[15] D. N. Jha, M. Nee, Z. Wen, A. Zomaya, and R. Ranjan, "SmartDBO: Smart Docker Benchmarking Orchestrator for Web-application," in *The World Wide Web Conference*, pp. 3555–3559, 2019, https://doi.org/10.1145/3308558.3314137.

[16] Y.-C. Tsao, Y.-K. Chen, S.-H. Chiu, J.-C. Lu, and T.-L. Vu, "An innovative demand forecasting approach for the server industry," *Technovation*, vol. 110, p. 102371, 2022, https://doi.org/10.1016/j.technovation.2021.102371.

[17] K. Jacksi and S. M. Abass, "Development History Of The World Wide Web," *Int. J. Sci. Technol. Res*, vol. 8, no. 9, 2019, https://www.ijstr.org/final-print/sep2019/Development-History-Of-The-World-Wide-Web.pdf.

[18] N. Moniz and L. Torgo, "A review on web content popularity prediction: Issues and open challenges," *Online Social Networks and Media*, vol. 12, pp. 1–20, 2019, https://doi.org/10.1016/j.osnem.2019.05.002.

[19] J. Zhou, J. Wei, and B. Xu, "Customer segmentation by web content mining," *Journal of Retailing and Consumer Services*, vol. 61, p. 102588, 2021, https://doi.org/10.1016/j.jretconser.2021.102588.

[20] M. Moravcik, P. Segec, M. Kontsek, J. Uramova, and J. Papan, "Comparison of LXC and Docker Technologies," in *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 481–486, 2020, https://doi.org/10.1109/ICETA51985.2020.9379212.

[21] A. Kropp and R. Torre, "Docker: containerize your application," in *Computing in Communication Networks*, Elsevier, pp. 231–244, 2020, https://doi.org/10.1016/B978-0-12-820488-7.00026-8.

[22] P. Jain, Y. Munjal, J. Gera, and P. Gupta, "Performance Analysis of Various Server Hosting Techniques," *Procedia Computer Science*, vol. 173, pp. 70–77, 2020, https://doi.org/10.1016/j.procs.2020.06.010.

[23] I. Merelli, F. Fornari, F. Tordini, D. D'Agostino, M. Aldinucci, and D. Cesini, "Exploiting Docker containers over Grid computing for a comprehensive study of chromatin conformation in different cell types," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 116–127, 2019, https://doi.org/10.1016/j.jpdc.2019.08.002.
[24] E. Casalicchio, "Container Orchestration: A Survey," in *Systems Modeling: Methodologies and Tools*, pp. 221–235, 2019, https://doi.org/10.1007/978-3-319-92378-9_14.
[25] I. M. A. Jawarneh *et al.*, "Container Orchestration Engines: A Thorough Functional and Performance Comparison," in *ICC IEEE International Conference on Communications (ICC)*, pp. 1–6, 2019, https://doi.org/10.1109/ICC.2019.8762053.
[26] N. Zhou, Y. Georgiou, L. Zhong, H. Zhou, and M. Pospieszny, "Container Orchestration on HPC Systems," in *IEEE 13th International Conference on Cloud Computing (CLOUD)*, pp. 34–36, 2020, https://doi.org/10.1109/CLOUD49709.2020.00017.
[27] O. Flauzac, F. Mauhourat, and F. Nolot, "A review of native container security for running applications," *Procedia Computer Science*, vol. 175, pp. 157–164, 2020, https://doi.org/10.1016/j.procs.2020.07.025.
[28] M. De Benedictis and A. Lioy, "Integrity verification of Docker containers for a lightweight cloud environment," *Future Generation Computer Systems*, vol. 97, pp. 236–246, 2019, https://doi.org/10.1016/j.future.2019.02.026.
[29] A. K. Yadav, M. L. Garg, and Ritika, "Docker Containers Versus Virtual Machine-Based Virtualization," in *Emerging Technologies in Data Mining and Information Security*, vol. 814, pp. 141–150., 2019, https://doi.org/10.1007/978-981-13-1501-5_12.
[30] S. Wang, L. Zhu, and M. Cheng, "Docker-based Web Server Instructional System," in *IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pp. 285–289, 2019, https://doi.org/10.1109/ICIS46139.2019.8940219.
[31] M. Moravcik and M. Kontsek, "Overview of Docker container orchestration tools," in *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pp. 475–480, 2020, https://doi.org/10.1109/ICETA51985.2020.9379236.
[32] B. Dordevic, M. Marjanovic, and V. Timcenko, "Performance comparison of native host and hyper-based KVM virtualization," in *28th Telecommunications Forum (TELFOR)*, pp. 1–4, 2020, https://doi.org/10.1109/TELFOR51502.2020.9306550.
[33] B. Dordevic, V. Timcenko, O. Pavlovic, and N. Davidovic, "Performance comparison of native host and hyper-based virtualization VirtualBox," in *20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–4, 2021, https://doi.org/10.1109/INFOTEH51037.2021.9400684.
[34] B. Dordevic, V. Timcenko, E. Nikolic, and N. Davidovic, "Comparing Performances of Native Host and Virtualization on ESXi hypervisor," in *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–4, 2021, https://doi.org/10.1109/INFOTEH51037.2021.9400648.
[35] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, and S. Albayrak, "Developing and Benchmarking Native Linux Applications on Android," in *MobileWireless Middleware, Operating Systems, and Applications*, vol. 7, pp. 381–392, 2009, https://doi.org/10.1007/978-3-642-01802-2_28.
[36] G. Rezende Alles, A. Carissimi, and L. Mello Schnorr, "Assessing the Computation and Communication Overhead of Linux Containers for HPC Applications," in *Symposium on High Performance Computing Systems (WSCAD)*, pp. 116–123, 2018, https://doi.org/10.1109/WSCAD.2018.00027.
[37] Apache.org, "Apache Benchmark tool." Accessed: Aug. 17, 2023. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html.
[38] A. Pagliari, F. Huet, and G. Urvoy-Keller, "Towards a High-Level Description for Generating Stream Processing Benchmark Applications," in *IEEE International Conference on Big Data (Big Data)*, pp. 3711–3716, 2019, https://doi.org/10.1109/BigData47090.2019.9006278.
[39] J. Crussell, T. M. Kroeger, A. Brown, and C. Phillips, "Virtually the Same: Comparing Physical and Virtual Testbeds," in *2019 International Conference on Computing, Networking and Communications (ICNC)*, pp. 847–853, 2019, https://doi.org/10.1109/ICCNC.2019.8685630.
[40] M. Tomisa, M. Milkovic, and M. Cacic, "Performance Evaluation of Dynamic and Static WordPress-based Websites," in *23rd International Computer Science and Engineering Conference (ICSEC)*, pp. 321–324, 2019, https://doi.org/10.1109/ICSEC47112.2019.8974709.
[41] Y. A. Auliya, Y. Nurdinsyah, and D. A. R. Wulandari, "Performance Comparison of Docker and LXD with ApacheBench," *J. Phys.: Conf. Ser.*, vol. 1211, p. 012042, 2019, https://doi.org/10.1088/1742-6596/1211/1/012042.
[42] A. R. Putri, R. Munadi, and R. M. Negara, "Performance analysis of multi services on container Docker, LXC, and LXD," *Bulletin EEI*, vol. 9, no. 5, pp. 2008–2011, 2020, https://doi.org/10.11591/eei.v9i5.1953.
[43] V. Sharma, H. K. Saxena, and A. K. Singh, "Docker for Multi-containers Web Application," in *2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 589–592, 2020, https://doi.org/10.1109/ICIMIA48430.2020.9074925.
[44] O. H. Jader, S. R. M. Zeebaree, and R. R. Zebari, "A State Of Art Survey For Web Server Performance Measurement And Load Balancing Mechanisms," *International Journal of Scientific & Technology Research*, vol. 8, no. 12, 2019, https://www.ijstr.org/final-print/dec2019/A-State-Of-Art-Survey-For-Web-Server-Performance-Measurement-And-Load-Balancing-Mechanisms.pdf.
[45] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, "Performance Evaluation of Docker Container and Virtual Machine," *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020, https://doi.org/10.1016/j.procs.2020.04.152.

[46] Y. Mao, Y. Fu, S. Gu, S. Vhaduri, L. Cheng, and Q. Liu, "Resource Management Schemes for Cloud-Native Platforms with Computing Containers of Docker and Kubernetes." *arXiv*, 2020. Accessed: Jul. 13, 2023. [Online]. Available: http://arxiv.org/abs/2010.10350.
[47] R. F. -Jordan, S. F. -Castell, J. S. -Garcia, J. L. -Ballester, and M. Cobos, "Performance comparison of container orchestration platforms with low cost devices in the fog, assisting Internet of Things applications," *Journal of Network and Computer Applications*, vol. 169, p. 102788, 2020, https://doi.org/10.1016/j.jnca.2020.102788.
[48] Branka, "Linux Statistics – 2023," Linux Statistics – 2023. Accessed: Oct. 24, 2023. [Online]. Available: https://truelist.co/blog/linux-statistics/.
[49] Datadoghq, "How ApacheBench works." Accessed: Sep. 07, 2023. [Online]. Available: https://www.datadoghq.com/blog/apachebench/#success-metrics.
[50] H. Y. Yun, S. H. Jin, and K. S. Kim, "Workload Stability-Aware Virtual Machine Consolidation Using Adaptive Harmony Search in Cloud Datacenters," *Applied Sciences*, vol. 11, no. 2, p. 798, 2021, https://doi.org/10.3390/app11020798.

BIOGRAPHY OF AUTHORS

Rizky Bintang Saputra, S1 Informatics, Faculty of Computer Science, Universitas Amikom Yogyakarta. Email rizkybs@students.amikom.ac.id.

Subektiningsih, is Lecturer at Informatics Department, Faculty of Computer Science, Universitas Amikom Yogyakarta. Have an interest in information security and digital forensics. Like the sound of the camera shutter when capturing, and likes to write on a personal blog. Can be contacted via email at subektiningsih@amikom.ac.id.