A Multi-Containerized Application using Docker Containers and Kubernetes Clusters

Alowolodu Olufunso Dayo
Cybersecurity Department Federal University of Technology
Akure

ABSTRACT
The Cloud is a virtual environment that provisions massive and scalable computing resources and services on a pay-as-you-go basis which is also the process of migrating from a physical environment to a virtual environment. Virtualization could be achieved either through virtual machines or containers. While virtual machines start with a full operating system and all its included software hereby making it bulkier, containers are built to contain only the necessary libraries and dependencies inside them. There has been a huge problem with developing and deploying applications in production and development environments. The same apps that work perfectly well in development can show glitches when deployed in production thereby making the gap between the Development and Operation to be wide which makes the software being deployed to be delivered at a slower pace with an increase in maintenance costs. This has necessitated the introduction of Docker and Kubernetes to mitigate against this and it will be tested with the Fibonacci sequence calculator.

General Terms
Cloud Computing

Keywords
Virtualization, Kubernetes, Docker, Clusters, Containers, applications

1. INTRODUCTION
Virtualization technology which is the bedrock of the Cloud Computing system has been existing for a long time and it depends on software to simulate the functionality of the hardware to create a virtual computer system (Alowolodu et al., 2016). This had enabled the running of more than one virtual system, multiple operating systems, and applications on a single server. However, Virtualization could only be achieved using the virtual machine which had some disadvantages in that Virtual machines require a full operating system to run which at times may be slow to start up, heavy, and therefore not portable. For developers, Developing, running, scaling, and deploying applications in a production environment has proven to be challenging (Alowolodu, 2019). When in a development environment, some applications work perfectly fine, but when deployed in production environments, they may have glitches. There is an increasing gulf between the development and operations time due to these environmental issues. As a result of this issue, software delivery has been slower with increased maintenance costs. To mitigate against this menace, multi containerization is being proposed.
When container technology came into place, Containers revolutionized virtualization. Containers do not require the entire operating system to run which in turn makes the system running time faster, increased lightweight, and portable (Miika, 2018). Containers emulate the user space in kernel mode, as opposed to traditional hypervisor virtualization where multiple virtual machines can run on a single physical machine through an intermediary layer. Containers isolate the workspaces of multiple users. This unique feature of container virtualization has led to its being called operating system-level virtualization. Typical container technology is the Docker whereby an application can be created along with all the dependencies required for it to run and this can be deployed on the Kubernetes. Kubernetes is the container management platform for an extensible, portable open-source platform that enables declarative configurations and automated execution (Kristof, 2018). Containers and DevOps are making their way to companies as well as smaller customers. With these, Organizations will benefit more from these new techniques by being able to resolve issues more efficiently and quickly. According to demand, each service can be scaled up or down which is a typical characteristic of Cloud infrastructures (Alexander, 2020).

2. RELATED WORKS
Virtualization is the process of migrating from a physical environment into a virtual environment. Virtualization has already been adopted by several companies since it reduces the overheads such as maintaining hardware that is present in large rooms or data centers with a wide variety of devices and cables. The use of bulky hardware was not completely solved by virtualization, but it did contribute to reducing the amount of unnecessary bulky and expensive hardware that was burdensome to most organizations as iterated by (Reddy, 2018).

In the past, there has also been a huge problem with developing and deploying applications in production and development environments. The same apps that work perfectly well in development are showing glitches when they are deployed in production. As a result of these issues, the gap between the Development and Operation Teams has widened. Due to these issues, the software is being delivered at a slower pace with an increase in maintenance costs, especially in virtualized environments.
According to (Aaron, 2019), Virtualization has always offered great support for applications that involve the use of an operating system's full functionality when deploying multiple applications on a server or having a range of operating systems to manage. For multiple applications where a minimizing number of servers is always a priority, containers are always the best choice of option. Wes et al., (2018) opined that traditional virtual machine deployments are compared to Linux containers in terms of performance. Workloads in these applications strain resources including memory, CPU, and storage, and that both VMs and containers need to be tuned to support I/O intensive applications.
Using Docker as a well-known representative of container-based approaches, Thanh (2015) analyzed the security level of

the service. An analysis of the two main areas is undertaken: Docker's internal security, and Docker's interaction with the Linux kernel's security features, including SELinux and AppArmor, as they serve to harden the host system.

Shu-Sheng et al. (2015) proposed a Transform-Extract-Load (TEL) based on virtualized data. Various business scenarios led to the development of heterogeneous databases and various strategies, such as query, cache, and disk objectification. TEL was designed to handle multiple sources of heterogeneous data migration, TELS for single-source migration and TELQ for real-time queries. Several metrics were used to evaluate the approach, including low response time, large storage space, and high operational efficiency.

Raghunath and Annappa (2015) proposed a decision-maker that predicts migration triggering. For adding and removing VM resources, this process is performed by vertical as well as horizontal scaling. By removing unnecessary migrations for triggering migration, this system improved resource utilization compared with thresholds.

Muhammad (2015) also iterated about the use of Docker in a bioinformatics computing platform. The work aimed to simplify software selection, automatic building, and deployment of containers with specific software on an existing cloud virtual server instance of a researcher.

Using agent-based virtualization, Monali and Swati (2015) presented smart home applications using sensor networks. In addition to flexibility, diversity promotion, and increased manageability, this application provided flexibility. They have also developed mathematical models for improving allocation and scheduling of resources, measuring energy consumption, minimizing it, and evaluating its performance for a smart home built from a VPN.

3. METHODOLOGY

The proposed system will be implemented with a Fibonacci sequence calculator. Creating this app requires some services i.e., react, node.js, Redis, and Postgres. Each of these services will be in its container, and docker will connect the services using docker-compose, which is a command-line tool used to connect different containers.

3.1 Docker vs Traditional Virtual Machine

An abstraction of physical hardware is a virtual machine (VM), which transforms one server into many. Hypervisors run virtual machines on top of servers and can be used to run more than one virtual machine on one computer. Virtual machines take up several gigabytes of storage space and several minutes to boot up in addition to containing a full version of the operating system and its dependencies. (Miika, 2018).

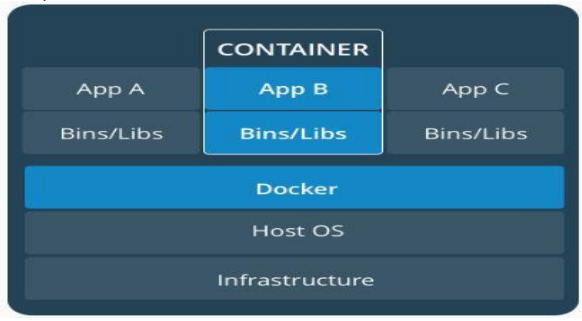

Figure 1 (a) The Container Architecture

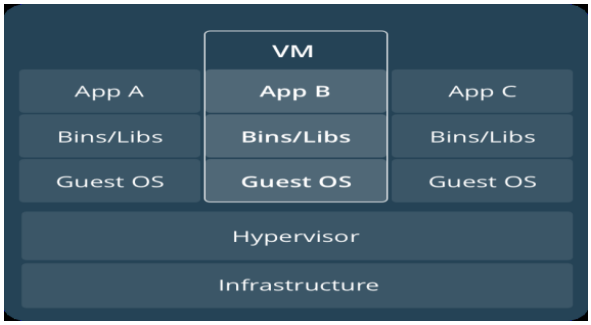

Figure 1 (b) The VM Architecture

The above figures depict the architecture of a container and VM. It is seen that Docker eliminates the need for a hypervisor in creating new environments for application development by using a Docker image.

Table 1 The VM vs Docker Container

	Virtual Machines	Docker Containers
Isolation Process level	Hardware	Operating System
Operating System	Separated	Shared
uptime	Long	Short
Resource usage	More	Less
Pre-built Images	Not easy to find and manage	Readily available
Size	Bigger because it contains the whole OS	Smaller with only docker
Creation Time	Several minutes	Within seconds

Table 1 shows that the Docker container is vanilla, which means it includes only the necessary bootable files for starting up the system. Since these containers are vanilla flavored, it is simple and fast to create them. Containers can be considered lightweight VMs, but they are not VMs.

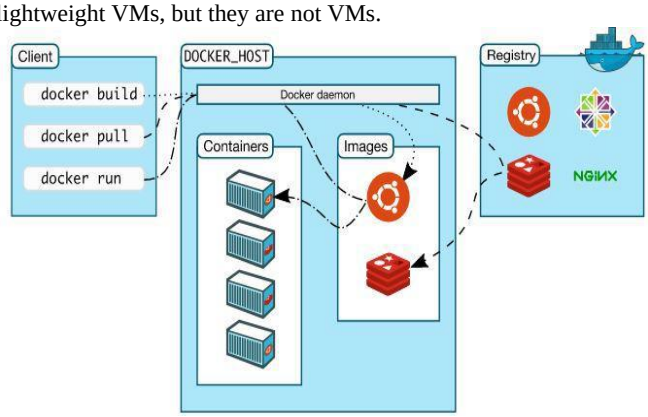

Figure 2 The Docker Architecture (Turnbull, 2014)

From figure 2 above diagram, Docker client, Docker Daemon, and Registry are the three main components of the Docker architecture. Docker client. Containers are Client-Server applications, which means that Docker clients communicate with the Docker servers which in turn execute all the tasks. The Docker client library comes with full HTTP request/response capabilities for use (Turnbull, 2014). Docker client and daemon can be run on the same host or can connect local Docker client to remote Docker Daemon which is running on the remote host like AWS Server.

3.2 Kubernetes

Developers interact with Kubernetes using deployment files. The deployment file is a configuration file that contains how the images are to work. The master has a Jotter, that records all its responsibilities or essentially all the activities in the form of these deployment files. On reading the file, it seems that the developer wants to run two copies of the worker then the master updates its little list of responsibilities and that there should be two copies of the worker image running, yet none are running now. Kube-server communicates with each of these nodes and instructs them to start up a copy of multi-worker. Inside of each of these nodes is a copy of docker running that reaches out to Docker hub, finds the worker image, then copies or download that image and stores it on a local image cache inside each of these nodes. Each node now uses the image to create a new container. Since each of these has started up a copy of the worker, the master reaches back to each one of these nodes and asks for a status update. Then it checks the little list of activities as shown in figure 3.4 and confirms that two copies of multi-worker are now running. The same step is applicable for each of the containers. By keeping track of the current status of all the different nodes within the cluster, and making sure they are functioning correctly, the Kube API server program maintains the overall status of the cluster.

3.3 The Fibonacci Calculator Architecture

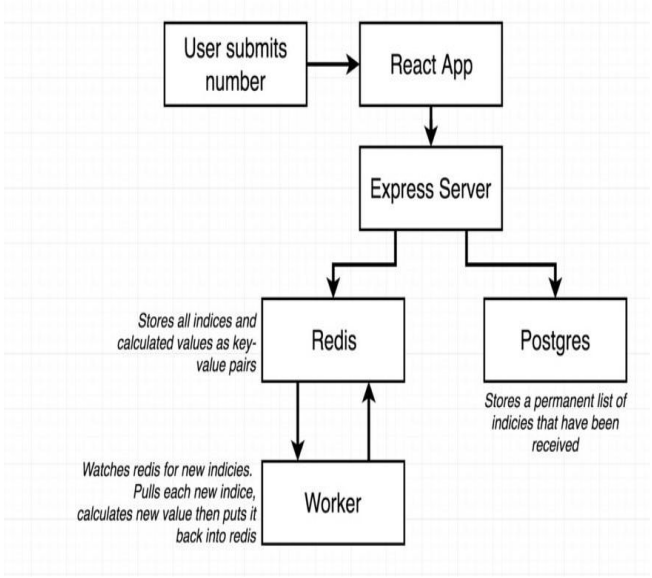

Figure 3 Fibonacci Calculator Architecture

When a user enters a number to the react application, the react application will make an ajax request to the express server to calculate a Fibonacci number, which will be stored in Postgres. (Which has a permanent list of all the indexes that have ever been submitted to the app), While the express server does that, it saves that index to the Redis database. When a new number shows up inside of the Redis database, it takes up a separate backend node.js process which is referred to as the worker process. Each time a new index appears in Redis, the worker process accesses it, then pulls out that index, the worker will calculate its Fibonacci value. It takes that calculated value and puts it back into Redis, in this way, react can request it and it will eventually appear on the screen. To achieve this, multiple Dockerfiles will be created, then docker-compose will be used to connect them.

3.4 Environmental Setup

Docker engine needs to be installed locally on the computer to be used. Docker can be downloaded from the official website(docker.com). Also, an Integrated Development Environment (IDE) such as vscode is required.

3.4.1 Creating the Source Code

To get started, a new folder called complex will be created, this folder comprises all the source code required to build this application. After creating the source code, then docker will be used to dockerize each of the services.

3.4.2. Dockerizing the Services

Inside the complex directory, there are the client, server, and worker folders and each of these folders represent the React Server, Express API, and the worker process respectively. The next step is adding docker containers to each of these applications so that it can start each of them up inside of a development environment. Dockerfiles for the re-act project, Express API, and the worker process will be created. In the project files, inside of each of those directories will have a similar Dockerfile workflow since each one of these projects will have a package.json file that records all the dependencies of the project. Fig 4 depicts a dockerfile template.

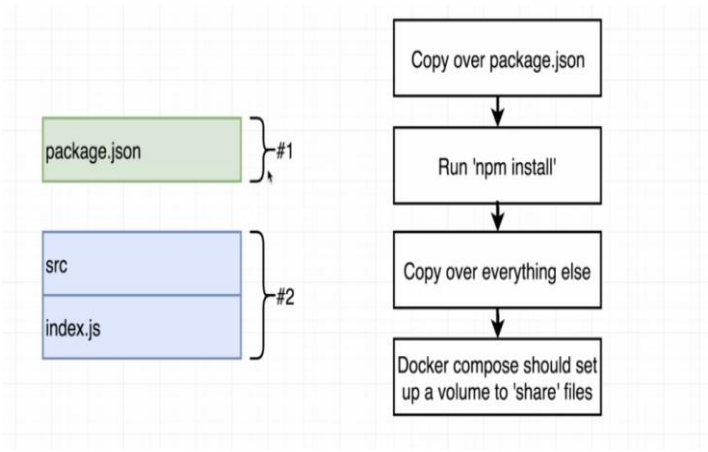

Figure 4 Docker File Template.

3.5 The Proposed System Architecture

The proposed system is a 2-tier architecture that comprises of Fibonacci sequence Docker and Kubernetes workflow. The Fibonacci sequence calculator contains different services, each on its folder. The client folder contains the react application, the worker folder includes the function for the Fibonacci calculator, the express folder comprises of express.js service, and Nginx consists of the Nginx web server. Each folder also has a Dockerfile which is essentially a plain text file that has a couple of lines of configuration placed inside it. This configuration defines how a container should behave. The Docker daemon also known as the docker server, allows developers to communicate with docker through the command-line interface (CLI). The Docker compose is a command-line interface tool that runs multiple containers. Docker-compose builds each container and pushes them to the docker hub which is a repository of free public images that can be downloaded and run on any computer.

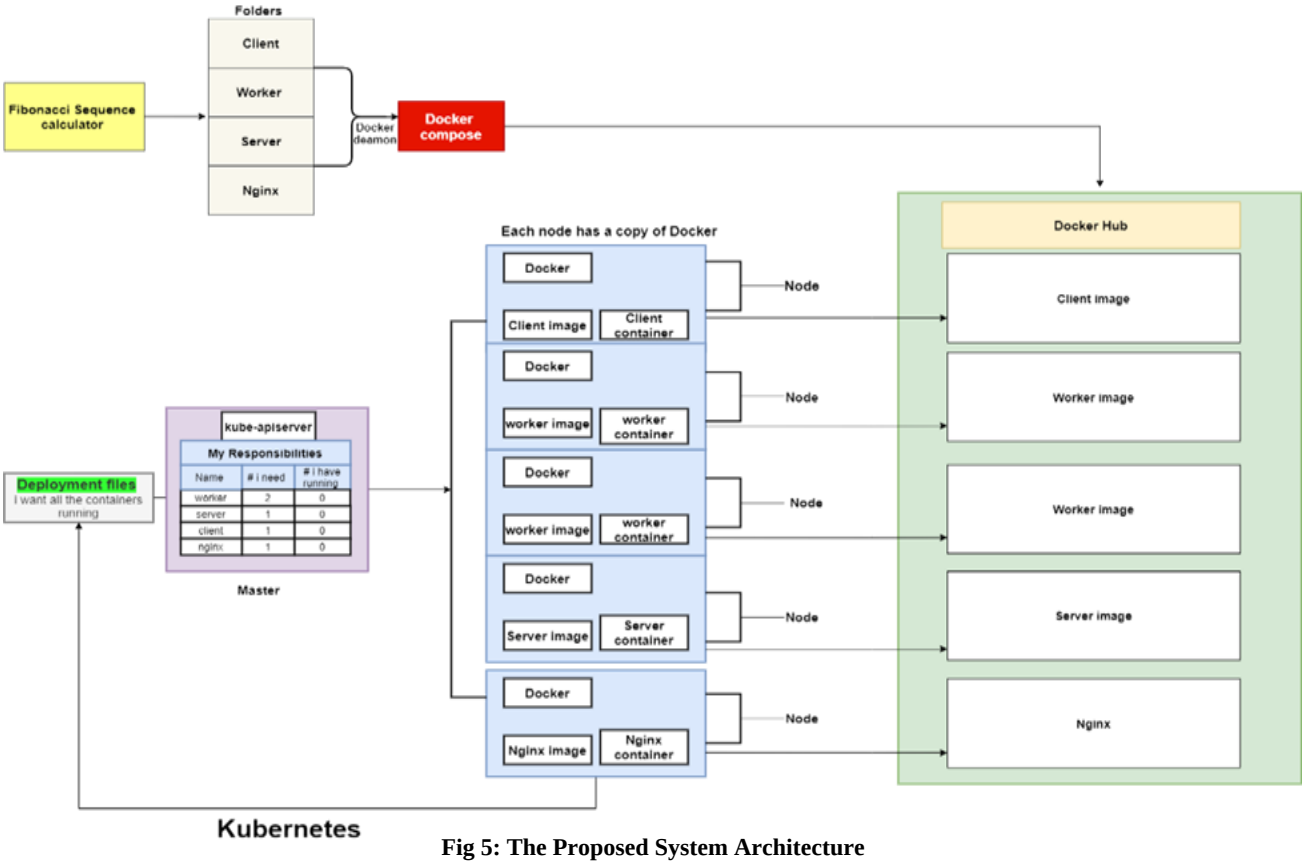

Fig 5: The Proposed System Architecture

4. RESULT AND DISCUSSION

The system had been designed in such a way that to migrate from a running application with Docker Compose to Kubernetes, this will aid the developers to scale their application and make it resilient so that deployments and bugs do not cause downtime. To interact with the Kubernetes cluster, developers make use of a program called Kubectl which is used to interact with a Kubernetes cluster in general and manage what all the different nodes are doing and what different containers they are running. Config files are made for the worker, express, and server image. Another configuration file is created which handles networking setup and makes sure that the container that is created with the first configuration file will be exposed and available to the outside world. The configuration files which have been written would be put together and fed into the Kubectl command-line tool. When passed into Kubectl, it will interpret both files and create two objects out of each file.

4.1 Deploying the application

The configuration files have been written and are ready to be loaded into the Kubernetes cluster and try to access a running container. The Kubectl command-line tool is used to perform this. The command is the "Apply" command and then the name of a configuration file. When creating the virtual machine on the computer, the IP address that was assigned to the virtual machine on the computer must be known so that it can be accessed from the web browser. To get this IP address, the command minikube IP will be run on the terminal, which will print out the IP address of that virtual machine. After getting the IP address, then map it to the node port.

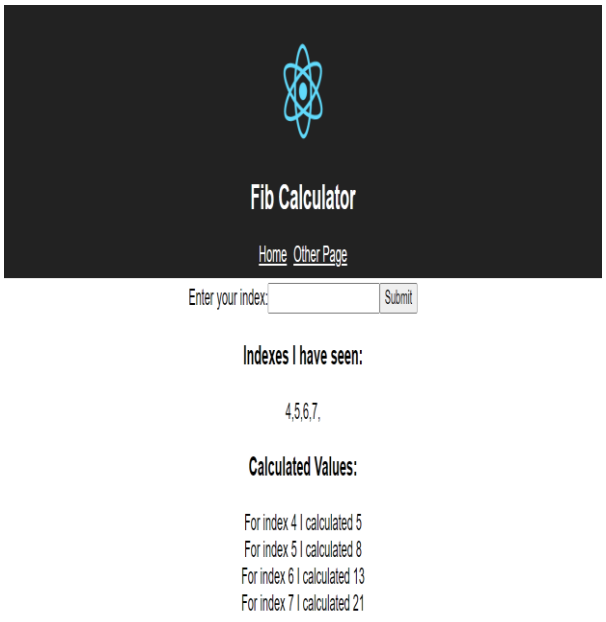

Figure 6 Application running in Kubernetes Cluster

4.2 Performance evaluation

The evaluation of Hyper-V VM and Docker was performed. Both were tasked with running a Fibonacci Sequence Calculator. During the evaluation, the following areas were assessed: computation time, CPU performance, memory throughput, and storage reads/writes. The two-virtualization technology, Docker, and VM were tested in the same environment and the same condition with the following

specifications core i5 processor at 2.7 GHz at a total of 2 cores and 8 GB RAM on Windows 10 O.S for uniformity and consistency.

4.2.1 Time of Computation

A computation time, also referred to as running time, is the time spent performing the execution of the Fibonacci series calculator that was tested on the VM and Docker. These range from the start-up time and the time it takes for the execution of the app to be calculated. The calculator was run at five different times and the average time was computed.

Table 2. Run Time

Virtualization	Computation Time (sec)					Avg
Docker	53	54	52	51	50	52
Virtual Machine	97	95	97	96	95	96

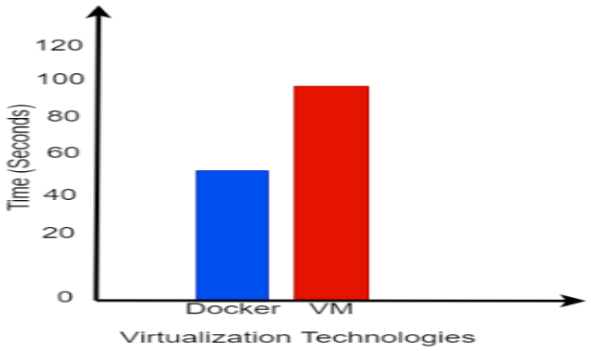

Figure 7 Time of Computation

Table 2 and figure 7 show that Docker takes less time (55sec) to load and compute than the VM (96 sec).

4.2.2 Memory Consumption

Table 3. Memory Consumption

Virtualization		In Percentage					Avg
Docker	CPU	34	32	32	34	33	33
	RAM	58	58	53	54	52	55
VM	CPU	53	52	50	47	48	50
	RAM	45	45	43	44	43	44

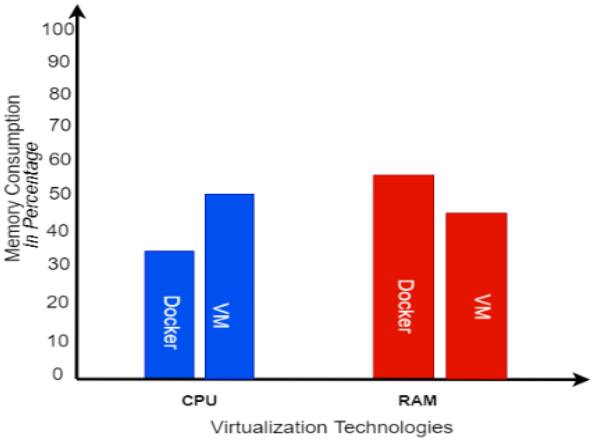

Figure 8 Memory Consumption

This section measures how much memory and ram is consumed by both virtualization technologies that is Docker and VM. Machine. Fig. 8 represents memory consumption and CPU utilization (in percentage). It is seen that Docker consumes less CPU at (33%) but more RAM (55%) than VM whose CPU performance is (50%) and RAM consumption at (44%). The reason for the greater Ram Consumption of the Docker is because the number of services that were used to build the Fibonacci Calculator on Docker is more (about 5) than that of the VM (just 1).

4.2.3 Disk I/O Performance

Disk I/O performance is essential to test the read and write operational speed of Docker and VM. When building an image in docker, it is required to copy(read) files (such as source code for the project) and folders (such as pictures, etc.), from the host machine to the container, also when saving(writing) images to docker hub. This also applies to VM, when sharing images from one server to another. Table 4 and Figure 9 show the comparison between both Virtualization technologies. The Disk I/O performance (read and write operation) of Docker is quite higher at (23) than the VM at (6).

Table 4. Disk I/O Performance

Virtualization		Read/write Performance					Avg
Docker	Read	25	25	23	21	21	23
	Write	7	7	6	5	5	6
VM	Read	2	2	2	2	2	2
	write	4	3	3	3	2	3

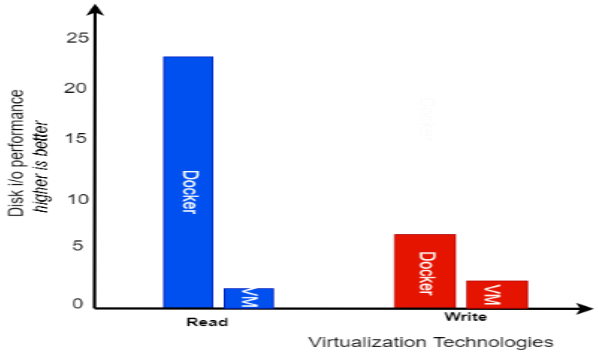

Figure 9 Disk I/O Performance (MBPS)

5. CONCLUSION

This application made use of several services, each of these services was running in a different container. Docker connects these containers using docker-compose. This application is further deployed to Kubernetes. Kubernetes helps to scale by creating multiple instances of the containers and handling each request independently as traffic requests increases. It is also performing systemic checks to ensure all containers are working as expected, if not, it "self-heals" them and helps to automate the containers. Docker containers have simplified the development of applications across all environments, including development, testing, and production. This study also concludes that when Kubernetes is utilized to deploy an application, there is a significant decrease in downtime when scaling up the number of containers. With these, Organizations and developers will benefit more from these new techniques by being able to resolve issues more efficiently and quickly. Developing, running, and scaling up applications in different environments will be an easy task without any stress.

6. REFERENCES

[1] Alexander S. Gillis, 2020; What are containers (container-based virtualization or containerization) <

https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization.>.

[2] Alowolodu Olufunso Dayo, Alese Boniface Kayode, Adetunmbi Olusola Adebayo (2016); Secured Cloud Application Platform using Elliptic Curve Cryptography, Proceeding of the World Congress on Engineering and Computer Science (WCECS 2016)

[3] Alowolodu Olufunso Dayo (2019); Cloud-Based Platform for Student Developers (case study: Computer Science Department, FUTA), American Journal of Information Science and Computer Engineering, Vol 5, No. 2, pp 29-37

[4] Aaron Strong, 2019, Containerization Vs Virtualization, What's the Difference? https://www.burwood.com/blog-archive/containerization-vs-virtualization

[5] Miika Moilanen, 2018 Deploying an application using Docker and Kubernetes. Moilanen_Miika_Opinnaytetyo.pdf (Theseus. fi)

[6] Kristof, 2018, Fix Observability format to be consistent with other items Kubernetes/website@9eb90fd · GitHub

[7] Muhamad Fitra Kacamarga, Bens Pardamean, and Hari Wijaya, Lightweight Virtualization in Cloud Computing for Research, 2015, Bioinformatics & Data Science Research Center, Bina Nusantara University Jakarta, Indonesia.

[8] Reddy Srinath Meadusani (2018); Virtualization Using Docker Containers: For Reproducible Environments and Containerized Applications" (2018). Culminating Projects in Information Assurance. 50. https://repository.stcloudstate.edu/msia_etds/50

[9] Wes Feller 2018, An updated Performance Comparison of Virtual Machines and Linux Containers, An IBM Research Report.

[10] Thanh Bui 2015, Analysis of Docker Survey (16) (PDF) Analysis of Docker Security (researchgate.net)

[9] Shu-Sheng Guo 2015 A new ETL Approach based on Data Virtualization *J. Comput. Sci. Technol.* 30, 311–323 (2015). https://doi.org/10.1007/s11390-015-1524-3

[11] Raman Raghunath and Annappa (2015), Virtual Machine Migration Triggering using Application Workload Prediction

[12] Monali and Swati Nikam (2015), A Review Paper On Virtualization Technology In Cloud Big Data Abstract

[13] Turnbull J., 2014 "The Docker Book," https://dockerbook.com/.