American Journal of Science and Engineering, 2020, Vol-1, Issue-3, XX-XX, 2020 in AJSE Publishers (http://ajse.us)

Hardware Utilization by using Docker

Marshia Mostafiz Mim
American International University-Bangladesh (AIUB)
mostafizmim24@gmail.com

Joydeb Karmakar
American International University-Bangladesh (AIUB)
krmkjoy@gmail.com

Mrinmoy Karmakar
American International University-Bangladesh (AIUB)
mrinmoy602@gmail.com

Moshfiq-Us-Saleheen Chowdhury
Military Institute of Science and Technology (MIST)
moshfiqussaleheenchowdhury@gmail.com

Jannatun Nayim Supti
American International University-Bangladesh (AIUB)
jannatun.supti@gmail.com

***Abstract*—Developers and system administrators may use Docker to construct, ship, and operate distributed applications. Docker's main advantage is that it enables code to be quickly tested and deployed into production across a variety of applications. This article looks at the performance of Docker containers. They are judged on the effectiveness of their system. This is predicated on making the most of the system's resources. Docker Swarm is an open-source project that lets you build, deploy, and operate applications in a virtualized container environment. The goal of this research is to use the host computer's resources to spread web server traffic across a Docker swarm. Using this method, a single point of failure in a web server cluster is less probable**

Keywords—Benchmark, Container, Docker, Load balancing, Resource Usage, Swarm, Web Cluster

I. INTRODUCTION

Docker is a new concept in Computer science by using this we can utilize hardware that can be helpful to all of us and society. Learning about the docker system very deeply is one of my motivations. I want to improve the system and I want to make it easier and more efficient. Improvement of this system is my main concern. Research can help us to develop the existing system. I think in our upcoming development and deployment world it will be very promising. In our personal computer or servers, there are a lot of unused memory storage left and a small part of it is used. Docker's key benefit is that it allows code to be tested and pushed into production as quickly as feasible. Docker is a free and open platform that allows developers and system administrators use Docker Hub, a cloud service for sharing programs and automating processes, and Docker Engine, a portable, lightweight runtime and packaging tool, to create, ship, and execute distributed applications. The primary benefit of Docker is that code can be tested and deployed as soon as possible into production in various applications are possible. Be executed in a language-independent way across Docker containers. This article investigates the performance of Docker containers. They are evaluated based on the performance of their system. This is founded on the Utilization of system resources Various benchmarking tools are available. This is what I used. The performance of the file system is measured. Bonnie++ is being used. Other system resources, such as CPU usage, Benchmarking is used to evaluate memory utilization and other factors. Python code (using psutil) was created. Obtaining specific results. This document also includes the findings of all of these testing. CPU utilization, memory usage, CPU count, and CPU times, as well as disk partition, network I/O counter, and so on, are the results. (Preeth, 2015)

Virtualization using containers is becoming increasingly common. It's a kind of mild virtualization that takes use of the Linux kernel's ability to separate applications and control resource allocations. Docker is a virtualization system based on containers that is extensively used today. Docker is an open-source software that allows you to create, ship, and run applications in a containerized environment. Docker may be used to serve millions of users by deploying several web application containers. It can decrease a lot of chances of the architecture which will have one of the failures. Arranging a few containers to form one service, on the other hand, is a challenging undertaking. Docker Swarm, a container cluster management solution, solves this problem. The internal load balancing mechanism in Docker Swarm was designed to distribute requests amongst workers in a fair and equitable manner based on the user's request. It has no means of knowing how much of each host system's resources are being used. It's concerning since it might result in an imbalanced load distribution across host systems. On each host system, we focused on memory consumption. We propose a technique for monitoring and distributing web traffic depending on memory use on each host computer. Our experiment turned out to be a success. A balanced load distribution is applied to each worker node. In a web server cluster, this strategy may assist to less the chances of one point of failure. (Bella, 2018)

Web servers have become a critical component of the internet's architecture. The web application runs on top of the most popular internet tools, such as Google, YouTube, and Gmail. Building a reliable web server is essential for safeguarding the company from system risks such as outages, message processing errors, insufficient data, and data loss. (M.Waliullah, 2014). Container-based virtualization, rather than virtual machines, has lately acquired traction, especially in the deployment of microservice architecture. (O.Sallou, 2015) (M.Villari, 2016). Container-based virtualization is a kind of virtualization that divides and regulates process resource allocations using Linux kernel capabilities. It doesn't need or

include its own operating system; instead, it makes advantage of the host kernel's functionality and resource isolation (CPU, memory, block I/O, network, and so on) to completely isolate the application's operating system. (Preet, 2015) (S.Julian, 2018).

Docker is an open-source software that allows you to create, ship, and run applications in a containerized environment. A Docker container isolates the application from the host infrastructure. By packaging the program and its dependencies into a container, it may be possible to minimize the time between creating and deploying code. (Preet, 2015).

There are two sorts of nodes in the Docker Swarm: management nodes and worker nodes. A Docker host computer may act as both a worker and a management node. The administration node uses its own IP address and port to provide the swarm's services to the outside world. (Docker, 2018).

II. Literature Review

Docker is an open platform that provides container-based virtualization. Container-based virtualization is also called OS-level virtualization. It delivers software in packages called Container. Containers are isolated spaces, and they are separated from one another. It is a client-side application program. It is also can act as a service and can be deployed onto any server. Portability is one of the absolute charms of the docker (M.Waliullah, 2014) (M.Chae, 2017) (Docker, 2018). Docker is advanced level virtualization.

Docker allows developers to package the application into a container. Containers also contain host OS configuration, networking information, also code or application dependencies. The main purpose of Docker is like one application or program is running on a developer pc or server perfectly but as soon as it deploys on another pc or environment it might not act like the same (Mochamad, 2018) (M. Villari, 2016) (J. Liu, 2006). Docker can help to solve this problem.

With Docker, you'll oversee your framework within the same ways you oversee your applications. In the Docker container, every dependency related to the application will be packeted up. There is another option in virtualization which is Virtual Machines (VM). Virtual Machine works like a fully new OS machine. It (VM) consumed hardware resources because it is hardware-based virtualization (Preeth, 2015) (Mochamad, 2018). There are some bad sides of virtual machines like consuming more hardware and creating a new VM is time-consuming. This problem can be solved by using OS-based virtualization Docker. On tracking data and operating strategies specified by the end-user, application developer and/or administrator, per container, features of the resources needed by the host can be allocated. Preeth et al. evaluated Docker container efficiency with a focus on device resource optimization. The experimental scope has been broadened to include researching container scalability to reduce resource waste and increasing prediction accuracy on a web application using a load balancer. In terms of virtualization technology performance, the authors suggest that container-based virtualization may be compared to a memory-based OS operating on bare metal (2016). Docker has a lower resource usage than virtual machines. (Preeth, 2015) (Mochamad, 2018) (S. Julian, 2018).

Docker, on the other hand, has several flaws.

In the Docker system, there are three types of networking hosts. Bridge networking for single hosts, overly networking for multi-host communication, and Macvlan networking are all used in docker containers. Docker containers and services are very powerful because they can easily be integrated with other Docker containers and services, as well as non-Docker workloads. Containers and services in Docker don't need to know whether their neighbors are Docker workloads. Docker will manage your Docker hosts whether they are running Linux, Windows, or a mix of the two. Docker may be more powerful than a virtual machine in certain situations (VM). Hardware is disabled in the virtual machine to make room for a new virtual OS that functions similarly to a real computer. However, since Docker is an OS-based virtualization, it uses fewer resources. Docker is a modular system. The framework is built on ECo-Ware, which defines self-resource adaptation as a meta-workflow strategy for dealing with applications. It comes with the TOSCA library to make infrastructure easier, and it's been tested on Amazon EC2 to look at resource metrics like average response time. Despite the fact that, resource management is embedded into the container. Virtual machines have a major issue that will influence cloud computing. That is resource squandering. In the massive cloud system, this will be a major issue. However, certain testing show that Docker's CPU management isn't always reliable. (Mochamad, 2018) (M. Waliullah, 2014) (O. Sallou, 2015). This is one of the drawbacks of docker. But in Ram management docker is better than VMs. Docket CPU management problems will be a big problem in the long run. In some research, CPU management is not good for some situations (Mochamad, 2018) (M.Villari, 2016) (M. Fazio, 2016). This particular issue has room for improvement. The phrase is most often used to describe information centers that are available to many users over the internet. If a CPU issue occurs in cloud computing, the whole process will be disrupted. As a result, hardware usage is critical.

III. Methodology

A. Proposed Research Methodology

According to my research topic, I need to follow two methods and they are formal method & the model method. In my research, I went to utilize the hardware of the Docker system (Preeth, 2015). It will help the upcoming user experience. How the system improvement helps the new user that we can measure by using model method (G Pierre, 2020). In Docker, there is some ram issue that makes upgrade hassles, performance-critical applications, multiple operating systems, security is a critical factor critical with some ram and CPU issue (O. Sallou, 2015) (S. Julian, 2018) (G Pierre, 2020). We can design a bespoke model to comprehend and address this problem on a small scale in this kind of situation. When working on a huge project, the model might be useful. Modelling is the process of reducing an actual or projected model to a small, but representative, collection of components and interactions that allows its

attributes to be expressed qualitatively and quantitatively. In big data analysis and workload can be understood by model method (M. Fazio, 2016) (Preeth, 2015) (N. Thoai, 2016). With the expanding prominence of holder advances, many exploration endeavors have been made to investigate the points of interest of holders, improve compartment execution and security, just as to contrast the holders and the VMs (Mochamad, 2018), (N. Thoai, 2016). Carl (Docker, 2018) addressed the typical challenges avoided by the reproducibility of a vast number of research projects and the evaluation of how they can be overcome by the Docker container technique. (M. Data, 2018) Roberto provided a performance assessment of container utilization in the Internet of Things industry in terms of container performance and security. Roberto demonstrated that the containerized layer had no effect when compared to native execution utilizing a single board computing system like the Raspberry Pi 2. To further understand why there is such a huge disparity between Docker and VM bootup speed, we must study the real system's memory use using the model technique (Docker, 2018), (M. Data, 2018). Then, to improve the system Formal method is very important. The formal method basically works with algorithms analysis, space complexity analysis, time complexity analysis which is important for hardware utilization of Docker. To improve the system complexity analysis is important and very relevant to think. There was some algorithmic error that occurs this kind of hardware utilization problem (M. Waliullah, 2014), (O. Sallou, 2015), (M. Fazio, 2016). Container virtualization is gaining popularity as a technology. As a result of IBM's mainframe implementation of virtualization in the 1970s (i.e., the hypervisor), container-based technologies have been developed (for example, Docker and Linux Container). It was newly introduced. The key benefit in containers is that near-native efficiency is attained. It arises from a review of the literature on performance measurement of container runtime environments that study works can be divided into workload distributed database studies and other concurrent workload studies (A. M. Joy, 2015). The space and time complexity should be reduced in order to reduce resource consumption (O. Sallou, 2015), (M. Villari, 2016). The formal procedure should be used for this. Formal techniques are often employed in computing science to verify truths about algorithms and systems. (S. Shirinbab, 2018), (C. Boettiger, 2015) and (Dua R, 2016). Researchers may be interested in formal speciation of a software component in order to automate the variation of that component's implementation. Alternatively, researchers may be interested in an algorithm's time or space complexity, as well as the accuracy or quality of the answers it generates. (M. Luthfi, 2017), (R. Morabito, 2016), (Dua R, 2016). The main reason for this research is to give a better system and the research will be successful when the hardware and fully optimized by following my research methodologies- model and formal methods.

B. A Survey on Docker Container and its use cases

The criteria and explanation of Docker Container and use cases are given below:

a) Simplifying Configuration-

This is the most common use case; Docker setups may be reused in different contexts several times. Virtual machines have an advantage when operating any platform with its configurations; Dockers have the same advantage, but without the Virtual Machine overhead, and allow users to put the configurations and environment into the code and distribute it. The application environment and infrastructure needs are separate. In terms of real-world use, it allows businesses to speed up project setup by allowing them to go right into work without going through the tedious process of setting up environments and configuration processes.

b) Code Pipeline Management-

A few minor differences can be observed as code written in a developer environment progresses through various stages (each of which uses different platforms/environments) and approaches the production stage; however, Dockers provide a consistent environment throughout all stages from development to production, allowing for a simple development and deployment pipeline. The steady nature of the Docker image and the ease with which it may be launched can help with the pipeline mentioned above management.

c) Docker for Production Efficiency-

In a development environment, Docker makes it simpler to meet two goals: keeping a developer near to production while still allowing them to work remotely. The second need is an interactive development environment, which Docker meets by making application code from the host OS accessible to the container through shared volumes. This provides a number of advantages, including the developer's ability to make changes to the source code from his preferred platform and see the results. Multi-Tenancy - Docker is utilized in multi-tenant systems since it helps to avoid significant application rewrites. The codebases of multi-tenant systems are far more complicated, stiff, and difficult to manage. Redesigning an application takes a significant amount of effort and money. Docker simplifies creating limited environments to execute numerous apps for each tenant easier and more cost-effectively. Docker's simple API makes it possible to spin up containers programmatically.

d) Debugging Tools-

Docker offers many tools that work well with the notion of containers. One of its features is the ability to checkpoint containers and their versions, distinguish two containers, and quickly repair applications when necessary.

e) Improved Disaster Recovery-

In the event of a disaster, a Docker image, also known as a snapshot, may be saved and retrieved at a specific point in time. A Docker image may be

used to transition between two distinct versions of the same software, and a file can be duplicated to new hardware using Docker.

f) Increased DevOps Adoption-

To standardize confined deployment, the DevOps community has developed foundations on Docker. Docker's relationship with DevOps has been established via CI/CD, and Docker ensures consistency in both testing and production settings. Docker simplifies machine setup by standardizing the configuration interface. Docker might be utilized to assist the company improve its DevOps.

IV. Related Work

As container technologies have grown in popularity, a number of research projects have been established to better understand container boundaries, increase container performance and security, and compare containers to virtual machines. (N. Thoai, 2016). Carl examined the common obstacles that prevent many research projects from becoming replicable, as well as how Docker container technology may assist. Many projects, for example, need certain requirements in order to recreate the same findings as the original researchers, but due to the project's diverse underlying OS and hardware, it's impossible to just provide an installation script. In order to tackle this issue, Docker was created, which delivers a tiny binary image that includes all of the necessary software. In Paolo's company and with his other pals You may use Docker to link the implementation of many containers and run a channel application in Genomic pipelines. (C. Boettiger, 2015) In the company of Douglas' buddies One of the benefits of using containers in HPC systems is their versatility and reproducibility, according to the research. Several scholars have investigated the advantages of cloud computing container technology. According to the research team, cloud-based platforms such as Platform-as-a-Service (PaaS) may help containers because of the simplicity with which they may be used, set up, and deployed. You can count on Roberto for container efficiency and safety. (Raho M, 2015) In the Internet of Things area, a performance assessment of employing containers was presented. Roberto proved that the containerized layer has a minor effect compared to native execution by managing containers on top of a single board computing device like the Raspberry Pi 2. The use of container technologies like Linux Containers, Linux VServer, and OpenVZto has been shown by Miguel and his colleagues to provide a very low overhead HPC environment when compared to native systems. Thanh (Docker, 2018) Docker's internal agents offer security, and how they interact with the Linux kernel's security features was investigated. Sergei et al. (J. Liu, 2006) used Intel SGX authorized execution technology to solve container security and prevent outside attacks. Many individuals have been interested to studies comparing receptacles and virtual machines. (A. M. Joy, 2015), (Z. Kozyev, 2017), (Lars, 2019), (Docker, 2018), (S. Shirinbab, 2018), (Preeth , 2015), (Nguyen, 2016), (Goldberg, 1974), (Chae, 2019).

There was just one physical machine used in Janki et al. study, which does not reflect the normal cloud architecture when looking at the performance of Spark processes operating inside a container cluster vs a virtual machine group. It was also endorsed by Claus (Z. Kozyev, 2017). Virtualization is used by containers and virtual machines alike (VMs). Using containers to package and manage applications and the PaaS cloud, the author claims, is a superior solution since containers better support microservices. Using these tools, we were able to measure how much CPU, RAM, and the network were being consumed by various containers and virtual machines. When random disk access is necessary, containers outperform VMs for disk and network I/O-intensive workloads, even though both have moderate CPU and memory performance overhead. Containers and VMs were permitted to produce 100000 factorials through I/O. Kyoung-Taek et al. (Nguyen, 2016) for this research and their boot uptime and computation performance were examined. VMs and containers cost the same amount of energy under idle, CPU/Memory stress, and network-intensive workloads, according to Roberto (Goldberg, 1974). However, containers require less power for these workloads.

V. Docker Usage

A. When to Use :

a) Learning About New Technologies:

Docker enables individuals to use a new tool in a disposable and isolated environment without spending time installing and configuring it. Several projects save docker images containing apps that have already been installed and configured.

b) Main Use Cases:

Docker Hub is a cloud registry service that allows customers to get Docker images created by other communities; it is a convenient way to get images for either Basic or Standard applications.

c) Program Isolation:

Managing each application component in its container eliminates dependency on other apps on a different server.

d) Dev Groups:

For developers working in various settings, Docker provides a close match between local development and production environments.

B. When Not to Use :

a) Sophisticated Apps:

Unlike simple applications, complicated applications will benefit from using pre-obtained docker files or fetching images from Docker Hub since modifying, constructing, and managing requests and replies across several containers on multiple servers is time-consuming.

b) Behaviour of Applications:

Docker outperforms VMs since Containers share the host kernel and imitate the complete operating system when it comes to performance. Dockers can be avoided to get the most remarkable performance from the server since processes operating on a native OS are quicker than processes running within a container.

c) Upgrade Troubles:

Docker is still a developing technology that needs regular upgrades to take advantage of new features.

VI. Performance Evaluation

For the purposes of this essay, a four-core LINUX machine running Docker was used, which did not need the use of a hypervisor. The host computer has 4GB of RAM, a 512GB hard drive, and a Core i5 processor running LINUX Ubuntu 12.04 64-bit OS.

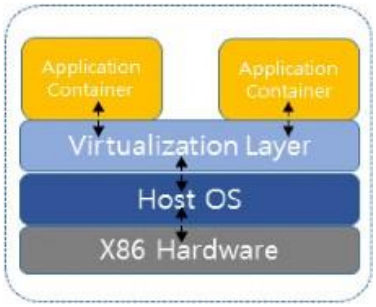

Fig. 1. Virtualization at Operating System Level

TABLE I. Table Type Styles

Virtual Machine	Docker
Virtual machines run on virtual hardware, and the guest operating system is loaded into memory.	All visitors have access to the physical memory of the Host OS.
Guests communicate via network devices, which may or may not be software.	Pipes, sockets, bridges, and other means of communication are used to connect visitors.
The hypervisor is in charge of security.	There are no security measures in place.
Because of its complexity, it has a higher overhead.	Because the containers are light, there is less overhead.
It is not feasible to share libraries or files.	File sharing is available (for example, using LINUX's SCP command).
It takes a long time to boot up.	Faster start-up.

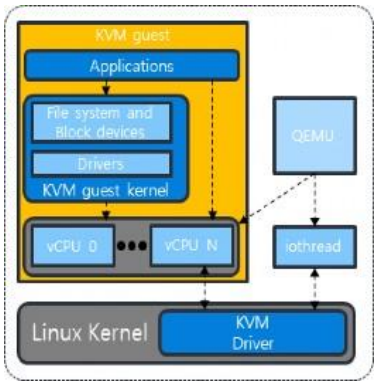

Fig. 2. KVM Architecture

A. *Bonnie ++ :*

Hard drives and file systems may be tested using the C++ Bonnie++ (O. Sallou, 2015) benchmarking suite. The following picture shows a Docker container running Bonnie++. Several I/O performance tests are included in Bonnie++. A file system's performance may be gauged by looking at how long it takes to read or write data. Data with a fixed size of 40MB was used to evaluate the container's reading and writing speeds. Bonnie++'s execution can be tracked down using the container id: 111c5ab491fe. There were precisely 40 million putc () operations per second in Bonnie's putc () call output rate of 507 k per second. Windows 98 was the most popular operating system.

TABLE II. Bonnie++ On Docker

Container ID	File size	Sequential output		Sequential input	
-	-	Output rate	CPU time	Input rate	CPU time
256e3sh643rg	46Mb	502Kb/s	96%	1342Kb/s	95%

On an Ubuntu 12.04 machine, the following is the result of executing Bonnie++. To highlight the performance differences between Docker and the host operating system, this experiment has been set up to do just that. The host and guest operating systems run at different speeds. The Host OS utilizes much less of the computer's resources than a container running on top of it. As a consequence, the Host OS is a little bit quicker than the Guest OS.

TABLE III. BONNIE++ ON HOST OS

Container ID	File size	Sequential output		Sequential input	
-	-	Output rate	CPU time	Input rate	CPU time
User-OptiPlex	46Mb	576Kb/s	96%	5784Kb/s	93%

B. Psutil :

A Python package known as (M.Villari, 2016) psutil shows data about running processes and how the system is being used (CPU, memory, storage, and network). Monitoring, profiling, assigning resources and controlling processes are the most common uses of this tool. The 32-bit and 64-bit architectures support a wide range of operating systems, including LINUX, Windows, FreeBSD, Sun Solaris, and many more options as well.

a) Memory management:

Retrieving the memory used by containers is a memory management feature: According to the table, memory use will be given in numerous areas, with each field represented by a byte.

• Entire: The whole amount of physical memory on the host system is shown here.

• Used: Previous apps have used up all of this RAM.

• Memory that isn't being used but is still accessible is referred to as "free memory." In addition to what has already been said.

• Active memory refers to memory that is now or recently in use. This is a command in the UNIX operating system.

• Buffers: Metadata from the file system is cached in memory.

• Cache: A memory that is used to keep track of a large number of objects. This is utilized to increase the performance of the system. A total of 20.2 percent of RAM was utilized. The memory usage test for the Docker container yielded the following results.

b) CPU times:

Returns the number of times the CPU has been used. Time spent in various modes is shown on the screen. The following table lists many aspects that denote the passage of time, each having its own unique qualities based on the platform on which it is shown.

TABLE IV. CPU TIMES OF DOCKER AND HOST OS

Machine	User	Nice	System	Idle	Iowait	Irq
Docker	24.22	20.41	18.20	13312.87	87.88	3.51
Host	24.93	20.41	18.64	13656.47	91.24	3.87

TABLE V. MEMORY USAGE OF DOCKER AND HOST OS

Machine	Total	Used	Free	Active	Buffers	Cached
Docker	3915586415	0965808211	2420113432	6468111184	1213455235	7513455412
Host	3915586415	1252315764	2623308546	5746203357	1213555360	7512332342

VII. FUTURE WORKS

The switch from virtual serves to container-based infrastructures in large-scale computing infrastructures allows for faster and more efficient software deployment. Fog computing systems, on the other hand, are generally made up of incredibly tiny machines like Raspberry PIs, and even launching a simple Docker container might take several minutes. The slowest of the three basic resources: network connection, CPU, or disk I/O, now affects deployment time, depending on the hardware. As hardware improves, the bottleneck may shift from one to the other. Docker-pi, on the other hand, will make the most of all available hardware to the greatest degree feasible, regardless of the system's characteristics. Our findings could aid practitioners and academics in making better decisions about how to set up cloud infrastructure and big data applications for optimal performance and resource use. In the future, we will examine many more features of the container and virtual machine environments as part of our research. Another approach is to investigate how changes in image file structure and design effect application performance across containers and virtual machines.

VIII. CONCLUSION

Docker is a container-based software development platform that is free and open source. Before you can comprehend Docker, you must first grasp the concept of containers. Containers are a "lightweight, stand-alone, executable piece of software that contains everything essential to run it," according to Docker. Docker can operate on both Windows and Linux systems since containers are platform agnostic. Docker may even be run within such a virtual machine if necessary. Docker's main goal is to enable client-server applications to operate in a distributed environment. Docker seems to perform well in Bonnie++ and the psutil performance tool, and its performance may be comparable to that of a bare-metal operating system. Docker promises to be faster than a virtualized node, however this has yet to be shown in practice. We expect the Docker community to address the main security problems in Docker container technology in the future.

REFERENCES

Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In *2015 international conference on control communication & computing India (ICCC)* (pp. 697-700). IEEE.

Bella, M. R. M., Data, M., & Yahya, W. (2018, November). Web server load balancing based on memory utilization using Docker swarm. In *2018 International Conference on Sustainable Information Engineering and Technology (SIET)* (pp. 220-223). IEEE.

Moniruzzaman, A. B. M., Waliullah, M., & Rahman, M. (2014). A High Availability Clusters Model Combined with Load Balancing and Shared Storage Technologies for Web Servers. *arXiv preprint arXiv:1411.7658*.

Sallou, O., & Monjeaud, C. (2015). Go-Docker, A batch scheduling system with Docker containers. IEEE International Conference of Cluster Computing, pp. 514-515.

Villari, M., Fazio, M., Dustdar, S., Rana, O., & Ranjan, R. (2016). Osmotic Computing: A New Paradigm for Edge/Cloud Integration, IEEE Cloud Computing, 3 (6), 76–83.

Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In *2015 international conference on control communication & computing India (ICCC)* (pp. 697-700). IEEE.

Julian, S., Shuey, M., & Cook, S. (2016, July). Containers in research: initial experiences with lightweight infrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale* (pp. 1-6).

Docs, D. (2018). Swarm mode key concepts. *Disponıvel em https://docs. docker. com/engine/swarm/key-concepts*.

Fazio, M., Celesti, A., Ranjan, R., Liu, C., Chen, L., & Villari, M. (2016). Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing*, *3*(5), 81-88.

Chae, M., Lee, H., & Lee, K. (2019). A performance comparison of linux containers and virtual machines using Docker and KVM. *Cluster Computing*, *22*(1), 1765-1775.

Huang, W., Liu, J., Abali, B., & Panda, D. K. (2006, June). A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing* (pp. 125-134).

Docs, D. (2018). Swarm mode key concepts. *Disponıvel em https://docs. docker. com/engine/swarm/key-concepts*.

Ahmed, A., & Pierre, G. (2020). Docker-pi: Docker container deployment in fog computing infrastructures. *International Journal of Cloud Computing*, *9*(1), 6-27.

Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In *2015 international conference on control communication & computing India (ICCC)* (pp. 697-700). IEEE.

Chung, M. T., Quang-Hung, N., Nguyen, M. T., & Thoai, N. (2016, July). Using docker in high performance computing applications. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)* (pp. 52-57). IEEE.

Bella, M. R. M., Data, M., & Yahya, W. (2018, November). Web server load balancing based on memory utilization using Docker swarm. In *2018 International Conference on Sustainable Information Engineering and Technology (SIET)* (pp. 220-223). IEEE.

Joy, A. M. (2015, March). Performance comparison between linux containers and virtual machines. In *2015 international conference on advances in computer engineering and applications* (pp. 342-346). IEEE.

Data, M., Luthfi, M., & Yahya, W. (2017, November). Optimizing single low-end LAMP server using NGINX reverse proxy caching. In *2017 International Conference on Sustainable Information Engineering and Technology (SIET)* (pp. 21-23). IEEE.

Shirinbab, S., Lundberg, L., & Casalicchio, E. (2020). Performance evaluation of containers and virtual machines when running Cassandra workload concurrently. *Concurrency and Computation: Practice and Experience*, *32*(17), e5693.

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, *49*(1), 71-79.

Morabito, R. (2016, April). A performance evaluation of container technologies on internet of things devices. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (pp. 999-1000). IEEE.

Dua, R., Kohli, V., Patil, S., & Patil, S. (2016, December). Performance analysis of union and cow file systems with docker. In *2016 International Conference on Computing, Analytics and Security Trends (CAST)* (pp. 550-555). IEEE.

Goldberg, R. P. (1974). Survey of virtual machine research. *Computer*, *7*(6), 34-45.

Raho, M., Spyridakis, A., Paolino, M., & Raho, D. (2015, November). KVM, Xen and Docker: A performance analysis for ARM based NFV and cloud computing. In *2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)* (pp. 1-8). IEEE.

Kozhirbayev, Z., & Sinnott, R. O. (2017). A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, *68*, 175-182.

Preeth, E. N., Mulerickal, F. J. P., Paul, B., & Sastri, Y. (2015, November). Evaluation of Docker containers based on hardware utilization. In *2015 international conference on control communication & computing India (ICCC)* (pp. 697-700). IEEE.

Chung, M. T., Quang-Hung, N., Nguyen, M. T., & Thoai, N. (2016, July). Using docker in high performance computing applications. In *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)* (pp. 52-57). IEEE.

Zhang, Q., Liu, L., Pu, C., Dou, Q., Wu, L., & Zhou, W. (2018, July). A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (pp. 178-185). IEEE.

Ahmed, A., & Pierre, G. (2018, July). Docker container deployment in fog computing infrastructures. In *2018 IEEE International Conference on Edge Computing (EDGE)* (pp. 1-8). IEEE.

Shirinbab, S., Lundberg, L., & Casalicchio, E. (2020). Performance evaluation of containers and virtual machines when running Cassandra workload concurrently. *Concurrency and Computation: Practice and Experience*, *32*(17), e5693.

Chae, M., Lee, H., & Lee, K. (2019). A performance comparison of linux containers and virtual machines using Docker and KVM. *Cluster Computing*, *22*(1), 1765-1775.