INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Performance evaluation of virtual cloud labs using hypervisor and container

Aalaa Galal A. Elbelgehy[1]*, Samir AbdElrazek[2], Hazem M. El-Bakry[3], A M Riad[3]

1 Demonstrate, Information Systems Department, Faculty of Computers and Information, Kafr Elsheikh University, Kafr Elsheikh, 33511, Egypt. Tel.: +20112150208
2 Associate Professor, Information Systems Department, Faculty of Computers and Information, Mansoura University, Mansoura, 35516, Egypt
3 Professor, Information Systems Department, Faculty of Computers and Information, Mansoura University, Mansoura, 35516, Egypt

OPEN ACCESS

Received: 21.11.2020
Accepted: 05.12.2020
Published: 30.12.2020

Citation: Elbelgehy AGA, AbdElrazek S, El-Bakry HM, Riad AM (2020) Performance evaluation of virtual cloud labs using hypervisor and container. Indian Journal of Science and Technology 13(48): 4646-4653. https://doi.org/10.17485/IJST/v13i48.2085

* **Corresponding author**.

Tel: +20112150208
Alaa_galal@fci.kfs.edu.eg

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: To measure the performance of docker swarm technology in virtual labs. **Methods :** The virtual laboratory is developed as a group of four system machines (VMs) on the same host computer as a cluster. The simulation depends on Linux OS, VirtualBox, Docker Swarm, Nginx, and Redis tools. Visualizing the tracing process by using portainer. **Findings:** The performance analysis of building virtual labs and running six main educational services using docker swarm virtualization technology are explained in detail. The experimental results have shown that the maximum utilization of the central processing unit (CPU) has reached 13% only for the nodes, 11% for the services, and 1% for the container, which considered very efficient in terms of processing. Moreover, the results have proved the effectiveness of the docker swarm in terms of memory usage since the maximum memory usage of nodes reached 101 MB, 103 MB for Container, and only 2% for each service. Additionally, the maximum network transition has reached (941 Bps) for service. **Novelty/Applications:** Building Cloud Virtual Labs enable students to connect remotely to the virtual machine at anytime and anywhere. Also, these labs enable instructors to trace the students' progress and manage the evaluation process.

Keywords: Container; cloud; docker; hypervisor; orchestration; swarm

1 Introduction

Lately, the virtualization term means talking about hypervisor-based virtualization. However, in recent years container-based virtualization became mature and especially docker, which gained a lot of attention. Shared computers can be provided via cloud computing over the internet. The users of cloud computing serve their data in third-party data centers although the data are far from the user. Virtualization does a vital part in both data centers and the cloud. Virtualization can be performed simply by integration by running many virtual machines simultaneously on the same

machine[1,2]. In recent years, Google and IBM are from a few open cloud computing systems that do not utilize hypervisors, but containers[3,4]. The reason is that each hypervisor has a higher limit for many Virtual Machines that could be operated on. Although, most of these applications do not even require half of the resources allocated to the VM by the CPU. This leads according to the improvement in Linux containers where only those sources required with the aid of the choice of the features stay used. In container-based systems, applications participate an operating system, therefore these implementations can be essentially smaller in estimation, otherwise hypervisor[5,6].

Container-based virtualization is an alternative technology to virtual machines and is rapidly replacing them in the cloud environment[7,8]. A container can be considered a small and separated virtual environment, which incorporates a set of particular conditions essential to run a particular application. Many insights are from the recently published reports that showed hybrid cloud appropriation developed three times within the final year, expanding from 19% to 57% of organizations studied[9]. In[10], observed that the containerized environment to run intensive Spark shuffle applications is not good at performance. Otherwise, the experimental results are good for using Spark with Docker for intensive calculation and copying applications and intermediate storage controllers with different virtualization frameworks. In[11], compared two virtual technologies between the virtual machine that uses Xen and the container operating system that uses VServer with the result could be a container-based framework that has twice the execution of hypervisor-based frameworks for server-type workloads. In[12], presented a novel educational platform that differs from existing CTF platforms by having superior availability, added up to computerization, the perceptibility of participants' behavior and a high degree of authenticity. However, the cost was too much because of the numbers of virtual machine[13–15], since the High Performance Computing (HPC) applications were tested, LXC demonstrates to be the most suitable of the container-based systems for HPC. Since containers are increasingly lighter and agile contrasted with virtual machines, you might have the option to run six to multiple times a bigger number of compartments than virtual machines on a similar equipment. In[16,17] illustrated a method for determining the mean value of overhead classes of server consolidation based on performance benchmarking and monitoring techniques and the result differ from the type of virtualization. In[18], introduced that the Linux namespaces, initially created by IBM, wrap a lot of framework assets and present them to a procedure to cause it to give the idea that they are committed to that procedure. In[19], developed an application on a virtual machine to monitor the performance of their uploaded applications deployed on Amazon Web Services (AWS) and Azure though RAM, Disk Size, and price. In[20], applied benchmark CPU and system tests to look at the exhibition of KVM, uniqueness HPC, Docker and LXC with the local instance of very good quality server equipment. Containerized virtualization is appropriate for overseeing microservices-based applications since it serves to rapidly dispatch and end the container for fast scalability, though VM-based virtualization requires impressive opportunity to begin and end the VM[21]. The authors in[22], presented that containers differ from a virtual system that has virtual machines. The virtual machine has a complete operating system that runs independently of the physical resources that are virtualized on those available on the host. In[23], showed the improvements performance of containers in MPI collective data movements by using two stage-methods compared with virtual cluster. In[24], a high-performance OpenStack-based Docker integration scheme was introduced, which implements a container management service called Yun. Docker solves one of the main problems facing system administrators and developers for years. The problem most of the time, the mismatch copy of some library or a few packages cannot be installed. This is where the docker steps in, and solves this problem forever, by making the image of the application complete, with all its dependencies and shipping it to the desired environment or server required. Container allowed us to sort out many challenges: pulling, packing, isolating and making application portable across systems with almost no burdens. Facing issues with tracking down dependencies, scaling your application, and updating individual components without affecting the entire application. The goal of containers is to write once, then run on any cloud.

This is the main issue we confront suppose you have one application server which can serve "x" customers. On the off chance that you have to serve double the measure of customers, you may expand the assets on the server or make another occasion of the application server that is stack offset with the first server. Docker accompanies an approach to make a "swarm" of group hubs by making that required number of utilization holders. It enables you to send any number of use servers over any number of hosts with a couple of directions. And afterward, you can downsize simply.

2 Materials and Methods

2.1 Docker

Docker is a Future Virtualization technology. Docker is an open-source framework that automates the implementation of applications in lightweight and portable containers and is one of the world's leading software container platforms. Containers do this by providing a complete runtime environment in a package, which includes the application, in addition to all dependencies, libraries and other binary files, and the configuration files necessary to run it[25]. Docker intends to address the difficulties of

virtualization resources, velocity, and execution in the software development process. The Docker container permits developers to perform applications and services by utilizing the technology or language most convenient to them [26].

Docker consists of:

1. The Docker Engine: The light and powerful combined open source containerization technology with a workflow to build and containerize your applications.
2. Docker Hub: The Software as a service (SaaS) service to share and manage applications. It is a free public image registry for Store and build images. It is essentially a client-server application with three significant parts: a command-line interface (CLI), a REST API and a server.

The server is a demon process called dockerd. Listen to Docker API requests and manage all Docker assets, such as images, containers, networks, and volumes. A Docker image is characterized in a Docker file that is a book document that contains Docker orders to construct an image from a base image that could be from a local machine or an online image registry. With Docker-Machine, machines can be provisioned, both virtual and physical, on a number cloud platforms as well as bare metal machines to run Docker containers.

2.2 Docker swarm mode

A clustering tool is software that allows an operator to talk to a single endpoint and to command and orchestrate a set of resources, in our case containers. Instead of manually distributing workloads (containers) on a cluster, a clustering tool that will decide where to start jobs (containers), how to store them, when to eventually restart them. The operator needs to only configure some behaviors, decide the cluster topology and size, tune settings, and enable or disable advanced features. Swarm can monitor the availability and resources usage of nodes using a swarm Master (manager). A Docker Swarm is a gathering of servers (physical or, for my situation, virtual) that are organized as a network and arranged to act as a single unit. Docker manages the network between servers in the swarm in a straight forwardly way, so it can have a multi-container application with the primary segments on one server, the database on another, the Redis cache on a third party, and so on. On the off chance that Docker sees a server disconnect, regardless of whether arranged or not, it will quickly move the containers from the failed server to those that are as yet dynamic, guaranteeing high accessibility of its services.

At the point when the Docker engine works in swarm mode, the administrator nodes execute the raft consensus algorithm to deal with the status of the global cluster. The most important part of the swarm are its remote APIs; it is 100% compatible with all versions of the Docker engine. This layer should allow for sharing resources, scheduling tasks and treating many processes in execution as a unified, scalable and well-behaved solution in all workloads. Swarm consequently attempts to recoup when containers or nodes crashed. Docker Swarm allows you to scale container applications by operating them in any number of instances on any number of nodes in your network. One of the most thing that Docker Swarm solve is orchestration.

2.3 Orchestration

Container Orchestration refers to the automated arrangement, coordination, and management of software containers. It can manage a large collection of container easily, that allows deploying Docker containers on clusters and automate the container life cycle. The Master's goal is to increase the utilization of the available resources. Systems manager through orchestration can perform infrastructure-related functions such as scale-up a running cluster, replacing a service or, fetch user application code from remote repository. Scheduling is a key component of container orchestration, and helped maximize the workload's availability, whilst making maximum use of the resources available for those workloads. Automated scheduling removes the need for manual deployment of services, which would otherwise be an onerous task, especially when those services require scaling up and down horizontally. Load Balancing used to distribute traffic evenly across the tasks in your service to prevent these failures. Nodes can take one of the following values of policies:

- Active: enables HA for instances and enable automatically restarting the VM if it stops unexpectedly due to one of these cause node power outage, node network failure and hypervisor failing for any reason.
- Monitor: monitors the state of the instance and notifies administrators if the instance stops, fails, or becomes unreachable.

3 Proposed simulation

This paper aims to measure the performance of docker swarm technology in virtual labs. For achieving our purpose, the proposed virtual labs are built according to the frame-work in Figure 1.

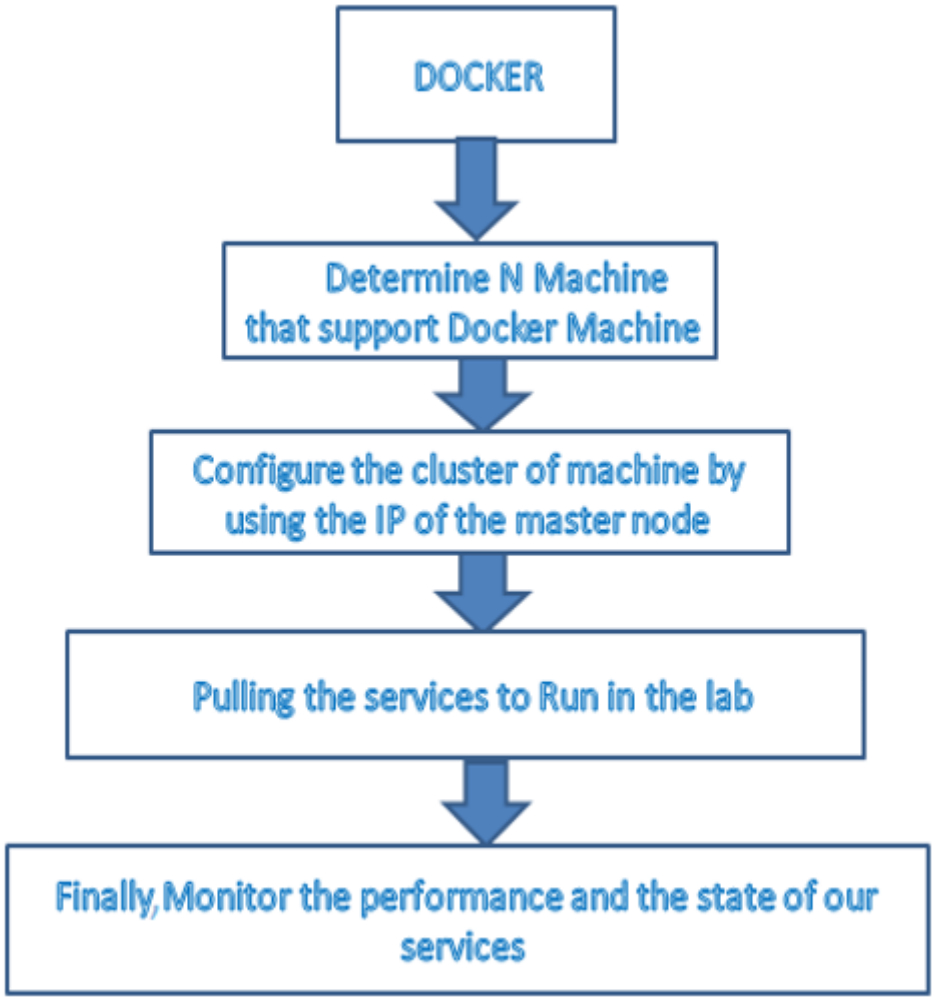

Fig 1. The frame-work of building virtual labs

This simulation depends on Linux OS, VirtualBox, Docker Swarm, Nginx, and Redis tools. This virtual laboratory is developed as a group of four system machines (VMs) on the same host computer as a cluster. Nevertheless, the corresponding laboratory can be performed in any of the Docker-compatible clouds by replacing the name of the controller to the name of the desired cloud[27]. A service could be a high-level concept relating to a collection of jobs to be implemented by workers. Services are deployed in a Docker, once the container is started or stopped for this selected service, they can be scaled or even replaced. The easiest way to implement an application service is to run it on a single server, similar to how you would run your development environment. If you want to expand your application, you can run Compose applications in a Swarm cluster.

In our simulation, VirtualBox be used as environment otherwise the Docker supported various clouds by replacing the driver name to the desired cloud according to some types like Digital Ocean, Amazon Web Services, Microsoft Azure, Google Cloud Platform and IBM Cloud. Docker-compose is a file following the YAML data standard and it defines a number of services run by Docker Swarm. Each service is based on an image, which in this case corresponds to the application images. Each service has a number of properties that can be specified through the compose file, such as environment variables, container replicas, volumes, constraints and networks[28,29].

First, to create a machine by installing a Docker-machine. For creating a Swarm cluster of nodes, Fist of all using Docker-machine that enabled creating many machines. To begin, four Docker hosts with Docker-Machine must be created. Docker-Machine automates these steps with one command instead of manually creating a Linux VM, generating and uploading certificates, and logging into it via SSH, and installing and configuring the Docker Daemon. The first step is to create a Docker

Swarm Manager through specific IP of specific machine which belongs to the Master node, which can be used as a discovery token. As running many Docker-machines for various projects, we might face problems cause of the dynamic assignments of IPs on startup. For Creating a Docker-Machine with a fixed IP, we must use this command "ifconfig eth1 192.168.99.100 netmask 255.255.255.0 broadcast 192.168.99.255 up" | docker-machine ssh Manager sudo tee /var/lib/boot2docker/bootsync.sh > /dev/null. We must stop and start the Machine, then regenerate-certs of each machine. The token was generated by using the Docker hub registry. It enabled us to connect all the nodes within the Swarm cluster by adding this token to other machines (slave nodes), which determined as worker node otherwise manager node. Docker Swarm is made up of one swarm master node and any number of slave nodes. To make the environment fully functional, the standard Nginx image will be run. It is a server officially available from the Docker hub registry. The Nginx exposed our service on port 80. The Redis image can also be run in a Docker cluster working as a DB with the same steps as Nginx.

We can specify the number of containers (or instances) to launch. This is specified via the replicas parameter. Nodes may be active, drained and paused. When the node is active, it is ready to accept tasks from the Master Node. Now, the virtual lab is ready with 4 different IP machines running the services. An application, no matter how good its user interface, will not claim market share if its response time is slow. That's why we spend so much time improving the performance and scalability of an application as its user base grows.

4 Results

For the purpose of managing and tracking the virtual labs, the "portainer" management soft-ware is used. Portainer is a lightweight management simple User Interface (UI), especially for Dockers visualization. Portainer is considered a simple single container. It is compatible with the Docker engine or even Docker swarm mode that can run on any one of them. The interface of portainer is a web service that builds visually and represents metric graphs based on time-series databases. A collection of predefined dashboards contains graphs on particular endpoints. It displays only the existing information about nodes, disks, interfaces, and so on; according to a particular time frame. In our experiments, Portainer is used for debugging containers and management not only Docker hosts but also swarm cluster[30]. The characteristics of the experimental environment are described below in Table 1.

Table 1. Experimental environment characteristics

Items	Characteristics
Processor	Intel(R) Core(TM) i3-2328M CPU @ 2.20GHz
Number of CPU, threads/core, cores /socket	4, 2, 2
RAM	6GB @1600MHz
Disk (file system type)	ATA DISK HDD 480GB (ext4)
Platforms	Ubuntu 18.04 , VirtualBox Graphical User Interface Version 5.2.34, Docker v 1.18.3
Monitoring tools	Grafana 5.1, Prometheus 2.2.1, cAdvisor 0.29.0

The performance metrics of service that installed in the virtual cluster lab were measured by using three open-source performance tools Grafana, Prometheus, and cAdvisor. Grafana is open-source software that visualizes and analytics data stored. Through query, visualize, alert on, and explore the metrics no matter where are stored. Prometheus is an open-source system monitoring and alerting toolkit that enables cAdvisor extracting the data sampled every 30 seconds. cAdvisor (Container Advisor) is a daemon that runs for every container, saves resource isolation parameters, extracts histograms for historical resource usage, and network statistics[31,32]. The CPU utilization, Memory Usage and Network transmission rate are our main evaluation metrics for measuring virtual lab performance. Figure 2 shows the CPU utilization for all nodes, container and the installed services. The results show the efficacy of docker swarm since only 13% CPU usage is required for nodes and 2% for Container. The memory usages are shown in Figure 3, where only 101MB are required for running single node and 2MB for running service. Finally, the network transmission rates are shown in Figure 4 where the maximum transmission rate was 941 Bps. The use of CPU usages is simple comparing with virtualization technique and memory usage also. It is can be concluded that the performance of Dockers swarm is very motivational and recommended for the field of E-learning in building virtual labs. The performance measurements metrics of container displayed how is a lightweight compared with virtualization and saves the resources utilized as possible. Finally, a scientific comparison between a virtual machine (Hypervisor) and Docker (Container) are described below in Table 2 in terms of security, speed, OS and etc.

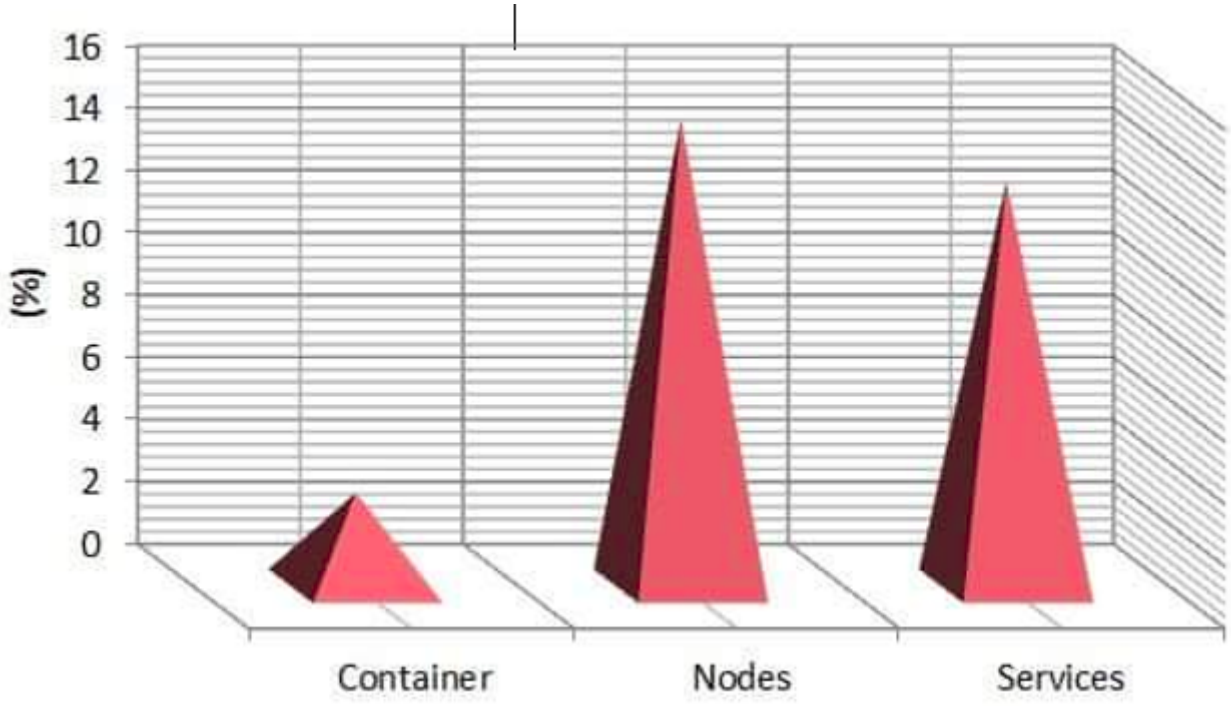

Fig 2. CPU utilization of simulated environment

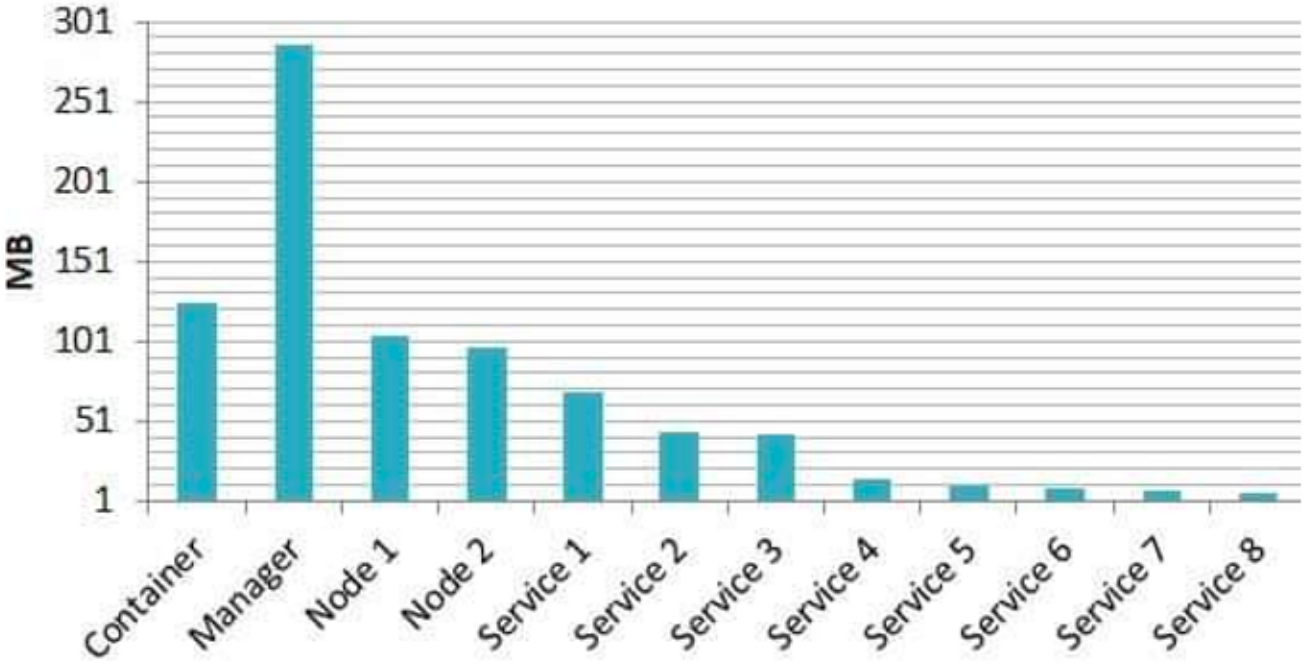

Fig 3. Memory usage of simulated environment

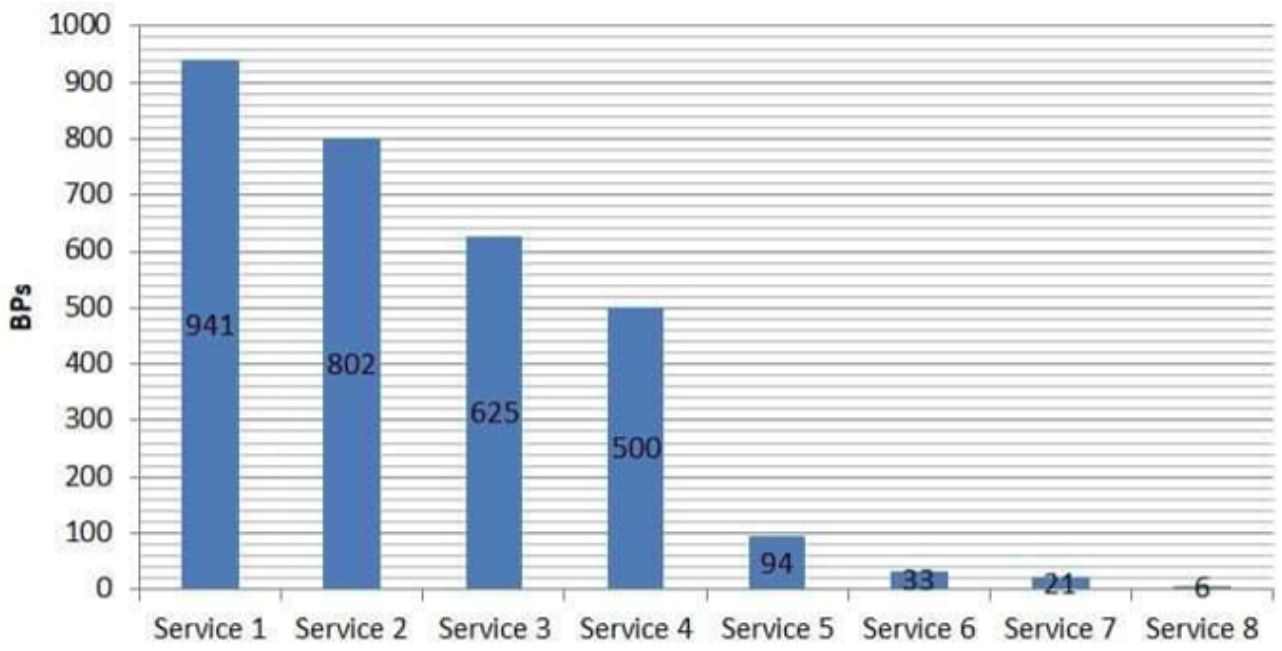

Fig 4. Network transmission rates of installed services

Table 2. Major differences between hypervisor and container.

	Hypervisor (Virtual Machine)	Container (Docker)
OS	Needs an Hypervisor and a full OS inside	Talks to the host kernel
Foot-Print	Bigger Footprint (RAM and Storage space	Smaller Footprint (No RAM and differential storage
Storage Space	VMs consumes storage space for each instance	Consumes very less space
Weight	Heavier	Lightweight
Startup time	It is in the order of minutes	It is in the order of seconds
Deployment	Deployment is tough	Easy Deployment with minimal requirements
Speed	Slower	Faster
Security	Security issues of running OS	Security issues limited to Applications

5 Conclusion and Future work

With the outbreak of COVID-19, The E-learning has played a major role for combating this paradigm. This study contributes in building a virtual laboratory for the distributed software development process on various clouds using docker swarm. It aims to measure the potential performance of docker swarm for building virtual labs. This virtual Labs tools was based on Docker Swarm, VirtualBox, Linux OS, Nginx, and Redis. The significant restriction of virtualization innovation is bottlenecks in I/O serious applications. The experimental results explained that the container-based virtualization system has better CPU and I/O performance on account of its capacity to discharge utilized unused and assets and work in isolation. Occasionally, Docker supported by Portainer as a simple UI and it can visualize the commands of the Swarm cluster environment. This simulation can help students through learning and instructor to keep tracing the tasks that the students do. Although, in many big companies managers can also follow the progress of distributed tasks. The enhancements are applied to choose the most effective criteria to achieve and improve the performance and velocity of DevOps. Docker swarm only require 300MB and 13% CPU utilization for running virtual lab, which is very wealthy in terms of minimizing resource utilization. Since containerization is an important issue for gaining the best achievements in enterprises, the future work will focus on enhancing the performance used criteria of microservices and serverless applications.

References

1) Ahmed M, Zomaya AY. An automated lightweight framework for scheduling and profiling parallel workflows simultaneously on multiple hypervisors. *Cloud computing*. 2017;p. 26.
2) Bharath MB, Ashoka DV. AAAS - framework in large virtualized environment. *Indian Journal of Science and Technology*. 2019;12(4):1–8.
3) Polenov M, Guzik V, Lukyanov V. Hypervisors comparison and their performance testing. In: Computer science on-line conference. Cham. Springer. 2018;p. 148–157. Available from: https://doi.org/10.1007/978-3-319-91186-1_16.
4) Desai PR. A survey of performance comparison between virtual machines and containers. 2016.
5) Yadav AK, Garg ML. Docker containers versus virtual machine-based virtualization. In: Emerging Technologies in Data Mining and Information Security . Singapore. Springer. 2019;p. 141–150. Available from: https://doi.org/10.1007/978-981-13-1501-5_12.
6) El-Razek SM, El-Bakry HM, El-Wahed WF, Mastorakis N. Collaborative virtual environment model for medical E-learning. In: and others, editor. Proceedings of the 9th WSEAS international conference on Applied computer and applied computational science. 2010;p. 191–195.
7) Elhoseny H, Elhoseny M, Abdelrazek S, Riad AM, Hassanien AE. Ubiquitous smart learning system for smart cities. In: and others, editor. 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE. 2017;p. 329–334.
8) El-Razek SM, El-Bakry HM, El-Wahed WF, Mastorakis N. Collaborative virtual environment model for medical E-learning. In: and others, editor. Proceedings of the 9th WSEAS international conference on Applied computer and applied computational science. 2010;p. 191–195.
9) L C. State of Cloud Adoption and Security. 2017.
10) Bhimani J, Yang Z, Leeser M, Mi N. Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In: and others, editor. 2017 IEEE High Performance Extreme Computing Conference (HPEC). IEEE. 2017;p. 1–7. Available from: https://doi.org/10.1109/HPEC.2017.8091086.
11) Yadav RR, Sousa ET, Callou GR. Performance comparison between virtual machines and docker containers. *IEEE Latin America Transactions*. 2009;16(8):2282–2290. Available from: https://doi.org/10.1109/TLA.2018.8528247.
12) Panum TK, Hageman K, Pedersen JM, Hansen RR. Haaukins: A Highly Accessible and Automated Virtualization Platform for Security Education. *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT*. 2019;2161:236–238. Available from: https://doi.org/10.1109/ICALT.2019.00073.
13) Allah NM, Qureshi MB, F J, Alrashed S, Taheri J. Deployment of real time systems in the cloud environment. 2020. Available from: https://doi.org/10.1007/s11227-020-03334-7.
14) Ismail A, Abdlerazek S, El-Henawy IM. Big data analytics in heart diseases prediction. *Journal of Theoretical and Applied Information Technology*. 2020;98(11).
15) Arango C, Dernat R, Sanabria J. Performance evaluation of container-based virtualization for high performance computing environments. *Revista UIS Ingenierías*. 2019;18(4):31–42. Available from: https://dx.doi.org/10.18273/revuin.v18n4-2019003.
16) Bermejo B, Juiz C. On the classification and quantification of server consolidation overheads. 2020.
17) Dhule C, Shrawankar U. POF-SVLM: pareto optimized framework for seamless VM live migration. 2020.
18) SerdarYegulalp. What is Docker? . 2019. Available from: https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html.
19) Saswade N, Bharadi V, Zanzane Y. Virtual machine monitoring in cloud computing. *Procedia Computer Science*. 2016;79:135–142. Available from: https://dx.doi.org/10.1016/j.procs.2016.03.018.
20) Kovács A. Comparison of different Linux containers. *2017 40th International Conference on Telecommunications and Signal Processing*. 2017;p. 47–51. Available from: https://doi.org/10.1109/TSP.2017.8075934.
21) Abdullah M, Iqbal W, Bukhari F. Containers vs virtual machines for auto-scaling multi-tier applications under dynamically increasing workloads. In: International Conference on Intelligent Technologies and Applications. Singapore. Springer. 2018;p. 153–167.
22) Singh S, Singh N. Containers & Docker: Emerging roles & future of Cloud technology. *2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. 2016;p. 804–807. Available from: https://doi.org/10.1109/ICATCCT.2016.7912109.
23) Huang D, Lu Y. Improving the efficiency of HPC data movement on container-based virtual cluster. *CCF Transactions on High Performance Computing*. 2020;10:1–4.
24) Yang S, Wang X, Wang X, An L, Zhang G. High-performance docker integration scheme based on OpenStack. World Wide Web. . 2020.
25) Ismail A, Abdlerazek S, El-Henawy IM. Development of smart healthcare system based on speech recognition using support vector machine and dynamic time warping. *Sustainability*. 2020;12:2403. Available from: https://dx.doi.org/10.3390/su12062403.
26) Khazaei H, Ravichandiran R, Park B, Bannazadeh H, Tizghadam A, Leon-Garcia A. Elascale: autoscaling and monitoring as a service. arXiv preprint arXiv:1711.03204. 2017.
27) Naik N. Building a virtual system of systems using Docker Swarm in multiple clouds. *2016 IEEE International Symposium on Systems Engineering (ISSE)*. 2016;p. 1–3. Available from: https://doi.org/10.1109/SysEng.2016.7753148.
28) . 2019. Available from: https://docs.docker.com/compose/compose-file/.
29) Poojara SR, Dharwadkar NV, Ghule V. Performance benchmarking of hypervisors-a case study. *Indian J Sci Technol*. 2017.
30) Portainer, Deployment portainer 1.15.3. 2019. Available from: http://portainer.readthedocs.io/en/stable/deployment.html#.
31) Schörgenhumer A, Kahlhofer M, Grünbacher P, Mössenböck H. Can we Predict Performance Events with Time Series Data from Monitoring Multiple Systems. *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering*. 2019;p. 9–12. Available from: https://doi.org/10.1145/3302541.3313101.
32) Raju IRK, Varma PS, Sundari MVR, Moses GJ. Deadline aware two stage scheduling algorithm in cloud computing. *Indian Journal of Science and Technology*. 2016;9(4). Available from: https://dx.doi.org/10.17485/ijst/2016/v9i4/80553.