International Journal of Innovative Research in Engineering
Volume 3, Issue 2 (March-April 2022), PP: 65-70.
www.theijire.com
ISSN No: 2582-8746

Survey on Comparison of Application Deployment Using Docker Containers and Virtual Machines

Sheik Harris F[1], Harshavardhan T[2], Manoj A J[3]
[1,2,3]*Information Science and Engineering, Kumaraguru College of Technology, Tamilnadu, India.*

How to cite this paper: Sheik Harris F[1], Harshavardhan T[2], Manoj A J[3] , "Survey on Comparison of Application Deployment Using Docker Containers and Virtual Machines", IJIRE-V3I02-65-70.

Abstract: *In the recent past, building up an application has been a drastic change from monolithic to microservice. Developing the microservice application also increased the usage of containers. Most enterprises shifted their long old running monolith application to microservice. Microservices when deployed in containers give high scalability and performance. A container is a form of operating level virtualization which helps you to deploy applications along with their dependencies. The most popular container technology used today is docker which makes it easier to create and manage containers. Docker SWARM is a container orchestrator technology used for scaling containers which comes along with the docker installation. But the most widely used container orchestrator is Kubernetes which is developed by Google. Deploying the microservices application in a container and managing with Kubernetes is more powerful than deploying a monolithic application in a virtual machine.*

Key Word*: microservices architecture; Virtual machine; Containers; Docker; Kubernetes; docker swarm.*

I.INTRODUCTION

Virtual Machine is the virtualization of a computer system. VMs are installed above the host operating system and hypervisors act as an intermediate between VM and host OS. Each VM can have different guest operating systems sharing common system resources underlying hardware.

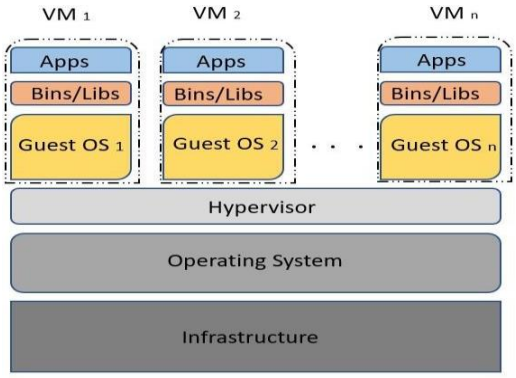

Figure 1: Virtualization Architecture

The traditional way of deploying the application is done in VMs. The difficulty faced is VM size is large and it takes more time to restart after shutdown. To overcome these difficulties, Container technology is developed. A Container is a lightweight silo and isolated environment for running applications on the host OS. Containers are built on top of the host Operating system kernel and contain lightweight OS APIs and services that run in user mode.

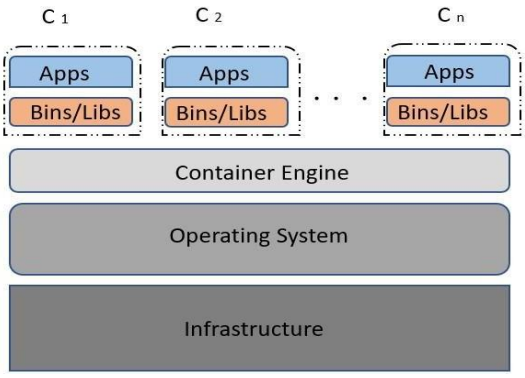

Figure 2: Container Architecture

Deploying the application service in containers can increase performance and availability for the application. As containers are lightweight they can be scaled up or down very quickly than VMs. When compared to VMs, containers are less secure, but containers are very sensitive so a less vulnerable threat makes the container shutdown and a new container is instantiated. Containers have no downtime so they can be used for applications that drive the traffic infrequently.

The most commonly used container management tool is docker. Docker is an open-source platform for shipping, running, and developing applications isolated from infrastructure. Docker provides tools for managing the lifecycle of containers. Docker is based on client-server architecture. The Docker client notifies the Docker daemon which performs the shipping, building, and running of Docker containers. Either the client or daemon can be in the same system or remote system.

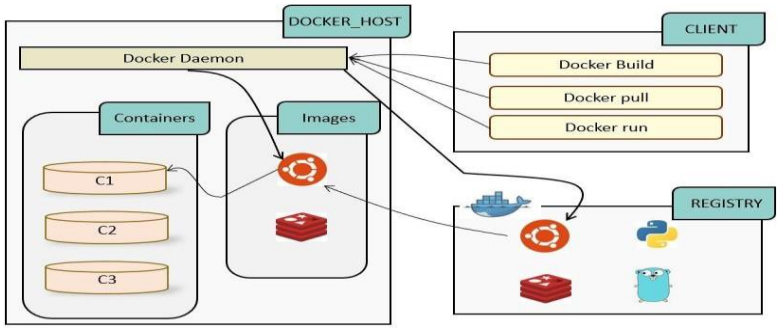

Figure 3: Docker Client-Server Architecture

The docker container can be created using docker images. A docker image is a read-only template with instructions for creating a docker container. To create an image, we need to create a Dockerfile. Each instruction in the Dockerfile develops a layer in the image. Whenever a modification is made to the Dockerfile only the modified layer is rebuilt. The Docker registry is the place where Docker images are stored. Docker Hub is the public registry. To manage and run all the containers, an operational engineer is assigned. But when the number of tasks of creating and managing the container increases exponentially a single person cannot do it. To overcome such a situation a container orchestrator tool is built, for example, Docker SWARM and Kubernetes.

Kubernetes is an open-source container orchestrator developed by Google to help manage containerized applications with different deployment environments. It gives high availability, scalability, and disaster recovery.

The main components of Kubernetes are pods, service, config-map, secret, ingress, deployment, and statefulset. A pod is an abstraction of the actual container, each pod gets its IP address. Through this IP pods communicate with each other. But when a pod fails a new pod is created and a new IP address is assigned, the other pods which were connected before the failure don't know the new IP. To overcome this, a service component is built that dynamically connects pods. Mainly there are two types of services: internal service and external service. The pods with internal service can only be accessed within the cluster. The pods which are connected to external services can be accessed outside of the cluster. The external service acts as a load balancer for the autoscaling of pods. The authentication between pods is achieved through configmap and secret components. For a stateless application, the pod is created using a YAML file called deployment which contains all the instructions to create the pod.

For a stateful application, the pod is created using a YAML file called statefulset which contains all the instructions to create the pod. The Ingress component acts as a DNS resolver for the applications.

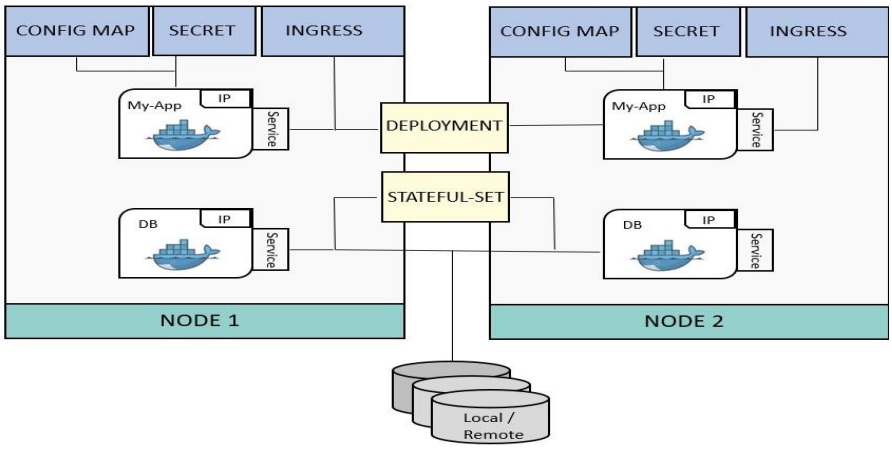

Figure 4: Kubernetes Components

The Kubernetes architecture consists mainly of two nodes namely worker node and master node. Generally, a node is a physical system. The worker node is the node that does the actual work of the application. It has three processes. Firstly, the kubelet process which interacts with both container and node. It is responsible for starting the pod with a container inside. Secondly, the Kube proxy which forwards the request between pods in a protected way. Thirdly, the necessary container environment is needed to run the container.

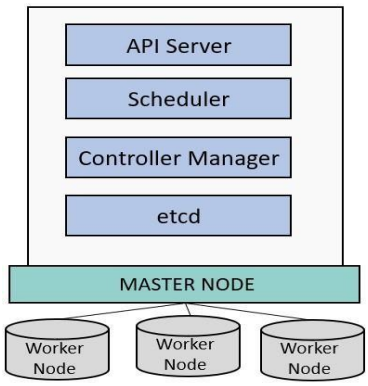

Figure 5: Master Node and Worker Node

The master node is the node that manages the worker nodes. It has four components namely, API server, scheduler, controller manager and etcd. The API server acts like a cluster gateway for authentication. The scheduler is responsible for scheduling the creation of a new pod. The Scheduler sends the request for the creation of a new pod to kubelet in the worker nodes. The Controller manager detects cluster change state. The etcd is a key-value store that stores the state of every pod in the cluster. If a pod fails, the state is recorded in the etcd, and the controller checks the etcd detects the failure and sends a request to the scheduler which in turn transfers the request to the kubelet in the worker node to create a new pod.

II. LITERATURE SURVEY

In the paper "An Introduction to Docker and Analysis of its Performances", the author compared the Docker container performance with other container technologies like KVM, Xenserver, and LXC. At the beginning of the paper, the advantage of containers over virtual machines is highlighted which is high speed, portability, scalability, rapid delivery, and density. As the docker does not use a hypervisor, and when compared to virtual machines, many containers can be run on a single host machine. The author took various papers which compared the performance of the docker with other environments and formulated a clear tabulation on it. From the tabulation, the docker's boot time is very minimal compared to KVM, and calculation speed is fast in docker. And Docker when compared with native, the overhead or wastage of resources is more in native. In conclusion, the performance of docker is much better than virtual machines as it has no guest OS and less resource overhead.[1]

In 2019, Chan-Yi published a paper where he compared the performance of a container, VM, and bare-metal using TensorFlow Image Classification across different neural networks. The experiment is conducted with three physical nodes of the same specification each for bare-metal, container, and VM. The neural networks shown were InceptionV3, ResNet-50, and AlexNet with the same batch size of 32 and same optimizer sgd. On a single instance, the network and disk I/O doesn't involve, for both VM and Container the degradation observed was less than 10%. Distributed TensorFlow is a network-bound application. For this application, the virtualization layer does cause a significant impact on the performance. Therefore, lightweight virtualization techniques like Container or even bare metal can be considered.

In conclusion, to build a cost-effective and performance-efficient cloud service for deep learning applications choosing the container over bare-metal and VM is highly recommended.[2]

In 2016, Bowen Ruan and his team wrote a paper on the performance study of containers in cloud environments. There are two types of containers: system-container (LXC) and application container (Docker), which one to use for different use cases. In the paper, they conducted a series of experiments to measure the performance gap between system containers and application containers, thereby evaluating the overhead of adding the extra VM layer between bare metal and container, and finally inspect the service quality of GKE and AWS EC2. The result concluded that system containers were more suitable for I/O-bound workload as application containers directly on the bare metal.[3]

In this paper, Sumit Maheshwari and her team together made a comparative study of virtual machines and containers for DevOps developers. They used their Apache based web service application to compare the performance of VM and Containers. The time to start and stop for VMs was comparatively higher than Container. DevOps developers should be able to choose the right virtualization technology when it comes to I/O heavy applications, the experiment showed containers provide lesser overhead performing almost equal to that of bare metal. They used a tool named netprof that performs network bandwidth benchmarking, and the VM produced high latency which is not good. The CPU performance when given a Floating-Point Operating Operation per Second (FLOPS) in a container is higher than in VMs. The paper also suggested when not to use Containers, when the security of the application is very highly concerned then using the container is not recommended as they have shared kernel space.[4]

Type	Time to Start	Time to Stop
Docker	39 ms	27 ms
VM	51,000 ms	29,000 ms

Figure 6: Restart time and Stop time Comparison

In this paper, the author pointed out the advantage of choosing containers over VMs for building up a high-performance Iaas platform for interactive applications like social media. The advantage of using Docker as a hypervisor inside Openstack is the time constraint needed to start a container inside an Openstack environment is very less than starting a KVM instance. Also, Containers within a physical machine generally do not have a separate IP address which is a very big benefit as the IPV4 address is very limited nowadays.[5]

This paper evaluates the usage of docker as an edge cloud platform. Edge computing uses a decentralized paradigm by utilizing compute, network, and storage resources closer to the user. As the container image is smaller than VMs making it is possible to launch applications in the edge much faster than VM-based appliances. The key feature to build an edge computing infrastructure is the termination of one service must not affect other services and to enable efficient management of application at the edge, service registry and service discovery are built. Docker provides these features by adding a docker registry at each edge site, thereby, the Docker daemon at each edge site can search and pull Docker images. The Docker SWARM is configured at each edge to manage multiple Docker Daemon. By default, Docker allows the remote deployment of containers through APIs. Overall Docker when used as an edge cloud platform provides elasticity, good performance, and fast deployment over a VM-based Edge computing platform. [6]

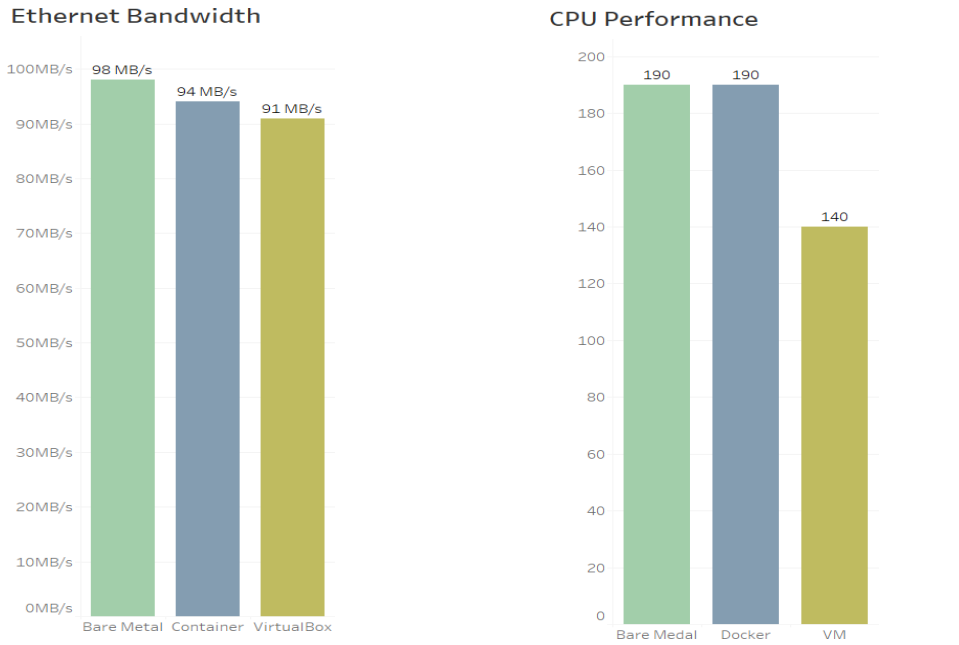

Figure 7: Bandwidth and CPU performance comparison

This paper gives the performance study on deploying a ReactJS web application using a virtual machine and docker container. The test environment considered is a Node.js server for serving a simple ReactJS application with system configuration for VM is ubuntu operating system and Oracle VM VirtualBox hypervisor and for containers Base Docker image as node:12.14.1 and Docker version 19.03.8. The result was that the start time, memory utilization, and image size were higher for the VM. Finally, CPU performance is maximum for Containers, which concludes choosing containers over VM for web application deployment is appropriate. [7]

The paper describes the use case and some case studies on the usage of Dockers. The key use cases of Docker are simple configuration which accelerates the project setup, Code pipeline Management, multi-tenancy, better disaster recovery, Debugging Capabilities and docker provide rapid deployment as there is no need of booting up an OS. The situation when to use dockers is when the application has a basic use case, and it can be built around isolated components or features. The situation when not to use dockers is for complex applications where there is a high dependency among components, when security is a critical factor, and when there is a need for multiple OS. An example case study of the usage of dockers is Spotify which is a digital music service that uses microservice architecture through container technology and IBM Control Desk software which provides IT service management entirely on the cloud using Docker containers. [8]

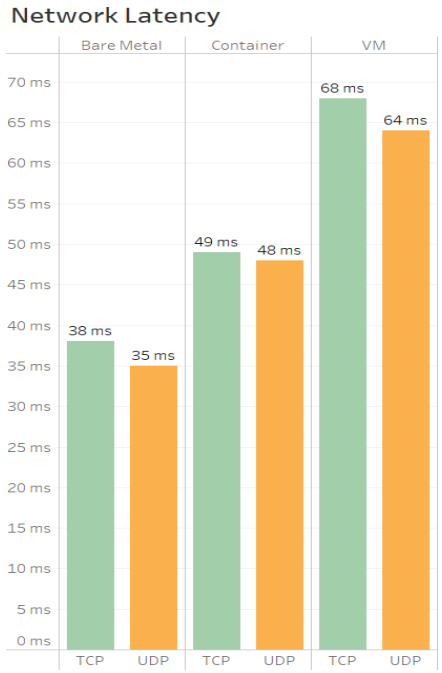

Figure 8: Network latency Comparison

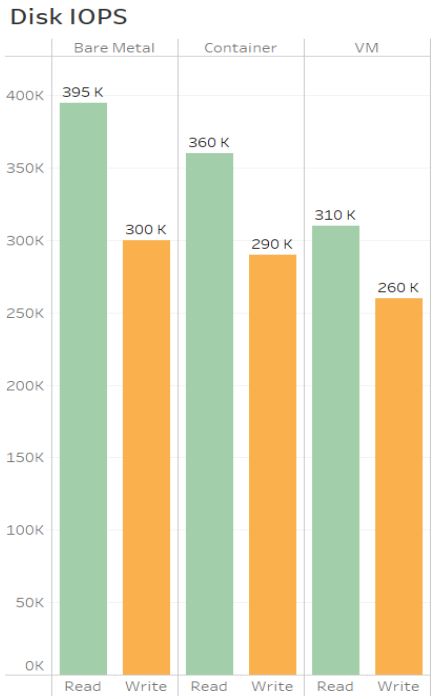

Figure 9: IOPS Comparison

In this paper, the author compares the performance of Docker SWARM and Kubernetes by implementing Walkability applications. The walkability application gives the index for measuring the walkability of the neighborhood. By using this index with other datasets like the price of the house, we can produce a better ML algorithm for predicting the price of a house in the given area. The application is deployed into the 16 nodes Docker Swarm cluster, the test file includes different multi-point input files consisting of 100, 500, and 1000 geolocation points. By adjusting the number of containers used for different points, the Docker SWARM produced an average running speed of 536s. Whereas in Kubernetes without giving the number of containers to be used as the Kubernetes is capable of auto-scaling the containers, the average running speed produced is 853s. In conclusion, Docker SWARM can be used if you can predict the load to the application beforehand, and Kubernetes is preferred if the load to the application is not predictable and also there is not much performance difference between Docker SWARM and Kubernetes. [9]

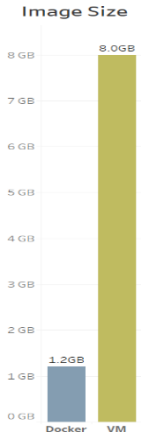

Figure 10: Image Size Comparison

The paper illustrates the key challenges based on developers, QA, deployment team while using Microservice architecture (MSA). The Microservice architecture is the design used for building applications as independent services. The key challenges in using this design are too many repositories to handle, difficult to find issues in a distributed system, internetworking of dockers, sharing base data among different services considering performance limitations, and debugging of microservice that relies on the other services can be tricky. Several experiments were done with suggestions of using MSA from developers, they concluded, there is a lack of tool or framework support for selecting third-party artifacts based on features. as well as the shortage of knowledge of the practitioners. [10]

III. CONCLUSION

Using monolithic applications can take more time and is less efficient in scaling. It is much more difficult to scale a monolithic application as it is deployed in a virtual machine, it is also costly for the business entity. Whereas splitting the application into services and deploying each service in a container make it easier for scaling. Also, it provides the advantage of scaling a particular service or feature of the application rather than scaling the entire application which is not possible in the deployment of monolithic applications. The deployment of microservices is usually done in containers. Containers are very lightweight and cost-effective when compared with virtual machines. Scaling up or Scaling down virtual machines is costlier for a business entity. Virtual Machine takes more time for restarting and it is not good for sensitive applications, whereas containers have no downtime. Virtual machines are more secure than containers, also VMs are less likely to crash. Containers are sensitive and the probability of crashing them is high. So, a small threat to the container can make them shut down which indirectly makes them secure. Deploying each service of an application to a container can give scalability to a particular feature that is driving more traffic and not to the entire application. The creation and management of the container can be done using container orchestrators like docker SWARM and Kubernetes. Compared to docker SWARM, Kubernetes stands out by providing autoscaling of containers with much less effort. Docker Swarm doesn't have in-built monitoring mechanisms, it supports monitoring only from third-party applications. In contrast, Kubernetes has built-in monitoring and also sports integration with third-party monitoring tools.

References

[1] Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi (2017), An Introduction to Docker and Analysis of its Performance, IJCSNS International Journal of Computer Science and Network Security, VOL.17 No.3 (March 2017).

[2] Chan-Yi Lin, Hsin-Yu Pai and Jerry Chou (2019), Comparison Between Bare-metal, Container and VM using TensorFlow Image Classification Benchmarks for Deep Learning Cloud Platform, Computer Science, National Tsing Hua University (2019).

[3] Bowen Ruan, Hang Huang, Song Wu, and Hai Jin (2016) ,A Performance Study of Containers in Cloud Environment. Services Computing Technology and System Lab Cluster and Grid Computing Lab School of Computer Science and Technology Huazhong University of Science and Technology (November 2016).

[4] Sumit Maheshwari, Saurabh Deochake, Ridip De, Anish Grover (2018) Comparative Study of Virtual Machines and Containers for DevOps Developers (August 2018).

[5] Alin Calinciuc, Cristian Constantin Spoiala, Corneliu Octavian Turcu, Constantin Filote (2016), OpenStack and Docker: building a high-performance IaaS platform for interactive social media applications, 2016 International Conference on Development and Application Systems (DAS) ,(May 2016)

[6] Bukhary Ikhwan Ismail, Ehsan Mostajeran Goortani, Mohd Bazli Ab Karim, Wong Ming Tat, Sharipah Setapa, Jing Yuan Luke, Ong Hong Hoe (2015) Evaluation of Docker as Edge Computing Platform, 2015 IEEE Conference on Open Systems (ICOS),(January 2016)

[7] Rohith Raj S1, Dr. Badari Nath K (2020) A Comparative Study of using Virtual Machine and Docker Container for deploying a ReactJS web application, International Research Journal of Engineering and Technology (IRJET), Volume: 07 Issue: 05 (May 2020)

[8] Bellishree P, Dr. Deepamala. N, A Survey on Docker Container and its Use Cases, International Research Journal of Engineering and Technology (IRJET), Volume: 07 Issue: 07 (July 2020).

[9] Lu Chen, Yiru Pan and Richard O. Sinnott (2020) Auto-scaling Walkability Analytics through Kubernetes and Docker SWARM on the Cloud, 10th International Conference on Cloud Computing and Services Science(2020)

[10] Javad Ghofrani and Daniel Lübke (2018) Challenges of Microservices Architecture: A Survey on the State of the Practice, Leibniz University Hannover,(February 2018)