Proceedings of the Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)
IEEE Xplore Part Number:CFP20OSV-ART; ISBN: 978-1-7281-5464-0

Energy Efficient Next-Gen of Virtualization for Cloud-native Applications in Modern Data Centres

Mr.BharaniDharan.G[1]
[1] *Research Scholar, Dept. Of Computer Science, VISTAS, Pallavaram, Chennai.*
Email: bharanmca@gmail.com

Dr.Jayalakshmi.S[2]
[2]*Professor, Dept. Of Computer Applications, VISTAS, Pallavaram, Chennai.*
Email: jai.scs@velsuniv.ac.in

***Abstract*—In the new software-driven world this is the need of the hour to achieve success by accelerating development, delivery of applications and services rapidly that make customers happy and business competitive. To achieve faster delivery of resources and applications to strengthen the business with more transformations containers have been introduced. At the existing scenario, the hyper-converged data centre uses the concept of virtualization which creates software abstraction of the underlying hardware using Hypervisor software that enables to execute several Virtual Machines (VMs) with dissimilar Operating System (OS) flavours. But virtualization is heavyweight and it may take more time to boot. The proposed containerization and docker in hybrid cloud composable data centre is lightweight supports Operating System (OS) level virtualization that isolates resources, libraries and other binaries bundled into a single package for agile modern scalable workloads. This paper focuses on the container-based virtualization to shape the future cloud-based modern data centre for supporting the micro-services based applications for faster deployment to Developer Operations (DevOps) teams in IT with resiliency, High Availability and better resource management with energy savings.**

Keywords-Hypervisor, Container, Docker, Virtualization, Agile, High Availability, Hyper-Converged, DevOps, Micro-services.

I. INTRODUCTION

In the past decade, the computing paradigm has a rapid shift from on-premise traditional infrastructure to on-demand Hybrid cloud Infrastructure. Most of the IT organisations and enterprises moved to cloud for the development of core business. Cloud Service Providers (CSP) has been providing flexible, scalable and elastic infrastructure of IT in term of virtualization technology with VMs [1]. The usage of VMs may incur energy and wasting the computing resources by running the same operations and content with various guest operating systems. To avoid this inefficiency and energy consumption, the container technology has been introduced to increase resource usage efficiency for business agility and improvement in deployment by sharing the same infrastructure with the same operating systems and runtime components. Containerization is evolving and provisioning major impact in Modern computing technologies. By means of containers, developers can encapsulate all components, binaries, libraries and make it as separate micro-services of an application with continuous integration exercise process to make the application more agile with resilient. Applications in the microservice approach consist of many decoupled services independent of each other. Each of these services performs a specific task that is developed and deployed independently [2].

Docker is also playing an emerging role in container-based cloud autonomic data centre. Scalable container service has been used by most cloud service providers like IBM, Amazon, Google, Alibaba. Docker containers can easily be deployed into the cloud-based environment [3].

The unique approach has been utilized by container-based virtualization illustrate about the virtualization layer which is available on the top and the application runs within the OS. It has to be noted here as default host OS is considered at the base [4]. In container-based Virtualization, a single OS will take care of all hardware calls.
In Containers, the layer of virtualization has provided a file system and similarly, the layer of kernel service separation has assisted to isolate resource from all VMs to make containers look like an individual server [5]. This paper is a review of the technology of docker and analyses its performance by a systematic literature review. The work is organised as follow. Next section discusses the literate review of existing VMs and the Next-Gen virtualisation is container-based. In Section III, the benefit of Container-based virtualization whereas Section deals with the components and significants of the docker container. In section V, the architecture of docker is illustrated and similarly in section VI microservice architecture and challenges. In section VII the arrangements of the container by Kubernetes And Docker Swarm. Section VIII illustrates the comparison of performance evaluation in proposed architecture with existing VMs architecture. Section IX concludes that software-defined DC may rely on container and VM placement mechanisms to support next-generation applications.

II. LITERATURE REVIEW

In Modern clouds, Virtualization is one of the vibrant concepts for sharing physical resources among various users [6]. Virtualization based on hypervisor helps the data centre resources can be utilised efficiently by means of consolidation to a single system and even assist for fault tolerance and replication with other geographical areas [7].

Virtualization is widely classified into two categories such as Full virtualization and Paravirtualization. In the process of Full virtualization has created virtual resources such as storage, processor and network to run various flavours of guest OS on a single physical machine which are unaware that they are existing in virtualised location [8]. Most importantly all operating systems think as it runs on bare-metal. Full Virtualisation is implemented through VMware. It is important to note that windows Hyper-V use Microsoft kernel and ESXi server by Linux Kernel. In Para-Virtualisation to satisfy the necessary resource requirements, the applications of guest are accomplished in quarantined fields which have identified as guests and instruct command straight to the OS of the host. The Para-Virtualisation can be used with the Zen family. Virtualization is heavyweight and can cause high resource overheads.

One of the significant factor done through microservice virtualization is to develop the cloud application performance whereas there are several researchers involved in analyzing various virtualization technique performance [9]. The alternate emerging technology to virtualization is containerization that assists in obtaining popularity in VMs because of its high scalability, high performance and light weighted. The Barik et al. has made a comparative analysis over the performance of VMs and container with several simulations [10]. However, the architecture of container differs from VMs and the main distinguish is about sharing of host OS can be done by container but in virtualization it can be done at kernel level. Therefore, the requirement of container is guest process which has been compatible with host kernel is shown in figure I. Thus, container is compared to VMs in which every VM has its individual guest OS whereas the container reduced its overhead by introducing dissimilar guest OS to various container process. This get resulted in requirement of less memory, reduced infrastructure cost with high performance.

The other Next-Gen virtualisation is container-based or OS-level based virtualisation is an alternate to hypervisor-based virtualisation effectively reduces resource overheads and improves resource utilisation with faster deployment in modern data centres for heterogeneous modern workloads is illustrated in Table I. In containerisation, each separate case is said to be Container which executes on the top of a shared OS along with proper and required isolation. In Containerisation the virtual objects are restricted to global kernel resources which result in utilizing resources like networking, CPU and memory for cost-efficient management, scalable and effectiveness efficient over cloud infrastructure [11].

TABLE. I: Ideal differences between Virtual Machines and Containers

S.No.	CHARACTERISITICS	VMs	CONTAINERS
01	Abstraction	Hardware-level abstraction	OS-level abstraction
02	Number of kernels	Multiple kernels for multiple VM's on a physical machine	Single kernel for various containers on a physical machine.
03	Booting time	Takes a few minutes	Takes a few seconds
04	Live Migration Time	Takes more time because of running heavy applications	Shorter time because of migrating specific OS components instead of the whole OS
05	Predictability in Live-Migration	Though post copy offers better prediction but not satisfactorily [12]	Predictability due to smaller memory footprint.

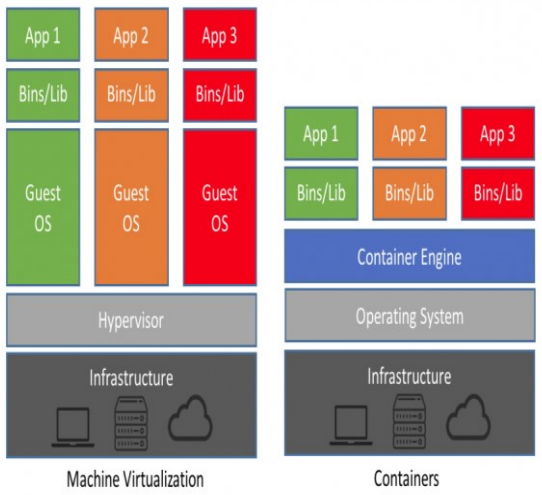

FIGURE.I: Virtual Machine vs. Container [13]

III. Benefits Of OS-Level Based Or Container Virtualisation

Compared to conventional Hypervisor based virtualization, containerization improves performance, energy and efficiency because of additional resource required for each OS is avoided. The Container-based virtualisation fits best when a single OS is required. Single Kernel is used to execute several

cases of an OS. In the OS level, virtualisation doesn't duplicate its functionality.
In Container-based virtualisation instances are very smaller and it is faster and easier for generating or migrating. Cloud Service Providers (CSP) have involved in containers due to the same hardware environment in which more instances of a container can be implemented.

- *Challenges faced in the use of containers:*

One of the major disadvantages present in container-based virtualization is security. However, the containers can able to share the components and kernel of the host OS. It is to be noted here that Virtual Machines shares only the hypervisor make less prone to attack [14].
In containers, the host OS has created a single point situation on failures of host OS may affect any containers.

IV. Docker Containers

The name of the organization is Docker that creates software named as Moby which is an open-source project whereas the Docker gets executed on Windows and Linux. The usage of a container is quite simple to generate, implement and execute applications designed with Docker tool. Docker helps the developers to get a clear view of the stack easily.

A. The different parts in Docker are:

- *Docker Engine:* It performs as a program which generates and executes Docker Container from the Docker image itself.
- *Docker images:* It is a file system and organised with several layers and every layer consists of file for that layer which is immutable. Docker images are used as a Docker container snapshot.
- *Docker Files:* It performs as a text document which consists of guidelines to compose an image that can be recognised using build engine. Docker file also defined the scenario within the container. Inside the container mapping volumes, the files get copied and access to resources may happen.

B. Significant advantages of Container Dockers are:

- *Fast and lightweight Docker:* Containers have performed as a lightweight and boots fast when compared to VMs because VM makes use of the entire OS to start. It is to be noted that each VM runs full OS instance to consume resources [15][16].
- *Utilization of resources:* The physical servers can able to execute more containers than VMs which can result in high resource utilization. However, the containers are thin, effective and portable that can able to execute on the physical host. Containers run as an isolated process and it contains application code with dependencies and also shares a kernel with various containers in the user's host OS space [17].
- *Exact fit for Microservices architecture:* Containers support microservices architecture as every microservices that can be implemented with no disturbance with other microservices. Containers are appropriate to service arrangement by means of agility, separation and easy deployment with new versions.
- *Portability:* Docker can embed any kind of application needs over a container is portable across various platforms. The application of distributed can able to create, implement and execute through containers. Inside the containers deployment, orchestration can be done so that application developers can execute a similar application on VMs or in the cloud.
- *Minimal resource utilization:* Docker can allocate limited resources to any kind of processes by LINUX control groups. It confirms that a single process doesn't consume more computer resources on underfed other processes.

V. Docker Architecture

A. Docker Engine:

It is performed as an application of client-server and it gets comprised into three parts namely

- *Daemon Process:* This is a server process which runs as a background process that executes continuously to process any commands by constantly listening to REST API.
- *REST API:* REST API can able to access via HTTP client and to communicate with Docker daemon.
- *Client:* It is performed as a Command Line Interface (CLI).

B.Architecture of Docker Client-Server:

The architecture of Docker system is majorly involved with client, registry and DOCKER-HOST (Daemon).

- *Docker Client:* Through the Docker client users can interact with Docker Daemon using the commands in CLI through Docker API interface. However, the Docker objects can be created by the command that gets executed by Docker. Hence, the Multiple Docker Daemons have been accessed by the Docker client.

- *Docker Registries:* Docker registry is a place where the images are created using Docker daemon which gets shared in a location. Moreover, the available public registries are Docker hubs that can be accessed from all users. Hence, it is to be noted that the Docker hub can also be utilized for configuring private registry. Thus, the Docker PULL is utilized to retrieve an image from the registry which gets configured and similarly Docker PUSH is used to store an image in the registry.

- *Docker daemon:* It performs as a server process which is determined and runs in the background looks on to REST API for the incoming request and to execute commands. API interface can be listened by daemon using File Descriptor, UNIX, Transmission Control Protocol (TCP) sockets.

- *Docker Objects:* The objects represent services, containers, images and storage.

- *Images:* To build an image, Docker file is used and the file system is a read-only that consists of commands for creating a container to execute an application. Thus, the Docker images can be utilized for implementing an application overproduction or test environment.

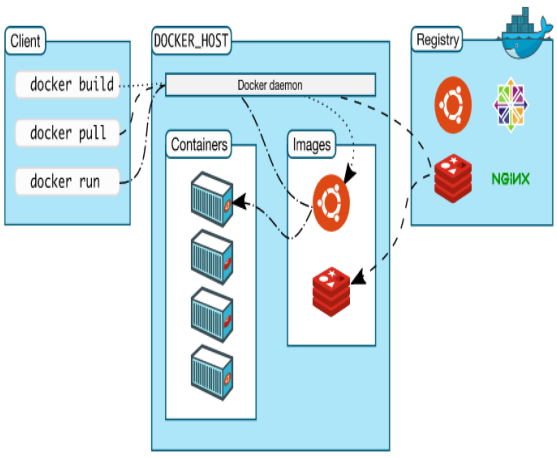

FIGURE.II: Docker Architecture [18]

The Docker client with the build, pull and run has been communicated to the daemon of docker that perform in generating better lift in the building, distributing and executing the docker container. However, the execution is done either on a similar system from docker client and docker daemon or it can able to connect docker client through remote of docker daemon. Therefore, the communication of docker client and daemon has used UNIX network interface by REST and API is shown in figure II.

VI. Evolution Of Microservices

In a monolithic application where the components of the application, interface, data access code has been strongly coupled as a single program. Most of the application is still based on monolithic because it is easy to implement, test and deploy.

Features of Microservices:

- Agile delivery: Services get decomposed into logically modular in which the independent microservices assist in to perform with Agile delivery, easily fits over DevOps model and faster time to market.
- High performance and high availability: Containerized microservices has been leveraged to high performance and availability. The request of microservices with asynchronous nature assists in improving the performance.
- Resiliency and fault tolerance: Container eco-system offers features such as clustering, load balancing, circuit breaking and others to offer high resiliency and fault tolerance for the microservices. Thus, the design is provided for with agile functionality degradation.

Challenges of Monolithic services:

- Difficult to test different modules of an application independently.
- Change in a piece of code in a particular module, deployment of the entire service will take a long time.
- Monolithic services have a single point of failure because bug present in any modules can down the complete service.

To avoid all the pitfalls of Monolithic architecture, microservices came to existence.
In the architecture of Microservices has an application which gets divided into several services which are independent to the database. It gives developers a lot of sophistication to choose the individual modules to work. These services are also being scaled based on their requirement and the services of testing independent are quite simple that brings modularity is illustrated in Table II.

TABLE.II: Comparison of Monolithic Architecture and Microservices

FEATURES	MONOLITHIC ARCHITECTURE	MICROSERVICES ARCHITECTURE
Maintenance	Maintenance is complex of coherency of modules.	Maintenance is easy because they are independent.
Deployment	Containers deployment is difficult because of its expansive nature.	Deploying individual services is easier.
Testing	Testing the entire modules that are coupled is a cumbersome task.	Individual components in Microservices are easier to test.
Scalability	Harder to scale and no flexibility	It can be scaled on demand
Start-up time	More time to boot	Due to the smaller size of individual services, start-up time is less.
Technology	A monolithic application is written in a single language with a single database.	Adopt to any new technologies because of each Microservice use the appropriate database that fits the needs

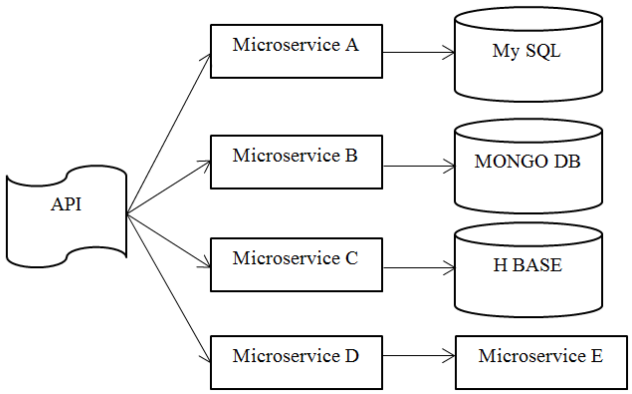

FIGURE. III: Microservices

Each Microservices is for the function of specific business and it gets suitable in operations of particular business function is shown in figure III. However, the architecture of Microservices has provided web based advancement for better flexibility and agile to support and manage code base. Therefore, the Docker is performed as an organizer and eye-opener to container and microservices oriented application deployment.

VII. DOCKER CONTAINER ORCHESTRATION USING KUBERNETES AND DOCKER SWARM

According to the Docker container, both Kubernetes and Docker swarm is the orchestrating tools. Kubernetes is open-source software for containers arrangements. In Greek, Kubernetes is also represented as K8s which illustrate the meaning as a pilot. However, Kubernetes is the ship captain with all executing containers present inside [19]. Therefore, Kubernetes container management has responsibilities in involving container implementation, load balance and scaling. Thus, it performs as an open-source platform for managing containerized applications and to automate deployments [20]. Kubernetes is for organizing and running multiple connected containers in rack server based Performance-Optimized Datacentres (PODS). According to the arrangement, the container Kubernetes is in the de-facto standard whereas Kubernetes has supported various modern workloads namely stately, stateless and processed workloads with boundless swift [21]. Kubernetes helps us to create ingress routing and to run and execute stately services. It will also help to maintain secret and manage passwords. Kubernetes can be useful in deploying a large cluster, managing a large number of containers to reliable and quick response time.

A. Kubernetes Features:

- *Load balancing with service Discovery:* Kubernetes automatically maintains all networking communication by assigning IP addresses in all containers and a DNS name to container set for maintaining load balance within the clusters is illustrated in Table 3.

- *Auto Bin-Packing:* Based on the requirements Kubernetes automatically schedules and pack the applications within the containers. It provides best-effort workloads by ensuring complete utilization of resources and saves unused resources [22].

- *Storage Automation:* With the aid of Kubernetes mounting, the storage system will be done based on user choice. User can choose for public cloud provider storage namely AWS or local storage or by shared network storage such as Network File System (NFS).

- *Automatic Rollbacks:* Kubernetes automatically roll back the changes for the user if something goes wrong with the configuration or not proper updates installed to an application.

- *Self Healing:* During the execution, if the containers fail, Kubernetes automatically restart the containers and kill those containers that do not respond to custom health checks [23].

B. KUBERNETES ARCHITECTURE

Kubernetes architecture mainly comprises of

1.Master Node:

Kubernetes cluster can be managed by the Master Node and it is the waypoint to the entire administrative task. The controlling point is hosted by the Master Node is available for the whole environment is shown in figure IV. Some of the services in Master Node are:

- *Control Manager:* It manages the process and the service that executes together on Kubernetes Master.
- *API Server:* This is to showcase the functionality to the users. Using **kubcctl** command the user can manage Kubernetes and also can talk to the API server.
- *Scheduler:* In some environment whether the Kubernetes pods or implements have been scheduled properly will be ensured by the scheduler.
- *ETCD:* It is the back end database with key-value pair gets stored in the database [23].

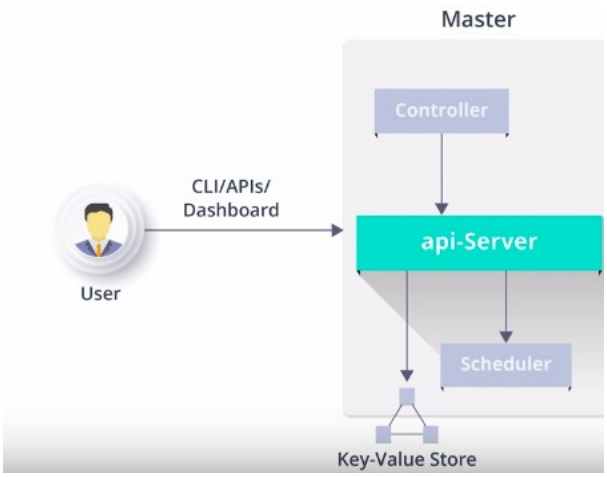

FIGURE.IV: Kubernetes Architecture (Master Node)

2. Worker-Slave Nodes:

Apart from the Master Node, there are also slave nodes [24]. If the architecture is large, more worker nodes will be there is shown in figure V.

- *Kublet:* One of the significant parts of the slave node is Container Engine (CE) and it will be essential and make it possible for executing containers. Kubernetes can talk to Kublet for executing container that executes on worker nodes.
- *Pods:* Kubernetes can take care of any hosts present in a cluster and can join any hosts available over a cluster to run container is known as POD. POD is mainly used for redundancy, scalability [25]. Group of POD's that can form an application and similarly storage is referenced to the POD which can be inside or outside the cluster to the POD. Thus, the Kube-proxy has performed as a network proxy that can execute on every node over the cluster.

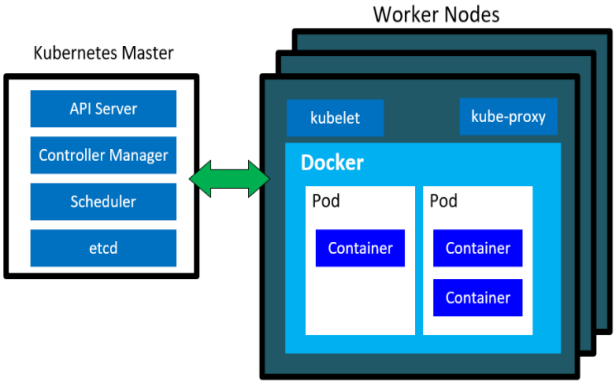

FIGURE.V: Kubernetes Architecture-Slave Node [26]

3. Kubernetes Cluster:

It contains a set of node machines for running containerized applications. A cluster contains a slave node and a master node. Running Kubernetes has shown running a cluster. Deploying programs on to the cluster intelligently handle the distributed work to the separate nodes. Kubernetes can able to supports clusters up to 5000 nodes.

C. DOCKER SWARM

A swarm acts as a machine crew which can be either physical or VMs that assist in executing the application of Docker and it gets configured for merging in a cluster is illustrated over Table 3. The Docker swarm has assisted user for managing several containers generally which has been implemented over various host machines. Swarm directors are the root machines in a swarm that can process the direction or allow different machines to join the swarm as workers or labourers. The IT administrators, developers can be easily managing a cluster of Docker nodes as a single Virtual machine is shown in figure VI.

Docker Swarm Features:

- *Autoload balancing:* There is autoload balancing present in the environment, and it can able to script that into how it can be written out and structure the Swarm environment.
- *Decentralized access:* Swarm makes it very simple to the teams for accessing and managing the environment
- *High scalability:* Load balancing has converted the Swarm environment into a highly scalable infrastructure.

Types of nodes in the swarm cluster:

- *Manager Nodes:* It will handle the cluster management tasks such as scheduling services, maintaining the state of the cluster and serving swarming mode through HTTP API endpoints.

- *Worker nodes:* There are instances of Docker engine particularly to execute containers. I do not involve in raft distributed state like scheduling decisions. It should be noted that no worker node without at least one manager node.

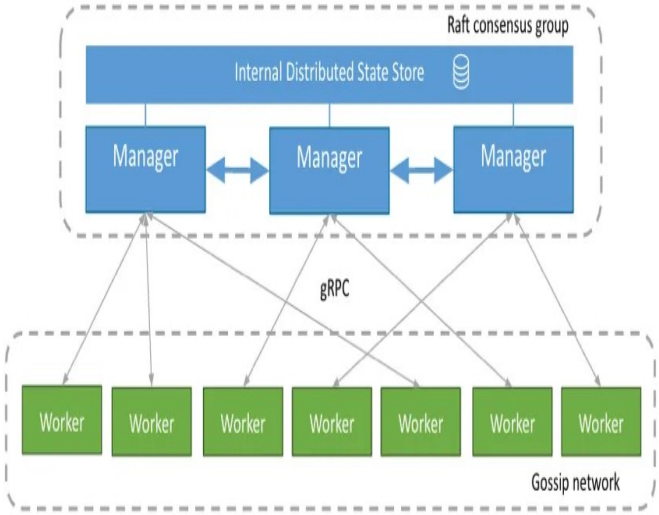

FIGURE. VI: Swarm Cluster [27]

This proposed architecture Orchestration of docker containerization has implemented with the modern DC to discover better energy efficiency and high agility due to automating deployment of Kubernetes and auto load balancing of docker swarm. The proposed architecture of docker conterinization is compared with existing VMs.

VIII. RESULT AND DISCUSSION

In this session, the implementation strategy of elastic resource allocation depends upon the performance measure whereas the proposed Docker containerization DC has determined the suitability to meet the requirement in various workload types using cloudStack4.5. Hence, this proposal has considered with standard DC in a combination of 15/15 as CPU/Memory modules using 3.4 GHz and capacity as 24 GB capacities correspondingly over CentOS6.9 for comparison of Docker containerization architecture of DC among existing VM architectured DC.

Resource Energy consumption

The ratio of total energy consumption and total interval numbers is said to be average energy consumption is formulated in equation 1. In addition, energy consumption is formulated in equation 2.

$$\text{Power}_{avg} = \frac{\sum_{i=1}^{n} \text{Power}}{N} \quad (1)$$

$$\text{Power} = k * P_{max} + (1 - k) * P_{max} * u \quad (2)$$

Where,
K = the coefficient of idle power consumption = 0.7
Pmax = Peak power
u = Utilization of CPU

The power consumed from the CPU during workload process and ideal are evaluated for all type of DC infrastructure whereas the proposed architecture DC consumed less power compared than that of VM infrastructure DC shown in figure VII due to high agility and automatically schedules of the applications within the containers by Kubernetes.

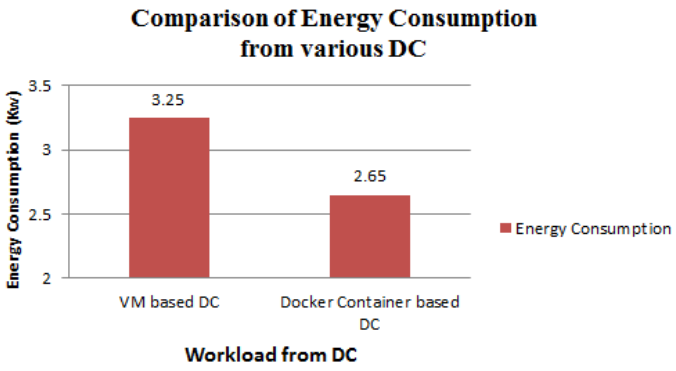

FIGURE. VII: Comparison of energy consumption

IX. Conclusion And Future Work

To cope up with executing modern applications, the Software-Driven data centres are relied on orchestration using containers, Dockers and Kubernetes to provision the resources to the cloud-native, modern workloads or the Dev-ops teams with agility by means of on-demand. This proposed architecture has evaluated in term of power consumption which illustrates less energy consumption than VMs architecture. The Modern infrastructure also provides auto-scaling with proactive and predictive modes, load balancing using server consolidations and VM placement mechanisms to support the next-generation applications which will help for the enterprise business growth and expansion that enhanced to generate the application deployment still more incredible, faster and automated. In future works, the study will be conducted on Ansible container and playbooks can be used.

References

[1] AkkaritSangpetchet. al., "Thoth: Automatic Resource Management with machine Learning for Container-based cloud platform", IEEE, Proceedings of 7th International Conference on Cloud Computing and Services Science, 2017, ISSN: 978-989-758-243-1.

[2] Z. Xiao, I. Wijegunaratne and X. Qiang, "Reflections on SOA andMicroservices," 2016 4th International Conference on EnterpriseSystems (ES), Melbourne, Australia, 2016, pp. 60-67.

[3] Bui, T. (2015). Analysis of docker security.arXiv preprint arXiv:1501.02967.

[4] Sachchidanand Singh et. al., "Conatainers and Docker: Emerging Roles and Future of Cloud Technology, IEEE, 2016, ISSN: 978-1-5090-2399-8/16.

[5] Virtualization performance and Container-based Virtualization, http://searchservervirtualization.techtarget.com/tip/virtualization-performance-and-container-based-virtualization.

[6] David Bernstein, Containers and Cloud: From LXC to docker to Kubernetes, IEEE Cloud Computing, 1(3): 81-84, 2014.

[7] Minxian Xu et. al., "A survey on load balancing algorithms for VM Placement in cloud computing, Concurrency and computation: Practice and Experience. 29(12): e4123, 2017.

[8] Rich Uhlig et al., Intel Virtualization Technology Computer, 38(5):48-56, 2015.

[9] H. Khazaei, C. Barna, N. Beigi-Mohammadi and M. Litoiu, "EfficiencyAnalysis of Provisioning Microservices," 2016 IEEE InternationalConference on Cloud Computing Technology and Science (CloudCom),Luxembourg City, 2016, pp. 261-268.

[10] R. K. Barik, R. K. Lenka, K. R. Rao and D. Ghose, "Performanceanalysis of virtual machines and containers in cloud computing," 2016International Conference on Computing, Communication andAutomation (ICCCA), Noida, 2016, pp. 1204-1210.

[11] JunzoWatadaet. al., "Emerging Trends, Techniques and open issues of Containerization, DOI: 10.1109/ACCESS. 2019-2945930, 2019.

[12] PetterSvard et al., "Principles and Performance Characteristics of Algorithms for LIVE VM Migration. ACM SIGOP S Operating Systems Review, 49(1): 142-155, 2015.

[13] https://blog.netapp.com/blogs/containers-VS-vms/

[14] KinnaryJangla, Accelerating Development Velocity using Docker: Docker across Microservices, IEEE, ISSN: 978-1-4842-3936-0, 2018.

[15] Docker leads the container technology charge in cloud-http://serachcloudcomputing.techtarget.com/feature/Docker-leads-the container-technology-charge-in-cloud.

[16] containers-Not Virtual Machine-Are the future cloud-http://www.linuxjournal.com/content/conatainers%E2%80%94 not-virtual-machines %E2%80%94 are-future-cloud?page=0,1.

[17] Microservices Architecture, Containers and Docker. http://www.ibm.com/developerworks/community/blogs/lba56fe3-432f-alab-58ba3910b073/entry/microservices_architecture_containers_and_docker?lang=eng.

[18] KinnaryJangla, Accelerating Development Velocity using Docker: Docker across Microservices, IEEE, ISSN: 978-1-4842-3936-0, 2018.

[19] Documentation of "PaaS and Container Clouds" by John Refrano, IBM,NY University, 2015.

[20] Publication of "Kuberenetes Fundamentals" by the Linux Foundation training, 2017.

[21] "Bringing Devops for Network with Ansible" by Redhat Enterprise.

[22] Basit Mustafa, Taow. James Lee, Stefan Thorpe "Kubernetes from the ground up, deply and scale performance and reliable containerized applications with Kubernetes" by Level Up Kubernetes Program, 2018.

[23] Article on "What is Docker Container?" by Saurabh Kulshrestha.

[24] Article on "How does Kubernetes Networking work" by Level up Programming.

[25] Arundel, Khare, Saito, lee and Carol Hsu "Devops: Puppet, Docker and Kubernetes-learning path" by Thomas Uphill, Packet publications, First Edition, 2017.

[26] Jay Shah and DushyantDubaria, "Building Modern Clouds: using Docker, Kubernetes and Google Platform, IEEE, ISSN: 978-1-7281-0554-3/19, 2019.

[27] Nikhil Marathe, Ankita Gandhi, Jaimeel M Shah, " Docker swarm and Kubernetes in cloud computing Environment", IEEE Xplore, ISSN: 978-1-5386-9439-8, 2019.

[28] Dmitry Paunin, "The best architecture with Docker and Kubernetes-myth orreality.