May-June 2020
ISSN: 0193-4120 Page No. 3779-3787

Federated Cloud & Containerization – A Dockers Perspective

[1]Divya Kshatriya, [2]Gopal Krishna Shyam

[1]Research Scholar, School of CSA, REVA University, Bengaluru 560064, India
[1]divya.kshatriya@gmail.com
[2]Professor, School of C&IT, REVA University, Bengaluru 560064, India
[2]gopalkrishnashyam@reva.edu.in

Article Info
Volume 83
Page Number: 3779-3787
Publication Issue:
May-June 2020

Abstract
As the ecosystem for Cloud Computing was conceptualized it was widely created as an outsourcing of all business IT resource requirement from a cloud service provider. Over the years there has been a progression in cloud computing offerings from not just operating as a single cloud service provider to a multi cloud service providers mutually offering federated cloud as a service. Federated cloud overcome the limitations of single cloud service provider by pooling up the data center resources of multiple clouds vendors and giving high performance on applications, high volume of storage, no downtime and efficient throughput. This requires seamless portability of applications with least overhead requirements and dynamic migrations of applications across the various cloud heterogeneous architecture without interoperability issues. With the advent of containerization technology that packages together application code with all its required dependent libraries and binaries eliminates the technical issues related to portability of applications and incompatibility issues of multiple cloud landscape. Docker containers have revolutionized and remodeled the process of building, installing and executing any application irrespective of the technology used in developing or the size of the application. These applications bundled in Docker containers make the orchestration of application components across federated cloud a seamless process with high level of flexibility. This paper is conceptualized with a deeper understanding on the impact of containerization on federated cloud.

Article History
Article Received:*19 November 2019*
Revised: *27 January 2020*
Accepted: *24 February 2020*
Publication: *12 May 2020*

Keywords: *Federated Cloud, Containerization, Virtualization Docker.*

1. Introduction

Cloud computing is the rendering of computing related services – that include servers, databases, storage, software, networking, intelligence &analytics – delivered over the Internet to enable flexible resources, quicker innovation, and scale economies. Typically, the customeronly pays for cloud services consumed by him/her, driving decreased operational costs, allowing effective management of information technology infrastructure and scaling up in line with growth or shrinkage of business demand [1]. Cloud federation is the topology of interconnection ofcloud computing environments across two or more providers aimed at traffic load balancing, handling demand spikes and lowering operational costs.Cloud federation mandates one provider to rent or wholesale computing resources to another provider. Such resources become a permanent or temporary augmentation of cloud computing environment maintained by buyer, based on the specific federation agreement signed-off among providers [2][3].

May-June 2020
ISSN: 0193-4120 Page No. 3779-3787

Cloud federation presents two notable advantages to cloud providers. Firstly, it empowers the providers to obtain revenuesthrough computing resources that otherwise would be underutilized or idle. Secondly, cloud federation allows the cloud providers to enlarge their geographic outreach and handle unexpected peaks in demand by effectively utilizing their existing data centres. While the advantage of administering cloud federation is enormous, nevertheless it's a challenging task to smoothly relocate applications across various cloud service providers [4].It necessitates the complexity of understanding a spectrum of cloud architectures and building applications to dynamically interoperate on these platforms. The innovation in container technology & micro-services applications has immensely facilitated this aspect.

A container is a modular unit of software that bundles together code and its underlying dependencies to enable the application to execute with performance and reliabilitybetween computing environments. Containers are self-contained and don't need a guest operating system while sharing its host kernel. Container enables the user with quicker application development & deployment no matter that the platform [5]. Docker containers are instrumental in popularizing the current day virtual containers. A Docker container image is a standalone,lightweight &executable software package that consists of everything required for executing an application: runtime, code, system tools&libraries and settings. Such technology has become a boon for both development and production environments. Docker containers are intrinsically portable and can be executed on a VM or in the cloud unmodified, the containers are movable between VMsand to bare metal without the need for intensive efforts to enable transition [6][13].

The key differentiator in container technology is "isolation." Isolation implies velocity – containers are much compact entities vis-à-vis virtual machines which promotes their faster deployment. Isolation implies performance – diminished boot-up times. Isolation means adaptability – containers are movable across various platforms &cloud vendors. Therefore, they've earned extensive relevance in federated cloud scenarios [5].

2. Containerization

A Brief History of Containers

Containerization has ushered a dominant trend in software development as substitution or complement to virtualization. It entails encapsulation or bundling together software code and its entire dependencies to allow it to execute consistently &uniformly across infrastructures. This technology is rapidly evolving, leading to quantitative for developers, operations teams, and overall software infrastructure.

Containers are not a newborn technology and have existing for over a decade. Still,prior to Docker's meteoric success starting in 2013 they were not well-known or wide-spread. For better appreciation of container technology, one would need to rewind go back in time from the origination of concept of containers from Chroot right back in 1979.

Chroot was introduced first 40 years ago in 1979 during the development of Unix V7. It was built to transform the supposed root filesystem of a process and its children. In simple terms, network namespaces or modern process isolation were removed. In 2000, FreeBSD jails expanded upon Chroot and popularized enhanced sandboxing features. Jails consist of their own network interfaces and IP addresses, that disallowed by default raw sockets. This ushered resemblance to virtual machines [7][8].

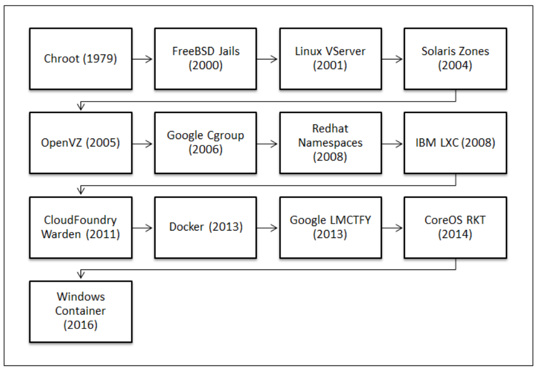

Figure 1: History of Containers

Shortly, the Linux community jumped on the bandwagon with Linux-VServer in 2001 and OpenVZ in 2005. Both were out-of-tree patches to the Linux kernel, and hence somewhat complicated to maintain. They offered reasonable process & network isolation, however were also laden with some downsides. It wasn't helpful that hosting providers offered these containers as light-weight virtual machines, creating frustration among people for not getting features provided by VMs.

Control groups (cgroups) are a Linux kernel feature launched in 2008 to insulate the resource usage (memory, CPU, network, disk, etc.) of process groups. Over the years, it underwent several changes however retained its central purpose, that is to furnish a unified interface for process isolation in the Linux kernel. Cgroups were redesigned in 2013, along with a new feature called Linux namespaces. Namespaces partition kernel resources to prevent a process in one namespace from viewing resources of another namespace. Work is still being done to make almost every part of the Linux kernel namespace-aware. The one with utmost importance are process ID, mount, interprocess communication, network, and user namespace. Cgroups and namespaces modified everything, since they are the elementary units of all contemporary container technologies on Linux [11].

Also in 2008, LXC took birth developed on cgroups& namespaces. It was the first available container tool that interoperated with the upstream Linux kernel. Nonetheless, the early versions were weaker in security vis-à-vis its prehistoric ancestors, Linux-VServer&OpenVZ. Root in an LXC container implied root on the host. This wasn't applicable any longer with LXC 1.0 which imported unprivileged containers with the aid of user namespaces.

With Warden in 2011, CloudFoundry enrolled into the arena using LXC as its base. It had independent client & server components, aimed at managing containers across a machine cluster. Subsequently, they switched LXC with their home-grown platform independent implementation. Warden containers commonly had only two layers: a read-only OS root file-system & a runtime file-system from elementary units called buildpacks. CloudFoundry is still existent however they've deserted Warden in exchange of contemporary standards.

Google, whose engineers' brainchild was cgroups, has already become a leading player in container technologies also launched their own open-source tool in 2013 called Let Me Contain That For You (LMCTFY). It could never take off the ground since development stopped with their interest getting drawn to new standard components in 2015 and introduced nsjail.

It was Linux Containers – LXC – which enabled instituting containers as a key virtualization technology reasonable for cloud data centers. LXC is a Linux operating system-level virtualization method for running multiple isolated Linux systems on a single host. The Namespaces &Cgroups features made Linux Containers possible. Docker came along later. Originally, it was a project to create single-application LXC containers. Ever since, Docker has introduced many noteworthy upgrades to the container concept, that includedistancing away from LXC as the container format. Docker containers allow the users to conveniently deploy, move, replicate, and back-up a workload, thus providing cloud-like flexibility to any infrastructure capable of running Docker.A Docker container image is a standalone,lightweight, executable software bundle which comprised of aggregate needsfor executing an application: runtime, code, system tools, system libraries &settings [7].

With open source Docker Engine emerging in 2013, an industry standard for containers with straightforward developer tools and a universal framework of packaging, expedited the uptake of this technology. Gartner, a renowned research firm,estimates that by 2020 over 50% of firms globally will adopt container technologies [10][13].

Back in 2014,when standards within container industry yet seemed out of reach, another key endeavor came into fray. Kubernetes was introduced Google engineers, massively motivated by know-how of company's internal container orchestration systems. It rapidly fascinated contributors from dominant industry players e.g. RedHat, Intel, CoreOS. Kubernetes is a sophisticated framework for automation of management, scaling &deploymentof containers. It was embraced by the CNCF, alongside majority of its key components. Kubernetes initially harnessed Docker for its container runtime. However, it's now compatible with any runtime through the Container Runtime Interface (CRI), e.g. CRI-O that utilizes the interface through containerd&runC [11].

Docker's share of the orchestration segment was Swarm,which is a self-contained tool to manage a cluster of docker daemons through the same API. It was antiquated by Swarm mode that is embedded in Docker since version 1.12.

3. Docker Containers

A.

B. Docker Containers

Container technology took off in 2013primarily as an open-source project, initially named *dotcloud*, with vision to create single-application Linux containers. Ever since, Docker containers has emerged as not only a favoured development tool but also proliferated as a runtime environment. Docker is build using Go and leverages constructs of the Linux kernel to distribute its functionality. A key driver of Docker being so famous is that it offers the commitment of "develop once, run anywhere." Docker renders anuncomplicated approach to packaging an application & associated dependencies (esp. runtime) within a unified container, and enables a runtime abstraction that facilitates the container to execute across various versions of Linux kernel. Using Docker, a developer can build a containerized application on their workstation and later conveniently deploy the same to any Docker-enabled server while not having to retune or retest it for the specific server environment – both on on-premise or on cloud scenarios. Additionally, Docker offers a mechanism of sharing & distribution of software which enables developers &operations teams to seamlessly share &reuse container content. This distributed mechanism, topped-up with transportability across machines, is the secret-sauce for Docker'simmense acclaim with developers &operations teams [11][13].

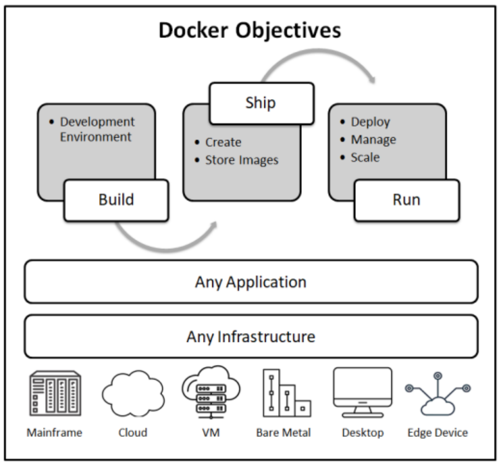

Figure 2: Docker Objectives

Docker Components

Docker is not only a development tool but also a runtime environment. To comprehend Docker, one must first grasp the concept of a Docker container image. A container invariably kicks-off with an image and is deemed an instance of the same.

An image represents a static blueprint of container's expected identity in runtime, inclusive of application code within the container & runtime configuration settings. Docker images comprise of read-only tiers (or layers), which implies the image is immutable once created.

A functioning Docker container is an instance of an image. Containers extracted out of the same image are each other's replica from perspective of their application code &runtime dependencies. However, dissimilar to images which are read-only, each deployed container is embedded with a writable layer (a.k.a. the container layer) atop the read-only content. Runtime changes, inclusive of any updates &writes to files &data are conserved in the container layer only. Hence, numerous parallel executing containers which share the same underlying image could comprise of varied container layers.

The deletion of a running container is accompanied with the deletion of writable container layer – it will then not persist. The exclusive approach to conserve changes is by doing ancategorical "docker - commit" before the container is destroyed. While a "docker – commit is done," the executing content of container, along with the writable layer, is persisted into a new container image and thereafter stored to the disk. This now becomes a new image disparate from the preceding image which has instantiated the container.

Employing this definitive "commit" command, one could construct a consecutive, distinct set of Docker images, each one developed atop the preceding image. Additionally, Docker harnesses a Copy-on-Write strategy in order to diminish the disk footprint of containers &images which share the ditto base components. This enables optimization of storage space and minimization of container start time. Besides the image concept, Docker also comprises of handful of specific components which are divergent from those in Linux containers.

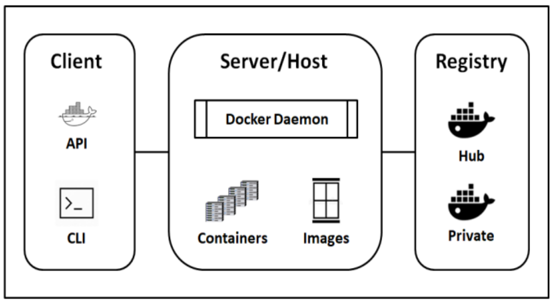

Figure 3: Components of Docker Architecture

*Docker daemon:*a.k.athe Docker Engine. Docker daemon is a narrow stratum between the containers and the Linux OS. It is the perpetual runtime environment which manages application containers. Any Docker container can execute on any server which has Docker-daemon enablement, irrespective of the substrata operating system.The daemon instantiates & maintains Docker objects e.g. containers, images, volumes &networks.A daemon can also communicate with its peers to manage Docker services.

Docker Registries:A Docker registry saves Docker images. Docker Hub is a communal registry for usages by anyone, and Docker is configured by default to scan for images on Docker Hub.

Dockerfile: Developers harness Dockerfile to construct container images, that in turn act as the basis of executing containers. A Docker file is a text document which holds all the configuration related information &commands required for assembling a container image. Using a Dockerfile, Docker daemon can seamlessly construct a container image. This process significantly streamlines the procedure for container creation. More precisely, in a Docker registry, one first specifies a "base image" based on which the build process is initiated. One later specifies a sequence of commands, which leads to building of a new container image.

Docker Command Line Interface (CLI) tools: Docker offers a list of CLI commands for lifecycle management of image-based containers. Docker commands span across development functions e.g. build, tagging&export, alongside runtime functions e.g. running, starting, deleting, &stopping a container, and much more.

Docker Objects:While using a Docker, one is creating &using images, networks, containers, plugins, volumes &other objects.An image is a read-only blueprint with prepareatory information for constructing a Docker container. Generally, an image is built on baseline image, by adding few customizations. For example, one may construct an image that is founded on the Ubuntu image, however installs the Apache web server &application,

alongside the configuration information required to enable one's application to execute.

A container is essentially an executable instance of an image. One could create, stop, start, delete or move a container by using the Docker API or CLI. One could help container establish connection to one or more networks, hook up storage to it, or even construct a new image founded on its current state.A container is identified by its image along with any configuration settings one provides to it during its creation or starting. While a container is detached, any modifications to its state which are unsaved in its persistent storage will be abandoned [12] [13].

Features of Docker Containers

One of the most differentiating attributes of Docker containers is their immutability which results inthe statelessness of containers.

As mentioned earlier, a Docker image, once constructed, does not modify. An executing container extracted from the image is embedded with a writable layer designed to ad-hoc contain the runtime modifications. In scenario of container committing prior to deletion using "docker – commit", the modifications will be saved in the writeable layer into a new image which is disparate from the preceding one.

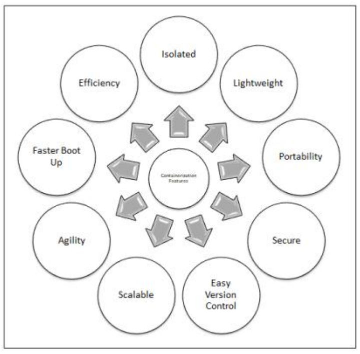

Figure 4: Features of Containers

Immutable images &containers result in an immutable infrastructure, which in turn has several compelling properties which are non-attainable with conventional systems. For example:

Version control: Using the categorical commit method, the Docker enforces forces one to perform version control. One can maintain traceability of consecutive image versions; if required rolled back to a preceding version (thereby to a prior system component) is completely made possible, as prior versions are preserved and never changed.

Neater updates and higher manageability of state modifications: Given the immutable infrastructure, one need not have to upgrade one's server infrastructure, that implies no requirement to modify configuration files, no software upgrades, no OS (operating system) updates, and so on.When modifications are required, one may simply create new containers and deploy them to substitute the old ones. This is a much superior, discrete & manageable approach for state change.

Curtailed drift: To prevent any drift, one can regularly and with pre-planning rejuvenate all the system components to assure they are equipped with the latest version. This procedure is a much effortless with containers which abstract out smaller system components as compared with conventional & heavy software [9][14][15].

Benefits of Docker Container

Due to the contemporary & soaring appeal for data-demanding applications to up-scale various platform needs periodic digital transformations. Containerization of applications offers several advantages, especially the below:

Portability among various platforms & clouds – it's genuinely write once, execute anywhere. Efficiency via harnessing way lower resources than VMs and ensuring much increased usage of compute resources [12]. Agility which enables developers to harmonize with their current DevOps environment. Increased velocity in the administration of upgrades. Containerization of monolithic applications by usage of micro-services aids in development teams creating services with its unique lifecycle and scaling policies. Enhanced security through isolation of applications from the host system and from one another.Quicker application start-up and seamless scaling.Affability to onboard applications on virtualized infrastructures or on bare metal servers. Convenient administration since install, rollback & upgrade processes are embedded into the Kubernetes platform. [9]

Docker's proprietary image format, its comprehensive APIs for container administration, and the ingenious mechanism of software distribution through registries have led to its platform popularity for both development & operations teams. Docker offers such noteworthy advantages to an organization.

Minimalistic, allegorical systems: Docker containers perform optimally if designed as small, modular, specific-objective applications. This leads to containers which are bare-essential in size, that again facilitates speedy delivery, continuous integration & deployment.

Predicable processes & transactions: The topmost pain-point associated with system operations has ever been the apparently arbitrary performance of the infrastructure & applications. Docker pushes one to entail petite & more manageable upgrades and offers a mechanism to curtail system drift; both pre-requisites are really what's required to construct predicable systems. Where drifts are minimized or avoided, one can achieve affirmation that the same application or system should perform identically, irrespective of number of times one deploys them.

Large-scale software reuse: Docker containers reuse tiers (or layers) belonging to other images; that encourages reuse of software. The methodology Docker shares images through registries is yet another enormous medium to proliferate the component sharing & reuse [17].

Authentic multi-cloud portability: Docker implements a genuine platform independence, that enables containers to transmigrate unrestricted among various cloud platforms, on-premises infrastructures, and even development workstations.

Docker has significantly changed the organizational methodology of building systems and services delivery. It has also started reshaping our thinking approach towards software design and the economics of software delivery [16].

4. Vm Versus Containerization

Virtualization is the process of constructing a software-centric, or virtual, depiction of something, e.g. servers, virtual applications, networks &storage.Virtual Machines (VMs) virtualize hardware. Virtualization depends on software to mimic hardware functionality and replicate a virtual computer system. This allows IT organizations to execute more than one virtual systems – and disparate OS (operating systems)&applications – on a unified server. A virtual computer system is known as a “virtual machine” (VM): a firmly confined software container embedded with an OS & application under-the-hood. Each self-sufficient VM is entirely autonomous. Deploying manifold VMs on a unified computer allows various OS & applications to execute on strictly one physical server, or “host.” A narrow tier (layer) of software named as “hypervisor” unbundles the virtual machines & host and real-time earmarks computing resources to individual virtual machine as required. Each guest VM comprises of a dedicated copy of an OSatop the host OS however the host’s hardware is shared among VMs on the same host [18][19.]

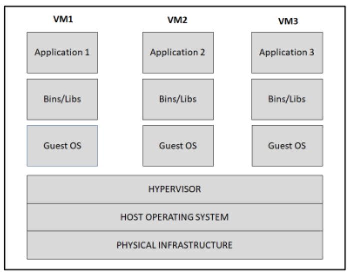

Figure 5: Virtualization Framework

Containers, nonetheless, virtualize the OS – every container enjoys its own dedicated CPU, block I/O, memory, network stack etc., however utilizes the host’s OS similar to other containers on the same host. Containers occupy lower boot volume and lower disk space. One can execute higher number of containers on the same host similar to with VM – up to 100x. It is also faster to initiate and remove a container compared to a VM [18][20].

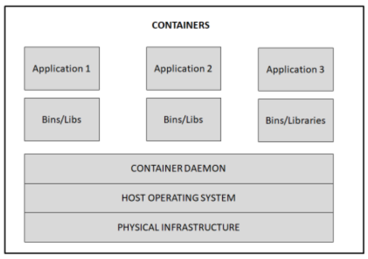

Figure 6: Containerization Framework

VM is in line with its name – a hardware machine that’s virtualized. On the other hand, a container is simply a process, that harnessing kernel features one can segregate and confine the resources accessible to it. Both are carrying out a different mission. They are analogous since they both offer secluded environments – they both can be harnessed to bundle up & dispense software [23].Yet, containers are usually much faster &smaller, that yields them a superior fit for rapid development cycles &micro-services. The compensation is that containers don’t implement authentic virtualization; for example, one can’t execute a windows container on a Linux host.VM’s are designed for applications which are typically more stable and don’t modify much frequently. However, containers are higher in flexibility and ensure a convenient & periodic updatesto one’s containers. In summary, the choice among VM’s and containers boils down to specific use-case. Table-1 depicts the key differences amongst virtualization and containerization[18][20][21][22][24].

Table 1: Comparison between VM and Containerization

PARAMETERS	VM	CONTAINERIZATION
Abstraction	OS from hardware	Application from OS
Isolation	Complete Isolation	Isolation using techniques like namespaces
Boot Up Time	In Minutes	In Seconds
Resource Requirement	Heavyweight	Lightweight
Space Allocation	Data volume cannot be shared	Data volume is shared and reused
Performance	Limited Performance due to multiple running VMs	Near Native Performance as they are hosted in single Docker Engine
Security	Very high	Low as compared to VM
Portability	Compatibility issue while porting to	Easily portable across multiple platforms

	different platform	
Version Management	Difficult to implement version control	Easy version control
Scalability	Slow provisioning and difficult to scale up	Real Time provisioning

5. Containerization In Federated Cloud

Earlier research also brings about the fact that with the emerging and existing technologies like Containerization, Kubernetes, Microservices, Docker enterprise edition will become the foundation for building the right infrastructure for propagation and management of federated clouds. This will help leverage the benefit of maximizing the resource utilization of each cloud provider and workload would be more efficiently managed amongst the participating cloud service providers of federated cloud. The scalability required by applications during peak requirements would be managed in no time as deployment would be seamless.

Current cloud landscape has many different service offerings for customer applications as well multiple deployment offerings. Service offerings range from software as a service, platform as a service, infrastructure as a service, network as a service and storage as a service. The deployment options also range as private cloud, public cloud, community cloud, hybrid cloud and federated cloud. Each cloud offering caters to unique business application requirements to multiple customers in various geographically distributed areas. At the same time in realism an individual cloud service provider cannot conform to all the possible use cases. Even a substantial large cloud service provider with all the available resources of range of servers, network infrastructure, database storage, virtual machines and tools will be able to optimally cater to business application requirements with high performance only when resources are located close to customers.It’s not technically, financially and operationally feasibleto have cloud ecosystem at multiple locations hence to meet up the ever growing requirement of cloud the federation model is well suited. Federated cloudecosystem requires to mutually pool up resources by different cloud vendors that are geographically distributed and are in alliance to support the resource requirement across multiple clouds [25][27].

Cloud environment and integration layers have emerged and matured over the decade to resolve the issues of federated cloud environments. By consolidating the different cloud platforms of multitude of cloud service providers by controlled and commonly managed interface with well governedpayment systems, the current cloud integrators can fundamentally ease the process associated with acquisition along with financially compensating for infrastructure anywhere and anytime.

However execution of development of software applications and setting up production environments in federated cloud landscape is a grave challenge even when establishing of the data center infrastructure is considerable a standard process. Largely all cloud platforms can be considered as relatively simple and easily compatible set of technological resources to support the execution of applications of any kinds. They are designed to have the capacity and capability to handle different kinds of workload requirements [28][29]

With the emergence of container technology like Docker that primarily focuses on the requirement of development teams and production teams to isolate any dependency of application to its infrastructure. The Docker Enterprise (DE) platform is an innovative platform with federated application management feature. This is an independent platform that supports development teams to create ship and run applications consistently and seamlessly across multiple cloud providers.Clouds have fluctuating deployment, migration, resource security, management, and replication behaviors. Federated management is designed to position among all of them and offers a consolidated perspective and automated model for migrating, deploying, and mirroring applications.DE facilitates deployment of significantly available workloads by leveraging either the Docker Swarm or Docker Kubernetes Service. DE provides automation of several tasks which orchestration needs e.g. provisioning of pods, cluster &containers resources [26].

Docker is employed by various large-sized organizations to enable large-scale continuous integration & delivery distribution of application, and for creation of distributed application architectures.To summarize, Docker delivers a portable PaaS (Platform-as-a-Service) environment, barring that in lieu of banking on a cloud vendor’s platform, each specific application encapsulates its own platform (e.g. binaries, libraries etc.) within it, equippedfor faster deployment on any appropriate infrastructure [25] [30]. The requirement for spinning up additional servers to deploy an application for a different geographic area is handled by integration layer of federated cloud marketplace to identify, administer, and deploy the infrastructure tier, and push the current Docker containers on top of that infrastructure, thereby significantly decreasing the lead-time to deployment. In contrast to earlier times, the process of contracting & negotiation with new cloud or hosting service providers, configuration of discrete server environments, applications installation i.e. Docker-Cloud integration model is less time consuming, considerably simpler, and much efficiently manageable.

6. Conclusion

Containerization has brought in a paradigm shift in the field of cloud computing that has enabled the rapid adoption of cloud within organizations across various types (segmented by size, industry, domains etc.). It has offered simultaneous benefits such as agility, portability, scalability, version management – a unique combination that has led to its popularity. Several organizations are exploring containers as medium to enhance their application life-cycle management by harnessing proficiencies such as continuous integration and continuous delivery. Containerization is a remediation to proprietary vendor lock-in cloud technologies by conforming to the principles of open-source, which also aids the easier adoption of federated cloud architectures. Containers and Docker in essence are not in battle with virtual machines; they are actually complementary to each other and meant for distinctive purposes.With the increasing preferences of organizations to leverage multiple cloud service providers with the objective to accrue best-of-breed features and to de-risk themselves, continuous developments in containerization (especially Docker) technologies are poised to offer a practical solution via federated cloud framework.

This paper provides perspective around Dockers containerization its evolution, features, advantages, comparison with virtualization, and its unique ability to facilitate and support the federated cloud architecture. It can provide a template for future papers on other complimentary cloud technologies.

References

[1] P. Mell and T. Grance, “The nist definition of cloud computing,” National Institute of Standards and Technology, vol. 53, no. 6, p. 50, 2009. [Online]. Available: https://www.nist.gov/publications/nist-definition-cloud-computing

[2] Bermbach, D., Kurze, T., Tai, S.: Cloud federation: eff ects of federated compute resources on quality of service and cost. In: Proceedings of IC2E 2013, pp. 31–37 (2013)

[3] A. Marosi, G. Kecskemeti, A. Kertesz, P. Kacsuk, “ FCM: An architecture for integrating IaaS cloud systems”, The Second International Conference on Cloud Computing, GRIDs, and Virtualization, pp. 7–12, 2011.

[4] R. Ranjan, “The Cloud Interoperability Challenge,” IEEE Cloud Computing, vol. 1, no. 2, pp. 20–24, 2014.

[5] BlessonVarghesea and RajkumarBuyya “Next generation cloud computing: New trends and research directions” Future Generation Computer Systems, Vol. 79, no. 3, pp. 849-861, February 2018.

[6] https://cloud.google.com/containers

[7] David Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, 2014.

[8] https://dzone.com/articles/evolution-of-linux-containers-future

[9] https://www.ibm.com/cloud/learn/containers

[10] What is a container? https://www.docker.com/resources/what-container

[11] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” Linux Journal, vol. 239, pp. 76-91, 2014.

[12] J. Turnbull, The Docker Book, 2014; www .dockerbook.com.

[13] https://docs.docker.com/engine/docker-overview/

[14] Claus Pahl, Antonio Brogi, Jacopo Soldani and Pooyan Jamshidi,” Cloud Container Technologies: A State-of-the-Art Review,” IEEE Transactions on Cloud Computing, vol. 7, no. 3, pp. 677-692 , 2019

[15] Preeth E N, Fr. Jaison Paul Mulerickal, Biju Paul and Yedhu Sastri, “Evaluation of Docker Containers Based on Hardware Utilization” 2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, India, 19-21Nov 2015

[16] Hao Zeng, Baosheng Wang, Wenping Deng and Weiqi Zhang, “Measurement and Evaluation for Docker Container Networking,” International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, China, Oct 2017

[17] Han, R., Guo, L., Ghanem, M., Guo, Y.: Lightweight resource scaling for cloud applications. In: International Symposium on Cluster, Cloud and Grid Computing (IEEE/ACM CCGrid), pp. 644–651 (2012)

[18] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, “Virtualization vs Containerization to support PaaS”, Proceeding of 2014 IEEE International Conference, pp. 610- 614 (2014)

[19] https://www.vmware.com/in/solutions/virtualization.html

[20] https://www.docker.com/blog/vm-or-containers/

[21] Ann Mary, “Performance Comparison Between Linux Containers and Virtual Machines ,” 2 015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 19-20 March 2015.

[22] Mathijs Jeroen Scheepers, “Virtualization and Containerization of Application Infrastructure: A Comparison,” 21st Twente

Student Conference on IT, June 23rd, 2014, Enschede, The Netherlands.

[23] Carlos de Alfonso, Amanda Calatrava, Germ´an Molt´ o,"Container-based Virtual Elastic Clusters," The Journal of Systems & Software, Jan 2017

[24] https://devopscon.io/blog/docker/docker-vs-virtual-machine-where-are-the-differences/

[25] Claus Pahl, "Containerization and the PaaS Cloud, " IEEE Cloud Computing , Vol. 2, no. 3, pp. 24-31, July 2015

[26] https://docs.docker.com/ee/

[27] Di Liu and Libin Zhao, " The Research and Implementation of Cloud Computing Platform Based on Docker," 2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing, Chengdu, China, 19-21 Dec 2014

[28] Moustafa Abdelbaky, Javier Diaz-Montes , Manish Parashar , Merve Unuvar and Malgorzara Steinder, "Docker Containers across Multiple Clouds and Data Centers," 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, 7-10 Dec. 2015

[29] Nitin Naik, "Building A Virtual System of Systems Using Docker Swarm in Multiple Clouds," 2016 IEEE International Symposium on Systems Engineering (ISSE), Edinburgh, UK, 3-5 Oct. 2016

[30] James Hadley, Yehia Elkhatib, Gordon Blair, and Utz Roedig , "MultiBox: Lightweight Containers for Vendor-Independent Multi-cloudDeployments" , Proceeding of Embracing Global Computing in Emerging Economies,pp. 79-90, Nov 2015