Platform Virtualization: Understanding Virtual Machines, LXC, Docker, Kubernetes and Ubernetes

Vijay Gupta
Assistant Professor, Department of IT,,
Jagannath International Management School, GGSIPU, Delhi, India

Sumit Chauhan
Associate Professor, Department of IT,
Management Education and Research Institute, GGSIPU, Delhi, India

Deepak Sharma
Assistant Professor, Department of IT,,
Jagannath International Management School, GGSIPU, Delhi, India

Abstract- Cloud computing is gaining rapid popularity among big players like Google, Oracle, Microsoft and others. Platform Virtualization provided the application isolation, optimized use of Hardware and deployment and operational flexibility. The last few years has seen the rise of Virtual Machines as main technology for Cloud computing, but VM's have some limitations like underutilization of hardware and high overhead. This paper describes the different Platform Virtualization technologies available in the market and the upcoming technology "Docker", with the advent of Container based virtualization like Docker, new problems of Container management and scaling across clouds is rising. This can be handled by two very new technologies Kubernetes and Ubernetes.

Keywords – Cloud Computing, Virtualization, Containers, Docker, LXC, Kubernetes, Ubernetes, Namespace, Docker Images.

I. INTRODUCTION

Platform virtualization means the ability to run different operating system executing completely independent applications at the same time over the same hardware. The isolation is provided between them by using hypervisor or operating system level virtualization. Hypervisor is software which creates virtual machines on computer (mainly on decent server); each virtual machine can host different operating system executing independent applications. Each virtual machine is known as guest machine and operating system is known as guest OS. The physical machine is known as host machine. Generally in cloud environment virtualization is done to run completely independent applications and no sharing is required between them. Hypervisor provides complete isolation between Virtual Machines. However when each virtual machine is running the same kernel and similar operating system distributions, the degree of isolation offered by hypervisors comes at the cost of efficiency relative to running all applications on a single kernel [1].

In Operating System based virtualization, the host OS creates multiple isolated user space instances. Each instance is known as container. In this approach we cannot run multiple guest OS, for example in Linux virtualization only Linux guest can be run on Linux host. Each container shares the kernel of the host operating system. The advantage is that the performance and efficiency of guest application is improved very much due to shared kernel. The short comings are we cannot install Linux container on windows and vice versa and the isolation between different containers is difficulty to achieve.

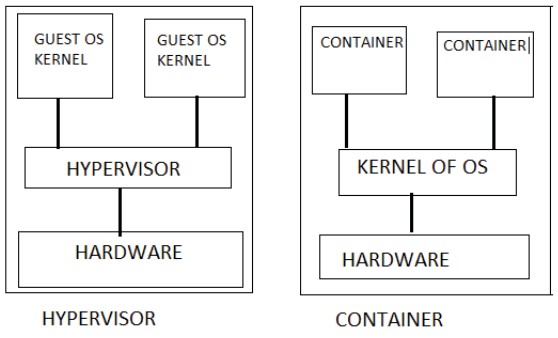

Fig. 1 Difference between b/w Hypervisor & OS based virtualization.

Containers can be classified in two types OS containers and Application containers. In OS level virtualization as depicted in the figure 1, that guest shares the host OS and kernel. You can install and run different applications inside the OS containers which share the kernel of the host and have independent user space. Each OS container can run multiple services and processes. Examples are Solaris zones, OpenVZ, LXC and Linux Vserver. Application containers can run single service or process by using host kernel. Applications containers are more light weight than OS containers. Example of application containers are Docker and Rocket.

II. BACKGROUND

A. *LXC*

Linux Containers (LXC) provides lightweight operating system virtualization as compare to Hypervisors. LXC is the successor of VServer and OpenVZ. So each container has their own file systems and network stack and every container can run its own Linux distribution [2]. The history of containers goes back to the UNIX chroot command (allows processes to see any directory as its root directory independent of the other processes), which was introduced as the part of UNIX version 7 in 1979. In 1998 an extended version of chroot was implemented in FreeBSD and called jail [3]. In Solaris 10, it was further improved and introduced as zones. Finally in Solaris 11 it was renamed as containers. Other UNIX vendors also introduced similar capabilities -IBM AIX workload partitions, HP-UX containers etc.

Linux also offered the same capabilities in the form of LXC. Isolation is an important aspect of containers, it is provided through Linux cgroups and namespaces. Namespaces are used to isolate resources like file systems, networking, user management and process ids. Cgroups are used for resource allocation and management [2]. Cgroups is short hand for control groups, cgroups can be used to manage and monitor the usage of following resources like CPU, Disk, and memory and network bandwidth. We can create different groups for example we can create a group#02 and assign the resource limits to this group like 35% of CPU, 100GB of Disk and 25% of memory. Further we can assign the processes to group#02, which implies that the assigned processes cannot use more than the resource limits.

Namespaces are important to provide security in case of multiple services running inside a server. If an intruder manages to hit one service then isolation must be present between the services to stop the intruder from affecting the other services. Linux supports following namespaces: IPC, UTS, Mount, PID, Network and User. Process ID (PID) is a process namespace which enables multiple process trees in the Linux. Historically, Linux maintains single process tree with root node having PID as 1, all other processes are the child processes of root node. A process, given it has sufficient privileges and satisfies certain conditions, can inspect another process by attaching a tracer to it or may even be able to kill it [4]. With Linux process namespace it is possible to have multiple nested process trees, the new nested process tree will have a root node with PID 1 and this done by some process in the parent namespace. Now the process in nested tree can have more than one PID, one for each namespace under which it falls. The processes inside the nested process tree have no knowledge of processes outside its tree.

In Linux there is one set of network interfaces and routing tables which is shared by different processes. Linux network namespace enables to share the network interfaces and routing tables between different Linux processes (or containers). Important point to be note is that we cannot assign physical interfaces to the namespace, virtual Ethernet (veth) are allocated to the network namespace. The procedure goes like this first we have to create virtual Ethernet and assign it to the network namespace. Further the network namespace is connected to the outside world via the default network namespace where the physical interface exist [5]. Mount namespaces isolate the set of file system mount points seen by a group of processes. Thus, processes in different mount namespaces can have different views of the file system hierarchy [6]. User namespace was fully implemented from kernel version 3.8 onwards; with user namespace processes can have different process id inside the user namespace and a different process id outside the user namespace. The benefit is that a process can have a root access inside its user namespace and unprivileged process outside its user space. Using the clone () system call with the CLONE_NEWUSER flag, a separate user namespace can be created [7]. UTS namespace is responsible for system identification. This includes the hostname and domain name. It allows a container to have its own hostname independently from the host system along with other containers [8]. IPC isolates the inter-process communication namely System V IPC and POSIX message queues.

III. DOCKER

Docker is a tool that makes it easy to package an application and all of its dependencies into a computer. It does this by providing a toolset and a unified API for managing kernel level such as LXC containers, cgroups and a copy on write file systems [2].

Docker is composed by three main components: the Docker daemon, the Docker client and the Docker.io registry [9]. Client server architecture is used by Docker. The Docker client is the primary interface to the user, it accepts the commands and pass it on to the Docker daemon. The client and daemon can run on same computer or can be on different computers. The Docker.io registry contains the Docker images, images are the snapshot of Docker containers. Images contains all the information required to run a container, for example an image can contain Ubuntu as operating system, apache as web server and web application installed on apache. Docker images are read only, either we can create a Docker image or download it from public repositories. We can consider a Docker image as a saved state of container. Docker registry can be private or public. Docker Hub is the public repository from where we can download Docker images built by programmers.

A. Docker Images

Docker Images are read only templates from which Docker containers are launched. Each image consists of a series of layers [11]. The Docker images can be updated or new images can be created. To update an existing image, first we have to create the container from the image, do the changes in the container and finally commit the container to save it as an image. To create a new image, we have to create a Dockerfile that contains the instructions to create the image and then use the Docker build command.

Each image consists of a series of layers. Docker makes use of union file systems to combine these layers into a single image [11]. Each layer either adds or replaces the layer below it. When the container is launched from the image, a writable layer is added on the top to capture and any changes and those changes in the container can be stored as new Docker image Fig 2. Each layer is a file system identified by unique name and can mounted when needed. Layers can be created from scratch or from the parent layer. The Fig 2 shows the container with four layers and a writable layer on top. In the Fig 2, the container have four read only layers, with unique hash codes and one writable container layer on top.

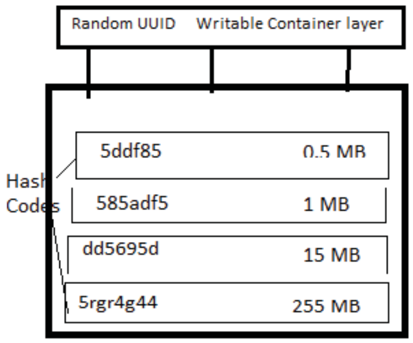

Fig. 2 Docker Container having four layers with hash code and writable top layer.

The difference between container and image is the top layer, which is writable. The benefit is that different containers can refer to the same image and changes made by containers will only affect the top layer of the container. When the container is deleted the top layer is also deleted and the image remains unchanged. If we want to save the changes made by container, then that container with changes on top layer can be saved as new image. Docker uses copy-on-write (CoW) strategy to manage images and containers. In CoW, the data is shared between different processes for reading and if any process requires modifying the data then a copy is created of data and that copy is modified, not affecting the other processes. All image and layers are stored inside the local host and managed by storage driver. Docker allows images to share layers, it means if an image has some layers which are shared by some image which is already loaded, and then only those layers are pulled which are different.

B. Storage Drivers

Storage Drivers are used to manage Docker image and Docker containers in its own unique way. Some of the storage drivers available in Docker are overlay, aufs, btrfs, devicemapper, vfs and zfs. The choice of storage driver depends on the backing file system to be used on local storage area. For example overlay and aufs can be used with ext4 and xfs, btrfs can be used with btrfs and devicemaper with direct –lvm.

AUFS is the first storage driver used with Docker. AUFS uses union mount to stack multiple directories and to provide a unified view. AUFS works at file level it means to modify the small part of the file, the complete file is moved to top layer of the container. If the file size is huge and it is located in lower layers then it takes lot of time. AUFS efficiently shares images between multiple containers, therefore making it a good choice for PaaS. AUFS is not supported by upstream kernel; so many Linux Distribution like RHEL did not use AUFS.
Red Hat and Docker collaborated to develop Devicemapper as an alternate to AUFS. Devicemapper works at block level instead of file level. It means read and write works on block level instead of complete file level, which also makes it faster than AUFS.

IV. LXC and DOCKER

LXC project is supported by Ubuntu. It is not well documented beyond Ubuntu, whereas Docker is well documented and aggressively marketed. LXC can run multiple processes (LXC is suitable replacement of VM's), whereas Docker is used to run single process or service. Initially Docker was based on LXC, but now Docker has developed its own libcontainer that uses namespace and cgroups. LXC does not support layered filesystem, whereas Docker supports layered file systems. Docker is built over Container technologies like LXC, whereas LXC is built on OS features like cgroups, namespace, resource isolation etc. Docker provides container management tools, whereas in LXC the developer has to do it himself. LXC provides process sandboxing, which is important for container portability provided by Docker. Container portability means running of same Docker image on different distributions of Linux and Hardware configuration.

V. CONTAINER ORCHESTRATION.

Container Orchestration allows users to define how to coordinate the containers in the cloud when the multi-container packaged application is deployed. Container orchestration defines not only the initial deployment of the containers, but also the management of the multi-containers as a single entity, such as availability, scaling and networking of the containers [12]. Some of the famous orchestration tools are Google's Kubernetes, Docker swarm and compose, Apache Mesos and Mesosphere Marathon.

Orchestration is a broad term that refers to container scheduling, cluster management and possibly the provisioning of additional hosts [13]. Scheduling refers to the loading a particular services on the host and running that services inside a container. Cluster management means managing group hosts; this involves adding and removing hosts from the cluster. It also involves providing a unified view of the hosts present on different clusters to the scheduler. Provisioning of additional host means adding new hosts to the existing clusters.

In June 2014, Google Developer Forum announced Kubernetes, an open source cluster manager for Docker Containers [3]. In July 2015, Kubernetes Version 1.0 was released and subsequently Google formed alliance with Linux Foundation to form the Cloud Native Computing Foundation (CNCF) and Kubernetes act as a seeding technology of CNCF. Kubernetes is based on processes that run on Docker hosts that bind hosts into clusters and manage containers [14] . Using Kubernetes, users can run containers on set of clusters. Kubernetes automatically chooses hosts on which the containers should be run using advance schedulers. Collection of containers is known as pod sharing single IP address and ports.

VI. CONTAINERS ACROSS MULTIPLE CLOUDS

Currently Docker containers are mostly focused on deployment at a single site/cloud level [10]. The limitation is that solutions cannot be scale out when on premise resources are busy or unavailable. The user cannot migrate to other providers. Kubernetes helps in managing the containers running on cluster of host located at single site, it is unable to manage to containers running on cluster of hosts located at different sites (across clouds). This limits the users to scale out the applications to other locations. The benefit of running containers across clouds is that companies can keep private data on site and public data on cloud provider sites.

The solution is to provide federation of clusters (federation of Kubernetes), in this federation multiple clusters on different sites are tied together and managed. Ubernetes is the project started by Google to provide federation of Kubernetes, where multiple clusters on different sites are tied together and managed. Ubernetes is a Google project an extension to Kubernetes that would enable the deployment of containers across K8S clusters [10].

VII. CONCLUSION

Virtualization technology is evolving very fast from last few years; it had provided new and better way of hardware utilization. Cloud computing is becoming popular and widely adopted by organizations like Google, Microsoft and IBM. Cloud computing provided independence to organization to choose their hardware requirements on real time basis, they can deploy their applications on cloud, to meet the dynamic hardware requirement of the rapidly changing and growing applications. Virtualization started with Hypervisor providing complete isolation between different and independent applications; soon it realizes that the isolation provided by Hypervisor is at the cost of performance and each virtual machine running the same copy of operating system is a hurdle for achieving the full performance of underlying hardware. The overhead involved in creating and managing the virtual machines is much more. To solve this overhead, the capabilities of operating system like namespace and cgroups are explored and operating system virtualization picked up. Operating system virtualization is lightweight and is useful in managing independent application that requires same operating system and spread over distributed systems. In OS virtualization containers replaced the virtual machines and provided easy and fast ways of creating and managing independent containers for application at the cost of isolation, but containers have better performance. LXC is the first technology to pick up in OS virtualization. LXC provides multiple process environments, which is still heavy weight i.e. inside a LXC container multiple processes can be run, which results in heavy weight processing. However using Docker, single process or service can be run inside the container. This results in very high portability of application. Docker is picking up very fast in market with support from Docker Company and with support of IT giants like Red Hat, Google, IBM and other companies. As the Docker popularity grows and companies start using it at commercial level, two new problems arises of managing the Docker container at single site and scaling the

Docker containers to multiple clouds. With this need, two new technologies are gaining popularity are Kubernetes and Ubernetes, both the projects are supported by Google and is in developing state.

REFERENCES

[1] Stephen Soltesz et al. "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors" in proceedings of the *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007, pp. 275-287

[2] Mathijs Jeroen Scheepers. "Virtualization and Containerization of Application Infrastructure: A Comparison" in *21st Twente Student Conference on IT,* 2014.

[3] David Bernstein. "Containers and Cloud: From LXC to Docker to Kubernetes." *IEEE Cloud Computing,* Vol 1, pp. 81-84, Sept. 2014.

[4] Mahmud Ridwan. "Separation Anxiety: A Tutorial for Isolating Your System with Linux Namespaces." Intenet:https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces, [May. 27, 2016].

[5] "Introducing Linux Network Namespaces." Internet: http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/, Sept. 04, 2013 [May. 27, 2016].

[6] Michael Kerrisk. "Namespaces in operation, part 1: namespaces overview." Internet: https://lwn.net/Articles/531114/, Jan. 04, 2013 [May. 27, 2016].

[7] Scott McCarty. "What's Next for Containers? User Namespaces." Internet: http://rhelblog.redhat.com/2015/07/07/whats-next-for-containers-user-namespaces/, July, 07, 2015 [May. 26, 2016].

[8] Micheal Crosby. "Creating Containers - Part 1." Internet: http://crosbymichael.com/creating-containers-part-1.html, Nov. 16, 2014 [May. 27, 2016].

[9] Andera Tossatto et al. "Container-Based Orchestration in Cloud: State of the Art and Challenges" in *proceedings of the Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2015 Ninth International Conference , 2015, pp. 70-75

[10] Moustafa AbdelBaky et al. "Docker Containers Across Multiple Clouds and Data Centers." in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 368-371.

[11] "Understand the Architecture." Internet: https://docs.docker.com/v1.8/introduction/understanding-docker/, [May. 30, 2016].

[12] Linsun. "Container Orchestration = Harmony for Born in the Cloud Applications." Internet: https://developer.ibm.com/bluemix/2015/11/04/container-orchestration-harmony-for-born-in-the-cloud-applications/, Nov. 04, 2015 [May. 30, 2016].

[13] Justin Ellingwood. "The Docker Ecosystem: Scheduling and Orchestration." Internet: https://www.digitalocean.com/community/tutorials/the-docker-ecosystem-scheduling-and-orchestration, Feb. 01, 2015 [May. 30. 2016].

[14] Claus Pahl. "Containerizatin and the PaaS Cloud." *IEEE Cloud Computing*, Vol 2, pp.24-84, May-June 2015.

[15] Barb Darro. "What big companies should know about Google's Ubernetes project". Internet: http://fortune.com/2015/06/26/why-big-companies-should-know-ubernetes/, Jun. 26, 2015 [May. 30, 2016].