Docker: An Emerging Container Technology

Prof. Jayshree Jha[1] Prof. Nutan Dhange [2] Prof. Ganesh Gourshete[3]
[1,2,3]Assistant Professor
[1,2,3]Department of Information Technology
[1,2,3]Atharva College of Engineering

Abstract— Presently many Cloud-computing platforms are based on virtual machines running on hypervisors which is shifting towards two trends that provide more flexible and efficient Cloud computing paradigm. One trend is the Resource-as-a-Service (RaaS) based Cloud, where the fine-grained resources can be rented at short time intervals. The other one is the container-based Cloud, where the lightweight containers replace the virtual machines. Applications with containers share an OS, and as a result their deployments will be considerably smaller in size than hypervisor deployments. Container-based deployment makes it possible to store hundreds of containers on a physical host. As containers use the host OS, rebooting a container doesn't imply rebooting the OS. In Linux implementations it is known that there's a great degree of binary application portability among all Linux variants, with libraries and binaries sporadically required to complete the portability. Therefore, it's rational to have one container package that will run on almost all Linux-based clouds. Docker is an open source project providing a systematic way to automate the faster deployment of Linux applications inside portable containers.

Key words: Cloud Computing, Hypervisor, Virtual Machine, Container, Docker

I. INTRODUCTION

Cloud computing is a computing paradigm, where a large pool of systems along with their resources are connected in private or public networks, to provide dynamically scalable infrastructure for application, data and content storage. With the advent of this technology, the cost of computation, application hosting, file storage and delivery is reduced significantly. More formally Cloud computing has been defined by the National Institute of Standards and Technology as "resource pooling," where the "provider's computing resources are pooled to serve several consumers using a multitenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer requirement."[1] It also demands for "dynamic flexibility," where "capabilities can be flexibly provisioned and released, in some cases dynamically, to scale rapidly outward and inward commensurate with demand." Cloud computing is based on virtualization techniques to accomplish this elasticity of large-scale shared resources. Virtualization demand some kind of technology that provides an isolation and multi-tenancy layer, and where computing resources are split up and dynamically shared using an operating technique that implements the specified multitenant model. Most of the current commercial Cloud computing system is based on Virtual Machine (VM) technology to facilitate virtualization. Each VM necessitate the loading of full guest OS images in addition to the binaries and libraries. Full OS image loading creates RAM and disk storage requirements and also is slow on startup. Linux Container does have this problem as it is based on kernel technology and is able to run a multitude of process, each running in an isolated environment. This technology is known as container based virtualization. Docker is a tool which makes it simple to package an application with all of its dependencies into such container and also provide easy provisioning and image construction. [2]

II. VIRTUALIZATION AND CONTAINER

The ever-growing popularity of Cloud computing led the provider to look up for different resource-sharing solutions to meet the needs of infrastructures resources. Hypervisor based Virtual machines have been the backbone for cloud computing at the infrastructure layer, as virtual machines are providing virtualized guest operating systems.

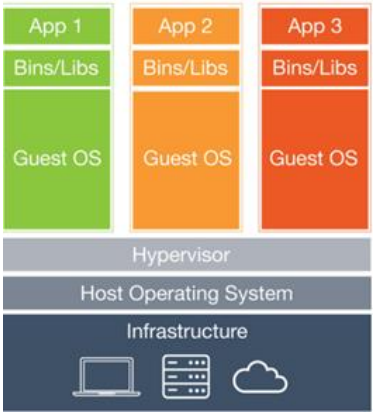

Fig. 1: Virtual Machine [2]

Figure 1 represents the structure of a virtual machine. Each virtual machine includes the application, the necessary binaries and libraries and an entire guest operating system - all of which may require large storage area on host machine. Containers are a similar, but lighter solution for virtualization. They provide a lightweight virtual environment that groups and isolates a set of processes and resources such as memory, CPU, disk, etc., from the host and any other containers. This isolation guarantees the security prospect as any processes residing inside the container cannot see any processes or resources outside the container. Containers run on a single machine and all of them share the same kernel so that they start instantly and thus make more efficient use of RAM. Images are constructed from layered filesystems so they can share common files, making disk usage and image downloads much more efficient.

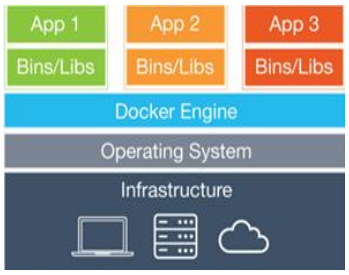

Fig. 2: Container [2]

The difference between a container and a full-fledged VM is that all containers share the same kernel of the host system. This gives them the advantage of being very fast with almost low performance overhead compared with VMs. They also utilize the different computing resources better because of the shared kernel.

However, like everything else, sharing the kernel also has its set of shortcomings. Two major shortcomings are discussed below:

- Type of containers - Type of containers that can be installed on the host system should be compatible with the kernel of the host. Hence, Windows container can't be installed on a Linux host or vice-versa.
- Isolation and security - The isolation between the host and the container is not as strong as hypervisor-based virtualization since all containers share the same kernel of the host and there have been cases in the past where a process in the container has managed to escape into the kernel space of the host.

  There are several ongoing researches in this which are working towards overcoming these shortcomings.

Figure 2 represents the structure of a container. Containers include the application and all of its dependencies, but share the kernel with other containers. They run as an isolated process in userspace on the host operating system. They're also not tied to any specific infrastructure [2].

Containers can be classified into two types based upon the area where they are used: OS containers and Application Container. OS containers are used when a fleet of identical or different flavors of distros are executed. Most of the time containers are created from templates or images that determine the structure and contents of the container. Thus it creates containers that have identical environments with the same package versions and configurations.

The idea behind application containers is that for each component in the application a separate container is created. This approach works especially well when a distributed, multi-component system using the microservices architecture is deployed. This gives independence to different team working on the same project to package their application as a single deployable container and can communicate with each other using the APIs and protocols that each of them support.

Different Container technologies which are mainly used for creating OS container are LXC, OpenVZ, Linux VServer, BSD Jails and Solaris zones. While OS containers are designed to run multiple processes and services, application containers are designed to package and run a single service. Container technologies like Docker and Rocket are examples of application containers.

III. DOCKER CONTAINERS

Docker is an open source product that has much capability from previous technologies such as LXC containers, Git type of version control and operation system virtualization. [3] With containers, applications share an operating system and whenever possible, also binaries and libraries. The result is that deployments using Docker will be drastically smaller in size than hypervisor deployments, making it possible to store hundreds of containers on a single physical host.

Basically, Docker extends LXC with a kernel and application level API that both run processes in isolation [3]. Docker relies heavily on two pieces of Linux kernel technology which are namespaces and control groups [4]. Docker make use of namespaces to completely segregate an application's view of the underlying operating environment, including networking, user IDs, file systems and all other processes [3]. Control groups are designed for organizing available resources to a container. These resources can be restricted of access to other processes [4].

Docker containers have many already described features, but one of the core ones is Docker engine, which builds and runs Docker containers. Docker containers run on Docker engine, which controls all containers. Docker client provides the user interface (UI) for user and also the interaction to containers. Therefore, the Docker daemon communicates with client and thus sends instructions to execute, deliver or build containers. [5].

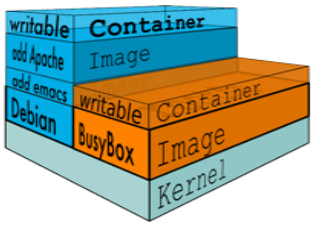

Fig. 3: Docker Container [2]

Figure 3 represents the framework of Docker container which wrap up a piece of software in a complete file system that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in [2].

Docker's advantages come from its five key factors in terms of usability: portable deployments, rapid delivery, scalability, faster build times and higher density with better performance [6]. Because of these promising features lots of research work is going on in this area.

IV. RELATED WORK

Ever since Docker has been introduced, it has been a well-covered research topic. There have been several research papers on Docker technology which mainly discuss the usefulness of Docker technology in overcoming challenges in virtualization and other fields.

Sean McDaniel, Stephen Herbein, and Michela Taufer in their paper [7] have discussed about the resource contention issue among the applications due to multiple application sharing same resources. They have proposed a two –tiered approach which augment Docker and Docker Swarm to be able to monitor and manage I/O in Linux containers. They have combined these capabilities that allows for better control, more accurate load balancing, and higher resource utilization without the effects of contention.

Carl Boettiger [8] has discussed about the usability of Docker containers in distribution of reproducible research. Key feature of Docker which ca be utilized in reproducible research is its ability to mimic as closely as possible the current workflow of and development practices of the user. Code executing inside a container on a local machine can appear identical to natively running code, but with the added benefit entire computational environment can be recreated with the help of few simple command. This work best with the Docker containers.

Claus Pahl [9] in his paper discussed about the huge potential of dockers like container technology in PaaSs clouds which will substantially advance Paas technology towards distributed heterogeneous clouds through light-weightness and interoperability.

Li Li, Tony Tang and Wu Chou [10] in their research paper have proposed a REST service framework for resource management using Docker container. Container based Docker technology is used for deployment of Ron based REST APIs to normalize the access to heterogeneous resources, including CPU, memory and storage.

V. CONCLUSION

In this research paper we have observed that Docker has huge potential in overcoming the challenges of distributed heterogeneous cloud computing. Apart from this we have also observed the usability of docker in reproducible research, Paass cloud and REST service. Overall Docker is really promising technology, which should be considered as replacement for conventional virtualization. Docker has superior features that guarantee simplified usability and scalability. However, for better performance and security heavy configuration is needed. Also, for the requirements of cloud giants better clustering system is needed.

REFERENCES

[1] P. Mell and T. Grance, The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology, NIST Special Publication 800-145, 2011.
[2] Docker. URL https://www.docker.io/. Accessed: 2013-05-30.
[3] Bernstein, D,Containers and cloud:From LXC to docker to kubernets. Cloud Computing, IEEE, 2014.
[4] Anderson, C., Docker[software engineering]. IEEE Software,2015
[5] Bui, T. , Analysis of Docker security, Aalto university T-110.5291, Seminar on Network Security,2014.
[6] Joy, A..M., Performance comparision between linux containers and virtual machines, International Conference on Advances in Computer Engineering and Applications,2015.
[7] Sean McDaniel, Stephen Herbein, and Michela Taufer, IEEE International Conference on Cluster Computing: A Two-Tiered Approach toI/O Quality of Service in Docker Containers,2015.
[8] Carl Boettiger, An introduction to Docker for reproducible research, SIGOPS Operating System Rev, 2015.
[9] Claus Pahl, Containerization and the Paas Cloud, IEEE cloud computing, 2015.
[10] Li Li, Tony Tang and Wu Chou, A REST Service Framework for Fine-Grained Resource Management in Container-Based Cloud, 8th International Conference on Cloud Computing, 2015