International Journal of Advance Research in Science and Engineering
Vol. No.5, Issue No. 04, April 2016
www.ijarse.com
IJARSE
ISSN 2319 - 8354

DOCKER: FUTURE TO VIRTUALIZATION

Siddhi Choudhari[1], Chetan Patil[2]

[1,2] *Computer Engineering Department, Lokmanya Tilak College of Engineering, Mumbai University, India*

ABSTRACT

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, light weight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud.

Keywords*: *Container, Docker, Open Source, Virtual Machine

I. INTRODUCTION

In a normal virtualized environment, one or more virtual machines run on top of a physical machine using a hypervisor like Xen, Hyper-V etc. Containers on the other hand run on user space on top of operating systems kernel. It can be called as OS level virtualization. Each container will have its isolated user space and you can run multiple containers on a host, each having its own user space. It means you can run different Linux systems (containers) on a single host. For example, you can run a RHEL and SUSE container on an Ubuntu server. The Ubuntu Server can be a virtual machine or a physical host. Containers are isolated in a host using the two Linux kernel features called namespaces and control groups. Container is not a new concept. Google has been using their own container technology in their Infrastructure for years. Solaris Zones, BSD jails, LXC are the few Linux container technology that has been around for years. In a nutshell, containers are a method for isolation and resource control, much like traditional virtualization, just without the hypervisor overhead. [1]

II. CONTAINER BASED VIRTUALIZATION

Resource virtualization consists of using an intermediate software layer on top of an underlying system in order to provide abstractions of multiple virtual resources. In general, the virtualized resources are called virtual machines (VM) and can be seen as isolated execution contexts. There are a variety of virtualization techniques. Today, one of the most popular is the hypervisor-based virtualization, which has Xen, VMware and KVM as its main representatives. The hypervisor based virtualization, in its most common form (hosted virtualization), consists of a virtual machine monitor (VMM) on top of a host OS that provides a full abstraction of VM. In this case, each VM has its own operating system that executes completely isolated from the others. This allows, for instance, the execution of multiple different operating systems on a single host. A lightweight alternative to the hypervisors is the container-based virtualization, also known as Operating System Level virtualization. Container – based virtualization is a virtualization method that uses a single kernel to run multiple instances on a

single operating system. This kind of virtualization partitions the physical machines resources, creating multiple isolated user-space instances. As can be seem, while hypervisor-based virtualization provides abstraction for full guest OS's (one per virtual machine), container-based virtualization works at the operation system level, providing abstractions directly for the guest processes. In practice, hypervisors work at the hardware abstraction level and containers at the system call/ABI layer. [2]

2.1. Introduction To Container

Operating-system-level virtualization is a server virtualization method where the kernel of an operating system allows for multiple isolated user space instances, instead of just one. Such instances (often called containers, virtualization engines (VE), virtual private servers (VPS), or jails) may look and feel like a real server from the point of view of its owners and users.

On Unix-like operating systems, this technology can be seen as an advanced implementation of the standard cheroot mechanism. In addition to isolation mechanisms, the kernel often provides resource management features to limit the impact of one container's activities on the other containers.

What does a container provide that a VM does not?

Simple deployment:

By packaging your application as a singularly addressable, registry-stored, one-command-line deployable component, a container radically simplifies the deployment of your app no matter where you're deploying it.

Rapid availability:

By abstracting just the OS rather than the whole physical computer, this package can "boot" in ~1/20th of a second compared to a minute or so for a modern VM.

Leverage micro services:

Containers allow developers and operators to further subdivide compute resources. If a micro VM instance seems like overkill for your app, or if scaling an entire VM at a time seems like a big step function, containers will make a big, positive impact in your systems.

2.2. Advantages Of Container

An obvious advantage is that a developer has, in their system, plenty of compute power to run multiple containers, making for easier and faster development. While it is certainly possible to run several virtual machines on a system, it's far from fast, easy, or lightweight. But if you're running thousands of programmatically driven tests per day, this starts to add up. With a container, you could do thousands of simple tests at the same cost, amounting to large savings for your production applications.

Another implication is the composability of application systems using this model, especially with applications using open source software. While it might be a daunting systems administration (not to mention pronunciation) task for a developer to install and configure MySQL, memcached, MongoDB, Hadoop, GlusterFS, RabbitMQ, node.js, nginx, etc. together on a single box to provide a platform for their application, it is much easier and vastly lower risk to start a few containers housing these applications with some very compact scripting. Consider the amount of error-prone, specialized, boilerplate work this model eliminates. Add to this the public

registration of canonical implementations for these types of core building blocks, and you have the beginnings of a real ecosystem for quality components.

2.3. HOW CONTAINER SOLVES the PROBLEM OF VIRTUALIZATION?

Over the past couple of years, hypervisor-based virtualization has become a major trend in virtualization. It's not hard to understand why; it is flexible and allows you to install nearly nay operating system. If you don't need many different operating systems running simultaneously, container-based virtualization is a good alternative and offers virtualization performance benefits. Consider an example where N goods are to be transported in N transporting systems. A standard container is loaded with virtually any goods, and stays sealed until it reaches final delivery. In between, it can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another. Moreover, there are two basic conditions; 1) the transport should be quick and smooth, 2) Interaction of goods. For example, Coffee beans next to spices. By using N*N matrix, with rows depicting goods and columns depicting vehicles, this problem can be solved. Each good can be sealed in a container and transported by any vehicle.

2.4. Architecture of Container

A Container consists of an operating system, user-added files and meta–data. As we have seen each container is built from an image. That image tells docker what the container holds, what process to run when the container is launched and a variety of other configuration data.

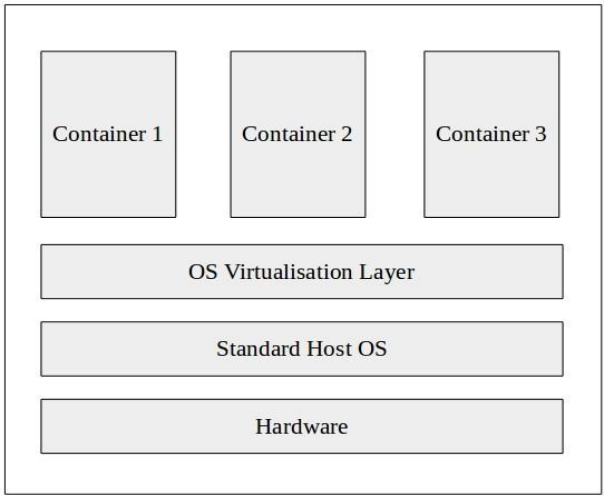

Fig.1. Container Architecture

2.5. Disadvantages of Container

Containers are not a panacea, particularly for enterprises that have already invested in hypervisor-centric technologies like SR-IOV, VT-D and Network Function Virtualization. In each case, the technology is designed to connect a virtual machine guest directly to a hardware or fabric by means of a special driver sitting inside the guest kernel. Since there's no separate guest kernel with containers, there's nothing to insert a driver into, and therefore no apparent way to make use of the technology. However, analysis only shows the technology doesn't work if one thinks in the hypervisor paradigm. Docker is a perfect solution to overcome these drawbacks.

III. DOCKER

Brandon Butler at Network World has come up with one of the best definitions so far,

"Docker is both an open source project and the name of a startup that focuses on Linux Containers. Containers are the idea of running multiple applications on a single host. It's similar to compute virtualization, but instead of virtualizing a server to create multiple operating systems, containers offer a more lightweight alternative by essentially virtualizing the operating system, allowing multiple workloads to run on a single host."

Docker is a popular open source project based on Linux containers. Docker is written in go and developed by Dotcloud (A PaaS Company). Docker is basically a container engine which uses the Linux Kernel features like namespaces and control groups to create containers on top of an operating system and automates application deployment on the container. It provides and light weight environment to run your application code. Docker has an efficient workflow for moving you application from developer's laptop, test environment to production. Docker is incredibly fast and it can run on host with compatible Linux Kernel.

Docker, a new container technology, is hotter than hot because it makes it possible to get far more apps running on the same old servers and it also makes it very easy to package and ship programs. Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Consisting of Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub, a cloud service for sharing applications and automating workflows, Docker enables apps to be quickly assembled from components and eliminates the friction between development, QA, and production environments. As a result, IT can ship faster and run the same app, unchanged, on laptops, data center VMs, and any cloud. [2]

3.1. How Docker Solves The Problem?

Virtual Machines:

Each virtualized application includes not only the application - which may be only 10s of MB - and the necessary binaries and libraries, but also an entire guest operating system - which may weigh 10s of GB.

Docker:

The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process in user space on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient.

For Developers:

With Docker, developers can build any app in any language using any tool chain. "Dockerized" apps are completely portable and can run anywhere - colleagues' OS X and Windows laptops, QA servers running Ubuntu in the cloud, and production data center VMs running Red Hat. Developers can get going quickly by starting with one of the 13,000+ apps available on Docker Hub. Docker manages and tracks changes and dependencies, making it easier for sysadmins to understand how the apps that developers build work. And with Docker Hub, developers can automate their build pipeline and share artefacts with collaborators through public or private repositories. Docker helps developers build and ship higher-quality applications, faster.

For Sysadmins:

Sysadmins use Docker to provide standardized environments for their development, QA, and production teams, reducing "works on my machine" finger-pointing. By "Dockerizing" the app platform and its dependencies, sysadmins abstract away differences in OS distributions and underlying infrastructure. In addition, standardizing on the Docker Engine as the unit of deployment gives sysadmins flexibility in where workloads run. Whether on-premise bare metal or data center VMs or public clouds, workload deployment is less constrained by infrastructure technology and is instead driven by business priorities and policies. Furthermore, the Docker Engine's lightweight runtime enables rapid scale-up and scale-down in response to changes in demand. Docker helps sysadmins deploy and run any app on any infrastructure, quickly and reliable [1]

3.2. Docker Comparison

On the other hand Docker containers are executed with the Docker engine rather than the hypervisor. Containers are therefore smaller than Virtual Machines and enable faster start up with better performance, less isolation and greater compatibility possible due to sharing of the host's kernel.

Table1. Docker Comparison

	VIRTUAL MACHINE	CONTAINER	DOCKER
Lightweight	No	Yes	Yes
Needs Guest Os In The Application	Yes	Yes	No
Needs Hypervisor In The Application	Yes	No	No
Strong Resource Management	Yes	No	Yes
Portable	No	Yes	Yes
Need Less Memory And Time	No	No	Yes

Docker Containers have much more potential than Virtual Machines. It's evident as Docker Containers are able to share a single kernel and share application libraries. Containers present a lower system overhead than Virtual Machines and performance of the application inside a container is generally same or better as compared to the same application running within a Virtual Machine.[8]

3.3. Docker Architecture

Docker Components: Docker is composed of following four components.

- Docker Client and Daemon

Docker has client-server architecture. Docker Daemon or server is responsible for all the actions that are related to containers. The daemon receives the commands from the Docker client though client or REST API's.

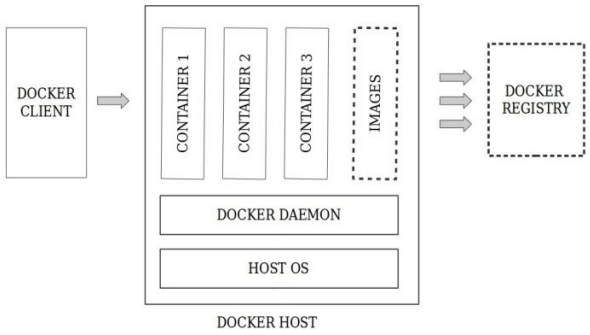

Fig.2. Docker Architecture

- Images

Images are the basic building blocks of Docker. Containers are built from images. Images can be configured with applications and used as a template for creating containers. Images are organized in a layered manner. Every change in an image is added as layer on top of it.

- Docker Registry

Docker registry is a repository for Docker images. Using Docker registry, you can build and share images with your team. A registry can be public or private. Docker Inc provides a hosted registry service called Docker Hub. It allows you to upload and download images from a central location. Docker hub acts like git, where you can build your images locally in your laptop, commit it and then can be pushed to the Docker hub.

- Containers

Container is the execution environment for Docker. Containers are created from images. It is a writable layer of the image. You can package your applications in a container, commit it and make it a golden image to build more containers from it. Containers can be started, stopped, committed and terminated. If you terminate a container without committing it, all the changes made to the container will be lost.

IV. CONCLUSION & FUTURE WORK

The best feature of Docker is collaboration. Docker images can be pushed to a repository and can be pulled down to any other host to run containers from that image. Moreover Docker hub has thousands of images created by users and you can pull those images down to your hosts based on your application requirements.

Naturally, Docker is a Technology of the year shoo-in not only for the agility it brings to developers, but for how far the project has come in the course of the past year. Docker's entire networking model has been upgraded .Its handling of storage has been reworked to such degree that it is now forming the basis for a cottage industry of third-party products. It has finally done away with the need to run containers as root. And it has gained a smorgasbord of additional tooling.

Also striking is the way Docker has influenced the direction and development of not only other software projects , but entire software industries VMware sensed, correctly, that containers were providing a better solution to many problems VMs were originally meant to solve , and reworked much of its product line to welcome containers as first-class citizens.

Google, Amazon, Red Hat, IBM, and Cisco- every data centre and cloud vendor is catching Docker fever. It has been long time since any one piece of software came along that had such a transformative effect, and it will be nothing short of fascinating to watch how Docker and its partners continue to shepherd its evolution through the coming year.

V. ACKNOWLEDGMENT

As an author, I am using this opportunity to express my gratitude to Prof. Jayendra Jadhav who guided us to work in this area. I also want to thank Prof. Pravin Nikumbh, Head of Computer Engineering Department, Lokmanya Tilak College of Engineering, Navi Mumbai, for giving space to work. Last but not the least, I want to thank our parents and friends for motivational support, aspiring guidance, invaluably constructive criticism and friendly advice at each and every stage.

REFERENCES

[1] http://events.linuxfoundation.org/sites/events/files/ slides / lcna13_petazzoni.pdf

[2] Search Server Virtualization, Retrieved on 21st-January-2015, fromhttp://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization

[3] Understanding Docker – Docker Documentation, Retrieved on 24th-January-2015, fromhttps://docs.docker.com/introduction/understanding-docker/

[4] System Administration Screen casts, Retrieved on 20th-January-2015, fromhttps://sysadmincasts.com/episodes/31-introduction-to-docker

[5] The New Stack, Retrieved on 24th-January-2015, fromhttp://thenewstack.io/a-great-introduction-to-docker-and-where-its-all-going/

[6] Red hat developer, Retrieved on 25th-January-2015, from http://developerblog.redhat.com/2014/05/15/practical-introduction-to-docker-containers/

[7] http://www.devopscube.com/wp-content/uploads/2014/12/docker-architecture-techtip39.png

[8] DevOps.com, Retrieved on 27th-January-2015, from http://devops.com/blogs/devops-toolbox/docker-vs-vms/

[9]. Dilip G. Durbude, B. B. Jadia, R. S. Sontakke. "LONG TERM HYDROLOGICAL SIMULATION MODELLING BASED ON PHYSICAL CHARACTERISTICS OF WATERSHE." *International Journal of Advanced Technology in Engineering and Science* 3.Special Issue No. 01 (2015): 28-41.