2015 International Conference on Advances in Computer Engineering and Applications (ICACEA)
IMS Engineering College, Ghaziabad, India

Performance Comparison Between Linux Containers and Virtual Machines

Prof. Ann Mary Joy
Department of Computer Science and Engineering,
IMS Engineering College,
Ghaziabad, India
annmaryjoy16@gmail.com

***Abstract*—With the advent of cloud computing and virtualization, modern distributed applications run on virtualized environments for hardware resource utilization and flexibility of operations in an infrastructure. However, when it comes to virtualization, resource overhead is involved. Linux containers can be an alternative to traditional virtualization technologies because of its high resource utilization and less overhead. This paper provides a comparison between Linux containers and virtual machines in terms of performance and scalability.**

Keywords—containers; docker; virtualization; cloud computing; hypervisor; kubernetes.

I. INTRODUCTION

Virtualization is playing a major role in private and public cloud Deployments. Most of the public cloud providers such as Amazon EC2, Google Compute Engine and Microsoft Azure leverage virtualization technologies to power their public cloud infrastructure. Virtual machines are powered by hypervisors. Hypervisor is software which provides isolation for virtual machines running on top of physical hosts. It is responsible for running different kernels on top of physical host. This in turn makes the application and process isolation very expensive. There will be a big impact if compute resources can be used more efficiently. The most popular hypervisors today are VMware, KVM, Xen and Hyper-V. Typical virtualized datacenter architecture is shown in Fig.1

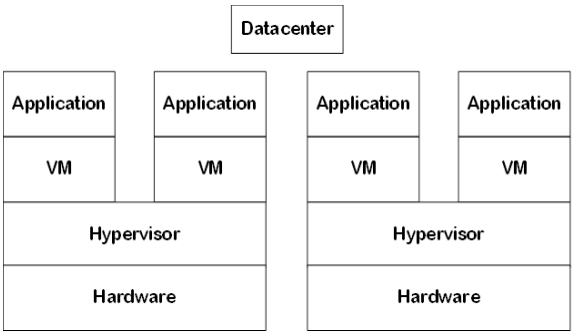

Fig. 1. Virtualized datacenter

The advancements in Linux kernel lead to the development of container based virtualization. A Linux container (LXC) holds a different approach than a hypervisor. It could be used as an alternative for hypervisor based virtualization. Linux containerization is an approach where you can run many processes in an isolated fashion. It uses only one kernel for multiple isolated environments or operating systems. The main isolation factor for containers are two Linux features called namespaces and control groups [9]. Moreover LXC's are very light weight because it does not virtualize the hardware instead all containers on physical host uses the single host kernel efficiently using process isolation.

Following are the features which make container useful and attractive to application developers and infrastructure administrators alike:

A. Portable Deployments

As containers are portable, the applications can be bundled in to a single unit and can be deployed to various environments without making any changes to the container.

B. Fast application delivery

The workflow of containers makes it easy for the developers, system administrators, QA and release team to collaborate and deploy the applications to production environment really fast.

Because of the standard container format, only the developers have to worry about the applications running inside the container and system administrators have to worry about deploying the container onto the servers only. This well segregated container management leads to faster application delivery.

Moreover, building new containers is fast because containers are very light weight and it takes seconds to build a new container. This in turn reduces the time for testing, development and deployment. A container can be built in iterations too. Thus providing a good visibility on how the final application has been built.

For example, when an application is packaged in a container by the developer, it can be shared among other team members. After that it can be pushed to the test environment for various tests. From the test environment you can then push all the tested containers to the production environment.

C. Scale and deploy with ease

Containers can virtually run on any Linux system as well as can be deployed on cloud environments, desktops, on

978-1-4673-6911-4/15/$31.00©2015 IEEE

premise datacenters, physical servers and so on. You can move containers from your desktop environment to cloud and back to the physical servers easily and quickly.

Another interesting factor about container is scalability. Scaling up and down containers is blazingly fast. You can scale up containers from one to hundred's and scale it down when it is not needed. Thus containers are ideally suited for scale out applications architected and built for the public cloud platforms.

D. Higher workloads with greater density

More container applications can be deployed on a host when compared to virtual machines. Since there is no need for container to use a hypervisor, the server resources can be well utilized and cost of extra server resources can be reduced. Because containers do not use a full operating system, the resource requirements are lesser as compared to virtual machines.

II. BACKGROUND

There are various advantages of using containers as compared to virtualization in terms of performance and scalability. A container based solution works well if you intend to run many hundreds of guests with a particular operating system, because they carry lesser overhead. The number of virtual machines available with container approach can be much higher as compared to virtualization as resources are available to the application rather than being consumed by multiple Guest OS instances running on a host.

One area where containers are weaker than VMs is isolation. VMs can take advantage of ring-1 hardware isolation such as that provided by Intel's VT-d and VT-x technologies [8]. Such isolation prevents VMs from 'breaking out' and interfering with each other. The difference between a VM and a container is shown in Fig. 2

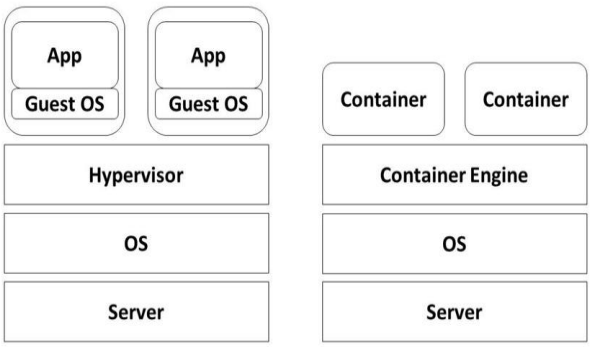

Fig. 2. Virtual machines vs containers

In this paper, a tool named Docker is used for running containers. And for testing the container scalability, a cluster manager named kubernetes is used.

A. Namespaces and Cgroups

In Linux there are six kinds of namespaces which can offer process level isolation for Linux resources. Namespaces ensure that each container sees only its own environment and doesn't affect or get access to processes running inside other containers. In addition, namespaces provide restricted access to file systems like chroot, by having a directory structure for a container.

The container can see only that directory structure and doesn't have any access to any level above it. Namespaces also allow containers to have its own network devices, so that each container can have its own IP address and hostname. This lets each container run independently of each other.

B. Docker

Docker leverages LXC (Linux Containers), which consists of Linux kernel features such as cgroups and namespaces for efficient process isolation and resource control [1]. Docker leverages a copy-on-write file system and this allows Docker to instantiate containers quickly because it leverages the pointers to the existing files. Copy-on-write file system also provides layering of containers, thus you can create a base container and then have another container which is based on the base container. A typical container architecture based on namespaces and control groups is shown in Fig. 3

C. Kubernetes

Kubernetes is a cluster management tool developed by Google for managing containerized applications. You can make a bunch of nodes appear as a one big computer and deploy container applications to your public cloud and private cloud. It abstracts away the discrete nodes and optimizes compute resources.

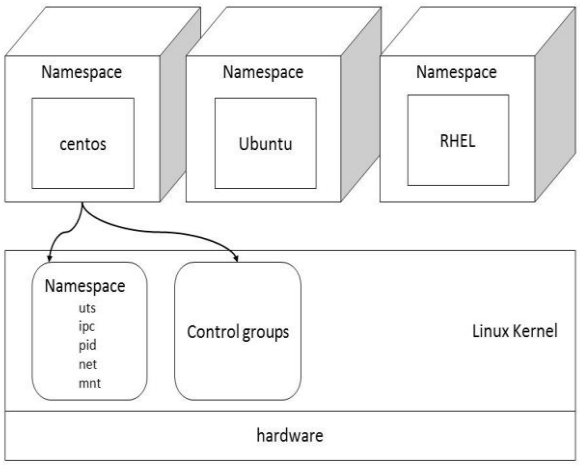

Fig. 3. Namespaces and control groups

Kubernetes uses a declarative approach to get the desired state for the applications mentioned by the user. When an application is deployed on a kubernetes cluster, the kubernetes master node decides in which underlying host the application has to be deployed. Kubernetes scheduler does the job of application deployment [2]. Moreover, kubernetes self-healing, auto restarting, replication and rescheduling mechanisms make it more robust and suitable for container based application.

D. Copy on write file system

In normal file system like ext4, all the new data will be overwritten on top of existing data and creates a new copy. Unlike other Linux file systems copy on write file system never overwrites the live data, instead it does all the updating using the existing unused blocks in the disk using copy on write functionality (COW). The new data will be live only when all the data has been updated to the disk.

For example, file systems are divided in to number of blocks, let's say 16 blocks. So each innode will have 16 pointers to blocks. If a file stored is less than 16 blocks, the innode will point to the block directly. If the data exceeds 16 blocks, the 16 block will become a pointer to more blocks creating an indirect pointer as shown in fig. 4.

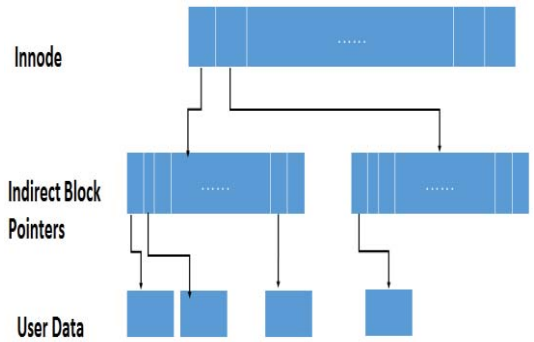

Fig. 4. Copy on write

When you modify an existing data, it will be written on unused blocks in the file system leaving the original data unmodified. All the indirect block pointers have to be modified in order to point to the new blocks of data as shown in fig. 5. But the file system will copy all the existing pointers to modify the copy. File system will then update the innode again by modifying the copy to refer to the new blocks of indirect pointers. Once the modification is complete, the pointers to original data remain unmodified and there will be new set of pointers, blocks and innode for the updated data.

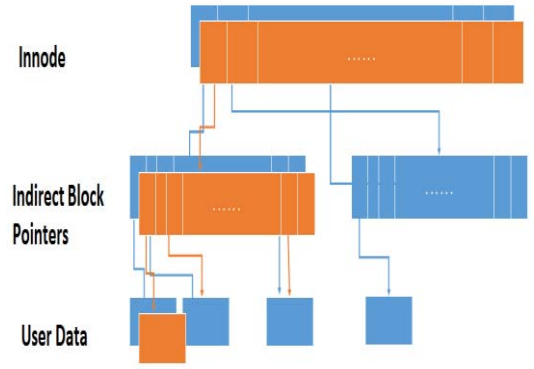

Fig. 5. Copy on write mechanism

Resources are handled using Copy on Write (COW) when same data is utilized by multiple tasks. When an application requests data from a file, the data is sent to memory or cache. Individual applications then have their own memory space. In the case when multiple applications request the same data, only one memory space is allowed by COW and that single memory space is pointed to by all applications. An application, which is changing data, is given its own memory space with the new updated information. The other applications continue using the older pointers with original data.

III. Related Work

In the previous literatures, the VMware virtualization is compared with VServer container technology. The findings reported the overhead involved in hypervisor based VMware virtual machines [3].

Another research is performed based on High performance computing using virtualized environments using OpenVZ containers. In his study he concluded that the container based isolation for network and memory was not matured but it performed well in terms of CPU utilization [4].

A recent study on network performance on Xen virtualized Amazon compute cloud revealed that even though the networking was not overused, the virtualization has huge impact on throughput [5].

Also, another comparison between major hypervisors and found difference in network throughput and disk I/O performance based on the hypervisor used [6].

It can be found from the research while comparing among Xen, KVM and OpenVZ container technology using micro benchmarks. OpenVZ is considered as the predecessor of Linux containers (LXC) [7].

A. Contribution

The studies done in the past primarily focused on comparing the virtual machine environments as well as the performance of application inside the virtual machine. The macro benchmarks comparing virtual machines and Linux containers in terms of performance and scalability was not addressed in those studies. The load on a particular virtual machine influences the communication to other virtual machines. Moreover, on a virtual machine failure, the time taken for vertical scaling of virtual machines affects the performance of the application. For addressing the difference between Virtual machines and containers in terms of network latency and fault tolerance, the macro benchmarks are used by focusing the whole application infrastructure.

Scalability for virtual machines depends on the application architecture. Virtual machines may be clustered or it can be set to auto scale based on CPU loads. In the same way, Linux containers can be clustered and scaled based on resource needs. Docker has made Linux container more portable and easily deployable on machine which supports Linux kernel. The performance is measured for same application running on Virtual machine and container environments to benchmark the latency involved in the two environments. And scalability is

measured based on the time taken for virtual machines and containers to scale up and down based on failure.

IV. Performance And Scability Comparision

The set up for comparing virtual machines and Linux containers include AWS ec2 cloud and two physical servers. Ubuntu 14.04 server operating system each having 2GB RAM is used for ec2 virtual machines and for hosting Docker on physical servers.

A. Application perfomace comparision

The set up for application performance comparison includes a front end application server hosting Joomla php application and backend server hosting postgreSQL database for storing and retrieving data for the Joomla application. The Joomla application is filled with test data. Jmeter utility is used for stress testing the application by simultaneously sending requests to the Joomla application till the server experiences problems in performance [10].

In 600 seconds JMeter tried to complete as many requests as it could, with a growing volume of parallel requests. At the initial time t=0ms, the testing software started with one concurrent request and send the next request only when the preceding one is ended. The number of synchronized requests was increased linearly, till 520 seconds it reached approximately 80 concurrent requests.

Fig. 6 show the number of requests that were successful within 600 seconds. The figure shows that Docker container was able to process far more requests within the 400 seconds, more than three times as many as virtual machine

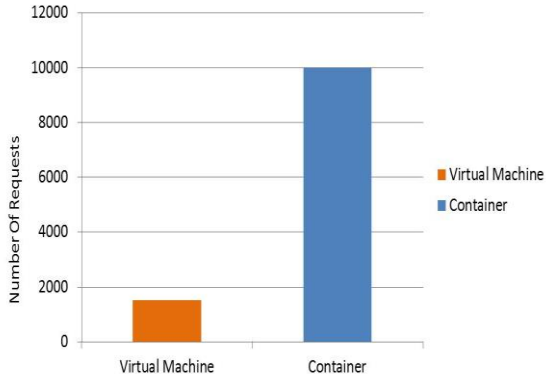

Fig. 6. Processed requests in 600 seconds

Fig. 7 show that the virtual machine takes more time to process even a single request. The trend line drawn in-between the data points take the average of 20 data points. Also, it is found that the virtual machine does not perform as consistently as Docker, even when it has physical memory accessible. When the number of links increased further than the server could handle using physical memory, the host starts swapping memory between its hard disk and physical memory.

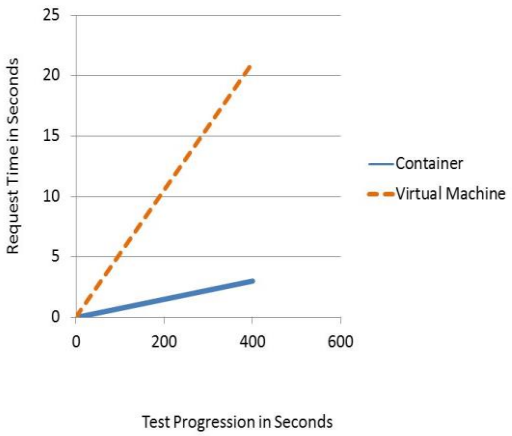

Fig. 7. Request time for first 400 seconds

B. Scalability comparision

For virtual machine scalability test, AWS ec2 auto scaling feature is used [11]. Auto scaling feature scales up virtual machine instances when it reaches maximum CPU load. Here the auto scaling instance count is set to 1. For testing the Linux container scalability, kubernetes clustering tool is used. Using kubernetes the two physical servers with Docker host are clustered.

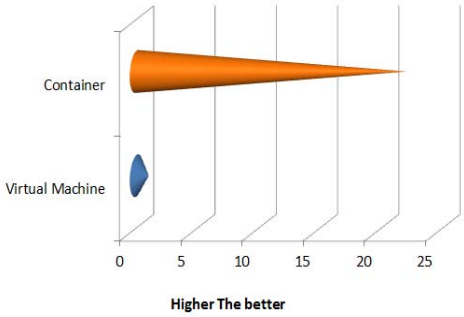

Fig. 8. Scalability between VM's and Containers

In this experiment both virtual machine and container environments runs a load balanced WordPress application. Jmeter is used to trigger the scaling by increasing the CPU utilization above 80% for scaling up the virtual machine. Same Jmeter is used for sending concurrent request to front end WordPress application to trigger container scaling using kubernetes cluster manager.

The results show that the time taken by the container to scale and process the service request is very much less than the time taken to scale a new virtual machine to handle the request. The calculated time for scaling up virtual machine is three minutes while a container took only eight seconds to scale up and handle the request. This is a big difference in scalability point of view. Containers outperformed virtual machines by 22x times faster in scalability as shown in Fig. 8.

V. CONCLUSION

The macro benchmarks by comparing virtual machines and Linux containers are shown in this paper. Containers have outperformed virtual machines in terms of performance and scalability. Because of its better scalability and resource utilization, containers can be used for application deployments to reduce resource overhead. However, there are use cases where virtual machines would be a better fit than Linux containers. One of the use cases is running applications with business critical data. Containers run with root privileges and

REFERENCES

[1] Docker. URL https://www.docker.io/
[2] Kubernetes URL https://www.kubernetes.io
[3] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.
[4] Benjamin, Vincent Neri and Frank Cappello. Scalability comparison of four host virtualization tools. Journel of Grid Computing.
[5] Miguel G Xavier, Marcelo V Neves, Fabio D Rossi and Cesar AFDe Rose. Performance Evaluation of container based virtualization for high performance computing environment.
[6] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center.
[7] J.Hwang, S.Zeng, F.Wu and T.Wood. A component based performance comparison of four hypervisors. In Integrated Network Management, 2013.
[8] Jianhua Che, Yong Yu, Congcong Shi and Weimin Lin. A synthetical performance evaluation of openvz,xen and kvm. In Services Computing Conference , 2010 IEEE.
[9] LXC URL https://linuxcontainers.org/
[10] Jmeter URL http://jmeter.apache.org/
[11] AWS autoscaling http://aws.amazon.com/autoscaling/