A Comparison of LXD, Docker and Virtual Machine

Sapan Gupta, Deepanshu Gera

Abstract—In the virtualization industry, the container-based virtualization has recently seen a sudden spurt in interest, thanks to the unprecedented popularity of Docker, a container management technology. LXD was introduced right after the appearance of Docker, as a lightweight hypervisor for Linux containers. In this paper, we present the results of an experimental study that analyzes the performance of a LXD container when compared to that of a Docker container and a Virtual Machine in a VMware ESX environment. Our results show that Docker containers perform better than LXD in almost all cases with a plain virtual machine performance as a baseline. We further discuss how LXD complements the Docker technology as a container management suite.

Index Terms—Benchmark, Containerization, Docker, LXD, Performance, Virtual Machine, Virtualization

1 Introduction

The advent of virtual machines heralded the world of infrastructure in cloud. Now, the resources could be better utilized and centrally controlled with the support for isolated workloads (operating systems). Next, the power of Linux Containers [1] was introduced for the application level isolation on an OS. Containers enabled sharing of the kernel resources with the host as opposed to the one to one mapping between a virtual machine and its kernel. Even though, containers are restricted to a single kernel, they result in a much smaller footprint compared to virtual machines as the sharing of the kernel reduces the overhead of virtualizing the hardware, as in the case of a virtual machine, and better utilizes the kernel resources.

Linux Containers utilize the power of many Linux kernel level technologies for an operating system level virtualization – concepts like kernel namespaces, chroot, control groups, Linux Security Modules, network bridges etc. Docker made the notion of application-level container virtualization a mainstream technology. It utilized the APIs of the low level Linux containers to provide a simple, clear and intuitive command line experience with the additional features of application packaging and distribution.

Soon after, LXD [2] was introduced as a lightweight full OS system container solution on top of Linux Containers. While Docker is designed to host a single application, LXD is comparable to a virtual machine with the ability to host multiple OS containers on a single host. LXD also uses Linux Container APIs via liblxc and a set of Go bindings. It provides an alternative to Linux Container's tools and template system, not unlike Docker, but with the additional features of management layer over network (REST API exposure), container memory snapshot, checkpoint and restore with live container migration and security by design. LXD offers the highest density of guest OS containers per host among any other container solution. LXD containers are designed to run clean Linux distributions or appliances.

LXD provides a way to build on top of its network APIs to fully automate the process of multiple container deployment and management. A popular example of this is the development of the OpenStack Nova plugin which allows the containers to be managed just like a normal virtual machine. Thus, Docker containers can be easily nested inside the LXD operating system containers to benefit from this maintainability and security of LXD.

This paper looks at the two kinds of recent containerization technologies introduced so far, LXD and Docker and examines the performances of a set of stress tests based on a number of benchmarks for various aspects such as computation power, memory bandwidth, memory latency, I/O bandwidth and memory snapshot in both types of containers residing on a VMware ESX host, against a plain Ubuntu virtual machine acting as a normalizing factor.

With this experiment, we want to understand the performance overhead that results due to the abstraction layers introduced by Docker and LXD when compared to a plain virtual machine. We expect these technologies to provide a similar performance experience in other hypervisors like KVM [3], Microsoft Hyper-V [4], and Xen [5] due to a similarity in the hardware acceleration features across this list. Since, Ubuntu is used as the guest and host OS in the experiment, this analysis should provide results which leads to a correct comparison.

We make the following contributions:

- We provide the latest comparison of LXD, Docker and virtual machine environments using the industry standard hardware and software for interesting benchmarks and workloads.
- We show that LXD is a comaparable to a virtual machine in many use cases.

2 Environment

We used 2.80 GHz Intel Xeon CPU E5-2680 processors for a total of 8 cores, 16 GB of RAM and 20 GB of disk storage for all the test VMs. This is a basic server configuration that is easily available in many Infrastructure as a Service (IaaS) cloud providers. We used Ubuntu 16.04.1 (XenialXerus) 64-bit with Linux kernel 4.4.0-38-generic, Docker 1.12.1 and LXD 2.2. Both types of containers reside on the VMs created on VMware ESX 6.0U2. For consistency, all the base VMs and the LXD containers used Ubuntu Xenial. Docker and LXD containers did not have any restrictions configured against using the full resources of the system under test. We used benchmark tests to

individually measure CPU, memory, and storage overhead. The performance evaluation of running a Docker inside a LXD container is out of scope for this research and is a part of future works.

3 BENCHMARKS

As part of our experiment, we used a number of standard micro benchmarks and workloads mentioned below. Each test is executed multiple times and the values reported are averaged out.

Micro benchmarks:

1. Stream [6]: measures memory bandwidth by exercising CPU cores at their maximum limit.
2. Dbench [7]: generates I/O workloads used to stress a filesystem to see at which workloads it becomes saturated and can be used to analyze how many concurrent requests can the server handle before the response starts to lag.
3. RAMspeed SMP [8]: is used to test how fast are both cache and memory subsystems via allocating certain memory space and start either writing to or reading from it using continuous blocks.
4. Parallel BZIP2 Compression [9]: is a parallel implementation of the bzip2 block-sorting file compressor that uses pthreads and achieves near-linear speedup on SMP machines.
5. Gzip Compression [10]: measures the performance of I/O for the compression of a 2 GB file.
6. John the Ripper (BlowFish) [11]: is a password cracking tool which takes text string samples, encrypts it with BlowFish and compares the output to the encrypted string.

4 EVALUATION

4.1 Computation speed performance

For the evaluation of the CPU performance among LXD, Docker and Virtual Machine, we use Stream CPU, John the Ripper (BlowFish) and Parallel BZIP2 Compression as benchmark tests.

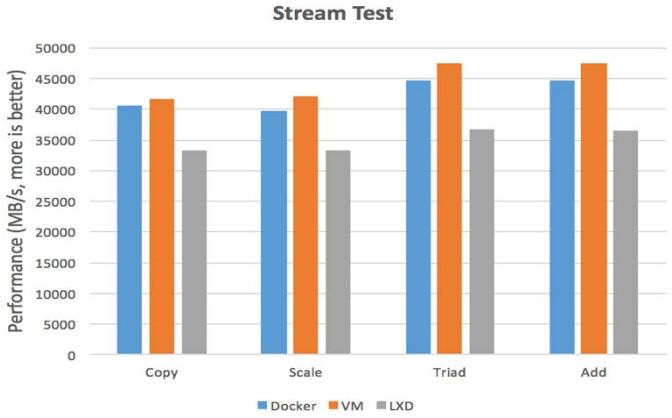

Fig. 1. Stream CPU performance for copy, scale, triad and add operations

Stream CPU counts how many bytes get moved from one placein memory to another. The Table 1 shows the count of bytes and FLOPs in each iteration of the STREAM loop.

TABLE 1:STREAM OPERATIONS

Name	Kernel	Bytes/iter	FLOPS/iter
COPY	a(i) = b(i)	16	0
SCALE	a(i) = q*b(i)	16	1
SUM	a(i) = b(i) + c(i)	24	1
TRIAD	a(i) = b(i) + q*c(i)	24	2

From among the multiple repetitions of these four operations, the top 10 results are chosen. As can be observed from the Fig. 1, though the docker container is slightly lower in performance than the virtual machine, the LXD container's performance is way lower than the other two. Keep in mind that the Stream tests are single processor benchmarks.

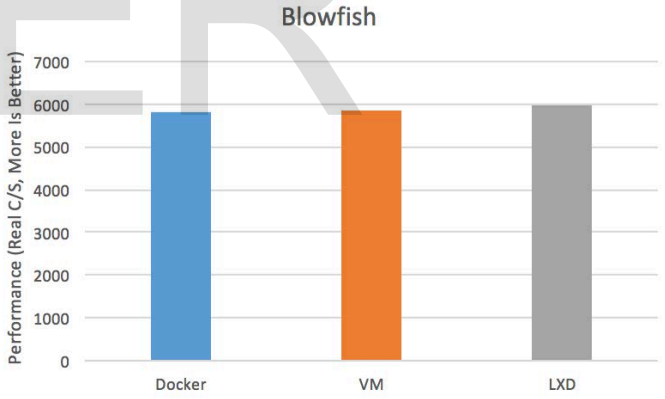

Fig. 2. Speed of execution of John the Ripper (BlowFish)

John the Ripper (BlowFish) has two types of workloads which include generating hashes of the candidate passwords and the comparison of the computed hashes against the encrypted strings. It is a multiprocessor test. According to the Fig. 2, LXD containers are performing a bit better than the virtual machine and the docker container. The performance speed of virtual machine and the docker are comparable to each other.

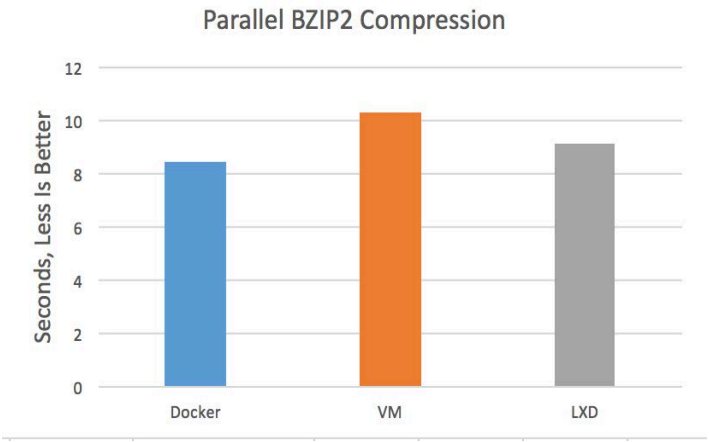

Fig. 3. Time taken by Parallel BZIP2 Compression

Parallel BZIP2 Compression measures the time to compress a .tar file and is a parallel multiprocessor test. On comparing the results in Fig. 3, we find that even though the virtual machine has performed slightly better than LXD container, the latter is still performing better than docker container.
As can be inferred from the results of the three benchmarks, LXD container performs relatively better when a parallel multiprocessor application is executed in its environment as compared to the uniprocessor workload.

4.2 Memory performance

Memory performance is measured by the executing the RAMspeed SMP benchmark tests on the three environments.

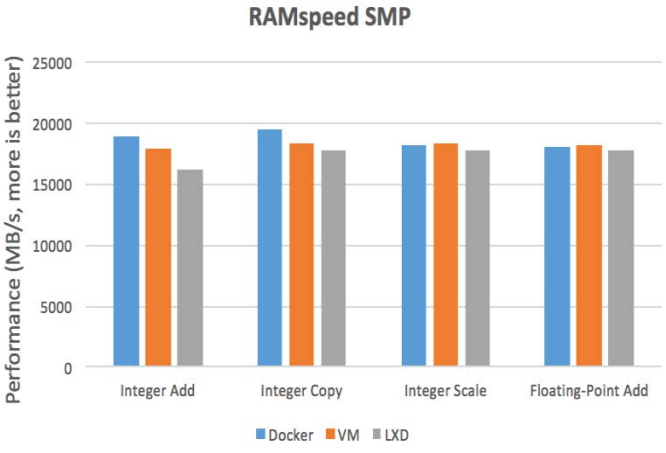

Fig. 4.Speed of completion of the four operations of RAMspeed SMP

RAMspeed SMP workloads consists of integer and float operations such as copying data from one memory location to another, adding data from two memory locations, multiplying the memory value and updating the location with the result. As depicted in the Fig. 4, the three virtualization environments do not differ from each other significantly. LXD container is lower in performance than the other two, but with less than 6% of performance difference.
So we can conclude that the memory overhead difference between virtual machine, docker container and LXD container is not of any big significance.

4.3 I/O storage performance

In order to evaluate the I/O bandwidth performance, we used Dbench, IOzone and AIO-Stress benchmark tests in our test environments.

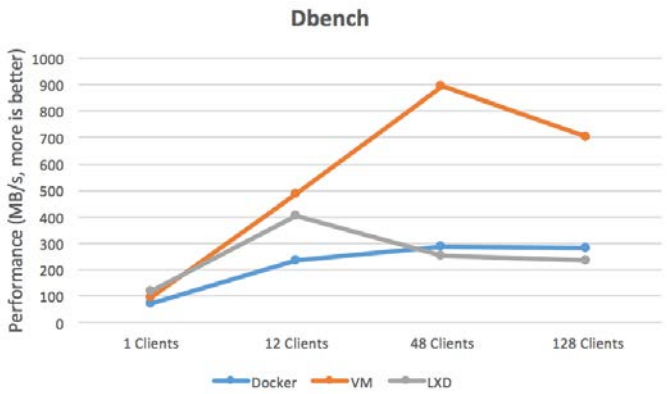

Fig. 5. Performance results for I/O for varying number of parallel client processes under Dbench benchmark

Dbench tool generates I/O workload by making calls to the local filesystem. Using this tool, predictions can be made about the maximum number of client applications that the local file system can handle, before the performance of I/O operations starts to degrade as the stress level increases. As can be seen from Fig. 5, the I/O operations in LXD, with ZFS as the default filesystem, takes a hit when the number of concurrent clients are more than 12. Whereas, VM can handle up to 128 clients simultaneously. I/O performance in docker, on the other hand, does not fluctuate much and remains fairly constant even after increasing the number of parallel clients from 48 to 128.

GzipCompression benchmark tests the I/O bandwidth performance for a real world example for the compression of a 2 GB file. Fig. 6 shows docker container to be slightly ahead in performance speed, followed by virtual machine and LXD container. Though there is a slight difference in performance, it is inconsequential in nature.

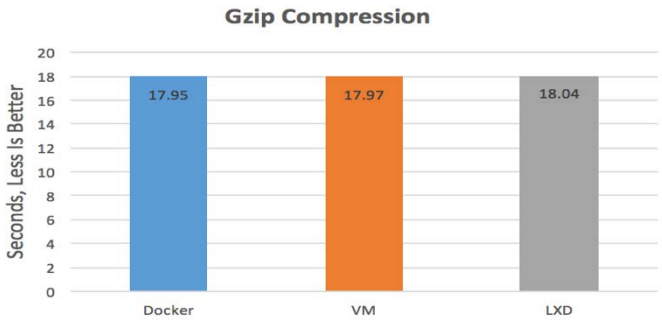

Fig. 6.Disk performance during compression test

LXD container performs equal to its peers in a single client environment, but starts losing out as the number of parallel clients increases. As the parallel reads and writes increase, the overhead latency in the layered design of LXD grows more significant. Thus, LXD container is better equipped for non-parallel I/O applications.

4.4 Stateful snapshot duration

Here, we compare the amount of time taken in capturing the snapshots (filesystem + memory) of both LXD container and virtual machine. Docker does not support the concept of memory snapshots, though it is under active development as a CRIU library, therefore, this section excludes docker from the comparison.

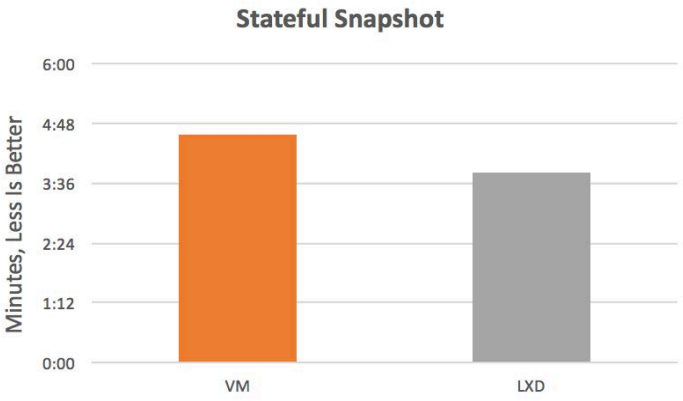

Fig. 7.Time taken to capture a stateful snapshot

When taking a snapshot of a virtual machine, the process of snapshot has to capture the kernel files also. This is not required in the case of LXD as it only requires the filesystem and the memory state of the container.As we can see from the Fig. 7, the process of taking a LXD snapshot is faster than that of a virtual machine.

5 Conclusion

Docker is tightly coupled with the native OS on which it runs. With the arrival of LXD, one is no longer restricted to a single flavor of OS container. With this paper, we have provided a validation to the performance doubts surrounding LXD. As we have seen in our experiment, Docker and LXD have minimalistic overhead, especially in the case of CPU and memory performance. One will still need to carefully evaluate I/O-intensive applications against the containers and the performance of these containers under mixed workloads.

LXD is comparable in performance to a virtual machine and supports the running of clean Linux flavors.It also has maintainability and remote automation in its arsenal. Thus, it can become an ideal host for application containerization technologies like Docker, which themselves are performant.

This concept of application virtualization on a lightweight OS virtualization in itself is not a new notion. VMware has already announced its vSphere Integrated Container strategy complemented with the release of its own stripped down version of Linux OS, known as PhotonOS, which supports docker like application containerization solutions and provides support for Administrator level manageability. But LXD provides the familiarity of Ubuntu with all the other features.

There are still a lot of stability issues with LXD like automatic network configuration on live migration of the container, disk and multiprocessor performance, which needs to get addressed. We will have to wait and see what LXD comes up with to survive in this rapidly evolving world of virtualization.

References

[1] LXC, https://linuxcontainers.org, last accessed 28/Sept/2016
[2] Stéphane Graber, Getting started with LXD, https://insights.ubuntu.com, last accessed 27/Sept/2016
[3] KVM, http://www.linux-kvm.org/page/Main_Page, last accessed 19/Sept/2016
[4] Microsoft Hyper-V, https://en.wikipedia.org/wiki/Hyper-V, last accessed 19/Sept/2016
[5] XEN, https://en.wikipedia.org/wiki/Xen, last accessed 19/Sept/2016
[6] Stream, http://www.cs.virginia.edu/stream/ref.html, last accessed 30/Sept/2016
[7] Dbench, https://dbench.samba.org/, last accessed 22/Sept/2016
[8] RAMspeed SMP, http://alasir.com/software/ramspeed/, last accessed 29/Sept/2016
[9] Parallel BZIP2 Compression, http://compression.ca/pbzip2/, last accessed 30/Sept/2016
[10] Gzipcompression benchmark, https://openbenchmarking.org/test/pts/compress-gzip/, last accessed 27/Sept/2016
[11] John the Ripper, http://www.openwall.com/john, last accessed 26/Sept/2016