



Hewlett Packard
Enterprise

Linux container performance on HPE ProLiant servers

Understanding performance differences between
containers and virtual machines

Introduction.....	3
Overview of containers and virtual machines	3
Overview of testing objectives.....	4
Complex business applications	4
Java-based containers and VMs.....	4
Performance	6
Processor usage.....	7
Memory usage	8
Summary.....	8
Distributed environment.....	9
Summary.....	10
Web-based applications.....	10
Hard reboot.....	10
Soft Reboot.....	13
Summary.....	15
Pure performance.....	16
Storage I/O performance (Sysbench mysql).....	16
Floating point computing performance (LINPACK).....	18
Memory performance (Stream).....	19
Storage I/O performance (FIO tester)	19
Network performance.....	20
Multiuser application environment	23
Conclusion.....	24
Key takeaways.....	25
Resources, contacts, or additional links.....	26

Introduction

When planning your strategy for implementing a solution stack, isolating applications and processes can be a way to create an environment that supports your applications with the best performance.

In a Linux environment, applications and processes can be isolated using either containers or virtual machines (VMs). Containers can yield the best performance; however, they might not be ideal for every solution stack. There are solution stacks where VMs provide advantages and are a better fit than containers. For example, some solution stacks include multi-tenant applications and do not need isolation of any kind—either container or VM; or, a solution stack may have applications that are operating system (OS) dependent and *require* VMs instead of containers.

It is not the intention of this paper to endorse either containers or VMs. Your solution stack requirements determine the need to separate and isolate an application natively, in a container, or in a VM. This paper provides general information about container technologies from a performance perspective and concludes with some general advice for implementing containers.

For a detailed overview of container technology, its history, how it works, and its benefits and shortcomings, see the [Linux container technology](#) technical white paper.

Overview of containers and virtual machines

Figure 1 illustrates the basic structure of VMs and containers.

As illustrated in the figure, VMs create a virtualization environment that uses the Linux kernel as a hypervisor. The Linux kernel manages the processor and memory resources by emulating a machine interface for each VM guest OS kernel, initialization system, library, and application(s).

Instead of using a hypervisor to manage VMs, the figure shows how containers isolate applications into separate environments (containers) that include processor, memory, and networking resources as part of the container itself. This environment provides OS-level virtualization. Containers have their own root; and, users and processes do not perform operations outside of the container environment. The host OS kernel manages container workloads directly, which reduces the overhead involved with managing system resources. This improves efficiency and therefore, improves performance.

For more details about container technology, its history, how it works, and its benefits and shortcomings, see the [Linux container technology](#) technical white paper.

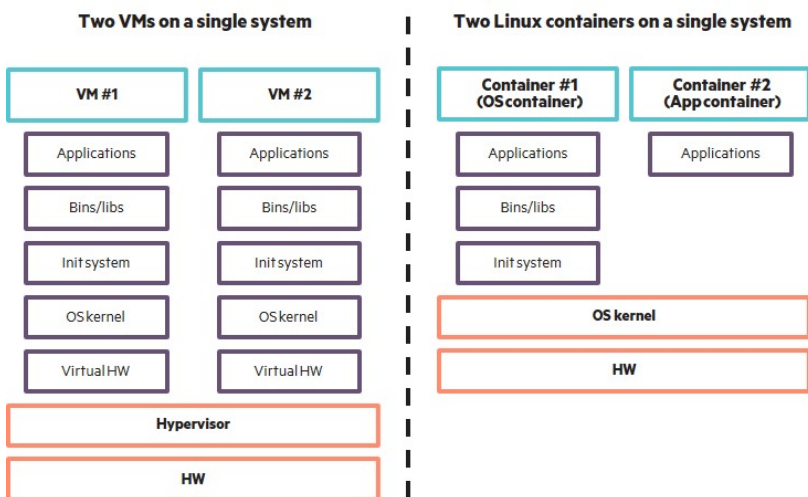


Figure 1. VM and container comparison

Overview of testing objectives

Our testing objectives were to understand the scalability of complex business applications and to understand the pure subsystem performance for containers and VMs. We performed benchmark testing that encompassed the following configurations. For each configuration, we ran different workloads to gather the comparative information on resource usage for processor, memory, networking, and storage.

- A bare-metal HPE ProLiant server configured with Red Hat Enterprise Linux (RHEL)
- An HPE ProLiant server configured with RHEL, RHEL Atomic Host, or Red Hat Enterprise Virtualization (RHEV) and applications isolated in Docker containers. Docker is the platform that we used for creating and managing containers.
- An HPE ProLiant server configured with RHEL, RHEL Atomic Host, or RHEV and applications isolated in kernel-based virtual machines (KVMs). We used KVMs for creating the virtualization infrastructure.

Complex business applications

To gather information about the relative scalability and performance for complex business applications, we focused on the following environments:

- Java-based containers and VMs
- Distributed environments
- Web-based applications

To understand processor usage and memory usage within these environments, we simulated Java-based applications running in a database online transaction processing (OLTP) workload. We focused on Java-based applications because they are prevalent in these environments.

To understand capacity and performance within these environments, we looked at the following capacity measurements in transitional workloads:

- Performance and deployment speed in a virtualization workload test
- Cloud deployment and performance using an OpenStack-based workload

Java-based containers and VMs

For our benchmarks that simulated a scalable, Java-based environment, we captured the following performance values. The number of operations were the same for all workloads.

- Pure throughput: The number of transactions completed during the test.
- Time-critical throughput: The number of required transactions that completed during the test.

The results (illustrated in Figure 2 and Figure 3) show that the workloads running in containers run 3% to 17% more Java operations per second (JOPS) than the workloads running in a comparable VM configuration, with the four-container configuration showing a noticeable advantage.

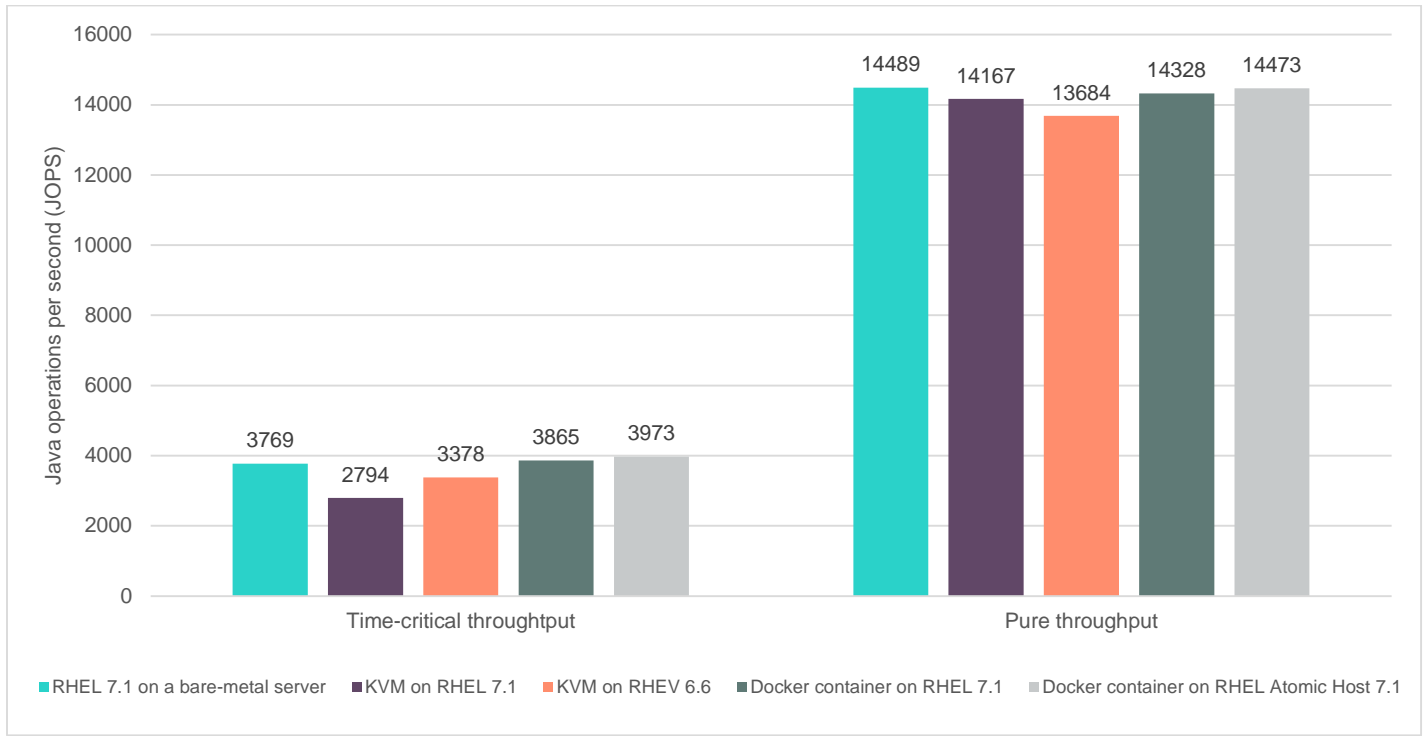


Figure 2. Comparison of two Docker containers and two KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

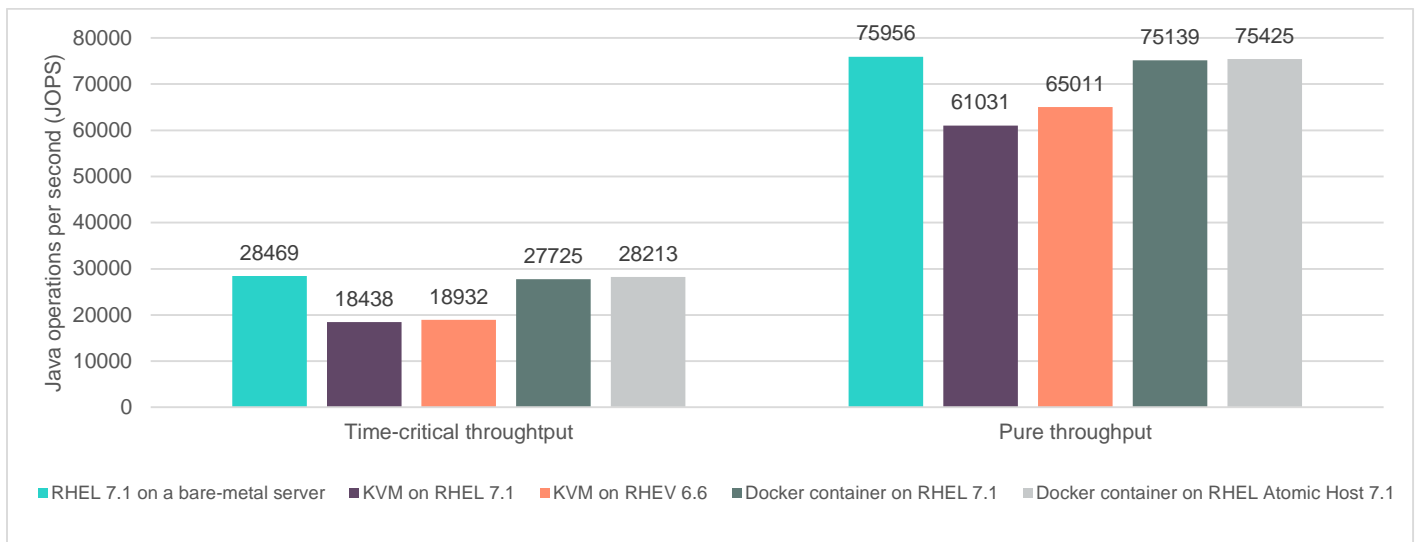


Figure 3. Comparison of four Docker containers and four KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

Performance

Our benchmark data for pure throughput and time-critical throughput showed that containers performed better than VMs. Figure 4 through Figure 9 illustrate the following comparison summary:

- Comparing one container to one VM:
 - There was a 9% performance improvement with the container.
 - There was a 6% reduction in processor use with the container.
 - There was a 3% increase in memory use with the container.
- Comparing four containers to four VMs:
 - There was a 12% performance improvement (throughput) with the containers (see Figure 4).
 - There was a 12% reduction in processor use with the containers (see Figure 6).
 - There was a 4% increase in memory use with the containers (see Figure 8).
- Comparing ten containers to ten VMs:
 - There was a 17% performance improvement (throughput) with the containers (see Figure 5).
 - There was a 17% reduction in processor use with the containers (see Figure 7).
 - There was a 6% increase in memory use with the containers (see Figure 9).

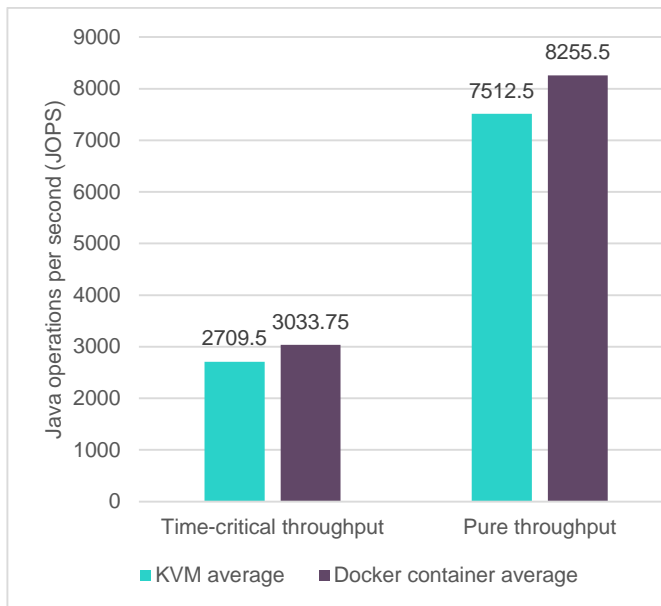


Figure 4. Comparison of four Docker containers and four KVMs [ProLiant DL380p Gen8 with RHEL 7.1]

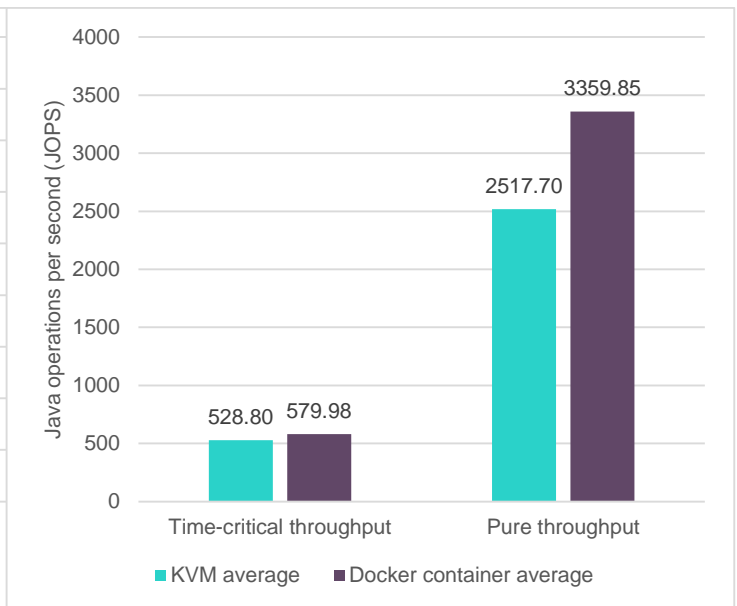


Figure 5. Comparison of ten Docker containers and ten KVMs [ProLiant DL380p Gen8 with RHEL 7.1]

Processor usage

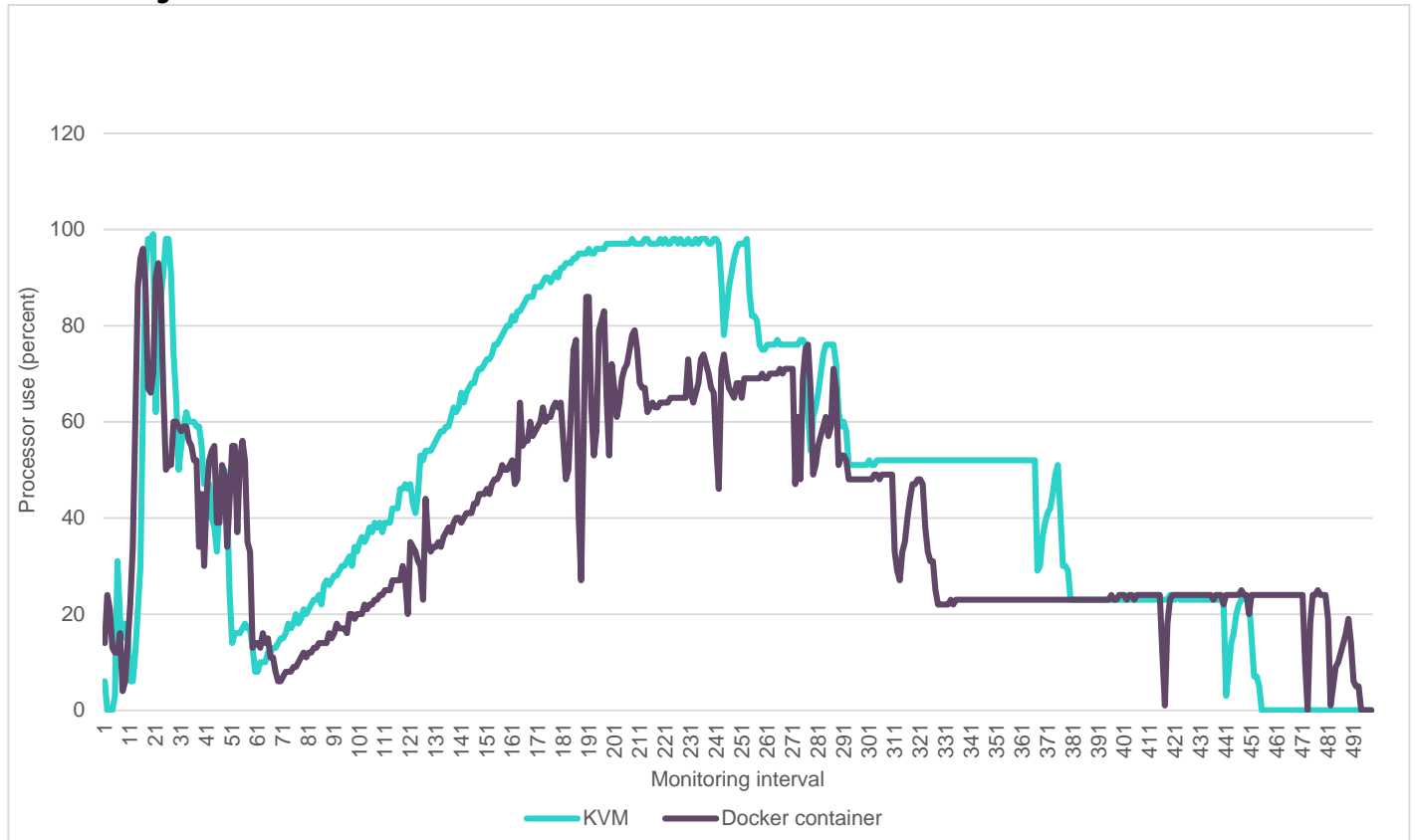


Figure 6. Comparison of four Docker containers and four KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

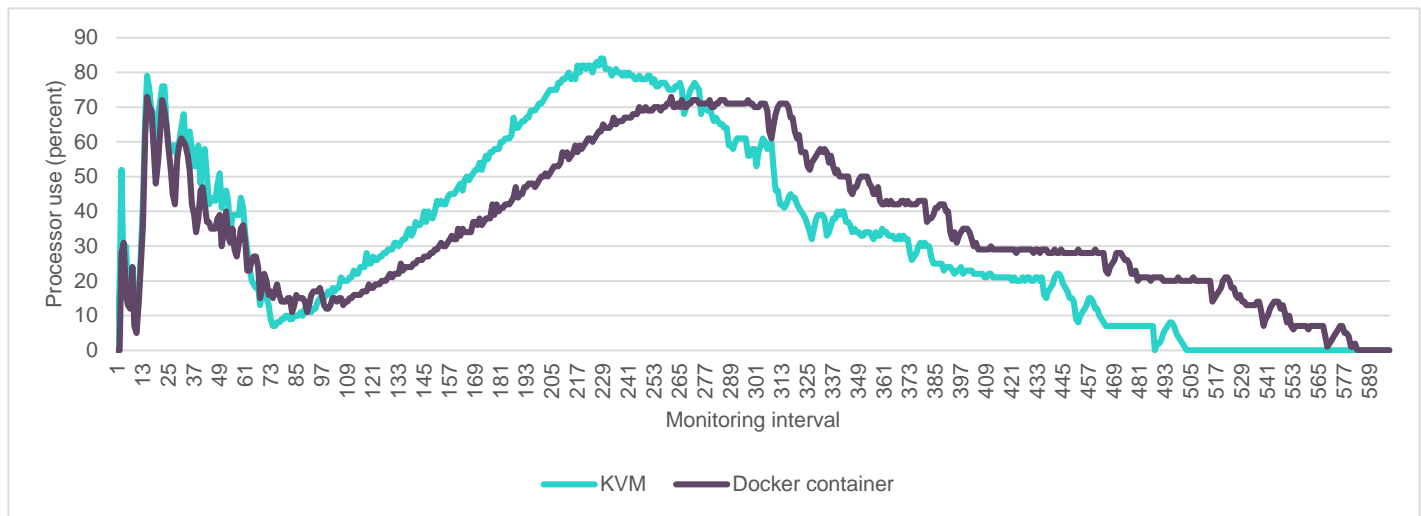


Figure 7. Comparison of ten Docker containers and ten KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

Memory usage

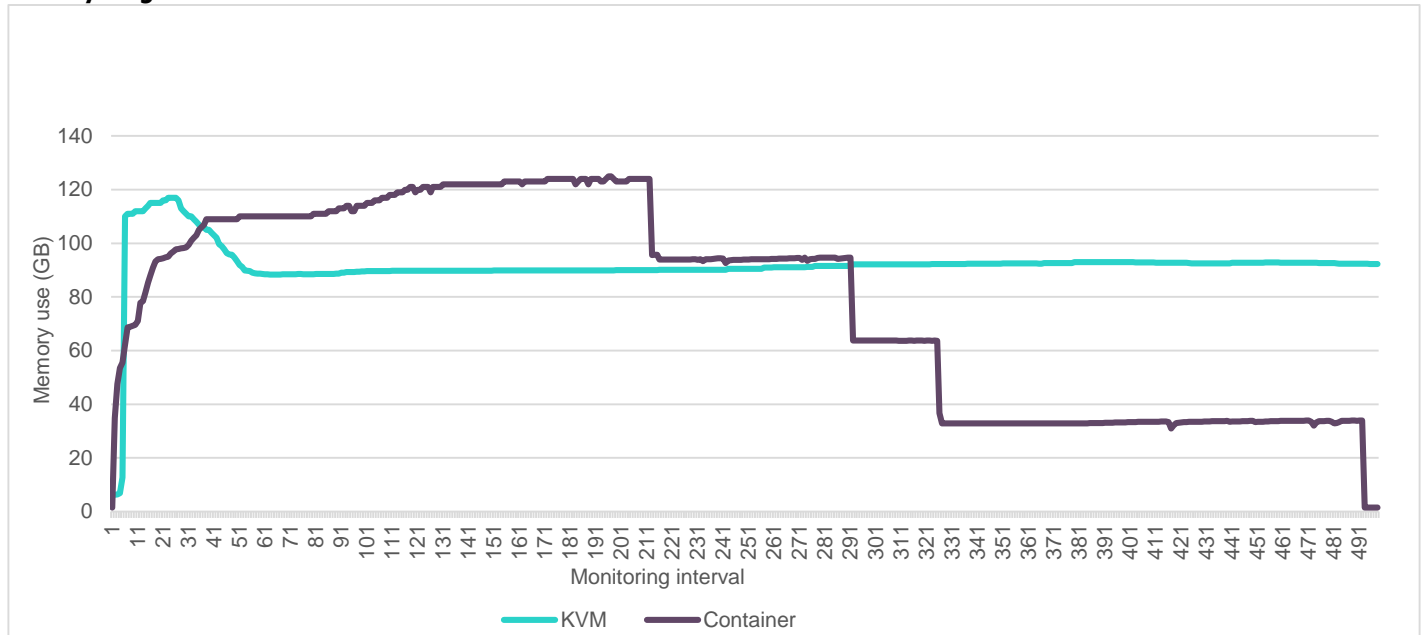


Figure 8. Comparison of four Docker containers and four KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

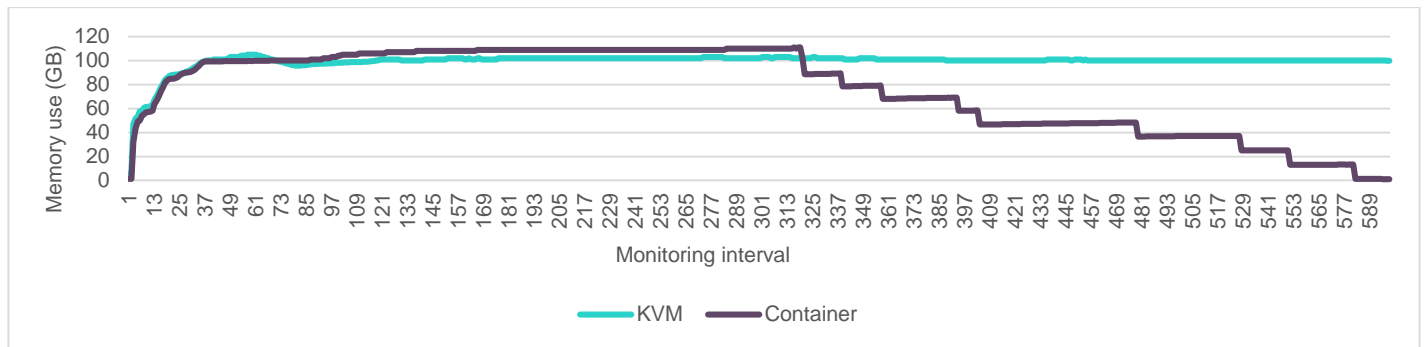


Figure 9. Comparison of ten Docker containers and ten KVMs (ProLiant DL380p Gen8 with RHEL 7.1)

Summary

Our testing shows that a Java-based business workload performs better on containers than on VMs.

VMs generally use more processor and memory because each VM has its own kernel. Each VM is overhead for the host because every kernel requires memory and processor resources.

Containers use control groups (cgroups) to allocate resources, including processor and memory, which allows a container to occupy more memory temporarily as needed. To get higher performance, a container might occupy around 3% more memory during a heavy workload; however, it releases that extra memory after the workload finishes. A VM uses the `/dev/kvm` module to isolate memory; therefore, a VM does not require more memory to handle different workloads but does require more memory overall because of the kernel.

Distributed environment

Our distributed environment evaluation simulates a scalable, distributed environment that includes extensive communication among containers, which is a common characteristic of microservices-based and business-to-business applications. For the test platform, we used a ProLiant DL580 Gen8 server running RHEL 7.1.

To show scaling, we consolidated multiple servers running different applications onto a single system with multiple VMs or containers until the overall throughput peaked. This scenario models the multi-tiered applications and database environments that require enterprise-class VMs based on real-world traffic demanded from a web server, including the following:

- Java application server
- Internet Message Access Protocol (IMAP) server
- Batch server

As the number of servers increases from one to three, performance (requests per second) is comparable between containers and VMs (see Figure 10). Although performance is similar, resource consumption declines with containers compared to VMs (see Figure 11). Resource consumption increases with VMs because each VM has a kernel, which requires its own processor and memory resources. Because the host kernel manages container resources directly, the container resource consumption declines as the number of servers increases.



Figure 10. Comparison of Docker containers and KVMs [ProLiant DL580 Gen8 with RHEL 7.1]

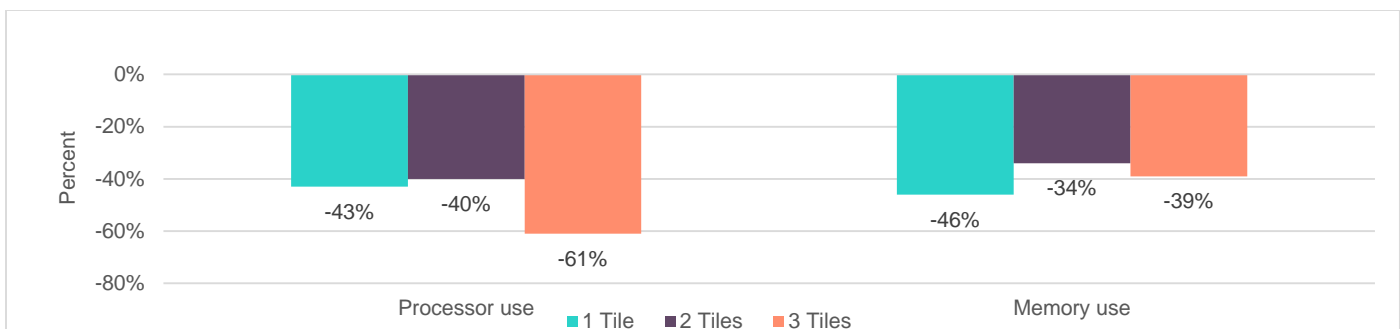


Figure 11. Comparison of Docker containers to KVMs [ProLiant DL580 Gen8 with RHEL 7.1]

Summary

Regardless of the workload, the resource consumption of containers is markedly less than VMs. Each VM has overhead because each VM's kernel requires processing resources and up to 1 GB of memory. The resources that the VM kernel requires is most apparent when a system is fully loaded or when there are many VMs. The OS kernel manages containers directly, so containers do not have the extra overhead.

Web-based applications

To evaluate how OpenStack works at scale, we simulated scalable, web-based applications running in an Openstack framework. We achieved this by automating and unifying multi-node OpenStack deployments, cloud verification, and profiling. We tested two scenarios that reflect impact of application service start up:

- Hard reboot, which power cycles the instance.
- Soft reboot, which performs a graceful shut down and restart of the instance.

For the test platform, we used a ProLiant DL580 Gen8 server running Ubuntu Server 14.04.

Hard reboot

This test showed that containers provide better performance to boot the server or to perform delete functions. When compared to a VM, the container performed as follows (see Figure 12):

- The average boot time for a container is 28.21% faster.
- The average delete time for a container is 34.35% faster.
- The average total time for a container is 30.57% faster.

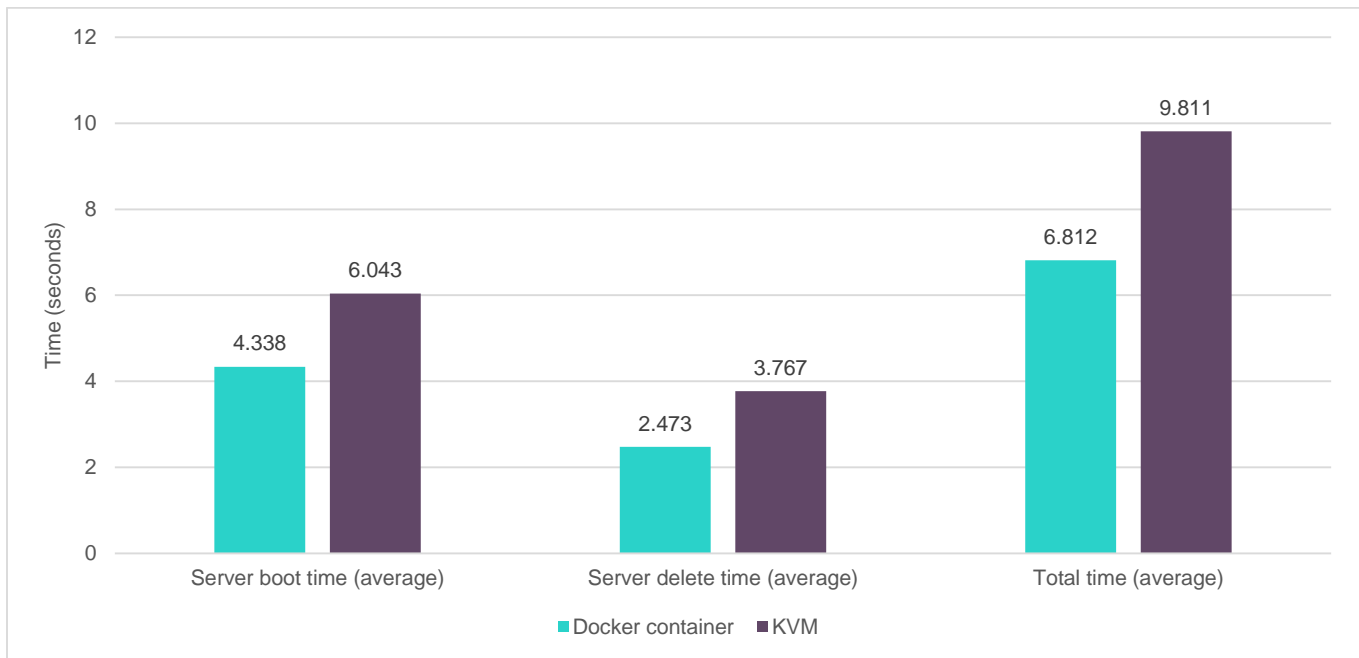


Figure 12. Comparison of Docker container and KVM [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

As illustrated in Figure 13 and Figure 14, containers run at a more stable, steady state, utilizing less peak processor bandwidth than VMs. When rebooting a container, the processor resource consumption is more stable because the container only needs to stop and restart a process. By comparison, a VM must go through the entire reboot process.

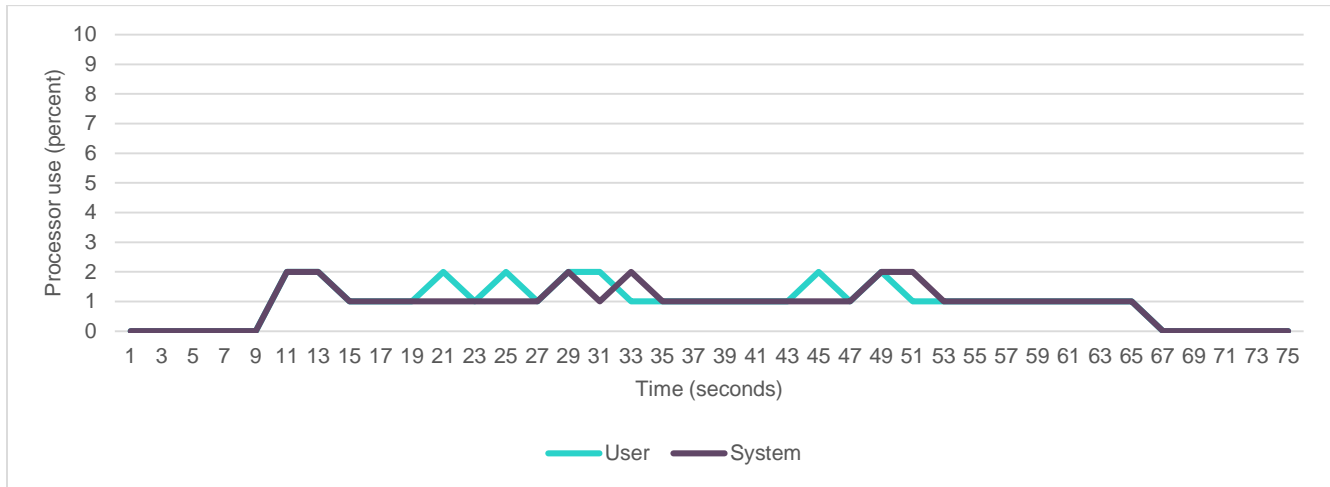


Figure 13. Docker container processor use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

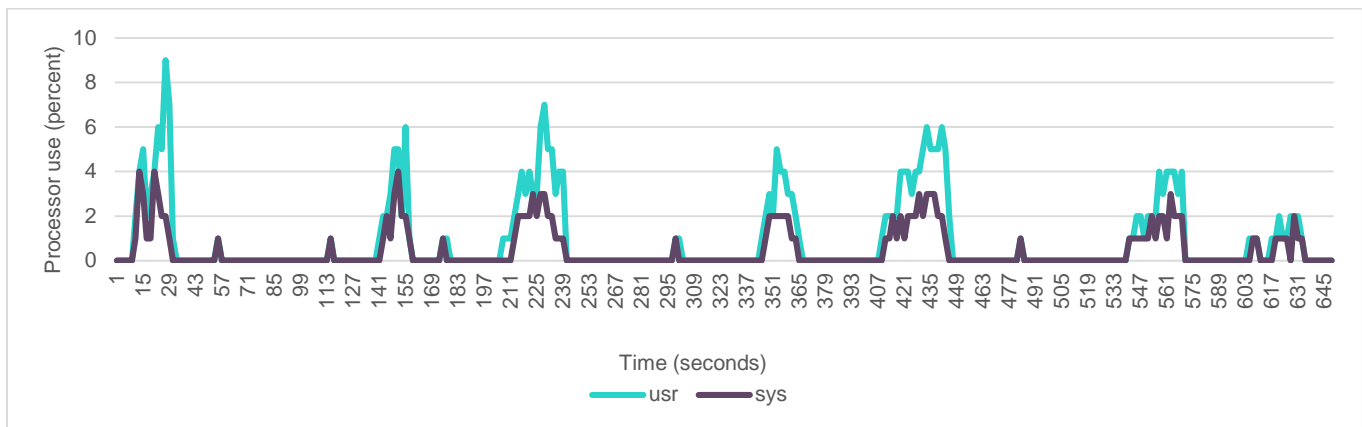


Figure 14. KVM processor use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

As illustrated in Figure 15 and Figure 16, the characteristics of containers and VMs result in better memory use with containers.

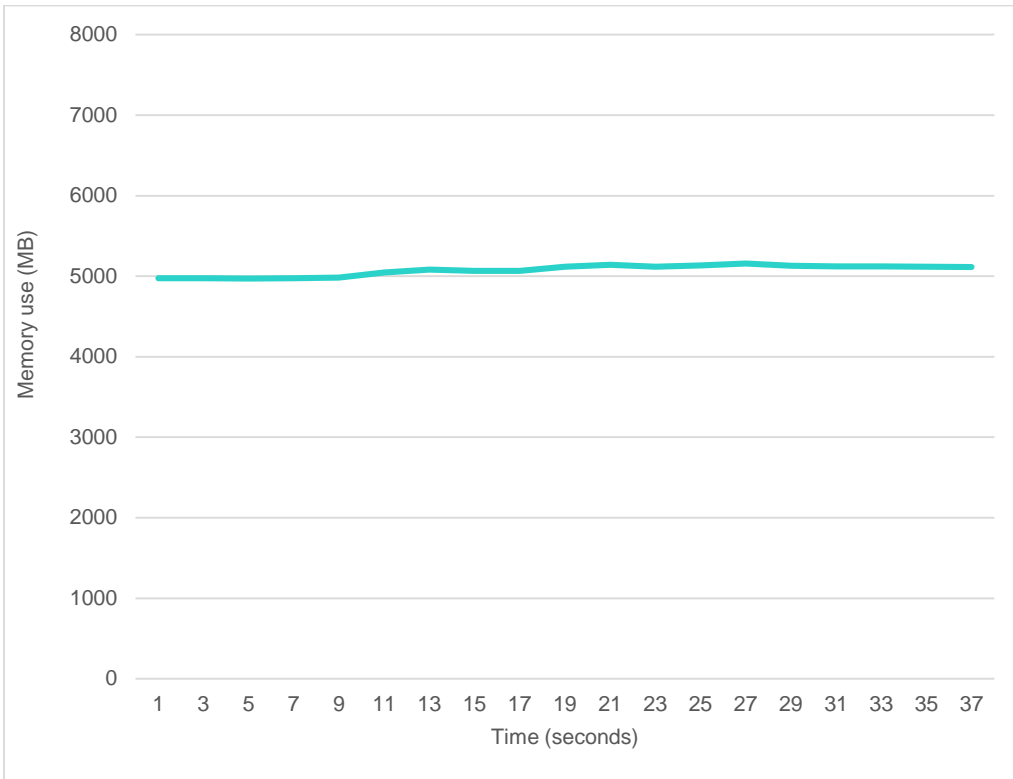


Figure 15. Docker container memory use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

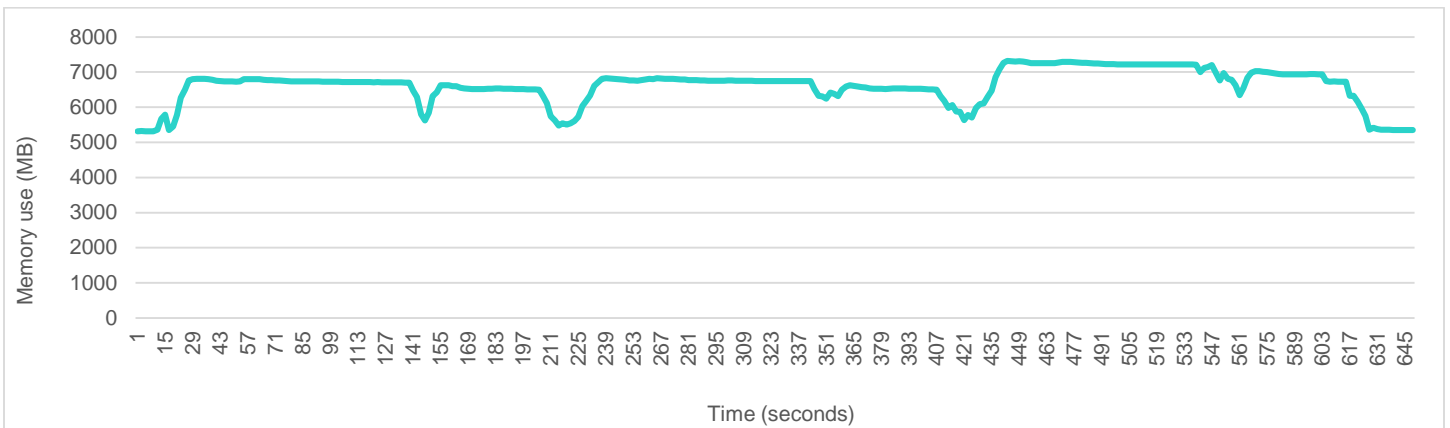


Figure 16. KVM memory use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

Soft Reboot

Our testing shows that containers provide better performance during a soft reboot, start, and stop. When compared to a VM, the container performed as follows (see Figure 17):

- The average soft reboot time for a container is 96.65% faster.
- The average start time for a container is 45.99% faster.
- The average stop time for a container is 96.13% faster.
- The average total time for a container is 90.76% faster.

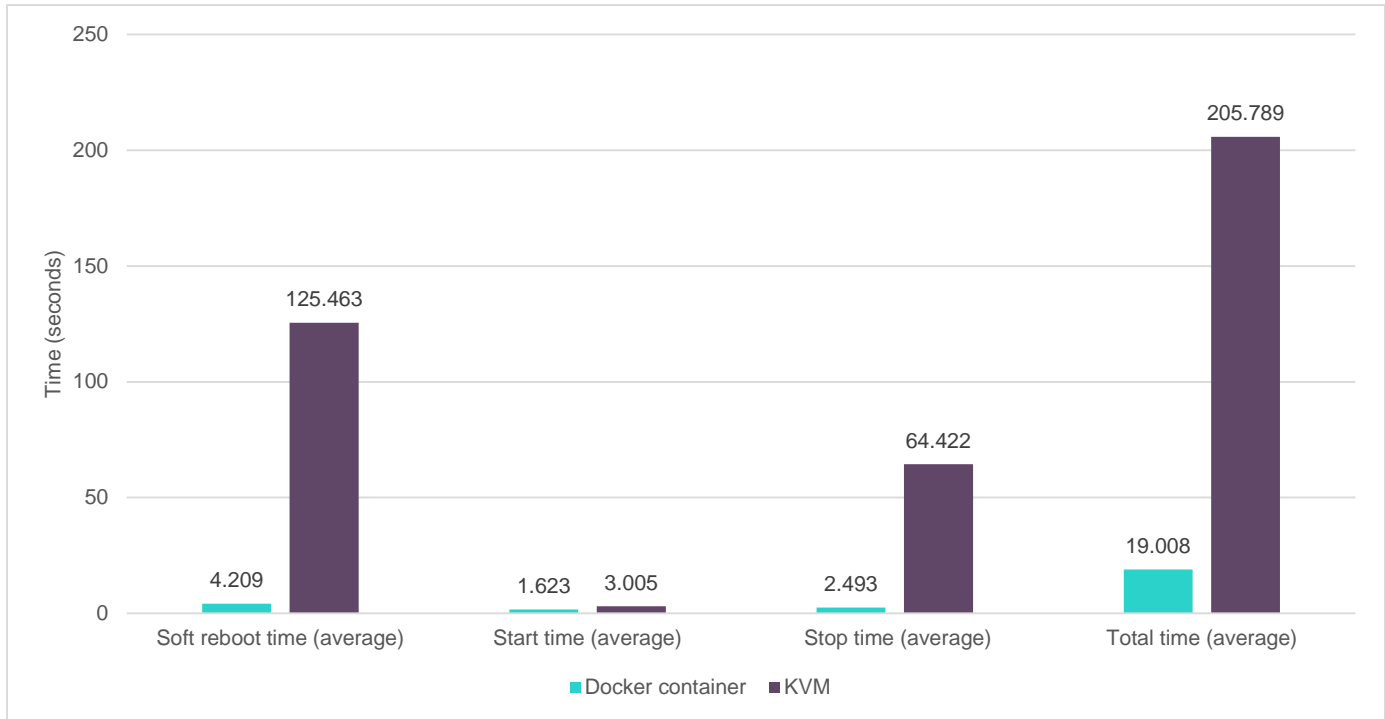


Figure 17. Container and VM soft reboot, start time, and stop time comparison [ProLiant DL580 Gen8 with Ubuntu Server 14.04].

As illustrated in Figure 18 and Figure 19, containers run at a more stable, steady state, utilizing less peak processor bandwidth than VMs. When rebooting a container, the processor resource consumption is more stable because the container only needs to stop and restart a process. By comparison, a VM must go through the entire reboot process.

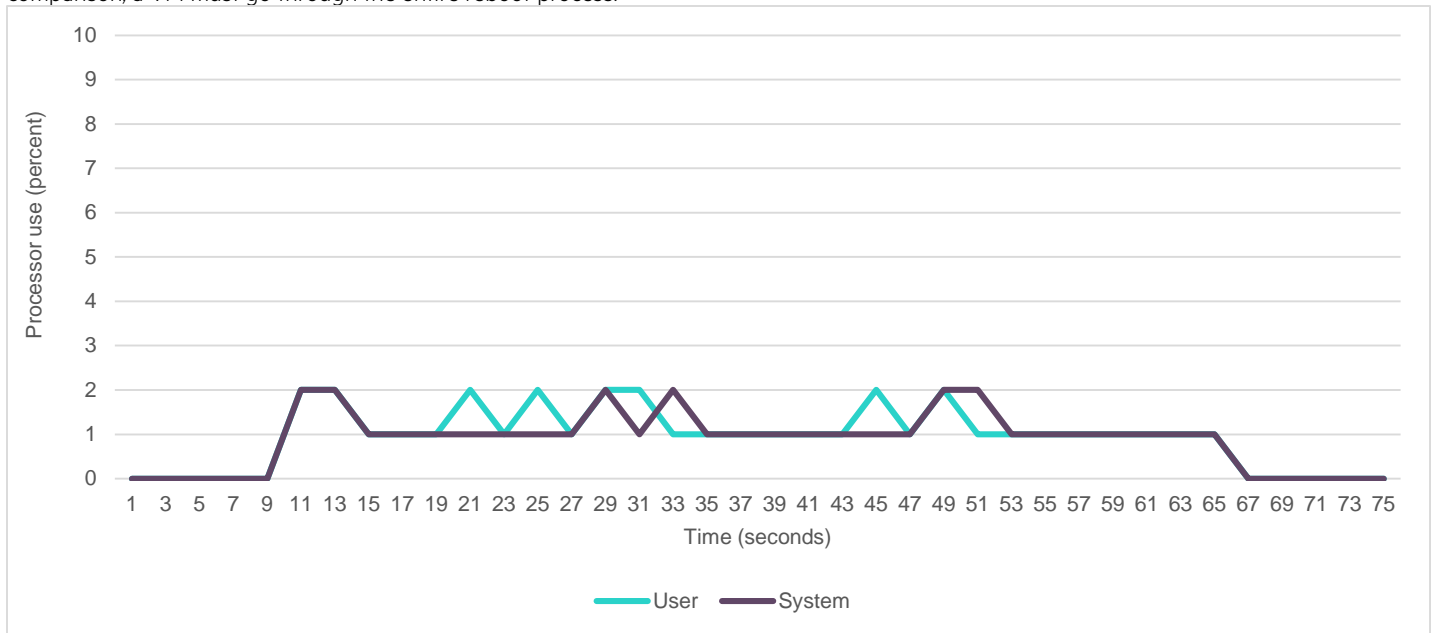


Figure 18. Docker container Compute Node processor use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

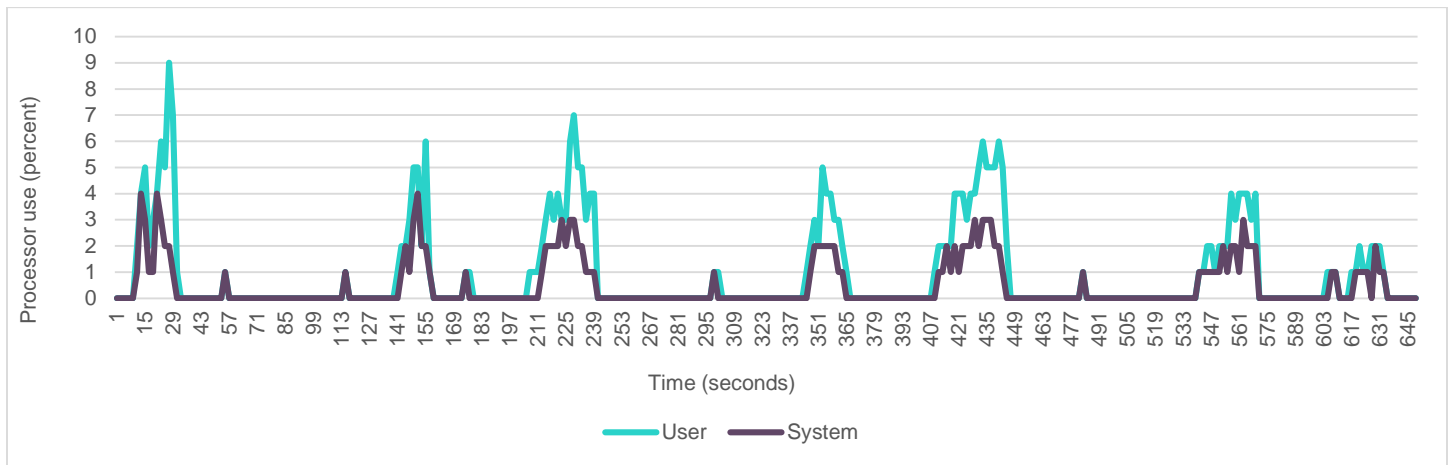


Figure 19. KVM Compute Node processor use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

As illustrated in Figure 20 and Figure 21, the characteristics of containers and VMs result in better memory use with containers.

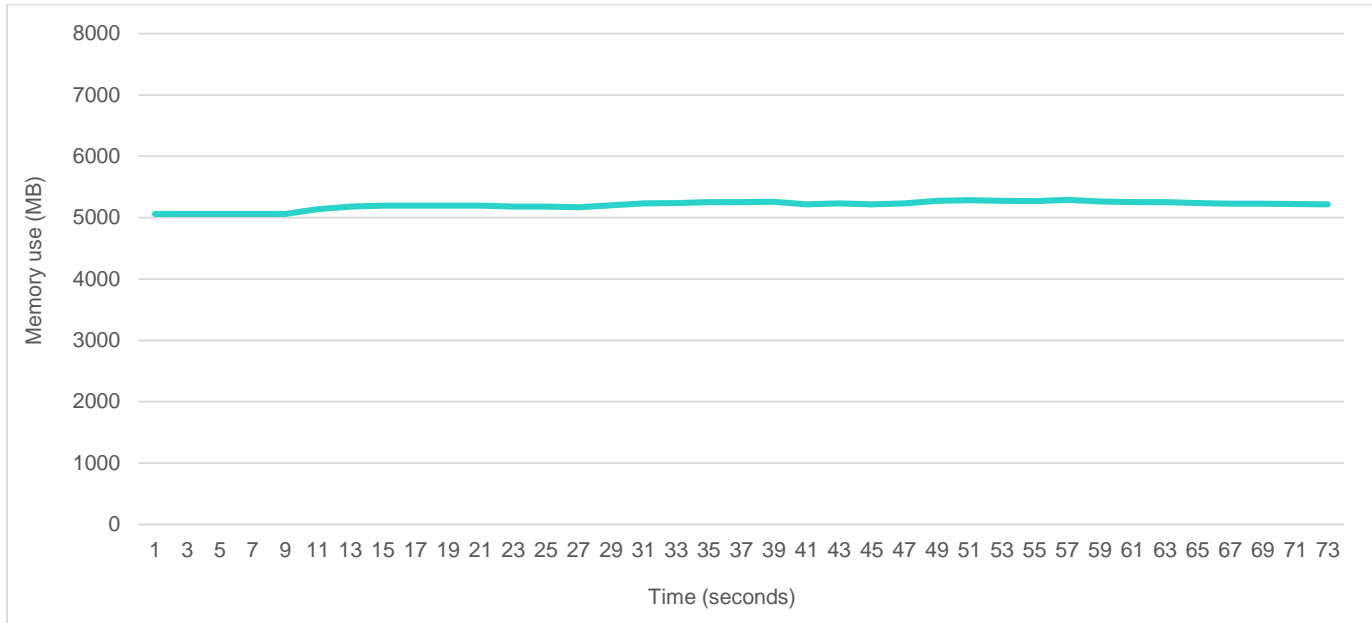


Figure 20. Docker container Compute Node memory use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

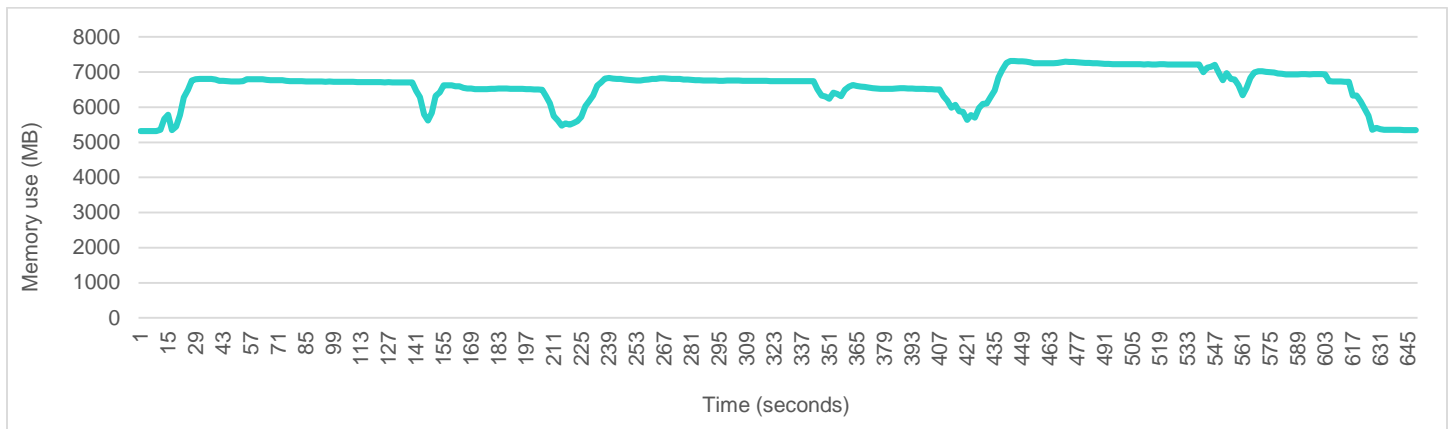


Figure 21. KVM Compute Node memory use [ProLiant DL580 Gen8 with Ubuntu Server 14.04]

Summary

Containers perform better and consume fewer resources when booting. The container boot process involves starting a process in a cgroup. Without initializing emulated hardware, as a VM must do, the container allocates resources by cgroup and namespace and then mounts the filesystem.

Booting a VM simulates a separate system, which requires more time and processor consumption. Additionally, because each VM has a kernel, more memory resources are required.

Pure performance

To measure performance in specific intensive situations, we tested intensive processor, memory, file system, and networking workloads comparing containers and VMs. To understand the pure performance characteristics of containers and VMs, we tested utilization of the following subsystems:

- Processor and memory performance (using the Sysbench CPU/memory benchmark)
- Floating point computing performance (using the Linear Algebra PACKage (LINPACK) benchmark)
- Memory performance (using the Stream (Sustainable Memory Bandwidth in High Performance Computers) benchmark)
- Storage I/O performance (using the Flexible I/O (FIO) tester and Sysbench mysql benchmarks)
- Network performance (using the Netperf benchmark)
- Multiuser application environment (using the AIM7 benchmark)

Storage I/O performance (Sysbench mysql)

Figure 22 shows results for storage I/O comparisons in an OLTP application environment running on MySQL, as follows:

- For read and write transactions per second for a container volume, which is a partition that is formatted into a file system is 37% to 86% higher than VM virtual I/O (VirtIO), which is a partition space of a disk.
- The maximum gap is 86% for 25 threads.

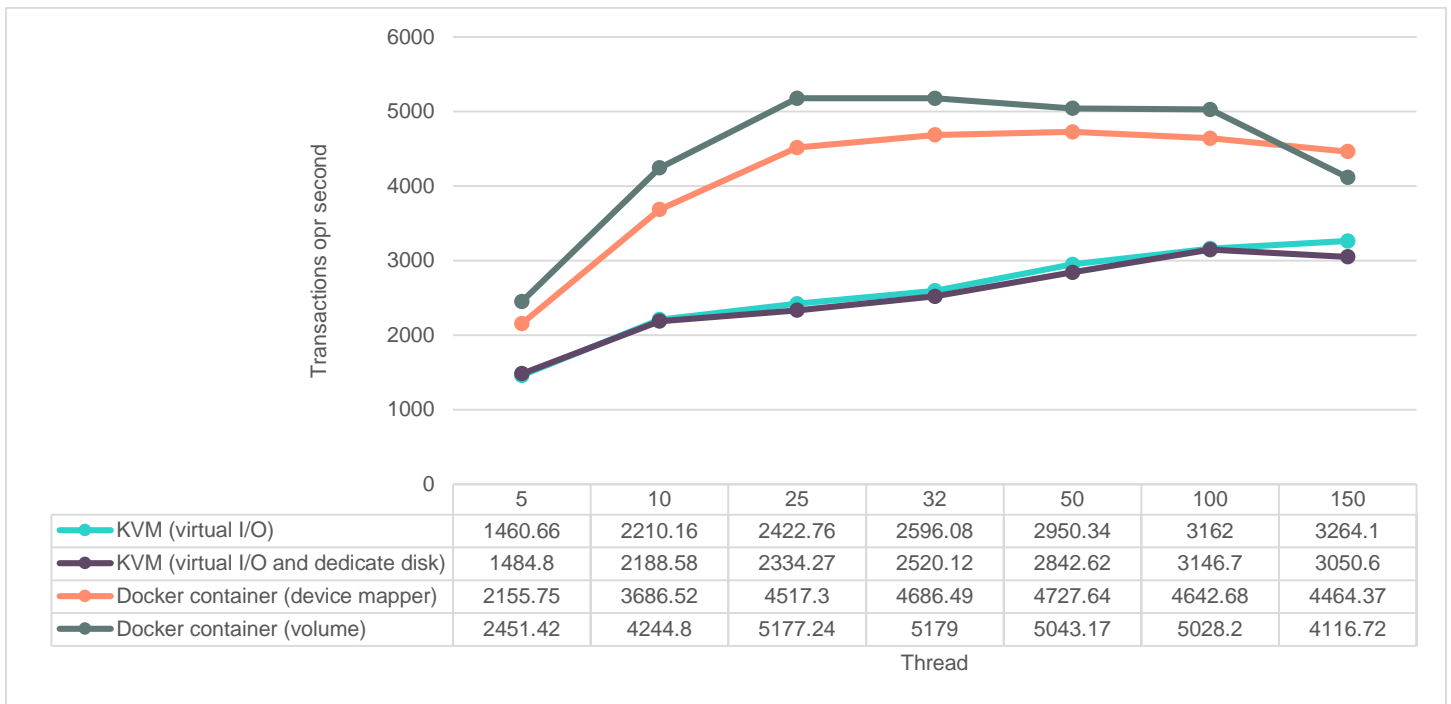


Figure 22. Results for OLTP application environment running on top of MySQL Server (ProLiant DL380p Gen8 Server with RHEL 7.1)

Processor and memory performance (Sysbench CPU/memory)

Figure 23 and Figure 24 show results for processor and memory performance comparisons in VMs/containers, which are as follows:

- For one container, the processor testing (total time) is 904.7625.
- For 30 containers the average time (total time) is 1089.704.

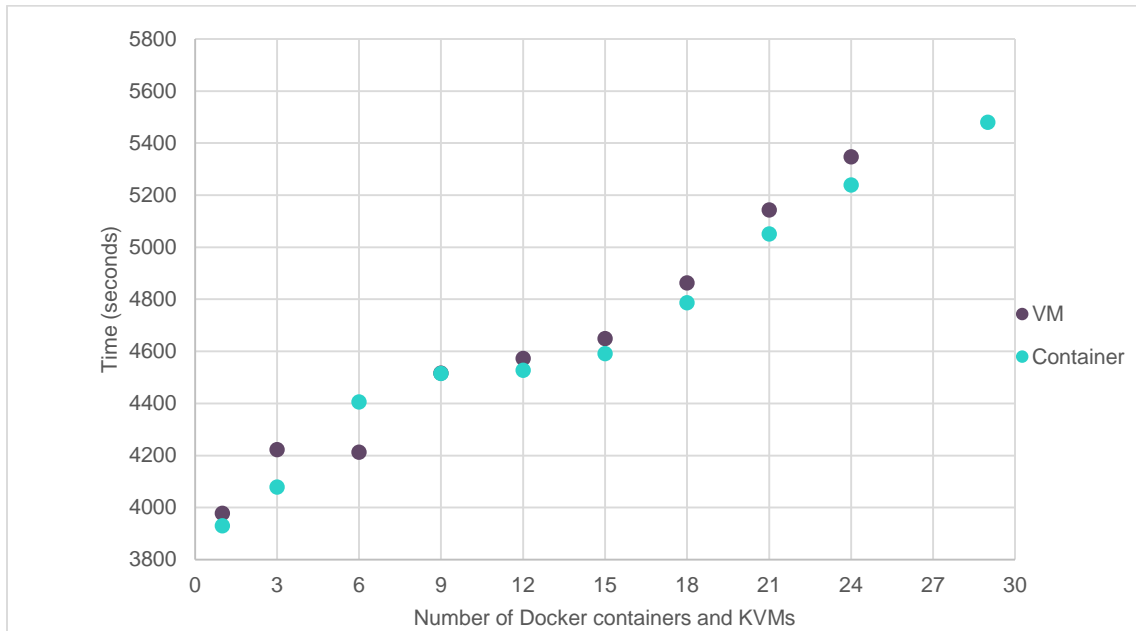


Figure 23. Processor test results (ProLiant BL460c Gen8 Server with RHEL 7.1)

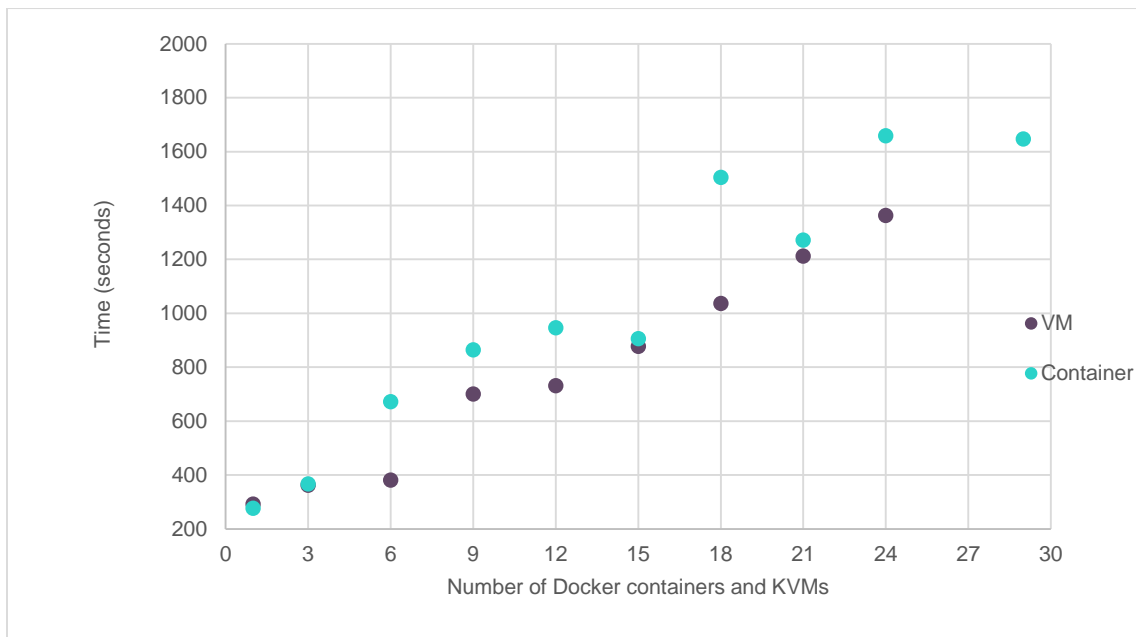


Figure 24. Memory test results (ProLiant BL460c Gen8 Server with RHEL 7.1)

Floating point computing performance (LINPACK)

To measure floating-point computing power, we evaluated a ProLiant DL580 Gen8 server running RHEL 7.1 and RHEL Atomic Host. Containers showed better performance (see Figure 25):

- With a floating point problem size of 79k, containers performed as follows:
 - Containers have the same performance with VMs on RHEL 7.1
 - Containers performance on RHEL Atomic Host improves by 7.8% compare to containers on RHEL7.1
- With a floating point problem size of 39k, containers performed as follows:
 - Containers perform 1.6% higher than VMs on RHEL 7.1
 - Containers performance on RHEL Atomic Host improves by 7% compare VMs on RHEV

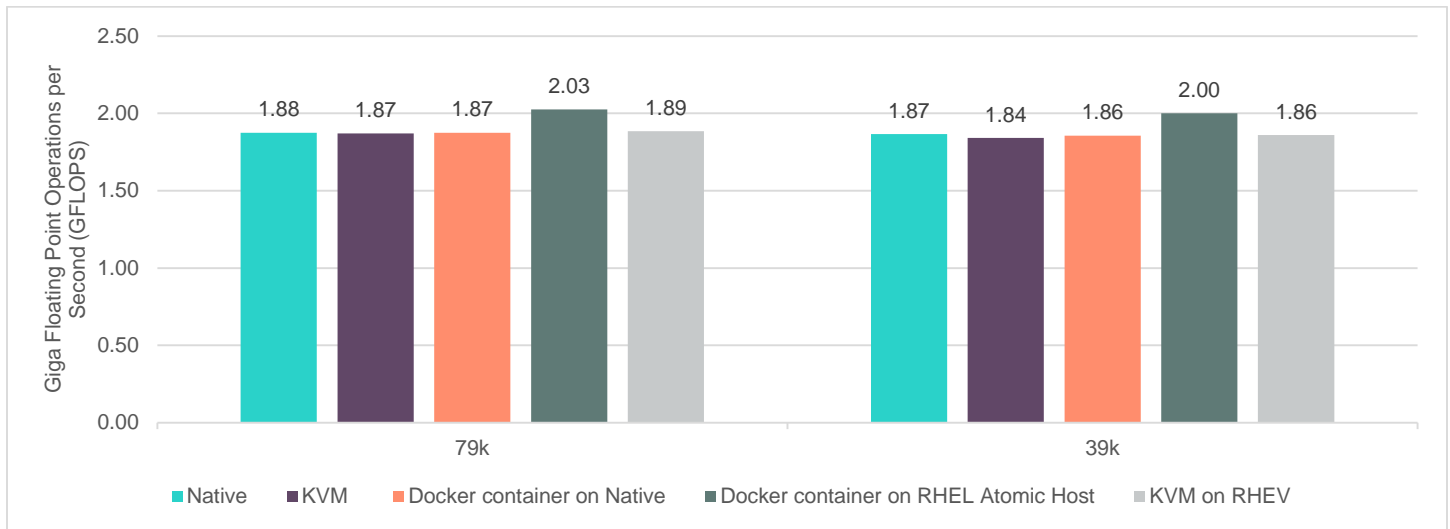


Figure 25. Results [ProLiant DL580 Gen8 with RHEL 7.1]

Memory performance (Stream)

The platform used for our memory evaluation was a ProLiant DL580 Gen8 server running RHEL 7.1. As illustrated in Figure 26, we found comparable results for memory use between containers and VMs.

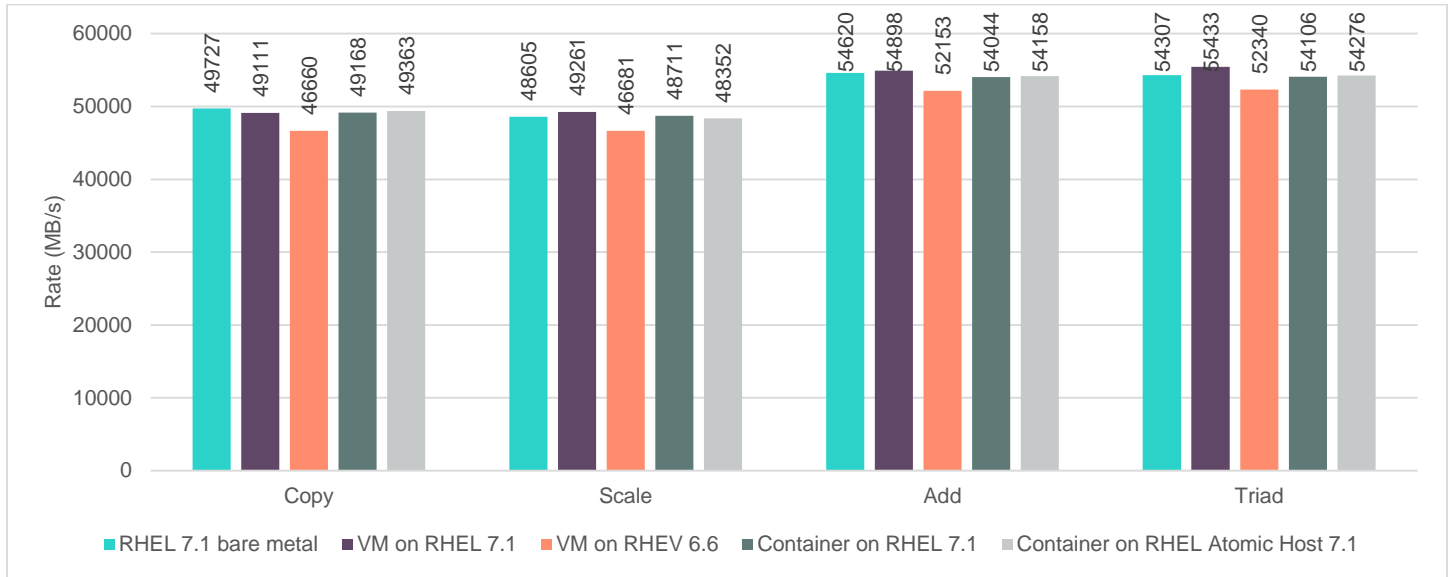


Figure 26. Memory performance results (ProLiant DL580 Gen8 with RHEL 7.1)

Storage I/O performance (FIO tester)

The platform that we used for our storage I/O evaluation was a ProLiant DL580 Gen8 server running RHEL 7.1. As illustrated in Figure 27, we found comparable storage use between containers and VMs.

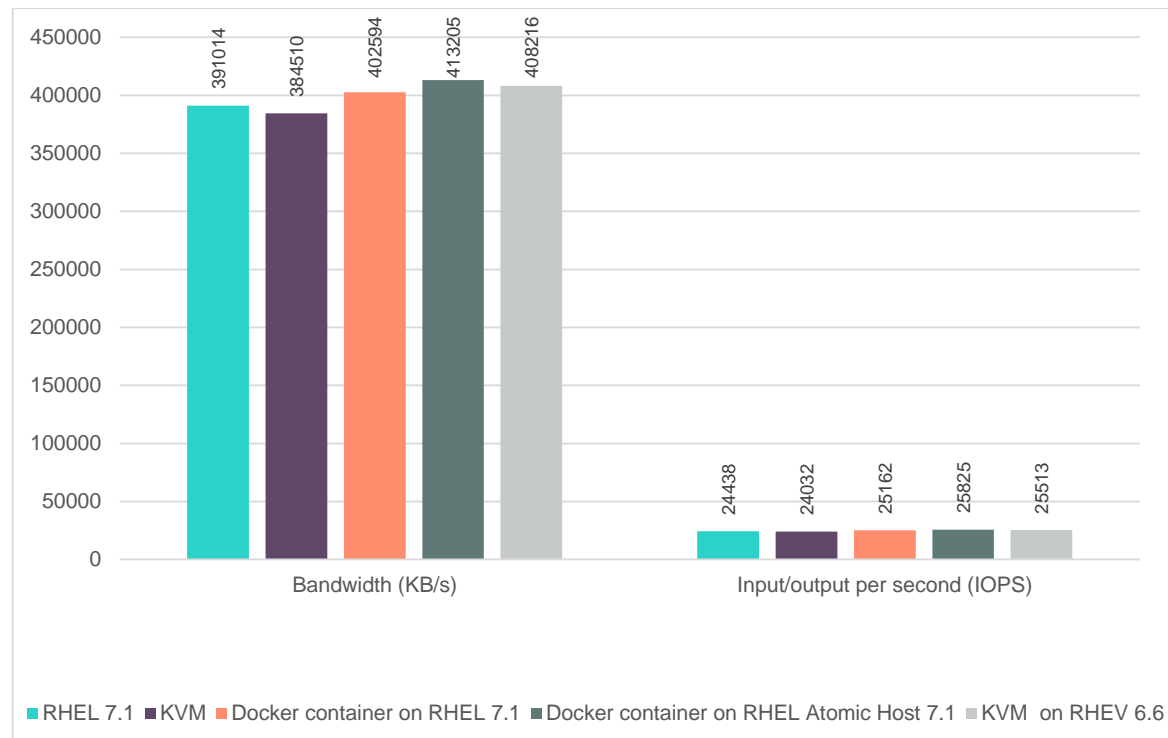


Figure 27. Storage I/O Reads (ProLiant DL580 Gen8 with RHEL 7.1)

As illustrated in Figure 28, the results show that the bandwidth (KB/s) of containers on RHEL 7.1 is 4.5% higher than VMs on RHEL 7.1. Regarding the IOPS, we found container performance on RHEL Atomic Host comparable to RHEL 7.1.

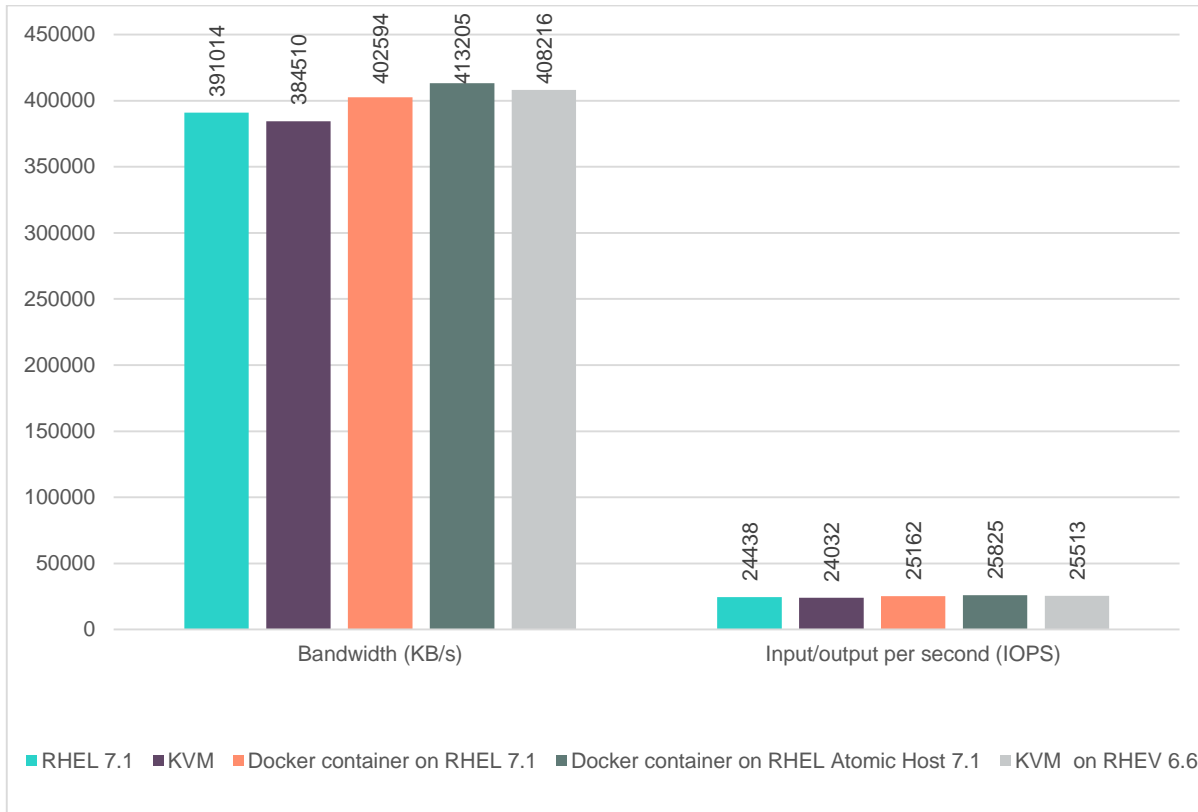


Figure 28. FIO tester write results [ProLiant DL580 Gen8 with RHEL 7.1]

Network performance

As illustrated in Figure 29 and Figure 30, container performance is similar to VM performance in the TCP Stream test, but in the TCP RR test, containers perform much better:

- Transmission Control Protocol (TCP) Stream (10^6 bits/sec):
 - Container performance on RHEL 7.1 is 6% lower than VM performance on RHEL 7.1.
 - Container performance on RHEL Atomic Host is 1.7% higher than VM performance on RHEL 7.1.
- TCP Request/Response (TCP RR):
 - Container performance on RHEL 7.1 is 27.1% better than VM performance on RHEL 7.1.
 - Container on RHEL Atomic Host performance is 23.4% better than VM performance on RHEL 7.1.

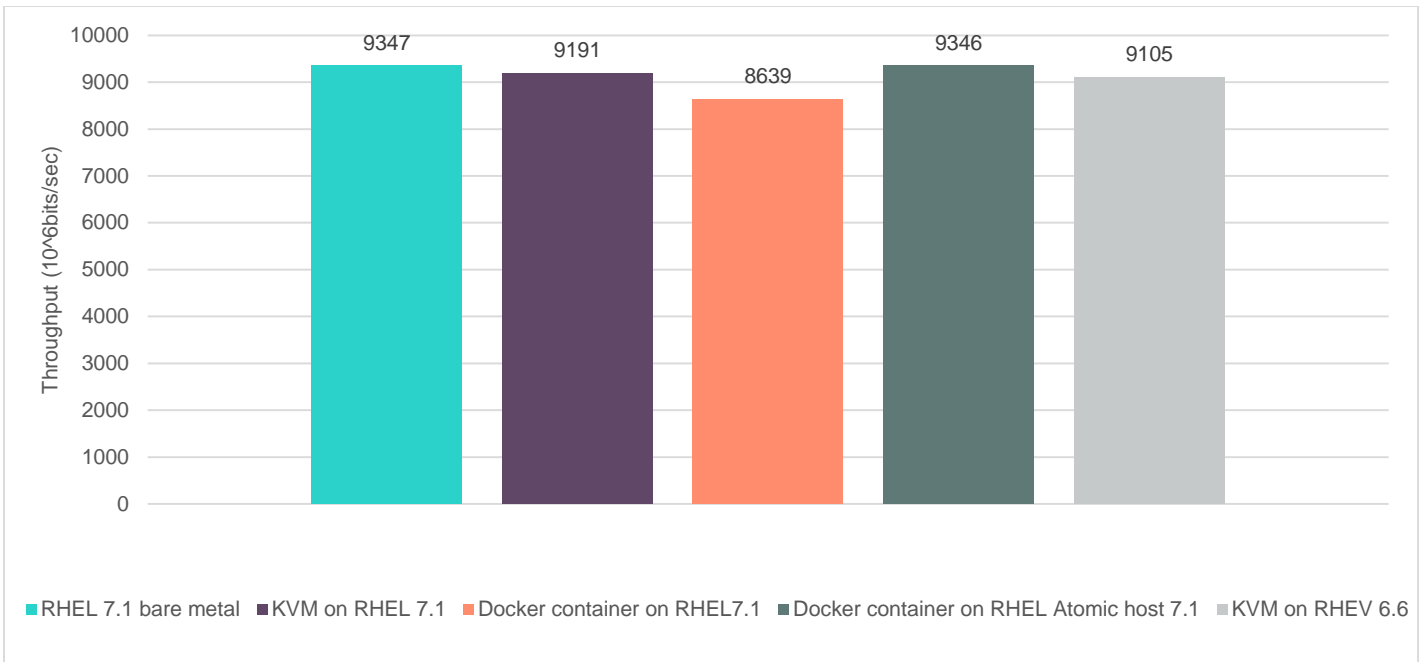


Figure 29. TCP Stream network performance (ProLiant DL580 Gen8 with RHEL 7.1)

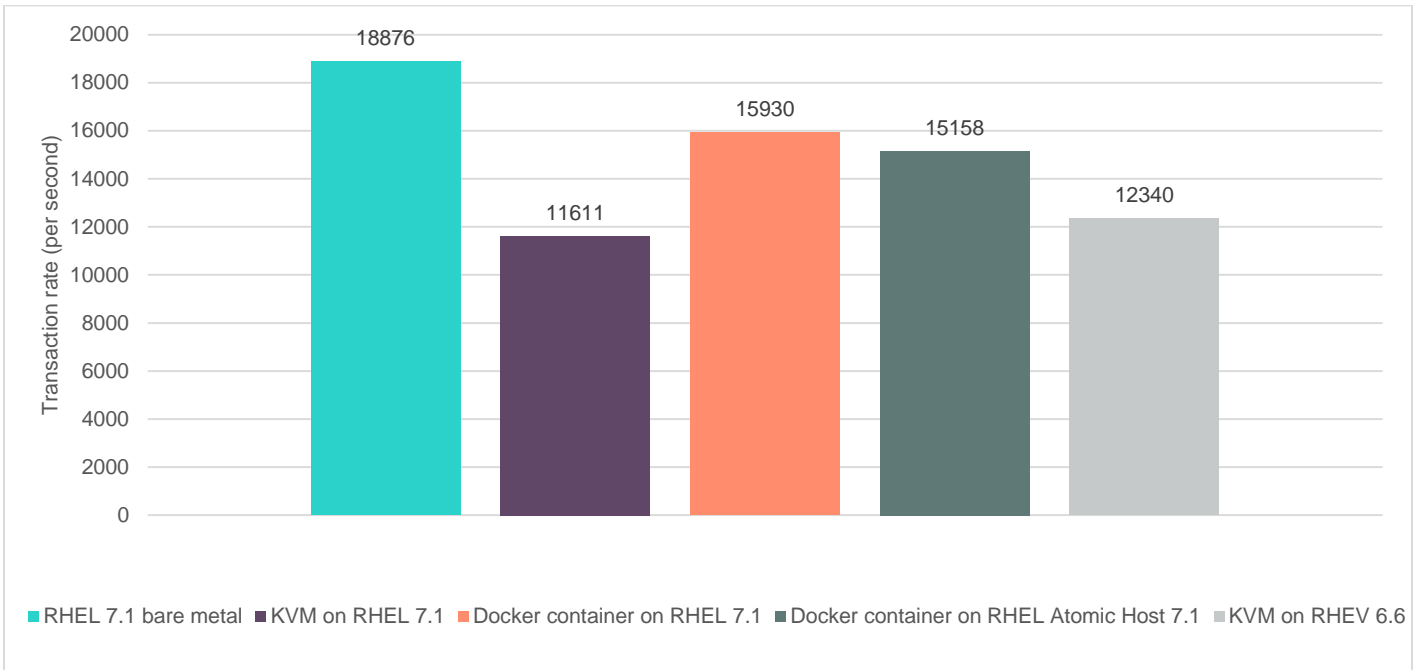


Figure 30. TCP RR network performance (ProLiant DL580 Gen8 with RHEL 7.1)

Figure 31 and Figure 32 illustrate the performance of containers in bridge network mode running on RHEL Atomic Host. Our results show that the bridge network mode of containers on RHEL Atomic host is not very stable.

- TCP Stream (10⁶ bits/sec):
 - There is 14.1% variance between the MAX and the MIN transaction rate among the 10 tests.
- TCP RR:
 - There is 39.03% variance between the MAX and the MIN transaction rate among the 10 tests.

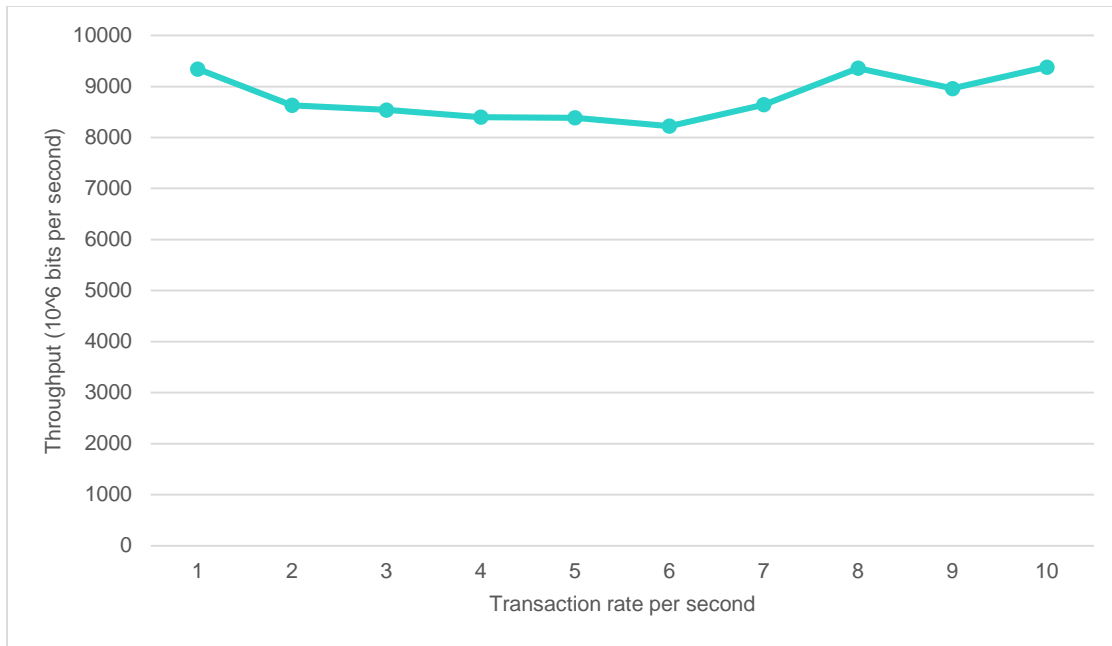


Figure 31. TCP Stream throughput [ProLiant DL380 Gen8 with RHEL 7.1]

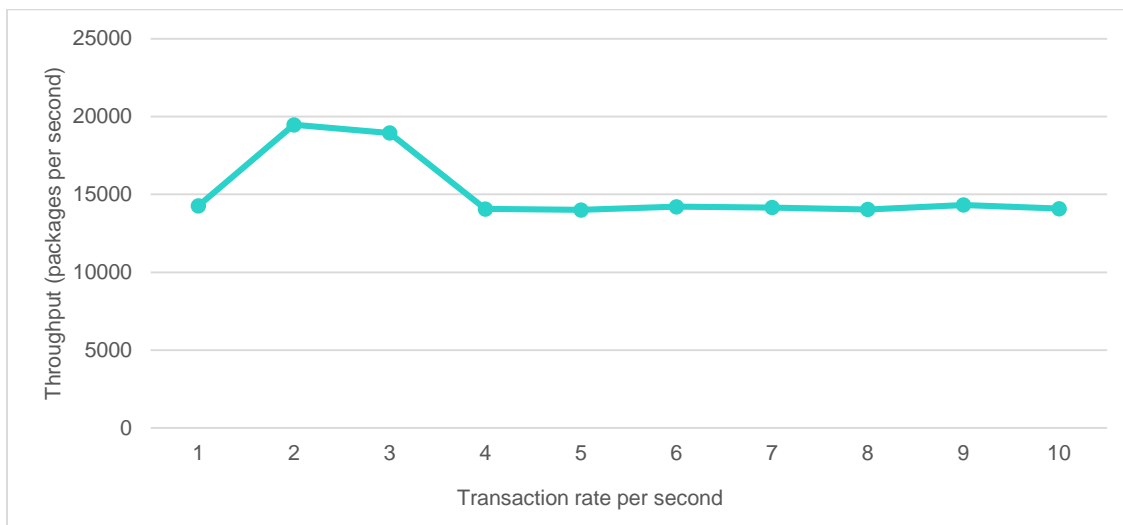


Figure 32. TCP RR transaction rates per second [ProLiant DL580 Gen8 with RHEL 7.1]

Multiuser application environment

AIM7 is a throughput benchmark that exercises many aspects of the OS by using different workloads such as, disk-file operations, process creation, user virtual memory operations, pipe I/O, and compute-bound operations. The configuration of Docker containers and KVMs for this environment are as follows:

- KVM configuration:
 - virtual CPU (vCPU) cores: 48 cores (vCPU=0–23, 48–71)
 - Memory: 128 GB (m=128 g)
 - Disk: VirtIO
- Docker container configuration
 - Cpuset: 48 cores (cpuset=0–23, 48–71)
 - Memory: 128 GB (m=128 g)
 - Disk: Use default Docker storage driver: Device Mapper
 - Data loop file: /var/lib/docker/devicemapper/devicemapper/data
 - Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata

As illustrated in Figure 33 and Figure 34, Sometimes container provide better performance and sometimes VM provide better performance:

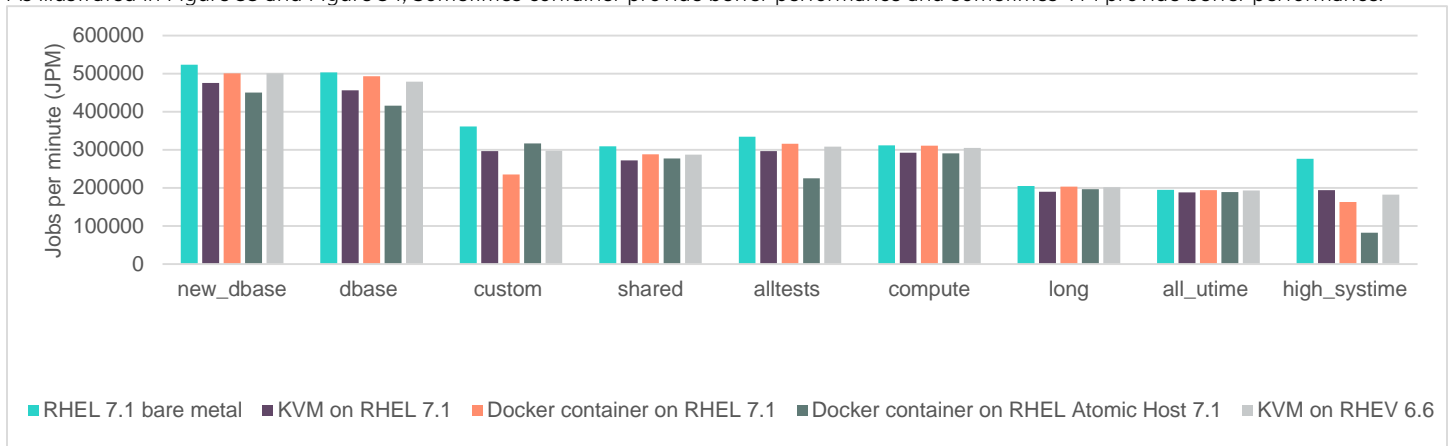


Figure 33. AIM7 throughput (ProLiant DL580 Gen8 with RHEL 7.1)

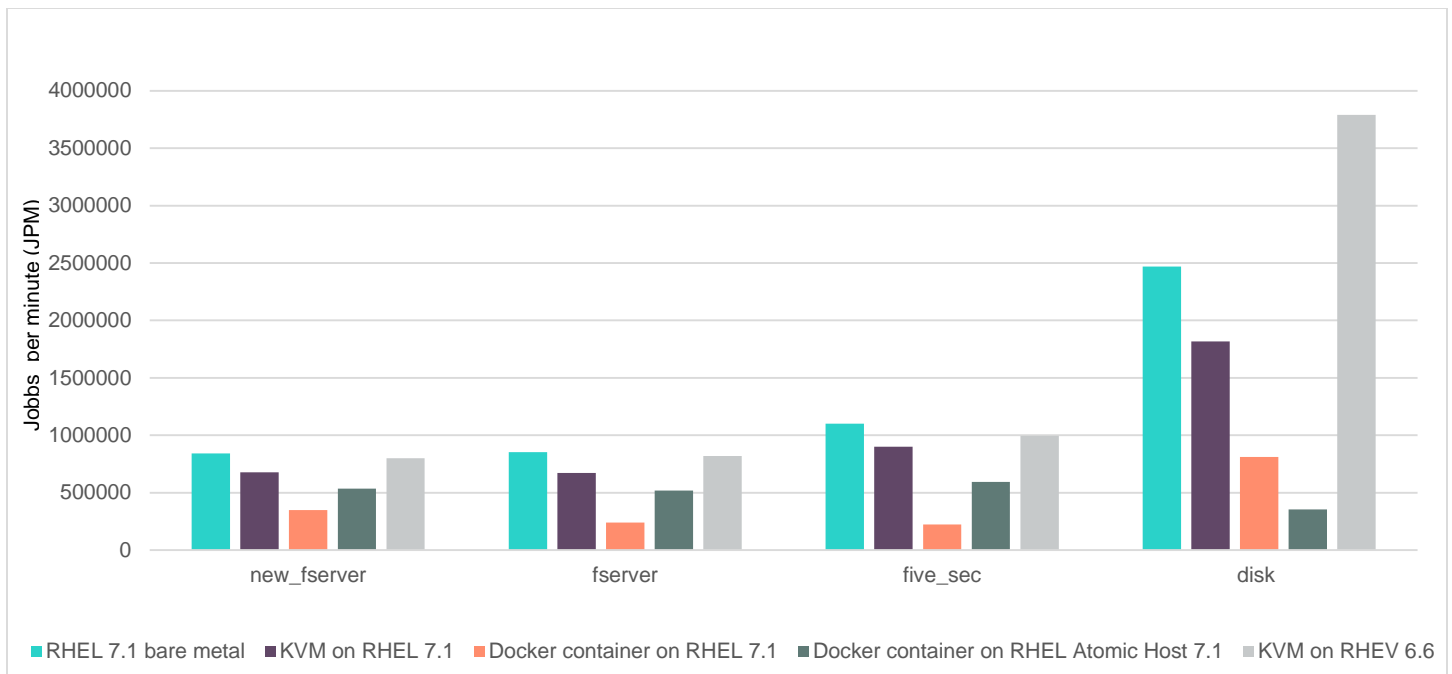


Figure 34. Additional AIM7 workload results (ProLiant DL580 Gen8 with RHEL 7.1)

Conclusion

Containers and VMs are both good technologies that provide an environment for hosting multiple applications on a given server. Containers can be a good choice for environments where there is homogeneity in the expectations among the hosted applications. VMs can be a better choice when stacking applications that require a mixed environment, including things like completely different OSes or significantly different OS versions.

Our testing shows that Containers usually have a performance edge, as long as the mix of apps is suitable to containerization. This performance edge comes usually from the lack of hypervisor overhead, elimination of duplicated OS services inside each VM and less waste of available memory. Our testing also shows that containers become more efficient as the number of containers increases. Figure 35 summarizes our pure performance test results. The figure illustrates that containers perform better than VMs in all of the performance testing. In most of the benchmark testing, we found that containers perform better with RHEL Atomic Host than with RHEL 7.1. The Netperf and Sysbench (mysql) results show that containers perform better than VMs because VMs use VirtIO to manage and transfer data from the VM to hardware, and then from hardware back to the VM.

From a resource perspective, VMs require more memory and processors than containers because each VM has its own kernel. Containers use OS-level virtualization and perform well with fewer processors and less memory. If the memory requirements exceed what is available in the server, both containers and VMs will continue to work; however, container performance remains better. If resource needs increase too much beyond the available system resources, some VMs will hang; containers will continue to work, but could become unstable.

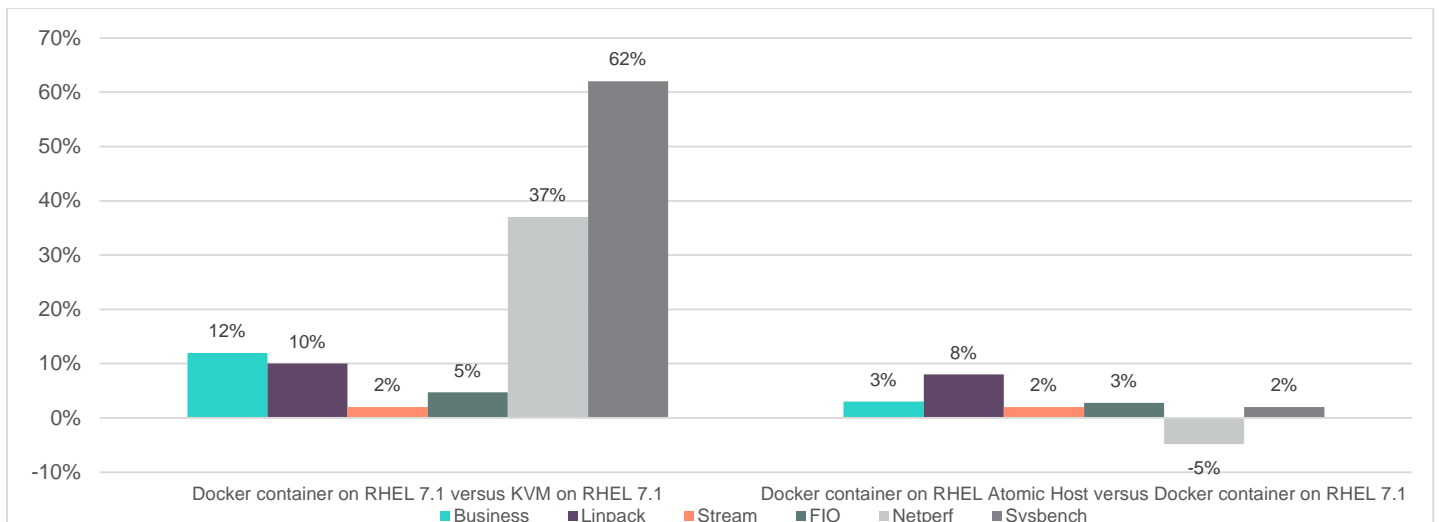


Figure 35. Comparison of workload testing

Key takeaways

As we were conducting our testing, we learned a few things that are useful to consider:

- When using Docker, do not use the container file system (loopback devices), which is the default file system. With this file system, the process for creating containers becomes progressively slower. Instead, use another storage option, such as `--storage-opt dm.thinpooldev`.
- If you have many Docker containers that do not require network access, use the container option “`-net=none`,” Unless you set this option, the network resources are assigned to Docker0, which decreases performance.
- Keep important or rapid read/write data outside of a container. A container’s file system has layers and the layers can decrease performance. For containers in a micro-service system, this means that the same image could replace the data and run on a new file system or on another machine. To avoid this issue, save the data to the host or to storage.
- Consider how you assign processor and memory resources for each container. Resource limits for each container may be required to make sure that containers do not interfere with each other’s resources.
- Allocate a bit more resources to the container than the solution stack requires. If you do not assign adequate resources, a Linux out-of-memory (OOM) killer could occur. OOM killer is a process that the Linux kernel uses when the system is critically low on memory.
- Granting privileges to a container could provide access to non-uniform memory access (NUMA) and result in the potential for improper configuration.

Resources, contacts, or additional links

AIM benchmark

sourceforge.net/projects/aimbench

Flexible I/O (FIO) tester

linux.die.net/man/1/fio

Netlib Repository (includes Linear Algebra PACKage (LAPACK) benchmark information)

netlib.org

Netperf Homepage

netperf.org/netperf/NetperfPage.html

Rally benchmark

<https://rally.readthedocs.io/en/latest/>

STREAM: Sustainable Memory Bandwidth in High Performance Computers

cs.virginia.edu/stream/

Sysbench manual

mysql.com/wp-content/uploads/2014/10/Sysbench-manual.pdf

HPE Servers technical white papers

hpe.com/docs/servertechnology



Sign up for updates

★ Rate this document



© Copyright 2016 Hewlett Packard Enterprise Development LP. The information contained herein is subject to change without notice. The only warranties for HPE products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HPE shall not be liable for technical or editorial errors or omissions contained herein.

Intel and Xeon are trademarks of Intel Corporation in the U.S. and other countries. Red Hat is a trademark of Red Hat, Inc. Linux is a registered trademark of Linus Torvalds.

4AA6-2761ENW, May, 2016