Evaluation of containers as a virtualisation alternative for HEP workloads

2015 J. Phys.: Conf. Ser. 664 022034

(http://iopscience.iop.org/1742-6596/664/2/022034)

Journal of Physics: Conference Series **664** (2015) 022034 doi:10.1088/1742-6596/664/2/022034

Evaluation of containers as a virtualisation alternative for HEP workloads

Gareth Roy[1], Andrew Washbrook[2], David Crooks[1], Gang Qin[1], Samuel Cadellin Skipsey[1], Gordon Stewart[1] and David Britton[1]

[1] School of Physics and Astronomy, University of Glasgow, Kelvin Building, University Avenue, G12 8QQ, United Kingdom

[2] SUPA, School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, United Kingdom

E-mail: gareth.roy@glasgow.ac.uk, awashbro@ph.ed.ac.uk

Abstract. In this paper the emerging technology of Linux containers is examined and evaluated for use in the High Energy Physics (HEP) community. Key technologies required to enable containerisation will be discussed along with emerging technologies used to manage container images. An evaluation of the requirements for containers within HEP will be made and benchmarking will be carried out to asses performance over a range of HEP workflows. The use of containers will be placed in a broader context and recommendations on future work will be given.

1. Introduction

Cloud computing enables ubiquitous, convenient and on-demand access to a shared pool of configurable computing resources that can be rapidly provisioned with minimal management effort. The flexible and scalable nature of the cloud computing model is attractive to both industry and academia. In HEP, the use of the "cloud" has become more prevalent with LHC experiments making use of standard Cloud technologies to take advantages of elastic resources in both private and commercial computing environments.

A key software technology that has eased transition to a cloud environment is the Virtual Machine (VM). VMs can be dynamically provisioned, managed and run a variety of Operating Systems tailored to user requirements. From a resource utilisation perspective however, VMs are considered a heavyweight solution. Upon instantiation a VM will contain a complete copy of an operating system and all associated services leading to an increase in resource consumption when compared to standard "bare metal" deployment. This level of virtualisation is not required by the majority of workloads processed in HEP and can lead to increases in execution time on workloads that perform intensive I/O file operations such as LHC data analysis.

An alternative solution which is gaining rapid traction within industry is containerisation. Here the Linux Kernel itself can virtualise and isolate a user-land level instance of the operating system in which applications can run. Fewer resources are potentially needed compared to a VM because nominally only shared system libraries and files needed by the application are virtualised so performance is close to that of the physical hardware with minimal tuning.

In this study the use of containers is assessed as a mechanism for running LHC experiment application payloads. Using currently available software tools (in this case the emerging standard

Docker [1]) deployment strategies have been investigated by the use of a distributed WLCG Tier-2 facility [2] as an example computing site. In particular the integration of containers with the OpenStack [3] cloud computing platform has been explored as well as strategies and methods for harmonising HEP software availability through CernVM-FS [4].

The relative performance of applications running in containers and VM platforms when compared with native execution will then be detailed by using benchmarking tools and representative workloads. We conclude by outlining some of the issues and challenges that need to be addressed before containers can be considered to be a viable alternative to virtualisation for HEP applications and services.

2. Containerisation

2.1. Overview and History

Linux Containerisation can be described as Operating System (OS) level virtualisation, where the OS itself creates a Virtual Environment (VE) consisting of a set of system resources allocated to a process (or processes) which encapsulate and isolate the execution of that process from the surrounding system. A VE differs from a VM in that a complete, running, copy of an OS is not needed to run a process, instead only the associated libraries needed by that process are required. A VE is lighter weight than a complete VM when running an application and therefore uses less overall system resources. Containerisation or partitioning of system resources was originally implemented in the "chroot" system developed for System 7 UNIX in 1979 [5] and incorporated into BSD version 4.0 in 1982. Similarly, a parallel can be made to Workload Partitions (WPAR) found in IBM AIX systems.

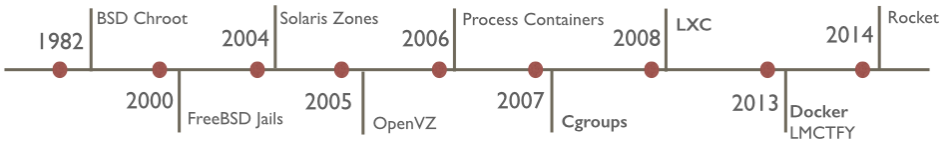

Figure 1. Timeline of software developments leading to Linux Containerisation.

A brief timeline of developments is shown in Figure 1. The "chroot" mechanism developed in 1982 allowed a process (or set of processes) to be executed within a different, apparent, root directory isolating their execution from the main system. The concept of "chroot" was expanded into to BSD "jails" in 2000 [6] and Solaris Containers in 2005 [7] providing a more complete set of fine grained resource controls. This led to VM like features such as saving state, and migrating processes. The main purpose of these VEs was to provide separation for build tasks, run historic code against a fixed set of system libraries, isolating user access and so forth. In 2005 OpenVZ [8] (a precursor to modern Linux Containers) used a modified Linux Kernel to provide separation of system resources that gave it a similar feature set to VM style Hypervisors. In 2007 namespaces were introduced into the mainline Linux Kernel allowing isolation and separation of resources. The creation of namespaces allowed the development of userland tools (such as LibVirt [9] and LXC [10]) to create and work with VEs produced from the mainline Kernel. Current implementations of Linux containers rely on two key technologies, Kernel Namespaces and Control Groups.

2.2. Kernel Namespaces

The isolation needed to effectively separate a set of running processes from the remaining OS is provided by Kernel Namespaces. Currently there are six separate namespaces found in the

Linux Kernel, namely Network, Mount, User, IPC, PID and UTS [11]. The Network namespace provides isolation of the network subsystem and allows each container to have its own IP address, routing tables etc. The Mount namespace prevents containers from accessing filesystems not within that namespace, localising execution. The User namespace controls User IDs and Group IDs allowing privileged execution within a container but not external to that container. The IPC namespace isolates all interprocess communication segregating running processes within containers. The PID namespace controls the assignment of Process IDs allowing IDs to be reused within containers leading to simple portability. Finally the UTS namespace allows each container to have a separate hostname or NIS domain name allowing multiple systems to inhabit a single machine.

Via these six namespaces containers offer similar isolation, portability and flexibility as found in standard VM platforms; allowing the saving of state, simple migration of containers and elastic scaling.

2.3. Control Groups

Another important technology used to create Linux Containers is the use of Control Groups (or CGroups) [12]. CGroups (initially called Process Containers), were developed to offer fine grained control of Linux system resources such as CPU usage. In conjunction with the isolation given by Linux Namespaces, CGroups allow resource constraints to be applied to the processes run in containers and allow a similar level of resource constraint as offered by a standard VM platform. They can be used to restrict CPU, IO, memory and network usage and are split up into hierarchical subsystems which allows new processes to inherit the same resource constraints that are applied to their parents.

2.4. Emerging Ecosystem and Tools

There are a number of emerging tools which can be used to manage the lifecycle of containers. As mentioned in the previous section the creation of tools such as LXC [10] or Google's LMCTFY [14] allowed the simple instantiation of container images. These low-level tools led to the development of platforms such as Docker [1] which enabled distributed access to container images from a central (or self hosted) registry. Another, recently developed, emerging standard is rkt [19] which promotes a standard container description that can be cryptographically signed. This cryptographic signing can be used to verify the provenance of the container which is essential when trusting a distributed image.

The model for container deployment is to provide a minimal hosting environment for a single application. Complex applications are created by combining a number of container building blocks; this has led to a number of different orchestration tools. The first of these, CoreOS [15], is a minimal Linux distribution that hosts a Docker instance and uses systemd [16] to ensure container images are running across a distributed platform. A number of other software vendors have released their own orchestration tools, Docker (the company) released Swarm [17], RedHat released Atomic [18] and Canonical released LXD [20]. These platforms allow the construction of a complicated system from discrete contained applications which can be monitored and managed as cohesive unit. In this study we will focus on Docker and examine how this can be integrated into current cloud deployments.

3. Container Deployment in HEP

To be of use to the HEP community, it is essential that containers are able to access experiment software and to work with current cloud technologies to take advantage of available deployments. In this section we will outline our initial investigation into OpenStack and CERNVM file system (CVMFS) [4] integration using Docker as a management tool for containers.

3.1. OpenStack Integration

It is beneficial for containers to be managed through existing cloud middleware widely used in the WLCG (such as OpenStack). A development platform for Tier-2 cloud resources was used for the evaluation of container deployment. The platform was built around the Juno release of OpenStack and presently comprises three machines: one management node running the network services and Horizon, the web management interface, one compute node running Nova and utilising KVM, and one compute node running Nova with Docker as well as the storage services (Cinder, Glance and Swift).

To enable the use of containers the Nova driver uses a Unix socket to communicate with the Docker ReST API. The docker driver must be obtained and built separately in order to make it available in OpenStack Juno release which was a relatively straight-forward process [13]. Once the driver has been built and configured the list of supported container formats defined in the Glance configuration file must be amended to include "docker" images in addition to those already present.

To make Docker images usable within OpenStack, they must be retrieved from the Docker image repositories and added to Glance. Once the container images have been retrieved, they can be instantiated and managed through the same portal or API in exactly the same way as a standard VM.

3.2. CVMFS integration

For container builds running LHC experiment applications it was necessary to have CVMFS available to access experiment-specific software environments and detector condition information. Due to the root privileges required by CVMFS interacting with the FUSE kernel module [21] extra care needed to be taken during the deployment process.

A container could either be instantiated to provide a CVMFS service to other containers resident on the same host or CVMFS could be installed on the native platform with the CVMFS volume mounted to the hosted container images. In the former option the container hosting a CVMFS service will run in a privileged mode implying root access to the underlying system and other hosted containers. In the latter option there is an impact on virtualisation flexibility; namely the dependence on container image dependent services on the host.

For the purposes of this study the latter option was chosen for simplicity of deployment. At this time no clear solution exists to deploy a container image with CVMFS operating in an unprivileged mode. However there is ongoing development in this area [22] which could resolve this issue.

4. Performance Studies

Performance studies were run to determine whether any penalties were incurred by migrating applications to a container environment and allowed the trade-off between application performance and increased flexibility to be evaluated. Furthermore it was instructive to determine the relative performance of applications running under native and container platforms compared to a VM platform running the same test workload. The performance tests were framed specifically in a HEP context to complement more general benchmark results on container platforms elsewhere [23].

4.1. Testing Environment

Two dedicated servers were used to run all performance tests described in this study. During application execution no other user-based processes were active to minimise extraneous activity on server resources.

Two different server architectures were employed in the study (Xeon E5-2650 v2 and Avoton C2750). The Xeon system comprised two E5-2650 v2 2.6Ghz CPUs, each with 8 cores and a

total of 64GB or RAM. Storage was provided by two 500GB 7200rpm Sata drives. The Avoton system comprised one 8 core C2750 2.4GHz CPU with 16GB or RAM and an 80GB SSD drive. The Avoton processors [24] are a recent generation of low power Atom-based system on chip devices popular for dense server environments. Testing results run on the Avoton-based server were expected to be less performant than the Xeon-based server and were considered a useful cross-check to determine performance consistency across different CPU architectures.

A container image was constructed for performance tests using a standard CentOS 6 container available via the Docker registry and augmented with HEP-specific supplementary libraries and CVMFS through a remotely mounted volume. A μCERNVM image [25] was used to provide the same functionality under the KVM [26] virtual machine platform. An investigation prior to full workload testing determined that the choice of backing storage for the VM (i.e. a raw image file or LVM partition) had little impact on the performance metrics measured on our test systems (with a variation of 0.02% difference in the average measurement).

4.2. Test Definition

The HEPSPEC benchmark [27] was firstly used to evaluate the baseline performance of the three chosen platforms (native, container and VM). The relative performance of HEP applications run at Grid computing centres was then approximated by the construction of tests using three classes of typical HEP workload: full detector simulation, detector reconstruction and Monte Carlo event generation. A profiling harness developed for this study orchestrated these standalone tests running across different platforms and enabled performance data to be collected and compared across multiple samples. The harness also ensured that all the tests were executed serially with no external processes affecting performance results.

Each workflow type was expected to be CPU-bound but with each having a different system resource usage profile during the lifetime of execution. The diversity of the resource profiles gave an indication of the overhead incurred from the virtualisation layer indirectly accessing underlying host resources such as physical memory or disk.

The key performance metric of interest for the HEP workloads was the average CPU time taken to process an individual event in the test sample. These were extracted from the monitoring routines native to the application framework and the same extraction method was applied consistently across all tests.

The number of events processed in each test was scaled accordingly to generate a representative sample and to negate spurious system activity that would inadvertently affect the timing results. At least three iterations of the same test were run to validate per-event timing consistency across samples. The timing results from tests running the same workflow type and on the same server were within 1% of the overall average.

For each workload type one instance of the test was run at any given time. In addition, a further series of tests investigated the performance under a fully loaded system by running concurrent instances of the same workload matching the number of cores on each server.

4.3. Performance Results

The plot on Figure 2 illustrates the HEPSPEC value per CPU core for both of the test servers (labelled Xeon and Avoton) for the 32-bit and 64-bit version of the benchmark. It is observed for all tests that the HEPSPEC values for benchmark execution in the container platform were within 2% of native performance. For the Avoton-based test server a near identical average score was found. In comparison, tests run under the KVM platform were found to be 14.7% less than native performance for the Xeon-based server and 15.3% for the Avoton-based server.

The results for simulation workload tests are shown in Figure 3 where the average CPU time per event as reported by the application is displayed. Results are shown for tests with one running test instance and concurrent instances on a fully loaded system. It was seen that

containers had near native performance in both scenarios and when running on both the Xeon and Avoton processors. For tests running in KVM the simulation workload suffered notable performance decreases of 13.4% (26.4%) for a single process and 24.7% (32.5%) for a fully loaded system on the Xeon (Avoton) based server compared to native performance.

Timing results are shown for the event generation and reconstruction workloads in Figure 4 for tests performed on the Avoton-based server. A similar performance trend is seen compared to the simulation workload and it was again observed that the container platform offers near-native performance regardless of workload type. However the performance reduction for the event generation test running under KVM was only minor compared to other workload types running under the same platform.

It must be noted that the VM performance results were generated on a virtual machine platform without any optimisations applied. Although some effort could be made to generate more favourable results [23] it could not be assumed that VM optimisations were applied consistently on resources available to HEP experiments. The unoptimised results were therefore considered more representative of the performance expected in a production environment.

A selection of the timing results indicated a marginally smaller average CPU time per event for containers than was observed for native execution for the same workload. This was a counter-intuitive result and was in part due to averaging results across multiple tests. One possible explanation found was that the CPU affinity of the container process and the application running natively were handled differently by the host operating system. A container process was found to bind to a CPU core more than a native application during the execution lifetime which may have led to more efficient use of physical memory. Although this was a minor effect this could warrant future investigation if containers were found to be consistently performing marginally better in comparable tests.

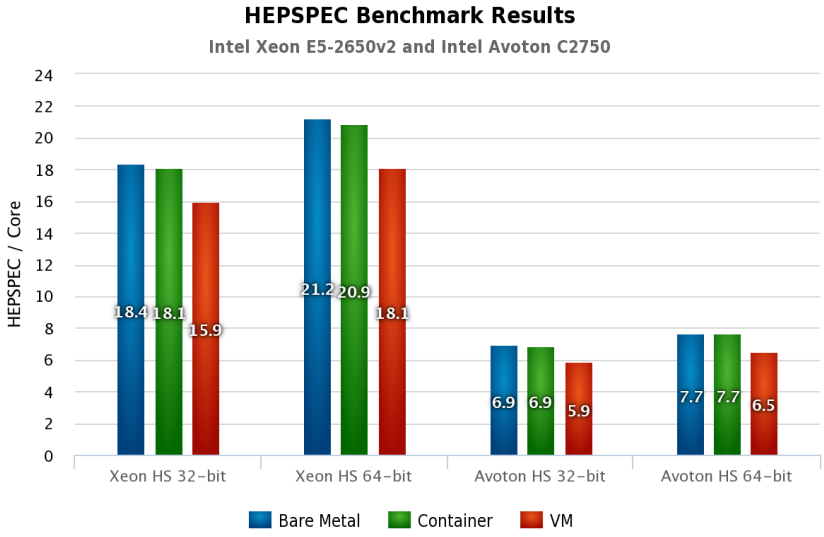

Figure 2. Comparison of HEPSPEC 32-bit and 64-bit results for both the Xeon and Avoton platforms, showing native, VM and container performance.

5. Conclusions

In this paper we have explored the emerging technology of Linux containers and begun evaluation of their use as an alternative to virtual machines in a HEP context. We have explored the history

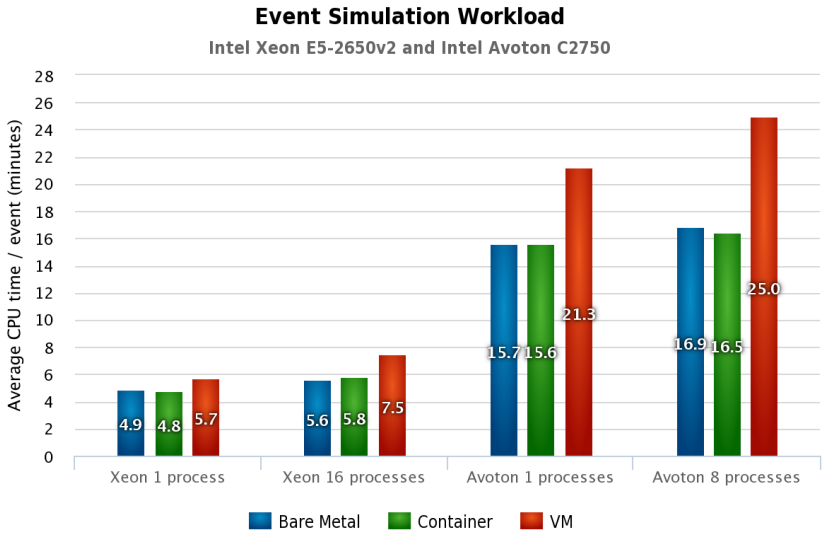

Figure 3. Comparison of Event Simulation results for both the Xeon and Avoton platforms, showing native, VM and container performance.

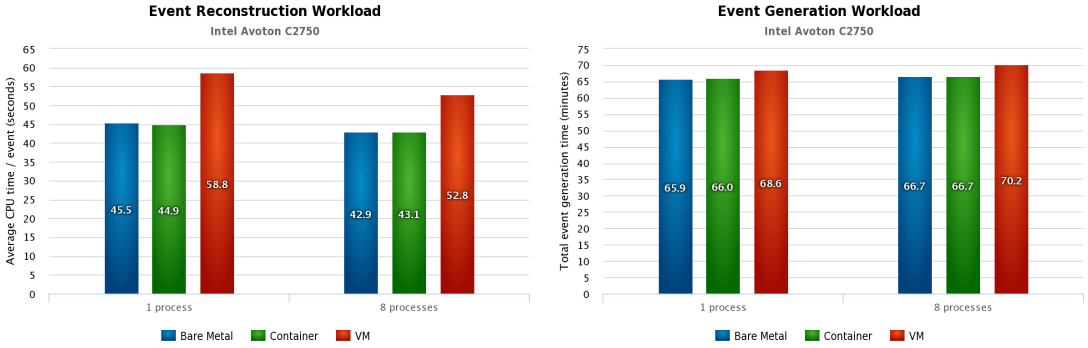

Figure 4. Comparison of Reconstruction and Generation results for the Avoton platform, showing native, VM and container performance.

of containerisation and outlined the technological components that are present in the current incarnation of containers found on the Linux platform. A brief survey of the emerging tools that are being developed to work with containers has also been presented along with an evaluation of the requirements that a HEP container would have.

By defining a suitable container image capable of handling typical HEP computing environments it has been shown, through the use of realistic HEP workloads, that the performance of containers is on par with native execution. Specifically it has been shown that on our test Intel based Xeon and Avoton systems container performance is within in 2% of native performance. Similar tests on the same physical hardware using KVM and the μCERNVM show a much larger reduction in performance (on the order of 10-20%) for the same test workloads.

While these results are compelling, in order to properly validate this study a larger number of simulations needs to be carried out. Running a larger number of simulations on a variety of hardware and cloud platforms as well as other, additional, types of HEP workload would improve

confidence in observed result that container performance is on par with native execution. To carry out a more comprehensive study, work needs to take place to create a generic HEP container image that can be deployed to Grid or Cloud computing platforms similar to the μCERNVM. The combination of this generic container and the test harness developed as part of this work would allow much larger scale testing to take place.

Containers are an interesting new technology which is gaining traction with industry and leading to new models of computation and software deployment methods. It is essential that the WLCG and HEP community investigate the emerging possibilities to gain from the advances being made.

References

[1] *Docker.io* [online] Available https://www.docker.io [accessed 14 May 2015]
[2] *ScotGrid* [online] Available at http://www.scotgrid.ac.uk/ [accessed 14 May 2015]
[3] *OpenStack* [online] Available at https://www.openstack.org/ [accessed 14 May 2015]
[4] *J Blomer et al.*; 2011 J. Phys.: Conf. Ser. 331 042003, "Distributing LHC application software and conditions databases using the CernVM file system"
[5] *UNIX v7 Manual* [online] Available at http://plan9.bell-labs.com/7thEdMan/ [accessed May 2015]
[6] *Kamp, Poul-Henning, and Robert NM Watson.* "Jails: Confining the omnipotent root." In Proceedings of the 2nd International SANE Conference, vol. 43, p. 116. 2000.
[7] *Lageman, Menno, and Sun Client Solutions.* "Solaris Containers - What They Are and How to Use Them." Sun BluePrints OnLine (2005): 819-2679.
[8] *OpenVZ* [online] Available at https://openvz.org/ [accessed 14 May 2015]
[9] *LibVirt: The Virtualisation API* [online] Available http://libvirt.org/ [accessed 14 May 2015]
[10] *LinuxContainers.org* [online] Available https://linuxcontainers.org/ [accessed 14 May 2014]
[11] *Biederman, Eric W., and Linux Networx.* "Multiple instances of the global linux namespaces." In Proceedings of the Linux Symposium. 2006.
[12] *Menage, Paul B.* "Adding generic process containers to the linux kernel." In Proceedings of the Linux Symposium, vol. 2, pp. 45-57. 2007.
[13] *OpenStack: Docker* [online] Available at https://wiki.openstack.org/wiki/Docker [accessed 14 May 2015]
[14] *Let me contain that for you* [online] Available https://github.com/google/lmctfy [accessed 16 January 2014]
[15] *CoreOS: Open Source Projects for Linux Containers* [online] Available at https://coreos.com/ [accessed 14 May 2015]
[16] *systemd* [online] Available at http://www.freedesktop.org/wiki/Software/systemd/ [accessed 14 May 2015]
[17] *Docker Swarm* [online] Available at https://docs.docker.com/swarm/ [accessed 14 May 2015]
[18] *Project Atomic* [online] Available at http://www.projectatomic.io/ [accessed 14 May 2015]
[19] *CoreOS is building a container runtime, rkt* [online] Available at https://coreos.com/blog/rocket/ [accessed 14 May 2015]
[20] *LXD* [online] Available at http://www.ubuntu.com/cloud/tools/lxd [accessed 14 May 2015]
[21] *FUSE: Filesystem in Userspace.* [online] Available at http://sourceforge. net/projects/fuse [accessed 14 May 2015]
[22] *Skeehan, D., et al.* "Opportunistic High Energy Physics Computing in User Space with Parrot. In Cluster, Cloud and Grid Computing (CCGrid)", 2014 14th IEEE/ACM International Symposium on (pp. 170-175). IEEE, 2014
[23] *Felter, Wes, Alexandre Ferreira, Ram Rajamony, and Juan Rubio.* "An Updated Performance Comparison of Virtual Machines and Linux Containers." technology 28 (2014): 32.
[24] *Burres, Bradley, et al.* "Intel Atom C2000 Processor Family: Power Efficient Processing Optimized for the Data Center." Micro, IEEE , vol.35, no.2, pp.26,34, Mar.-Apr. 2015
[25] *Blomer, Jakob, et al.* "Micro-CernVM: slashing the cost of building and deploying virtual machines." In Journal of Physics: Conference Series, vol. 513, no. 3, p. 032009. IOP Publishing, 2014.
[26] *Kernel Virtual Machine* [online] Available http://www.linux-kvm.org [accessed 14 May 2015]
[27] *Merino, G.* "Transition to a new CPU benchmarking unit for the WLCG." HEPIX Benchmarking WK (2009).