Docker Containers
A new way of high performance computing?

Filip Reichel
M.Sc. Computer Science
Computer Science and Mathematics
OTH Regensburg, Regensburg, Germany
Email: reichel@eumx.net

***Abstract*—The purpose of this work is to examine the usage of Docker Containers within the area of High Performance Computing. Docker containers can be seen as small operating systems that serve for running only one single program or service. Since it is not very easy to understand the needs of modern HPC clusters without prior understanding of the underlying techniques an introduction to the Docker principle and to its main technologies will be given. While this may seem a little bit technical at first sight it is necessary to understand the following examples of scientific applications that will be presented afterwards. They main goal of HPC is to get complex calculations done on a limited amount of resources. It is important to use the available hardware as efficient as possible to reduce the overall calculation time and improve the throughput.**

The current state of Docker containers for HPC is promising and they start getting adopted on a wide basis. However there were two big problems that needed to be solved. One problem was the bad network performance which needed to disable the separate namespaces for networking and using a bridged configuration instead. The second big obstacle was the filesystem overlay which should actually prevent the container from writing to the host's filesystem directly. This decreased write performance and needed to be bypassed by mounting the filesystems directly as data volumes.

The big advantage of using Docker is the very small footprint a container has on the host's operating system. It only consumes resources as work is done in the container. Also the easy management of the containers and the tools provided by the Docker project add to the success of the new technology. Finally the overall performance of Docker containers is very good and reaches nearly that of native operation.

***Keywords*—docker container; virtual machines; LXC; high performance computing; system architecture;**

I. Introduction

Performing complex mathematic calculations requires squeezing the last bit of power out of the given hardware. The operation of such a high performance computing cluster (HPC cluster) is a costly endeavour. It is important to max out the usage of the servers to keep operating costs at an acceptable level. One important application of HPC clusters is the calculation of the complex models required to generate the daily weather forecasts. Furthermore it is common to perform calculations for clients from the industry and the finance sector. As can be seen the variety of use cases for a HPC cluster is enormous. As a consequence of that it is not surprising that the requirements of the clients become more and more complex and often require the support of new technologies and new computing models. Among those new technologies are Hadoop which uses Map-Reduce-Algorithms and MongoDB as a representative of the new NoSQL databases [8, 9].

Since many modern technologies were not designed specifically for the deployment in distributed computing environments workarounds had to be found. Another problem was that the new software might have dependencies that the existing operating system could not satisfy or that even conflicted with already installed versions in the system. To make those new applications work with the existing environment required a lot of trial-and-error to find appropriate solutions. Since it is not good to use the productive system to try out new things the only way to keep the productive system clean is to use another host for testing purposes. In former times it was the usual way to setup a dedicated server to test out new things. Doing this on a regular basis took a lot of time. So new ways of testing had to be found.

In the beginning the researchers had to test on virtual machines which meant that resources from the host system were given to the guest operating system. This did work out quite well but it also meant to have more powerful hardware for the host system. In many cases all this overhead was unnecessary. In 2013 a new system called "Docker Containers" was announced. The technology behind Docker was not really new, it actually existed for quite some time in the linux kernel [7]. However Docker created a set of tools and an API around the necessary parts to make its use more convenient. From now on it was possible to use linux containers within a linux host operating system.

II. Linux containers

In order to understand the benefits of Docker Containers it is necessary to take a closer look at the underlying principles of linux containers (LXC). The main difference is the target that is emulated, it can either be the hardware that is virtualized (HVM) or the software which is running on it. LXC uses the latter approach and emulates the software that is to be executed.

A. Hardware virtualization versus container virtualization

Emulating a whole computer system with its hardware is called hardware virtualization. It behaves similiar to using a dedicated stand-alone server. Nevertheless it only emulates the physical hardware. A stand-alone server always needs an operating system on it and of course the actual software to be tested. So for a quick experiment the effort is way to high to do that every time from scratch. Actually it would be sufficient to emulate only the software to be tested.

This is where linux containers (LXC) come in handy. They do provide exactly what is needed — a container just for the software to be tested. This means that only the processes that the container creates for testing are running within the containers environment. All other parts that are necessary to execute the code need to be supplied by the host operating system.

B. System Architecture of LXC

The underlying concept is of course a little bit more complex. In order to achieve its goal of running as a lightweight process in the kernel LXC needs to run isolated from the host environment. That is necessary because of security reasons. LXC containers need to have root priviledges for operating properly and could harm the host if not secured properly.

In order to isolate the container's concerns from the host's a separate kernel and user namespace are used. As a result the container does not see the processes of the host and they do not interphere with each other. Another benefit of doing this is that it is guaranteed that the root of the container cannot gain admin access to the host's operating system. The second measurement to secure the host is to prevent the guest from changing the host's system files. This is realized by using a copy-on-write-filesystem. Copy-on-write means that the actual filesystem has an overlay which blocks all modifications to files and records the changes in an abstraction layer instead. Lastly the container shares resources with the host so that it only needs to execute the missing dependencies itself. This is also the main danger for the LXC, if the host dies, the guest also is not operating any more just like in an ordinary HVM. Of course there exists also a resource management, called 'cgroups', to restrict resources for the LXC. In addition to managing resources they can also deliver statistics from the LXC, its current state or its healthiness [7].

C. Benefits of using LXC

The main advantage of this architecture is that it uses resources much more efficient than pure hardware virtualization. Since almost all dependencies are taken from the hosts operating system only dependencies that either are missing on the host or are conflicting with them need to be executed by the container. This also affects the startup und shutdown time of the container. From the view of the host it is only one new process per container that needs to be handled. Furthermore a container does only consume resources from the host if it is executing some work. By using AuFS as a layered filesystem it is possible to use one base image for many different containers, this saves disk space. It is also possible to version container images so that only the changes from the last saved state need to be stored. The characteristics presented above make it easy and fast to deploy new containers on any docker compatible host [7].

III. Docker Containers

As mentioned before the technology behind Docker is not new since it uses LXC. Almost all components existed already in the kernel and were considered production ready. The revolutionary about Docker is that it made these technologies accessible with a small set of tools and an unified API.

A. Docker repositories

"One of Docker's killer features is the ability to find, download and start container images that were created by other developers quickly. The place where images are stored is called a registry [...] You can think of the registry along with the Docker client as the equivalent of Node's NPM, Perl's CPAN or Ruby's RubyGems." [7]

Docker's public registry offers a wide variety of base images to start customizing your own images with and images that contain standard software like web servers, databases and content management systems. It is also possible to maintain private repositories that can be used to store proprietary code.

B. Running Docker images

Docker containers can be managed easily by using the command line interface or by sending remote requests to the REST API. The CLI can search the public repositories for images and download the desired image onto the machine. Docker then checks automatically if the requested image was built from other images and downloads them as well if needed. The container then only needs to be started and is ready to use. Docker automatically creates a virtual network interface for each running container. The network address can be obtained by showing the containers configuration. When the container is not needed any longer it can be shutdown immediately. [1, 7]

IV. Docker in scientific applications

The usage of Docker containers in scientific applications has recently gained momentum. Several studies tried to examine the performance impact of using Docker containers regarding performance, throughput and scalability.

A. Performance comparison of Virtual Machines and Linux Containers

Cloud computing makes extensive use of virtual machines to balance workload and manage resources. Felter et al. examined in a recent study whether linux containers can provide enough performance for scientific applications [4]. They used the hardware hypervisor KVM [2] and the container manager Docker for their comparisons. All tests were executed on a IBM System x3650 M4 server (2x Xeon E5-2665 (16 cores), 256 GB of RAM). The researchers found out that the Docker containers outperform the KVM solution in all aspects examined. There is nearly no overhead on CPU and memory, but the disk throughput is better with Docker because of the use of volumes. If the file system overlay AuFS is used the performance is nearly as poor as KVM's. To handle high disk throughput the use of volumes is the better option. When using network intensive applications it has to be taken care of the negative impact of network address translation (NAT), because each packet has to be modified before sending it out to the hosts network interface. Unfortunately the benefit of a separate network namespace will get lost without NAT. Felter et al. stated that it may be possible to get good performance from KVM if configured very carefully, but this is an error-prone and complex task. They concluded that the application of ressource intensive tasks is possible but ease of use and performance must be carefully weighted against each other.

B. Containers in Research: Initial Experiences with Lightweight Infrastructure

The research team at Purdue University in Indiana had to face the problem that the users of the HPC Cluster needed a setup that was by default not supported on this HPC system. As they estimated an implementation time of several weeks to fullfill the requirements they had to look for other possibilities [5].

Julian et al. tried to implement a High Performance Computing (HPC) Cluster based on Docker containers. They used the Moab HPC scheduler to create and destroy Docker containers based on the current workload. They found out, that the time to implement new requirements could be cut down to one day instead of several weeks. However they had to find a solution to the problem of docker's filesystem overlays as they decreased performance significantly. In the end they found a plugin for docker that mounts NFS shares directly into the container. Although not the ideal option it improved throughput so that it nearly reached native performance. The team noted that there are still missing important features from HPC applications like remote direct memory access (RDMA) and native support for data volume management for parallel filesystems. They expect Docker to be widely accepted in HPC environments when these design flaws will have been fixed in the future.

Recently HPC tool vendors have also begun integrating native support for Docker. IBM for example has added Docker container integration in its Platform LSF to run containers on an HPC cluster. This allows containers to be executed on an LSF-managed cluster like a conventional job, but with a fully self-contained environment [6]. The xCAT cluster provisioning suite has also added Docker integration. The latest versions now support using containers to manage xCAT clusters [3].

V. Conclusion

The two reports presented before point out clearly the potential of the new technology. It allows the operators of HPC clusters to run software that has dependencies which can not be satisfied by the operating system. Another interesting possibility is to run more than one job on the same computing node. Thanks to the architecture of Docker containers it is no problem to run several containers on one single node as a new container is only a new process to be executed from the view of the host's kernel. Although the implementation of Docker containers within the professional tools for managing HPC clusters is at an early state it was possible to reach near native performance. The researchers found out that the filesystem overlay which is a core feature of LXC has to be bypassed to improve disk write performance. Furthermore the usage of separate network namespaces is decreasing the overall network performance since all packages sent out to the host have to be modified. When using the network interface in bridged mode the throughput reaches the expected values. One problem that needs to be solved to operate properly is the access to paralleled filesystems over the network. At the moment there is no official plugin available to mount NFS shares properly, only a user contributed plugin can be used for mounting such data volumes directly. Finally the direct remote access to system memory over network is missing for a fully powered HPC infrastructure. Since the community behind Docker is very active in this field it will only be a matter of time until the missing features will be implemented properly.

References

[1] "Docker Inc." 2016 (accessed 28.12.2016). [Online]. Available: https://www.docker.com/

[2] "KVM," 2016 (accessed 28.12.2016). [Online]. Available: http://www.linux-kvm.org/

[3] "Docker," 2015 (accessed 28.12.2016). [Online]. Available: http://xcat-docs.readthedocs.io/en/stable/advanced/docker/index.html

[4] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 171–172.

[5] S. Julian, M. Shuey, and S. Cook, "Containers in research: Initial experiences with lightweight infrastructure," in *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, ser. XSEDE16. New York, NY, USA: ACM, 2016, pp. 25:1–25:6. [Online]. Available: http://doi.acm.org/10.1145/2949550.2949562

[6] B. McMillan and C. Chen, “High performance docking. technical report,” 2014 (accessed 28.12.2016). [Online]. Available: https://public.dhe.ibm.com/common/ssi/ecm/dc/en/dcw03059usen/DCW03059USEN.PDF

[7] D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[8] E. Redmond and J. R. Wilson, *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf, 2012.

[9] L. Wiese, *Advanced Data Management: For SQL, NoSQL, Cloud and Distributed Databases*, ser. De Gruyter Textbook. De Gruyter, 2015.