EPJ Web of Conferences **214**, 07018 (2019) https://doi.org/10.1051/epjconf/201921407018
CHEP 2018

Grid services in a box: container management in ALICE

Maxim Storetvedt[1*], *Maarten* Litmaath[2], *Latchezar* Betev[2], *Håvard* Helstrup[1], *Kristin* Fanebust Hetland[1], and *Bjarte* Kileng[1]

[1]Faculty of Engineering and Science, Western Norway University of Applied Sciences, Bergen, Norway
[2]CERN, Geneva, Switzerland

Abstract. Virtualization and containers are established tools for providing simplified deployment, elasticity and workflow isolation. These benefits are especially advantageous in containers, which dispense with the resource overhead associated with virtual machines in cases where virtualization of the full hardware stack is not considered necessary. Containers are also simpler to setup and maintain in production environments–deployed and currently operational systems serving end-users, where service disruptions should be avoided.
This contribution addresses container configuration and deployment to run central and site services on the ALICE Grid system, specifically to achieve containerized VO-boxes. We describe the methods through which we minimize the manual interaction, while retaining the simplicity and scalability associated with container deployment, the so-called "service in a box". Furthermore, we explore ways to increase fault tolerance, aimed at reducing the risk of service downtime, and identify possible performance bottlenecks.

1 Introduction

Containers are an implementation of the operating-system-level virtualization paradigm, using mechanisms such as Linux cgroups and namespaces. Running as isolated wrappers on top of a preexisting kernel, containers can reduce the resource overhead for situations where virtualizing the full hardware stack is not necessary. This enables containers to have close to no overhead, achieving performances comparable to that of a native operating system (OS).

The lack of overhead allows containers to be both simple to deploy and maintain, while allowing for features such as elasticity and workflow isolation. These benefits have earlier motivated both the ATLAS [1] and LHCb [2] experiments to investigate and adopt containers within their Grid environments [3][4]. CMS [5] Grid jobs make use of the Singularity [6] container manager since 2018 [7].

To investigate the viability of containers within the ALICE [8] Grid, a number of ALICE VOBOXes [9] have been redeployed within containers. A VOBOX at a grid site is a machine on which VO-specific services may be run for a supported VO. Each ALICE site operates at least one VOBOX on which ALICE-specific services handle job submission to the site, as well as monitoring the site's jobs, storage services and network performance. In this pilot project, the containerized VOBOXes are expected to be capable of satisfying the same

*e-mail: msto@hvl.no

requirements as expected from Virtual Machines (VMs), specifically in regards to performance and availability. VMs are actively used in production environments to run VOBOXes, and have usually proven to require little-to-no manual interventions while achieving minimal downtime and solid performance. Exceptions to that rule, discussed in Sec. 4, sparked the idea of using containers instead.

While using containers within a development/testing environment is a straightforward process, having containerized VOBOXes in a production environment introduces more complexity. Several containerized VOBOXes have now been deployed in such environments within the ALICE Grid since mid-2017, where the initial experiences in using and managing such containerized VOBOXes will be the main focus of this contribution.

Containers will also be used within ALICE to provide isolation for Grid worker nodes–a feature found within the new JAliEn Grid middleware [10]. This is detailed in a separate contribution dedicated to JAliEn [11].

2 Key configuration points

Compared to the more common VMs, containers have no virtual hardware stack. This difference becomes evident when attempting to use and configure containers for purposes typically handled by VMs, such as VOBOXes, and preparing them for use within production environments. Appropriate steps must be taken to ensure sufficient stability, security and performance.

2.1 Platform

All the containerized VOBOXes deployed in production environments within ALICE use Docker [12] as their container base. This platform is both mature and well documented, and contains all the features necessary to run the appropriate services. VOBOXes require a full networking stack to function as intended, as opposed to the container bridging found within Docker–which in reality uses a network address translation (NAT) scheme to provide networking.

2.2 Networking

VOBOX containers are networked using the MACVLAN [13] network driver. Within Linux systems the implementation can be compared to a ”reverse VLAN”–as opposed to mapping the OS-side of a network interface to multiple virtual networks, the network side of an interface is mapped to multiple virtual interfaces. This is illustrated in Fig. 1, where a single physical interface is supplied with two virtual interfaces (one for each of the two illustrated containers). Each of these interfaces is assigned a MAC-address, and traffic sent from the virtual interfaces is sent directly to the underlying network[1]. This allows containers networked using MACVLAN each to have their own IP address, and appear as conventional machines on the network.

[1]As traffic is routed directly to the underlying network, the virtual network interfaces will be invisible to the physical host.

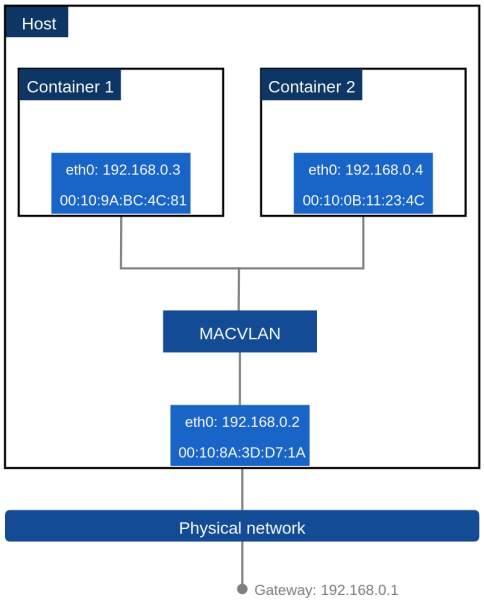

Figure 1. MACVLAN architecture overview for a possible network with two containers. Each container is assigned a virtual interface through MACVLAN, with their own MAC and IP addresses.

2.3 Host configuration

Unlike the more traditional VM-hypervisor, a container host shares its kernel with its running containers. While this contributes to less overhead, it creates challenges when attempting to use containers for VOBOX purposes:

- The deployed VOBOX containers are responsible for running multiple services. With each container sharing the host kernel, this is equivalent to having all services running on the host itself. As the number of containers grows, the number of processes on the host will increase rapidly. Consequently, the number of used file pointers will quickly reach the user maximum[2] causing crashes, system hangs and process terminations. For more than two VOBOX containers on a single host, the system default value must be increased.
- AutoFS–a software used to automatically mount directories on demand–must be disabled to allow stable CVMFS [14] access. This is caused by a known bug within AutoFS when used in conjunction with containers: if the AutoFS directory has not been accessed on the host first, it will fail within all containers. Disabling AutoFS bypasses this issue. A script can be used to automatically mount the required directories instead.
- For the purpose of isolation, Docker will limit the access privileges that containers have on the underlying kernel. Using the default settings, many common tools and services used within VOBOXes fail to start. Granting access to kernel capabilities is possible, and will allow more tools to function. It is also mandatory in order to deploy containers in production environments. As a result, all VOBOXes used within ALICE run with full, or near-full, access privileges. Considering the convenience and stability provided by having full access to the kernel capabilities, the risk was considered minimal given that the containers will mainly be handled by system administrators.

[2]Defaults are set to 4096 (hard) and 1024 (soft) on CentOS 7.

3 Preventing downtime

The containerized VOBOXes within ALICE all use the "Live Restore" [15] feature provided by Docker. When enabled, it allows containers to exist independently of the background Docker service (dockerd)–as opposed to having all containers terminate should the service crash or disappear, which is the default Docker behavior. In addition to the added redundancy, this feature also makes it possible to update Docker without causing downtime. However, the feature is only intended as a temporary measure. Containers must occasionally offload their stored logs, for which they depend on the dockerd service. If the service remains absent for longer durations, the container log-buffers will eventually overflow, causing crashes.

Should a container fail or terminate, a container management tool can be used to handle automatic restarts, such as Swarm [16] and Kubernetes [17]. While the former comes bundled with Docker, and the latter is used elsewhere within the Grid, none of these are currently used within ALICE to manage site-service containers. Given the low number of containerized VOBOXes per site, the effort required to manually maintain these containers does not currently necessitate any additional tool.

Restarted containers need ways to quickly restore their previous services. Within ALICE, containerized VOBOXes use Dockerfiles to start all necessary services at launch. While this is also possible using 3rd-party tools, the current approach requires no additional software installations or configurations. However, any changes to a Dockerfile requires rebuilding and redeploying the associated container image, which results in downtime. To avoid excessive downtime when testing new configurations, a separate script was introduced, containing all commands to run at launch. The script is placed in the container base image at "/etc/init.sh", with a static reference to the location kept in the Dockerfile. The commands within this file can be freely changed, without having to rebuild the container image to apply changes. This flexibility comes at the cost of having to maintain a separate script within each container, as opposed to having a single Dockerfile.

4 Observed performance

Containerized VOBOXes have been operating in production environments within ALICE since July 2017. This has allowed performance to be monitored over longer periods, with different configurations and storage drivers–such as AUFS and Overlay2. On average, containers were found to be able to run as many (or more) jobs, with less load on the host system compared to conventional VMs. A comparison can be seen in Fig. 2 and Fig. 3, where a containerized VOBOX is compared to a VM VOBOX in terms of performance and load for the same interval. For a few days, the containerized VOBOX was configured to manage many more concurrent jobs compared to the VM in Fig. 2, while Fig. 3 shows the load on the underlying system to be less than on the VM for the same interval.

It must be noted that container I/O performance will gradually degrade as additional changes are committed when using file-level storage drivers like AUFS or Overlay2. As with most copy-on-write filesystems, all changes are stored as a separate layer on top of the base container image, with a new layer for each commit. When using the AUFS storage driver, each of these layers is composed of a read-only AUFS branch (directory), except for the top layer used by the running container which remains writable. The first time a container writes to a file, the file has to be located in the corresponding lower AUFS branch and thereafter copied into the current writable layer. As the filesize grows and the number of layers increases, so too will the I/O latency[18]. To reduce the impact on performance when using

file-level storage drivers, container images must thus be flattened before being put into a production environment–having all additional layers merged into one. This is done by exporting and re-importing the image through Docker.

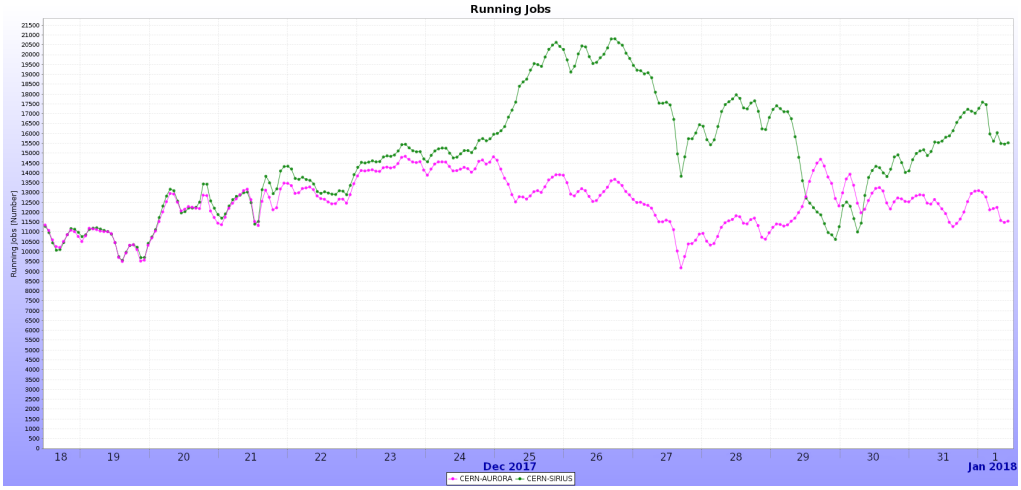

Figure 2. Container VOBOX (green) running jobs in a production environment, compared to a VM VOBOX (magenta) from December 18, 2017 to January 1, 2018. The container was configured to manage many more concurrent jobs, which resulted in 114k more jobs being handled (a 13% increase) compared to its VM counterpart at the end of the duration.

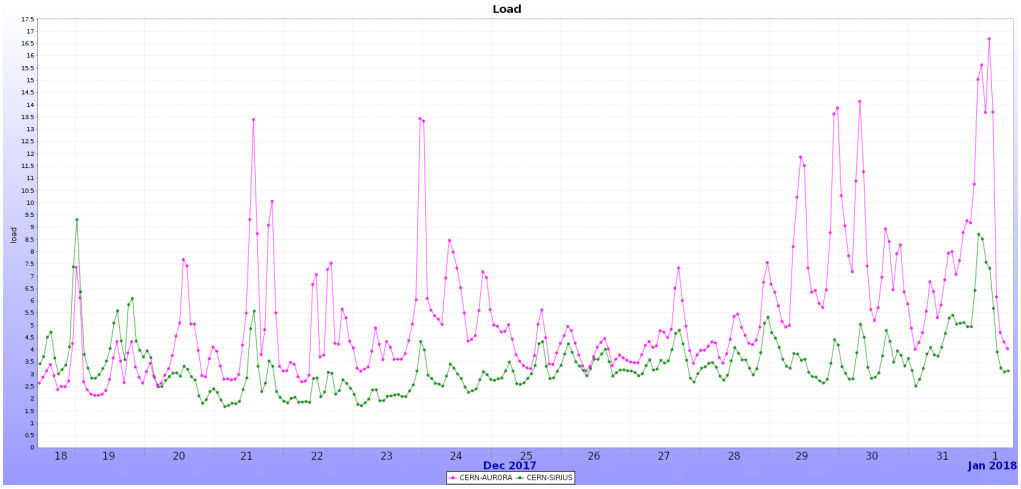

Figure 3. Display of the load on the underlying system for the container VOBOX (green) and VM VOBOX (magenta) from Fig. 2, for the same interval as above. The container can be seen to have less load on the underlying system when compared to the VM.

From a stability perspective, containers were also found to be less prone to faults compared to their VM counterparts in the ALICE central-services cluster. None of the containerized VOBOXes experienced connectivity issues or other odd behavior–a previously recurring

issue for several VOBOXes running in VMs. Figure 4 shows the maximum number of concurrent jobs for an ALICE VOBOX (Corona), a VM VOBOX known for underperforming, while also being prone to network freezes–no longer responding on the network, making it unable to handle jobs until it is rebooted manually. The VOBOX was redeployed as a container in early November, showing an immediate gain in the maximum number of concurrent jobs. No network freezes or other connectivity issues have been observed since redeployment.

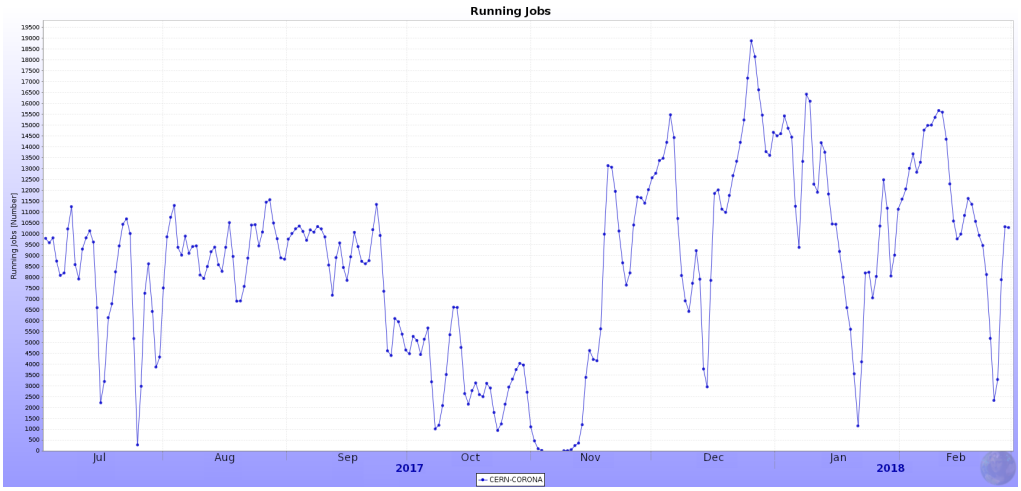

Figure 4. Running jobs for the VOBOX Corona. Initially a VM, before being redeployed as a container in early November. The maximum number of concurrent jobs was seen to increase from this point, without unexpected network freezes.

5 Outlook

With container VOBOXes running within production environments for over a year, yielding positive results in terms of both performance and stability, ALICE is ready for VOBOX services to be run in containers where this is desirable. A number of VOBOXes for new use cases have thus been deployed as containers instead of VMs.

As mentioned in Sec. 1, containers will also be used to provide payload isolation for Grid worker nodes. This is a feature which will be found in the new JAliEn Grid middleware. These containers used for payload isolation will, unlike the VOBOXes, run on Singularity [6], a lightweight container platform optimized for high-performance computing. With features such as support for unprivileged containers, simple image distribution and emphasis on singular, potentially long-running tasks, the platform might be seen as another choice for running Grid services. However, the absence of a networking stack makes it unsuitable for running the services needed to provide a functioning VOBOX–one of the reasons why Docker was chosen for this specific purpose. Most of the network functionality required is nowadays built into the Linux kernel itself, but using this in conjunction with Singularity requires explicitly configuring networking for each container namespace–a process that must be repeated each time a container is restarted.

Singularity is expected to simplify network setup by introducing official support for networking in version 3.0, not yet released at the time of writing [19]. To avoid maintaining containers on two separate platforms, and to benefit from the more lightweight approach to

containers, several VOBOXes may be moved to Singularity when it supports proper networking. By applying the experiences gained through Docker, this ought to be straightforward.

References

[1] The ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST3 S08003, DOI: 10.1088/1748-0221/3/08/S08003

[2] The LHCb Collaboration, *The LHCb Detector at the CERN LHC*, JINST3 S08005, DOI: 10.1088/1748-0221/3/08/S08005

[3] J. Elmsheuser, L. Heinrich, G. Stewart and M. Vogel, *Using containers with ATLAS offline software*, ACAT 2017. URL: https://cds.cern.ch/record/2279133 (Accessed 12.10.2018)

[4] Andrew McNab, *LHCb container status*, Container WG, CERN, Dec 2017. URL: https://indico.cern.ch/event/684575/contributions/2813991/ (Accessed 12.10.2018)

[5] The CMS Collaboration, *The CMS experiment at the CERN LHC*, JINST 3 S08004, DOI: 10.1088/1748-0221/3/08/S08004

[6] G. M. Kurtzer, V. Sochat, *Singularity: Scientific containers for mobility of compute*, PLoS ONE 12(5): e0177459, DOI: 10.1371/journal.pone.0177459.

[7] Brian Bockelman, *Moving CMS to a Container-based Infrastructure*, MAGIC Meeting, April 2018. URL: https://www.nitrd.gov/nitrdgroups/images/c/c6/CMS_Containers_04042018.pdf (Accessed 12.10.2018)

[8] ALICE Collaboration, *The ALICE Experiment at the CERN LHC*, JINST3 S08002, DOI: 10.1088/1748-0221/3/08/S08002

[9] WLCG VOBOX: https://twiki.cern.ch/twiki/bin/view/LCG/WLCGvoboxDeployment (Accessed 12.10.2018)

[10] A.G Grigoras, C. Grigoras, M.M Pedreira, P. Saiz, S. Schreiner, *JAliEn – A new interface between the AliEn jobs and the central services*, Journal of Physics: Conference Series 523, DOI:10.1088/1742-6596/523/1/012010

[11] M.M Pedreira, *JAliEn: the new ALICE high-performance and high-scalability Grid framework*, these proceedings.

[12] Docker: https://www.docker.com/ (Accessed 12.10.2018)

[13] Docker MACVLAN: https://docs.docker.com/v17.09/engine/userguide/networking/get-started-macvlan/ (Accessed 12.10.2018)

[14] J. Blomer, P. Buncic, R. Meusel, *The CernVM File System*, CERN Technical Report, 2013. URL: http://jblomer.web.cern.ch/jblomer/cvmfstech-2.1-0.pdf (Accessed 12.10.2018)

[15] Docker Live Restore: https://docs.docker.com/config/containers/live-restore/ (Accessed 12.10.2018)

[16] Docker Swarm: https://docs.docker.com/engine/swarm/ (Accessed 12.10.2018)

[17] Kubernetes: https://kubernetes.io/ (Accessed 12.10.2018)

[18] B. Ruan, H. Huang, S. Wu and H. Jin, *A Performance Study of Containers in Cloud Environment*, APSCC 2016 Proceedings (pp.343-356), DOI: 10.1007/978-3-319-49178-3_27

[19] Keith Cunningham, *A glimpse into Singularity 3.0*, Syslabs, 2018. URL: https://www.sylabs.io/2018/08/a-glimpse-into-singularity-3-0/ (Accessed 12.10.2018)