A container model for resource provision at a WLCG Tier-2

IOP Conf. Series: Journal of Physics: Conf. Series **1085** (2018) 032026 doi:10.1088/1742-6596/1085/3/032026

A container model for resource provision at a WLCG Tier-2

G Roy[1], G Stewart[1], D Crooks[1], S C Skipsey[1] and D Britton[1]

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

E-mail: gareth.roy@glasgow.ac.uk

Abstract. Containers are more and more becoming prevalent in Industry as the standard method of software deployment. They have many benefits for shipping software by encapsulating dependencies and turning complex software deployments into single portable units. Similar to Virtual Machines, but with a lower overall resource requirement, greater flexibility and more transparency they are a compelling choice for software deployment. The use of containers is becoming attractive to WLCG experiments as a means to encapsulate their payloads, ensure that userland environments are consistent and to segregate running jobs from one another to improve isolation. Technologies such as Docker and Singularity are already being used and tested by larger WLCG experiments along with CERN IT.

Our purpose in this paper is to explore the use of containers at a medium to large WLCG Tier-2 as a method of reducing the manpower required to run such a site. By examining the requirements of WLCG payloads (such as the availability of CVMFS, Trust Anchors or VOMS information) a model of a contained compute platform is developed and presented. Along with providing compute it standardised monitoring solutions can be bundled to provide a complete toolbox for local System Administrators to provide resources quickly and securely.

1. Introduction

The use of linux containers has become prevalent throughout Industry with a large an varied ecosystem of tools developing and maturing. Containers use namespaces, a native Linux kernel feature, to isolate running processes from one another in their own wrapped up environment. The attraction of containers derives from the ability to wrap complex software packages, along with their many and varied dependancies, into a single unit which can be deployed and configured via simple tools.

The ability to wrap up complex software projects into a single easily deployable unit is a compelling model for running LHC experiment application payloads. Due to a reduced funding many of the Worldwide LHC Compute Grid (WLCG) sites are operating with limited manpower but are required to maintain large and complex stacks of middleware. There is much ongoing effort to investigate novel ways to simplify these systems and still support the computational requirements of the LHC experiments, containers may be one solution to this problem.

UKI-SCOTGRID-GLASGOW [2] (ScotGrid) has deployed some of these novel solutions in an attempt to simplify it's operations. ScotGrid it is a medium sized Tier-2 facility consisting of approximately 64000 HEPSPEC and 3.8 PB of Storage. It primarily supports ATLAS and LHCb, while acting as a CMS Tier-3 along with supporting other smaller experiments and local

user groups. At present ScotGrid currently provides 25% of it's computational resources as VAC. VAC is a VM lifecycle manager that implements the vacuum model on a group of autonomous worker nodes [5].

While VAC is simple to deploy, lightweight and auto-updates without the interaction of a system administrator the use of VMs leads to issues for the local site. As the VMs contain a complete running version of Linux they can be slow to start, and use a large amount of resources over and above that required by the running payload. They are also opaque to the local site monitoring making it difficult to get an accurate representation of what is happening on site resources.

Our goal is to take the idea of autonomous worker node from VAC and combine that with a standard site configuration (in the case of ScotGrid an HTCondor batch system and ARC CEs) through the use of containers. To accomplish this we use standard software tools (in this case Docker [1]) to develop a container that is capable of running WLCG application payloads, focusing in this paper on ATLAS multi-core payloads. Along with creating the container a set of monitoring tools and methods of deploying the container into production will be assessed.

2. Requirements

Before we can create a container image to run WLCG workloads it is essential to identify the requirements that such payloads would need. At present most workloads running on the WLCG infrastructure assume a Redhat based operating system such as Scientific Linux or Centos.

Along with this most CERN experimental software, or experiment workload management systems would expect a Grid site to provide:

- Access to experiment software and other WLCG packages usually via the Cern Virtual Machine Filesystem (CVMFS)[3].
- Access to EGI and Grid toolkits, libraries and software to allow interaction with Grid systems for file transfer, authentication etc.
- Availability of a common set of libraries and a known good set of packages.
- Up to date trust anchors and VO information to allow authentication and authorisation.

Additionally, to properly manage containers in production and reduce the overall workload of site administrators (which is especially important with the reduction in manpower available to many Grid sites due to flat-cash funding scenarios) the following is required:

- Lightweight resource provision, which is simple to deploy and makes best use of available hardware resources.
- Leverage existing infrastructure, and allow a staged migration to more novel methods of deployment.
- A complete set of monitoring tools that help to quickly identify problems and easily drill down between layers of information.
- Use standard and supported tools which allows access to a community of expertise to round out administrators knowledge.

3. Container Components

3.1. CVMFS

As mentioned in our statement of requirements gaining access to experiment software in most case requires the use of CVMFS. CVMFS is a fuse mounted caching filesystem that allows lazy access to software via a series of hierarchical squid proxies. In order to access the files stored within CVMFS from within a container we can either a) bind mount the CVMFS directory from

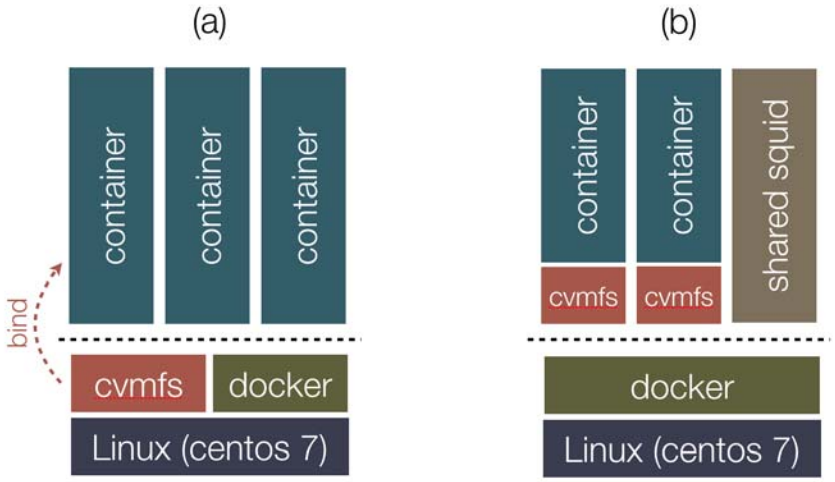


Figure 1: Methods of accessing CVMFS from with a container, either a) bind mounting from the host or b) via fuse within the container itself.

the container host to a directory within the container or b) use fuse from within the container itself to mount CVMFS. This is illustrated in Figure 1.

Each method of accessing CVMFS comes with trade-offs. Bind mounting CVMFS from the host requires that all required repositories are statically mounted via entries in the hosts `/etc/fstab`. This can in some cases cause systems to have difficulty in booting if the CVMFS directories are not available at startup. Bind mounting does allow a shared cache between all containers which improves performance and comes with a simple set of semantics in how volumes are added to the container.

Fuse mounting CVMFS within the container also has it's trade-offs, it requires `autofs` to be installed within the container and running as a service. As each container runs it's own instance of CVMFS there is no shared cache which reduces efficiency (this can be improved by including a shared squid on each container host as shown in Figure 1). Finally, the container requires escalated privileges specifically adding the MKNOD and SYS_ADMIN capabilities as shown below:

```
--cap-add MKNOD --cap-add SYS_ADMIN --device /dev/fuse.
```

The benefits of running CVMFS within the container itself are that repositories are no longer statically mounted and do not need to be known before the container is started, CVMFS is no longer required on the host so read-only linux distributions (such as coreos, rancheros) could be used to increase security.

In the remainder of this paper we use the method of bind mounting CVMFS from the container host as all repositories for running ATLAS multi-core production payloads are known before instantiation.

3.2. Middleware

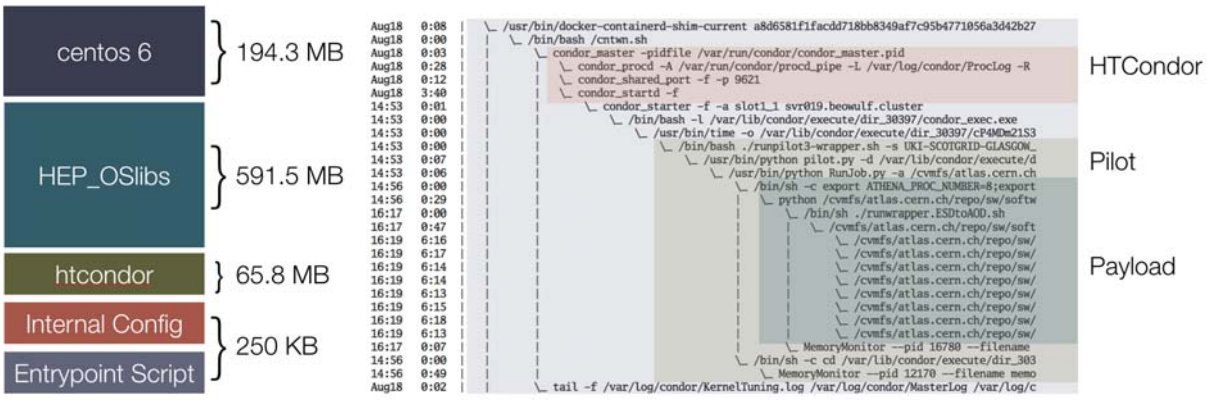
For WLCG payloads to be able to perform authentication checks, stage data and communicate with Grid workload management servers it is essential to have the correct middleware available. In most cases these packages are installed from the EGI provided Unified Middleware Distribution (UMD) which contains meta-packages for worker node installations. However, to

reduce the container size, these packages are not installed, instead a method similar to the one employed by ATLAS VAC virtual machines is used.

The grid environment is loaded from a CVMFS mounted "tarball" distribution available in the `grid.cern.ch` repository. In practice the grid environment is loaded via HTCondor at job runtime with a custom user-job-wrapper script that sets up the appropriate environment variables. This means that the Grid middleware packages are decoupled from the container image, reducing it's size. A second benefit of this method is it grants access to all of the required CA certificates and CRLs, again via CVMFS. These are updated regularly meaning they do not need to be installed within the container, nor does a CRL fetching service need to be included within the container image.

3.3. HEP Libraries

Each of the LHC experiments has a requirements for additional software packages to be available on a Grid worker node to allow proper execution of application payloads. All of the packages required that are not found in a bare installation of Scientific Linux or Centos are described by the meta-package, `HEP_OSlibs`. The contents of this meta-package are decided by representatives of the four LHC experiments via the Librarian and Integrators Meeting (LIM), the Architects Forum (AF) and WLCG Operations and Coordination meetings. To ensure that WLCG payloads correctly operate with our container environment we install the HEP_OSlibs within the container. Unfortunately this increases the size of the container by approximately 600MB, it is hoped that this dependancy can be removed with the move to Singularity [6] as an encapsulation mechanism of WLCG payloads.



(a) Layers and size of final container image

(b) Process tree of container running an ATLAS multi-core payload

Figure 2: Details of final container layers, and example of running container.

3.4. Final Container

The different layers found in our final container is shown in Figure 2(a). The total size of this container is 851.9MB and is based initially on Centos 6 for backwards compatibility. As mentioned in our previous sections on requirements it includes `HEP_OSlibs` as a common baseline for WLCG experiment code.

Along with local site configuration, it contains HTCondor client packages to connect to ScotGrid's batch farm. The container is configured with standard pool accounts and has CGROUPS disabled as all resource utilisation is restricted at the container level. HTCondor

is also configured to use the Condor Connection Broker (CCB) to allow communication to key condor processes as container network access is restricted to the internal Docker network. The container itself is distributed to the hosts via a private Docker registry running on our head node, with each node in our test pool spawning one container per eight CPU cores.

An example ATLAS multi-core payload is shown in Figure 2(b), the process tree contains the HTCondor `startd` process, the ATLAS pilot and the multi-core payload running eight `athena.py` processes. Unlike a VM it is possible to observe these processes using `ps` from the container host making monitoring of payloads simple, this aids observability and traceability of any errant action.

4. Monitoring and Logging

Local monitoring is essential to enable sites to identify issues and optimise workloads, in this section we identify standard tools which can be deployed as containers that allow the gathering, aggregation and alerting on those metrics as well as capturing logging information for each running container.

4.1. Metrics

Gathering instantaneous metric for each running container is important to identify bottlenecks or issues with job performance. For metric generation we use the Google sponsored cAdvisor [7]. cAdvisor can be run as a container along with our containerised worker nodes. It provides process lists, cpu utilisation per core, memory utilisation and network usage for each running container. Figure 3 illustrates some of the information obtained from a running instance of an ATLAS multi-core production payload. The cAdvisor library is also integrated within Kubernetes which mean when moving to this as a deployment strategy low level metric monitoring remains the same.

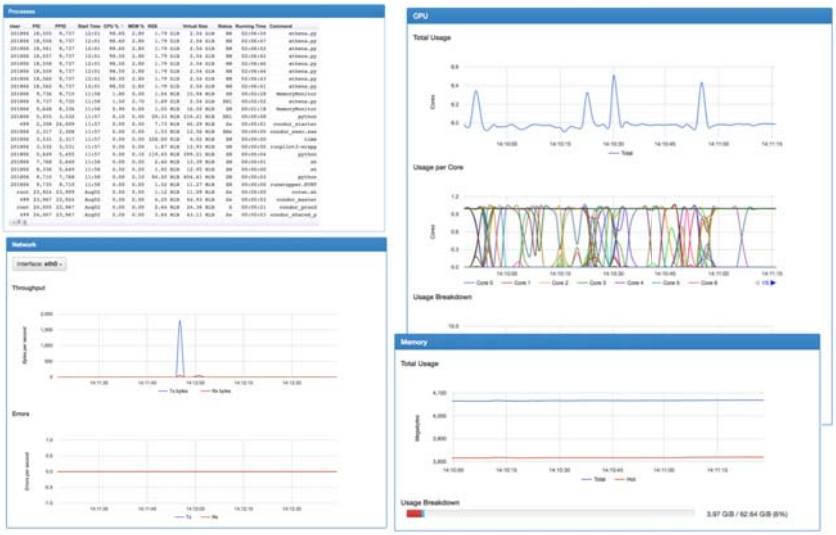
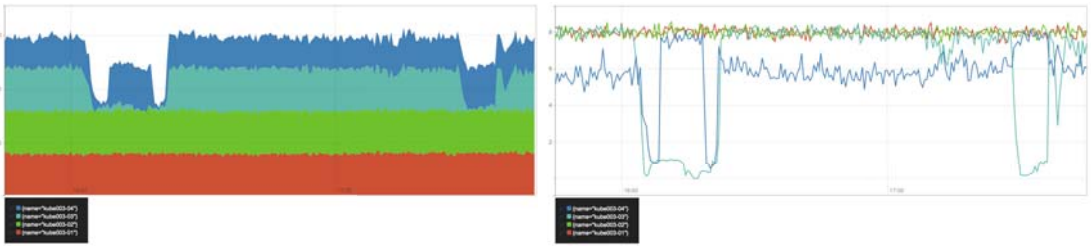


Figure 3: Graphical display of metrics gathered and presented by cAdvisor.

4.2. Aggregation

As well as interactively and graphically presenting system usage cAdvisor also exports it's metrics in a format that can be read by another software package, Prometheus [8]. Prometheus acts as an aggregation platform to collect metrics from multiple sources, store and aggregate them. It

allows querying, plotting and alerting based on the information it finds. Prometheus uses a pull model for gathering metrics accessing http endpoints located at `/metrics` which has a number of benefits. The pull model allows Prometheus to scale by adding additional collectors to any size required and it gives a heartbeat for each container host to ensure the host is alive. An example of the metrics and graphing available via Prometheus is shown in Figure 4 where the CPU load is shown for four containers each running ATLAS multi-core production pilots.



(a) CPU load stacked graph (b) CPU load

Figure 4: Plots showing the CPU load (a) Stacked and (b) Overlayed of one container host running four ATLAS mult-core containers.

4.3. Logging

As well as gathering metrics it is also important to gather logging information to diagnose problems or detect issues. As with cAdvisor a solution that can be deployed as a container would allow management in a similar fashion to the containerised worker nodes.

Using a combination of Logspout [9] and Oklog [10] we can export logging information from each running container, aggregate and search them. The configuration is shown in Figure 5. Logspout acts to collect logs and ships them to an Oklog instance which can be used to search via simple regex. Logspout is capable of sending logs to a number of different endpoints so as the number of logs scale Oklog could be replaced with an Elasticsearch service for better querying and alerting.

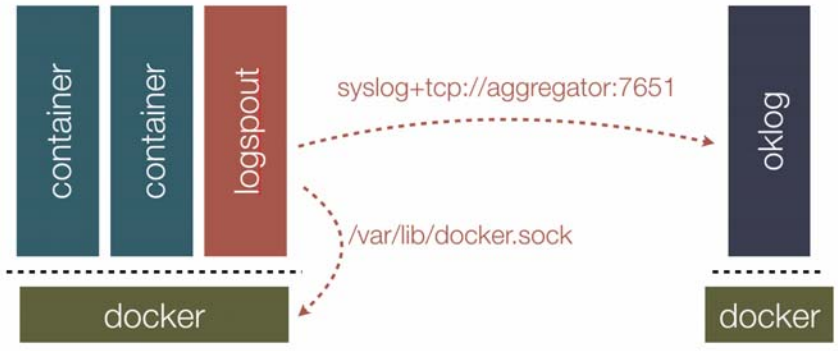


Figure 5: Logging configuration using container deployed logspout for collecting loging information and oklog for log aggregation.

5. Conclusions

In this paper we have examined the requirements of container to run WLCG application payloads. A nominal container model has been created and proven to work by running ATLAS muti-core payloads. A set of tools has been selected to allow sites to integrate monitoring and logging via containers as with the worker nodes.

In future work it is hoped to extend the worker node container to auto-expire and couple with a CI/CD pipeline to automate updates to containers as well as Integrate with Kubernetes (or other mechanism) to get auto-container redeploys (via services).

References

[1] *Docker.com*, "Docker" [software], version 1.12.6, 2017. Available from https://www.docker.com [accessed 18 Oct 2017]
[2] *ScotGrid* [online] Available at http://www.scotgrid.ac.uk/ [accessed 14 May 2015]
[3] *J Blomer et al.*; 2011 J. Phys.: Conf. Ser. 331 042003, "Distributing LHC application software and conditions databases using the CernVM file system"
[4] *docker-volume-cvmfs*, "docker-volume-cvmfs" [software], version v0.3.1, 2017. Available from https://gitlab.cern.ch/cloud-infrastructure/docker-volume-cvmfs/ [accessed 18 Oct 2017]
[5] *A McNab*; "Running jobs in the vacuum" (A McNab et al 2014 J. Phys.: Conf. Ser. 513 032065)
[6] *Singularity*, "singularity" [software], version 2.3.2-dist, 2017. Available from http://singularity.lbl.gov/ [accessed 18 Oct 2017]
[7] *cAdvisor*, "cadvisor" [software], version 0.26.3, 2017. Available from https://github.com/google/cadvisor [accessed 18 Oct 2017]
[8] *Prometheus*, "prometheus" [software], version 1.7.1, 2017. Available from https://prometheus.io/ [accessed 18 Oct 2017]
[9] *Logspout*, "logspout" [software], version 3.2.3, 2017. Available from https://github.com/gliderlabs/logspout [accessed 18 Oct 2017]
[10] *oklog*, "oklog" [software], version 0.2.2, 2017. Available from https://github.com/oklog/oklog [accessed 18 Oct 2017]