Using Kubernetes as an ATLAS computing site

Fernando Harald Barreiro Megino[1,*], *Jeffrey Ryan* Albert[2], *Frank* Berghaus[2], *Kaushik De*[1], *FaHui* Lin[1], *Danika* MacDonell[2], *Tadashi* Maeno[3], *Ricardo Brito* Da Rocha[4], *Rolf* Seuster[2], *Ryan Paul* Taylor[2], and *Ming-Jyuan* Yang[5] on behalf of the ATLAS collaboration

[1]University of Texas at Arlington, United States of America
[2]University of Victoria, Canada
[3]Brookhaven National Laboratory, United States of America
[4]European Organization for Nuclear Research, Switzerland
[5]Academia Sinica, Taiwan

Abstract. In recent years containerization has revolutionized cloud environments, providing a secure, lightweight, standardized way to package and execute software. Solutions such as Kubernetes enable orchestration of containers in a cluster, including for the purpose of job scheduling. Kubernetes is becoming a de facto standard, available at all major cloud computing providers, and is gaining increased attention from some WLCG sites. In particular, CERN IT has integrated Kubernetes into their cloud infrastructure by providing an interface to instantly create Kubernetes clusters, and the University of Victoria is pursuing an infrastructure-as-code approach to deploying Kubernetes as a flexible and resilient platform for running services and delivering resources.

ATLAS has partnered with CERN IT and the University of Victoria to explore and demonstrate the feasibility of running an ATLAS computing site directly on Kubernetes, replacing all grid computing services. We have interfaced ATLAS' workload submission engine PanDA with Kubernetes, to directly submit and monitor the status of containerized jobs. We describe the integration and deployment details, and focus on the lessons learned from running a wide variety of ATLAS production payloads on Kubernetes using clusters of several thousand cores at CERN and the Tier 2 computing site in Victoria.

1 Introduction

The ATLAS experiment [1] has a long history of exploiting and adapting to new technologies in cloud computing [2][3][4][5]. Containerization and Kubernetes are a continuation of this trend in a new paradigm. The idea to use Kubernetes as a batch system was conceived during the ATLAS-Google Data Ocean project [6], in which ATLAS transferred data to Google Cloud Storage and then processed it using Google Compute Engine virtual machines, according to the traditional Infrastructure-as-a-Service paradigm. However, containers are a lightweight and more performant alternative to virtual machines, and Kubernetes provides a robust, feature-rich and fully declarative platform for

* Corresponding author: barreiro [at] uta [dot] edu

automation and orchestration of containerized applications on any infrastructure. Furthermore, Kubernetes could be an open-source replacement for traditional WLCG [7] batch services that is widely adopted across the industry, available on all major cloud providers, and provides a common interface across cloud providers and sites. The software stack required at traditional WLCG sites could be considerably simplified by removing (or encapsulating in containers) HEP-specific middleware. Moreover, since ATLAS has already migrated to a fully container-based workload [8], it is simpler and more natural to provide container-native infrastructure at sites, which can further obviate other setup steps and enables new initiatives such as analysis preservation.

We have explored linking PanDA [9][10], ATLAS' Workload Management System (WMS), with Kubernetes through the Harvester [11] resource interface. This contribution will describe our initial integration model, ideas for improvement, and experience gained using Kubernetes clusters at CERN and the University of Victoria (UVic).

2 Site perspective: Kubernetes installation

There are many methods and tools for provisioning Kubernetes clusters; a suitable one must be chosen based on the circumstances and infrastructure at the site. At CERN, computing resources are managed and provided primarily via one large Openstack cloud, and the deployment of Kubernetes clusters as a service is integrated into the cloud with Magnum [12]. At UVic, where a variety of resources are available including bare metal nodes and several institutional clouds, Kubespray [13] was selected as a portable and flexible solution for deploying Kubernetes clusters based on the Infrastructure-as-Code paradigm. Following this approach, a cluster can be deployed or fully rebuilt from scratch within about 15 minutes.

ATLAS computing jobs rely on CVMFS [14], a distributed read-only file system widely used in HEP, to access the experiment software. Two approaches have been used to integrate CVMFS into Kubernetes clusters: at the host level underneath Kubernetes, and at the application layer inside the cluster. In the host-based approach, a standard CVMFS client is installed on each node in the cluster, and pods can mount CVMFS repositories using a hostPath [15] volume. To ensure security, we enable the Kubernetes PodSecurityPolicy admission controller and enforce a policy that whitelists the path /cvmfs for read-only access. This approach avoids containerizing the CVMFS client and is suitable if CVMFS is deemed to be essential infrastructure for a dedicated, purpose-built Kubernetes cluster for HEP applications, as at UVic. On the other hand, at CERN the csi-cvmfs driver [16] is used to provide pods with access to CVMFS, leveraging the Kubernetes Container Storage Interface. This method contains CVMFS within the application layer of the cluster, and is suitable on shared or general-purpose clusters or commercial clouds, where it is not desirable or possible to modify the nodes in the cluster with installation of domain-specific software such as CVMFS.

CVMFS can also be used to optimize the distribution of container images. Using standard container runtimes, the initial startup time of a typical container is dominated by pulling the image, even though only about 6% of the image data is actually read [17]. Containers that provide complex scientific software stacks like ATLAS' are often significantly larger ($\gtrsim$1 GB), exacerbating the inefficiency of this approach. At UVic, we integrated the CVMFS Docker Graph Driver plugin [18][19] into the Kubernetes cluster, so that Docker loads image data on demand instead of pulling the entire image before starting a container. In addition to accelerating initial container startup time by a factor of 4, this saves significant amounts of bandwidth and removes a bottleneck for rapid and dynamic scalability of the cluster, and integrates well with the host-based deployment of CVMFS.

To keep the cluster secure, at UVic we configured Role Based Access Control to allow only those API actions which are suitable and necessary to run jobs on the cluster, and applied another PodSecurityPolicy to forbid privileged pods. To monitor the status and resource usage of the cluster, we use Prometheus [20], as shown in Figure 1.

Fig. 1. Prometheus monitoring showing resource usage for one of the Kubernetes clusters

3 Central perspective: integration with the Workload Management System through Harvester

Harvester is the new resource-facing service developed by the PanDA team to interface the central workload management system with distributed computing resources. It is flexible and extensible, capable of exploiting HPCs, grid sites and clouds. Using new plugins developed by our ASGC colleagues, Harvester can now also make use of Kubernetes clusters.

The integration between Harvester and Kubernetes is illustrated in Figure 2. Harvester uses the Kubernetes *Job* resource type to control pods. A *Job* is an abstraction of a pod that is used to run non-persistent workloads, and ensures that a pod terminates successfully. This relieves Harvester from having to track the status of individual pods and resubmit ones that fail if e.g. a node experiences a problem. Harvester defines the specifications (e.g. memory, CPU) for each *Job*, and Kubernetes schedules pods on nodes where the required resources are available. The Harvester credential manager plugin regularly places fresh X509 proxies in the Kubernetes cluster as secrets [21], which Kubernetes then makes available to the pods so that pilot jobs can authenticate to grid services. We use the standard ADC CentOS 7 container image available on Dockerhub [22]. Currently the container image is statically defined per queue, but we plan to improve this so that it can be dynamically defined on a per-job basis. Lastly, the pod runs a startup script and launches the pilot, and the pilot retrieves a job from PanDA and executes it.

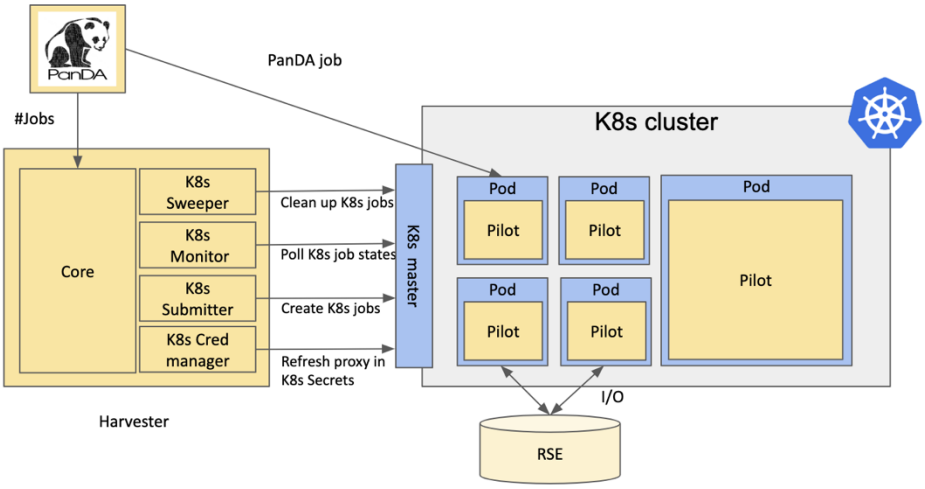

Fig. 2. Harvester-Kubernetes integration schematic.

4 Results

We have integrated Harvester with several Kubernetes clusters, most notably at CERN and UVic.

4.1 CERN

The first large-scale test was conducted at CERN in early 2019 on temporarily available resources. We created a 2000-core cluster consisting of 8-core nodes, and executed 1-core and 8-core ATLAS production jobs on the cluster. The main findings are depicted in Figure 3. Initially, we used the default Kubernetes scheduling algorithm, which balances load across nodes in a round-robin manner. This resulted in 1-core jobs being spread among nodes, blocking 8-core jobs from running on those nodes, thus decreasing the overall utilization of the cluster as more 1-core jobs run (see red ellipses in top image of Figure 3). To address this, we tuned the scheduling policy to pack the nodes, preferentially placing 1-core jobs on nodes that are already partially full rather than on empty nodes. This improved the resource usage of the cluster: there is only one small inefficiency when some nodes were draining 1-core jobs in order to start 8-core jobs (see grey ellipse in bottom image of Figure 3), but this is fundamentally unavoidable in any batch system. We are currently planning to repeat this exercise and potentially set up a permanent cluster at CERN.

Fig. 3. CPU usage on the CERN Kubernetes cluster with the default scheduling algorithm (top) vs the node-packing policy (bottom). Resources that are unused due to inefficient scheduling are indicated with red ellipses. Also note the cluster size decreased from 2000 cores to 1000 cores, due to issues with the csi-cvmfs driver and the infrastructure at CERN, which were addressed at the time.

4.2 University of Victoria

The second exercise was run on clusters at UVic. In order to evaluate and test several aspects of Kubernetes configuration, a small development cluster of 130 cores was deployed early on. The observations on this test cluster regarding Kubernetes scheduling were the same as at CERN (see Figure 4). The insights and solutions gained from the test cluster influenced the deployment of a larger, permanent T2 cluster with security and performance enhancements, which has been running stably without interventions since November 2019 (see Figure 5). 94% of wallclock time on this cluster is consumed by successful jobs, which will improve further once we address some known issues.

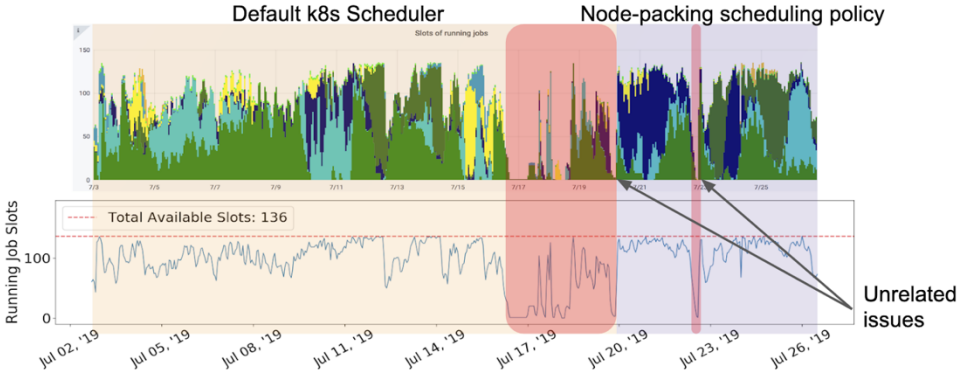

Fig. 4. Default Kubernetes scheduling vs node packing scheduling on the UVic test cluster

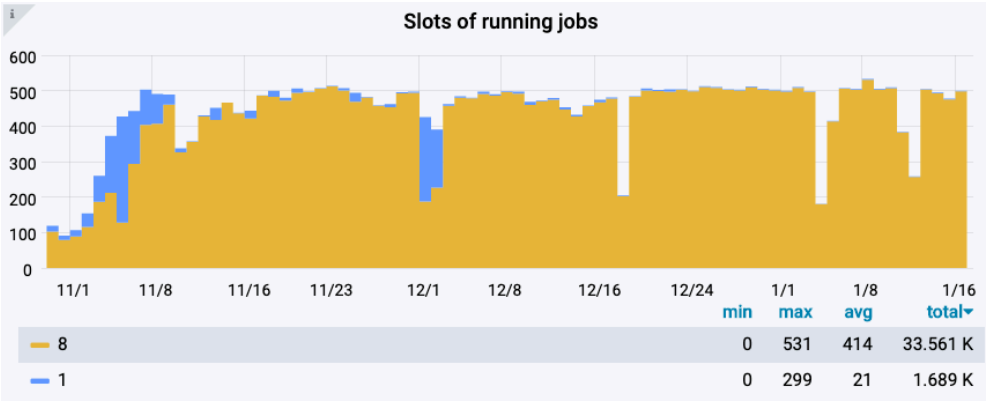

Fig. 5. Stable operations of the 500-core UVic production cluster CA-VICTORIA-K8S-T2. The abrupt brief dips in usage are due to scheduled downtimes for network maintenance external to the site

4 Conclusions and future work

Kubernetes is the de facto standard for container orchestration, and many WLCG sites have expressed interest in using it. Kubernetes is usually used to manage applications and services, but can also be used directly as a batch system. We have demonstrated the use of Kubernetes, integrated with Harvester, as a native batch cluster for ATLAS production jobs at multiple sites, thereby significantly reducing the number of layers in the software and service stack of a traditional WLCG computing site.

The scale we have demonstrated so far is still smaller than ATLAS' typical grid usage, and some optimizations still need to be implemented. We are also interested in evaluating commercial cloud providers, and we have explored cluster federation, but have not yet found a satisfactory solution for this.

Furthermore, while this approach is very promising, we note that significant effort remains ahead to develop the maturity of the platform for our needs. The WLCG is a highly specialized infrastructure that has evolved over the last 15 years. Many implementation details have yet to be explored, such as accounting, traceability, fairshare-based scheduling, scaling the authentication and authorization model for wider use, and further adapting the workload and workflow to the container paradigm. More investigation, development and

evolution will be needed from the WLCG community to fully benefit from container-native computing in the Kubernetes ecosystem.

Acknowledgements

This research was enabled in part by support provided by Compute Canada (www.computecanada.ca), National Science Foundation (www.nsf.gov), US Department of Energy (www.energy.gov/) and WestGrid (westgrid.ca).

References

1. The ATLAS Collaboration, J. Inst. **3** S08003 (2008) doi:10.1088/1748-0221/3/08/S08003
2. F. Barreiro Megino et al., J. Phys. Conf. Ser. **396** 032011 (2012) doi:10.1088/1742-6596/396/3/032011
3. S. Panitkin et al., J. Phys. Conf. Ser. **513** 062037 (2014) doi:10.1088/1742-6596/513/6/062037
4. R. P. Taylor et al., J. Phys. Conf. Ser. **664** 022038 (2015) doi:10.1088/1742-6596/664/2/022038
5. R. P. Taylor et al., J. Phys. Conf. Ser. **898** 052008 (2017) doi:10.1088/1742-6596/898/5/052008
6. M. Barisits et al. on behalf of the ATLAS Collaboration, EPJ Web Conf. **214** 04020 (2019) doi:10.1051/epjconf/201921404020
7. LHC Computing Grid: Technical Design Report, document LCG-TDR-001, CERN-LHCC-2005-024 (The LCG TDR Editorial Board) (2005)
8. A. Forti et al., Containers usage on the ATLAS grid infrastructure, 23rd International Conference on Computing in High Energy and Nuclear Physics 2018, Sofia, Bulgaria
9. P. Nilsson et al. on behalf of the ATLAS Collaboration, J. Phys. Conf. Ser. **513** 032071 (2014) doi:10.1088/1742-6596/513/3/032071
10. T. Maeno et al., J. Phys. Conf. Ser. **898** 052002 (2017)
11. T. Maeno et al., EPJ Web Conf., **214** (2019) 03030
12. OpenStack Magnum https://docs.openstack.org/magnum/latest/
13. Kubespray https://kubespray.io/
14. J. Blomer et al., J. Phys.: Conf. Ser. **898** 062031 (2017)
15. Kubernetes hostPath volumes https://kubernetes.io/docs/concepts/storage/volumes/#hostpath
16. Kubernetes csi-cvmfs driver https://clouddocs.web.cern.ch/containers/tutorials/cvmfs.html
17. T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, 14th USENIX Conference on File and Storage Technologies 194430 (2016)
18. N. Hardi, J. Blomer, G. Ganis, R. Popescu, J. Phys. Conf. Ser. **1085** 032019 (2018) doi:10.1088/1742-6596/1085/3/032019
19. S. Mosciatti (2018) *Efficient container distribution at global scale*. Polytechnic University of Milan, Milan, Italy. https://hdl.handle.net/10589/144807
20. Prometheus https://prometheus.io/

21. Kubernetes Secrets https://kubernetes.io/docs/concepts/configuration/secret/
22. ATLAS Distributed Computing CentOS7 image
https://hub.docker.com/r/atlasadc/atlas-grid-centos7