ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

CONTAINERIZED CONTROL STRUCTURE FOR ACCELERATORS

I. Arredondo*, J. Jugo, University of the Basque Country, Leioa, Spain

Abstract

Nowadays, modern accelerators are starting to use virtualization to implement their control systems. Following this idea, one of the possibilities is to use containers. Containers are highly scalable, easy to produce/reproduce, easy to share, resilient, elastic and low cost in terms of computational resources. All of those are characteristics that fit with the necessities of a well defined and versatile control system. In this paper, a control structure based on this paradigm is discussed. Firstly, the technologies available for this task are briefly compared, starting from containerizing tools and following with the container orchestration technologies. As a result, Kubernetes and Docker are selected. Then, the basis of Kubernetes/Docker and how it fits into the control of an accelerator is stated. Following the control applications suitable to be containerized are analyzed: electronic log systems, archiving engines, middleware servers, etc. Finally, a particular structure for an accelerator based on EPICS as middleware is sketched.

INTRODUCTION

Virtualization technology[1] has became a standard in many fields related to the IT. Especially, at DevOps and Cloud services. This standardization and the benefits it provides has cause its adoption in other areas. In the large facilities field, there are being efforts on this direction. As an example Virtual Machines (VMs) are used at [1–3] for software standardization and maintaining. At [4–6] the authors add High Availability (HA) and resources optimization to the control system using also VMs. In conclusion, the use of virtualization provides benefits to the control system architecture. We can state that a good control system architecture for a large facility should be maintainable, robust, easy to scale and efficient:

- Maintainable. A good architecture should be easy to describe and deploy automatically. It needs to be based on a limited number of standards and reliable technologies. Those technologies should be proven and tested, supported and maintained by a vendor or a community or both, being positive if they are easy to share. Moreover, the system must be prepared to fit new technological challenges.
- Robust. The control system must have an agile and secure response to fault events, including fault tolerance early error detection and fast recovery.
- Easy to scale. Adding new elements and plant changes to accomplish control system upgrades.
- Efficient. Use the resources efficiently and maintain low energy consumption.

Additionally, taking into account the very particular necessities for large scientific facilities, the use of open source technologies is very valuable.

Virtualization with VMs cover almost all of these characteristics, facilitating the maintenance of large control systems. However, as reported in [4, 6], they need a long period for recovery from serious faults as machine crash. Also, they do not use the machines resources in the most efficient way, because they need to implement an Hypervisor for every VM.

On the other hand, Container based virtualization covers the control system requirements, using the resources in a more efficient way. Moreover, the use of containers in combination with a container orchestration tool can overcome also the slow fault recovery. This work discusses the application of containerized solutions in accelerators, analyzing the existing solutions and a possible valid architecture.

It is worth stating that low level control system does not fit with the mainstream containerization philosophy. It is more focused on Cloud and microservices. However, there are efforts focused on that, as in [7].

CONTAINERIZATION

Containerization (or operating system-level virtualization) is a method of virtualization that enables a layer of isolation creating different user-spaces instances in the same kernel.

There are different container technologies: Docker, LXC, OpenVZ, BSD Jails, Solaris Zones, Turbo, etc[2]. Most of the large scientific facilities (LHC, ITER, SNS, CHSNS, KEK,...) relay on Linux as their main Operating System, so we will focus the discussion on Linux based containers.

In Linux, containerization is based on the creation of user-spaces instances, at the kernel level. It is done via two kernel components: *namespaces* and *cgroups*. Namespaces divide the operating system into virtual segments. While cgroups can apply constraints, or limitations, to system resources [8].

Among the containers based on Linux, the most popular are: Docker, Flockport (LXC) and Rocket (rkt).

In [9] the authors compare mainly Docker and LXC stating that Docker is better at computation time, memory throughput and network I/O performance. While LXC is better at Disk I/O performance.

Both of them report no over-heads on memory utilization or CPU, whilst I/O and operating system interactions incurred some ones. That means, as expected, that containers are a good lightweight option for virtualization.

* inigo.arredondo@ehu.eus

[1] Note that the scope of this paper evolves very rapidly, so a lot of information is not yet available through peer reviewed papers. Consequently, some of the references and the information is taken from specialized webpages.

[2] https://en.wikipedia.org/wiki/Operating-system-level_virtualization

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

On the other hand, both rkt and Docker evolve quickly and since 2017 both are Cloud Native Computing Foundation[3] incubation projects[4]. They have some technical differences that mainly concern to bootstrapping and security. Although there are opinions in favor of both of them, we have not found any peer reviewed reference comparing thoroughly the two technologies. It is worth mentioning that rkt engine can manage also Docker containers.

In this paper, our purpose is to focus, mainly, on the accelerators control services that are not a critical part of the operation. This means that we do not need a data fast processing. Consequently, the election of the technology will be lead by parameters like ease of utilization, support, active development, compatibility with orchestration tools, etc.

Based on these parameters, in our opinion the best option is Docker. Nowadays, it is the most used technology, it has a company and a very active community support, it is being released regularly and it is the base of the major container orchestration tools.

Containers vs VMs

As we stated at the introduction most of the efforts done towards control systems virtualization have been performed using VMs. However, comparing the performance between VMs and Containers , we can conclude that Containers fit better with the implementation of some function of a control system for a large facility,.

From [10–13], the main benefits of each solution can thus be summarized:

- Benefits of containers:
  - Fast application delivery: Maintenance and update.
  - Better scaling: In case of more resources need. About 22 times better than VMs.
  - More rapid spawning and termination.
  - Better resource utilization (lightweight). Higher workloads with grater density. Containers are able to run multiple isolated processes in a host without the overhead caused by the Hypervisor layers introduced by VMs.
- Benefits of VMs:
  - Better compatibility and isolation.
  - More robust that containers.

Most of the papers cited in the introduction state that one of the major benefits of VM virtualization is the optimization of resources. Using containers this will be considerably improved. The other benefits, such as scaling or HA still being achieved using containers.

On the other hand, one of the most important drawbacks of the VM virtualization is the Host crash. Containers are more fragile than VMs so a crash of any container can be fatal. However, this issue can be improved managing the system with a container orchestration tool. Thanks to the rapid spawning and ease of provisioning of containers, the orchestration tool can rebuild the system fast and automatically in any machine of the cluster.

Recently, containers are under use and experimentation in large scientific facilities. This fact can be observed in the Docker Hub[5], where the number of containers related with large facilities such as EPICS and other common tools is growing.

ORCHESTRATION

In a large scientific facility it is not enough with managing containers manually. When a wide range of containers needs to be run in several physical machines with the constraints imposed by a production environment, a container orchestration tool becomes essential.

A good container orchestration software needs to provide:

- Provisioning: It is able to provide and launch containers in an orderly way. This means to choose the appropriate node based on the available resources and user's restrictions.
- Configuration-as-text: It is possible to define the structure of the cluster using text files. This eases the edition, versioning and sharing.
- Monitoring: Within the framework it has a tool to check the health of the containers in the cluster.
- Rolling Upgrades and Rollback: It includes the opportunity of upgrading the system incrementally. So instead of changing the version of the whole system at once, they provide a way to automate the change step by step. This allows the maintainers to check the correct functioning of the new release. In case of a bad behavior, an automated rollback is also possible.
- Policies for Placement, Scalability etc. : As introduced in the provisioning feature, these tools fetch the optimal placement for the containers based on cluster metrics (load balancing). They also provide high availability tools and automatic scaling if unused resources are available. Note that this feature can solve the problems of robustness of the containers vs VMs in many applications due to the rapid spawning of containers.
- Service Discovery: Containers expose services through a network, hence the service discovery is one of the key features of these tools.

[3] https://www.cncf.io/

[4] https://thenewstack.io/separate-votes-cncf-adopts-dockers-containerd-coreos-rkt/

[5] https://hub.docker.com/

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

- Ease of Administration: The orchestration tools need to be integrated with the other systems of the IT infrastructure. Consequently the have to be easy to deploy, configure and maintain.

These features fit very well with the characteristics of a good control system summarized at the introduction of this paper.

Nowadays, the most relevant container orchestration systems are[6]:

- Kubernetes (k8s): Kubernetes is an orchestration system for Docker and Rocket containers. It automates the deployment, scaling and management based on user-defined parameters.
- Mesosphere Marathon: Marathon is a LXC and Docker container orchestration framework for Apache Mesos.[7]
- Docker Swarm: Docker Swarm provides native clustering for Docker containers.

A lot of reports can be found comparing these three options (mainly web). We can summarize that:

- Kubernetes: Very good and proven open source tool for medium-large clusters. It is being developed by a large and active community. It is managed by the Cloud Native Computing Foundation and has support of companies like Amazon, CoreOS, Mesosphere, Samsung, Microsoft, Red Hat, IBM, Intel, Oracle, Docker, Cisco, Google,...
- Swarm: It provides the best Docker compatibility and it is easy to use. It is preferred for no very large clusters. Also open source.
- Mesos: It is a well proven open source tool (Twitter, eBay, Netflix,...). Designed and tested for very large clusters.

Table 1 shows how high profile companies are interested in container orchestration tools. Particularly, all of them provide infrastructure for k8s.

Control systems of large scientific facilities will need to run robust medium-large scale clusters. Therefore, Kubernetes and Mesosphere are preferred.

In addition, particular needs of accelerators can differ from the mainstream use of the container orchestrators, thus specialized tools may be built. For this reason, we consider that Kubernetes is the best option due to the large and active community behind it.

[6] https://www.infoworld.com/article/3205304/containers/orchestration-tools-enable-companies-to-fully-exploit-linux-container-technology.html

https://www.sdxcentral.com/articles/news/kubernetes-swarm-and-mesos-central-in-container-orchestration-space/2017/07/

[7] It is possible to run Kubernetes on top of Mesos.

Table 1: Orchestration Tools Provided as a Service by Selected Companies

	k8s	Mesos	Swarm
Alibaba Cloud Container Service	✓		✓
Amazon EC2 Container Service	✓	✓	✓
Microsoft Azure Container Service	✓	✓	✓
Google Container Engine	✓		
IBM Bluemix Container Service	✓		

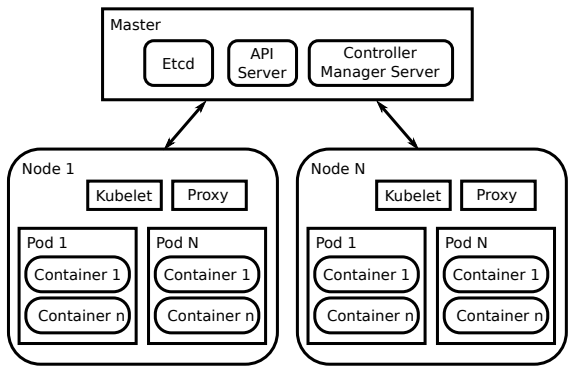

Figure 1: Kubernetes architecture.

Kubernetes

In this section we will describe very briefly the Kubernetes architecture [14] that is presented in Figure 1.

A Kubernetes cluster is composed by Masters and Nodes. The firsts ones run specific software to rule the others. The best way to understand their roles is describing the software they execute.

- Masters:
  - API Server: Both masters and nodes make API calls to perform their tasks. These are handled by the API server.
  - Etcd: It is a service to keep and replicate the configuration and run state of the cluster.
  - Scheduler and Controller Manager: They are in charge of scheduling containers onto nodes. They also ensure the amount of them that have to be running.
- Nodes:
  - Kubelet: A daemon that obey the instructions from the master. It includes, create, destroy and monitor the containers on the node.
  - Proxy: A network proxy to isolate the name of the service provided by the container from the IP address of this container.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

  - cAdvisor (optional): A daemon to provide statistics about the running containers.

Additionally, the management of the containers and the associated infrastructure is composed by the following items.

- Pods: The containers are packed into Pods. In this way related containers can share resources easily. Also, if some services are dependent to each other, it is possible to create/destroy all at once. This improve the management.
- Replication Controllers: One of the main characteristics of the orchestration tools is to maintain a user defined number of containers (Pods) running. In this manner, the high availability is granted. Kubernetes automatically create/destroy Pod replicas in case of failure. Besides, it generate the replicas taking into account the statistics of the nodes to improve the resources utilization of the cluster. All this is carried out by the replication controllers.
- Services: Any Pod can be created in any node of the cluster and have a self assigned IP. Therefore, we need a way to separate the service itself from the particular situation of the containers. This is performed by the services.
- Volumes: They are virtual file systems defined at the Pod level.
- Labels: They are a key/value pairs to name Kubernetes objects. The main objective is to be able to organize the cluster by using them.

Kubernetes includes all the features stated for a good orchestration tool. In consequence, by means of Kubernetes we can design a control system that fits with the specifications given in the introduction.

It is important to mention that in [14], the author report a de-registration / eviction / replacement / registration cycle of a Kubernetes Pod on the order of 300 milliseconds. And he states that the same process to replace a running VM instance behind a load balancer is almost always on the order of minutes.

This fact is very important to avoid one of the main drawbacks of using virtualization in the control systems. With Kubernetes if any machine crashes, their services will be automatically replicated to another machine within a few seconds.

CONTAINER BASED CONTROL ARCHITECTURE

As stated at the introduction this paper is focused on managing non critical services of the control system. These include alarm handlers, data archivers, electronic logs, LDAP services, slow non critical controls and user interface managers. Those services containerized using Docker/Kubernetes can be a useful alternative applicable in an EPICS based control architecture. The idea is to deploy the typical software tools used in an EPICS based implementation in a Kubernetes cluster.

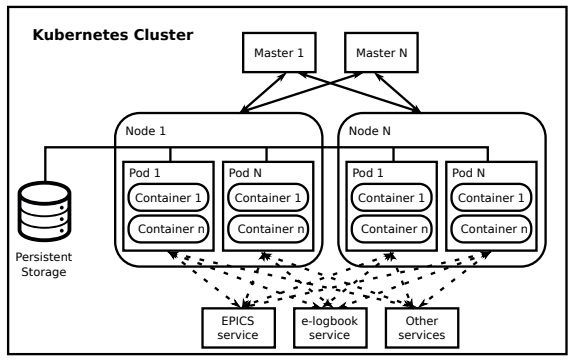

Figure 2: Kubernetes architecture for control system.

In Figure 2 we depict a very basic architecture of a Kubernetes cluster for an EPICS based control system. Depending on the scale it will be needed an specific amount of masters and nodes. Even tough it is always recommended to have enough of them to maintain HA and load balancing.

Kubernetes automatically manages the Pod location based on user parameters. Therefore, the service is not related with the node itself, but with the Pod. So, all the Pods, no matter where they are, have to be linked with the service they are providing. That is why in the Figure 2 the links between the Pods and the services are slashed lines.

Finally, a persistent storage will be added to store data, configuration files, etc. All the Pods will be linked with the data they need through a distributed file system as NFS.

In this way, the architecture is valid to implement different tools: an e-logbook, the EPICS Archiver Appliance, BEAST alarm handler, open LDAP, EPICS base and EPICS soft IOCs, among others. The last kind of In/Out Controllers (IOCs) can be called "virtual" IOCs.

In general, EPICS IOCs are out of the scope of a typical service which can be provided as a container, since IOCs are normally linked to some hardware, i.e. they manage "real" I/Os connected to physical elements. However, containerization of IOCs is very valuable for defining virtual IOCs, which can be useful for testing purposes, implementing IOC servers and IOCs connected with network devices as suggested in [6].

On the other hand, orchestration of those virtual IOCs will take into account that all the Process Variables (PVs) must be unique in the net. In consequence, it is not possible to have more than one replica of the Pod containing the same EPICS virtual IOC.

Other services as an e-logbook or open LDAP can be deployed as a common service.

Initial tests include the definition of EPICS base and EPICS virtual IOC services using minikube[8]. In the tests, a virtual IOC has been implemented inside a Pod and a service exposes it. The published PVs are accessible in an EPICS

[8] https://kubernetes.io/docs/getting-started-guides/minikube/

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

network as a normal IOC even if the Pod is restarted. We measured less than 10 seconds from an EPICS virtual IOC crash to its recover. That is, just monitoring one PV, removing the Pod which is deploying the virtual IOC, it restarts automatically and the connection is recovered again in less than 10 seconds.

Those good initial results encourages us to implement more complete tests, following the architecture shown in Figure 2.

CONCLUSIONS

Virtualization is nowadays one of the most useful tools in IT technologies. Particularly, in these days, the use of containers is growing rapidly. This technology mainly adds high availability, load balancing, deployment automation, resources use optimization, maintainability, scaling and sharing.

In this paper, the implementation of a control system for large facilities based on containerization is discussed. In order to select the proper tools, all the main container based virtualization options and container orchestrators are compared. The conclusion is that the binomial Docker/Kubernetes is nowadays the one that suits the best for a control system.

A control architecture that relies on EPICS is also sketched with promising results, including the implementation of containerized EPICS IOCs, which are useful as virtual IOCs

As future work we will intend to build a large Kubernetes deployment to fit better with the actual control system of a large facility. This goal will require to containerize several applications, figure out the scheme that suits best for EPICS maintainability and select a data adequate storage technology. Finally, high stress tests will need to be carried out.

ACKNOWLEDGEMENT

The authors are very grateful to the University of the Basque Country by the partial support of this work.

REFERENCES

[1] L. Fernández, R. Andersson, H. Hagenrud, T. Korhonen, R. Mudingay, and B. Zupanc, "How to Build and Maintain a Development Environment for the Development of Controls Software Applications: An Example of "Infrastructure as Code" within the Physics Accelerator Community." in *Proc. of International Particle Accelerator Conference (IPAC'16), Busan, Korea, May 8-13, 2016*, ser. International Particle Accelerator Conference, no. 7. Geneva, Switzerland: JACoW, June 2016, paper WEPOR050, pp. 2781–2783, doi:10.18429/JACoW-IPAC2016-WEPOR050. [Online]. Available: http://jacow.org/ipac2016/papers/wepor050.pdf

[2] T. Pal and T. Korhonen, "Configuration Management for Software and Firmware at PSI Accelerators," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), Kobe, Japan, 12-16 October 2009*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 9. Geneva, Switzerland: JACoW, Dec. 2009, paper TUP063, pp. 227–229. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tup063.pdf

[3] A. Adakin, D. Chubarov, V. Nikultsev, S. Belov, V. Kaplin, A. Sukharev, A. Zaytsev, N. Kuchin, S. Lomakin, and V. Kalyuzhny, "Virtualized High Performance Computing Infrastructure of Novosibirsk Scientific Center," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), Grenoble, France, 10-14 October 2011*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 9. Geneva, Switzerland: JACoW, Dec. 2011, paper WEBHAUST06, pp. 630–633. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/webhaust06.pdf

[4] D. Engel, R. Muller, P. Laux, P. Stange, and R. Fleischhaue, "Virtualization Infrastructure within the Controls Environment of the Light Sources at HZB," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), San Francisco, CA, USA, 7-11 October 2013*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 13. Geneva, Switzerland: JACoW, Dec. 2013, paper THPPC005, pp. 1100–1102. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/thppc005.pdf

[5] R. Kapeller, R. Krempaska, C. Higgs, and H. Lutz, "From Real to Virtual-How to Provide a High-Avaliblity Computer Server Infrastructure," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), San Francisco, CA, USA, 7-11 October 2013*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 13. Geneva, Switzerland: JACoW, Dec. 2013, paper THMIB03, pp. 1076–1078. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/THMIB03.pdf

[6] N. Kamikubota, S. Yamada, K. Sato, N. Yamamoto, T. Iitsuka, S. Motohashi, D. Takahashi, S. Yoshida, and H. Nemoto, "Experience of Virtual Machines in J-PARC MR Control," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), San Francisco, CA, USA, 7-11 October 2013*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 13. Geneva, Switzerland: JACoW, Dec. 2013, paper MOPPC131, pp. 417–419. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/moppc131.pdf

[7] T. Korhonen, "Modern System Architectures in Embedded Systems," in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), Grenoble, France, 10-14 October 2011*, ser. International Conference on Accelerator and Large Experimental Physics Control Systems, no. 9. Geneva, Switzerland: JACoW, Dec. 2011, paper THDAULT01, pp. 1260–1263. [Online]. Available: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/THDAULT01.pdf

[8] J. Schipp, "Chapter 1 - containers," in *Deploying Secure Containers for Training and Development*, J. Schipp, Ed. Syngress, 2016, pp. 1 – 11. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128047170000016

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA170

[9] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175 – 182, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16303041

[10] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*, March 2015, pp. 342–346.

[11] V. Medel, O. Rana, J. Á. Bañares, and U. Arronategui, "Modelling performance resource management in kubernetes," in *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, Dec 2016, pp. 257–262.

[12] C. de Alfonso, A. Calatrava, and G. Moltó, "Container-based virtual elastic clusters," *Journal of Systems and Software*, vol. 127, pp. 1 – 11, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217300146

[13] Z. Li, Y. Zhang, and Y. Liu, "Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications," *Tsinghua Science and Technology*, vol. 22, no. 01, pp. 1–9, February 2017.

[14] D. K. Rensin, *Kubernetes - Scheduling the Future at Cloud Scale*, 1005 Gravenstein Highway North Sebastopol, CA 95472, 2015. [Online]. Available: http://www.oreilly.com/webops-perf/free/kubernetes.csp