EPJ Web of Conferences **295**, 02031 (2024) https://doi.org/10.1051/epjconf/202429502031
CHEP 2023

Towards a container-based architecture for CMS data acquisition

Vassileios Amoiridis[1], *Ulf* Behrens[2], *Andrea* Bocci[1], *James* Branson[3], *Philipp* Brummer[1], *Eric* Cano[1], *Sergio* Cittolin[3], *Joao* Da Silva Almeida Da Quintanilha[1], *Georgiana-Lavinia* Darlea[4], *Christian* Deldicque[1], *Marc* Dobson[1], *Antonin* Dvorak[1], *Dominique* Gigi[1], *Frank* Glege[1], *Guillelmo* Gomez-Ceballos[4], *Patrycja* Gorniak[1], *Neven* Gutić[1], *Jeroen* Hegeman[1], *Guillermo* Izquierdo Moreno[1], *Thomas Owen* James[1], *Wassef* Karimeh[1], *Miltiadis* Kartalas[1], *Rafał Dominik* Krawczyk[2], *Wei* Li[2], *Kenneth* Long[4], *Frans* Meijers[1], *Emilio* Meschi[1], *Srećko* Morović[3], *Luciano* Orsini[1], *Christoph* Paus[4], *Andrea* Petrucci[3], *Marco* Pieri[3], *Dinyar Sebastian* Rabady[1], *Attila* Racz[1], *Theodoros* Rizopoulos[1], *Hannes* Sakulin[1], *Christoph* Schwick[1], *Dainius* Šimelevičius[1,5,*], *Polyneikis* Tzanis[1], *Cristina* Vazquez Velez[1], *Petr* Žejdl[1], *Yousen* Zhang[2], and *Dominika* Zogatova[1]

[1]CERN, Geneva, Switzerland
[2]Rice University, Houston, Texas, USA
[3]UCSD, San Diego, California, USA
[4]MIT, Cambridge, Massachusetts, USA
[5]Vilnius University, Vilnius, Lithuania

Abstract.
The CMS data acquisition (DAQ) is implemented as a service-oriented architecture where DAQ applications, as well as general applications such as monitoring and error reporting, are run as self-contained services. The task of deployment and operation of services is achieved by using several heterogeneous facilities, custom configuration data and scripts in several languages. In this work, we restructure the existing system into a homogeneous, scalable cloud architecture adopting a uniform paradigm, where all applications are orchestrated in a uniform environment with standardized facilities. In this new paradigm DAQ applications are organized as groups of containers and the required software is packaged into container images. Automation of all aspects of coordinating and managing containers is provided by the Kubernetes environment, where a set of physical and virtual machines is unified in a single pool of compute resources. We demonstrate that a container-based cloud architecture provides an across-the-board solution that can be applied for DAQ in CMS. We show strengths and advantages of running DAQ applications in a container infrastructure as compared to a traditional application model.

1 Introduction

The Compact Muon Solenoid (CMS) [1, 2] is one out of four large particle detectors (experiments) at the Large Hadron Collider (LHC) [3] at CERN in Geneva, Switzerland. As a versatile and advanced particle detector, CMS plays an important role in fundamental physics [4].

*e-mail: dainius.simelevicius@cern.ch

It allows to capture and analyze the intricate interactions between particles produced in high-energy collisions within the LHC. The Data Acquisition (DAQ) system is responsible for assembling, filtering, and storing the large influx of data generated by particle collisions within the detector [1]. The CMS DAQ system's main task is to read-out data from the different parts of the detector and assemble them into coherent structures corresponding to one bunch-crossing of the LHC (event building). These tasks require reliable monitoring and error reporting to guarantee the quality of the data collected. The system has been built upon a service-oriented architecture, where applications are encapsulated as self-contained services. This approach has enabled efficient deployment and operation of various applications, albeit using a mix of different tools, configurations, and scripting languages. In this context, two distinct strategies have emerged to manage the life cycle of services. Short-lived services, such as read-out and event building, are managed by a custom infrastructure [5]. On the other hand, long-running, auxiliary services are managed using systemd [6]. The main goal of this work is to transform the existing architecture into a homogeneous, scalable cloud system – a unified environment where all applications are managed uniformly with standardized facilities. In addition to introducing modern technologies, this work intends to demonstrate that a container-based cloud architecture can serve as a comprehensive solution for the CMS DAQ. This is a status report of work in progress since not all the goals have been achieved yet.

2 Motivation

After several years of successful operation of the CMS DAQ system with the initial approach [7], we identified points of improvement to refine the future system. User experience and lessons learned provided the base to reassess the software. To embrace opportunities coming from new technologies, providing an evolutionary framework to cope with the ever changing environment, a full revision of the current design was desirable.

One major concern with the current system lies in the infrastructure lock-in, resulting from the system's reliance on a specific operating system (OS) distribution and libraries. This constraint limits flexibility and the ability to evolve towards alternative software libraries or compilers available in other (more recent) operating system distributions.

Additionally, the transition from development to deployment proves to be time-consuming, marked by distinctly different environments (e.g., different hostnames) for development, validation, and production stages. This disparity hampers efficiency of the maintenance and development activities, such as bug fixes and new feature introduction. Furthermore, the work required to set up the computing infrastructure for development and validation turns out to be time- and resource-intensive.

The binding of CMS DAQ auxiliary service applications to either physical or virtual machines in the existing system, limits optimal resource utilization, because a safety margin on resources, such as memory and CPU, are required on each individual host.

The process of software update and rollback is often arduous, complicating the system management. It necessitates substantial system administration efforts, diverting resources from more strategic tasks.

A lack of portability due to the configuration binding to physical hostnames, makes changes in computing infrastructure labor-intensive. A minor change in a computing environment might require a full iteration from software reconfiguration to release and deployment (this applies to CMS DAQ auxiliary service applications).

The need for custom (private) scripts for software deployment and operation in test environments, adds supplementary burden on developers and users due to the differences between the two environments.

Finally, two distinct approaches are currently employed to operate and control the life cycle of services. A custom-built infrastructure of Run Control Management System (RCMS) (a web-based, graphical run control system) and Jobcontrol (a service used to start, stop and monitor processes in a distributed environment), working in tandem is used for short-lived application services (e.g., event building and read-out) [8], while for long-running, auxiliary services such as monitoring and error reporting systemd is used [6]. A single uniform approach is desirable to reduce the inherent complexity of the system.

3 Container-based architecture

Containerization transforms the network from being machine-oriented to being application-oriented. As such in container-based approach, DAQ applications are composed as cohesive groups of containers [9–11]. The required software is encapsulated within container images (e.g., Docker images) [9]. The containerization, a hallmark of modern software deployment, carries with it a host of benefits [12]. The orchestration of containers, once a cumbersome task, is streamlined through orchestration systems such as Kubernetes [13, 14], an industry-proven container orchestration platform [15, 16]. Containerization of software and container orchestration allow the use of available computational resources as a unified resource pool while different versions of software, down to the OS distribution specific libraries, can coexist within a single network host. Software changes are prepared at build time, while target machines are not affected by the ongoing modifications. See figure 1 for a graphical depiction of traditional compared to containerized approach.

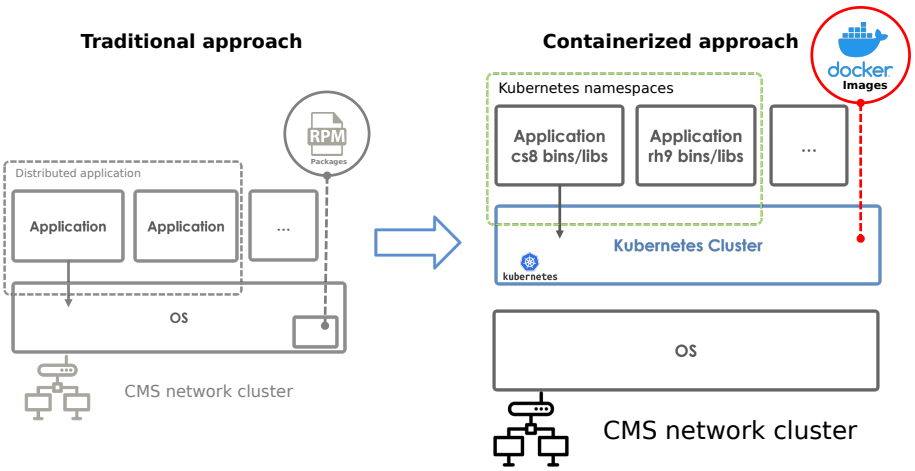

Figure 1. In traditional approach, a physical machine has libraries specific to one OS distribution installed. In containerized approach, OS distribution specific libraries are installed into container images. Containers with libraries from different OS distributions can be used on the same physical machine.

The containerized approach also provides reliability in software validation across both test and production environments, which are managed in the same manner. The construction of container images at build time ensures consistent installation across diverse targets, as container images encapsulate all required application dependencies. In turn, tests in production environment do not cause disruption on the target environment as rollbacks or re-installations are no longer needed.

In summary, containerization of software compounded by container orchestration addresses all the points of concern identified in section 2 at least to some extent.

4 Pilot project

4.1 Goals

The transition of the DAQ system from a traditional approach to a container-based architecture is being carried out through a pilot project. One of the goals of the pilot project is to rework DAQ system elements in a cloud approach based on Kubernetes patterns. Additionally, it aims at adopting a full software life cycle for the new environment, focusing on the development, delivery, deployment and operation. The project also seeks to resolve issues concerning the building and maintenance of the Kubernetes cluster and hardware resources, as well as determine implications on the current system administration environment.

The ultimate outcome of the pilot project is to implement a fully functional DAQ column from back-end electronics, collecting data from the detector front-ends, to High Level Trigger (HLT), including run control and monitoring.

The scope of the project spans all dimensions of expertise within the CMS DAQ system, encompassing operation, development, release, deployment, system administration, networking, security, and integration.

The aforementioned goals can be achieved through the implementation of the following study modules:

1. Performance and scalability;
2. Hardware access and binding (VME bus, PCI bus, remote direct memory access (RDMA), FEROL [17], etc.);
3. System fine-tuning (driver interrupts, non-uniform memory access (NUMA) settings, etc.);
4. User access (Graphical User Interface (GUI), Command Line Interface (CLI), Dashboards, API, etc.);
5. Networking (configuration, fine-tuning TCP/IP, RDMA over Converged Ethernet (RoCE), etc.);
6. Control and monitoring;
7. Configuration;
8. Additional technologies and their integration (Elasticsearch, Opensearch, Oracle, WinCC Open Architecture (WinCC OA), etc.);
9. Scalable event builder prototype;
10. Startup measurements.

4.2 Current achievements

At the time of writing of this paper, the work has already been performed in several areas. An in-depth exploration of user access options, encompassing GUIs, CLIs, Dashboards, and the Kubernetes API, was conducted as part of study module 4.

Due to the CMS DAQ's utilization of high bandwidth RDMA devices [18–21], an investigation and prototyping effort was made on RDMA-based device access (RoCE) in Kubernetes, demonstrating successful communication through RDMA devices in study modules 2 and 5.

The evaluation of various Kubernetes cluster networking approaches, including the building of prototypes with several Container Network Interface (CNI) plugins like flannel, Calico, and ipvlan [22], was undertaken in study module 5. Study module 7 involved the prototyping of different software configuration techniques within pods, such as custom scripting in Python and Helm [23] scripting.

In the realm of monitoring, deployment and testing of Elasticsearch and Opensearch within the Kubernetes cluster were carried out to assess their viability for monitoring functionalities in study modules 6 and 8. Multiple event building prototypes designed to operate within a Kubernetes cluster were implemented across study modules 1, 3, 6 and 9. Finally, measurements of startup time were conducted in study modules 1 and 10, with additional details provided in section 4.3.

4.3 Measurements

In the DAQ system, minimizing the periods of system downtime is of utmost importance. Since there are situations where system reconfiguration or restart is necessary, this leads to a requirement to minimize the startup time, and the current DAQ system is optimized for fast startup [5] (by startup we mean the full restart of the system). Thus it is imperative for the new design to also perform well in this regard.

In this work we performed measurements of system startup time. We considered the overall startup time to be the time interval from the moment right before starting installation of necessary Kubernetes objects using Helm tool until all the pods are ready to perform their function. Each pod running a single application, had to connect with every other application in the Kubernetes cluster via RDMA semantics. This interconnectivity is a typical requirement for an event building application. In addition, we measured the time needed for intermediate steps during startup in order to determine which steps are the most time consuming.

The measurements were performed on a Kubernetes cluster with 106 physical nodes (these machines were used for event building task by CMS DAQ during LHC Run 2). Each node was running a single pod. Required container images were pulled on each node before performing measurements. Each measurement was performed five times and the mean value of the startup times was calculated. After each measurement, all the pods and all other measurement-related Kubernetes objects (e.g., services, configmaps, namespace) were deleted before performing a new measurement.

The intermediate startup steps measured were the following: "pod scheduled" – all pods are scheduled to run on required nodes, "containers ready" – all containers are created, applications are being started, "app alive" – the time needed to start a simplified application without the requirement for interconnection, "pod connectable" – all DNS names required by an application are resolved to IP addresses, "app ready" – application with full configuration is successfully started, "app connected" - all required network connections specific to an application to fulfill its function and inter-operate are established. The measurement results are shown in figure 2.

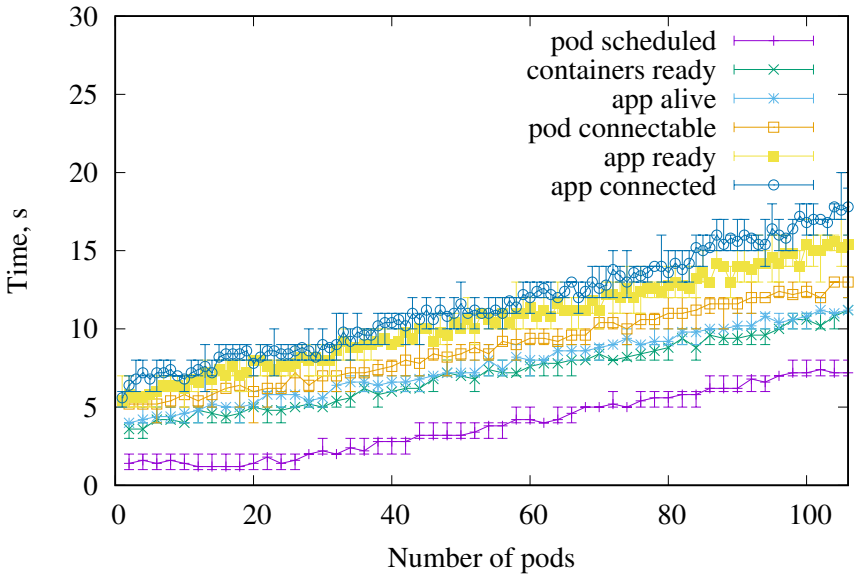

Figure 2. Startup time measurements. Points in the plot signify mean values, while bars signify maximum and minimum values measured.

Startup step	**Time**
Pod scheduled	7.2 s
Containers ready	11.2 s
App alive	11.2 s
Pod connectable	13 s
App ready	15.4 s
App connected	17.8 s

Table 1. The mean values of the different cumulative components of startup time with the maximum number of pods (106).

We observe that the startup time grows with the increased number of pods. As shown in table 1, it takes 7.2 s to schedule pods on all 106 nodes available in the cluster and it takes another 4 s to bring containers to the ready state. In order to speedup the transition to the "pod connectable" state, we optimized the DNS names resolution by using a custom script interrogating Kubernetes API. We can see that after containers are ready it takes 1.8 s in each pod to resolve DNS names required to access every other pod. After pods are connectable our test application starts on each single pod in 2.4 s and it takes an additional 2.4 s for all applications to connect to each other in the cluster.

The estimated number of CMS DAQ event builder nodes in Phase 2 is 200 [24], while our measurements were performed on 106 nodes, also different hardware will be used in Phase 2. Therefore additional measurements are needed to get conclusive results. However, 35 s was considered an acceptable startup time for CMS DAQ during LHC Run 1 [5] and our measurements in our current setup are therefore promising: if the observed trend persists for

higher number of nodes, our measured 17.8 s for 106 nodes would translate into a comparable value with 35 s, with the required number of nodes and the new hardware.

5 Conclusion and future work

Encouraging results were achieved so far. Containerization was shown to be an appealing technology for DAQ applications as a replacement for a bare metal infrastructure. Containerization and orchestration solve most of the limitations experienced by running with a traditional infrastructure. The approach fits well with distributed inter-communicating applications such as the event builder. No limitations of the approach were discovered during the investigation of predefined objectives. Future work concerns unresolved study modules, finalization of a working DAQ column from back-end electronics to HLT, repeating startup measurements on the actual machines and network infrastructure that will be used by CMS DAQ in Phase 2, whenever these resources will become available, and finally measuring the performance of the actual event building in the real Phase 2 system.

References

[1] *CMS, The Compact Muon Solenoid: technical proposal* (CERN, Geneva, 1994), `http://cds.cern.ch/record/290969`
[2] The CMS Collaboration, *Development of the CMS detector for the CERN LHC Run 3* (2023), CERN-EP-2023-136, submitted to JINST
[3] T.S. Pettersson, P. Lefèvre (LHC Study Group), *The Large Hadron Collider: conceptual design* (1995), `http://cds.cern.ch/record/291782`
[4] S. Chatrchyan, V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan et al., Physics Letters B **716**, 30 (2012)
[5] G. Bauer, U. Behrens, K. Biery, V. Boyer, J. Branson, E. Cano, H. Cheung, M. Ciganek, S. Cittolin, J. Coarasa et al., Journal of Physics: Conference Series **219**, 022003 (2010)
[6] *System and service manager*, `https://systemd.io/`, accessed: 2023-08-19
[7] H. Sakulin et al. (CMS), EPJ Web Conf. **214**, 01015 (2019)
[8] G. Bauer, V. Boyer, J. Branson, A. Brett, E. Cano, A. Carboni, M. Ciganek, S. Cittolin, V. O'dell, S. Erhan et al., Journal of Physics: Conference Series **119**, 022010 (2008)
[9] L. Rice, *Container security: fundamental technology concepts that protect containerized applications*, 1st edn. (O'Reilly Media, Sebastopol, CA, 2020)
[10] *Use containers to build, share and run your applications*, `https://www.docker.com/resources/what-container/`, accessed: 2023-08-19
[11] *Open container initiative*, `https://opencontainers.org/`, accessed: 2023-08-19
[12] D. Walsh, *Podman in Action* (Manning Publications, 2023)
[13] *Production-grade container orchestration*, `https://kubernetes.io/`, accessed: 2023-08-19
[14] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, ACM Queue **14**, 70 (2016)
[15] B. Johansson, M. Rågberger, T. Nolte, A.V. Papadopoulos, *Kubernetes Orchestration of High Availability Distributed Control Systems*, in *2022 IEEE International Conference on Industrial Technology (ICIT)* (2022), pp. 1–8
[16] A. Chazapis, C. Pinto, Y. Gkoufas, C. Kozanitis, A. Bilas, *A Unified Storage Layer for Supporting Distributed Workflows in Kubernetes*, in *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (Association for Computing Machinery, New York, NY, USA, 2021), CHEOPS '21

[17] G. Bauer, T. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, J.A. Coarasa, G.L. Darlea, C. Deldicque, M. Dobson et al., Journal of Instrumentation **8**, C12039 (2013)

[18] T. Shanley, J. Winkles, *InfiniBand Network Architecture*, Mindshare PC System Architecture (Addison-Wesley, 2003)

[19] M. Beck, M. Kagan, *Performance evaluation of the RDMA over ethernet (RoCE) standard in enterprise data centers infrastructure*, in *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching* (2011), pp. 9–15

[20] T. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, G.L. Darlea, C. Deldicque, M. Dobson, A. Dupont, S. Erhan et al., IEEE Transactions on Nuclear Science **62**, 1099 (2015)

[21] T.A. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, G.L. Darlea, C. Deldicque, M. Dobson, A. Dupont, S. Erhan et al., *Boosting Event Building Performance using Infiniband FDR for CMS Upgrade* (2014), `https://cds.cern.ch/record/1712209`

[22] *The container network interface*, `https://www.cni.dev/`, accessed: 2023-08-19

[23] *Helm. The package manager for Kubernetes*, `https://helm.sh/`, accessed: 2023-08-19

[24] CMS Collaboration, *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger* (2021), `https://cds.cern.ch/record/2759072`