EPJ Web of Conferences **337**, 01075 (2025) https://doi.org/10.1051/epjconf/202533701075
CHEP 2024

Architecting software applications in containerized environment for CMS data acquisition

Jaafar Alawieh[1], *Kareen* Arutjunjan[1], *Miguel* Bacharov Durasov[1], *Ulf* Behrens[2], *Andrea* Bocci[1], *James* Branson[3], *Philipp Maximilian* Brummer[1], *Jan Andrzej* Bugajski[1], *Eric* Cano[1], *Sergio* Cittolin[3], *Albert* Corominas I Mariscot[1], *Georgiana-Lavinia* Darlea[4], *Christian* Deldicque[1], *Marc* Dobson[1], *Antonin* Dvorak[1], *Christos* Emmanouil[1], *Antra* Gaile[1], *Dominique* Gigi[1], *Frank* Glege[1], *Guillelmo* Gomez-Ceballos[4], *Patrycja* Gorniak[1], *Jeroen* Hegeman[1], *Guillermo* Izquierdo Moreno[1], *Thomas Owen* James[1], *Tejeswini* Jayakumar[1], *Wassef* Karimeh[1], *Rafał* Krawczyk[2], *Wei* Li[2], *Kenneth* Long[4], *Frans* Meijers[1], *Emilio* Meschi[1], *Srećko* Morović[3], *Babatunde John* Odetayo[1,5], *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[1,6,*], *Polyneikis* Tzanis[1], *Cristina* Vazquez Velez[1], and *Petr* Žejdl[1]

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

Abstract. The data acquisition (DAQ) system stands as an essential component within the CMS experiment at CERN. It relies on a large network system of computers with demanding requirements on control, monitoring, configuration and high throughput communication. Furthermore, the DAQ system must accommodate various application scenarios, such as interfacing with external systems, accessing custom electronics devices for data readout, and event building. We present a versatile and highly modular programmable C++ framework designed for crafting applications tailored to various needs, facilitating development through the composition and integration of modules to achieve the desired DAQ capabilities. This framework takes advantage of reusable components and readily available off-the-shelf technologies. Applications are structured to seamlessly integrate into a containerized ecosystem, where the hierarchy of components and their aggregation is specified to form the final deployable unit to be used across multiple computers or nodes within an orchestrating environment. The utilization of the framework, along with the containerization of applications, enables coping with the complexity of implementing the CMS DAQ system by providing standardized structures and components to achieve a uniform and consistent architecture.

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

1 Introduction

The Data Acquisition (DAQ) system at CERN's CMS experiment [1–3] is tasked with the essential role of acquiring, processing, and transmitting data from the detector to required computing systems for further analysis. The DAQ system operates within a distributed computing environment, ensuring high throughput communication, complex system control, and continuous monitoring [2]. It interacts with custom electronics, supports event building, and interfaces with external systems. In response to these challenges, a modular C++ framework called XDAQ [4, 5] has been developed to support the creation of DAQ applications, enabling a more structured and efficient approach to system design and implementation. The framework has been modernized to leverage containerization and orchestration technologies to allow scalable, reliable, and efficient software architecture that can meet the evolving needs of the CMS DAQ system. Designing DAQ applications within a containerized and orchestrated environment offers an approach for managing complexity, enhancing deployment, and ensuring high availability and fault tolerance in large-scale distributed systems [6].

2 The role of the software framework in the CMS DAQ system

The XDAQ framework, a mature and reliable software solution, has been used in the CMS DAQ system for many years [5, 7–10]. It offers a pre-designed approach to developing data acquisition applications, providing a uniform and consistent methodology for application configuration. The framework's reusable components simplify development for users and ensures reliability through an extensive testing in several application scenarios.

The framework is providing uniform APIs facilitating the optional use of several off-the-shelf technologies. In addition, it supports integration with containerization and orchestration environments.

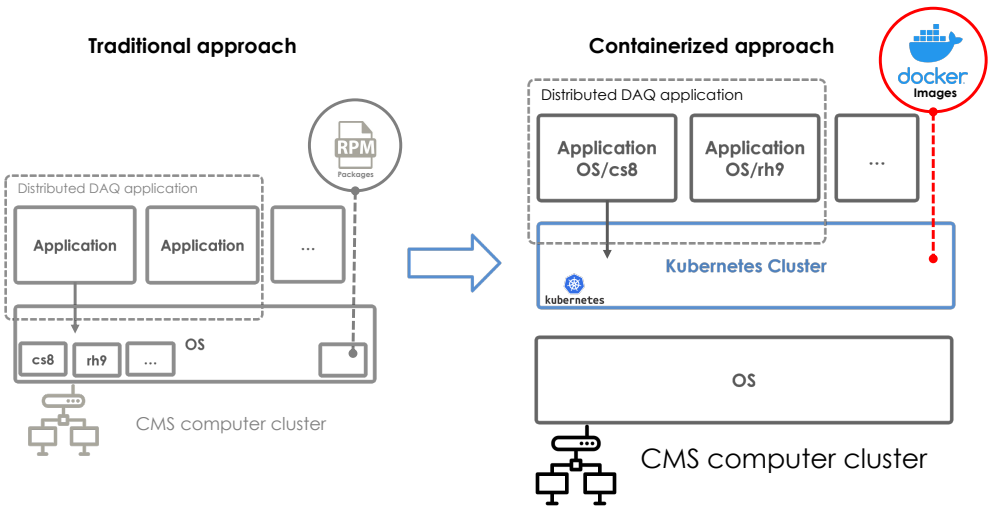

Figure 1. In traditional computer cluster application software and its dependencies are installed directly on different physical or virtual machine with different OS platforms. In containerized approach all applications and their dependencies are packaged into container images independently of physical or virtual machine installation.

3 Embracing containerization and orchestration

Containerization [11] represents a paradigm shift in how the CMS DAQ system is designed and deployed [12]. An homogeneous architecture is achieved by encapsulating applications and their dependencies within containers and operating them in standardized orchestration environment. Containerization helps to avoid infrastructure lock-in caused by dependency on a specific operating system and libraries as shown in Figure 1. The approach improves portability, as containers are independent of the underlying infrastructure, allowing applications to run consistently across different environments. Orchestration provides a standardized infrastructure and tools that enable the seamless start and execution of applications on provided computational resources, whether they are bare-metal machines, virtual machines, or system-on-chip (SoC) devices. Several orchestration environments can run in parallel and independently, distributed over different resources as shown in Figure 2. Therefore, containerized applications can be developed and assessed in isolated test environment before final deployment in the production system, which enhances the reliability of software validation process. Both testing and production environments benefit from consistent installation procedures, achieved through image construction at build time. This guarantees reproducibility across different targets, reducing errors and inconsistencies. An additional advantage of containerized applications is the capability to be validated straight in production environments, without disrupting installations, eliminating the need for software rollbacks and complex updates. The containerization in the CMS DAQ system is facilitated by Kubernetes [6, 13], an orchestration platform that provides tools for deploying, scaling and managing containerized applications across a distributed computing environment.

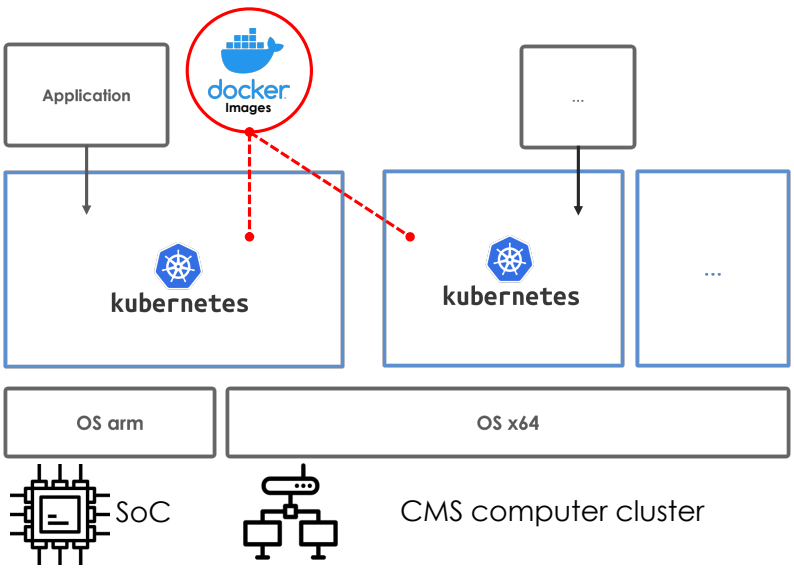

Figure 2. Partitioning of resources through multiple Kubernetes clusters.

4 The architecture of the containerized CMS DAQ system

A data acquisition application represents a specialized software component designed to perform specific tasks within a DAQ system, such as event building, monitoring, or error reporting. Traditionally, DAQ infrastructures have been machine-oriented, with software closely

tied to specific hardware resources. Containerization is shifting this paradigm, enabling infrastructures to become application-oriented. This approach abstracts the software from underlying hardware, enhancing flexibility, scalability, and portability.

In this new paradigm DAQ applications can be deployed in multiple Kubernetes clusters serving distinct purposes, such as supporting central DAQ operations, development environments, or individual sub-detector functionalities. Each cluster ensuring allocation and protection of dedicated DAQ resources, enabling optimized performance and isolation for various components of the system.

4.1 Fundamentals of application architecture

The XDAQ framework provides a standardized design pattern for the implementation of data acquisition applications [4]. At the core of a data acquisition application developed with XDAQ lies the XDAQ executive, a key architectural component. The XDAQ executive corresponds to a Linux-based executable program capable of hosting multiple XDAQ modules. These modules can be either custom-developed or reusable, facilitating modular and scalable DAQ application design. The framework includes interfaces for essential functionalities such as configuration, control, monitoring, communication and device access, see Figure 3.

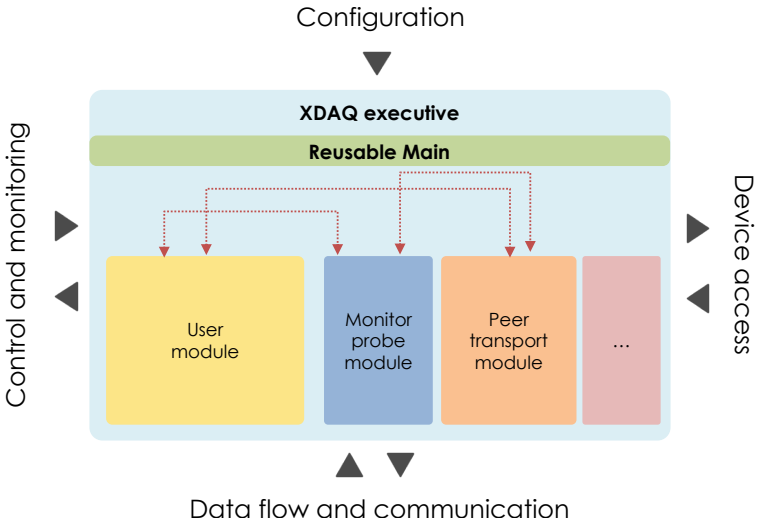

Figure 3. Modularized structure of XDAQ executive with inter-communicating modules and external interfaces.

4.2 XDAQ executives in Kubernetes

Kubernetes environment is well suited for DAQ applications which are often designed as highly inter-communicating distributed systems. In this environment XDAQ executives run inside Kubernetes pods, each assigned a unique and consistent network identity. This identity is defined via Kubernetes manifests, enabling the executives to maintain addresses regardless of changes in the physical machine hostnames or deployment to different Kubernetes clusters. Kubernetes provides flexible tools for managing replicas of executives, employing

mechanisms such as StatefulSet, Deployment and DaemonSet to ensure redundancy and scalability as shown in Figure 4, and ConfigMaps to allow configuration data to be injected into application pods during initialization or runtime [13].

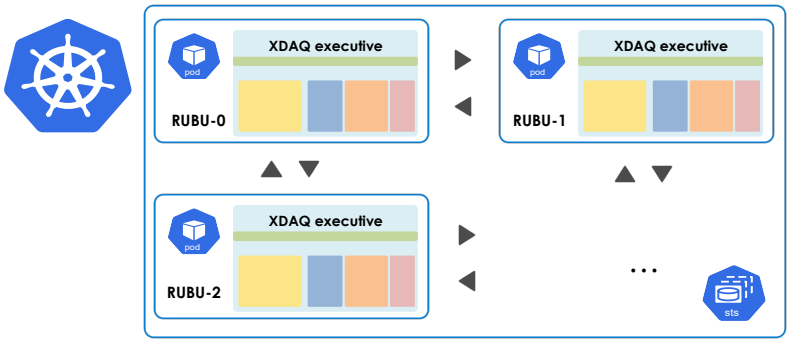

Figure 4. Multiple replicas of XDAQ executives running inside Pods created using Kubernetes StatefulSet object.

4.3 Framework integration into containerized environments

The XDAQ framework seamlessly integrates into containerized environments, utilizing technologies commonly associated with Kubernetes ecosystems. These include communication protocols such as HTTP and REST [14], along with configuration formats like YAML and JSON. Some other utilities already present in the framework support interaction with Kubernetes, such as: access to Kubernetes API, interface to Kubernetes events, readiness and liveness probe support, embedded programmable web servers that can be accessed through ingress controllers, etc. The framework also supports the dynamic parameterization of XDAQ executives running within Kubernetes pods, ensuring adaptability to varying deployment scenarios and operational requirements.

5 Complementary off-the-shelf technologies in the CMS DAQ system

The modernization of the CMS DAQ system is supported by and integrates various off-the-shelf technologies to provide data communication, application monitoring, logging and error reporting. The MQTT [15] protocol enables lightweight and efficient communication between applications via message brokers, making it a highly suitable for distributed systems with substantial communication requirements leveraging a publisher-subscriber model.

The framework benefits from MQTT client libraries such as Mosquitto (C) and Paho (C++), facilitating lightweight and reliable message exchange for both publishing and subscribing. The message broker (e.g., Mosquitto, EMQX, RabbitMQ) ensures scalable and high-performance distribution of MQTT messages. In addition, configurable data gathering, processing and forwarding facilities (e.g., Fluentd and Fluentbit) interfacing message brokers, enable data transmission across system components, including highly scalable NoSQL data storages (e.g., Opensearch, Elasticsearch), which offer a reliable approach, providing efficient indexing and search capabilities to support collection of monitoring, logging and error reporting data.

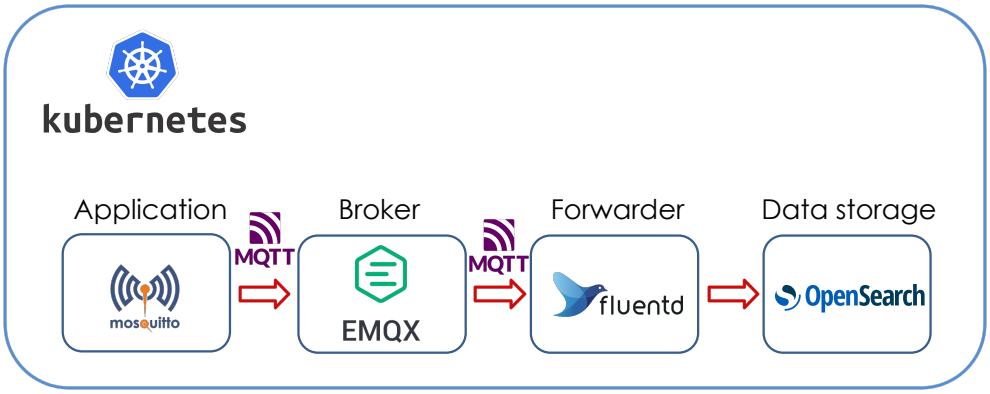

Figure 5. Full chain of monitoring report from a source XDAQ application up to data storage, based on off-the-shelf technologies.

All these components can be deployed within a containerized environment, running as Kubernetes pods, providing scalability, fault tolerance, and efficient resource management in the large-scale distributed computing infrastructure of the CMS DAQ system (see Figure 5).

6 Steps from development to deployment

The process of developing, releasing and deploying of CMS DAQ software, follows a systematic sequence of steps. At each step, various components, such as source code, software libraries, RPM (Red Hat Package Manager) files, and container images, are processed and

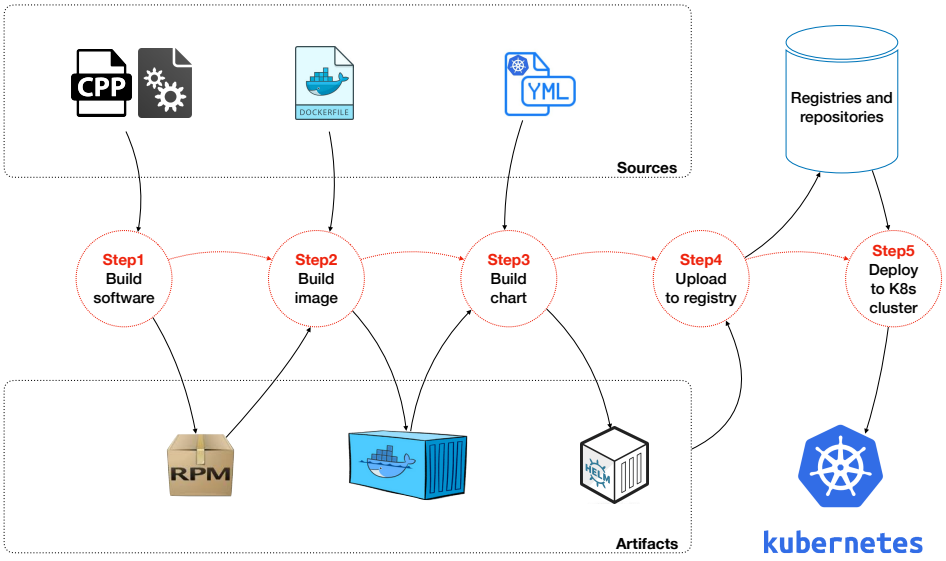

Figure 6. Sequence of steps with required input sources to generate necessary artifacts for the final deployment in Kubernetes environment.

combined, ultimately resulting in the final artifact, a Helm chart, which can be directly deployed in a Kubernetes cluster.

As shown in Figure 6, initially, the application software is compiled and packaged into RPMs, facilitating its distribution and installation across compatible systems. Subsequently, the packaged application is containerized by constructing container images, encapsulating all dependencies required for execution in a consistent and reproducible manner.

After containerization, a set of Kubernetes manifests and Helm templates that describe Kubernetes resources (Deployments, Services, ConfigMaps, etc.) are packaged into Helm charts. These charts serve as a standardized mechanism for managing Kubernetes applications, enabling efficient configuration, versioning and deployment. Finally, the Helm charts are installed into a Kubernetes cluster, ensuring application deployment within the orchestrating environment. All input sources and build instructions needed to generate artifacts are effectively managed in Git repositories and automated using GitLab CI/CD pipelines. All artifacts are stored in registries provided by Gitlab infrastructure.

7 Conclusion

The transition to a containerized architecture, supported by the XDAQ framework, addresses many of the challenges posed by traditional infrastructures. The integration of Kubernetes as the orchestration platform provides a reliable and efficient mechanism for deploying, managing, and scaling DAQ applications across distributed computing resources. The adoption of off-the-shelf technologies in the CMS DAQ system offers significant advantages. Publisher-subscriber model supported by MQTT protocol as well as the large range of MQTT-compatible software is scalable and well suited for monitoring, logging and error reporting in CMS DAQ system. The systematic approach to software development and deployment, encompassing the packaging of applications into RPMs, containerization and the use of Helm charts for Kubernetes resource management, establishes a consistent and reproducible workflow. Overall, the enhancements introduced in the CMS DAQ system align with the evolving requirements of high-energy physics experiments, ensuring the reliability and efficiency needed to support the CMS experiment's data acquisition demands both now and in the future.

References

[1] T.S. Pettersson, P. Lefèvre (LHC Study Group), The Large Hadron Collider: conceptual design (1995), https://doi.org/10.17181/CERN-AC-95-05-LHC

[2] S. Chatrchyan et al. (CMS), The CMS experiment at the CERN LHC, JINST **3**, S08004 (2008).

[3] A. Hayrapetyan et al. (CMS), Development of the CMS detector for the CERN LHC Run 3, JINST **19**, P05064 (2024).

[4] J. Gutleber, S. Murray, L. Orsini, Towards a homogeneous architecture for high-energy physics data acquisition systems, Computer Physics Communications **153**, 155 (2003).

[5] V. Brigljevic, G. Bruno, E. Cano, A. Csilling, S. Cittolin, D. Gigi, F. Glege, M. Gulmini, J. Gutleber, C. Jacobs et al., Using XDAQ in application scenarios of the CMS experiment, eConf C **0303241**, MOGT008 (2003).

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, Omega, and Kubernetes, ACM Queue **14**, 70 (2016).

[7] J. Gutleber, L. Orsini, Software architecture for processing clusters based on I2O, Cluster Computing **5**, 55 (2002).

[8] G. Antchev, L. Berti, E. Cano, S. Cittolin, S. Erhan, D. Gigi, J. Gutleber, C. Jacobs, G. Maron, F. Meijers et al., The Data Acquisition System for the CMS Experiment at the LHC, in *7th International Conference on Advanced Technology and Particle Physics* (2002), pp. 540–549

[9] G. Bauer, U. Behrens, K. Biery, J. Branson, E. Cano, H. Cheung, M. Ciganek, S. Cittolin, J.A. Coarasa, C. Deldicque et al., Monitoring the CMS data acquisition system, Journal of Physics: Conference Series **219**, 022042 (2010).

[10] G. Bauer et al., The data-acquisition system of the CMS experiment at the LHC, J. Phys. Conf. Ser. **331**, 022021 (2011).

[11] L. Rice, Container security: fundamental technology concepts that protect containerized applications, 1st edn. (O'Reilly Media, Sebastopol, CA, 2020)

[12] V. Amoiridis, U. Behrens, A. Bocci, J. Branson, P. Brummer, E. Cano, S. Cittolin, J. Da Silva Almeida Da Quintanilha, G.L. Darlea, C. Deldicque et al., Towards a container-based architecture for CMS data acquisition, EPJ Web of Conf. **295**, 02031 (2024).

[13] Kubernetes, https://kubernetes.io/, accessed: 2025-02-13

[14] R.T. Fielding, Ph.D. thesis, University of California, Irvine (2000), https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[15] A. Banks, E. Briggs, K. Borgendale, R. Gupta, MQTT version 5.0, OASIS standard (2019), accessed: 2025-02-13, https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html