Evolution of the ATLAS TDAQ online software framework towards Phase-II upgrade: use of Kubernetes as an orchestrator of the ATLAS Event Filter computing farm

Alina Corso Radu[1,*], *Giuseppe* Avolio[2], *Matteo* Ferrari[2], *Andrei* Kazarov[3], *Igor* Soloviev[1], and *Serguei* Kolos[1]

[1]University of California Irvine, Physics & Astronomy Department, Irvine, CA 92697, US
[2]CERN, EP Department, Espl. des Particules 1/1211, 23 Genève, Switzerland
[3]University of Johannesburg, Department of Mechanical Engineering Science, Johannesburg, 2006, SA

Abstract. The ATLAS experiment at the LHC at CERN continuously evolves its TDAQ system to meet the challenges of new physics goals and technological advancements. As ATLAS prepares for the Phase-II Run 4 of the LHC, significant enhancements in the TDAQ Controls and Configuration (TDAQ-CC) tools have been designed to ensure efficient data collection, processing, and management. This abstract presents the evolution of ATLAS TDAQ-CC system leading up to Phase-II Run 4. As part of the evolution towards Phase-II, Kubernetes has been chosen to orchestrate the Event Filter (EF) farm. By leveraging Kubernetes, ATLAS can dynamically allocate computing resources, scale processing capacity in response to changing data taking conditions and ensure high availability of data processing services. The integration of the Kubernetes with the TDAQ Run Control framework enables perfect synchronisation between the experiment's data acquisition components and the computing infrastructure. We will discuss the architectural considerations and implementation challenges involved in Kubernetes integration with the ATLAS TDAQ-CC system. We will highlight the benefits of using Kubernetes as an EF farm orchestrator, including improved resource utilization, enhanced fault tolerance, and simplified deployment and management of data processing workflows. In addition, we will report on the extensive testing of Kubernetes that was conducted using a farm of 2500 servers within the experiment data taking environment, demonstrating its scalability and robustness in handling the demands of the ATLAS TDAQ system for Phase-II. The adoption of Kubernetes represents a significant step forward in the evolution of ATLAS TDAQ-CC system, aligning with industry best practices in container orchestration.

ATL-DAQ-PROC-2025-004
10 February 2025

1 Introduction

The ATLAS experiment at CERN [1] is preparing for the HL-LHC era, which aims to achieve a luminosity of $7.5 \times 10^{34}\text{cm}^{-2}\text{s}^{-1}$. This enhancement will result in over 200 proton-proton interactions per bunch crossing. This unprecedented interaction rate will lead to a dramatic increase in the complexity of events that need to be processed, leading to a higher volume of

*e-mail: alina.radu@cern.ch

data, far exceeding the current processing capabilities. To accommodate the higher luminosity and to exploit full HL-LHC potential, the trigger rates of the system must increase, for example, the Level-0 (L0) hardware trigger will reduce the data rate from 40 MHz to approximately 1 MHz which is ten times more than Run 3 L1 rate. In addition to the trigger rate increases, the size of the events themselves will also grow to reach approximately 4.6 MB, compared to the 1.5 MB event size in Run 3. This substantial increase in event size means that the data acquisition throughput will need to scale accordingly, placing additional stress on the system's resources.

To address these challenges, the Phase-II upgrade of the TDAQ system introduces both hardware and software improvements [2]. One of the many critical aspects is the orchestration of the EF computing farm, which will manage about five thousand computing nodes, enabling real-time event filtering and processing. Kubernetes has been selected to manage this complex distributed system due to its efficiency in resource management, fault tolerance, and scalability. The EF farm currently operates under a static configuration and relies on in-house process management system (PMG), which is part of the TDAQ Controls and Configuration software [3]. However, this approach is not suitable enough to handle the demands of HL-LHC. A robust and dynamic orchestration mechanism is required to ensure efficient resource utilization, fault tolerance, and scalability. The solution must manage thousands of computational nodes in a flexible and automated manner, enabling real-time data processing without compromising performance or reliability.

This paper discusses the requirements for the EF farm orchestrator, the choice of Kubernetes, and its integration into the ATLAS TDAQ system.

2 Kubernetes as the EF Farm Orchestrator

Kubernetes [4], an open-source container orchestration platform, has been adopted as the solution to meet the demanding requirements of the EF farm during the Phase-II upgrade of the ATLAS experiment. The EF farm, that will consist of maximum five thousand computing nodes, requires a robust mechanism to manage workloads, to ensure system reliability, and to allocate resources dynamically. Kubernetes addresses these challenges through its container orchestration capabilities, providing a scalable and fault-tolerant solution.

One of the critical requirements for the EF farm orchestrator is the ability to manage the lifecycle of EF processes seamlessly. These processes must run continuously in quasi real time to process physics data or offline-simulation workloads while sharing the same computing resources efficiently. Kubernetes allows the activation or deactivation of computational units at runtime, optimizing CPU and memory usage across the cluster. The flexibility in resource allocation ensures that computing resources are utilized optimally, minimizing waste and reducing overhead. Kubernetes also provides the capability to monitor, control, and manage thousands of active processes. By offering precise control over starting, stopping, and monitoring workloads, Kubernetes ensures system reliability even under extreme operational conditions. Additionally, the platform allows users to specify detailed process requirements, such as command-line parameters and environment variables, ensuring that EF processes can be properly configured and deployed with high granularity. Fault tolerance is another significant advantage of Kubernetes as an EF farm orchestrator. The platform dynamically adjusts to real-time conditions, automatically recovering processes that fail due to hardware or software issues. This resilience is critical for maintaining the stability of the EF computing farm, particularly during long data-taking periods where interruptions must be minimized.

Furthermore, Kubernetes supports containerized applications, which align with modern industry practices in cloud-native computing. By deploying EF processes as containers, the

system benefits from simplified software management, easier updates, and improved portability across different environments. This containerized approach also enhances scalability, as Kubernetes can easily scale workloads up or down based on changing data-taking conditions, ensuring that the system adapts efficiently to changes in luminosity for example.

3 Testing Kubernetes in the ATLAS EF Farm

The testing of Kubernetes for the EF farm was carried out using dedicated clusters installed in two distinct environments: the TDAQ lab and the ATLAS experiment's closed network (P1). These environments were designed to simulate conditions as close as possible to the production setup. The P1 cluster is configured to mirror real-world requirements and consists of four control plane nodes, four ETCD nodes, two Prometheus nodes for monitoring, and about 2600 worker nodes, representing more than 50% of the maximum farm size expected for Run 4 in terms of number of servers.

The primary objectives of these tests included evaluating Kubernetes' scalability, fault tolerance, and efficiency in resource management. Scalability was tested by running workloads across the 2600-node EF farm, ensuring that the Kubernetes orchestration layer could manage such a large number of nodes without degradation in performance. The tests also focused on optimizing the scheduling of Pods (the smallest deployable unit in Kubernetes) using different strategies. One approach, known as daemon set, ensures that a Pod is deployed on every node within the cluster. This method proved effective for high-availability tasks but did not necessarily optimize the resource usage. Multiple PODs per node can be deployed in this case using more than one daemon set. A second strategy, called topological spread, distributes Pods evenly across available nodes, ensuring better load balancing and improved fault tolerance. A third one, resource-based scheduling strategy prioritizes node availability based on CPU and memory resources, providing an efficient balance between scheduling time and resource utilization. Some results of these tests are illustrated in Figure 1.

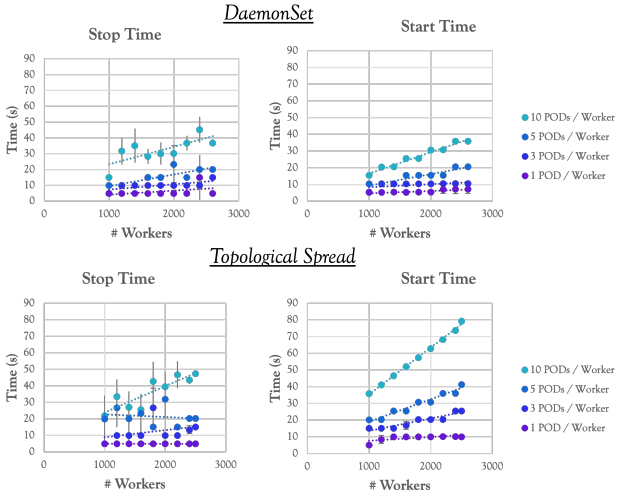

Figure 1. Start and stop time for different numbers of workers and for different numbers of Pods per worker node using two strategies: daemon set and topological spread

The plots show the Pods start and stop times using two different scheduling strategies, namely daemon set and topological spread. The data demonstrates that daemon set has a faster deployment time but in practice shows less flexibility to varying workloads. In contrast, topological spread requires additional time for scheduling but demonstrates the potential for easily handling larger, evenly distributed workloads efficiently. The plots compare start and stop times for different numbers of Pods per worker node, showing a consistent increase in start and stop time as the number of Pods per node rises. These results highlight the trade-offs between resource allocation efficiency and scheduling overhead, emphasizing the importance of choosing an appropriate strategy for different workloads.

The comparative testing of these strategies shown in Figure 2 revealed important insights. While topological spread offered better flexibility and ease of use, it was more costly regard-

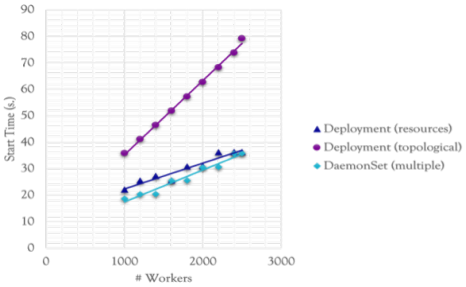

Figure 2. Start time comparison for 10 Pods per worker node using different strategies

ing scheduling overhead. On the other hand, resource-based scheduling proved to be the most efficient in terms of deployment time, making it a preferred approach for scenarios requiring quick resource allocation. These findings demonstrated Kubernetes' flexibility in adapting to different operational needs while maintaining high performance and reliability. It is worth mentioning that by upgrading the control plane nodes hardware from 2 x Intel Xeon E5-2650 v3 6C/12T to 2 x AMD EPYC 7313 16C/32T, a 50% gain in PODs start time was achieved using the topological spread deployment (results shown in Figures 1 and 2 refer to the new hardware). This improvement makes the topological spread strategy more affordable in terms of deployment time, highlighting its viability under upgraded hardware conditions.

In summary, the extensive testing conducted across large-scale cluster validated Kubernetes' capability to orchestrate the EF farm effectively. The platform showed excellent scalability, fault tolerance, and resource management, ensuring that the ATLAS TDAQ system can meet the challenges of data-taking under HL-LHC conditions. The ability to fine-tune scheduling strategies further highlighted Kubernetes' adaptability, making it a robust solution for managing the EF computing farm.

4 Integration with the TDAQ Run Control Framework

The integration of Kubernetes into the existing TDAQ Run Control framework represents a significant advancement in the management of the ATLAS EF farm. The Run Control framework serves as the central system for synchronizing and managing data acquisition components across thousands of servers. To ensure a seamless operation, this framework leverages a Finite State Machine (FSM) model, which orchestrates the different states of the data acquisition processes and maintains synchronization across all participating components.

The Kubernetes integration was accomplished through the development of a specialized application referred to as the Kubernetes-TDAQ bridge or KEFController. This application uses Custom Resource Definitions (CRDs) which allow Kubernetes to deploy EF applications while respecting the FSM structure, ensuring that data-taking workflows remain synchronized across the distributed infrastructure.

The bridge application implements the Kubernetes operator pattern, employing an in-house C++ wrapper built on top of the Kubernetes C-library [5]. This wrapper serves as the interface between Kubernetes cluster and the TDAQ framework, facilitating tasks such as process orchestration, error handling, and resource monitoring. During a run, the bridge application ensures that Kubernetes processes align with TDAQ's finite state transitions, enabling dynamic resource management and robust error recovery.

Moreover, the integration process incorporates key TDAQ tools and services to maintain system stability. The TDAQ-native configuration system (OKS) is used to define and manage the operational parameters of EF applications, ensuring a standardized setup across the infrastructure. Error handling is implemented using the TDAQ Error Reporting System (ERS), which captures and logs issues related to EF processes and Kubernetes tasks. Finally, the system health and performance are monitored continuously, with the collected data being fed back to the TDAQ Expert System (CHIP) for analysis and troubleshooting. A schema of the KEFController application and its interaction is shown in Figure 3.

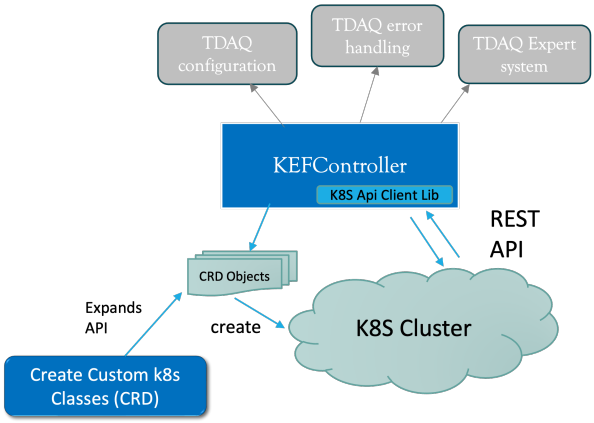

Figure 3. Schema of the Kubernetes-TDAQ bridge application and its interactions

By combining Kubernetes orchestration with the TDAQ Run Control framework, the EF farm gains the ability to manage resources dynamically while maintaining synchronization with the overall data acquisition system. This integration ensures that EF processes are executed reliably and efficiently, even under the extreme conditions of HL-LHC data-taking. The result is a cohesive and fault-tolerant system capable of meeting the stringent operational demands of the upgraded ATLAS experiment.

5 Deployment scenarios

The deployment of the EF processes and the needed dataflow (DF) services requires careful planning to properly balance maintainability, resource efficiency, and performance. Resource requests must be accurately defined to ensure optimal scheduling and avoid inefficiencies during deployment. Given the significant scale of the EF computing farm, where approximately

64,000 processing units are needed according to the current estimations, it was necessary to evaluate deployment strategies that could effectively manage such a high load. Several approaches are currently under consideration for deploying EF processes.

The default 'single-container per Pod' approach offers fine-grained control over resource allocation, as each EF process operates independently within its own container. While this provides precise scheduling and monitoring, it introduces important overhead when managing tens of thousands of individual Pods, increasing complexity and deployment times. The 'combined-containers per Pod' strategy combines containers for multiple EF processes and DF services into a single Pod. This strategy simplifies the deployment process, as it reduces the overall number of Pods that Kubernetes must manage, thereby lowering scheduling overhead and system complexity. However, this approach has limitations, such as reduced flexibility for independent scaling of DF and EF components, as their lifecycles remain tightly coupled. The 'multi-Pod' solution (different PODs for EF and DF), balances these trade-offs by allowing grouped processes to share CPU and memory while maintaining modularity across Pods. This strategy enhances flexibility and ensures that resource efficiency remains a priority.

Each strategy presents trade-offs. While the single-container approach offers greater granularity and control, the 'combined-containers' approach improves deployment efficiency at the cost of flexibility. Manual interventions at the Pod level can become challenging when managing multiple containers within a single Pod. Nevertheless, health checks can be implemented to monitor individual containers, mitigating some of these limitations.

Future work will focus on refining these strategies through further automation and advanced monitoring solutions. Machine learning techniques will be explored to optimize resource management, predict workload requirements, and reduce manual intervention. By aligning these efforts with industry cloud-native practices, the ATLAS EF farm will achieve higher performance and reliability, ensuring it can handle the HL-LHC's demanding data-processing requirements.

6 Simulation workloads using EF farm

The Kubernetes cluster installation at P1 is extending its usage beyond traditional Event Filter (EF) workloads to include offline-simulation jobs, project called KSim@P1. The primary objective of KSim@P1 is to transition from virtual machines (VMs) to containerized environments orchestrated by Kubernetes. This shift offers several advantages, such as simplified deployment, enhanced scheduling strategies, and better control over resource usage. Additionally, Kubernetes' existing tools, like Prometheus for monitoring, are leveraged to ensure efficient operation. A critical design goal is maintaining network isolation to protect experimental networks and ensure that simulation jobs do not compromise system stability. Furthermore, Kubernetes facilitates seamless access to necessary data sources such as CVMFS, while enabling real-time log retrieval for debugging by experts.

KSim@P1 uses HTCondor [6] to manage simulation jobs, which are executed in containerized environments using Singularity as the runtime. Preliminary tests at P1, conducted on a single server, demonstrated the stability of this configuration. Expanded testing in the TDAQ tbed environment further validated these results, with over 500 cores running simulation jobs for a week without issues, as it is illustrated in Figure 4.

The successful integration of simulation jobs into Kubernetes has showcased its flexibility and robustness. However, there is room for improvement. Future developments will focus on scaling the system, refining resource management strategies, and ensuring seamless coexistence with TDAQ workloads. Additional tests will explore new deployment strategies, such as multi-container Pods, to optimize resource usage for large-scale simulations. These

Figure 4. Grafana screenshots showing resources utilization by KSim@P1 running in TDAQ tbed cluster

enhancements aim to position Kubernetes as a unified platform for both physics data acquisition and offline-simulation tasks, streamlining operations and improving overall system efficiency.

7 Conclusion

The adoption of Kubernetes for orchestrating the ATLAS EF farm represents a significant step forward in preparing for the HL-LHC upgrade. Kubernetes addresses key challenges such as scalability, fault tolerance, and resource optimization, ensuring the TDAQ system can manage increasing data volumes and complex event processing. Extensive testing has demonstrated Kubernetes' ability to meet the demands of HL-LHC data acquisition. By integrating Kubernetes with the TDAQ Run Control framework, ATLAS is now equipped to handle the challenges of higher luminosity and increased data complexity.

References

[1] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider. JINST 3 S08003 (2008). https://doi.org/10.1088/1748-0221/3/08/S08003

[2] ATLAS Collaboration, ATLAS Trigger and Data Acquisition Phase-II Upgrade Technical Design Report. Tech. rep. CERN-LHCC-2017-020; ATLAS-TDR-029. Geneva: CERN, Dec. 2017 https://cds.cern.ch/record/2285584

[3] A. Kazarov et al, The Controls and Configuration Software of the ATLAS Data Acquisition System: evolution towards LHC Run 3, EPJ Web Conf. 251 (2021). https://doi.org/10.1051/epjconf/202125104019

[4] Kubernetes: Production-Grade Container Orchestration. https://kubernetes.io/

[5] Kubernetes C client library project: https://github.com/kubernetes-client/c/

[6] F. Berghaus et al, Sim@P1: Using Cloudscheduler for offline processing on the ATLAS HLT farm, EPJ Web of Conferences 214 (2019). https://doi.org/10.1051/epjconf/201921407021