ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

AN UPDATE ON THE CERN JOURNEY FROM BARE METAL TO ORCHESTRATED CONTAINERIZATION FOR CONTROLS

T. Oulevey*, B. Copy†, J.-B. de Martel‡, F. Locci, S. Page, R.Rocha, C. Roderick, M. Vanden Eynden§
CERN, Geneva, Switzerland

Abstract

At CERN, work has been undertaken since 2019 to transition from running Accelerator controls software on bare metal to running in an orchestrated, containerized environment. This will allow engineers to optimise infrastructure cost, to improve disaster recovery and business continuity, and to streamline DevOps practices along with better security.

Container adoption requires developers to apply portable practices including aspects related to persistence integration, network exposure, and secrets management. It also promotes process isolation and supports enhanced observability.

Building on containerization, orchestration platforms (such as Kubernetes) can be used to drive the life cycle of independent services into a larger scale infrastructure.

This paper describes the strategies employed at CERN to make a smooth transition towards an orchestrated containerised environment and discusses the challenges based on the experience gained during an extended proof-of-concept phase.

BACKGROUND

The CERN Technical Network

CERN operates several distinct networks serving different purposes. Two of them will be referenced during this paper:

1. The General-Purpose Network (GPN), used for a large number of non-accelerator-related operations.

2. The Accelerator Technical Network (TN), which can be seen as a separate protected network, used solely for on-line accelerator operations.

The TN was created in the late 1990s as a private, independent physical network, dedicated to the operation of the CERN accelerator complex. In 2005, the CNIC (Computing and Network Infrastructure for Controls) working group [1] was established with the goal of reviewing, proposing and putting in place relevant network and security measures to better protect the TN. For security reasons, the provision of central IT services on the TN is an ever-growing challenge, in particular with the emergence of cloud-based licensing policies and remote hosted solutions.

* thomas.oulevey@cern.ch
† brice.copy@cern.ch
‡ jean-baptiste.de.martel@cern.ch
§ marc.vanden.eynden@cern.ch

The CERN Accelerator Data Centre

The CERN Accelerator Data Centre (ADC) is located on the French CERN site in Prévessin-Moëns, next to the CERN Control Centre (CCC) [2]. It hosts around 500 high-performance and high-availability servers, currently running the CERN CentOS7 operating system. In 2023, a project was launched to transition towards RHEL9 in 2024, as the future operating system for accelerator controls.

Since the dawn of the Large Hadron Collider (LHC) era, the ADC operates as a pure bare-metal infrastructure, each server being attributed to equipment groups in charge of the control software of dedicated accelerator sub-systems (*e.g.* cryogenics, radio frequency, beam instrumentation, etc.)

While offering a lot of flexibility to the numerous software development teams, this ADC model is no longer sustainable for reasons such as the lack of optimization and under use of the global computing power, security aspects, a plethora of operational software DevOps practices, wide-spread dependencies towards third-party software solutions, and, last but not least, a lack of agility in terms of evolution.

Platform Engineering

Platform engineering in the context of Kubernetes involves designing, implementing, and maintaining the underlying infrastructure to support containerized applications. This includes configuring and optimizing Kubernetes clusters, managing container orchestration, and ensuring high availability and scalability. It enables development teams to focus on building and deploying applications efficiently within the Kubernetes ecosystem. It also allows system administrators to abstract the underlying infrastructure, making tasks like operating system updates, server hardware replacement, and network changes nearly transparent to the applications and services running on the platform. This abstraction provides a layer of insulation between the infrastructure and the workloads, promoting consistency and ease of management across diverse environments.

This approach and added value may come at the expense of increased complexity and a steeper learning curve for development, DevOps and administration teams, particularly in terms of grasping container orchestration concepts and addressing the associated challenges. System administrators must exercise particular care, to guarantee security and enforce updates, especially during the periodic CERN accelerator Technical Stops.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

CERN'S JOURNEY TOWARDS A KUBERNETES-BASED ACCELERATOR CONTROL SYSTEM

Goal and strategic drivers

The goal is to deploy Kubernetes on the TN, then adapt all relevant controls software systems to run on this platform, instead of bare metal, no later than during CERN's Long Shutdown 3 (2026-2029). The main strategic drivers for doing this, are, in order or importance:

1. To be better equipped for disaster recovery and business continuity, by becoming more agile in terms of hardware management, software deployment and validation.
2. To optimise infrastructure resources in terms of cost and energy.
3. To streamline and align (as much as possible) DevOps practices (both internally and with industry). This is expected to reduce the costs associated with the diverse in-house DevOps practices of today. It will also facilitate the movement of people working on various Controls sub-systems, and the on-boarding of newcomers by adhering to modern, industry practices.

CERN Cloud Offering

A CERN Cloud offering [3], provided by the IT department, already exists since several years and consists of:

- OpenStack as the underlying multi tenancy layer.
- A Kubernetes-as-a-service service, similar to that available from all the major public cloud providers.
- A Registry Service, based on the Harbor product.
- Add-ons to Kubernetes, offering integration with CERN identity management, monitoring, networking, and storage.

The CERN Cloud is only available on the GPN and therefore not usable for the accelerator controls, which run exclusively on the TN. A proof of concept (PoC) project was therefore launched to overcome this limitation.

Kubernetes PoC for Accelerator Controls

The aim of the PoC was to deploy an IT-managed Kubernetes solution on the isolated TN and to onboard a limited, but representative, number of controls software use cases, to:

- Establish a clear understanding of the technical challenges to be overcome to have Kubernetes on the TN.
- Gain experience using the functionalities of an orchestrator and understand the impact on future Controls System Administration Team practices and responsibilities.
- Gain a better understanding of the effort that would be required in terms of adapting application software DevOps practices.

In terms of infrastructure, several key aspects were addressed, such as the network topology, monitoring and logging, security and access control, storage, and performance. The PoC highlighted important differences with respect to the standard IT Kubernetes-as-a-service setup, leading to recommendations for aspects to be further analyzed, in order to transition towards a production-ready platform for accelerator control.

Objectives within the PoC included:

- The setup should be fully hosted on the TN and should remain operational with no connectivity to or from the GPN.
- The setup should consist of computing resources located mainly in the ADC, supplemented by additional resources located in CERN's central IT Data Centre ITDC [4] (physically located on the Swiss CERN site), to support Business Continuity and Disaster Recovery (BC/DR) requirements.
- There should be no critical dependencies on services not hosted on the TN.
- Periodic tasks such as image replication or certificate generation and renewal may require GPN access, but should not prevent normal service operation.

In the following paragraphs, the main Kubernetes components and associated tools will be described.

Control Plane

The control plane is responsible for making global decisions about the cluster, such as scheduling or responding to events. The main components include:

- apiserver: exposes the Kubernetes API and acts as the front end to the control plane.
- etcd: a consistent and highly available key value store, acting as backing store for all cluster data.
- scheduler: watches resources for assignment.
- controller-manager: a daemon that embeds the core control loops shipped with Kubernetes.

All control plane components can be deployed in a highly available mode, with multiple replicas spread across multiple nodes to prevent single points of failure.

Worker Nodes

In the PoC, it was decided to operate Kubernetes worker nodes as Virtual Machines (VM). VMs can be easily provisioned and scaled as needed, making it simpler to add or remove worker nodes based on workload requirements. This flexibility promotes efficient resource utilization.

Each node in the cluster runs a set of components responsible for running pods and providing the Kubernetes runtime environment. The main components include:

- kubelet: ensures that containers are running in a pod when scheduled.
- kube-proxy: a network proxy implementing the Kubernetes Service concept and maintaining a set of network rules on the nodes for pod communication.
- container runtime: the software responsible for running containers, with support for multiple options imple-

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

menting the CRI (Container Runtime Interface) such as *containerd* or CRI-O.

Networking

Kubernetes networking addresses the following requirements:

- Containers within a pod can communicate with each other via loopback, as they share the same networking namespace.
- Cluster networking provides communication between different pods.
- The Service API allows exposure of an application running in pods so that it is reachable from outside of the cluster, with Ingress provisioning extra functionality for HTTP and Websocket networking (Layer 7).
- Services can also be configured to only be exposed within the cluster.

Every pod gets its own unique cluster-wide IP address, avoiding the need to create explicit links between pods. In most cases, users never need to deal with mapping container ports to host ports. In a similar way, each service also gets an assigned IP address which can be either local to the cluster or managed by an external load-balancing service, allowing external access via Layer 4 (L4). In both cases, Kubernetes creates DNS records that can allow contact with consistent DNS names instead of IP addresses. The networking is abstracted by the Container Networking Interface (CNI) for which multiple implementations are available with slightly different features. Examples include Flannel, Calico and Cilium.

For the PoC, a new "tn1" network region was set up with two Availability Zones (AZ), corresponding to the aforementioned ITDC and ADC locations. These AZs are exposed to end users and can be used to create clusters with node groups in each AZ for improved availability. This information is also exposed to the Kubernetes clusters and can be exploited by workloads to spread replicas across different failure domains, using "topology constraints".

Container Registry

The CERN IT department provides a registry service for containers, Helm charts [5], and any other OCI artifacts. It relies on the upstream Harbor Project [6] (a Cloud-Native Computing Foundation - CNCF [7] - graduated project) and adds the following features to the basic artifact storage and handling:

- Multi-tenancy and Role-Based Access Control (RBAC)
- Project quotas
- Security and vulnerability analysis with external scanner integration
- Content signing and validation
- Automated replication across OCI registries (also with non-Harbor registries)
- Pull-through caches to external registries
- An extensible API and Web UI with external scanners

As part of the PoC, a new instance of the registry service was deployed inside the TN, and made available to both Kubernetes and other, non-orchestrated, containerized workloads used within the accelerator control system.

To comply with the TN policy and operational concerns regarding the container image lifecycle, a set of automated replication rules was defined, where all new container image tags pushed to the GPN registry are, transparently to users, automatically replicated to the TN instance. This means in practice, that in order to make a container image available on the TN via the new registry service, it must first be made available in the GPN registry, from where it can be replicated to the TN. Such replication of images is only done on demand and via an approval process. To support this process, the replication functionality is integrated with a dedicated system that is used to manage which images should be made available to the accelerator control system. This provides an API and web-application, via which, users are able to request new images to be made available on the TN, and refresh pre-approved images at any time.

PILOT PROJECTS AND APPLICATIONS

Several representative accelerator control system applications were selected as pilot services to run on the new PoC platform. The software development teams adapted their applications and DevOps processes to leverage the new platform. Regular meetings were held between all participants, to share progress and problems, exchange knowledge, and agree on next steps. For each pilot service, an effort was made to document the changes made to configuration and deployment aspects, the integration needed with external services, and any notable differences with respect to the current bare-metal-based deployments. The intention of this was to build up a document defining best practices required by other controls software services in the future, in the context of the eventual move of all applicable applications to run on Kubernetes. In the following paragraphs, some of the most representative use cases will be described and the challenges that needed to be addressed will be discussed.

Controls Middleware (CMW)

The Controls Middleware (CMW) [8] powers the client-server communications within the CERN accelerator control system. It uses an underlying protocol known as RDA3 for exchanging accelerator device data.

The following tasks and tests were initially planned:

- Run the CMW Directory Service and RDA3 servers (written both in C++ and Java) on Kubernetes and validate their seamless interactions with RDA3 clients running inside and outside the Kubernetes cluster.
- Validate the correct functioning of rolling upgrades and fail-overs of the CMW Directory Service.
- Test the feasibility of making rolling upgrades and fail-overs of RDA3 clients and servers, and identify any limitations in the current CMW implementation.
- Explore the use of Kubernetes services and APIs for deployment, monitoring, logging, etc. and assess which in-house developed DevOps services could be replaced.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

Configuration and Deployment The CMW directory service was already running in a clustered and redundant configuration outside of Kubernetes, using the JGroups messaging framework to manage communication between replicas of the service. It was already containerized for integration-testing purposes; but before the PoC, this container image had never been used inside a Kubernetes cluster.

One significant challenge was to get JGroups to locate and communicate with the other peers in the Directory Service cluster. As JGroups requires incoming and outgoing IP addresses to be identical, it was not possible to configure static lookup of replicas with one Kubernetes service per replica, because incoming IPs are pod IPs and outgoing IPs are service IPs. Fortunately, a JGroups plugin, KUBE_PING [9], handles peer autodiscovery inside a Kubernetes cluster.

The CMW Directory Service achieves redundancy with client-side load balancing. This means that clients need to access each replica separately, and not through a server-side load-balancer. To achieve this on Kubernetes without heavy changes to the application architecture, a first approach was tried using a separate Kubernetes deployment per replica. However, this prevented the use of Kubernetes rolling deployment features. On the other hand, configuring Kubernetes services to only route traffic to the local node replica (externalTrafficPolicy: Local) gives slow feedback to clients that a replica is not available. This is because connection attempts need to time-out, before clients notice that the replica is down and try to connect to another replica. A simple solution was found by leveraging the host networking feature of Kubernetes, which is similar to our current bare-metal offering. All of the above networking approaches rely on controlling on which nodes the replicas are scheduled, which is not standard Kubernetes practice. This was achieved with a simple service-specific node tag (cmw.cern.ch/cmw-directory-server) and a corresponding nodeSelector. The CMW Directory Service is a Spring Boot application, which comes with pre-existing liveness and readiness probes which made the creation of a Kubernetes Deployment a trivial task and enabled rolling upgrades immediately. ArgoCD was installed and configured to synchronise the cluster configuration with the Git repository. A first approach with Helm charts was explored, but for simplicity, Kubernetes templates were chosen as a solution.

Challenges New dependencies have been introduced into the Git service, which is hosted on the GPN. As a result, ensuring the availability of the Git repository has become a critical concern for the TN-based deployment model. However manual configuration changes can be applied to the Kubernetes cluster in case of an emergency to mitigate network or Git service incidents. The CMW Directory Service also requires the injection of the CERN Oracle database configuration (tnsnames.ora managed by the CERN IT department) as a Kubernetes configmap to keep it up-to-date.

Post-Mortem Analysis (PMA)

The Post-Mortem service is a part of the mission-critical CERN Machine-Protection system. It collects information from many accelerator devices and other systems in the event of a beam dump or operational issue, then a Post-Mortem Analysis (PMA) sub-system automatically analyses the data, using analysis modules developed by domain experts. Afterward, the PMA system provides the operation teams and equipment experts with the analysis results. The PMA output indicates if it is safe to resume operations and this information is consumed by interlock systems which can prevent beam injection. Five PMA systems are used daily by LHC and Super Proton Synchrotron (SPS) operations and each has a master process (the PMA "Engine"), that drives the execution of several analysis modules, in the right order. Today, a PMA system is a monolithic application, in the sense that the PMA engine and the analysis modules all run within the same JVM process. This has two consequences. Firstly, a single misbehaving analysis module can kill the whole process. Secondly, whenever a domain expert provides a new version of their analysis module, the whole PMA application needs to be manually rebuilt and redeployed by the engineers responsible for PMA. The main goals for this use case within the PoC were to:

1. Investigate the possibility of running analysis modules in separate processes on Kubernetes.

2. Provide a self-service solution for PMA analysis module developers (i.e. domain experts) to update their analysis modules.

Configuration and Deployment A major part of the development effort focused on splitting the monolithic implementation into a distributed architecture. This involved isolating analysis modules and exposing them over the network. It also required redesigning the analysis workflow, and in particular, the interactions between the analysis engine and the analysis modules. Unlike the original monolithic implementation, the updated analysis modules now receive a single, self-contained request with all of the information needed for the analysis. They then stream events back to the engine, as the analysis progresses. This allows to move all states from the analysis modules to the engine, which in turn, enables rolling updates of the now stateless analysis modules. This significant architectural change also helps to achieve the aforementioned goal of domain experts being able to deploy new versions of their analysis modules in a self-service manner. The new "self-service" module (see Fig. 1) is based on Gitlab CI pipelines. Gitlab repositories with analysis modules now have a new CI configuration which, when triggered, builds a new container image containing the analysis module. This container image is replicated across various container registries, and made available on the TN. Finally, the CI pipeline deploys the new analysis module into the staging environment of the PMA application. Thanks to this, users are able to deploy and validate their changes autonomously,

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

yet in a safe way, without any intervention from the PMA system maintainers.

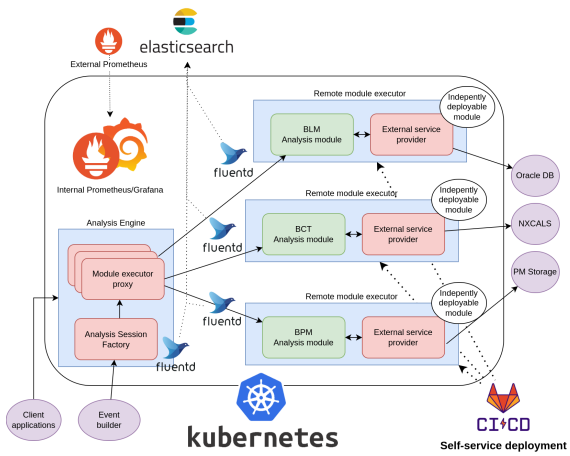

Figure 1: The "self-service" PMA module architecture.

Challenges While this new Kubernetes-based PMA implementation has many advantages and solves the aforementioned goals, it also brings some additional challenges. Firstly, the new approach is more complex than the previous bare-metal deployments, where services run as simple Java processes, on a dedicated machine, and can be monitored and diagnosed with standard Unix and Java tools. The large number of components involved in Kubernetes makes troubleshooting more complex, especially when it comes to network related issues. The way that Kubernetes networking works also does not play well with JMX (Java Management eXtensions), a Java-specific tool which allows engineers to both monitor and manage Java processes at runtime. However, Kubernetes-compatible drop-in replacements exist for both metric acquisition (Prometheus [10] JMX exporter or Spring Boot actuator) and management operations (Jolokia, Hawtio).

It also takes longer to deploy a new version of the application compared to the bare-metal setup, due to additional CI jobs and image replication delays (2-3 minutes) which could probably be optimized. Running CPU-bound workloads on virtualized nodes incurs a 30% performance penalty, and the performance can vary widely over time. Preliminary tests showed that running on bare-metal Kubernetes nodes (as opposed to running on virtual nodes) yields performance much closer to pure bare-metal deployment.

Controls Configuration Data API (CCDA)

The Controls Configuration Service (CCS) [11] centralizes the configuration of the entire control system. Its CCDA (Controls Configuration Data Access) API [12] provides, amongst other things, REST-based services to retrieve controls configuration data. The whole CCDA system is based on a classical three-tier architecture, consisting of a client layer, a server layer and a database layer. The server layer consists of a Spring Boot application hosting stateless CCDA service instances to handle incoming user requests. A Spring Gateway application acts as the main entry point, redirecting user requests to appropriate server API instances. Each deployed API instance registers itself with Eureka, which maintains a registry of all available API instances and provides service discovery functionality. Eureka also provides load balancing to optimise traffic. Requests sent to the CCDA service are either to retrieve or manipulate data from an underlying Oracle database.

Configuration and Deployment The Kubernetes cluster architecture is set up with one service per business application (i.e the CCDA). To replicate the existing production bare-metal setup, two pods have been initiated for each application. These pods are where the CCDA service instances run, handling incoming requests, which are properly routed to them based on suitable rules on the ingress node. A Zipkin [13] service was also integrated to gather CCDA telemetry data. Again, routing rules were configured in the ingress node, this time to make Zipkin accessible from the network outside of the Kubernetes cluster.

Helm was used for the deployment, because multiple services (e.g. CCDA and Zipkin) have the same application structure making it possible to standardise the configuration via templates. The API server instances are automatically deployed on the Kubernetes cluster via a GitLab CI pipeline. For this purpose, the Kubernetes configuration and other passwords are being saved as GitLab variables. The pipeline first builds the container images, pushes them into the GitLab registry and then deploys the application on the cluster using Helm.

Integration with External Services The CCDA services use a logger module, which is included in the container image as part of the deployed Jar. Logs are then sent to an external Elasticsearch service and can be visualised through Kibana [14] dashboards. The Oracle database is managed outside of the Kubernetes cluster, but the pods were able to connect to it seamlessly, without requiring any further setup.

Challenges While response times were similar between bare metal machines and Kubernetes, some slight differences were observed in the number of requests that timed out. The higher number of requests that took longer than expected can be attributed to the test cluster having less computing resources than the bare metal machine. Better performance would be expected when deploying to a cluster with more resources available. Access to logs and network debugging need more specialized tools and understanding within the Kubernetes ecosystem and CCS service managers need to embrace the new approaches. The CCDA also needs to be available on the GPN, something which is not yet possible without a system administrator intervention.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

FUTURE WORK REQUIRED TO REACH PRODUCTION

The PoC was highly successful resulting in:

- The deployment and stable running of multiple Kubernetes clusters on the TN.
- Adaptation deployment and running of several representative controls software services on the aforementioned clusters.
- Valuable experience gained in this new eco-system, from both system administration and developer perspectives, including changes to DevOps practices.
- Comprehensive documentation of adaptations required within the control system portfolio in order to use and profit from the power of Kubernetes.
- Identification of areas requiring further investment, in order to take the next step of transitioning from a PoC to a production-ready platform.

The following paragraphs will elaborate on the last item in the above list, by describing areas where further work is required or on-going.

Storage The Kubernetes clusters have both local storage (volatile) and persistent storage. The local storage is using the cluster nodes, with no guarantees of resilience. The persistent storage relies on an NFS (version 3) backend server, managed by the accelerator controls system administrators. Investigations are on-going to study hyper-converged solutions such as OpenEBS [15] or Longhorn [16] to improve NVMe usage (*i.e.* by dropping RAID1 support) and reduce costs.

Monitoring and Logging The Control Plane cluster includes its own Prometheus instance that collects metrics from all nodes (physical and virtual), as well as availability metrics for the different Control Plane components of OpenStack and the Kubernetes Service. An add-on is provided for end-user managed clusters that sets up a dedicated Prometheus instance. By default all metrics exist and are kept only inside the cluster. They are accessible to everyone with cluster access. Additional work is foreseen to allow the centralized collection, aggregation and long-term storage of all metrics, in the pre-existing accelerator control system monitoring service.

Hardware Infrastructure Considerations An investigation is on-going into the management of all server metadata (including location and network topology) using OpenDCIM [17]. Quad Servers (4 systems in 2U rack space), are the standard hardware infrastructure used in the ADC for running controls software service. Spreading replicas of Kubernetes nodes across multiple quads, and across multiple racks would minimise the impact of hardware interventions on the corresponding Kubernetes deployed services.

Although end users should remain unaware of the underlying servers running Kubernetes clusters, there is still a case for keeping bare-metal nodes within clusters to address performance challenges and to accommodate specific hardware requirements like GPUs or accelerator timing system receiver cards.

Access Control Authentication and authorisation via OIDC/OAuth2 is available on the GPN via the CERN Single-Sign-On (based on Keycloak [18]) and a custom application portal developed at CERN. Roles can be used to define Role-Based Access Control (RBAC) policies to manage access to the different resources and APIs. This feature has not been configured during the PoC phase but will be soon investigated.

It is expected that the TN infrastructure will benefit from ongoing developments by the IT department for the GPN-based Kubernetes clusters. Of particular relevance are runtime checks in all clusters with tools like Falco [19], which allow to define policies and alerts regarding: unexpected processes and shells launched in existing containers; unexpected inbound or outbound network requests etc.

Credentials And Secrets Management A dedicated sub-project is foreseen, to analyse, prototype, and then implement a secure vault service to provide secrets and key rotation to Kubernetes clusters. It is expected that this will also support legacy, non-Kubernetes deployed service and configuration management tools.

Other Cyber-Security Considerations By design, Kubernetes solves many cyber security challenges found in non-orchestrated environments (*i.e.* container isolation, resource segregation and scaling, secret management etc...). However, it also introduces new potential risks:

- **Visibility of dynamic resources** : As resources such as pods, storage and logs can come and go within a cluster dynamically, it can become more difficult to track and monitor any related problematic or malicious activity.
- **Misconfiguration and cluster-level privilege issues**: The powerful declarative nature of cluster configuration can, in certain cases, introduce misunderstandings or leave room for interpretation that results in increased risk exposure. For instance, should a certain environment variable rather be handled as a secret? Should a certain service account be given the privileges to control CPU consumption of a given process?
- **Supply-chain security** : Containerization brings the opportunity of having a minimal number of images for the different services and applications, reducing significantly the number of packaged dependencies and with it, the potential attack vectors. A centralized registry also improves the ability to identify and track vulnerable applications, when compared to traditional deployments. Further exploration of the growing ecosystem around

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TH2A003

supply chains for containerized applications is a unique opportunity in this area.

Out of the three areas of risks listed above, securing the supply chain is the one that does not really fit in the mandate of cluster infrastructure administrators [20]. In the context of accelerator controls (and not only at CERN), the detection and removal of a threat present in an operational Kubernetes cluster image can only be delegated with safe certainty to the application developers that produce the image. Software Bill Of Materials (SBOM) standards offer a common language to everyone involved to produce a complete inventory of operational software. The usage of SBOM tools is a clear strategy towards increased transparency, accountability and compliance. While SBOM usage is still in its initial phases at CERN, it will undoubtedly be required in the near future to deploy containers to operational Kubernetes clusters on the TN.

CONCLUSION

Overall, the PoC resulted in a successful deployment of several accelerator controls applications and services, running on the Kubernetes orchestrated container platform installed in the CERN Accelerator Data Centre and connected to the Technical Network.

Work on the pilot projects within the PoC activity, highlighted the opportunity to streamline software DevOps practices and remove dependencies on in-house developed DevOps solutions. The pilots also led to ways of simplifying controls software service deployments and upgrade scheduling, as well as increasing resilience.

The main challenges reported included a steep learning curve, linked to the complexity brought by the additional layers and the work required to integrate with existing debugging and troubleshooting tools.

Aspects such as robust Secret Management and use of modern network concepts on the Technical Network need to be implemented in order to have a common viable solution to run all CERN controls software services on the Kubernetes platform.

The next step on the journey has already started, and work is on-going to address the open points from the PoC and prepare a production ready platform by the first half of 2024. The subsequent steps will be to plan and then carry out the adaptation and migration of all of the controls software services towards the new Kubernetes platform, over the next five years.

ACKNOWLEDGEMENTS

This journey towards the adoption of Kubernetes for Accelerator Controls at CERN would not have been possible without the participation of many people from both the CERN Accelerator Sector and IT Department. We thank everyone for their effort and apologize to anyone we missed below. Vito Baggiolini, Karim Ben Othman, Jean-Baptiste De Martel, Felix Ehm, Diana Gaponcic, Stefanno Gennaro, Romain Gorbonosov, Diogo Guerra, Eric Grancher, Enzo Genuardi, Kevin Kessler, Nuno Laurentino Mendes, Stephane Lalouette, Arkadiusz Podkowa, Marek Seda, Giuseppe Simonetti, Spyridon Trigazis, Manuel Treu, Karl-Andreas Turvas, Robert Vasek, Martin Voelkle, Nikos Tsipinakis.

This paper is dedicated to the memory of our colleague, Remi Voirin, whose passing occurred during the course of this initiative.

REFERENCES

[1] CERN CNIC (Computing and Network Infrastructure for Controls) working group, https://indico.cern.ch/category/691/

[2] CERN Controls Centre (CCC), https://home.cern/tags/control-centre

[3] The CERN Cloud, https://clouddocs.web.cern.ch/overview/overview.html

[4] The CERN IT Data Centre, https://home.web.cern.ch/science/computing/data-centre

[5] Helm - The Kubernetes Package Manager, https://helm.sh/

[6] Harbor, https://goharbor.io

[7] CNCF - The Cloud Native Computing Foundation, https://www.cncf.io/

[8] J. Lauener *et al.*, "How to design & implement a modern communication Middleware based on ZeroMQ", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 45–51. doi:10.18429/JACoW-ICALEPCS2019-WEPHA163

[9] Kubernetes discovery protocol for JGroups, https://github.com/jgroups-extras/jgroups-kubernetes

[10] Prometheus - Monitoring system and time series database, https://prometheus.io

[11] L. Burdzanowski *et al.*, "CERN Controls Configuration Service – A challenge in usability", in *Proc. ICALEPCS'17*, Barcelona, Spain, Oct. 2017, pp. 159–165. doi:10.18429/JACoW-ICALEPCS2017-TUBPL01

[12] B. Urbaniec *et al.*, "Developing Modern High-Level Controls APIs", presented at the ICALEPCS'23, Cape Town, South Africa, Oct. 2023, paper TH2AO04, this conference.

[13] Zipkin - a distributed tracing system, https://zipkin.io/

[14] Kibana - a browser-based analytics and search dashboard, https://www.elastic.co/fr/kibana

[15] OpenEBS - Kubernetes storage simplified, https://openebs.io

[16] Longhorn - a distributed block storage system, https://github.com/longhorn/longhorn

[17] OpenDCIM, a web based Data Centre Infrastructure Management, https://opendcim.org

[18] Keycloak - Open Source Identity and Access Management, https://keycloak.org

[19] Falco - Detect security threats in real time, https://falco.org/

[20] B. Copy *et al.*, "Protecting your controls infrastructure supply chain", presented at the ICALEPCS'23, Cape Town, South Africa, Oct 2023, paper MO4BCO03, this conference.