# AUTOMATED SOFTWARE WORKFLOW FOR ACCELERATOR CONTROL SYSTEMS USING CONTAINERIZATION

A. Aljadaa*[1], A. Ahmad[1], A. Abbadi[1], M. Alzubi[1,2], R. Khrais[1]
[1]SESAME Synchrotron, Allan, Jordan
[2]Princess Sumaya University for Technology, Amman, Jordan

*Abstract*

A new software workflow has been developed and deployed at SESAME that solves several issues large facilities may face in their lifecycle. The workflow includes containerizing software in independent units, orchestrating those units from a central manager, managing installation, release, and rollback of deployments, and automating the process through a continuous deployment pipeline. This workflow reduces development and deployment time while eliminating human error.

## INTRODUCTION

Managing hundreds of Input/Output Controllers (IOCs) and Graphical User Interfaces (GUIs) in a multi-network facility is challenging due to various sources of failure: hardware failures in servers and networks, operating system issues like high CPU and memory usage, and application issues such as crashes and memory leaks. Traditional manual deployment workflows compound these challenges by introducing human error.

This paper presents an automated workflow implemented at SESAME that containerizes software, orchestrates deployments via Kubernetes, manages releases with Helm, and automates the entire process through continuous deployment. Similar approaches have been successfully implemented at other synchrotron facilities [1]. This approach improves operational efficiency and ensures deployment consistency.

The work focuses on IOCs and GUIs built on the Linux operating system. This paper discusses the following stages: containerization, orchestration with Kubernetes, release management with Helm, and continuous deployment automation.

Figure 1 illustrates the workflow, which is discussed in detail in the following sections.

This workflow has been deployed in production for all machine IOCs at SESAME Synchrotron Light.

## CONTAINERIZATION

To address environment inconsistencies between development and production, IOCs and graphical user interfaces are containerized using Docker [2], which was selected for its widespread adoption and robust ecosystem. The containers are configured to run on the host network for direct network access.

All images are stored and versioned in a private Docker registry. Image versioning facilitates rapid rollbacks in case

* amro.aljadaa@sesame.org.jo

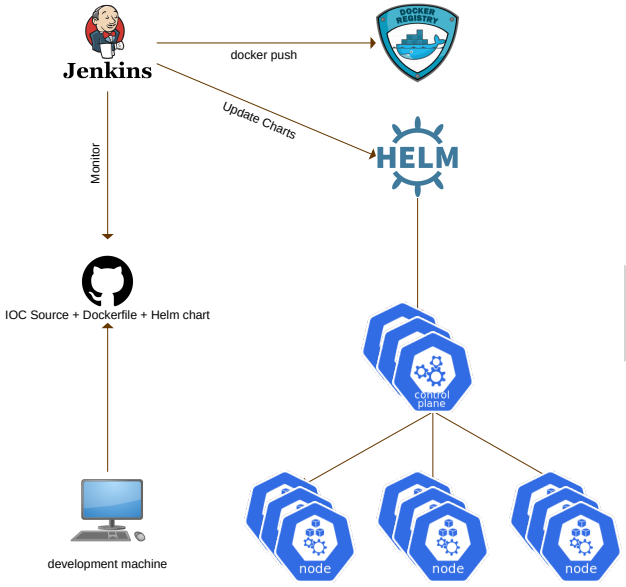


Figure 1: Complete workflow from development to production deployment.

of deployment issues. To optimize these images, a multilayer approach is followed when building Docker images. This modularizes the development process by creating images for common dependencies, thereby reducing development time and standardizing the work. Moreover, this approach reduces the final image size by keeping only the binaries required to run the IOC. The size is reduced to approximately 25 MB instead of several hundred megabytes or even gigabytes.

Figure 2 illustrates the multilayer approach, demonstrating both the modular organization of dependencies and how intermediate build layers are discarded to produce a lightweight Alpine runtime image.

While containerization provides isolation and consistency, centralized management of these containers requires orchestration.

## ORCHESTRATION

To manage Docker IOCs from a central node, Kubernetes [3] was selected as the container orchestration tool.

Kubernetes plays a vital role in IOC management, acting as a central manager that provides a global view of the entire system and aggregates logs. Two complementary tools are employed for system management: the Kubernetes dashboard and k9s [4]. The dashboard provides a web-based graphical interface for interactive monitoring and control, while k9s delivers a terminal-based UI with several advantages, including graphics independence that enables

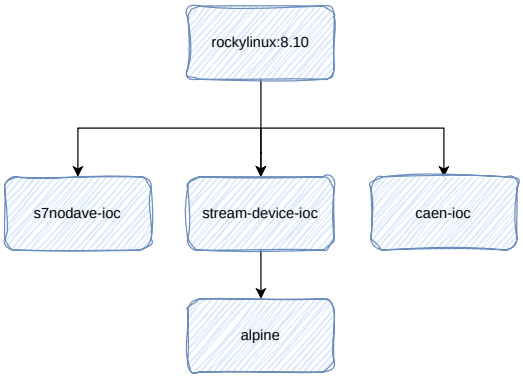


Figure 2: Multilayer Docker image build process.

remote management over low-bandwidth connections, faster keyboard-driven navigation, real-time cluster monitoring, and native Helm release management capabilities. Both tools enable manual IOC control and shell access to pod containers.

Beyond monitoring and management, Kubernetes provides sophisticated deployment capabilities. Affinity rules are used to instruct the orchestrator where to deploy the IOCs, which is particularly useful for devices that require high bandwidth, such as high-speed cameras.

Data persistence across IOC restarts and power cuts is critical. Persistent volumes and persistent volume claims are employed for autosave functionality.

Kubernetes also improves service availability by automatically redeploying the IOCs from failed servers to healthy ones. If an IOC fails due to an application issue, Kubernetes automatically restarts the IOC.

Finally, Kubernetes enables resource management by applying CPU and memory constraints to pods. The Kubernetes scheduler automatically optimizes pod placement across nodes based on available resources.

While Kubernetes provides robust orchestration capabilities, managing deployment versions and rollbacks across hundreds of IOCs requires additional tooling. Helm addresses this need by providing package management and version control.

## HELM

Helm [5] is utilized for managing Kubernetes deployment releases, serving as a package manager that simplifies the complexity of Kubernetes deployments. It enables fast updates and rollbacks through version management, allowing deployment of IOC instances to the cluster and management of multiple IOC versions. Helm's templating system provides consistency across deployments while allowing fine-grained customization for individual IOC requirements.

Each Helm release maintains a complete version history, enabling rollback to any previous state if issues arise. This release management capability is particularly valuable when managing hundreds of IOCs, as it ensures deployment consistency and simplifies troubleshooting. The versioning system provides a clear audit trail of all deployment changes, enabling easy identification of the currently running IOC version.

```
chart/
    Chart.yaml   — name · description · version
    values.yaml  — IOC name · node selector · image
    templates/
        deployment.yaml
        persistentVolume.yaml
        persistentVolumeClaim.yaml
```

Figure 3: Helm chart structure and components.

All Kubernetes resources required to deploy an IOC are packaged in a Helm chart according to Figure 3. A typical chart contains several components organized in a standardized directory structure. The Chart.yaml file contains metadata including chart name, description, and version. The values.yaml file contains configuration parameters that shape the deployment template, serving as the single source of truth for deployment customization.

The values.yaml file defines key parameters for each IOC deployment, including the IOC name, node selector rules for affinity-based placement, Docker image specifications (repository, name, tag, and pull policy), and behavioral flags such as automatic restart on failure. This structured approach allows a single chart template to be reused across different IOCs by simply modifying the values file, promoting consistency while enabling customization.

The deployment template utilizes Helm's templating syntax to reference values dynamically. It defines the Kubernetes deployment resource with specifications for pod metadata, labels, and container configuration. The template implements host networking to enable direct network access for IOC communication. Node affinity rules are configured using the nodeSelector values, ensuring IOCs are scheduled on appropriate nodes based on hardware requirements.

Data persistence is handled through dedicated PersistentVolume and PersistentVolumeClaim templates. The PersistentVolume template configures NFS-backed storage with appropriate capacity and access modes, while the PersistentVolumeClaim template requests storage resources and binds to the volume. These are mounted into the container at the autosave directory, ensuring IOC state persistence across pod restarts and node failures.

## CONTINUOUS DEPLOYMENT

With containerization, orchestration, and package management established, the final component automates the entire build-to-deployment cycle. Jenkins [6] is employed as the continuous deployment platform for this purpose. The pipeline is triggered by filesystem changes detected through FSTrigger, which monitors Git push events to the repository. The deployment workflow proceeds only when a Git tag is

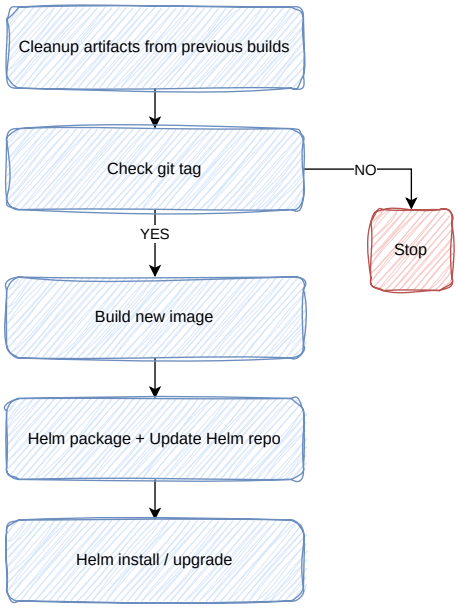


Figure 4: Continuous deployment pipeline.

detected on the latest commit; regular commits without tags bypass the deployment process, ensuring controlled releases to production.

The deployment process is divided into stages as shown in Figure 4. The pipeline begins by cloning the IOC's Git repository and checking for the presence of a Git tag. If no tag is detected, the build is skipped. When a tag is present, the pipeline proceeds through the following stages:

The first stage builds the Docker image and pushes it to the private Docker registry. The second stage packages the Helm chart and transfers it to the Helm repository server, where the repository index is updated to make the new chart version available. The final stage performs either a Helm installation for new IOCs or an upgrade for existing deployments.

This automated pipeline significantly improved deployment speed across all target machines while eliminating human errors such as incomplete deployments and inconsistent configurations. Build failures are immediately reported via email with detailed logs, enabling rapid issue resolution. The pipeline ensures that every deployment follows the same standardized process, guaranteeing consistency across hundreds of IOCs.

## CONCLUSION

This paper presented a comprehensive automated workflow for managing IOCs at SESAME using containerization, Kubernetes orchestration, Helm package management, and continuous deployment. The system has been successfully deployed in production for all machine IOCs, significantly reducing deployment time and eliminating human errors associated with manual workflows. This approach demonstrates the benefits of adopting modern industry-standard tools in large scientific facilities.

## REFERENCES

[1] G. Knap, T. M. Cobb, Y. Moazzam, U. K. Pedersen, and C. J. Reynolds, "Kubernetes for EPICS IOCs", in *Proc. 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21)*, Shanghai, China, Oct. 2021, pp. 835–838. doi:10.18429/JACoW-ICALEPCS2021-THBL04

[2] Docker Inc., "Docker: Enterprise Container Platform", https://www.docker.com

[3] Cloud Native Computing Foundation, "Kubernetes: Production-Grade Container Orchestration", https://kubernetes.io

[4] F. Galiana, "K9s: Kubernetes CLI to Manage Clusters in Style", https://k9scli.io

[5] Helm Community, "Helm: The Package Manager for Kubernetes", https://helm.sh

[6] Jenkins Community, "Jenkins", https://www.jenkins.io