



Human-Centric Orchestration Interfaces for Industry 5.0: A Prototype-Based Usability Study of Containerized Microservices Management

Muhammad Usman, Bilal Maqbool & Muhammad Shahid Iqbal

To cite this article: Muhammad Usman, Bilal Maqbool & Muhammad Shahid Iqbal (16 Jul 2026): Human-Centric Orchestration Interfaces for Industry 5.0: A Prototype-Based Usability Study of Containerized Microservices Management, International Journal of Human-Computer Interaction, DOI: [10.1080/10447318.2026.2701961](https://doi.org/10.1080/10447318.2026.2701961)

To link to this article: <https://doi.org/10.1080/10447318.2026.2701961>



© 2026 The Author(s). Published with license by Taylor & Francis Group, LLC



Published online: 16 Jul 2026.



Submit your article to this journal [↗](#)



Article views: 179



View related articles [↗](#)



View Crossmark data [↗](#)

Human-Centric Orchestration Interfaces for Industry 5.0: A Prototype-Based Usability Study of Containerized Microservices Management

Muhammad Usman , Bilal Maqbool  and Muhammad Shahid Iqbal 

Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden

ABSTRACT

The transition towards Industry 5.0 can leverage Kubernetes-driven computing infrastructure to support cyber-physical systems and IoT-based automation of containerized microservices. However, usability challenges of Kubernetes (e.g., fragmented workflows and syntax-sensitive configuration languages) hinder effective human-machine collaboration. To address these challenges, we present a web-based orchestration system prototype that streamlines containerized workloads management through guided workflows, multimodal features, and real-time monitoring. By reducing dependence on command-line expertise and providing a user-friendly interface, the prototyped system enhances the reliability and transparency of interactions between humans and complex computing platforms. We evaluated the prototyped system through a mixed-methods usability study including task analysis, the System Usability Scale (SUS), and interviews. Results demonstrate significant improvements in usability, with SUS score of 77.25 indicating “Good” overall usability. Overall, these findings show that usability-centered orchestration interfaces can strengthen human trust, improve collaboration, and accelerate the adoption of cloud-native platforms as key enablers of Industry 5.0.

KEYWORDS

Industry 5.0; human-computer interaction; container orchestration; Kubernetes; cyber-physical systems

1. Introduction

A shift toward Industry 5.0 is mainly characterized by a strong focus on digitalization, human-centricity, and intelligent automation in manufacturing systems (Ghobakhloo et al., 2024; Patil, 2025; Sharma et al., 2024). Central to this paradigm is the collaboration between humans and complex digital infrastructures, where users with different roles and expertise levels (e.g., operational roles like plant operators, supervisory roles like production managers, and engineering roles like automation engineers) interact with cyber-physical systems (CPS) and digital twins that increasingly depend on scalable and reliable computing infrastructures (Jamil et al., 2024). In this context, containerized microservices have emerged as the de facto standard for deploying applications over distributed computing infrastructures (Chahed et al., 2023). These microservices aim to provide portability, rapid deployment capabilities, and support for diverse industrial applications, such as smart factories and Smart Cities (Khanh et al., 2023; Usman et al., 2022).

Container orchestration platforms exemplified by Kubernetes play a significant role in managing containerized workloads across distributed computing infrastructures (Eyvazov et al., 2024; Queiroz et al., 2024). For instance, within an advanced manufacturing system, Kubernetes-based orchestration can enable the execution and integration of heterogeneous services, ranging from digital twins to AI-enabled decision support systems. Despite these powerful capabilities, Kubernetes itself introduces substantial usability challenges that hinder effective human-machine collaboration in industrial settings. In

CONTACT Muhammad Usman  muhammad.usman@kau.se  Department of Mathematics and Computer Science, Karlstad University, Karlstad, Sweden

© 2026 The Author(s). Published with license by Taylor & Francis Group, LLC

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. The terms on which this article has been published allow the posting of the Accepted Manuscript in a repository by the author(s) or with their consent.

practice, users across operational, supervisory, and engineering roles are required to interact with fragmented workflows, complex Yet Another Markup Language (YAML) specifications, and low-level command-line interfaces. Such interaction models are misaligned with Industry 5.0 principles, where technology should adapt to human needs rather than requiring users to adapt to technical complexity.

Existing container orchestration platforms generally require a high level of prior IT expertise and lack interfaces tailored for users with heterogeneous responsibilities. For example, the default Kubernetes dashboard presents a cluttered interface with numerous options, many of which are irrelevant or pose a security risk when provided to non-expert IT users. This results in cognitive overload and increases the risk of misconfigurations that may negatively affect safety-critical industrial workloads. At the same time, engineering users often require fine-grained control, configurability, and detailed observability to manage complex deployments when experimenting or migrating applications to production environments. Supervisory users who are positioned between these two extremes require visibility and controlled interaction rather than low-level configuration capabilities. However, the current user interfaces lack the ability to explicitly distinguish among heterogeneous user groups, highlighting a broader gap in HCI for Industry 5.0, where usability, transparency, and role-tailored abstractions are essential.

To address these challenges, this study introduces a web-based orchestration system prototype design to simplify interaction with Kubernetes-based compute infrastructures by explicitly accounting for users with different levels of expertise. The proposed system covers the entire workload lifecycle (templating, deployment, monitoring, and cleanup) through guided workflows and adaptive interfaces tailored to these role categories. By reducing reliance on YAML specifications and Command-line tools, the system lowers the entry barrier for non-expert users, while still preserving advanced functionality for engineering users. Integrated real-time monitoring is provided to further enhance transparency and supports informed operational decision-making across industrial roles.

We also conducted a comprehensive mixed-methods usability study to evaluate the effectiveness of the proposed prototype. The study involves task-based workflows, quantitative analysis of task completion, and observations, followed by the System Usability Scale (SUS) survey and user interviews. The results demonstrate measurable improvements in usability and efficiency, with an overall SUS score of 77.25, indicating good usability.

In summary, the key contributions of this study are as follows:

- Design and prototype a simplified orchestration interface that enhances usability and reduces complexity in managing containerized workloads.
- Design and conduct a structured usability study involving participants from diverse technical backgrounds having prior experience of working with containerized applications and orchestration systems.
- Provide empirical insights into the role of human-centric design in improving trust, transparency, and adoption of orchestration platforms as enablers of Industry 5.0.

The remainder of this article is organized as follows. [Section 2](#) analyzes container orchestration tools and discusses their implications for usability-oriented interface design. [Section 3](#) provides an overview of the proposed system design and implementation, followed by the usability study design in [Section 4](#). [Section 5](#) presents the findings of the usability study, followed by a discussion of its shortcomings and iteratively enhanced system interfaces in [Section 6](#). Finally, [Section 7](#) concludes the article with future directions.

2. Containerized platforms usability challenges and requirements

In this section, we examine the contemporary landscape of container orchestration by focusing on the usability implications for users (i.e., those experimenting, developing, and deploying containerized applications) with varying levels of technical expertise. While container orchestration is widely recognized as an enabler of cloud-native systems, it is also gaining increasing popularity across industrial applications, such as digital twins (Borsatti et al., 2024). In these industrial domains, usability is not merely a developer productivity issue but also a matter of ensuring reliable, transparent, and safe

human-machine collaboration. By analyzing existing studies and tools, we identify key gaps in onboarding complexity, interface design, role-based access, and performance-tuning support. From this analysis, we identify critical requirements for developing human-centered orchestration interfaces that can support heterogeneous user groups in Industry 5.0.

2.1. Research focus of existing studies

Container technologies have become an integral part of modern distributed and cloud-native infrastructures (Deng et al., 2024), with Kubernetes emerging as the de facto standard for orchestrating containerized microservices. This adoption of container orchestration platforms has transformed deployment workflows in both industry and academia (Kumar et al., 2024; Spišáková et al., 2022). In particular, with the rise of smart things (e.g., factories and IoT-enabled manufacturing), these platforms now host applications ranging from large-scale monitoring systems to digital twin-based process optimization (Wermann & Wickboldt, 2025).

Numerous studies have sought to optimize performance and resource efficiency in containerized systems (Alamouh & Eichelberger, 2024; Centofanti et al., 2023; Correia et al., 2023). Mondal et al. (2024) investigate Kubernetes autoscaling and scheduling strategies to minimize user involvement. KOptim (Menouer et al., 2024) streamlines SLA-based resource configuration for non-expert users, while Kakati and Brorsson (2025) explore usability implications of container platforms in performance-critical contexts. Even a simulation framework like EdgeSimPy (Souza et al., 2023) lacks an accessible front-end for experimentation by diverse user groups. The majority of these efforts overlooked human-centered design aspects, including user onboarding, interface transparency, and trust in container orchestration platforms. Overall, this gap highlights a lack of research from an HCI perspective on container orchestration platforms.

2.2. Limitations and challenges of existing user interfaces

Although Kubernetes is gaining widespread adoption, it introduces several challenges that directly impact usability and accessibility in complex, distributed computing environments, particularly Industry 5.0 infrastructures. In such environments, users across operational, supervisory, and engineering roles interact with containerized applications at different levels of abstraction and responsibility. However, existing orchestration interfaces largely assume advanced engineering-level technical knowledge, which creates significant barriers for non-expert industrial users. The following challenges highlight these limitations:

1. *Effort-intensive onboarding and steep learning curve:* For users in operational or supervisory roles in manufacturing environments, onboarding into Kubernetes-based orchestration remains a major obstacle. The abstract syntax of configuration languages and cognitively demanding interfaces requires either extensive training or reliance on detailed documentation (Anumandla, 2024). This slows adoption in industrial domains where rapid experimentation and iterative deployment are often required, but where users may not possess deep cloud-native or orchestration expertise.
2. *Default dashboard complexity:* The default Kubernetes dashboard provides cluster-level visibility, but it is not tailored to users with diverse roles or responsibilities. While it may be suitable for engineering users who require infrastructure-level visibility, it overwhelms operational or supervisory users who primarily need simple workload-centric interaction. Third-party solutions like Rancher (2026) and Openshift (2026) also offer modified versions of Kubernetes together with operator-focused dashboards. However, these platforms assume a high degree of technical familiarity, which exposes users to low-level Kubernetes concepts that are often unnecessary for operational or supervisory roles.
3. *Lack of usability focus:* User-interfacing tools such as Kubectl (2026) and Helm (2026) are quite powerful but primarily designed for expert users. These tools rely heavily on verbose, low-level YAML manifests, which means even a minor syntax error can cause a deployment to fail. In

Industry 5.0 settings, such as smart factories, these errors may compromise system reliability, disrupt production processes, and reduce trust in container orchestration platforms.

4. *Usability-focused empirical studies:* Research on container orchestration mainly focused on performance optimization, scheduling, and scalability, while systematic usability evaluations remain limited (Stewart, 2024). Some studies have explored visual deployment techniques (Kurrewar et al., 2025; Piedade et al., 2022; Savusalo, 2023), but comprehensive empirical evaluations across heterogeneous user groups are still lacking. An early effort in Usman (2025) explored Kubernetes-based experimentation in Industrial IoT settings, but did not include participant-based usability evaluations. To the best of our knowledge, no prior work has conducted a holistic, human-centered usability evaluation of Kubernetes interfaces that explicitly considers users with heterogeneous role categories.

2.3. User-friendly interface requirements

Addressing these identified challenges requires applying well-established HCI principles, such as progressive disclosure, error prevention, and learnability, to the user interfaces of container orchestration platforms. From the perspective of Industry 5.0, usability is essential to ensure transparency, trust, and effective collaboration between humans and increasingly autonomous digital infrastructures.

1. *Progressive, role-aware interfaces:* Interfaces must adapt to the expertise of users, exposing advanced configuration options only when required. For instance, operational users may interact with high-level workload abstractions, supervisory users get controlled visibility and parameterization capabilities, while engineering users require fine-grained configuration and infrastructure-level control. By leveraging Role-Based Access Control (RBAC) (Gu et al., 2025), orchestration systems can enforce the principle of least privilege and reduce the risk of unintentional misconfiguration that may impact both cyber and physical systems.
2. *Integrated monitoring for operational and supervisory decision-making:* Fine-tuning resource usage is crucial in container-based infrastructures, particularly for real-time, performance-sensitive Industry 5.0 applications. Interfaces should provide integrated, instant access to both workload- and infrastructure-level monitoring data to support proactive optimization, bridging the gap between workload behavior and operational decision-making without requiring deep technical expertise or manual correlation of disparate data sources.
3. *Human-centered error handling and feedback:* Beyond functional controls, orchestration interfaces should prioritize deployed workload error prevention and transparent feedback to users. Clear validation, contextual warnings, and understandable system responses are especially important for operational and supervisory roles, while still supporting detailed diagnostics required by engineering users. Such design choices directly support Industry 5.0 goals of safe and effective human interaction with complex, autonomous systems.

In summary, although existing container orchestration platforms like Kubernetes offer advanced technical capabilities, they fall short of meeting the human-centered requirements crucial to Industry 5.0 environments. Overcoming these challenges requires container orchestration platforms that explicitly account for heterogeneous industrial user roles and prioritize usability, transparency, and adaptive access as core design principles.

3. Prototyped system: Design and implementation

We developed a lightweight, user-friendly container orchestration interface that abstracts core Kubernetes operations to support users across operational, supervisory, and engineering roles when managing microservices-based applications. Rather than extending Kubernetes through custom control-plane components, the proposed system is implemented as an external wrapper that interacts with existing Kubernetes clusters via standard APIs. This design choice prioritizes usability, portability, and ease of deployment while avoiding intrusive modifications to cluster internals. An abstracted overview of the prototyped system architecture, including end-to-end user workflow, is shown in Figure 1.

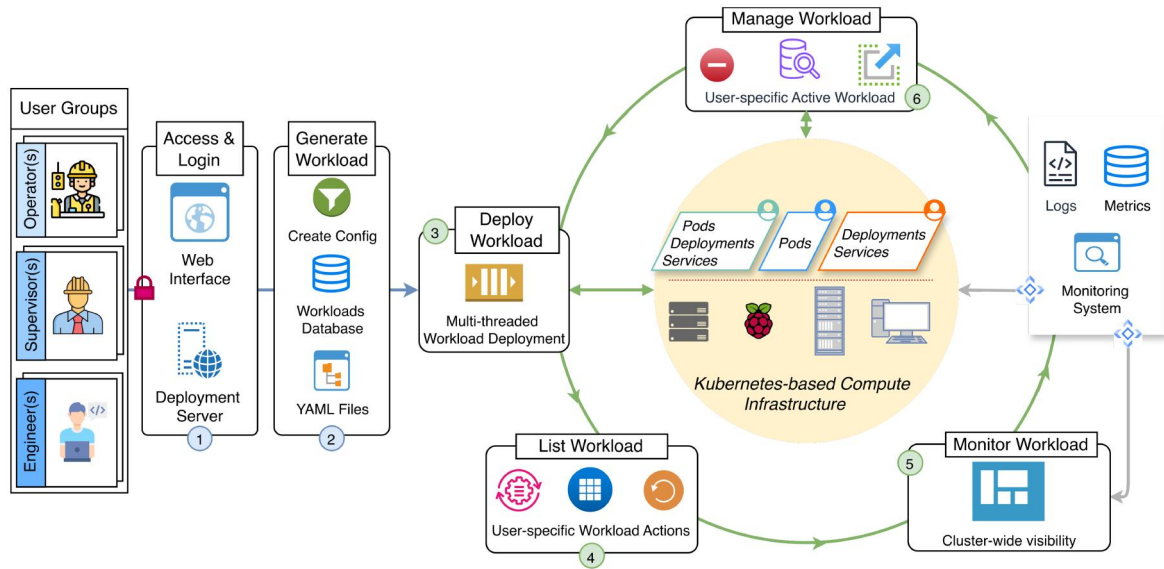


Figure 1. System-wide design of the prototyped orchestration system illustrating an end-to-end workload management workflow.

3.1. User expertise-based UI design

The proposed system adopts a role-aware interaction model that aligns the visibility and complexity of orchestration features with users' responsibilities and expertise. This design follows the HCI principle of progressive disclosure, which usability studies have shown reduces cognitive load, prevents configuration errors, and improves user confidence when interacting with complex systems. For generalizability, the proposed interface design decomposes into abstractions across three levels:

- **Beginner** – Users with minimal container or orchestration experience (e.g., plant operators or domain engineers). These users typically understand which data should be processed and which actions should be executed, but they are abstracted from deployment internals. The interface exposes only core, low-risk operations (e.g., deploying available applications, launching predefined workload templates, and configuring a limited set of parameters).
- **Intermediate** – Users overseeing performance, experimentation, or service-level adjustments and possessing prior exposure to containerized application development (e.g., production managers or industrial engineers). They can reason about application structure but lack operational expertise in clusters. They are granted access to moderate-level operations such as modifying complex workload parameters, scaling replicas, and redeploying applications.
- **Advanced** – Users with considerable container orchestration experience (e.g., automation engineers or DevOps engineers) who are responsible for infrastructure-level concerns. They are provided with near-complete control, including deployment via custom YAML or Helm charts, workload editing, and advanced workload configuration options.

This adaptive design follows the HCI principle of progressive disclosure and supports human-centric system interaction by allowing users to operate at an abstraction level aligned with their role and responsibility. As a result, the system supports trust, safety, and efficiency across diverse application environments, including Industry 5.0, where human oversight remains crucial.

3.2. Key features

The system design aims to minimize configuration overhead and reduce cognitive load for users with different levels of expertise. Instead of exposing low-level Kubernetes operations, the interface presents task-oriented abstractions that correspond well to how industrial users view applications and processes. It also

provides continuous feedback and visibility throughout the major steps during the microservices-based workload management lifecycle. Furthermore, it supports reproducibility by allowing users to create workload templates for repeatable deployments. The key features of the proposed system are summarized as follows:

- **Workload lifecycle management:** Allows users to create workload templates, configure runtime parameters, deploy workloads over specific cluster nodes, scale any existing workloads, or delete any dangling workloads with minimal clicks. This abstraction enables rapid iteration in industrial testbeds and aligns with production-oriented workflows where users reason in terms of services rather than infrastructure primitives. The system prioritizes deployment reliability by avoiding direct manipulation especially in case of Helm values.yaml complex files, which often leads to syntax errors. Instead, it implements a dynamic key-value override mechanism, allowing advanced users to safely update chart default parameters during deployment without the risk of incorrectly restructuring the underlying configuration.
- **Monitoring and observability:** Provides real-time monitoring of various metrics and application logs (e.g., Pod CPU and memory usage) through an integrated, user-friendly interface without requiring deep technical monitoring expertise or navigation to fragmented tools.
- **History and accountability:** The system maintains a record of user actions, workloads deployed, and corresponding timestamps. This feature aims to support traceability and enhance transparency and accountability, which are important aspects in collaborative Industry 5.0 scenarios involving multiple users interacting with shared computational resources.
- **Resource isolation and usage control:** While the proposed system simplifies access through user-friendly interfaces, it also ensures fine-grained control and reduces the risk of errors in deployed workloads. This is realized by automatically creating isolated segments for each user, thereby preventing interference with other critical workloads running in the same compute environment. Furthermore, each user is assigned an upper limit on available resources to prevent outages that could affect mission-critical applications, including those controlling key industrial processes.

3.3. Software design and implementation details

The proposed system is implemented as a containerized, web-based orchestration interface that integrates directly with Kubernetes through its native APIs. The system follows a client-server architecture, in which a web-based front-end communicates with a backend orchestration layer that interacts with the Kubernetes control plane. The entire system is packaged and deployed as a single container, enabling straightforward installation in any Kubernetes cluster.

Integration with Kubernetes is performed using the standard `kubeconfig` approach. This standard approach allows the system to authenticate and communicate securely with the Kubernetes API server. All cluster operations, such as workload creation, deployment, monitoring, and deletion, are performed using the Kubernetes Python client library (Python-client, 2026). Using this Python-based library eliminates the need for direct `kubectl` usage or command-line access. Overall, this interaction design ensures portability across clusters and compatibility with existing Kubernetes security and access-control mechanisms. A deployment YAML file (see Listing 1) is employed to configure and deploy the system container within a target Kubernetes cluster, including resource limits and the target node.

The proposed system does not introduce any custom resource definitions (CRDs) or controller extensions. Instead, it relies exclusively on native Kubernetes abstractions and resource definitions such as Pods, Deployments, and Namespaces. This design choice is made to reduce operational complexity and lower the barrier to adoption, particularly for users with limited Kubernetes expertise. While the current implementation is not operator-based, the architecture is compatible with such an approach and could be extended in future work to support controller-driven lifecycle management. In order to ensure secure multi-tenancy and resource isolation, the proposed system implements an automated lifecycle manager that leverages `Kubernetes Python client`. When a request to create a new user is received, the system programmatically enforces an isolated execution environment through four distinct steps:

- **Namespace provisioning:** First, a dedicated Kubernetes Namespace is created to logically isolate the workloads of the new user from those of existing ones.

- Resource governance: Next, a Kubernetes `ResourceQuota` object is applied to the newly created namespace, imposing hard limits on CPU, memory, and pod counts to prevent resource overuse and ensure system stability.
- RBAC enforcement: After that, the system defines a specialized `Role` with granular permissions that permit only necessary actions on standard resources like deployments, services, and secrets.
- Identity binding: Finally, a `RoleBinding` step links this role to the default `ServiceAccount` of the new namespace. This approach restricts the operational scope of the user to their own namespace, which effectively enforces the principle of least privilege and minimizes security risks associated with shared cluster access.

The UI is developed using HTML and CSS to achieve a responsive layout and interactive views. Several screenshots of the developed UI are shown in [Figure 2a,b](#), illustrating simple and easily navigable options compared to the default Kubernetes dashboard (see [Figure 2c](#)) for managing containerized microservices-based workloads. Client-side logic is implemented using JavaScript, including dynamic form inputs and real-time data fetching. The template rendering engine bundled with Flask (2026) serves HTML views to users based on their expertise levels. To meet the monitoring and troubleshooting requirements for individual workloads, a metadata exporter has been developed that automatically retrieves key identifiers for each workload.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: workload-manager
  namespace: simulation
spec:
  replicas: 1
  selector:
    matchLabels:
      app: workload-manager
  template:
    metadata:
      labels:
        app: workload-manager
    spec:
      containers:
      - name: workload-manager
        image: usman476/workload-manager:0.5.0
        imagePullPolicy: "Always"
        ports:
        - containerPort: 9000
        volumeMounts:
        - name: kubeconfig
          mountPath: /home/.kube
          readOnly: true
      resources:
        requests:
          cpu: 250m
          memory: 256Mi
        limits:
          cpu: 500m
          memory: 512Mi
      volumes:
      - name: kubeconfig
        hostPath:
          path: <<PATH_TO_FILE>>
      tolerations:
      - key: "node-role.kubernetes.io/control-plane"
        operator: "Exists"
        effect: "NoSchedule"
      nodeSelector:
        kubernetes.io/hostname: master
```

Listing 1. Workload deployment YAML configuration file (simplified).

Workload Orchestration

Logged in user: *beginner*

- Create Workload Type
- Deploy Workload**
- List Deployed Workloads
- Delete Running Workloads
- Monitor Workloads
- Logout

Deploy Workload(s) in the Cluster

Run Duration (seconds):

Workload Type:

No. of Replicas (Deployment only):

Cluster Node for Deployment:

Remove

Add More Workload

Submit

(a) Prototyped interface for deploying workload in the distributed compute cluster (beginner-level user).

Workload Orchestration

Logged in user: *beginner*

- Create Workload Type
- Deploy Workload
- List Deployed Workloads**
- Delete Running Workloads
- Monitor Workloads
- Logout

List Deployed Workload(s) in the Cluster

Workload	Owner	Created At	Duration	Replicas	Node
CPU-Burner	beginner	2025-08-29T15:08:48.8	120	1	worker1
CoAP-Server	beginner	2025-08-29T15:09:00.2	60	1	any
Data-Generator	beginner	2025-08-29T15:09:27.8	60	1	worker2

(b) Prototyped interface for listing deployed workloads (beginner-level user).

kubernetes

advanced

Search

Workloads > Pods

- Workloads
- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods**
- Replica Sets
- Replication Controllers
- Stateful Sets
- Service
- Ingresses
- Ingress Classes
- Services
- Config and Storage
- Config Maps
- Persistent Volume Claims
- Secrets
- Storage Classes
- Cluster
- Cluster Role Bindings

Pods

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
web-ui-app-0	usman476/web-ui:latest	app: web-ui-app app: kubernetes.io/pod-index: 0 controller-revision-hash: web-ui-app-66c9bdf7	worker1	Terminating	0	-	-	9 minutes ago
temp-actuator-deployment-0	usman476/mqmt-kafka-faust:latest	app: temp-actuator-deployment app: kubernetes.io/pod-index: 0 controller-revision-hash: temp-actuator-deployment-6c74f48d84	worker1	Terminating	0	-	-	9 minutes ago
temp-analyzer-deployment-0	usman476/mqmt-kafka-faust:latest	app: temp-analyzer-deployment app: kubernetes.io/pod-index: 0 controller-revision-hash: temp-analyzer-deployment-79b7c5d4c	worker1	Terminating	2	-	-	9 minutes ago

(c) Kubernetes default dashboard with a partial view showing overly exposed options to the user.

Figure 2. Kubernetes default dashboard compared with the prototyped UIs. (a) Prototyped interface for deploying workload in the distributed compute cluster (beginner-level user). (b) Prototyped interface for listing deployed workloads (beginner-level user). (c) Kubernetes default dashboard with a partial view showing overly exposed options to the user.

A few other well-known tools are also integrated into the prototype to support key functionalities. Grafana (Elradi, 2025) serves as the primary monitoring system, providing access to time-series metrics from the underlying infrastructure and containerized workloads. To manage structured data (e.g., user credentials, permissions, and workload deployment logs), SQLite (2026) is used to provide a lightweight relational database. For time-series data management, Prometheus (Elradi, 2025) serves as the core monitoring backend, supporting scalable metric storage in containerized environments. Loki (2026) is used as the logs backend to support the collection and storage of application logs in an organized manner.

4. Usability study methodology

The usability evaluation was designed to assess how effectively the prototype system supports users with varying levels of expertise in containerized microservices orchestration over distributed compute infrastructure. The study adhered to standard HCI evaluation practices, incorporating task-based performance analysis, standardized usability scales, and qualitative feedback to capture both behavioral and experiential aspects of use.

4.1. Participant background, consent procedure, and ethical considerations

Prior to participation, all individuals were briefed about the purpose, scope, and voluntary nature of the study, and were asked to sign an informed consent form in line with ethical research protocols. Consent included agreement to audio recording, which supported accurate transcription and thematic coding of qualitative data. Participants were assured of anonymity, confidentiality, and their right to withdraw at any time without consequence. Participants completed a brief demographic and background questionnaire that covered age, education, professional experience, and familiarity with container and orchestration technologies. This served two purposes: (1) to classify participants into beginner, intermediate, or advanced expertise groups (as described in Section 3), and (2) to contextualize both quantitative and qualitative findings.

4.2. Task design and execution

The study used role-based task scenarios, which are aligned with the system's expertise-based interface design. Each participant completed a microservices management workflow, including workload creation, deployment, monitoring, and deletion, adapted in complexity according to their assigned expertise group.

Participants in the *beginner group* were exposed to a simplified interface with default settings. Participants created a workload of type Pod using the dynamic configuration method, which auto-generated the underlying YAML manifests. After specifying the workload, they deployed it to a selected cluster node, monitored runtime metrics (CPU, memory, network), and deleted it via single-click controls. This task assessed whether non-experts could successfully complete the orchestration lifecycle without prior knowledge of YAML or CLI.

Participants in the *intermediate group* were presented with additional dynamic YAML configuration options, such as environment variables, resource limits, and image pull policies. Tasks required creating a workload that included at least one of these advanced parameters, followed by the same deployment, monitoring, and deletion workflow as the beginner group.

Participants in the *advanced group* were granted full access to configuration options, including scheduling rules, affinity constraints, and security settings. Participants created workloads reflecting production-oriented deployment strategies, combining multiple parameters. They then executed the same lifecycle tasks, enabling evaluation of whether the system supported complex orchestration scenarios without undue burden.

4.3. Post-task SUS survey

After completing assigned tasks, participants completed the System Usability Scale (SUS), a ten-item Likert-scale questionnaire widely used to assess subjective usability Brooke (1996). The SUS helped capture perceptions of ease of use, complexity, confidence, and learnability, enabling a standardized comparison of system usability across different expertise levels.

4.4. Post-task semi-structured interview

Following the SUS survey, all participants took part in a semi-structured interview. It was designed to obtain qualitative insights into their experiences with the container orchestration prototype. The purpose of this interview was to complement the quantitative SUS results by exploring user perceptions in depth, including task-specific reflections, usability and accessibility concerns, cognitive workload, and suggestions for improvement. Each interview lasted approximately 15–20 min and was conducted individually, allowing participants to share their personal experiences without external influence. The interview was organized into six thematic sections, each targeting a specific aspect of user interaction with the system.

1. *Task-specific reflections* invited participants to discuss the clarity of instructions and their overall experience with creating workloads. Participants were encouraged to identify parts of the process they found intuitive, as well as those they found confusing or difficult. This section aimed to uncover usability friction points and user expectations during key system interactions.
2. *Usability and accessibility* focused on identifying barriers related to interface design, navigation, or system responsiveness. This was particularly relevant for ensuring the system could accommodate users with diverse abilities or needs, and to assess whether the interaction model aligned with usability and accessibility principles.
3. *Cognitive load and complexity* asked whether the number and nature of the available configuration options matched their level of expertise. This question aimed to assess how well the system accommodated users' cognitive capacities and whether it provided sufficient support or overburdened users across different skill levels.
4. *Suggestions and future features* invited participants to propose features or improvements they would like to see in subsequent iterations of the system. This input was critical for guiding the system's iterative design and ensuring alignment with real-world user needs and workflows.
5. *Reflection on learning and usage* asked whether participants could envision using the system in a professional or experimental context. This question helped assess the perceived practical value and learning curve of the tool, providing insight into its potential for adoption and long-term engagement.
6. *Additional concerns or issues* allowed participants to ask questions or offer comments that were not previously covered. This open-ended conclusion allowed participants to share personal opinions that fell outside the predefined structure of the interview.

5. Results and analysis

We employed physical machines to create a Kubernetes cluster to verify the operation of the proposed prototype system and conduct the usability study. The testbed consisted of one master node and three worker nodes. All nodes are equipped with an Intel(R) Core(TM) i7-6700 CPU operating at 3.40 GHz, 16 GB of memory, 256 GB of disk space, Ubuntu 24.04, and Kubernetes 1.30.0.

We analyzed ten post-task, semi-structured interviews conducted after participants used the prototype system to create, deploy, monitor, and delete workloads. Data analysis followed a reflexive thematic analysis with inductive coding Braun and Clarke (2012), where codes were iteratively grouped into themes and refined for coherence and distinctiveness. Where relevant, we report prevalence (i.e., number of participants who expressed an idea) and illustrate findings with anonymized participant quotations.

5.1. Demographics of study participants

The study included ten participants (P1–P10; see Table 1), as usability research shows diminishing returns in problem discovery as more users are added. Virzi reported that roughly 80% of usability problems can be detected with 4–5 participants in typical formative tests (Virzi (1992)). The depth and triangulation of our protocol (observation, interviews, user trials, questionnaires) captured reasoning, confusion points, and coping strategies, which is precisely the kind of evidence formative usability studies aim to produce. Besides, although our participants primarily come from academia, they possess the requisite domain expertise to evaluate the interaction design for efficacy, enabling us to validate the core usability of the prototyped system before conducting any large-scale industrial field trials.

Most participants were mid-career professionals: five were aged 35–44, three were 25–34, one was 18–24, and one was 45–54. Educational level was high, with six holding Master's degrees, three holding PhDs, and one holding a Bachelor's degree. Eight participants worked in academia or education, while two worked in industry roles (telecommunications, cloud infrastructure/SRE). Self-reported orchestration expertise was distributed across levels: four beginners, three intermediates, and three advanced users. The frequency of containerized workload use ranged from rare (less than monthly, reported by five participants) to daily (one participant). Learning paths were diverse, comprising self-taught (4), on-the-job (4), and formal training (2), providing varied perspectives. Overall, the sample represented a technically literate but heterogeneous group, mirroring the mix of users that orchestration tools are increasingly expected to support in both experimental and industrial environments.

5.2. Self-reported expertise in core tools and concepts

Table 2 details self-rated experience with core tools and concepts. Ratings show a cohort that is comfortable with foundational tooling but less confident with packaging and cluster-level operations. Most participants report high proficiency with the Linux/Unix command line. They are capable Docker users, yet Kubernetes skills are grouped around basic to intermediate levels. Helm stands out as the weakest area, with many novices and only one participant reporting proficiency. YAML editing sits in the middle; many can read and modify files, but few have advanced mastery. Experience with monitoring tools

Table 1. Overview of participants' background and task completion times.

ID	Skill level	Age group	Degree	Work sector	Usage frequency	How skills were acquired	Duration (min.sec)
P1	Advanced	25–34	PhD	Academia	Weekly	Workplace practice	9.55
P2	Beginner	25–34	MSc	Academia	Very infrequent	Self-directed study	5.06
P3	Interm.	35–44	MSc	Academia	Very infrequent	Formal instruction	14.44
P4	Beginner	25–34	PhD	Telecom industry	Very infrequent	Workplace practice	9.46
P5	Beginner	35–44	MSc	Academia	Very infrequent	Self-directed study	7.25
P6	Advanced	35–44	BSc	Cloud/SRE	Daily	Workplace practice	10.58
P7	Advanced	45–54	PhD	Academia	Weekly	Workplace practice	7.00
P8	Interm.	35–44	MSc	Academia	Very infrequent	Formal instruction	8.30
P9	Interm.	35–44	MSc	Academia	Occasional (few/mo)	Self-directed study	7.30
P10	Beginner	18–24	MSc	Academia	Occasional (few/mo)	Self-directed study	6.30

Table 2. Participant self-assessed familiarity with selected tools and technological concepts (scale 1–5).

Technology/concept	1	2	3	4	5
Linux/Unix shell commands		P2		P1, P3, P5, P9, P10	P4, P6, P7, P8
Docker (image build & container run)		P5	P2, P3, P8, P10	P1, P4, P6, P9	P7
Kubernetes (kubectl, deployments, etc.)	P5	P6, P7, P8	P2, P3, P10	P1, P4, P9	
Helm (package charts, values, etc.)	P1, P5, P6, P8	P3, P4, P7	P2, P10	P9	
YAML configuration (syntax and use)	P5	P3, P4	P1, P2, P6, P8	P9, P10	P7
Resource allocation (CPU, memory limits)		P1, P5	P2, P3, P4, P6, P8, P9	P7, P10	
Resource separation & control (Namespaces, cGroups)	P5	P2, P3	P1, P4, P6, P8, P9, P10	P7	
System monitoring (Prometheus, Grafana, etc.)		P2, P3, P4, P8	P1, P6, P7, P9	P5	P10
Cloud infrastructure (AWS, GCP, Azure)	P1	P3, P4, P6, P7, P8	P2, P5, P9, P10		
Networking basics (service, ingress, etc.)	P5	P2	P1, P3, P6, P8, P9	P4, P7	P10
Text-based UIs or terminal dashboards		P2, P5	P1, P3	P4, P9	P6, P7, P8, P10

Scale meaning: 1 = No prior exposure; 2 = Minimal knowledge; 3 = General awareness; 4 = Comfortable user; 5 = Highly experienced. Each participant (P1–P10) appears exactly once in every row, distributed across the five categories.

and cloud platforms is similarly mid-tier: exposure is common, but advanced use is rare. Networking concepts are generally understood at a basic level.

Overall, this skill distribution mirrors roles that are commonly found in industrial environments. Participants are well equipped to reason about what an application should do, similar to operators who understand data sources, processing needs, and expected actions, but are less experienced with how such applications are deployed and operated at scale. For the prototype evaluation, this means success hinged less on command syntax recall and more on whether the interface could effectively expose defaults, guide decision-making, and reveal complexity in a progressive manner. This aligns well with the design goal of the system in supporting human-centric interaction in Industry 5.0 contexts by decoupling operational expertise from domain knowledge.

5.3. Time to complete assigned tasks

Task performance was measured as the time required to complete each assigned workflow. Completion times ranged from 5:06 to 14:44 min, with a mean of 8:36 and a median of 7:44.

- Beginners completed tasks in a mean time of 7:05 (median 6:44).
- Intermediates' completion time averaged 9:56 (median 8:03).
- Advanced users' completion time averaged 9:17 (median 9:55).

These differences in task completion times primarily reflect the way tasks were allocated based on participants' expertise levels, rather than direct differences in efficiency. Beginners were assigned simpler scenarios, while intermediates and advanced users worked with increasingly complex tasks. Importantly, all measurements represent participants' first encounter with the prototyped system. The relatively short times, even for complex tasks, indicate that the interface was easily accessible and learnable from the outset, reducing entry barriers without constraining expert workflows.

5.4. Quick onboarding and task clarity

Participants consistently described the interface as easier to use than CLI-based Kubernetes workflows for creating, deploying, monitoring, and deleting workloads. Nine participants explicitly characterized the process as straightforward or simple to follow, frequently citing the stepwise structure. As P2 noted, it was "very simple and clear to follow step by step," while P8 contrasted it with raw YAML: "probably easier than writing the YAML file. My only prior Kubernetes experience was on the command line, and it was a bit complex to work with." These statements indicate that the design successfully reduced configuration overhead and cognitive load, particularly during onboarding. By abstracting orchestration details into guided forms and constrained choice sets, the system enabled smooth progression from intent to runnable workloads. This result is particularly relevant for Industry 5.0 scenarios, where operators and supervisors are expected to supervise and adapt digital systems without being burdened by underlying technical complexity.

5.5. Noted differences across expertise-levels

The prototyped system supports three user groups and allows users to select from "beginner," "intermediate," or "advanced" expertise modes based on their own experience. Due to this, participants in this study experienced the system differently and expected diverse support from it.

- Beginners valued the guided flow as a safe environment, which reduced their fear of mistakes and helped build confidence. P10 noted, "It was easier for me at the first step as a new user." This mirrors operators who require operational safety and clarity without exposure to low-level system details.
- Intermediates appreciated the structure but wanted more control and explanations, especially regarding resource units and the generated YAML. For instance, P9 requested "more description about the resources," while P1 asked for "a descriptive YAML for what you're deploying." These users resemble supervisors who want to understand system behavior without assuming full operational responsibility.

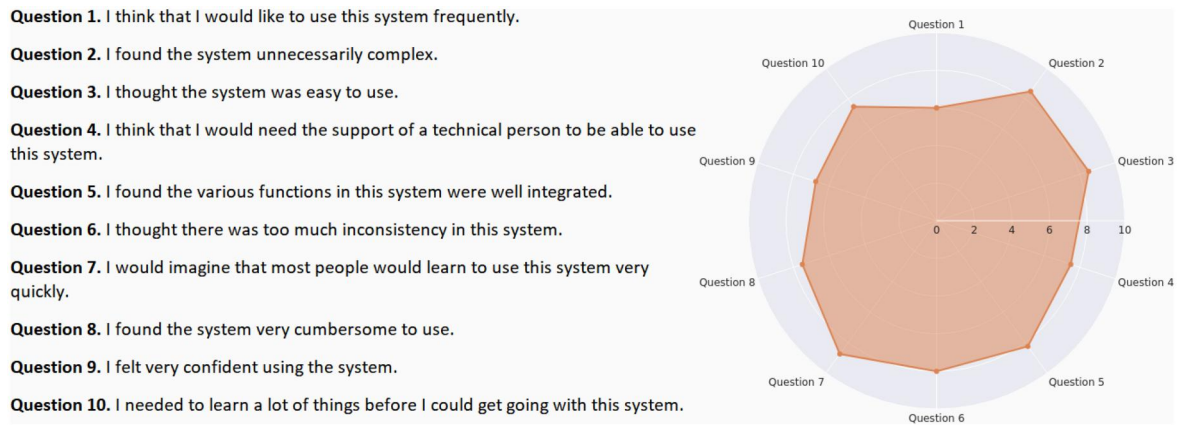


Figure 3. SUS survey score.

- Advanced users emphasized speed, diagnostics, and direct access. P7 highlighted the need for “log output ... to know if it boots,” and P8 suggested optional shell access to the running pod. These stated expectations align well with the engineering role that is responsible for managing, troubleshooting, and optimizing the overall IT processes.

Overall, the system’s information architecture and functions worked differently depending on the user’s level of expertise. Beginners benefited most from the simplicity and clear, step-by-step instructions. Intermediate-level expertise users appreciated the depth of control, while seeking more explanations about controls and visibility into deployed workload states. Advanced users perceived the system differently, viewing it as a primary developer assistant that could be further improved by providing speed and diagnostics. This diverse spectrum of requirements also highlights a need for progressively adaptive interfaces in Industry 5.0 environments, where heterogeneous human roles must collaborate around shared digital infrastructures.

5.6. System usability scale (SUS) survey results

To complement the qualitative findings, we also collected quantitative usability ratings using the SUS survey (Figure 3). The overall SUS score was 77.25, with a median of 80 and a standard deviation of 10.57. According to established SUS benchmarks, this score falls within the “Good” usability range, corresponding to a grade of B and classified as acceptable. Consistency across responses (80%) further supports this result.

Examining the per-question normalized scores (scaled 0–10), the tool scored particularly high on ease of use and confidence in using the system. Items related to system consistency (Q2, 8.5; Q3, 8.5; Q7, 8.75) and learnability (Q5, 8.25; Q6, 8.0) were especially strong, reinforcing interview findings that the stepwise workflow reduced barriers for new users. Slightly lower scores were observed for Q1 (6.0) and Q9 (6.75), both of which addressed the need for support materials and contextual help.

Overall, the SUS results quantitatively validate the qualitative observations: the system achieves a high level of usability, particularly in clarity, consistency, and ease of learning. At the same time, the slightly lower scores on certain items highlight areas for refinement, such as terminology support, contextual feedback, and transparency. Addressing these aspects could elevate usability from “Good” to “Excellent” in future iterations.

6. Discussion around identified gaps

6.1. Feedback gaps

Participants consistently emphasized the need for clearer feedback about system state and progress. For example, P1 suggested a pop-up notification “to confirm success or failure.” P2 was unsure whether a delete action was completed, and therefore thought that “it could be done if the delete button is pressed

twice,” implying that unclear feedback nudged toward duplicate actions. Similarly, participants sought immediate confirmation that pods had started, and several (P6–P8) requested access to logs for diagnostic purposes.

In the context of Industry 5.0, this reflects the principle of human-centric transparency, which is the ability of orchestration systems to provide situational awareness without requiring deep technical probing. Enhancing real-time feedback through confirmations, visual status indicators, and optional expert-level logs to reduce unnecessary repetition, build user trust, and foster confidence across user roles.

6.2. Lack of input clarity

Participants without deep Kubernetes experience highlighted field semantics and domain-specific terminology as sources of avoidable uncertainty. Importantly, they did not request additional configuration options. Rather, they requested better descriptions of the existing options. For example, P4 suggested that “information around the fields could be more elaborate” while P9 requested “a little more description about the resources, like CPU/memory.”

In general, the tool is considered helpful in reducing configuration complexity, but occasional jargon and a lack of parameter descriptions can reintroduce cognitive overhead. In practice, simple additions such as tooltips, sample values, and explicit unit labels would reduce cognitive load for beginners while remaining unobtrusive for experts. This aligns with efforts in industrial applications to adapt to diverse user knowledge levels, where technology accommodates varying levels without compromising efficiency.

6.3. Information-architecture cohesion

Participants raised concerns about fragmentation across core actions, including listing, monitoring, and deletion. For instance, P6 argued that “many separated options can be combined” to create “fewer navigation flows,” suggesting “some sort of combined view” in which actions are co-located with state and evidence.

This recommendation illustrates the HCI principle of task flow cohesion, where interfaces should be organized around user goals (in this case, workload management) rather than technical system abstractions. A workload-centric architecture reflects the industrial applications’ vision of intuitive interaction, reduced navigation overhead, and a strengthened bond between human intent and system action.

6.4. An option to export YAML templates

Four participants requested the option to inspect, edit, and export the declarative manifests corresponding to their choices. P1 asked to “show a descriptive YAML for what you’re deploying” to support verification, while P6 preferred to upload YAML and “do some adjustments on the fly instead of cumbersome text boxes.” P7 framed exportability as a prerequisite for production use: after testing, they wanted to “print out YAML to use in production.”

Importantly, these requests do not contradict the simplicity. Rather, they specify how the GUI can align with prevailing engineering practices. This can even enable learning by inspection for beginners, support reproducibility through saved manifests, and make the simplified GUI an entry point into CI/CD pipelines. In Industry 5.0 contexts, this bridges human creativity and machine precision, supporting reproducibility and integration into CI/CD pipelines.

6.5. Validity and limitations

The study’s internal validity is supported by concurrent stance across participants. Interviews were conducted immediately after successful task completion, which may raise positive sentiment through recency and completion biases. To mitigate internal validity, we maintained the same study conditions across participants’ expertise levels (i.e., identical task scripts, the same device/configuration, and a standardized, non-leading interview guide). We triangulated interviews with behavioral traces (task completion time) and a post-task SUS, so interpretations did not rely on sentiment alone.

External validity is naturally limited by the technically skewed sample ($N = 10$). We have used maximum-variation sampling within our target user group, spanning from beginner to advanced proficiency on core tools (e.g., Linux, Docker, Kubernetes, Helm) to capture a broader spectrum of the intended population rather than a single expertise tier. We designed tasks based on realistic workflows and provided detailed descriptions of the context and participant characteristics to support transferability to similar settings.

In addition, while this study focuses on usability and HCI aspects, we acknowledge that system-level performance benchmarking of the proposed prototype and integrated monitoring stack (i.e., Prometheus and Loki) remains a separate concern dependent on individual cluster scaling and deployment requirements.

6.6. Enhancing the prototype through empirical usability findings

Based on the usability study findings, the proposed orchestration system design and implementation are enhanced to address several usability-related issues. Figures 4 and 5 illustrate key modifications and enhancements. These refinements include clearer semantics, adaptive feedback mechanisms, and integrated diagnostics, all of which strengthen the human-centered design aspect of the system.

Containerized Workload(s) Management Dashboard

Workload Orchestration

Logged in User: advanced
Expertise-level: advanced

- Create Workload Template
- Deploy New Workloads
- List Deployed Workloads
- Monitor Cluster-wide Resources
- Manage Running Workloads
- Logout

Deploy Workload(s) in the Cluster

Workload Details

Run Duration (seconds): 600

Workload Template: Kafka-Broker

Chart Name: kafka

Chart Version: 31.1.0

Set Values: key1=value1;key2=value2

Remove Entry

Add More Clear All Deploy

(a) Prototyped system interface with added tooltips for each input.

Containerized Workload(s) Management Dashboard

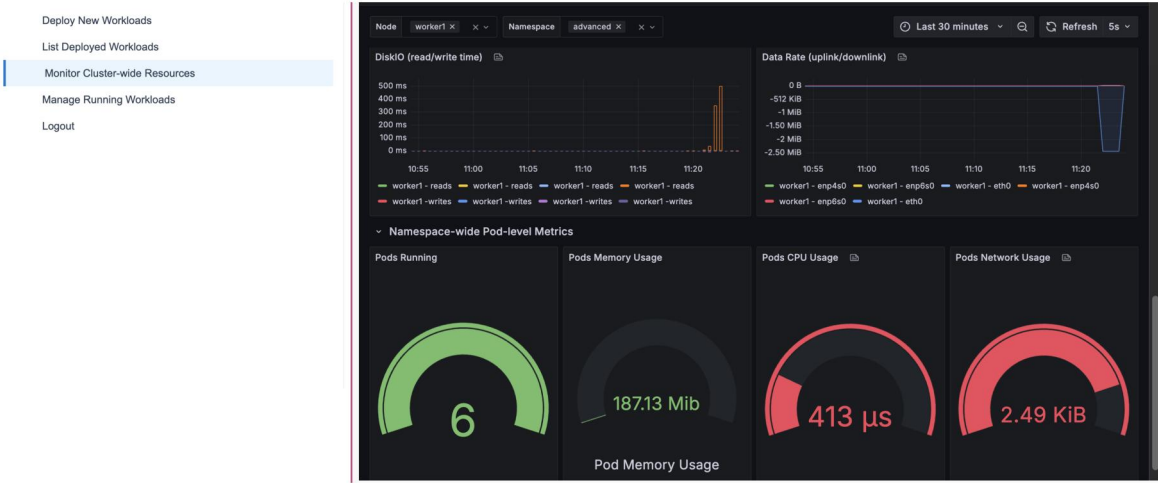
Expertise-level: advanced

- Create Workload Template
- Deploy New Workloads
- List Deployed Workloads
- Monitor Cluster-wide Resources
- Manage Running Workloads
- Logout

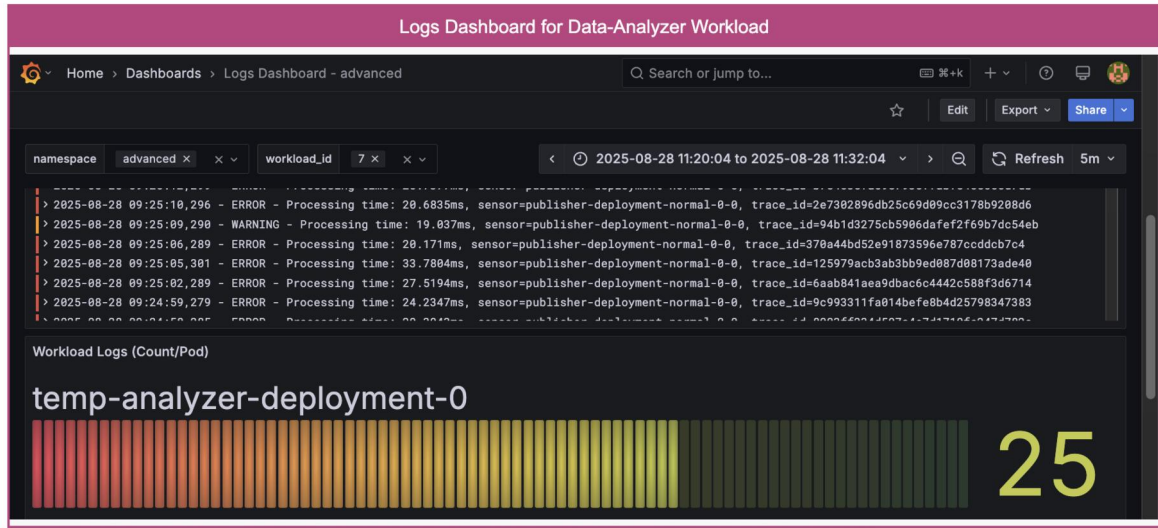
Workload	Method	Created At	Duration	Replicas	Node	Chart	Version	Values	Status	Actions
CoAP-Server	yaml	2025-08-26T15:15:23	120	1	any				Finished	[Icons]
Kafka-Broker	helm	2025-08-28T09:18:07	600			kafka	31.1.0		Running	[Icons]
Mqtt-Broker	helm	2025-08-28T09:18:56	600			mosquitto			Running	[Icons]
MySQL-Server	helm	2025-08-28T09:19:21	600			mysql		primary.persistence.enabled=false	Running	[Icons]
Data-Publisher	yaml	2025-08-28T09:20:41	600	1	worker1				Running	[Icons]
Data-Subscriber	yaml	2025-08-28T09:20:52	600	1	worker1				Running	[Icons]
Data-Analyzer	yaml	2025-08-28T09:21:04	600	1	worker1				Running	[Icons]
Data-Actuator	yaml	2025-08-28T09:21:12	600	1	worker1				Running	[Icons]
Data-UI	yaml	2025-08-28T09:21:20	600	1	worker1				Running	[Icons]

(b) Prototyped system interface with enhanced control and improved feedback.

Figure 4. Updated orchestration system interfaces based on collected user feedback. (a) Prototyped system interface with added tooltips for each input. (b) Prototyped system interface with enhanced control and improved feedback.



(a) Interface to access metrics data for observing cluster-wide physical and user-specific namespace-wide container resources performance.



(b) Interface to access log data for observing a particular workload's internal states.

Figure 5. Updated orchestration system interfaces based on collected user feedback. (a) Interface to access metrics data for observing cluster-wide physical and user-specific namespace-wide container resources performance. (b) Interface to access log data for observing a particular workload's internal states.

The updated interface now provides clear descriptions of input fields through contextual tooltips (see Figure 4a), which primarily support beginner users (i.e., operators or supervisors) who are unfamiliar with container resource semantics. In addition, certain input fields, such as those for CPU resource allocation, provide explicit unit descriptions and validate them before generating a workload template. These updates reduce the likelihood of configuration errors and aim to support intermediate users (i.e., supervisors) who understand application requirements but lack Kubernetes-specific expertise. The interface layout is further restructured (see Figure 4b) to organize interactions around workloads rather than low-level infrastructure objects such as Pods, Namespaces, or raw YAML syntax. This workload-centric organization enables users to manage applications as holistic entities, thereby simplifying interactions with distributed infrastructures.

Instead of exposing raw metrics in isolation, the system now presents a workload-centric view that integrates both metrics (see Figure 5a) and logs (see Figure 5b) within a shared temporal context. Each workload deployed through the system is assigned a unique identifier tag to enable consistent correlation across diverse monitoring data. Overall, this approach aligns with findings from the usability study, which noted that information organized around user tasks or goals is easier to interpret and

verify than isolated technical details. By reducing cognitive load across both deployment and monitoring processes, the revised prototype helps further strengthen user confidence and trust, which are key to human-centered orchestration in Industry 5.0 environments.

7. Conclusion

This study evaluated the usability of a newly developed orchestration interface designed to reduce configuration overhead, lower cognitive load, and support reproducible deployments across distributed, containerized infrastructures hosting applications, such as those in Industry 5.0. The designed prototype prioritized error prevention through guided, template-based workflows, rather than reactive diagnostics. By integrating qualitative (thematic analysis of interviews) and quantitative (SUS and task analysis) methods, we demonstrated that the system provides both efficiency and transparency. The resulting SUS score of 77.25 indicates that the interface achieves good overall usability, positioning it as a promising step toward human-centered orchestration systems.

A key challenge faced during the system interface design was harmonizing the requirements of diverse users. We addressed this challenge by allowing participants to self-select their expertise level, enabling the interface to scale from beginner-friendly simplicity to advanced transparency and control. This approach aims to align with Industry 5.0 principles, in which technology should augment human capabilities and adapt to individual needs rather than impose uniform complexity. In this sense, the interface not only reduces barriers for beginner users (i.e., operators and supervisors) but also provides expert users (i.e., engineers) with meaningful control, making it suitable as both a learning platform and a productivity tool for reproducible orchestration.

The interviews conducted during this study highlighted certain limitations and opportunities for refining the developed orchestration system interface. In particular, participants noted gaps in system feedback, in the clarity of terminology, and in the absence of certain monitoring information for advanced use. We successfully addressed a number of highlighted issues by applying clearer semantics, adaptive feedback mechanisms, and integrated diagnostics to the interface, which strengthens the system's human-centered design.

As an exploratory study in this domain, these findings necessitate further enhancements and empirical validation. Identified technical issues, such as providing granular access to deployment manifests, enabling deep debugging capabilities, and integrating semantic error mapping for complex orchestration states for engineering roles, need to be pursued in future iterations of the system. Also, the current participant pool represented a specific technical demographic. Future work will focus on a broader evaluation involving heterogeneous participants representing operational, supervisory, and engineering roles from both OT and IT domains. This will be followed by longitudinal studies to evaluate how the system performs in real-world, collaborative Industry 5.0 operational environments.

Authors' contributions

CRedit: **Muhammad Usman**: Conceptualization, Investigation, Methodology, Project administration, Resources, Software, Supervision, Writing – original draft, Writing – review & editing; **Bilal Maqbool**: Data curation, Formal analysis, Methodology, Writing – original draft, Writing – review; **Muhammad Shahid Iqbal**: Data curation, Methodology, Resources, Writing – original draft.

Disclosure statement

The authors report no competing interests to declare.

Funding

This work was supported by the Knowledge Foundation of Sweden (KKS); EU H2020 research and innovation program under grant agreement No. 101139172 (6 G-PATH); and Region Värmland through the DHINO project (ref: RUN/220266 and ref: 2023/828).

ORCID

Muhammad Usman  <http://orcid.org/0000-0002-9598-0704>

Bilal Maqbool  <http://orcid.org/0000-0002-1309-2413>

Muhammad Shahid Iqbal  <http://orcid.org/0009-0000-8335-3941>

Data availability statement

The data that support the findings of this study are available from the corresponding author, Muhammad Usman, upon reasonable request.

References

- Alamouh, A., & Eichelberger, H. (2024). Open source container orchestration for Industry 4.0—Requirements and systematic feature analysis. *International Journal on Software Tools for Technology Transfer*, 26(5), 527–550. <https://doi.org/10.1007/s10009-024-00767-w>
- Anumandla, S. K. R. (2024). Automating container orchestration: Innovations and challenges in Kubernetes implementation. *Robotics Xplore: USA Tech Digest*, 1(1), 29–43.
- Borsatti, D., Cerroni, W., Foschini, L., Ya Grabarnik, G., Manca, L., Poltronieri, F., Scotece, D., Shwartz, L., Stefanelli, C., Tortonesi, M., & Zaccarini, M. (2024). Kubetwin: A digital twin framework for Kubernetes deployments at scale. *IEEE Transactions on Network and Service Management*, 21(4), 3889–3903. <https://doi.org/10.1109/TNSM.2024.3405175>
- Braun, V., & Clarke, V. (2012). Thematic analysis. In Cooper, H., Camic, P. M., Long, D. L., Panter, A. T., Rindskopf, D., and Sher, K. J. (Eds.), *APA handbook of research methods in psychology, Vol. 2. Research designs: Quantitative, qualitative, neuropsychological, and biological* (pp. 57–71). American Psychological Association.
- Brooke, J. (1996). Sus—a quick and dirty usability scale. *Usability Evaluation in Industry*, 189(194), 4–7.
- Centofanti, C., Tiberti, W., Marotta, A., Graziosi, F., & Cassioli, D. (2023). *Latency-aware Kubernetes scheduling for microservices orchestration at the edge* [Paper presentation]. IEEE 9th International Conference on Network Softwarization (NetSoft) (pp. 426–431). IEEE.
- Chahed, H., Usman, M., Chatterjee, A., Bayram, F., Chaudhary, R., Brunstrom, A., Taheri, J., Ahmed, B. S., & Kassler, A. (2023). AIDA—A holistic AI-driven networking and processing framework for industrial IoT applications. *Internet of Things*, 22, 100805. <https://doi.org/10.1016/j.iot.2023.100805>
- Correia, M., Oliveira, W., & Cecílio, J. (2023). Monintainer: An orchestration-independent extensible container-based monitoring solution for large clusters. *Journal of Systems Architecture*, 145, 103035. <https://doi.org/10.1016/j.sysarc.2023.103035>
- Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., Yin, J., Dustdar, S., & Zomaya, A. Y. (2024). Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1), 12–46. <https://doi.org/10.1109/JPROC.2024.3353855>
- Elradi, M. D. (2025). Prometheus & Grafana: A metrics-focused monitoring stack. *Journal of Computer Allied Intelligence*, 3(3), 28–39. <https://doi.org/10.69996/jcai.2025015>
- Eyvazov, F., Ali, T. E., Ali, F. I., & Zoltan, A. D. (2024). *Beyond containers: Orchestrating microservices with Minikube, Kubernetes, Docker, and Compose for seamless deployment and scalability* [Paper presentation]. 2024 11th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) (pp. 1–6). IEEE.
- Flask. (2026). *Flask quickstart*. <https://flask.palletsprojects.com/en/stable/quickstart/>
- Ghobakhloo, M., Mahdiraji, H. A., Iranmanesh, M., & Jafari-Sadeghi, V. (2024). From Industry 4.0 digital manufacturing to Industry 5.0 digital society: A roadmap toward human-centric, sustainable, and resilient production. *Information Systems Frontiers*, 1–33. <https://doi.org/10.1007/s10796-024-10476-z>
- Gu, Y., Tan, X., Zhang, Y., Gao, S., & Yang, M. (2025). *EPScan: Automated detection of excessive RBAC permissions in Kubernetes applications* [Paper presentation]. 2025 IEEE Symposium on Security and Privacy (SP) (pp. 3199–3217). IEEE. <https://doi.org/10.1109/SP61157.2025.00011>
- Helm. (2026). *Helm documentation*. <https://helm.sh/docs/>
- Jamil, M. N., Schelén, O., Monrat, A. A., & Andersson, K. (2024). Enabling industrial internet of things by leveraging distributed edge-to-cloud computing: Challenges and opportunities. *IEEE Access*, 12, 127294–127308. <https://doi.org/10.1109/ACCESS.2024.3454812>
- Kakati, S., & Brorsson, M. H. (2025). *Performance and usability implications of multiplatform and WebAssembly containers* [Paper presentation]. 15th International Conference on Cloud Computing and Services Science. Science and Technology Publications. <https://doi.org/10.5220/0013203200003950>
- Khanh, Q. V., Nguyen, V. H., Minh, Q. N., Van, A. D., Le Anh, N., & Chehri, A. (2023). An efficient edge computing management mechanism for sustainable Smart Cities. *Sustainable Computing: Informatics and Systems*, 38, 100867. <https://doi.org/10.1016/j.suscom.2023.100867>

- Kubectl. (2026). *Kubectl documentation*. <https://kubernetes.io/docs/reference/kubectl/>.
- Kumar, R. J., Singh, J. R. J., Sagayam, K. M., Kumar, R., Bhattacharya, P., & Aggarwal, G. (2024). *Enhancing data center efficiency with Docker & Kubernetes: Industry 4.0 Container-as-a-Service (CAAS) for IoE cloud technology* [Paper presentation]. 2024 4th International Conference on Technological Advancements in Computational Sciences (ICTACS) (pp. 1545–1551). IEEE.
- Kurrewar, S., Dhokane, S., Dahake, A., Yadav, R. K., Wyawahare, N., & Morris, N. C. (2025). *Streamlining Kubernetes deployments through GitOps methodologies* [Paper presentation]. 2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECs) (pp. 1–7). IEEE.
- Loki. (2026). *Loki documentation*. <https://grafana.com/docs/loki/latest/get-started/>.
- Menouer, T., Cérin, C., & Darmon, P. (2024). *KOptim: Kubernetes optimization framework* [Paper presentation]. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (pp. 900–908). IEEE.
- Mondal, S. K., Zheng, Z., & Cheng, Y. (2024). On the optimization of Kubernetes toward the enhancement of cloud computing. *Mathematics*, 12(16), 2476. <https://doi.org/10.3390/math12162476>
- Openshift. (2026). *Openshift documentation*. https://docs.redhat.com/en/documentation/openshift/_container/_platform
- Patil, M. (2025). Cloud computing for Industry 5.0. In *Industry 5.0: Key technologies and drivers* (pp. 119–154). Springer.
- Piedade, B., Dias, J. P., & Correia, F. F. (2022). Visual notations in container orchestrations: An empirical study with Docker compose. *Software and Systems Modeling*, 21(5), 1983–2005. <https://doi.org/10.1007/s10270-022-01027-8>
- Python-client. (2026). *Python client*. <https://kubernetes.readthedocs.io/en/latest/>.
- Queiroz, R., Cruz, T., Mendes, J., Sousa, P., & Simões, P. (2024). Container-based virtualization for real-time industrial systems—A systematic review. *ACM Computing Surveys*, 56(3), 1–38. <https://doi.org/10.1145/3617591>
- Rancher. (2026). *Rancher documentation*. <https://ranchermanager.docs.rancher.com/>.
- Savusalo, T. (2023). *Application for managing container-based software development environments* [Master's thesis]. OuluREPO.
- Sharma, M., Tomar, A., & Hazra, A. (2024). Edge computing for Industry 5.0: Fundamental, applications, and research challenges. *IEEE Internet of Things Journal*, 11(11), 19070–19093. <https://doi.org/10.1109/JIOT.2024.3359297>
- Souza, P. S., Ferreto, T., & Calheiros, R. N. (2023). EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*, 148, 446–459. <https://doi.org/10.1016/j.future.2023.06.013>
- Spišáková, V., Klusáček, D., & Hejtmánek, L. (2022). Using Kubernetes in academic environment: Problems and approaches. In *Workshop on job scheduling strategies for parallel processing* (pp. 235–253). Springer. https://doi.org/10.1007/978-3-031-22698-4_12
- SQLite. (2026). *SQLite documentation*. <https://www.sqlite.org/docs.html>
- Stewart, W. (2024). *Redefining K8s platforms*. <https://thenewstack.io/make-workloads-not-infrastructure-redefining-k8s-platforms/>
- Usman, M. (2025). *Effortless lifecycle management for experimental IIoT workloads in containerized environments* [Paper presentation]. 2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN) (pp. 202–206). IEEE. <https://doi.org/10.1109/ICIN64016.2025.10942663>
- Usman, M., Ferlin, S., Brunstrom, A., & Taheri, J. (2022). A survey on observability of distributed edge & container-based microservices. *IEEE Access*, 10, 86904–86919. <https://doi.org/10.1109/ACCESS.2022.3193102>
- Virzi, R. A. (1992). Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 34(4), 457–468. <https://doi.org/10.1177/001872089203400407>
- Wermann, A. G., & Wickboldt, J. A. (2025). KTWIN: A serverless Kubernetes-based digital twin platform. *Computer Networks*, 259, 111095. <https://doi.org/10.1016/j.comnet.2025.111095>

About the authors

Muhammad Usman is an Associate Senior Lecturer in the Department of Mathematics and Computer Science at Karlstad University, Sweden. He received his MS and PhD in Computer Science from GIST, Republic of Korea. His research focuses on cloud and edge computing, industrial automation, distributed systems, DevOps automation, and data visualization.

Bilal Maqbool is a Postdoc researcher in the Department of Mathematics and Computer Science at Karlstad University, Sweden. He received his PhD in Computer Science from Karlstad University, Sweden. His research interests focus on Human–Computer Interaction (HCI), digital health, and AI-assisted usability evaluation within modern computing environments.

Muhammad Shahid Iqbal is a Research Engineer in the Department of Mathematics and Computer Science at Karlstad University, Sweden. He holds a PhD from National Yang Ming Chiao Tung University, Taiwan. His research interests span computer network design, programmable networks, and artificial intelligence, with a focus on AI-as-a-service applications.