On-Premise Kubernetes in Software Engineering Education: An Experience Report

Matthias Linhuber
matthias.linhuber@tum.de
Technical University of Munich
Munich, Germany

Jan Philip Bernius
bernius@siemens.com
Siemens AG
Munich, Germany

Stephan Krusche
krusche@tum.de
Technical University of Munich
Munich, Germany

Abstract

Container orchestration with Kubernetes has become essential in modern software engineering. Graduates are increasingly expected to demonstrate proficiency in its deployment and operation. Yet educational institutions face barriers when adopting Kubernetes in their curricula. Cloud-hosted solutions raise concerns around data privacy, cost predictability, and vendor independence. Self-hosted on-premise solutions seem complex and require in-depth expertise to operate.

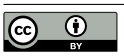
This experience report presents the journey of deploying and evaluating a multi-node on-premise Kubernetes cluster in a project-based software engineering course at the Technical University of Munich over two semesters. First, this paper outlines and compares on-premise Kubernetes components and deployment options in light of the constraints of software engineering education. Based on this analysis, the paper details a deployment employing RKE2, Ansible, and Rancher and provides tangible recommendations.

Second, this experience report evaluates the deployment in three real-world projects developed in an industry-academia collaboration. These projects involved students with minimal prior Kubernetes experience. Key observations reveal that (1) novice students can effectively deploy containerized applications on on-premise Kubernetes with minimal introductory guidance, (2) the operational overhead for administrators proved lower than traditional VM-based deployments, and (3) students successfully manage microservice architectures with moderate ongoing support. The experience demonstrates that carefully designed on-premise Kubernetes deployments can bridge the gap between educational constraints and industry requirements.

CCS Concepts

• **Applied computing** → **Education**; • **Software and its engineering** → **Software system structures**.

ACM Reference Format:
Matthias Linhuber, Jan Philip Bernius, and Stephan Krusche. 2026. On-Premise Kubernetes in Software Engineering Education: An Experience Report. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE-SEET '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3786580.3786953



ICSE-SEET '26, Rio de Janeiro, Brazil

ACM ISBN 979-8-4007-2423-7/2026/04
https://doi.org/10.1145/3786580.3786953

1 Introduction

Modern software engineering education increasingly requires hands-on experience with container orchestration technologies, particularly Kubernetes[1] [38]. These skills have become essential to prepare graduates for industry roles. Educational institutions seek to integrate Kubernetes into their curricula. This integration bridges the gap between theoretical computer science education and practical industry requirements [12, 21, 33]. However, providing students with authentic, industry-relevant infrastructure for learning Kubernetes presents significant challenges.

While cloud-hosted Kubernetes services like Amazon EKS, Google GKE, or Azure AKS simplify deployment and management, they introduce barriers for educational contexts. The Uptime Institute survey shows that 48% of organizations run workloads on-premise in 2023 [13], with some organizations even shifting workloads back from the cloud in a trend known as *cloud repatriation* [19, 30].

Organizations choose on-premise Kubernetes deployments for various reasons. These include data security and compliance requirements in regulated industries [2, 28], cost considerations where cloud expenses become substantial [16], infrastructure control requirements, vendor lock-in avoidance [37], and low-latency requirements for edge computing scenarios [4]. Educational institutions face similar constraints. They also require cost predictability for multi-semester courses and control over sensitive student data. On-premise deployments also expose students to infrastructure challenges abstracted away by managed cloud services, providing an authentic experience with deployment patterns encountered across diverse industry contexts.

However, deploying and managing multi-node on-premise Kubernetes clusters presents technical challenges. Unlike single-node setups or cloud-managed services, multi-node on-premise deployments require expertise in infrastructure provisioning, network configuration, and storage management that may exceed available institutional resources [9, 27]. Administrators must balance providing industry-relevant infrastructure with maintaining usability for students and minimizing operational overhead for staff.

This experience report addresses these pedagogical and practical challenges of deploying and evaluating on-premise Kubernetes infrastructure for software engineering education. The report focuses on three key aspects: making informed technical decisions within educational constraints, minimizing operational overhead for teaching staff, and ensuring accessibility for students with varying Kubernetes experience levels. The primary contributions provide actionable guidance for educators considering similar deployments: (1) comparison of on-premise Kubernetes deployment approaches

[1] Cloud Native Computing Foundation Annual Survey 2023: https://www.cncf.io/reports/cncf-annual-survey-2023/

with rationales grounded in educational constraints, (2) component selection guidelines addressing the tension between industry authenticity and student accessibility, (3) a replicable deployment architecture with automation tools developed specifically to reduce operational overhead for resource-constrained educational environments, and (4) pedagogical insights from real-world student experiences, including both successful patterns and encountered challenges. To ensure broad applicability across diverse institutional contexts, the discussion scope is limited to free and open-source software solutions.

The authors validated this approach in an industry-oriented software engineering course at the Technical University of Munich (TUM) over two semesters, where student teams with minimal prior Kubernetes experience collaborated with industry partners (Siemens AG and Interhyp AG) on real-world projects requiring containerized deployments. The evaluation examines multiple dimensions of educational effectiveness, including: project success, required instructional support, administrative overhead, and infrastructure reliability.

Results show that students with minimal Kubernetes background successfully deployed microservice applications after a brief three hour introduction workshop, administrative overhead proved comparable to traditional VM-based approaches, and the infrastructure reliability met the demands of semester-long development projects. We provide a replication package including Ansible roles, configuration templates, and documentation to facilitate adoption by other educational institutions at the end of this paper.

The remainder of this paper is organized as follows: Section 2 discusses related work on Kubernetes deployments in educational and on-premise environments. Section 3 examines architectural considerations for multi-node Kubernetes clusters in educational settings and highlights deployment recommendations. Section 4 presents the concrete deployment approach with component selection rationale and automation tools. Section 5 evaluates the proposed architecture through a case study in a university software engineering course and extracts the lessons learned. Finally, Section 6 summarizes key findings and provides actionable recommendations for educators.

2 Related Work

Kubernetes is a broadly researched topic, and there is a wealth of literature we build upon in this paper. This section focuses on related work that (1) outlines the use in education and (2) addresses the challenges of deploying and maintaining multi-node Kubernetes clusters in on-premise environments.

De Smet et al. present a cloud-based JupyterHub platform running on Kubernetes for programming courses with automated grading [11]. Wang et al. describe a cloud-based DevOps learning platform with guided exercises for containerization and orchestration [38]. Spišaková et al. examine Kubernetes challenges in academic environments, focusing on scheduling and multi-tenancy issues [35]. Lui presents a K3s-based virtual lab platform for computing education emphasizing security isolation [23]. The approach in this study differs by providing direct on-premise cluster access for authentic infrastructure experience while maintaining control over resources and data.

Telenyk et al. compare the performance and resource utilization of Kubernetes and its lightweight alternatives (MicroK8s, K3S) in cluster lifecycle operations [36]. Prior work by Muddinagiri et al. [26], Böhm and Wirtz [3], and Koziolek and Eskandani [20] has focused on single-node Kubernetes deployments and lightweight distributions like K3S and MicroK8s for testing and resource-constrained environments. In contrast, we focus on multi-node Kubernetes deployments and associated maintenance challenges in on-premise environments.

Bryant et al. analyze various on-premise Kubernetes deployment methods (kubeadm with Kubespray, OpenShift/OKD, Rancher), highlighting trade-offs in resource requirements, flexibility, and ease of use [9]. Ruiz et al. and Mondal et al. focus on optimizing existing clusters for performance and quality of service [25, 32]. Dakić et al. develop specialized Kubernetes deployment methods for high-performance computing workloads [10].

Mušić and Fortuna propose Kubitect, an open-source tool for simplifying on-premise Kubernetes deployment in small, non-enterprise environments using Terraform and Kubespray [27]. Packard et al. describe deploying multi-node Kubernetes clusters on bare-metal servers using kubeadm and MetalLB for scientific computing [29]. The work in this study builds upon these findings but focuses on general-purpose deployment with detailed analysis of multi-node on-premise challenges.

3 On-Premise Kubernetes Architecture Considerations

This section provides an overview of available options for on-premise Kubernetes deployments and their auxiliary components, emphasizing their suitability for educational environments. The selection of Kubernetes components impacts both the learning experience for students and the operational burden on teaching staff. This section addresses considerations for selecting a Kubernetes distribution, orchestrator, provisioning tools, and auxiliary systems (networking, load balancing, storage) that balance three competing educational priorities: industry authenticity (exposing students to production-grade tools), student accessibility (minimizing complexity barriers for learners), and administrative feasibility (reducing setup and maintenance overhead for resource-constrained institutions).

The discussion focuses on component options that meet the requirements of the educational deployment context stated in Table 1. These requirements emerged from the specific constraints and goals of the educational deployment context. Institutional constraints drove several requirements: the university's data protection policies mandate on-premise deployment for student project data, budget limitations require exclusive use of free and open-source software, and limited IT staff availability necessitates ease of maintenance. We focus the discussion on multi-node deployments to support larger student cohorts and provide an authentic scaling behavior. This induces follow up requirements for secure cluster networking, load balancing and high availability.

Pedagogical goals shaped additional requirements: each student should be able to access the cluster using their existing university credentials to minimize onboarding friction, and a graphical user interface should be available to lower the entry barrier for students

new to Kubernetes. Default Kubernetes API objects (e.g., Service type LoadBalancer and Persistent Volumes) should be supported to ensure that students learn transferable skills applicable across different environments. The requirements follow the terminology defined by Bradner [6].

Table 1: Scope of Requirements for On-Premise Kubernetes Deployments

Requirement	Description
On-Premise Deployment	Kubernetes must support deployment on bare-metal servers or VMs within on-premise environments
Multi-node	The cluster must consist of multiple nodes
Load Balancing	Service type LoadBalancer for external access must be supported
Network Security	Cluster-internal traffic must be encrypted
High Availability	The cluster must ensure high availability of the control plane and cluster ingress
Access Control	Users should be able to use existing login credentials
Maintenance	Setup and maintenance must be as easy as possible
Persistent Storage	Persistent storage solutions should be available
Cost / Licenses	Only free and open-source software should be used
User Interface	A graphical user interface for cluster management should be available

Certain characteristics, while common in cloud-native environments, are not primary concerns for this on-premise deployment. Elasticity, the ability to dynamically scale the number of nodes based on demand, is typically a key feature of cloud-based Kubernetes but is not a strict requirement in this context.

3.1 Kubernetes Distributions

Selecting the appropriate Kubernetes distribution is crucial for ensuring an efficient, secure, and maintainable on-premise deployment in educational contexts. Various Kubernetes distributions provide different features, optimizations, and levels of complexity [9, 36]. The distribution choice directly impacts both student learning experiences and administrative workload. Ease of deployment affects the time investment required from administrators to establish and maintain the infrastructure. API compatibility with upstream Kubernetes ensures that students learn transferable skills applicable across different environments rather than distribution-specific peculiarities.

Security and compliance features matter for educational institutions managing student data under privacy regulations. Long-term maintainability influences the sustainability of the infrastructure across multiple semester cohorts. As one of the core requirements for this deployment is to use only free and open-source software, commercial distributions like OpenShift, Mirantis, or VMWare Tanzu were not considered. Several self-hostable Kubernetes distributions suit multi-node on-premise environments, each catering to different operational requirements. Table 2 summarizes the most relevant options.

Table 2: Comparison of Kubernetes distributions for on-premise deployments

Name	Description
Vanilla Kubernetes	The upstream Kubernetes project, maintained by the Cloud Native Computing Foundation (CNCF)
RKE2	A single-binary security-focused distribution optimized for regulatory compliance and automated installation, maintained by Suse Rancher Labs
k3s	A single-binary lightweight Kubernetes distribution designed for resource-constrained environments also developed by Suse. k3s is focused on edge computing and IoT applications but can also be used for on-premise clusters
K0s	A Kubernetes distribution optimized for edge computing and IoT applications, focusing on simplicity and minimalism maintained by Mirantis
MicroK8s	A Canonical-maintained Kubernetes distribution, optimized for easy installation and local development

Vanilla Kubernetes, maintained by the CNCF, offers broad API compatibility but demands extensive manual configuration of control plane components, etcd, and kubelet [9], making it less suitable for resource-constrained educational environments prioritizing ease of deployment.

RKE2 is a security-centered Kubernetes distribution, providing a lightweight yet fully compliant option focused on security and compliance. It uses a server-agent architecture where the server runs the control plane, etcd, kubelet, etc - and agents function as worker nodes. Its design optimizes automated installation and regulatory compliance, suiting on-premise deployments with stringent security needs.

MicroK8s, K0s, and k3s cater to resource-constrained settings like edge computing or IoT, utilizing a server-agent approach similar to RKE2. They operate as a single binary across x86 and ARM platforms, offering flexibility for diverse hardware setups. While MicroK8s allows multi-node clustering for production, it primarily supports straightforward local development and testing. Its installation on Linux via snap may not suit all distributions, yet it features native GPU support and optional enterprise support. K3s focuses on edge computing, CI, and IoT applications but remains viable for low-overhead on-premise deployment. K0s is also optimized to be used in production environments for bare metal and edge computing use cases.

Recommendation 1: RKE2 balances industry authenticity and administrative feasibility for institutions with limited IT staff and data protection requirements. Automated installation reduces setup time while maintaining full Kubernetes API compatibility for industry-transferable skills.

3.2 Kubernetes Orchestrators and Provisioning Tools

Installing Kubernetes on multiple nodes by hand is time-consuming and error-prone. Kubernetes orchestrators and provisioning tools automate the deployment and management of Kubernetes clusters, simplifying the process and ensuring consistency. In addition to the installation process, orchestrators may manage the lifecycle of the cluster, including scaling, upgrading, access control, and monitoring.

Table 3: Kubernetes orchestration and provisioning tools

Tool	Description
kubeadm	A Kubernetes-native tool for bootstrapping clusters, kubeadm is part of the Kubernetes project and the official way to create a vanilla cluster
Kubespray	A community-driven Kubernetes cluster configuration management tool, Kubespray is based on Ansible and supports multiple cloud providers and on-premise environments
Rancher	Rancher is a complete software stack for teams using Kubernetes, providing a web UI to adopt, create and manage clusters
Ansible	A configuration management tool that is used to configure hosts declaratively
Specific Installer	Some Kubernetes distributions provide their own installation tools that manage the deployment process

Kubeadm initializes control plane components but requires manual networking and storage configuration, offering flexibility at the cost of high setup complexity.

Kubespray automates multi-node deployment using Ansible playbooks with built-in high availability and security, reducing setup complexity while requiring substantial Kubernetes knowledge.

Rancher is an enterprise-ready Kubernetes management platform that simplifies cluster deployment and operations. It allows automatic setup of RKE2-based clusters. Rancher provides a graphical user interface (GUI) with multi-cluster management capabilities, enabling administrators to create, manage, and monitor multiple student clusters from a unified dashboard. Key features for educational scaffolding include centralized access control across all managed clusters and project-level role-based access control (RBAC) that enables fine-grained permission management for different student groups.

This combination allows administrators to efficiently provision isolated environments for individual students or teams while maintaining appropriate access levels. Rancher also integrates security policies and automated upgrades, making it well-suited for large-scale educational deployments. Rancher runs in a Kubernetes cluster itself, which simplifies its deployment, but its installation requires a running Kubernetes cluster first.

Ansible is an infrastructure automation tool that may be used to provision and configure Kubernetes clusters. It allows users to define declarative infrastructure-as-code definitions, ensuring repeatable and consistent deployments. Ansible appears here as prebuilt roles and playbooks are available for Kubernetes deployment, making it a popular choice for custom installations. For environments where Ansible is already in use, it may be the preferred choice for Kubernetes deployments.

Some Kubernetes distributions provide their own installers to streamline deployment. K0s offers a set of tools to deploy its single-binary Kubernetes nodes without requiring extensive configuration. MicroK8s, installs via snap, making it accessible for developers and small-scale deployments.

> **Recommendation 2:** Combine Ansible with Rancher for educational deployments. Ansible enables reproducible infrastructure-as-code deployments. Rancher's GUI reduces the learning curve for novice students through visual cluster state feedback.

3.3 High Available Control Plane

The control plane of a Kubernetes cluster is responsible for managing the cluster state, scheduling workloads, and maintaining communication between nodes. Ensuring high availability of the control plane is essential for maintaining cluster stability and resilience against failures. The foundation for a highly available control plane is a redundant cluster database (etcd for most Kubernetes distributions). Other control plane components (API server, scheduler, controller manager) must also be deployed on multiple nodes. These constraints are unique to the Kubernetes distribution and orchestrator used. The remaining challenge is to ensure a highly available API endpoint for clients to connect to the control plane. There are several approaches to ensure transparent failover between the control plane nodes, outlined in Table 4.

Table 4: Approaches for ensuring control plane high availability

Approach	Description
DNS RR	DNS round-robin to distribute requests across multiple control plane nodes [7, 18]
Load Balancer	A cluster external load balancer distributes incoming requests across multiple control plane nodes
Virtual IP	A virtual IP is assigned to the control plane nodes, allowing clients to connect to the cluster using a single IP address. If one of the control plane nodes fails, the IP is reassigned to another node internally.

DNS round-robin distributes requests across control plane nodes but lacks immediate failover mechanisms. External load balancers provide flexible traffic distribution but require additional infrastructure and introduce operational complexity. Virtual IP (VIP) using Kube-VIP assigns a single IP that automatically reassigns on node failure using ARP [1] or BGP [31], running within the cluster to simplify setup and maintenance.

> **Recommendation 3:** Use Kube-VIP for virtual IP-based high availability. It operates within the cluster, reducing administrative overhead while demonstrating industry-standard control plane reliability practices.

3.4 Cluster Networking and Load Balancing

Networking in Kubernetes is fundamental for enabling communication between pods, services, and external clients. Kubernetes networking follows a flat network model, where every pod is assigned a unique IP address and can communicate with other pods without the need for Network Address Translation. However, the implementation details vary depending on the selected container networking interface (CNI) plugin and the method used for external load balancing. Several CNI plugins exist, each catering to different operational and security needs. This section focuses on three popular options listed in Table 5.

Table 5: Container Network Interface plugins for Kubernetes

Plugin	Description
Flannel	A simple and lightweight overlay network primarily using VXLAN to encapsulate traffic between nodes.
Calico	A policy-driven networking solution that supports both overlay and non-overlay modes.
Cilium	A modern networking plugin built on eBPF, offering advanced observability, security enforcement, and high-performance networking.

Flannel is a straightforward overlay network that encapsulates traffic between nodes using VXLAN, providing a simple and lightweight solution for Kubernetes networking. It is well-suited for environments where minimal configuration and ease of use are essential, offering a balance between performance and simplicity.

Calico is a policy-driven networking solution that supports both overlay and non-overlay modes, providing flexibility for different deployment scenarios. Calico offers built-in network policies that enable fine-grained security controls and encryption channels for cluster traffic, making it ideal for environments requiring strict security enforcement and access controls.

Cilium is a modern networking plugin built on eBPF, offering advanced observability, security enforcement, and high-performance networking. It provides deep packet inspection and security monitoring capabilities, making it suitable for environments with stringent security requirements and performance-sensitive workloads.

Recommendation 4: Calico balances security and operational overhead for institutions with data protection requirements. Encrypted overlay networks satisfy data protection policies; network policies enable students to learn production-grade security concepts.

Beyond the CNI plugin handling internal cluster traffic, a component supporting the `ServiceType` LoadBalancer is required to natively expose services to external clients. While cloud providers offer managed load balancers, on-premise deployments require additional tools to manage external traffic distribution. The most common solution for this is MetalLB, a software load balancer that integrates with Kubernetes and provides Layer 2 and Layer 3 load balancing capabilities. MetalLB can be configured to use either ARP [1] or BGP [31] to announce service IPs to the network, allowing external clients to access services running within the cluster.

Recommendation 5: Use MetalLB for LoadBalancer service support in on-premise deployments. It provides cloud-equivalent abstractions ensuring industry transferability. Layer 2 mode (ARP/NDP) requires minimal network configuration while exposing students to production patterns.

3.5 Storage

Kubernetes offers native abstractions for persistent storage in the form of Persistent Volumes (PVs) and Persistent Volume Claims (PVCs). These abstractions allow users to request storage resources and dynamically provision them based on predefined storage classes.

Storage solution selection presents a particularly complex decision point for an educational on-premise Kubernetes deployment. Operational considerations matter significantly: complex storage solutions may exceed available IT staff expertise or time budgets, while other solutions may have negative impact on storage performance.

Educational institutions face a strategic choice between leveraging existing storage infrastructure versus deploying Kubernetes-native storage solutions. Institutions with established storage infrastructure (NFS, iSCSI, Ceph) can integrate these systems via Container Storage Interface (CSI) drivers, avoiding the operational overhead of managing additional storage systems while potentially sacrificing some Kubernetes-native storage features. Institutions without existing storage infrastructure must deploy storage solutions specifically for Kubernetes, accepting additional operational complexity in exchange for tighter integration with Kubernetes abstractions.

This section examines both approaches, focusing on the storage solutions listed in Table 6.

Table 6: Storage solutions for Kubernetes persistent volumes

Solution	Description
Rook	A Kubernetes-native storage orchestrator that automates the deployment and management of Ceph
Longhorn	A lightweight, cloud-native distributed block storage system designed for Kubernetes
OpenEBS	A container-native storage solution offering different storage engines, including local persistent volumes and replicated storage across nodes

For Kubernetes-native distributed storage, several options exist with different complexity-capability trade-offs. Ceph represents the mature, feature-rich end of the spectrum, offering high scalability, redundancy, and performance characteristics suitable for production enterprise workloads. Rook simplifies Ceph deployment within Kubernetes through operator-based automation, though it still requires significant expertise to configure and troubleshoot. The operational complexity of Rook/Ceph may exceed the capacity of educational IT staff, particularly when storage issues arise requiring deep system knowledge to diagnose.

Longhorn and OpenEBS cater to smaller-scale deployments where lightweight storage management and minimal administration overhead are priorities. These solutions reduce operational complexity

compared to Ceph while providing adequate functionality for typical educational workloads. All Kubernetes-native solutions use node local storage to build distributed storage systems, meaning storage performance and capacity depend on the node hardware provisioned for the cluster. For educational deployments, this approach requires either dedicating storage-optimized hardware or accepting storage performance limitations.

Alternatively, if external storage solutions already exist in the institutional environment, Kubernetes can leverage these systems via the Container Storage Interface (CSI). CSI provides a standardized integration mechanism, allowing Kubernetes to utilize existing storage infrastructure managed by experienced IT staff outside the Kubernetes cluster. This approach separates storage operations from Kubernetes cluster management, potentially reducing the expertise burden on those maintaining the Kubernetes infrastructure. CSI drivers are available for various storage systems, including NFS, iSCSI, and Ceph

From a pedagogical perspective, the storage choice is not as important as the other components discussed in this section. The Kubernetes abstraction of Persistent Volumes and Persistent Volume Claims remains consistent regardless of the underlying storage solution. This is why we prioritize operational feasibility over pedagogical fidelity in the storage decision.

Recommendation 6: Leverage existing external storage via CSI drivers when available to minimize operational overhead.
For institutions without existing storage, consider Longhorn as a cluster-internal solution with low operational overhead.

4 On-Premise Kubernetes Deployment

This section presents the practical implementation of the on-premise Kubernetes deployment. The deployment description documents the technical architecture for replication and explains the rationale behind specific choices to help educators adapt the approach to their institutional constraints.

The selected component combination (RKE2, Ansible, Rancher, Calico, external Ceph storage) reflects trade-offs specific to the institutional context: institutional data protection requirements, limited IT staff availability, budget constraints requiring open-source solutions, and the need to support students with varying Kubernetes experience levels. Different institutions may face different constraint combinations, requiring different component selections while following similar decision-making processes outlined in the previous section.

4.1 Target Environment

The deployment runs on a four-node Proxmox cluster at TUM with high-performance hardware (AMD EPYC processors, 512GB RAM per node, 4x10 Gbit networking). The deployment leverages existing university infrastructure: centralized DNS/DHCP services, hosted Ceph storage for persistent volumes, and dual-stack IPv4/IPv6 networking with floating IPs for high availability. IP resolution uses ARP [1] and NDP [34] as BGP peering is unavailable. Based on the analysis in Section 3, we selected components shown in Table 7.

Table 7: Selected Kubernetes Components

Component	Selected Solution
Kubernetes Distribution	RKE2 v1.27.9+rke2r1
Deployment Method	Custom Ansible role
Management	Rancher
Control Plane HA	Kube-VIP v0.6.4
Networking	Calico v3.26.3
Load Balancing	MetalLB v0.14.5
Storage	External Ceph via CSI

We developed a custom Ansible role[2] to implement the RKE2-based architecture defined in the previous section. This automation eliminates manual configuration errors and enables consistent deployments across multiple student cohorts. The combination of automated deployment with Rancher GUI offers a good balance of cluster configuration options and easy access for students. We opted for Ansible as the deployment method to set up all clusters similarly, including the Rancher management cluster which cannot be deployed via Rancher itself.

The network infrastructure is realized using Calico, selected for its advanced networking capabilities and security benefits. This choice enables the assignment of both IPv4 and IPv6 addresses to pods and services, ensuring compatibility with the dual-stack environment. Additionally, Calico provides network policy features that allow for fine-grained control over pod-to-pod communication as well as encrypted cluster traffic by default. MetalLB serves as the load balancer implementation, operating in Layer 2 mode using ARP/NDP announcements. It is configured to allocate two public dual-stack IP pairs: one dedicated to the ingress controller for HTTP/HTTPS traffic and one for additional services requiring public IP accessibility.

The storage infrastructure utilizes the university's Ceph clusters through the Ceph CSI driver, providing reliable persistent storage backed by an established infrastructure.

Based on these components, the deployment involves two clusters: a Rancher management cluster and an application cluster. The management cluster is responsible for managing the application clusters, while the application cluster hosts the workloads.

4.2 Cluster Deployment

Rancher Management Cluster. The Rancher management cluster consists of three VMs (4 vCPUs, 16GB RAM each) deployed across different physical hosts for robustness. Deployment uses the RKE2 Ansible role with all nodes serving as both control plane and worker. The cluster uses NGINX ingress controller with cert-manager for Let's Encrypt TLS certificates. Rancher is installed via Helm chart with custom values for dual-stack networking and integrates with Keycloak for authentication using the university's identity management infrastructure.

Application Cluster. The cluster architecture can be instantiated and scaled to support multiple application specific clusters. A typical application cluster deployment requires approximately two to three hours from initial VM provisioning to a fully operational

[2] https://github.com/Mtze/rke2-ansible

state with all components installed and configured. This cluster design was instantiated as a dedicated downstream applications cluster to be used by students to run their workloads.

A downstream cluster consists of three control plane nodes with 4 vCPUs / 12GB RAM, and a variable number of worker nodes, depending on the workload requirements. For the application cluster, the deployment uses five worker nodes, each provisioned with 8 vCPUs / 16GB RAM. The cluster deployment is also based on the RKE2 Ansible role. After the base cluster is operational, it is imported into Rancher using its generic cluster registration process.

The deployment provisions each application cluster with a standard set of components to provide essential functionality: The cluster deploys the NGINX ingress controller for managing external access to applications, along with cert-manager for automated TLS certificate management. The external DNS system is configured with a wildcard DNS subdomain entry, which points to the ingress controller's public floating IP address, allowing for dynamic DNS updates for all services exposed through the ingress. Storage provisioning utilizes the external Ceph cluster through the CSI driver, defined as the default storage class. Cluster monitoring is implemented using the Prometheus Operator deployed through Rancher's monitoring integration.

Rancher's integration with Keycloak manages access control. The system provides Rancher access on a project level, which are mapped to Keycloak groups. This allows for a flexible and easy-to-use access control mechanism, enabling the addition of new students and groups without requiring changes to the Kubernetes cluster configuration.

With this infrastructure deployed and operational, we conducted a case study to assess the educational effectiveness and practical feasibility of the on-premise Kubernetes approach.

5 Case Study - On-Premise Kubernetes in a Project-based University Course

In this section, we present an experience report in the form of a case study to evaluate the proposed on-premise Kubernetes architecture in an educational-industrial collaboration setting. This section describes the course environment and structure, and analyzes the feasibility, usability, and required support effort of the architecture through a mixed-methods approach. The evaluation is based on three student projects conducted in cooperation with industry partners, providing insights into the practical application of Kubernetes in real-world scenarios. This study is based on the case study reporting guidelines from Martinsuo and Huemann [24].

5.1 Case Study Context

The authors conducted the evaluation in the context of the iPraktikum project-based course at TUM over the course of two semesters. The iPraktikum is a project-based course where students work in teams of 6-8 students on a project from industry partners [8]. The evaluation involved 22 students across three project teams (one team of 6 students and two teams of 8 students each). These partners provide a problem statement and act as customers for the project. The course organizers assemble student teams based on preferences, skills, and team diversity criteria (Internationality, Gender, Experience, etc.) [14, 15]. Each team is supported by an agile coach and a project lead acting as scrum master and product owner respectively.

Cross-cutting *Workflows* manage loosely coupled specialized inter-team roles for deployment, UI/UX, and software architecture. One student from every team is assigned to a workflow to foster cross-team feedback and a knowledge transfer culture. Each workflow is supported by external experts in the respective domain to give guidance and support. This course structure closely resembles an agile matrix organization, where the teams are cross-functional and self-organizing. The course provides students with a realistic project experience and prepares them for industry careers by focusing on soft skills, such as teamwork, communication, and project management in addition to technical development skills.

5.1.1 Case Description. The authors conducted the case study in three projects: two with Siemens AG as industry customer and one with Interhyp AG as customer. Each project included the development of a client-server application. The customers required the server to be containerized and the client should be implemented as an iOS application. Siemens required to use Kubernetes as part of their problem statements. Interhyp did not have a Kubernetes requirement but the students decided to use it for their deployment. All teams used the same on-premise Kubernetes cluster but got individual Rancher projects for their team. Team members accessed Rancher using their university credentials. Within this Rancher project, they were free to create namespaces and deploy their applications. The architecture and the deployment strategy for their server implementation were designed by the students without additional architectural constraints from the course organizers.

Students participated in a in a three hour Kubernetes workshop at the beginning of the semester. The workshop covered the basics of Kubernetes, such as Pods, Deployments, Services, Ingresses, and Helm, as well as a transition strategy from Docker to Kubernetes. The authors conducted the workshop and designed it to provide the students with the necessary knowledge to develop their applications on Kubernetes. Students were encouraged to use the Kubernetes documentation and other resources to deepen their knowledge.

This setting provides a valid test environment to assess the cluster functionality and usability through industry-focused team projects with varied student experience levels [5], mirroring challenges faced by industry development teams [17].

5.1.2 Research Objectives. The evaluation assesses architecture feasibility, identifying challenges and benefits through five research questions: **RQ1:** Is the suggested Kubernetes architecture working as expected in a real-world scenario? **RQ2:** Is the suggested Kubernetes architecture accessible for novice students? **RQ3:** How was the overall student experience with the Kubernetes architecture? **RQ4:** How much support from experts is required to use the on-premise Kubernetes cluster? **RQ5:** How much organizational overhead is induced by providing on-premise Kubernetes clusters compared to Docker or VM-based deployments?

5.1.3 Research Method. The evaluation uses a mixed-method approach with semi-structured interviews and team observations. Post-project five interviews[3] examined student experience with Kubernetes, challenges faced, and required support, covering prior

[3] Interview questions contained in the reproducibility package

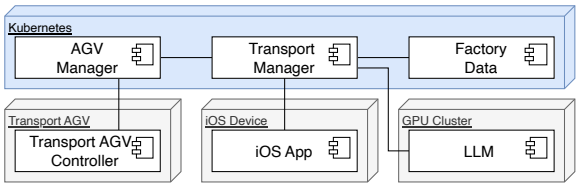


Figure 1: Deployment Diagram of the Siemens Project 2024

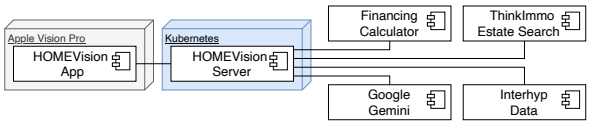


Figure 2: Deployment Diagram of the Interhyp Project 2024/2025

experience, cluster usability, and overall reflections. The interviews lasted 20-30 minutes each and were conducted by one of the authors taking notes and later summarized in the results. Development processes were observed throughout the semester, documenting challenges and support requirements.

5.2 Project 1 - Siemens - Summer Term 2024

Students developed an AI-driven interface enabling natural language interaction between factory workers and automated guided vehicles (AGVs). The system transforms voice or text commands into machine-readable transport jobs, supporting factory jargon and multiple languages to increase adaptability and user-friendliness in industrial logistics.

5.2.1 Application Architecture. The architecture (Figure 1) includes an iOS application for speech-to-text conversion and chat interface, communicating with a *Transport Manager* that interprets input using LLM and factory data (layout, machine locations, jargon). Transport jobs are sent to an *AGV Manager* for scheduling and then to physical AGVs via an *AGV Controller*.

The *Transport Manager*, *Factory Data*, and *AGV Manager* components run as deployments on the on-premise Kubernetes cluster. Communication between the components is facilitated by Kubernetes services. The *Transport AGV Controller* component is deployed on the physical AGV and communicates with the *AGV Manager* component via an MQTT broker. The iOS application communicates with the *Transport Manager* component via a REST API exposed via a Kubernetes Ingress.

5.2.2 Results. The team completed the project and deployed their application on the on-premise Kubernetes cluster.

Observations show that the students used only basic Kubernetes features such as Deployments, Services, and Ingresses. They struggled in the beginning to understand the Kubernetes concepts, and it took them some time to get an initial deployment working. The students used Helm to package their application and to deploy it for production use, but also used it to create separate test environments. They set up continuous integration to build all container images but did not set up continuous deployment. The team successfully accessed the cluster and used the Rancher UI without issues. Students successfully used kubectl for deployment and debugging after initial onboarding. After the initial setup, the team used the deployment strategy regularly to deploy new versions or variants of their application.

Two students from the team were interviewed after the project was completed. The students reported minimal prior Kubernetes experience and indicated that the workshop provided sufficient foundational knowledge to begin development. Students indicated in interviews that the workshop duration was insufficient, specifically requesting more comprehensive coverage of debugging and troubleshooting techniques. Initial deployment encountered permission issues with Ingress resources, which course organizers resolved promptly. Interview data revealed that students used the Rancher UI for cluster state visualization but primarily relied on kubectl for deployment and debugging operations. Both students reported satisfaction with the on-premise Kubernetes cluster performance and reliability, expressing willingness to use it for future projects based on observed usability after initial setup. One student reported that cluster interaction patterns were indistinguishable from cloud provider experiences.

5.3 Project 2 - Interhyp - Winter Term 2024/2025

Students developed *HOMEvision*, a smart real estate assistant combining natural language processing with AR visualization. An LLM translates user descriptions into search parameters, analyzes soft criteria (quiet surroundings, accessibility), and presents results through Apple Vision Pro or iPad AR interfaces.

5.3.1 Application Architecture. The architecture of the Interhyp project is shown in Figure 2. They developed a client-server application with a REST API as the communication interface. The client application is a visionOS application that uses the REST API to communicate with the server. The golang-based server offers a unified REST API to request data from the *ThinkImmo Estate Search* and the *Interhyp Data* APIs and processes the responses using *Google Gemini* and an external *Financing Calculator* to provide the user with a personalized real estate search result.

They use one deployment and one service to run the *Interhyp Server* component and an Ingress to expose the REST API to the client application. Since their server implementation is stateless, they scaled the deployment to multiple replicas to deal with higher API load. The team decided against using Helm and applied their configuration directly to the cluster using kubectl.

5.3.2 Results. The team completed the project and deployed their application on the on-premise Kubernetes cluster. Observations show that the team only used minimal Kubernetes features but got their initial deployment working within a few hours without any support from the course organizers. The only active support required was to set up a custom DNS entry for their Ingress outside the Kubernetes cluster and the corresponding TLS certificate.

One student from the team was interviewed after the project was completed. The interviewed student reported no previous Kubernetes experience and could not attend the workshop due to scheduling conflicts. They used the workshop materials and the Kubernetes documentation to get started. The student reported finding

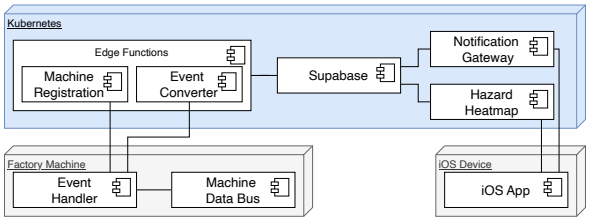


Figure 3: Deployment Diagram of the Siemens Project 2024/2025

the Rancher UI challenging for troubleshooting but valuable for visualizing changes performed via kubectl. Observations showed the team encountered difficulties during initial setup, specifically with Ingress configuration and TLS certificate management for their REST API. They resolved the issue together with the course organizers. After the initial setup, the team reported positive experience with the on-premise Kubernetes cluster, characterizing deployment operations as straightforward. The student reported that Kubernetes deployment proved less complex and more efficient compared to Virtual Machine deployment in their experience.

5.4 Project 3 - Siemens - Winter Term 2024/2025

Students developed a connected environmental protection, health management and safety (EHS) Assistant for real-time workplace safety in industrial environments. The system integrates machine data from Siemens Industrial Edge devices to detect and respond to environmental, health, and safety hazards, warning workers of potential dangers in dynamically changing factory settings.

5.4.1 Application Architecture. The architecture (Figure 3) includes edge *Event Handlers* on machines translating sensor data into safety events, forwarded to an *Event Hazard Converter* that evaluates events based on worker preferences and produces warnings. A *Push Notification Gateway* sends alerts to workers' iOS devices, while a *Hazard Heatmap* visualizes hazards on factory floor maps.

They used Helm to install the off-the-shelf component Supabase to provide a database and authentication service for their application. They also built a Helm chart for their own applications, including the *Push Notification Gateway* and the *Hazard Heatmap*. For both components, they used a Deployment and a Service to run the application and expose it to the cluster. They deployed an Ingress to expose the *Hazard Heatmap* to the client application. To manage the application settings, they employed Kubernetes ConfigMaps and Secrets.

5.4.2 Results. The team completed the project and deployed their application on the on-premise Kubernetes cluster.

The resulting system is based on a microservice architecture with multiple components that communicate with each other, which is well suited for a Kubernetes deployment. Observations showed the team deployed their own components to the cluster efficiently with minimal support. They used an off-the-shelf foundation (Supabase) which offered a Helm chart for the deployment. The Supabase Helm chart did not support dual-stack deployments, which resulted in debugging challenges that students reported as difficult to resolve independently. Instructor intervention was required to resolve the deployment issues.

Two students from the team were interviewed after the project was completed. Interview participants reported theoretical Kubernetes knowledge without prior practical experience. One student attended the workshop and reported it provided adequate initial orientation, though insufficient time for comprehensive understanding of Kubernetes features. The other student was not able to attend but used the workshop materials over the course of the semester. The team used Rancher as their primary visualization tool but requested more comprehensive introduction to its capabilities. The team reported no difficulties with their own deployments but encountered significant challenges with the Supabase Helm chart, comparing it unfavorably to their prior Docker deployment experience. Interview data indicated students preferred the Kubernetes deployment over Virtual Machines, citing reduced deployment complexity and faster iteration cycles. Both students expressed willingness to use the on-premise Kubernetes cluster for future projects.

5.5 Discussion

Based on the case study experiences, this section analyzes the research questions, examining successes and challenges to derive pedagogical insights for educators considering similar deployments.

RQ1 The Kubernetes architecture demonstrated functional reliability throughout both semesters, supporting three project teams. Students utilized LoadBalancer services, Ingress resources, and persistent storage without encountering on-premise-specific limitations, developing deployment practices directly transferable to cloud environments.

However, the dual-stack IPv4/IPv6 network configuration introduced friction when students deployed third-party Helm charts. The Supabase deployment in Project 3 assumed IPv4-only networking, causing connectivity failures requiring instructor intervention. This reveals a key trade-off: advanced configurations provide infrastructure authenticity but may introduce troubleshooting overhead when intersecting with immature third-party software.

> **Lesson 1:** The suggested architecture works as expected - all required features were available to the students.

> **Lesson 2:** Dual-stack networking increases authenticity but may cause third-party software compatibility issues. Educators must balance real-world complexity against troubleshooting overhead.

RQ2 The architecture demonstrated accessibility for students with minimal prior Kubernetes experience. All participating students deployed containerized applications, with the Rancher graphical interface playing a crucial role during initial learning. Students used Rancher for gaining situational awareness during deployment failures and understanding resource relationships.

An interesting usage pattern emerged: students initially relied on Rancher's graphical forms but transitioned to kubectl and manifest files as expertise developed, using Rancher primarily for monitoring. This progression demonstrates Rancher's effectiveness as learning scaffolding that does not prevent adoption of command-line workflows.

However, Kubernetes complexity did not disappear through infrastructure design. The three hour workshop provided minimal viable knowledge, but substantial learning occurred through semester-long iterative deployment. Infrastructure design and graphical tools reduce accessibility barriers but complement rather than replace instructional scaffolding.

> **Lesson 3:** Rancher reduces initial learning barriers through visual cluster state feedback. It serves as effective scaffolding for developing Kubernetes mental models.

> **Lesson 4:** Students naturally transition from graphical tools to command-line workflows as expertise develops.

> **Lesson 5:** With an initial introduction to Kubernetes, even novice students can effectively use the on-premise Kubernetes cluster as deployment target.

RQ3 Student experience with the on-premise Kubernetes cluster proved positive, with all interviewed students expressing willingness to use similar infrastructure for future projects. Students particularly valued deployment reproducibility, enabling reliable environment recreation and parallel test instances without manual server configuration.

An unexpected pattern emerged: teams using Kubernetes consistently maintained multiple parallel deployment environments (development, testing, production), while VM-based deployments in previous iterations typically used single shared environments. Kubernetes' declarative configuration reduced the perceived cost of creating environments sufficiently that students organically adopted professional multi-environment practices without explicit requirement—a valuable implicit learning outcome.

> **Lesson 6:** Declarative configuration reduces deployment effort, enabling students to organically adopt multi-environment practices (dev/test/production) without explicit requirements.

> **Lesson 7:** Even for simple application architectures where Kubernetes is not strictly a necessity, students found Kubernetes to be a good fit as deployment target.

RQ4 Required support proved moderate: the workshop provided minimum viable onboarding to enable students to begin work, though it was not optimal—students reported insufficient time for deeper understanding of debugging and troubleshooting. Despite this limitation, students asked only occasional questions about best practices and solved most issues independently after initial setup, requiring less continuous support than expected.

> **Lesson 8:** Successful Kubernetes integration requires structured initial instruction, but students achieve substantial self-sufficiency for routine deployment operations after this foundation.

RQ5 Organizational overhead proved lower than VM-based deployments. While initial cluster setup is comparable to per-team VM provisioning, Kubernetes reduces ongoing support burden as students leverage standard documentation rather than institution-specific configurations. Kubernetes competencies also provide greater industry transferability than specialized VM setups.

> **Lesson 9:** The initial setup of the on-premise Kubernetes cluster following the architecture takes two to three hours.

> **Lesson 10:** Kubernetes requires higher initial investment in automation and onboarding than VM deployments, but reduces ongoing operational overhead through standardization and student self-sufficiency.

This experience report acknowledges several limitations. First, the study did not investigate elasticity features such as auto-scaling or dynamic node provisioning, focusing instead on pre-provisioned infrastructure with fixed resource allocation. Second, the case study was limited in scale, involving only three projects over two semesters with a moderate number of participants, all categorized as juniors in terms of industry experience. Third, the evaluation was conducted in a single institutional context with specific infrastructure constraints and may not represent the diversity of educational environments. Finally, all findings are anecdotal based on qualitative observations and interviews, lacking quantitative metrics for performance, cost, or learning outcomes, and may not generalize to other educational contexts or student populations.

6 Conclusion

This paper examines open-source options for production-grade on-premise Kubernetes deployments and introduces an architecture prioritizing maintainability and accessibility for educational contexts. The case study demonstrates viability using an RKE2-based deployment with automated provisioning, enabling students with minimal prior experience to successfully deploy containerized applications.

Future work includes: publishing workshop materials to support educator adoption, investigating automated elasticity for resource optimization, and expanding evaluation to industry settings for broader applicability validation.

Replication Package

The authors provide a replication package including Ansible roles, configuration templates, and documentation to facilitate adoption by other institutions [22]. The replication package is available online[4].

Acknowledgment

We would like to thank our industry partners at Siemens AG (Jochen Nickles) and Interhyp AG (Felipe Wiemann and Xaver Braun) for their collaboration and support in this project.

In accordance with ACM/IEEE guidelines, we disclose that we used GitHub Copilot, Claude, and ChatGPT to assist with grammar enhancements, reformulations, and general text editing throughout this paper. We have carefully reviewed the content to ensure it meets the standards of academic integrity.

[4]https://github.com/Mtze/paper-on-premise-k8s-replication-package/releases/tag/seet26

References

[1] 1982. *RFC 826 - An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.Bit Ethernet Address for Transmission on Ethernet Hardware.* Technical Report 826. RFC Editor. doi:10.17487/RFC0826

[2] Amina Ahmed Nacer, Claude Godart, Guillaume Rosinosky, Abdelkamel Tari, and Samir Youcef. 2019. Business Process Outsourcing to the Cloud: Balancing Costs with Security Risks. *Computers in Industry* 104 (Jan. 2019), 59–74. doi:10.1016/j.compind.2018.10.003

[3] Sebastian Böhm and Guido Wirtz. 2021. Profiling Lightweight Container Platforms: MicroK8s and K3s in Comparison to Kubernetes.. In *ZEUS*. 65–73.

[4] Sebastian Böhm and Guido Wirtz. 2022. Cloud-Edge Orchestration for Smart Cities: A Review of Kubernetes-based Orchestration Architectures. *EAI Endorsed Transactions on Smart Cities* 6, 18 (May 2022), e2. doi:10.4108/eetsc.v6i18.1197

[5] P. Bourque, L. Buglione, A. Abran, and A. April. 2003. Blooms Taxonomy Levels for Three Software Engineer Profiles. In *Eleventh Annual International Workshop on Software Technology and Engineering Practice*. IEEE, Amsterdam, The Netherlands, 123–129. doi:10.1109/STEP.2003.6

[6] Scott O. Bradner. 1997. *RFC 2119 - Key Words for Use in Rfcs to Indicate Requirement Levels.* Technical Report 2119. RFC Editor. doi:10.17487/RFC2119

[7] Thomas P. Brisco. 1995. *RFC 1794 - DNS Support for Load Balancing.* Technical Report 1794. RFC Editor. doi:10.17487/RFC1794

[8] Bernd Bruegge, Stephan Krusche, and Lukas Alperowitz. 2015. Software Engineering Project Courses with Industrial Clients. *ACM Transactions on Computing Education* 15, 4 (Dec. 2015), 1–31. doi:10.1145/2732155

[9] Lincoln Bryant, Robert W. Gardner, Fengping Hu, David Jordan, and Ryan P. Taylor. 2024. Kubernetes Deployment Options for On-Prem Clusters. doi:10.48550/arXiv.2407.01620 arXiv:2407.01620 [physics]

[10] Vedran Dakić, Mario Kovač, and Jurica Slovinac. 2024. Evolving High-Performance Computing Data Centers with Kubernetes, Performance Analysis, and Dynamic Workload Placement Based on Machine Learning Scheduling. *Electronics* 13, 13 (July 2024), 2651. doi:10.3390/electronics13132651

[11] Ruben De Smet, Steffen Thielemans, Jan Lemeire, An Braeken, and Kris Steenhaut. 2022. Educational Software-as-a-Service Based on JupyterHub and Nbgrader Running on Kubernetes. In *2022 IEEE 9th International Conference on E-Learning in Industrial Electronics (ICELIE)*. 1–6. doi:10.1109/ICELIE55228.2022.9969419

[12] Yuri Demchenko, Zhiming Zhao, Jayachander Surbiryala, Spiros Koulouzis, Zeshun Shi, Xiaofeng Liao, and Jelena Gordiyenko. 2019. Teaching DevOps and Cloud Based Software Engineering in University Curricula. In *2019 15th International Conference on eScience (eScience)*. IEEE, San Diego, CA, USA, 548–552. doi:10.1109/eScience.2019.00075

[13] Douglas Donnellan, Daniel Bizo, Jacqueline Davis, Andy Lawrence, Dr Owen Rogers, Lenny Simon, and Max Smolaks. 2023. Uptime Institute Global Data Center Survey 2023. (2023).

[14] Dora Dzvonyar, Lukas Alperowitz, Dominic Henze, and Bernd Bruegge. 2018. Team Composition in Software Engineering Project Courses. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials (Seem '18)*. Association for Computing Machinery, New York, NY, USA, 16–23. doi:10.1145/3194779.3194782

[15] Dora F Dzvonyar. 2019. *TEMPO: A Framework for Team Composition and Management in Project-Based Organizations.* Ph. D. Dissertation.

[16] Ferretti Marco and Santangelo Luigi. 2020. Cloud vs On-Premise HPC: A Model for Comprehensive Cost Assessment. In *Advances in Parallel Computing*. IOS Press. doi:10.3233/APC200026

[17] Vahid Garousi, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer. 2019. Aligning Software Engineering Education with Industrial Needs: A Meta-Analysis. *Journal of Systems and Software* 156 (Oct. 2019), 65–83. doi:10.1016/j.jss.2019.06.044

[18] Arnt Gulbrandsen and Dr. Levon Esibov. 2000. *RFC 2782 - A DNS RR for Specifying the Location of Services (DNS SRV).* Technical Report 2782. RFC Editor. doi:10.17487/RFC2782

[19] Kiran Jewargi. 2023. Public Cloud to Cloud Repatriation Trend. *Scholars Journal of Engineering and Technology* 11, 1 (Jan. 2023), 1–3. doi:10.36347/sjet.2023.v11i01.001

[20] Heiko Koziolek and Nafise Eskandani. 2023. Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, K3s, K0s, and Microshift. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. ACM, Coimbra Portugal, 17–29. doi:10.1145/3578244.3583737

[21] Kati Kuusinen and Sofus Albertsen. 2019. Industry-Academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, Montreal, QC, Canada, 23–27. doi:10.1109/ICSE-SEET.2019.00011

[22] Matthias Linhuber, Jan Philip Bernius, and Stephan Krusche. 2025. On-Premise Kubernetes in Software Engineering Education: An Experience Report - Replication Package. doi:10.17605/OSF.IO/G8TSW

[23] Richard Lui. 2024. A Secure and Scalable Virtual Lab Platform for Computing Education. *International Journal of Information and Education Technology* 14, 1 (2024), 59–64. doi:10.18178/ijiet.2024.14.1.2024

[24] Miia Martinsuo and Martina Huemann. 2021. Reporting Case Studies for Making an Impact. *International Journal of Project Management* 39, 8 (Nov. 2021), 827–833. doi:10.1016/j.ijproman.2021.11.005

[25] Subrota Kumar Mondal, Zhen Zheng, and Yuning Cheng. 2024. On the Optimization of Kubernetes toward the Enhancement of Cloud Computing. *Mathematics* 12, 16 (Aug. 2024), 2476. doi:10.3390/math12162476

[26] Ruchika Muddinagiri, Shubham Ambavane, and Simran Bayas. 2019. Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube. In *2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*. IEEE, SHEGAON, India, 239–243. doi:10.1109/ICITAET47105.2019.9170208

[27] Din Mušić and Carolina Fortuna. 2022. Kubitect - a Solution for On-premise Cluster Deployment. In *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*. IEEE, Vancouver, WA, USA, 273–278. doi:10.1109/UCC56403.2022.00049

[28] Din Mušić, Jernej Hribar, and Carolina Fortuna. 2024. Digital Transformation with a Lightweight On-Premise PaaS. *Future Generation Computer Systems* 160 (Nov. 2024), 619–629. doi:10.1016/j.future.2024.06.026

[29] Michael Packard, Joe Stubbs, Justin Drake, and Christian Garcia. 2021. Real-World, Self-Hosted Kubernetes Experience. In *Practice and Experience in Advanced Research Computing*. ACM, Boston MA USA, 1–5. doi:10.1145/3437359.3465603

[30] Melanie Posey, Dan Thompson, and Pedro Schweizer. 2021. Cloud Repatriation: What It Is, What It Isn't, and Why It's Not Going Away.

[31] Yakov Rekhter, Susan Hares, and Tony Li. 2006. *RFC 4271 - A Border Gateway Protocol 4 (BGP-4).* RFC 4271. RFC Editor. doi:10.17487/RFC4271

[32] Lluis Mas Ruiz, Pere Pinol Pueyo, Jordi Mateo-Fornes, Jordi Vilaplana Mayoral, and Francesc Solsona Tehas. 2022. Autoscaling Pods on an On-Premise Kubernetes Infrastructure QoS-Aware. *IEEE Access* 10 (2022), 33083–33094. doi:10.1109/ACCESS.2022.3158743

[33] Edgar Sarmiento-Calisaya, Alvaro Mamani-Aliaga, and Julio Cesar Sampaio Do Prado Leite. 2024. Introducing Computer Science Undergraduate Students to DevOps Technologies from Software Engineering Fundamentals. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. ACM, Lisbon Portugal, 348–358. doi:10.1145/3639474.3640071

[34] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. 2007. *RFC 4861 - Neighbor Discovery for IP Version 6 (IPv6).* Technical Report 4861. RFC Editor. doi:10.17487/RFC4861

[35] Viktória Spišaková, Dalibor Klusáček, and Lukáš Hejtmánek. 2023. Using Kubernetes in Academic Environment: Problems and Approaches. In *Job Scheduling Strategies for Parallel Processing*, Dalibor Klusáček, Corbalán Julita, and Gonzalo P. Rodrigo (Eds.). Vol. 13592. Springer Nature Switzerland, Cham, 235–253. doi:10.1007/978-3-031-22698-4_12

[36] Sergii Telenyk, Oleksii Sopov, Eduard Zharikov, and Grzegorz Nowakowski. 2021. A Comparison of Kubernetes and Kubernetes-Compatible Platforms. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Vol. 1. 313–317. doi:10.1109/IDAACS53288.2021.9660392

[37] Eddy Truyen, Nane Kratzke, Dimitri Van Landuyt, Bert Lagaisse, and Wouter Joosen. 2020. Managing Feature Compatibility in Kubernetes: Vendor Comparison and Analysis. *IEEE Access* 8 (2020), 228420–228439. doi:10.1109/ACCESS.2020.3045768

[38] Sonajie Wang, Roshan Neupane, Ashish Pandey, Xiyao Cheng, and Prasad Calyam. 2021. Online Learning Platform for Application-Inspired Cloud and DevOps Curriculum. In *2021 IEEE 28th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW)*. IEEE, Bengaluru, India, 35–42. doi:10.1109/HiPCW54834.2021.00012