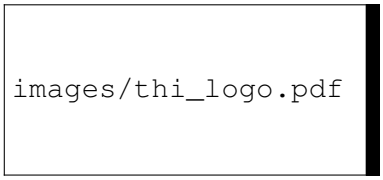


Technische Hochschule Ingolstadt

Specialist area Computer Science
Bachelor's course Computer Science

Bachelor's thesis

Subject:	Conception, Implementation, and Evaluation of a Highly Scalable and Highly Available Kubernetes-Based SaaS Platform on Kubernetes Control Plane (KCP)
Name and Surname:	David Linhardt
Matriculation number:	00122706
Issued on:	2025-04-09
Submitted on:	2025-09-09
First examiner:	Prof. Dr. Bernd Hafenrichter
Second examiner:	Prof. Dr. Ludwig Lausser

Abstract

Contents

Glossary

1 Introduction

1.1 Problem Statement and Motivation

1.2 Objectives and Scope

1.3 Structure of the Thesis

2 Fundamentals

2.1 Kubernetes and Multi-Tenancy

Kubernetes as the Foundation for Cloud-Native Applications As the de facto standard for deploying and managing *cloud-native applications*, Kubernetes plays a pivotal role in modern cloud architecture (Poulton and Joglekar 2021, p. 7–8). Kubernetes works as an application orchestrator for *containerized, cloud-native microservice* apps, meaning it can deploy applications and dynamically respond to changes (Poulton and Joglekar 2021, p. 3). It offers a platform for declarative configuration and automation for containerized workloads, enabling organizations to run distributed applications and services at scale (Kubernetes 2024; Red Hat 2024).

The Importance of Multi-Tenancy in Modern SaaS Platforms Multi-tenancy plays a fundamental role in modern cloud computing. By allowing multiple tenants to share the same infrastructure through virtualization, it significantly increases resource utilization, reduces operational costs, and enables essential features such as VM mobility and dynamic resource allocation (AlJahdali et al. 2014, pp. 345–346). These benefits are critical for cloud providers, as they make the cloud business model economically viable and scalable. In the context of modern SaaS platforms, multi-tenancy goes even further by enabling unified management, frictionless onboarding, and simplified operational processes that allow providers to add new tenants without introducing incremental complexity or cost (AWS 2022, pp. 9–11).

However, while multi-tenancy is indispensable for achieving efficiency, scalability, and cost-effectiveness, it simultaneously introduces complex security challenges, especially in shared environments where resource isolation is limited. In particular, the potential for cross-tenant access and side-channel attacks makes security in multi-tenant environments a primary concern (AlJahdali et al. 2014, pp. 345–346). As such, understanding and addressing multi-tenancy from both operational and security perspectives is essential when designing and securing modern cloud-native platforms (AWS 2022, pp. 9–11; *Information technology - Cloud computing - Part 2: Concepts* 2023, p. 4).

The Challenges of Multi-Tenancy and the Need for Solutions

Kubernetes Control Plane (KCP) as a Promising Approach

Background: The Evolution of Kubernetes Kubernetes, an open-source container orchestration platform developed by Google, emerged from the need to manage the complexities of containerized applications effectively and to support large-scale deployments in a cloud-native environment (Google Cloud 2025; Kubernetes 2024). It was originally developed at Google and released as open source in 2014 (Google Cloud 2025). Kubernetes was conceived as a successor to Google's internal container management system called Borg, and designed to streamline the process of deploying, scaling, and managing applications composed of microservices running in containers (**verma2015**; **bernstein2014**).

Background: Containerization as an Enabler of Kubernetes *Containerization* is a way to bundle an application's code with all its dependencies to run on any infrastructure thus enhancing portability (AWS 2025; Docker 2025). The lightweight nature and isolation can be leveraged by cloud-native software by enabling vertical and horizontal autoscaling facilitated by quick container boot times, along with self-healing mechanisms and support for distributed, resilient infrastructures (Kubernetes 2025b; Kubernetes 2025c; AWS 2025; Davis 2019, pp. 58–59) Furthermore it complements the microservice architectural pattern by enabling isolated, low overhead deployments, ensuring consistent environments (Balalaie, Heydarnoori, and Jamshidi 2016, p. 209).

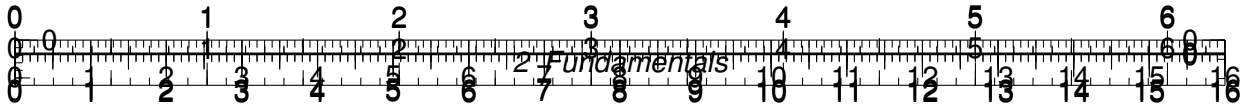
Background: The Role of Microservices in Cloud-Native Architectures *Microservices* play a pivotal role in cloud-native architectures by promoting agility, scalability, and maintainability of applications. By decomposing applications into independent, granular services, microservices

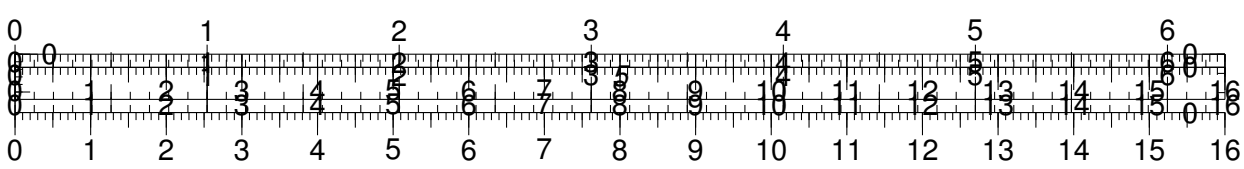
facilitate development, testing, and deployment using diverse technology stacks, enhancing interoperability across platforms (Waseem, Liang, and Shahin 2020, p. 1; Larrucea et al. 2018, p. 1) and help prevent failures in one component from propagating across the system, by isolating functionality into distinct, self-contained services (Davis 2019, p. 62). This architectural style aligns well with cloud environments, as it allows services to evolve independently, effectively addressing challenges associated with scaling and maintenance without being tied to a singular technological framework (Balalaie, Heydarnoori, and Jamshidi 2016, pp. 202–203). Furthermore, the integration of microservices with platforms like Kubernetes enhances deployment automation and orchestration, thus providing substantial elasticity to accommodate fluctuating workloads (Haugeland et al. 2021, p. 170). Additionally, migrating legacy applications to microservices can foster modernization and efficiency, thus positioning organizations favorably in competitive landscapes (Balalaie, Heydarnoori, and Jamshidi 2016, p. 214). Overall, the synergy between microservices and cloud-native architectures stems from their inherent capability to optimize resource utilization and streamline continuous integration and deployment processes.

Background: Kubernetes Resource Isolation Mechanisms Kubernetes employs several resource isolation mechanisms, primarily through the use of *cgroups* (control groups) and *namespaces* to limit resource allocation for containers. Cgroups are a Linux kernel feature that organizes processes into hierarchical groups for fine-grained resource limitation and monitoring via a pseudo-filesystem called *cgroupfs* (Kubernetes 2025a; Project 2024). *Namespaces* are a mechanism for isolating groups of resources withing a single cluster and scoping resource names to prevent naming conflicts across different teams or projects (Kubernetes 2025d). However, these mechanisms may not always provide sufficient isolation necessary for multi-tenant architectures, because the logical segregation offered by namespaces does not address the fundamental security concerns associated with multi-tenancy (Nguyen and Y. Kim 2022, p. 651) and research indicates that the native isolation strategies can lead to performance interference, where containers that share nodes can experience significant degradation in performance due to CPU contention (E. Kim, Lee, and Yoo 2021, p. 158). Specifically, critical services may be adversely affected when non-critical services monopolize available resources, which undermines the quality of service in multi-tenant environments (Li et al. 2019, p. 30410). Moreover, while Kubernetes allows for container orchestration and resource scheduling, it can lead to resource fragmentation, further exacerbating the issue of performance isolation (Jian et al. 2023, p. 1). A common approach in multi-tenant scenarios is to deploy separate clusters for each tenant, which incurs substantial overhead—particularly in environments utilizing virtual machines for isolation (Şenel et al. 2023, pp. 144574–144575). In summary, although Kubernetes offers essential isolation mechanisms, the complexities of resource sharing and

performance consistency in multi-tenant applications highlight the need for enhanced strategies to ensure robust resource management and performance isolation (Nguyen and Y. Kim 2022, p. 651; Jian et al. 2023, p. 2; E. Kim, Lee, and Yoo 2021, p. 158)

Relevance to SaaS and this Thesis





References

AlJahdali, H., A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu (2014). “Multi-tenancy in Cloud Computing”. In: *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pp. 344–351. DOI: 10.1109/SOSE.2014.50.

AWS (2022). *AWS Whitepaper - SaaS Architecture Fundamentals*. AWS. URL: https://docs.aws.amazon.com/whitepapers/latest/saas-architecture-fundamentals/re-defining-multi-tenancy.html (visited on 05/01/2025).

AWS (2025). *What is Containerization?* URL: https://aws.amazon.com/what-is/containerization/ (visited on 05/01/2025).

Balalaie, A., A. Heydarnoori, and P. Jamshidi (2016). "Migrating to cloud-native architectures using microservices: an experience report". In: pp. 201–215. DOI: 10.1007/978-3-319-33313-7_15.

Davis, C. (2019). *Cloud Native Patterns - Designing change-tolerant software*. Shelter Island, NY: Manning. ISBN: 9781617294297.

Docker (2025). *Use containers to Build, Share and Run your applications*. URL: https://www.docker.com/resources/what-container/ (visited on 05/01/2025).

Google Cloud (2025). *What is Kubernetes?* URL: https://cloud.google.com/learn/what-is-kubernetes (visited on 05/01/2025).

Haugeland, S., P. Nguyen, H. Song, and F. Chauvel (2021). "Migrating monoliths to microservices-based customizable multi-tenant cloud-native apps". In: pp. 170–177. DOI: 10.1109/seaa53835.2021.00030.

Information technology - Cloud computing - Part 2: Concepts (2023). Standard.

Jian, Z., X. Xie, Y. Fang, Y. Jiang, T. Li, and Y. Lu (2023). "Drs: a deep reinforcement learning enhanced kubernetes scheduler for microservice-based system". In: DOI: 10.22541/au.167285897.72278925/v1.

Kim, E., K. Lee, and C. Yoo (Jan. 2021). "On the Resource Management of Kubernetes". In: *2021 International Conference on Information Networking (ICOIN)*. IEEE, pp. 154–158. DOI: 10.1109/icoin50884.2021.9333977. URL: http://dx.doi.org/10.1109/icoin50884.2021.9333977.

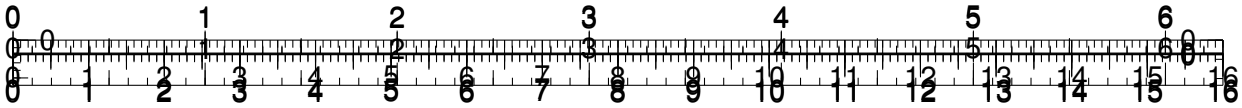
Kubernetes (2024). *Concepts / Overview*. URL: https://kubernetes.io/docs/concepts/overview/ (visited on 05/01/2025).

Kubernetes (2025a). *About cgroup v2*. URL: https://kubernetes.io/docs/concepts/architecture/cgroups/ (visited on 05/02/2025).

Kubernetes (2025b). *Concepts / Workloads / Autoscaling Workloads*. URL: https://kubernetes.io/docs/concepts/workloads/autoscaling/ (visited on 05/01/2025).

Kubernetes (2025c). *Kubernetes Self-Healing*. URL: https://kubernetes.io/docs/concepts/architecture/self-healing/ (visited on 05/01/2025).

Kubernetes (2025d). *Namespaces*. URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ (visited on 05/02/2025).



Larrucea, X., I. Santamaria, R. Colomo-Palacios, and C. Ebert (2018). “Microservices”. In: *IEEE Software* 35.3, pp. 96–100. DOI: 10.1109/MS.2018.2141030.

Li, Y., J. Zhang, C. Jiang, J. Wan, and Z. Ren (2019). “Pine: optimizing performance isolation in container environments”. In: *Ieee Access* 7, pp. 30410–30422. DOI: 10.1109/access.2019.2900451.

Nguyen, N. T. and Y. Kim (Oct. 2022). “A Design of Resource Allocation Structure for Multi-Tenant Services in Kubernetes Cluster”. In: *2022 27th Asia Pacific Conference on Communications (APCC)*. IEEE, pp. 651–654. DOI: 10.1109/apcc55198.2022.9943782.

Poulton, N. and P. Joglekar (2021). *The Kubernetes Book*. 2021 Edition. No ISBN provided. Independently published, p. 243.

Project, T. L. D. (2024). *cgroups(7): Linux control groups*. 6.10. Online; accessed 2025-05-02. Linux man-pages project. URL: https://man7.org/linux/man-pages/man7/cgroups.7.html.

Red Hat (2024). *What is Kubernetes?* URL: https://www.redhat.com/en/topics/containers/what-is-kubernetes (visited on 05/01/2025).

Şenel, B., M. Mouchet, J. Cappos, T. Friedman, O. Fourmaux, and R. McGeer (2023). “Multitenant containers as a service (caas) for clouds and edge clouds”. In: *Ieee Access* 11, pp. 144574–144601. DOI: 10.1109/access.2023.3344486.

Waseem, M., P. Liang, and M. Shahin (2020). “A systematic mapping study on microservices architecture in devops”. In: *Journal of Systems and Software* 170, p. 110798. DOI: 10.1016/j.jss.2020.110798.

List of Figures

Appendix