images/thi_logo.pdf

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
Issued on:	TODO: Insert Issue Date
Submitted on:	TODO: Insert Submit Date
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 [p. 7–8]poulton2021. Kubernetes works as an application orchestrator for *containerized, cloud-native microservice* apps, meaning it can deploy applications and dynamically respond to changes [p. 3]poulton2021. It offers a platform for declarative configuration and automation for containerized workloads, enabling organizations to run distributed applications and services at scale kubernetesOverview,redhatWhatIsKubernetes.

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 [pp. 345–346]IEEEMultiTenancySecurityConcerns. 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 [pp. 9–11]awsSaaSArchitectureFundamentals.
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 [pp. 345–346]IEEEMultiTenancySecurityConcerns. As such, understanding and addressing multi-tenancy from both operational and security perspectives is essential when designing and securing modern cloud-native platforms (pp. 9–11awsSaaSArchitectureFundamentals; p. 4isoConcepts).

The Challenges of Multi-Tenancy and the Need for Solutions

Kubernetes Control Plane (KCP) as a Promising Approach

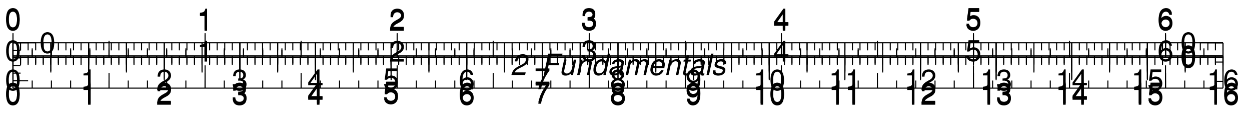
Background: The Evolution of Kubernetes Kubernetes was originally developed at Google and released as open source in 2014 googlecloudWhatIsKubernetes.

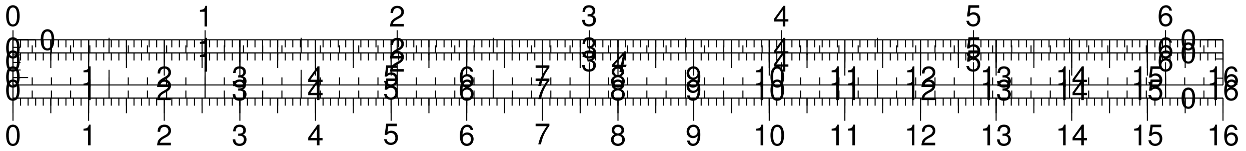
Background: Containerization as an Enabler of Kubernetes *Containerization* is a way to bundle an application's code with all its dependencies to run on any infrastructure thus enhancing portability awsWhatIsContainerization,dockerWhatContainer. The lightweight nature and isolation can be leveraged by 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 (kubernetesAutoscalingWorkloads; kubernetes-SelfHealing; awsWhatIsContainerization; pp. 58–59davis2019) Furthermore it complements the microservice architectural pattern by enabling isolated, low overhead deployments, ensuring consistent environments [p. 209]balalaie2016.

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 (p. 1waseem2020; p. 1larrucea2018) and help prevent failures in one component from propagating across the system, by isolating functionality into distinct, self-contained services [p. 62]davis2019. 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 [pp. 202–203]balalaie2016. Furthermore, the integration of microservices with platforms like Kubernetes enhances deployment automation and orchestration, thus providing substantial elasticity to accommodate fluctuating workloads [p. 170]haugeland2021. Additionally, migrating legacy applications to microservices can foster modernization and efficiency, thus positioning organizations favorably in competitive landscapes [p. 214]balalaie2016. 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* (kubernetesCgroupsV2; cgroups7). *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 kubernetesNamespaces. 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 [p. 651]nguyen2022 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 [p. 158]kim2021. Specifically, critical services may be adversely affected when non-critical services monopolize available resources, which undermines the quality of service in multi-tenant environments [p. 30410]li2019.
Moreover, while Kubernetes allows for container orchestration and resource scheduling, it can lead to resource fragmentation, further exacerbating the issue of performance isolation [p. 1]jian2023. 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 [pp. 144574–144575]senel2023. 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 (p. 651nguyen2022; p. 2jian2023; p. 158kim2021)

Relevance to SaaS and this Thesis





2.2 Kubernetes Control Plane (KCP)

2.3 SaaS Architecture and Automation

3 State of the Art and Related Work

3.1 Zero-Downtime Deployment Strategies

3.2 Kubernetes Scaling Methods

3.3 Multi-Tenancy Concepts in the Cloud

4 Conceptual Design

4.1 System Requirements

4.2 Architecture Design with KCP for SaaS

4.3 Automated Deployment Strategies

5 Prototypical Implementation

5.1 Infrastructure with KCP

5.2 Tenant Provisioning (Automation, Multi-Tenancy)

5.3 Scaling Mechanisms (Horizontal Pod Autoscaler)

5.4 Monitoring and Logging (Prometheus, Grafana)

6 Evaluation

6.1 Performance Measurements (Downtime, Latency, Scaling)

6.2 Scaling Scenarios & Optimizations

6.3 Discussion of Results

6.4 Related Work

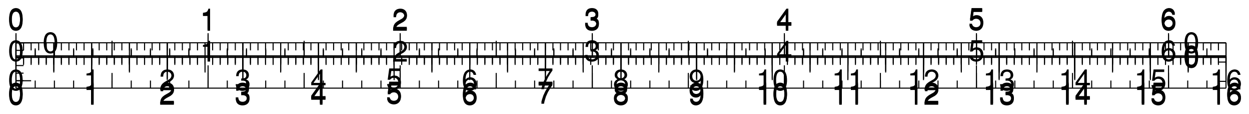
7 Conclusion and Outlook

7.1 Summary

7.2 Personal Conclusion

7.3 Future Outlook

References



List of Figures

Appendix