

KubeGuard: Anomaly Detection and Auto-Remediation for Kubernetes- Deployed Thuto LMS



A final year project submitted by

Tatolo Matlali 201903092

and

Naleli Seboka Mokake 201901918

In partial fulfillment to the requirements for

B.Eng. Computer Systems and Networks
To the

Department of Mathematics and Computer Science
Faculty of Science and Technology
National University of Lesotho.

27 May 2024

Supervisor: **Mr. Lebajoa Mphatsi**

Acknowledgements

Matlali_T.M

I would like to extend my heartfelt gratitude to my family, Koena A. Matlali, Khathatso A. Matlali, and my girlfriend, Mapolo A. Nkuka, for their unwavering support and encouragement throughout my academic journey. Their constant presence, love, and belief in my abilities have been the foundation upon which I have been able to grow and thrive. I owe so much of my success and personal development to their steadfast support - they have been my rock, my motivation, and my support structure, guiding me through the challenges of school life. I am truly grateful for their sacrifices and unconditional care, which have been instrumental in shaping me into the person I am today. Lastly, I would like to express my deepest gratitude to my project partner, Naleli A. Mokake, who has been an unwavering source of strength throughout this project. His support has not only been technical but has also carried me through the direst of circumstances in my life.

Mokake_N.S

I would like to express my deep gratitude to my parents and extended family, who have been with me through this journey, for their continued support and encouragement. Also, my completion of this project could not have been accomplished without the support of Lineo Jockey, Lehlohonolo Moeling, my classmates, and lastly project partner, Tatolo Matlali. Your support and encouragement when the times got rough are much appreciated and duly noted.

Lastly, we would like to express our profound gratitude and sincere appreciation to Mr. Lebajoa Mphatsi, our esteemed supervisor, mentor, and friend. His unwavering support, invaluable guidance, and tireless efforts have been instrumental in the success of this project. Mr. Mphatsi's wealth of knowledge, expertise, and dedication have been a constant source of inspiration and motivation for our team. His insightful feedback, constructive critiques, and thoughtful suggestions have played a pivotal role in shaping the project's trajectory and enabling us to overcome challenges.

Throughout this journey, Mr. Mphatsi has not only provided technical guidance but has also fostered an environment of collaboration, open communication, and continuous learning. His approachable demeanor and genuine concern for our professional growth have made this experience truly rewarding and enriching. We are forever indebted to Mr. Mphatsi for his unwavering commitment and the countless hours he has invested in our success. His mentorship has profoundly impacted our personal and professional development, equipping us with the skills and confidence to tackle future endeavor.

[This page is intentionally left blank]

Abstract

The KubeGuard project aimed to enhance the scalability, reliability, and operability of the Thuto Learning Management System (LMS) at the National University of Lesotho by deploying it on a Kubernetes cluster. The project addressed challenges faced by the existing Thuto LMS deployment, such as limited scalability and lack of advanced monitoring capabilities. The project team adopted an agile methodology, leveraging containerization with Docker and deploying the Thuto LMS on a Kubernetes cluster using Helm charts. Key features were implemented, including anomaly detection for CPU and memory usage, an alerting system to notify administrators of anomalies, auto-remediation mechanisms to automatically scale and recover from issues, and a comprehensive web-based dashboard for real-time monitoring, historical data visualization, and reporting. Testing of the deployed Thuto LMS on Kubernetes confirmed successful containerization of the application and database, seamless deployment using Helm charts, and accessibility via a web browser. The monitoring stack, comprising Prometheus and Grafana, was integrated to collect and visualize metrics, define alerting rules, and set thresholds for anomaly detection. Auto-remediation was achieved through Kubernetes' built-in features and custom scripts, ensuring the system's resilience and availability.

Table of Contents

Table of Contents	iv
Table of Figures	vi
List of Tables	vii
CHAPTER 1	1
Introduction	1
1.1 Context	1
1.2 Problem Statement	1
1.3 Possible Solutions	3
1.4 Proposed Solution	4
1.5 Aims and Objectives	4
1.5.1 Aims	4
1.5.2 Objectives	5
1.6 Scope and Limitations	5
1.6.1 Scope	5
1.6.2 Limitations	6
1.7 Report Outline	6
CHAPTER 2	7
Background and Literature Review	7
2.1 Background	7
2.2 Literature Review	12
CHAPTER 3	15
SYSTEM ANALYSIS AND SYSTEM DESIGN	15
3.1 System Requirements	15
3.2 System Design	21
CHAPTER 4	28

Implementation and Testing.....	28
4.1 Methodology:	28
4.2 System Implementation.....	30
4.3 System Testing.....	51
CHAPTER 5	59
Conclusion, Limitations, and Recommendations.....	59
5.1 Conclusion.....	59
5.2 Limitations	60
5.3 Recommendations (Future Work).....	60
CHAPTER 6	61
References	61
CHAPTER 7	65
Appendix	65
7.1 Tables	65
7.2 Images	66

Table of Figures

Figure 1- Thuto LMS during down-time	3
Figure 2 - Containerization architecture	8
Figure 3 – Container orchestration overview.....	9
Figure 4 – Thuto Current Architecture	11
Figure 5 – Thuto web servers’ architecture	12
Figure 6 - Use Case diagram.....	16
Figure 7 - High Level System Architecture	23
Figure 8 - Kubernetes Cluster system Architecture	24
Figure 9 - Anomaly detection and Alerting	25
Figure 10 - Auto-Remediation for Thuto LMS Anomalies - HPA.....	27
Figure 11 - Agile Methodology	29
Figure 12 - Project development mindmap.....	30
Figure 13 - Helm architecture	34
Figure 14 - Prometheus architecture integrated with Grafana and Alertmanager	36
Figure 15 - Architecture of Loki integrated with Grafana and Nginx	38
Figure 16 - Grafana Architecture.....	40
Figure 17 - Docker running a container from an image.....	41
Figure 18 - Architecture of ingress nginx functioning as a reverse proxy.....	45
Figure 19 - Containerization of MySQL for Sakai	47
Figure 20 - Containerization of Sakai (1/2)	48
Figure 21 - Containerization of Sakai (2/2)	48
Figure 22 - The deployment of Sakai together with the database and PVCs and its Ingress controller	49
Figure 23 - The Helm charts as organized for the release of the deployment	50
Figure 24 - The architecture of the prometheus stack for the purposes of logging and monitoring	51
Figure 25 - The overall system deployment as viewed with kubectl.....	53
Figure 26 - The horizontal pod autoscaler definition for Sakai	53
Figure 27 - The definition of parameters for HPA for Sakai	54
Figure 28 - Accessing Thuto LMS on the web, on the defined host: thuto.k8s.....	55
Figure 29 - The workloads of the deployment incl. Deployments, Pods & Replica Sets.....	55
Figure 30 - The ingress nginx controller showing the domain name (thuto.k8s)	56
Figure 31 - The services of the cluster which are used to communicate within cluster	56
Figure 32 - The defined thresholds for the Thuto LMS metrics (CPU, Memory).....	57
Figure 33 - The visualization of the workloads for the deployment using Grafana (1/2)	57
Figure 34 - The visualization of the workloads for the deployment using Grafana (2/2)	58

List of Tables

Table 1 - Differences between different lightweight distributions of Kubernetes.....	31
Table 2 - System testing results and comments	51

CHAPTER 1

Introduction

The project description is articulated in this chapter along with a concise overview of the project, including its goals, objectives, and expected outcomes. The chapter also brings forward the problem the project is intending to tackle and a brief outline of how chapters have been organized in the report.

1.1 Context

In the rapidly evolving landscape of online education, learning management systems (LMS) have become indispensable tools for facilitating seamless access to educational resources and enabling remote learning experiences. At the National University of Lesotho, Thuto, an LMS powered by the Sakai open-source framework, has been a prominent solution within the institution's educational ecosystem used for sharing course materials, online classes etc. However, as the demand for technology enhanced learning continues to surge, the traditional deployment architecture for Thuto faces significant challenges in terms of scalability, resource management, and high availability, hindering its ability to provide a seamless and reliable learning experience for the NUL students. Therefore, to keep up with the growing demand for Thuto LMS, proper deployment choices must be made and as well as leveraging anomaly detection and auto-remediation mechanisms alongside the chosen deployment approach so as to curb issues of scalability, availability and resource utilization.

1.2 Problem Statement

1.2.1 Problem 1

The current deployment architecture of the Thuto (LMS) faces the issue of inefficient resource load balancing and resources inefficiency. The existing load balancing mechanism used in the current Thuto deployment utilizes the Round-Robin scheduling algorithm Round-robin scheduling

algorithm for dispatching requests to the cluster server nodes. This scheduling algorithm distributes requests in a cyclical fashion, without considering the resource utilization of each server.

1.2.2 Problem 2

The unprecedented outbreak of COVID-19 globally triggered large-scale institutional and behavioral shock effects in different facets of human activity, including education and health of students and families [6]. COVID-19 affected an estimated 1, 5 billion students globally, ranging from primary to tertiary institutions, as they failed to attend school, fearing the severity of the virus [7] and National University of Lesotho and other institutions of higher learning were forced to halt their face-to-face teaching and learning. As a result, the NUL was forced to migrate to online teaching and learning through leveraging Thuto LMS full time. During this period, Thuto LMS faced a surge in traffic since everyone was now forced to use Thuto to partake in learning and the LMS started showing signs of sluggishness, and at times, it also encountered down-times during high demand. These drawbacks negatively affected the students whereby they couldn't access learning resources when the system was down while some failed to hand in their submissions on time due to slow response times and this heavily.

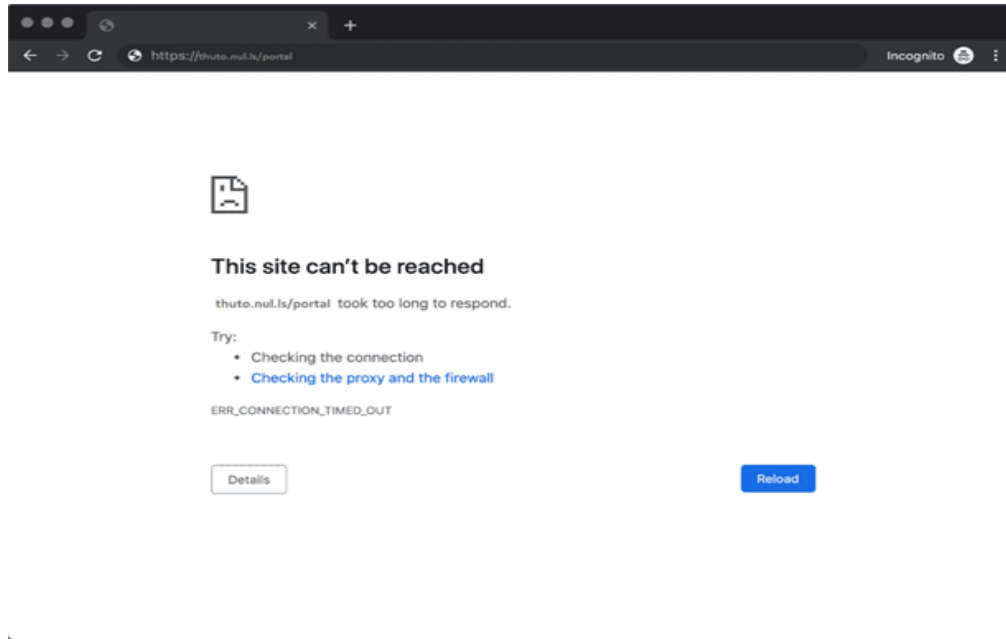


Figure 1- Thuto LMS during down-time

1.3 Possible Solutions

1.3.1 Vertically scaling the existing infrastructure

The existing infrastructure can be provided with extra resources such as CPUs, RAMs, Storage to enhance server performance in order to meet demand.

1.3.2 Employing redundant servers

More web duplicate servers can be deployed to already existing Thuto infrastructure to help achieve high availability.

1.3.3 Leveraging virtual machines

VMs can be leveraged to help host the application server and the database server on a single physical server and thus utilizing resources.

1.3.4 Cloud-based deployment

The Thuto LMS can be migrated from physical infrastructure to the cloud infrastructure whereby it can benefit from auto-scaling capabilities and load balancing features to dynamically adjust resources based on demand.

1.3.5 Containerization and Orchestration

Adopting containerization technologies like Docker and leveraging container orchestration tools like Docker Swarm and Kubernetes to achieve seamless scalability, high availability, and efficient resource utilization.

1.4 Proposed Solution

1.4.1 Anomaly detection and auto-remediation for Kubernetes deployed Thuto LMS

The proposed solution for Thuto LMS involves containerizing Thuto web application and MySQL and then deploying the two on a Kubernetes cluster, thus leveraging powerful features and capabilities of the orchestration platform. The goal is do away with manual detection and remediation of anomalies and to make efficient use of the existing Thuto LMS infrastructure while we also aim at obtaining a highly scalable and available system. These deployment approaches will make use of containerization technology and container orchestration tools along with anomaly detection and auto-remediation mechanisms to achieve the desired system, the system will both benefit the end-users of the LMS and the NUL as well.

1.5 Aims and Objectives

1.5.1 Aims

The goal of this project is to resolve several issues mentioned earlier associated with Thuto LMS such as resource underutilization and scalability issues. This project aims at containerizing the

Thuto web app along with its database and deploying the two on a K8s cluster where anomaly detection and auto-remediation features can be leveraged to an ideal system.

1.5.2 Objectives

To achieve the above aims, the following objectives shall be undertaken:

- Review the current deployment architecture of the Thuto LMS.
- Acquire a deep understanding of K8s, Sakai LMS, and the specific requirements and constraints of the system.
- Containerizing Thuto LMS web app and MySQL database.
- Designing a highly available and scalable K8s.
- Leveraging detection and auto-remediation mechanisms within the K8s
- Deploying our Thuto LMS on the proposed K8s cluster.
- Assess the deployment of the Thuto LMS on K8s cluster.
- Integrating a dashboard to the K8s cluster.
- Integrating an alerting system to the K8s cluster.

1.6 Scope and Limitations

1.6.1 Scope

- Containerize Thuto LMS web app using Docker.
- Design a highly available and scalable k8s cluster for Thuto LMS with anomaly detection, auto-remediation, and alerting capabilities.
- Deploy and assess the Thuto LMS on the designed k8s cluster.
- Design and integrate a web-based dashboard with the k8s cluster.

1.6.2 Limitations

- The sakai source code will be used in place of the Thuto source code.
- Money to upgrade our personal computers RAMs.

1.7 Report Outline

- **Chapter 1 – Introduction:** This section introduces anomaly detection and auto-remediation for K8s deployed Thuto LMS concepts and systems used to achieve the development of this project. It articulates the project's focus, the scope and limitations of the project.
- **Chapter 2 – Background and Literature Review:** This chapter provides a comprehensive review of the existing literature, research, and technologies related to containerization, Kubernetes, anomaly detection, and auto remediation in the context of LMS.
- **Chapter 3 – System Analysis and Design:** This chapter details the proposed system design and architecture for Thuto LMS deployment on Kubernetes including the components, interactions, and workflows involved in the anomaly detection and auto-remediation processes.
- **Chapter 4 – Implementation and Testing:** This chapter covers implementation and testing aspects of the project including testing methodologies and evaluation criteria used to validate the system scalability, resource utilization and reliability.
- **Chapter 5 – Conclusion, Limitations, and Recommendations:** This chapter illustrates the conclusion, limitations, and future recommendations regarding the project in discussion.
- **Chapter 6 – References:** This section provides a list of references used to build up this report.
- **Chapter 7 – Appendices:** Diagrams, Figures, Screenshots, Code, and Test results of the systems are provided here.

CHAPTER 2

Background and Literature Review

2.1 Background

2.1.1 Containerization

Containerization refers to the practice of packaging software code with its dependencies, such as libraries, runtime environment, and configurations into a single executable container thus allowing applications to run consistently across different computing environments. Containers enable developers to create new cloud-native applications as containerized microservices or repackage existing applications into containers. The “write once, run anywhere” capability of containerization allows apps to be deployed across on-premises infrastructure, hybrid clouds, and multi-cloud environments.

Containerization has become the de facto standard for modern cloud-native applications, and it is widely used by organizations to roll out new applications and modernize existing ones for the cloud [15]. Containers offer a variety of benefits such as portability which allows easy deployment across different environments, consistency that ensures predictable behavior, scalability that makes it easy to scale up and down, efficiency with a smaller footprint than virtual machines, rapid deployment capabilities etc.

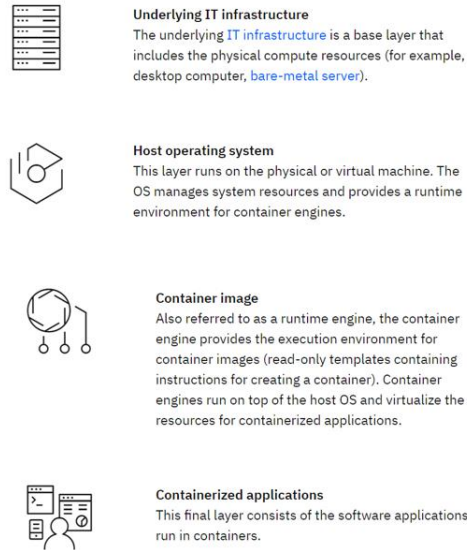


Figure 2 - Containerization architecture

2.1.2 Container Orchestration

Container orchestration stepped into the DevOps field due to the rising use of container technology to address issues like scaling, networking, and container lifecycle management. Orchestration tools automate tasks like provisioning, deployment, and load balancing, simplifying container management at scale. They tackle issues such as resources allocation, service continuity, and security, ensuring efficient operation in complex environments. The rise of microservices led to the increased usage of container technologies because the containers offer the perfect host for small independent applications like microservices. This microservice trend has resulted in applications that now comprise thousands of containers and managing those loads of containers across multiple environments using ad hoc scripts and self-made tools can be complex and some impossible hence the need for having container orchestration tools.

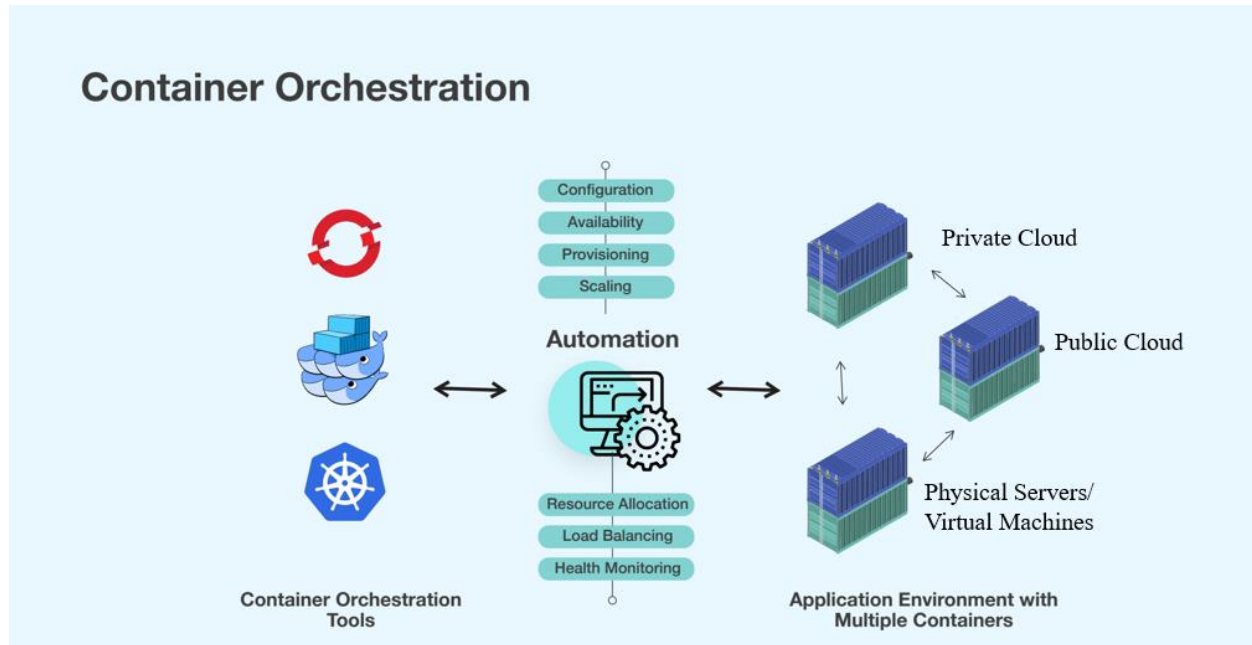


Figure 3 – Container orchestration overview

2.1.3 The choice of containerization tool

Why Kubernetes?

Compared to alternative orchestration tools like Docker Swarm and Apache Mesos, Kubernetes offers a more comprehensive and feature-rich solution. Its advanced capabilities, such as automatic load balancing, self-healing, and horizontal scaling, make it well-suited for deploying and managing complex, mission-critical applications.

Additionally, Kubernetes provides a robust set of resources and abstractions, including Deployments, Services, and Ingress, which simplify the management of containerized applications. Its declarative approach to configuration management, using manifests or Helm charts, promotes consistency and reproducibility across different environments.

2.1.4 Anomaly Detection and Auto-Remediation Mechanism

An anomaly detection is a system designed to identify patterns in data that deviate significantly from the expected behavior. This mechanism typically employs various statistical techniques to monitor and analyze data streams in real-time, flagging unusual patterns that may indicate issues such as performance bottlenecks, security breaches, or system failures. Auto-remediation systems on the other hand react instantly to detected anomalies, executing predefined actions without human intervention, ensuring rapid response, minimizing downtime, and preventing minor issues from escalating into major problems. They apply remediation steps uniformly, reducing the risk of human error, and can handle a large number of incidents simultaneously, making them highly scalable.

2.1.5 Learning Management System

A learning management system (LMS) is a software application or web-based technology used to plan, implement, and assess a specific learning process. Typically, a learning management system provides an instructor with a way to create and deliver content, monitor student participation, and assess student performance online. The LMS may also provide students with the ability to use interactive features such as threaded discussions, video conferencing, and discussion forums. In this ecosystem, suitable real time discussions between students and instructors can be held in the form of chats and forums to better exchange ideas and knowledge, and consequently build a more enjoyable and productive educational experience. A typical LMS, such as Sakai, usually provides several tools (e.g., Resources, Forums, Online Tests, Chat rooms, etc.) for e-learning [2].

2.1.6 Overview of Thuto's Current Architecture

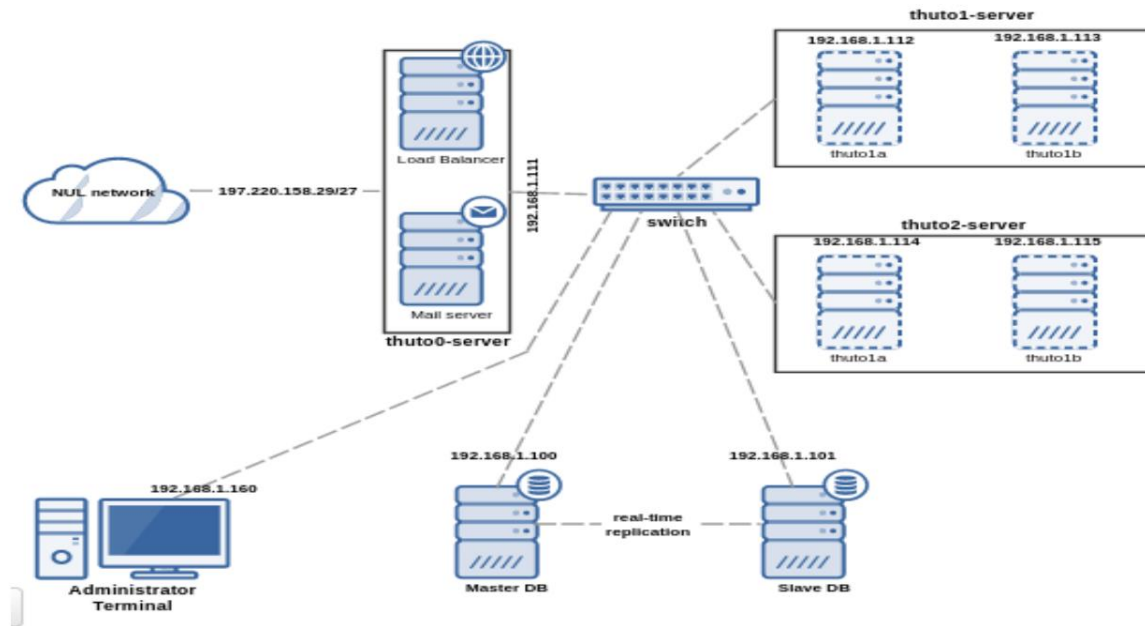


Figure 4 – Thuto Current Architecture

The Thuto system is currently composed- of five physical servers: **thuto0-server**, **thuto1-server**, **thuto2-server**, **thuto3-server**, **thuto4-server**, and **thuto5-server**.

The **thuto0-server** hosts the Apache mod_proxy load balancer responsible for dispatching traffic to <https://thuto.nul.ls/> across the cluster nodes. Additionally, this server runs the Postfix mail server for sending email notifications to Thuto users.

thuto1-server and **thuto2-server** are the web application servers hosting the Thuto web application. Each of these servers runs two virtual machines (**thuto1a**, **thuto1b**, **thuto2a**, and **thuto2b**), resulting in a total of four cluster nodes to increase concurrent session handling capacity. Each virtual machine is allocated 6GB RAM and 4 virtual CPUs. On each virtual machine, Tomcat 7 web application servlet container runs an instance of the Thuto web application. The Tomcat servlet containers are configured to automatically start after a power outage to ensure minimal disruption in case of power loss.

Finally, `thuto4-server` and `thuto5-server` are the database servers. MySQL is running on both servers, persisting data for Thuto. Master-Slave Replication has been implemented to achieve high availability for the MySQL database instances [2].

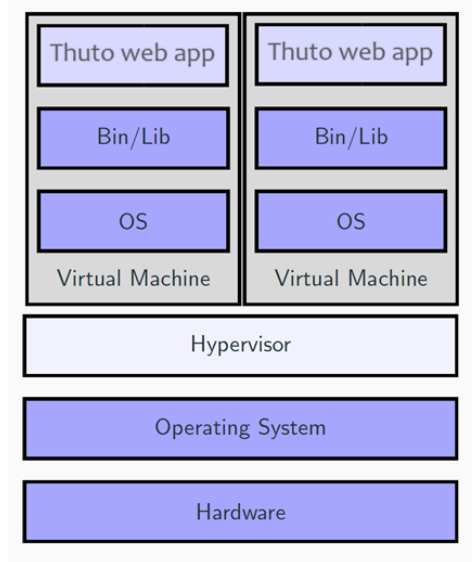


Figure 5 – Thuto web servers' architecture

2.2 Literature Review

The advent of containerization and container orchestration technologies like Docker and Kubernetes has revolutionized the way applications are deployed and managed, especially in cloud-native environments. Kubernetes, in particular, has emerged as the de facto standard for container orchestration, enabling organizations to achieve high scalability, availability, and efficient resource utilization for their applications.

Anomaly detection and auto-remediation are critical aspects of maintaining a reliable and robust Kubernetes deployment. Researchers have explored various techniques and approaches to address these challenges. Kosińska J. et al. (2022) proposed an anomaly detection framework for Kubernetes clusters based on a combination of machine learning algorithms and rule-based methods. Their approach leverages metrics collected from Kubernetes components and applications to identify anomalous behavior patterns.

Furthermore, Shen et al. (2020) developed an auto-remediation system for Kubernetes deployments that can automatically detect and mitigate various types of faults, such as node failures, resource exhaustion, and application crashes. Their solution utilizes a combination of monitoring tools, custom controllers, and automated recovery procedures to ensure the resilience and self-healing capabilities of the Kubernetes cluster.

In the context of learning management systems (LMSs), the adoption of cloud-native technologies has gained traction due to the increasing demand for online education and the need for scalable and resilient platforms. Alhadrami et al. (2021) explored the deployment of Moodle, a popular LMS, on Kubernetes, highlighting the benefits of containerization and orchestration for improved performance, scalability, and availability.

Monitoring and observability are essential components of any Kubernetes deployment, enabling administrators to gain insights into the health and performance of the cluster and applications. Prometheus, an open-source monitoring, and alerting solution has become a widely adopted tool in the Kubernetes ecosystem. Researchers have investigated the integration of Prometheus with Kubernetes and its effectiveness in monitoring and alerting for various use cases (Rabah et al., 2021; Wang et al., 2020).

Similarly, Grafana, a popular data visualization and monitoring platform, has been extensively used in conjunction with Prometheus to provide comprehensive dashboards and visualizations for Kubernetes deployments. Studies by Xu et al. (2021) and Lee et al. (2022) have explored the integration of Grafana with Kubernetes and its utility in monitoring and troubleshooting applications deployed on the platform.

While the existing literature provides valuable insights into various aspects of Kubernetes deployments, anomaly detection, auto-remediation, and monitoring, there is a need for comprehensive solutions tailored specifically to learning management systems like Thuto LMS. The KubeGuard project aims to address this gap by developing a robust and efficient deployment of Thuto LMS on Kubernetes, leveraging advanced anomaly detection, auto-remediation, and monitoring capabilities to ensure a seamless and reliable online learning experience.

CHAPTER 3

SYSTEM ANALYSIS AND SYSTEM DESIGN

This section of the report delves into the comprehensive analysis and design of the proposed enhancements for the Thuto Learning Management System (LMS). The primary focus is on improving Thuto LMS by deploying it on Kubernetes, which will provide a scalable and resilient infrastructure. Key features to be integrated include anomaly detection, an alerting system, auto-remediation for workloads, and a dashboard and reporting tool. Each component is meticulously designed to address specific operational needs and ensure a seamless user experience. The following analysis outlines the requirements, architecture, and design considerations for effectively implementing these features on Kubernetes.

3.1 System Requirements

In this section we describe the system requirements that were gathered during the requirements elicitation. This includes both the functional and non-functional system requirements. The use case diagram below serves as a summary of the functional requirements for the system.

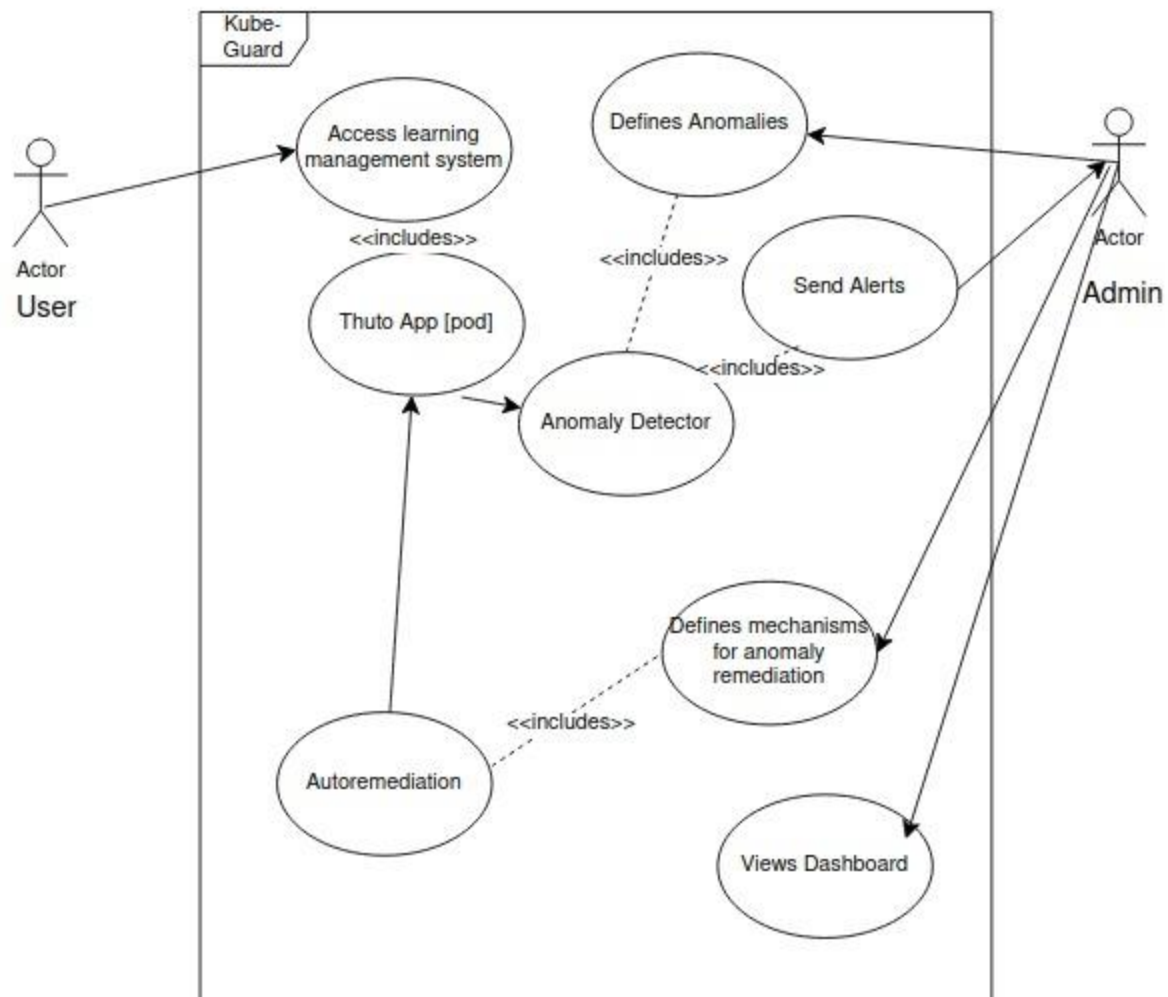


Figure 6 - Use Case diagram

3.1.1 Functional Requirements

3.1.1.1 Learning Management System (Thuto LMS)

The successful deployment of Thuto LMS on Kubernetes should enable users (students and teachers) of Thuto LMS to access the application seamlessly on the web. They should be able to carry out their daily activities as required by their needs, on the application.

3.1.1.2 Definition of Anomaly

Upon completion of this solution, the administrator of the Kubernetes-deployed Thuto LMS should be able to define anomalies within the specific context of the LMS's operation. For the purpose of this project, anomalies are defined as unusual CPU and Memory (RAM) utilization based on the system's demands at any given time. Another anomaly could be the failure rate of a pod exceeding the expected rate for a normal Docker container.

3.1.1.3 Send Alerts To Admin

The solution should also include a mechanism for alerting the admin whenever a defined anomaly is triggered. Alerts should be sent via Slack, PagerDuty, or email. Since the solution brought by KubeGuard seeks to minimize human intervention, it is important thus, for the admin to be notified automatically when an anomaly has been triggered, lest he be unaware of critical issues that require immediate attention and potentially compromise the system's performance and reliability.

3.1.1.4 Dashboards (Monitoring)

A crucial component of the enhanced Thuto LMS is the development of a user-friendly, web-based dashboard for monitoring purposes. This dashboard should provide real-time and historical insights into the health and performance of the Thuto LMS, enabling administrators to visualize anomalies and access detailed reports on key performance indicators (KPIs) specific to the LMS.

The dashboard will feature:

3.1.1.4.1 Real-Time Monitoring

Display live metrics such as CPU and Memory usage, user activity, course enrollments, and content access. This allows administrators to detect and respond to issues promptly.

3.1.1.4.2 Historical Data Visualization

Provide historical data on system performance and resource utilization, helping administrators to identify trends and patterns over time.

3.1.1.4.3 Anomaly Visualization

Highlight detected anomalies, providing detailed information about the nature and potential impact of these anomalies.

3.1.1.4.4 Customizable Views

Allow administrators to customize the dashboard view to focus on specific metrics or time periods, enhancing their ability to monitor the system effectively.

3.1.1.4.5 Reporting Tools

Generate detailed reports that summarize system performance, anomalies detected, and actions taken. These reports can be used for regular reviews and strategic planning.

3.1.1.5 Anomaly Detection

For effective anomaly detection, the integration of Prometheus and Grafana with Kubernetes will be essential. Prometheus will collect and store metrics from the Thuto LMS environment, such as CPU and memory utilization, user activity, and system health indicators. Grafana will then visualize this data, providing a comprehensive and accessible interface for monitoring real-time and historical metrics. By setting up alerting rules within Prometheus and kubernetes, administrators can define thresholds for anomalies. When these thresholds are breached, alerts are triggered and sent via the specified channels (e.g., email, Slack). Added to that Horizontal Pod Autoscaler (HPA), a feature in kubernetes will be used to set thresholds for CPU and memory utilization so that whenever a pod reaches the set threshold.

3.1.1.6 Auto-Remediation

The auto-remediation process will leverage Kubernetes' inherent self-healing capabilities along with custom remediation actions. When Prometheus detects an anomaly, such as a pod experiencing high resource utilization or failing, Kubernetes can automatically restart the affected pods to maintain the application's stability. Additionally, the system can trigger specific remediation scripts or actions defined in KubeGuard to address more complex issues. For example, if high CPU utilization is detected, the auto-remediation process might involve scaling the application horizontally by adding more pod replicas or adjusting resource limits. HPA will also be used for auto-remediation, should another be automatically scaled to operate and share the load between the pods.

By integrating Prometheus, Grafana, and Kubernetes' auto-remediation features, the Thuto LMS will benefit from a robust monitoring and maintenance system that minimizes downtime and ensures continuous, reliable operation. This advanced setup will provide a proactive approach to system management, allowing administrators to focus on strategic improvements rather than constant troubleshooting.

3.1.2 Non-Functional Requirements

3.1.2.1 Reliability /Accessibility

The kubernetes deployed Thuto LMS should always be accessible at all times except during scheduled maintenance periods.

3.1.2.2 Scalability

The development of this system includes use of helm charts, and microk8s deployment. Helm charts and templating are scalable. Whenever new features are to be added to the system, new charts can be easily added to the already existing release of Thuto LMS. For instance, Thuto LMS currently uses a monolithic architecture, whenever a microservices architecture is adopted, the charts for all the containers may be easily added to the already existing system. Microk8s is a

kubernetes distribution which supports a single node cluster as well as a multi-node cluster. This says Thuto LMS can always be expanded to use more than one node according to the needs of its availability. Lastly, kubernetes clusters can be easily deployed on cloud infrastructure so this is an advantage for the purposes of heavy or resource intensive periods like COVID-19 pandemic when all learning and teaching was facilitated online.

3.1.2.3 Security

The chosen Kubernetes setup is designed to ensure robust security despite its singular node configuration. Leveraging Kubernetes' inherent features, including strong container isolation facilitated by namespaces, cgroups, and seccomp profiles, alongside the enforcement of strict pod security policies, our deployment mitigates potential risks. Moreover, resource quotas, network policies, and Role-Based Access Control (RBAC) mechanisms are employed to regulate resource usage and restrict unauthorized access. To safeguard communication within the cluster and with external entities, Transport Layer Security (TLS) encryption is implemented, while sensitive information is securely managed using Kubernetes secrets. Kubernetes secrets are encoded using *base 63 encoding (b64enc)*. Continuous monitoring via tools like Prometheus and comprehensive audit logging further enhance our security posture, enabling real-time threat detection and analysis. Additionally, proactive measures such as regular updates for Kubernetes and container images, along with a well-defined incident response plan and robust backup procedures, bolster our resilience against potential security breaches.

3.1.3 Software Requirements

3.1.3.1 Docker

Containerization of the application components, Sakai and SakaiDB (MySQL) using Docker for consistent deployment and scalability. This will also be used as runtime for the container pods.

3.1.3.2 Kubernetes Distribution

Kubernetes will be needed for container orchestration to manage container lifecycle, scaling, and resilience.

3.1.3.3 Load Balancer

NGINX will be needed to act as the reverse proxy to handle incoming traffic efficiently. It is the most required because it can serve static content, cache requests and efficiently proxy requests to the application.

3.1.3.4 Helm

Helm will be needed to provide seamless deployment which is scalable and also templating the deployment into charts that can be easily upgraded and released into newer versions.

3.1.4 Constraints

3.1.4.1 Availability

As the administrator is going to need to access real time performance metrics of the LMS, the dashboards designed for monitoring should be highly available. The alertmanager should also be highly available as it will need to provide alerts on time as soon as the system needs intervention.

3.1.4.2 Highly available database

Thuto LMS uses a centralized database which handles a lot of requests in a small unit of time during high peak days. The database will need to be able to handle such requests.

3.2 System Design

KubeGuard represents a pivotal enhancement to the security, reliability, and scalability of the Thuto Learning Management System (LMS) deployed on Kubernetes infrastructure. This section

delineates the architectural blueprint and constituent elements of KubeGuard, aimed at fortifying Thuto LMS against anomalies and automating remediation processes.

3.2.1 System Architecture

KubeGuard's architecture capitalizes on the native capabilities of Kubernetes for container orchestration and management. Leveraging Prometheus for metric collection and Grafana for visualization, KubeGuard extends the Kubernetes ecosystem to encompass advanced anomaly detection algorithms tailored to Thuto LMS metrics. This architecture ensures seamless integration and real-time monitoring of Thuto LMS within the Kubernetes environment.

3.2.3 Anomaly Detection for Thuto LMS

By harnessing Prometheus's robust monitoring capabilities, KubeGuard implements tailored anomaly detection algorithms specific to Thuto LMS metrics. Prometheus exporters collect Thuto LMS-related data, enabling KubeGuard to establish baselines and detect deviations indicative of anomalies or security threats within the Kubernetes-deployed environment.

3.2.3 Alerting System

KubeGuard orchestrates a comprehensive alerting system, leveraging Prometheus Alertmanager in conjunction with Grafana for configurable alert notifications. Administrators can set alert thresholds and notification channels within Grafana, ensuring timely detection and response to anomalies detected within the Thuto LMS Kubernetes deployment.

3.2.4 Auto-Remediation for Thuto LMS Anomalies

Building upon Kubernetes' native capabilities for auto-scaling and resource management, KubeGuard's Auto-Remediation module empowers administrators to automate corrective actions specific to Thuto LMS workloads. Leveraging Kubernetes features like ***Horizontal Pod Autoscaling (HPA)***, KubeGuard dynamically adjusts resource allocations, scales server instances, or reallocates workloads in response to detected anomalies, ensuring optimal performance and stability of Thuto LMS within the Kubernetes cluster.

3.2.5 Dashboard and Reporting

The integration of Grafana within KubeGuard provides administrators with a centralized Dashboard and Reporting interface, offering real-time insights and historical analyses of Thuto LMS metrics within the Kubernetes environment. Through customizable dashboards and pre-configured visualizations, stakeholders gain actionable insights into system performance, anomaly trends, and operational KPIs, facilitating informed decision-making and proactive management of Thuto LMS deployments on Kubernetes.

3.2.1 System Architecture

3.2.1.1 The high-level System Architecture

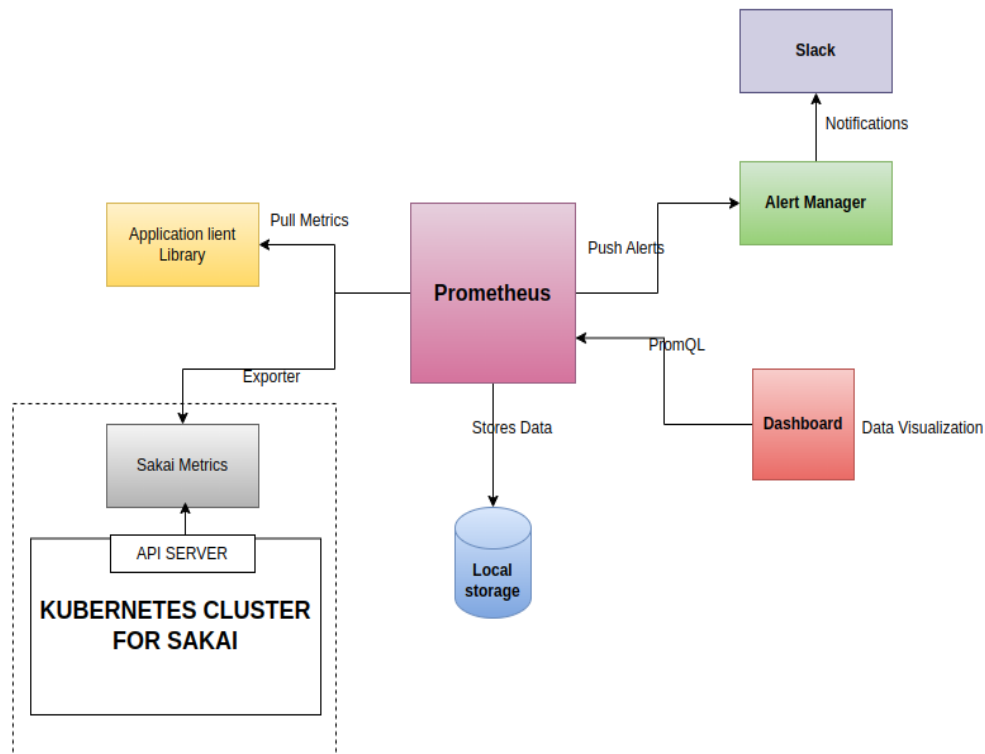


Figure 7 - High Level System Architecture

3.2.1.2 The Kubernetes Cluster System Architecture

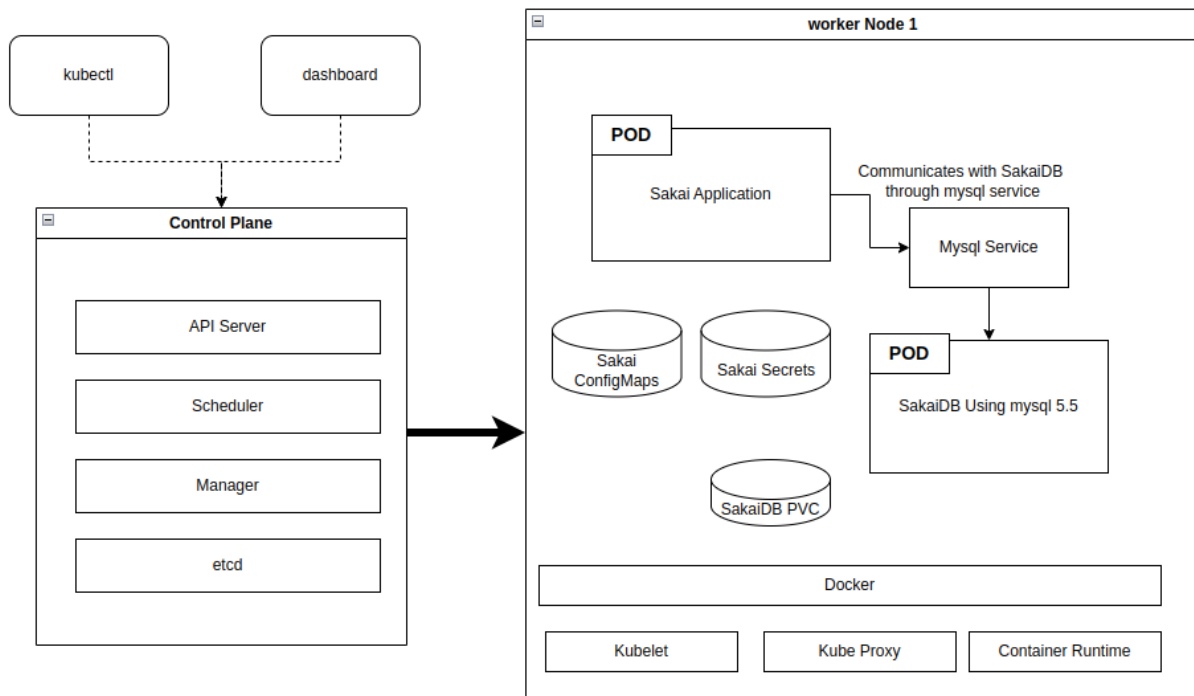


Figure 8 - Kubernetes Cluster system Architecture

3.2.2 Anomaly Detection and Alerting

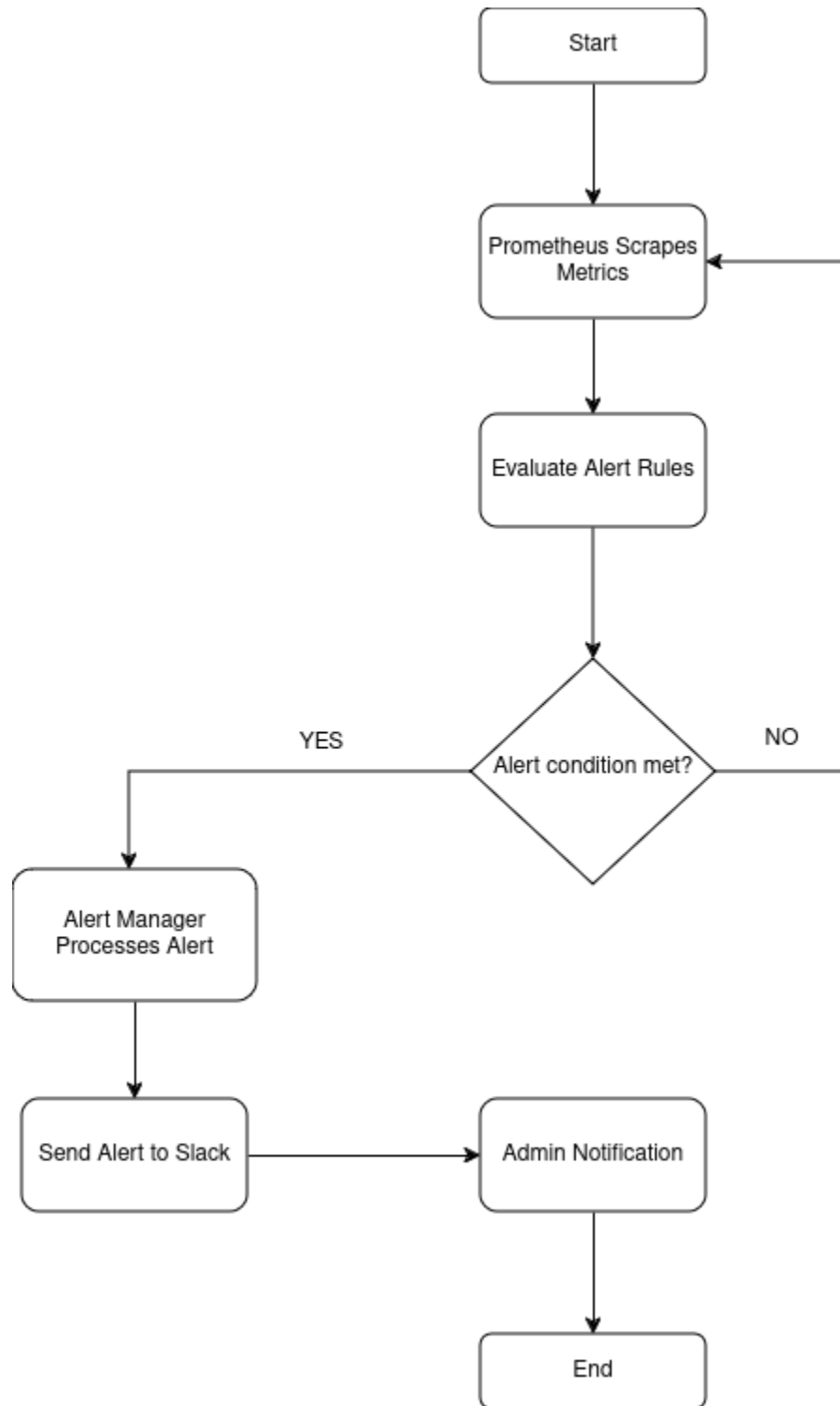


Figure 9 - Anomaly detection and Alerting

3.2.3 Auto-Remediation for Thuto LMS Anomalies - HPA

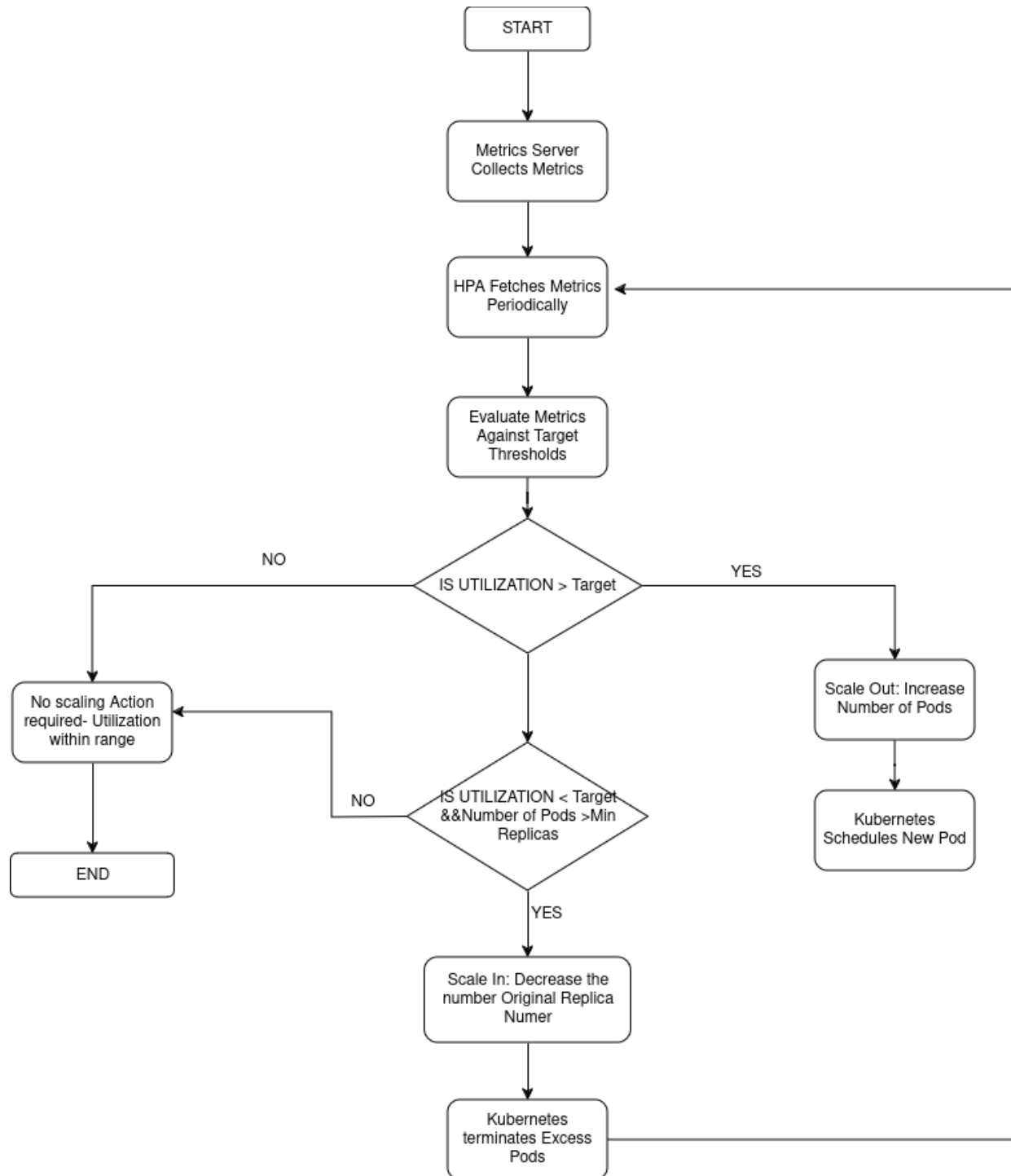


Figure 10 - Auto-Remediation for Thuto LMS Anomalies - HPA

CHAPTER 4

Implementation and Testing

After meticulously planning and designing the system architecture in the previous chapters, this chapter focuses on the practical realization of the proposed solution. It outlines the implementation process, encompassing the development environment, tools, and technologies employed. Additionally, it discusses the comprehensive testing strategy adopted to ensure the system's functionality, reliability, and conformance to specified requirements. The testing phase encompasses various levels, including unit testing, integration testing, and system testing, utilizing industry-standard techniques and methodologies. The chapter also highlights any challenges encountered during implementation and the corresponding mitigation strategies employed. By providing a detailed account of the implementation and testing processes, this chapter aims to demonstrate the project's adherence to best practices and the rigorous measures undertaken to deliver a high-quality, robust solution.

4.1 Methodology:

4.1.2 Agile

Throughout the development process of KubGuard, we adopted an agile methodology. The Agile methodology is an iterative and incremental approach to software development that emphasizes flexibility, collaboration, and continuous improvement. In the context of this project, which involves deploying the Thuto Learning Management System (LMS) on Kubernetes and implementing monitoring, auto-remediation, and alerting mechanisms, the Agile methodology is well-suited for several reasons.

Firstly, the Agile approach divides the project into smaller, manageable iterations or sprints, allowing for incremental development and delivery of features. This aligns well with the project's objectives, as it enables the team to focus on deploying the LMS on Kubernetes first, followed by implementing monitoring, auto-remediation, and alerting capabilities in subsequent iterations.

Secondly, Agile promotes close collaboration between cross-functional teams, including developers, operations, and stakeholders. This collaboration is crucial for a project involving the deployment and monitoring of a complex system like the Thuto LMS on Kubernetes, as it requires expertise from various domains, such as infrastructure management and monitoring.

Furthermore, Agile emphasizes continuous feedback and improvement through regular retrospectives and refinement of processes. This aspect is particularly valuable for this project, as it allows the team to continuously refine and optimize the monitoring, auto-remediation, and alerting mechanisms based on real-world usage and feedback from stakeholders.

Additionally, the Agile methodology embraces change and allows for adjustments in requirements or priorities throughout the project's lifecycle. This flexibility is beneficial for a project involving Kubernetes and monitoring, as the technology landscape and best practices in these areas are constantly evolving, and the team may need to adapt to new developments or changes in requirements.

Overall, the Agile methodology's iterative approach, emphasis on collaboration, continuous improvement, and adaptability to change make it a suitable choice for the development process of KubGuard, a project that involves deploying a complex system on Kubernetes and implementing monitoring, auto-remediation, and alerting mechanisms.



Figure 11 - Agile Methodology

4.1.2 Project Development Mindmap

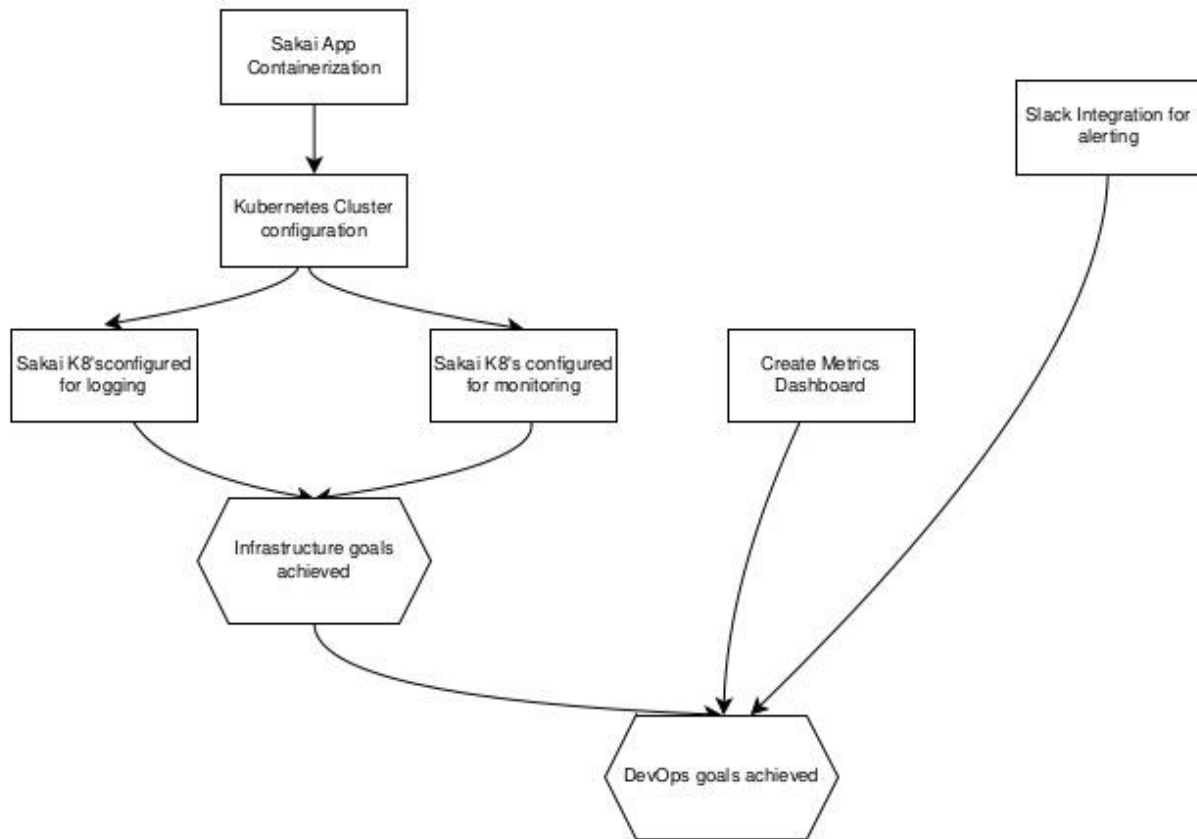


Figure 12 - Project development mindmap

4.2 System Implementation

4.2.1. System Implementation technologies

The purpose of this subsection is to provide an overview of the technological stack and tools used in the implementation of the system, allowing readers to understand the technical choices made and their potential implications. It serves as a reference for the technologies involved and may also provide justifications for the selected approaches.

4.2.1.1 Microk8s

Table 1 - Differences between different lightweight distributions of Kubernetes

Feature	Microk8s	K3s	Minikube
CNCF	yes	yes	yes
Vanilla Kubernetes	yes	no	yes
Architecture support	x86, ARM64, s390x, POWER9	x86, ARM64, ARMhf	x86, ARM64, ARMv7, ppc64, s390x
Enterprise support	yes	yes	no
Single-node support	yes	yes	yes
Multi-node support	yes	yes	no
Automatic high availability	yes	yes	no
Automatic updates	yes	yes	no
Memory requirements	540MB	512MB	644MB
Add-on functionality	yes	no	yes
Container runtime	containerd, kata	CRI-O, containerd	Docker, containerd, CRI-O
Networking	Calico, Cilium, CoreDNS, Traefik, NGINX, Ambassador,	Flannel, CoreDNS, Traefik, Canal, Klipper	Calico, Cilium, Flannel, ingress, DNS,

	Multus, MetalLB		Kindnet
Default storage options	Hostpath storage, OpenEBS, Ceph	Hostpath storage, Longhorn	Hostpath storage
GPU acceleration	yes	yes	yes

4.2.1.1.1 Why Microk8s?

We chose MicroK8s for our deployment because it simplifies Kubernetes consumption by abstracting away much of the complexity involved in managing cluster lifecycles. Its user-friendly interface automates or simplifies key operations such as deployment, clustering, and enabling auxiliary services essential for a production-grade K8s environment. Unlike other lightweight distributions, MicroK8s offers a single-command installation, automatic high availability clustering, and automatic updates.

Robustness is crucial, especially in production environments, and MicroK8s excels here. It provides transactional over-the-air patching and security fixes through the snap package, as well as self-healing high availability for Kubernetes control plane services. This makes it ideal for supporting mission-critical workloads.

Additionally, MicroK8s offers a complete, CNCF-compliant Kubernetes environment with all the services needed to run OCI containers at scale. This includes networking and load balancing, storage, service mesh, observability, GPU and FPGA acceleration, multi-cluster management, and more. This comprehensive feature set is unique among lightweight Kubernetes distributions, making MicroK8s the best choice for our deployment.

4.2.1.2 Helm v2, 3

Helm is a package manager for Kubernetes that simplifies the deployment and management of applications on Kubernetes clusters. It provides a way to define, install, and upgrade complex

Kubernetes applications using a packaging format called "charts." The importance of Helm in Kubernetes deployment lies in its ability to streamline and standardize the deployment process, making it easier to manage and maintain applications throughout their lifecycle.

Helm offers several key benefits and importance in Kubernetes deployment. Firstly, it promotes packaging and reusability by allowing all the Kubernetes resources required for an application, such as Deployments, Services, ConfigMaps, and Secrets, to be packaged together in a chart. This packaging approach makes it easier to share and distribute applications across different environments or teams. Additionally, Helm supports versioning for charts, enabling you to track changes and easily roll back to a previous version if needed, which is crucial for maintaining application stability and enabling safe upgrades or rollbacks.

Moreover, Helm charts can include configuration values that can be customized during installation or upgrade, facilitating easy management of application configurations across different environments, such as development, staging, and production. Helm also provides a mechanism for managing dependencies between charts, enabling the installation and management of complex applications that rely on multiple components or sub-charts.

Furthermore, Helm allows you to preview the changes that will be made during an installation or upgrade before applying them, ensuring that the deployment process is predictable and reducing the risk of unintended changes. It also has a large and active community, with a wide range of publicly available charts for popular applications and services, making it easier to find and deploy pre-existing solutions, saving time and effort.

In the context of Kubernetes deployment, Helm plays a crucial role in streamlining the process of packaging, distributing, and managing applications. It simplifies the deployment of complex applications by providing a consistent and repeatable way to install and upgrade them across different environments. Additionally, Helm's versioning, rollback, and configuration management capabilities help ensure application stability and facilitate easier maintenance and troubleshooting.

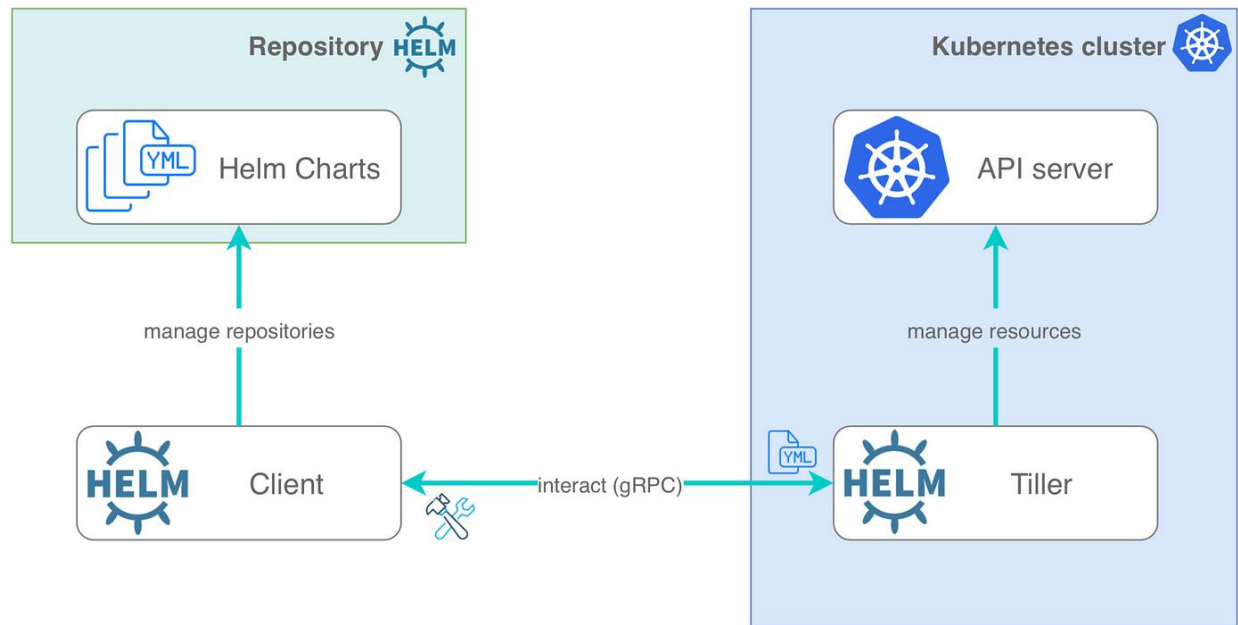


Figure 13 - Helm architecture

4.2.1.3 Prometheus

Prometheus is an open-source monitoring and alerting system that is widely adopted in the Kubernetes ecosystem. It is designed to collect and store metrics from various sources, including Kubernetes cluster components, applications, and infrastructure. Prometheus plays a crucial role in Kubernetes deployments due to its ability to provide comprehensive monitoring and alerting capabilities.

Prometheus collects metrics from targets (e.g., Kubernetes nodes, pods, services) using a pull-based model. It scrapes metrics data from these targets at regular intervals, allowing you to monitor various aspects of your Kubernetes cluster, such as resource utilization, application performance, and cluster health. Additionally, Prometheus provides a flexible and powerful query language (PromQL) that allows you to analyze and visualize collected metrics. PromQL enables you to perform complex queries, aggregations, and calculations on the collected data, enabling advanced monitoring and troubleshooting capabilities.

Moreover, Prometheus supports configurable alerting rules based on metric expressions. When specific conditions are met, Prometheus can trigger alerts and send notifications to various channels (e.g., email, Slack, PagerDuty), enabling you to promptly respond to issues or anomalies within your Kubernetes cluster. Prometheus also integrates seamlessly with Kubernetes through service discovery mechanisms and Kubernetes API integration, automatically discovering and monitoring Kubernetes resources without requiring manual configuration changes as your cluster scales or evolves.

Prometheus is designed to be highly scalable and can handle large volumes of metrics data. It supports horizontal scaling by running multiple Prometheus instances in a federation, ensuring high availability and fault tolerance. Furthermore, Prometheus can be extended with various exporters and integrations, allowing you to collect metrics from a wide range of sources, providing a comprehensive monitoring solution for your entire Kubernetes environment.

The importance of Prometheus in Kubernetes deployments lies in its ability to provide a comprehensive monitoring solution tailored to the dynamic and distributed nature of Kubernetes clusters. By collecting and analyzing metrics from various components and applications, Prometheus enables you to gain insights into the health and performance of your Kubernetes deployment. Additionally, its alerting capabilities allow you to proactively identify and respond to issues, ensuring the stability and reliability of your applications running on Kubernetes. Prometheus's seamless integration with Kubernetes and its scalability make it a popular choice for monitoring large-scale Kubernetes deployments, enabling organizations to maintain visibility and control over their containerized applications and infrastructure.

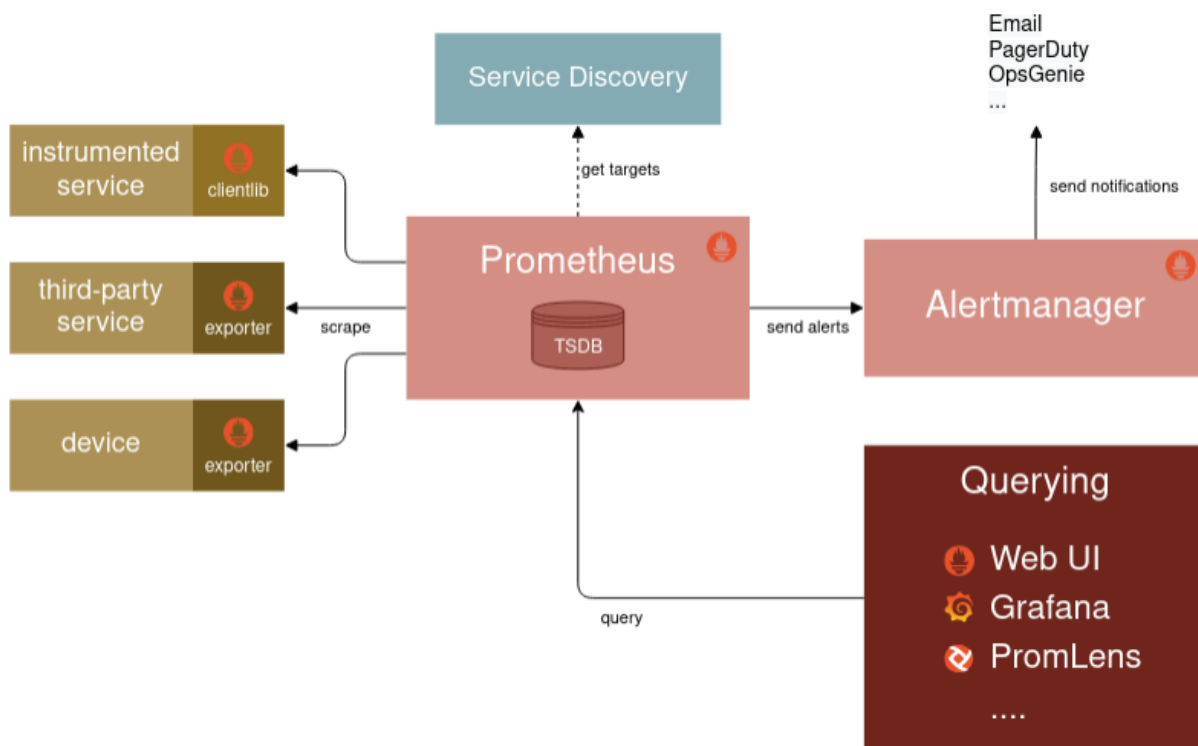


Figure 14 - Prometheus architecture integrated with Grafana and Alertmanager

4.2.1.4 Loki

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system inspired by Prometheus. It is designed to be lightweight and cost-effective, making it well-suited for log analytics in cloud-native environments like Kubernetes clusters. In a Kubernetes cluster, Loki can be used for centralizing and analyzing logs from various components and applications.

Loki collects logs from different sources, such as containers, pods, and nodes within the Kubernetes cluster. It uses a client-side agent called Promtail, which is responsible for collecting and forwarding logs to the Loki server. Loki then indexes the collected logs based on their labels and timestamps, making it easy to search and filter logs based on various criteria. The indexed logs are stored in a cost-effective and scalable object storage system, like Amazon S3, Google Cloud Storage, or a local file system.

Loki provides a powerful query language called LogQL, which is similar to PromQL used by Prometheus. LogQL allows users to filter, search, and analyze logs based on different criteria, such as labels, timestamps, and log content patterns. Loki also integrates well with Prometheus, enabling users to correlate and analyze logs alongside metrics collected by Prometheus. This integration allows for comprehensive monitoring and troubleshooting of applications and infrastructure within the Kubernetes cluster.

Loki is designed to be horizontally scalable, allowing you to scale out the components (like the Loki server and Promtail agents) as the logging volume increases. It also supports high availability configurations, ensuring that logs are available even in the event of component failures. Additionally, Loki supports multi-tenancy, allowing you to isolate and secure log data for different teams, projects, or environments within the same Kubernetes cluster.

One of the key advantages of Loki is its cost-effective storage approach. Loki's architecture allows for cost-effective storage of logs by compressing and storing them in object storage systems, which are typically more cost-effective than traditional database systems for log storage. By deploying Loki in a Kubernetes cluster, organizations can benefit from a centralized, scalable, and efficient log management solution. Loki's integration with Prometheus and its ability to handle large volumes of log data make it a valuable tool for monitoring, troubleshooting, and analyzing applications and infrastructure in cloud-native environments.

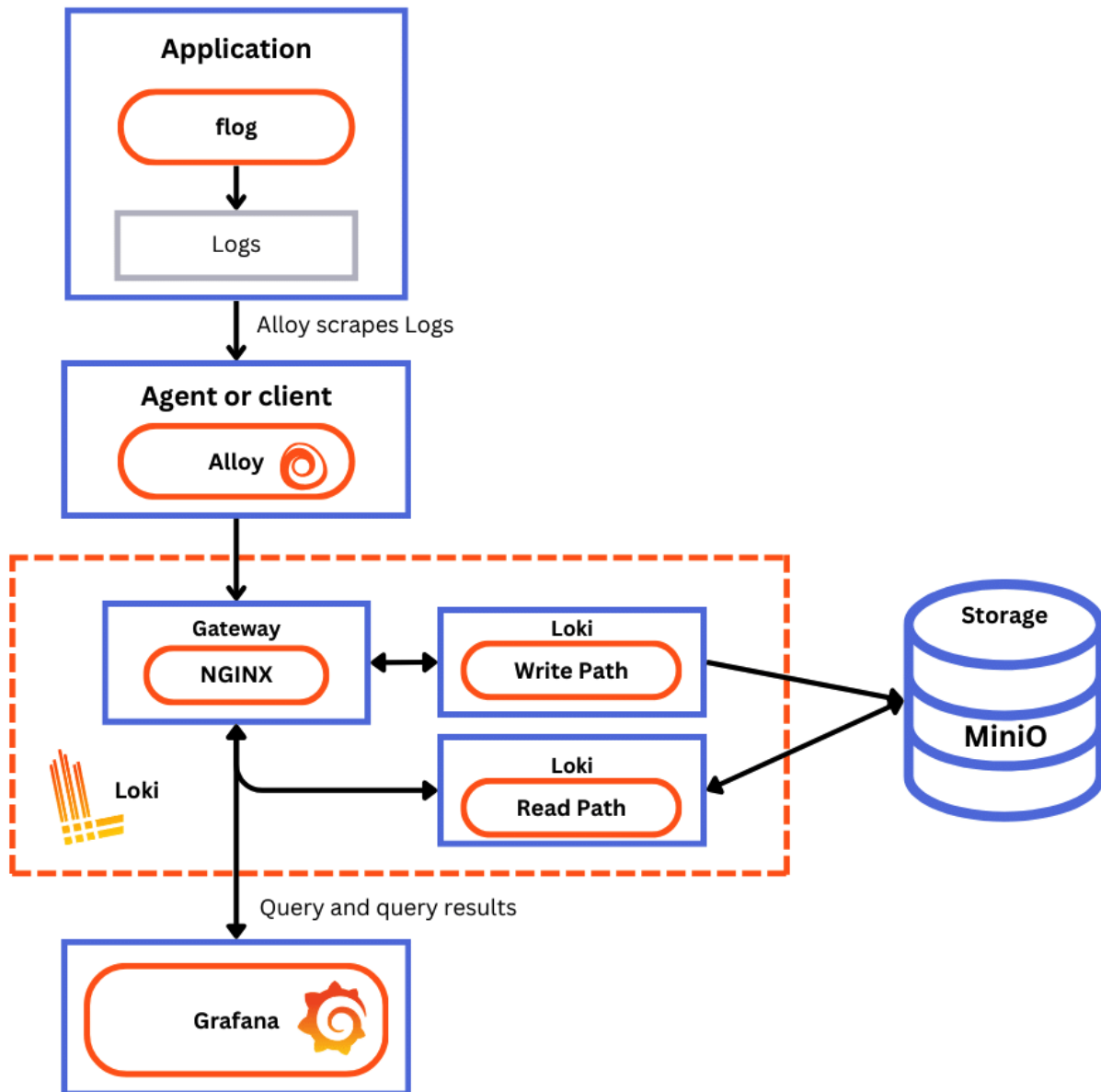


Figure 15 - Architecture of Loki integrated with Grafana and Nginx

4.2.1.5 Grafana

Grafana is an open-source data visualization and monitoring platform that provides a unified interface for querying, analyzing, and visualizing metrics and logs from various data sources. In the context of Kubernetes, Grafana can be an invaluable tool for monitoring and alerting within the cluster.

When integrated with monitoring systems like Prometheus and Loki, Grafana allows users to create customizable dashboards that display real-time metrics and logs from Kubernetes components, such as nodes, pods, and services. These dashboards provide a comprehensive view of the cluster's health and performance, enabling administrators to quickly identify and troubleshoot issues.

Grafana's alerting functionality is particularly useful for Kubernetes monitoring. Users can define alert rules based on specific metrics or log patterns, and Grafana will send notifications through various channels (e.g., email, Slack, PagerDuty) when those conditions are met. This proactive alerting system helps ensure that potential problems are detected and addressed before they escalate, minimizing downtime and ensuring the stability of applications running in the Kubernetes cluster.

Furthermore, Grafana supports a wide range of data sources beyond Prometheus and Loki, allowing users to integrate data from other monitoring tools, databases, and even custom data sources. This flexibility enables comprehensive monitoring and analysis of the entire Kubernetes ecosystem, including applications, infrastructure, and external dependencies.

Grafana's user-friendly interface and extensive plugin ecosystem also contribute to its popularity in the Kubernetes community. Users can customize and extend Grafana's functionality with various plugins, such as dashboards, data sources, and authentication mechanisms, tailoring the platform to their specific needs.

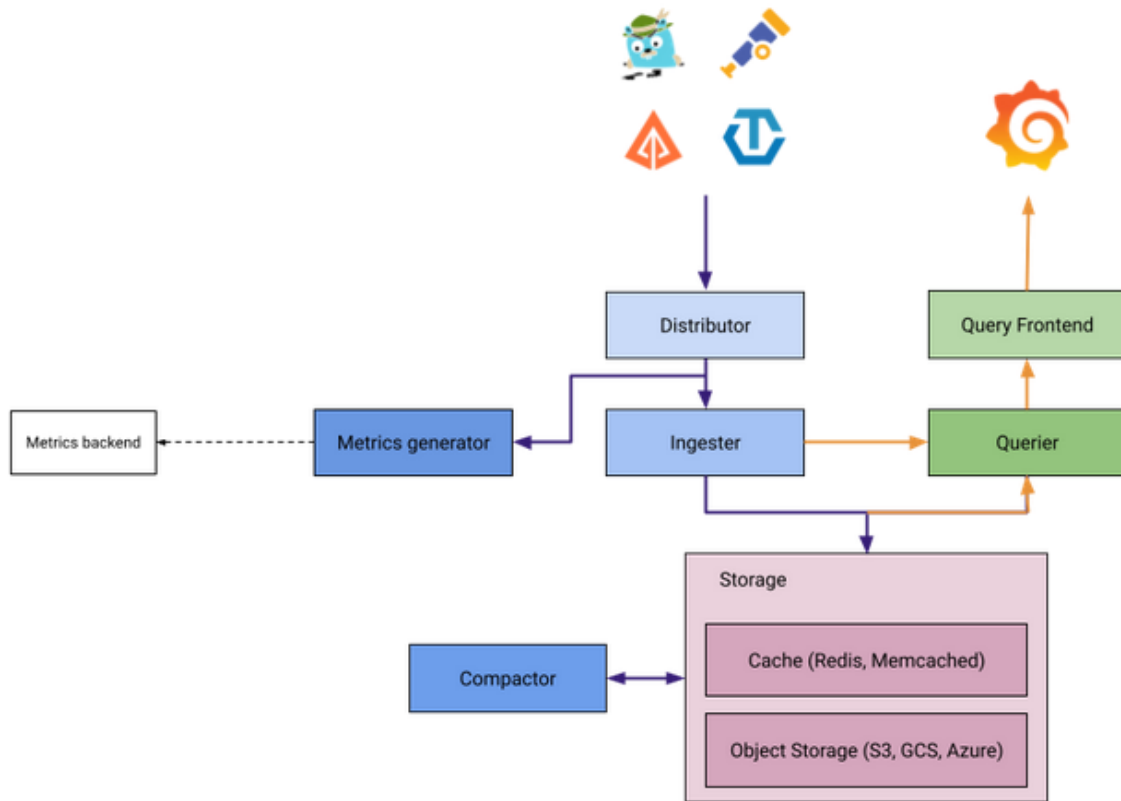


Figure 16 - Grafana Architecture

4.2.1.5 Kubernetes dashboard

Kubernetes dashboards are web-based user interfaces that provide a centralized view and management console for Kubernetes clusters. They serve as an important tool for administrators, developers, and operators, enabling them to monitor, manage, and troubleshoot their cluster resources and applications efficiently. Kubernetes dashboards offer a comprehensive overview of the entire cluster, including nodes, pods, deployments, services, and other resources. This bird's-eye view allows users to quickly assess the overall cluster status, identify potential issues, and track resource usage and allocation.

Dashboards provide a user-friendly interface for managing Kubernetes resources. Users can create, edit, scale, and delete resources such as deployments, services, and persistent volumes directly from the dashboard. This streamlines the resource management process and reduces the need for command-line interactions. Additionally, Kubernetes dashboards display real-time metrics and

logs for various cluster components, enabling users to monitor the health and performance of their applications and infrastructure. Visualizations and graphs help identify bottlenecks, resource constraints, and other issues quickly. Dashboards often provide access to logs for easier troubleshooting and application management.

Many Kubernetes dashboards support role-based access control (RBAC) and multi-tenancy features, allowing organizations to grant appropriate access levels to different teams or users, ensuring secure and isolated management of cluster resources and applications. Dashboards can also integrate with other tools and services within the Kubernetes ecosystem, such as monitoring systems, logging platforms, and continuous integration/deployment (CI/CD) pipelines. This integration provides a centralized platform for managing and monitoring the entire Kubernetes infrastructure.

Alternative to this is the kubernetes CLI (kubectl), which is the most powerful, and API communication to kubernetes clusters but the kubernetes dashboard is often more user friendly.

4.2.1.6 Docker

Docker plays a crucial role in Kubernetes deployments, making it an essential component of the ecosystem. Its importance can be highlighted through various aspects, from containerizing applications to serving as the container runtime.

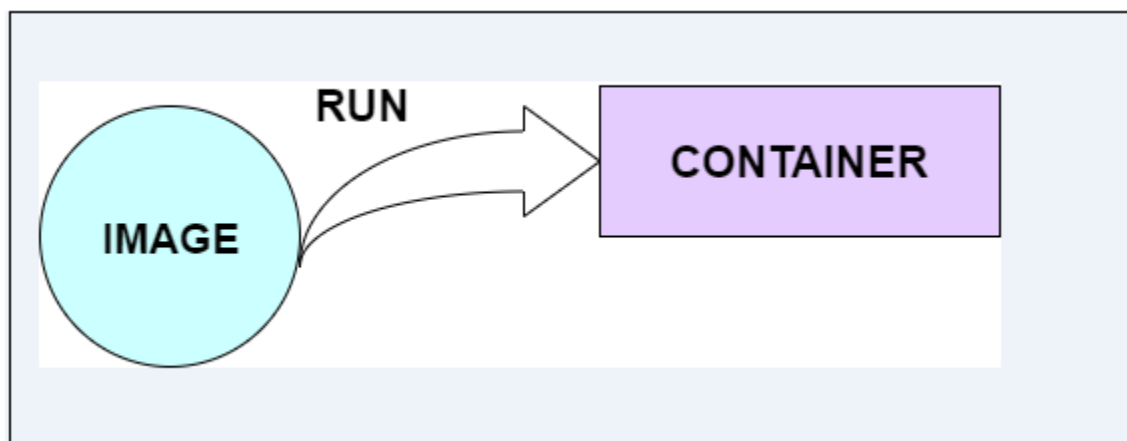


Figure 17 - Docker running a container from an image

First and foremost, Docker provides a standardized and efficient way to package applications and their dependencies into lightweight, portable containers. This containerization process ensures that applications can run consistently across different environments, eliminating the infamous "it works on my machine" issue. By encapsulating applications within containers, Kubernetes can easily manage and orchestrate their deployment, scaling, and load balancing across the cluster.

Furthermore, Docker serves as the default container runtime in Kubernetes. A container runtime is responsible for managing the lifecycle of containers, including creating, running, and stopping them. Kubernetes leverages Docker's capabilities to efficiently manage and schedule containers across multiple nodes in the cluster. This integration between Kubernetes and Docker ensures seamless communication and resource allocation, enabling efficient utilization of cluster resources.

Additionally, Docker provides a comprehensive set of tools and features that facilitate the development, deployment, and management of containerized applications. These tools include features like image building, container networking, and persistent storage management. Kubernetes takes advantage of these features, allowing developers and operators to streamline the application deployment process and ensure consistent behavior across different environments.

Docker's layered filesystem and image caching mechanisms also contribute to efficient resource utilization in Kubernetes clusters. By sharing common layers among containers, Docker reduces the overall storage footprint and minimizes network bandwidth usage during image distribution and deployment. This efficiency becomes increasingly important as the number of applications and containers in a Kubernetes cluster grows.

Moreover, Docker's ecosystem and community support have fostered the development of a vast array of third-party tools and integrations. These tools enable seamless integration with continuous integration/continuous deployment (CI/CD) pipelines, monitoring solutions, and other essential components of a modern application delivery infrastructure. Kubernetes leverages this rich ecosystem, enabling organizations to build robust and scalable application deployment pipelines.

4.2.1.7 MySQL 5.5

MySQL 5.5, released in December 2010, was a significant release of the popular open-source relational database management system, introducing several enhancements and new features aimed at supporting application databases more efficiently and effectively. One of the key introductions was semi-synchronous replication, which provided a middle ground between asynchronous and fully synchronous replication, ensuring data integrity while minimizing the impact on write performance, making it suitable for applications that require high availability and durability.

MySQL 5.5 was used because it is the chosen database manager for Sakai 20.

4.2.1.8 Sakai v20

Sakai is an open-source learning management system (LMS) developed by a community of institutions, organizations, and individuals. It is a robust and highly customizable platform that provides a comprehensive set of tools and features for delivering and managing online courses, facilitating collaboration, and supporting various educational activities.

The decision to use Sakai version 20 as the foundation for deploying Thuto LMS on Kubernetes for testing purposes can be justified by several factors. Firstly, Sakai is a mature and well-established LMS with a strong community support and a wide range of functionalities that cater to the diverse needs of educational institutions. By leveraging Sakai's codebase, the development team can benefit from the extensive features and tools already available, reducing the effort required to build a custom LMS from scratch.

Furthermore, Sakai's modular architecture and extensible nature make it highly customizable and adaptable to specific requirements. The development team can tailor the LMS to meet the unique needs of Thuto LMS by modifying or extending existing components or adding new functionality as needed. This flexibility ensures that the deployed version of Thuto LMS on Kubernetes accurately reflects the desired features and functionalities.

Additionally, testing the deployment of Thuto LMS on Kubernetes using Sakai version 20 provides a realistic and representative environment for evaluating the system's performance, scalability, and reliability. Kubernetes is a powerful container orchestration platform that enables efficient management and scaling of containerized applications. By deploying Sakai 20 on Kubernetes, the development team can assess the feasibility and challenges of running a complex LMS in a containerized environment, gaining valuable insights and experience for future production deployments.

4.2.1.9 Ingress Nginx

Ingress Nginx is a popular open-source solution for providing ingress control and load balancing for applications running in Kubernetes clusters. An ingress controller is a crucial component in Kubernetes that acts as an entry point for external traffic into the cluster, routing incoming requests to the appropriate services based on defined rules.

The decision to use Ingress Nginx in the Kubernetes deployment of Thuto LMS can be justified by several factors. Firstly, Ingress Nginx is a widely adopted and well-maintained project, backed by a large and active community. This ensures reliable support, frequent updates, and a responsive issue-handling process, which is essential for deploying and managing mission-critical applications like Thuto LMS.

Moreover, Ingress Nginx offers a rich set of features and capabilities that enhance the functionality and flexibility of the Kubernetes deployment. It supports advanced routing rules, SSL/TLS termination, path-based routing, and multiple load balancing algorithms, allowing for fine-grained control over traffic management. This granular control enables the efficient distribution of incoming requests to the appropriate Thuto LMS components, ensuring optimal performance and resource utilization.

Additionally, Ingress Nginx is highly configurable and extensible, allowing for seamless integration with various monitoring and logging tools, as well as third-party services like API gateways or content delivery networks (CDNs). This extensibility ensures that the Thuto LMS

deployment can be tailored to meet specific requirements and integrate with existing infrastructure or services.

Furthermore, Ingress Nginx provides robust security features, such as support for HTTP/2, gRPC, and WebSockets, as well as the ability to enforce security policies and implement rate limiting. These features are crucial for ensuring the secure and reliable delivery of Thuto LMS services to end-users.

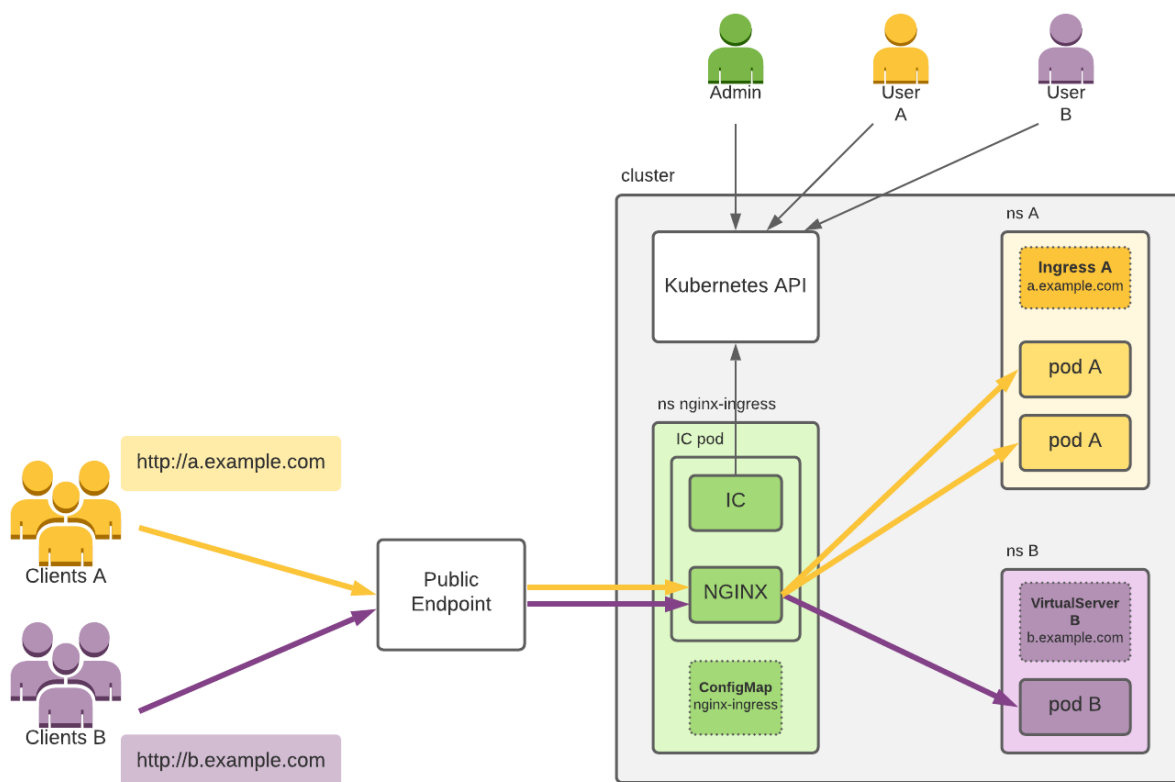


Figure 18 - Architecture of ingress nginx functioning as a reverse proxy

4.2.2 The actual steps followed in implementing the system

This section typically outlines the step-by-step process that was undertaken to bring the system from conceptualization to a fully implemented and functional state. This section aims to provide a clear and detailed account of the various stages involved in the implementation process, allowing

readers to understand the methodology, tools, and techniques employed throughout the project's lifecycle.

4.2.2.1 Dockerization of MySQL 5.5 and Sakai v20 code

Containerizing an application into Docker images is considered the first step in achieving a Kubernetes deployment because Kubernetes is designed to orchestrate and manage containerized applications. This step is crucial for several reasons.

Firstly, Docker containers provide a consistent and isolated runtime environment for applications, ensuring that the application runs the same way across different environments (development, testing, staging, and production). This portability is essential for seamless deployment and scalability in Kubernetes clusters. Additionally, Docker containers isolate applications from each other and from the host system, providing a lightweight and efficient way to package and distribute applications. This isolation ensures that applications do not interfere with each other and can share resources on the same host.

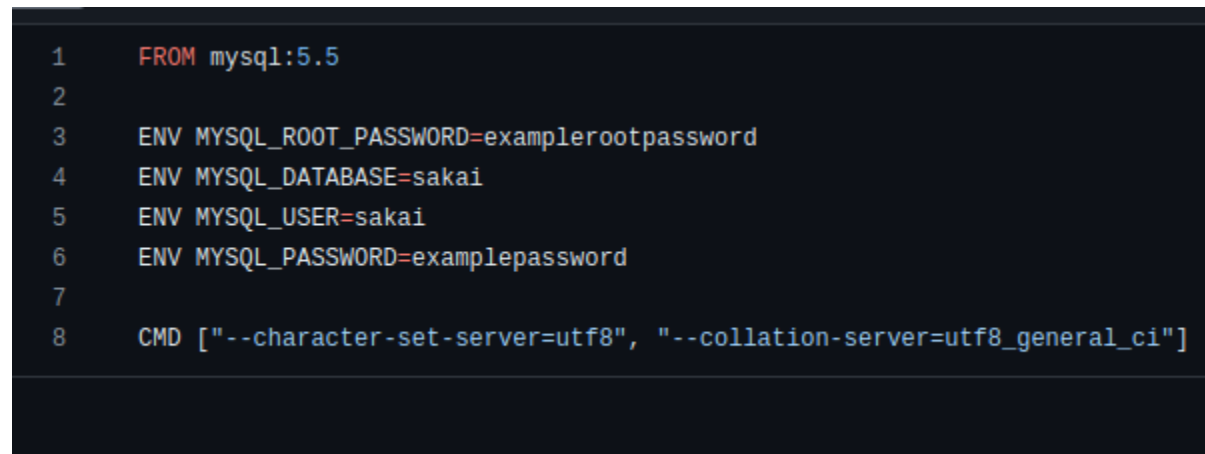
Furthermore, Docker images are self-contained and immutable, meaning that they include all the dependencies and configurations required to run the application. This makes it easier to deploy applications consistently across different environments without worrying about missing dependencies or configuration issues. Moreover, Kubernetes is designed to manage and scale containerized applications efficiently. By packaging applications into Docker images, Kubernetes can easily spin up multiple instances of the application across different nodes in the cluster, enabling horizontal scaling and high availability.

Docker images also ensure that the application and its dependencies are packaged together in a consistent and reproducible manner. This consistency helps to eliminate discrepancies between development, testing, and production environments, reducing the risk of issues caused by environmental differences. Additionally, Kubernetes natively supports Docker containers and provides a robust set of tools and APIs for managing and orchestrating containerized applications.

By containerizing applications with Docker, developers can leverage the full power and features of Kubernetes for deployment, scaling, and management.

4.2.2.1.1 MySQL 5.5 Dockerized for Sakai

The following screenshot depicts that the MySQL image was built with the specifications of Sakai, so it is specific to the Sakai app.



```
1 FROM mysql:5.5
2
3 ENV MYSQL_ROOT_PASSWORD=examplerootpassword
4 ENV MYSQL_DATABASE=sakai
5 ENV MYSQL_USER=sakai
6 ENV MYSQL_PASSWORD=examplepassword
7
8 CMD ["--character-set-server=utf8", "--collation-server=utf8_general_ci"]
```

Figure 19 - Containerization of MySQL for Sakai

4.2.2.1.2 Sakai v20 Dockerization

The following images depict that the Sakai image was built with the sakai.properties.

```

1 FROM maven:3.6.1-jdk-8 as build
2
3 ARG release=master
4
5 COPY lib/settings.xml /usr/share/maven/conf/settings.xml
6 RUN mkdir /deploy
7 WORKDIR /deploy
8 RUN git clone https://github.com/sakaiproject/sakai.git
9 WORKDIR /deploy/sakai
10 RUN git checkout ${release}
11 WORKDIR /deploy/sakai/master
12 RUN mvn clean install
13 WORKDIR /deploy/sakai
14 RUN mvn clean install sakai:deploy -Dmaven.test.skip=true
15
16
17 FROM tomcat:9.0.20-jre8
18
19 COPY lib/server.xml /usr/local/tomcat/conf/server.xml
20 COPY lib/context.xml /usr/local/tomcat/conf/context.xml
21 COPY --from=build /deploy/components /usr/local/tomcat/components/
22 COPY --from=build /deploy/lib /usr/local/tomcat/sakai-lib/
23 COPY --from=build /deploy/webapps /usr/local/tomcat/webapps/
24
25 RUN mkdir -p /usr/local/sakai/properties
26 COPY sakai.properties /usr/local/sakai/properties/
27
28 ENV CATALINA_OPTS MEMORY -Xms2000m -Xmx2000m
29 ENV CATALINA_OPTS \
30 -server \
31 -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+UseConcMarkSweepGC -XX:+UseParNewGC \
32 -XX:+CMSParallelRemarkEnabled -XX:+UseCompressedOops -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=80 -XX:TargetSurvivorRatio=90 \
33 -Djava.awt.headless=true \
34 -Dsun.net.inetaddr.ttl=0 \
35 -Dsakai.component.shutdownonerror=true \
36 -Duser.language=en -Duser.country=US \
37 -Dsakai.home=/usr/local/sakai/properties/ -Dsakai.security=/usr/local/tomcat/sakai/ \
38 -Duser.timezone=US/Eastern \
39 -Dsun.net.client.defaultConnectTimeout=300000 \
40 -Dsun.net.client.defaultReadTimeout=1800000 \
41 -Dorg.apache.jasper.compiler.Parser.STRICT_QUOTE_ESCAPING=false \
42 -Dsun.lang.ClassLoader.allowArraySyntax=true \
43 -Dhttp.agent=Sakai \
44 -Djava.util.Arrays.useLegacyMergeSort=true
45
46 RUN sed -i '/^common.loader=/ s/$/, "${catina.base}\sakai-lib\*.jar"/' /usr/local/tomcat/conf/catalina.properties

```

Figure 20 - Containerization of Sakai (1/2)

```

47
48 RUN curl -L -o /usr/local/tomcat/lib/mysql-connector-java-5.1.47.jar https://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.47/mysql-connector-java-5.1.47.jar
49
50 RUN mkdir -p /usr/local/tomcat/sakai
51 COPY security.properties /usr/local/tomcat/sakai/
52 COPY lib/entrypoint.sh /entrypoint.sh
53 RUN chmod +x /entrypoint.sh
54 ENTRYPOINT ["/entrypoint.sh"]
55

```

Figure 21 - Containerization of Sakai (2/2)

4.2.2.2 Deployment on kubernetes

The following is a depiction of the kubernetes cluster as viewed using kubeview. This shows the connection between the deployments and services as well as their associated persistent volume claims (PVCs). Added to that is the ingress nginx controller which is the reverse proxy for routing data externally from the cluster.

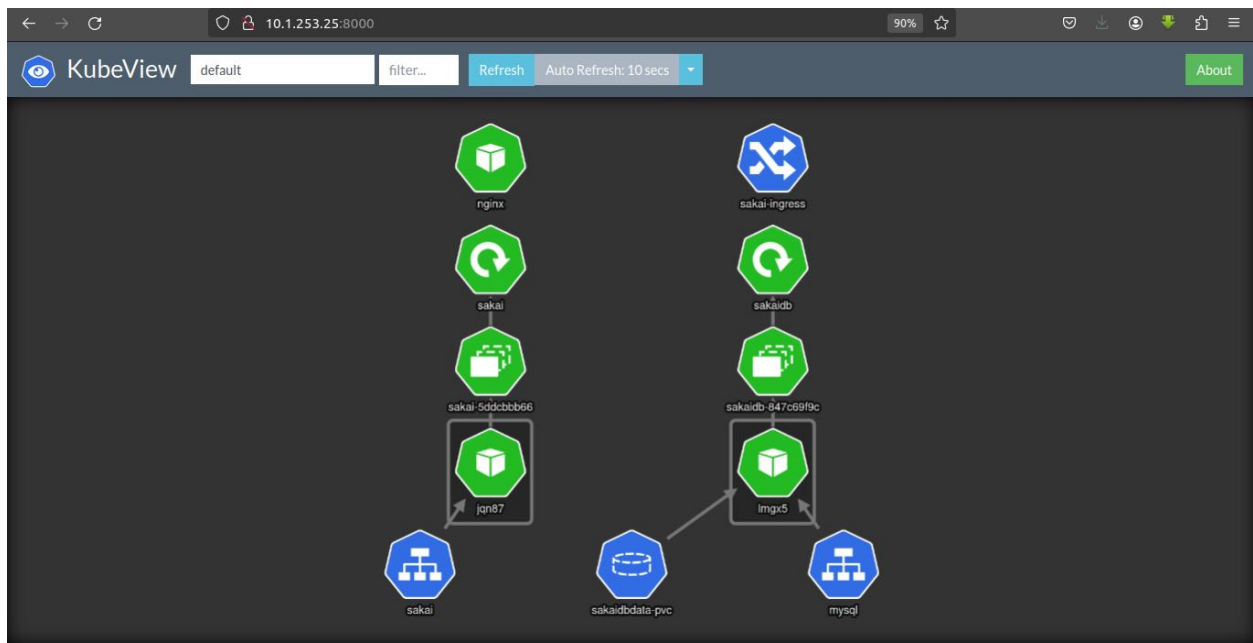


Figure 22 - The deployment of Sakai together with the database and PVCs and its Ingress controller

4.2.2.3 Deployment using helm charts templating

The following is a depiction of the helm charts as organized into a single template and the storageclasses for kubernetes. The use of helm charts to template the deployment is really helpful in that, the charts can easily be scaled up whenever necessary.

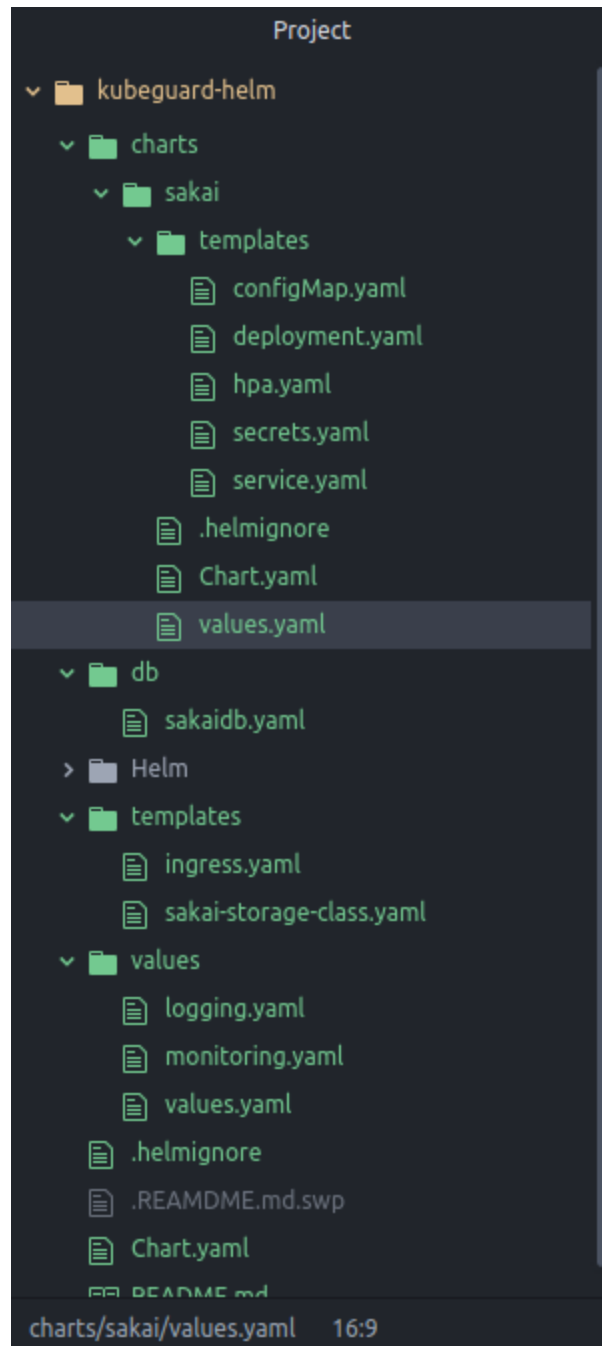


Figure 23 - The Helm charts as organized for the release of the deployment

4.2.2.4 Integration of Monitoring and Logging - The Prometheus Stack

The prometheus stack visualization using kubeview. This stack was helpful in monitoring and defining alerting rules using Grafana and the *promQL*.

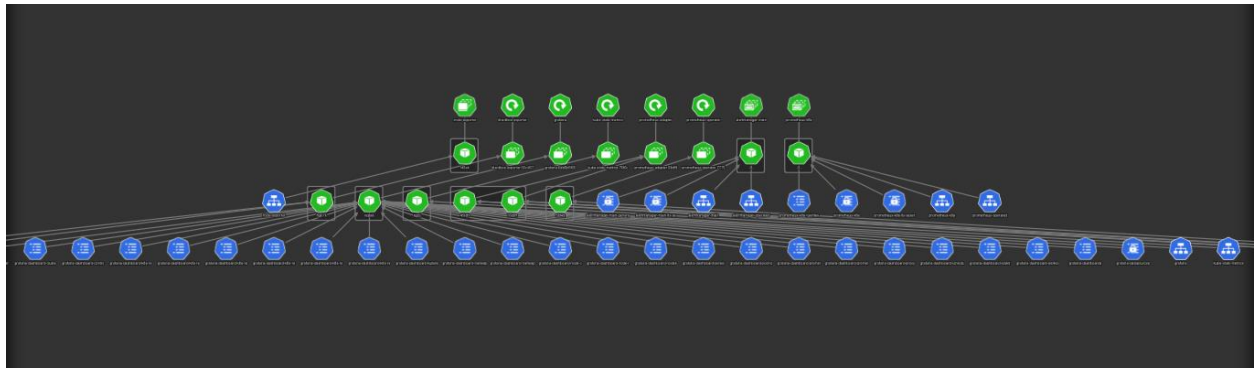


Figure 24 - The architecture of the prometheus stack for the purposes of logging and monitoring

4.3 System Testing

The following table provides a comprehensive overview of the project requirements, test results, and their alignment with the specified objectives. It serves as a concise yet informative representation of the project's achievements, accompanied by insightful comments that offer valuable context and clarification.

This tabular representation offers a clear and structured way to present the project's progress, allowing readers to easily correlate the requirements with their corresponding test results and supplementary comments. The comments section provides an opportunity to elaborate on any nuances, challenges, or significant achievements that may not be immediately apparent from the test results alone.

Table 2 - System testing results and comments

Requirements	Test Results	Comments
Containerize Thuto LMS	Thuto LMS by use of Sakai 20 code was successfully dockerized together with its relevant MySQL. A standalone deployment on docker was achieved.	Successful

Deploy Thuto LMS on Kubernetes	Thuto LMS successfully deployed on Microk8s using helm charts. It was accessible on the web browser by the use of ingress nginx reverse proxy.	Successful
Anomaly Detection for Thuto LMS Metrics	Anomalies such as CPU utilization and Memory utilization were detected on the dashboard of kubernetes as well as on the grafana dashboard.	Although these were not tested on the level of stress-testing the set threshold, they could be displayed by graphs real-time.
Alerting System	The alertmanager for prometheus was successfully deployed as part of the monitoring namespace. Alerts using the grafana dashboard were also set.	The functionality of the alerting system was not fully tested as the requirement of the SMTP server for emails was not achieved.
Auto-Remediation for Thuto LMS Workloads	Auto-remediation techniques such as HPA were used to set threshold for the purposes of upscaling pods when necessary were set. Automatic restart of pods as a form of auto-remediation was also set.	HPA was not tested due to the unavailability of stress-testers. The restarting of pods automatically whenever they died was tested and was successful.
Dashboard and Reporting for	Kubernetes dashboard as well	The requirement to create our


```
resources: {}
  requests:
    cpu: 10m # Request 100 millicpu (0.1 cpu)
    memory: 28Mi # Request 128 MiB of memory
  limits:
    cpu: 500m
    memory: 512Mi
autoscaling:
  enabled: true
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 80
  targetMemoryUtilizationPercentage: 80
```

Figure 27 - The definition of parameters for HPA for Sakai

4.2.3 Test Results

This section provides a comprehensive overview of the testing efforts undertaken to validate the successful implementation and functionality of the system. It details the various test cases executed, their corresponding results, and any relevant observations or insights derived from the testing process. This section serves as a transparent assessment of the system's performance, enabling stakeholders and project members to ensure that the implemented solution meets the desired quality standards.

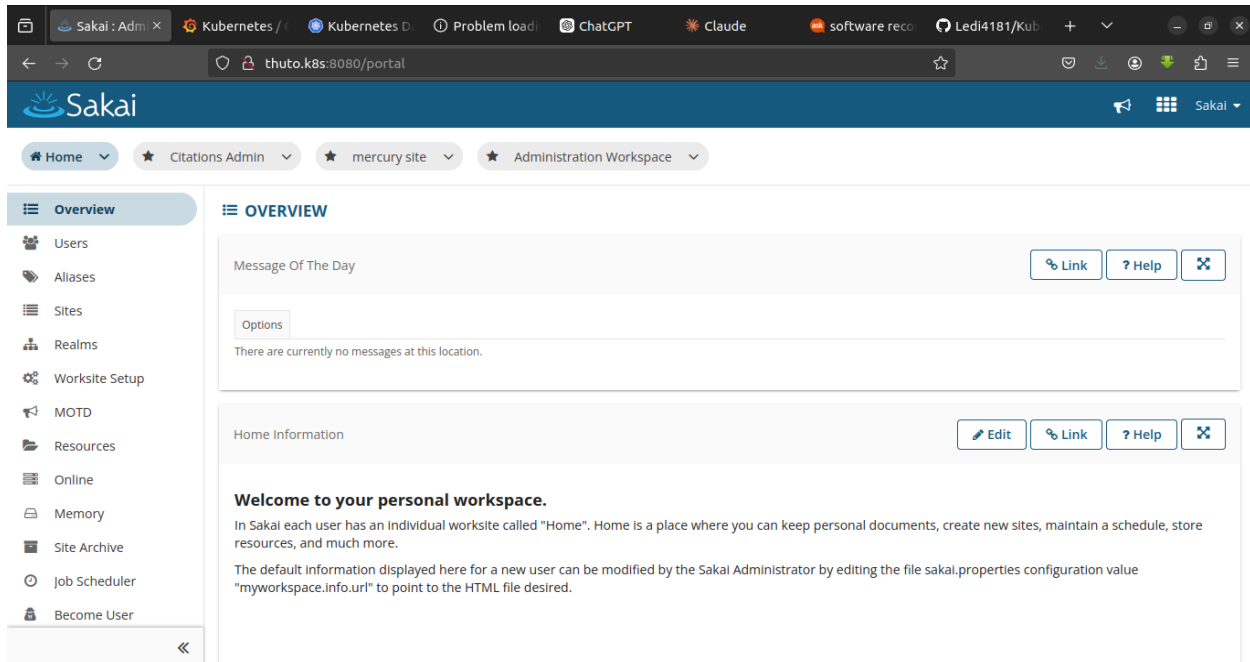


Figure 28 - Accessing Thuto LMS on the web, on the defined host: *thuto.k8s*

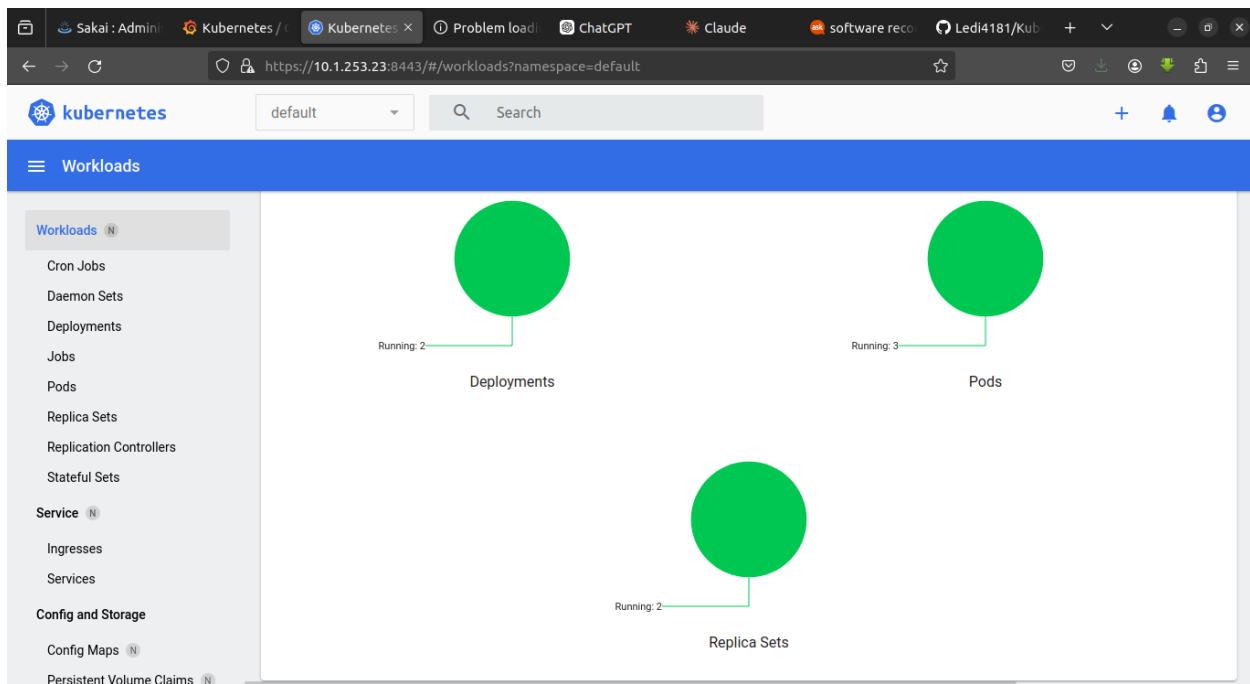


Figure 29 - The workloads of the deployment incl. Deployments, Pods & Replica Sets

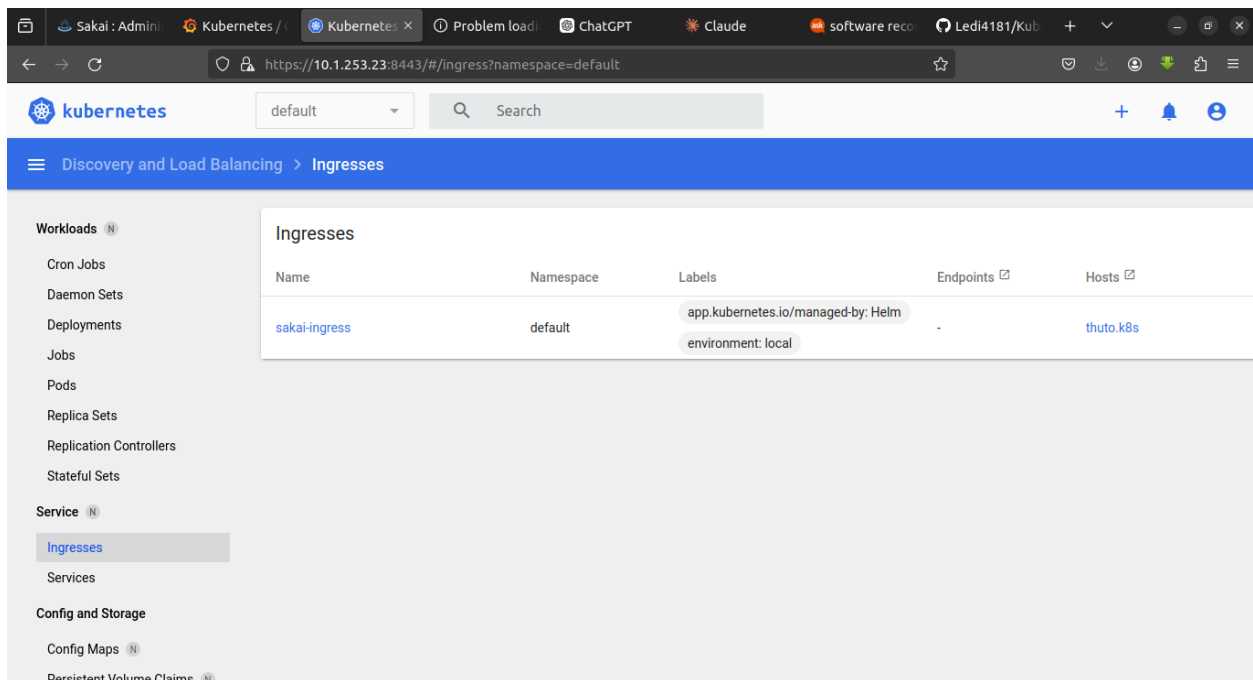


Figure 30 - The ingress nginx controller showing the domain name (thuto.k8s)

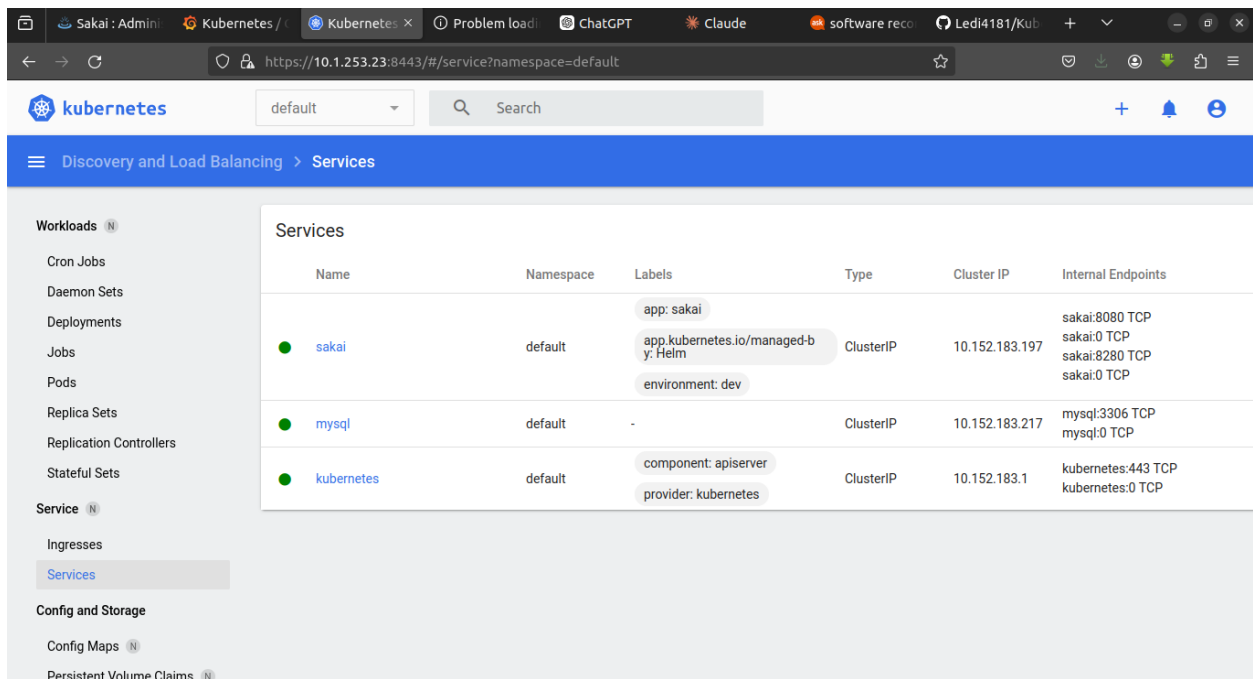


Figure 31 - The services of the cluster which are used to communicate within cluster

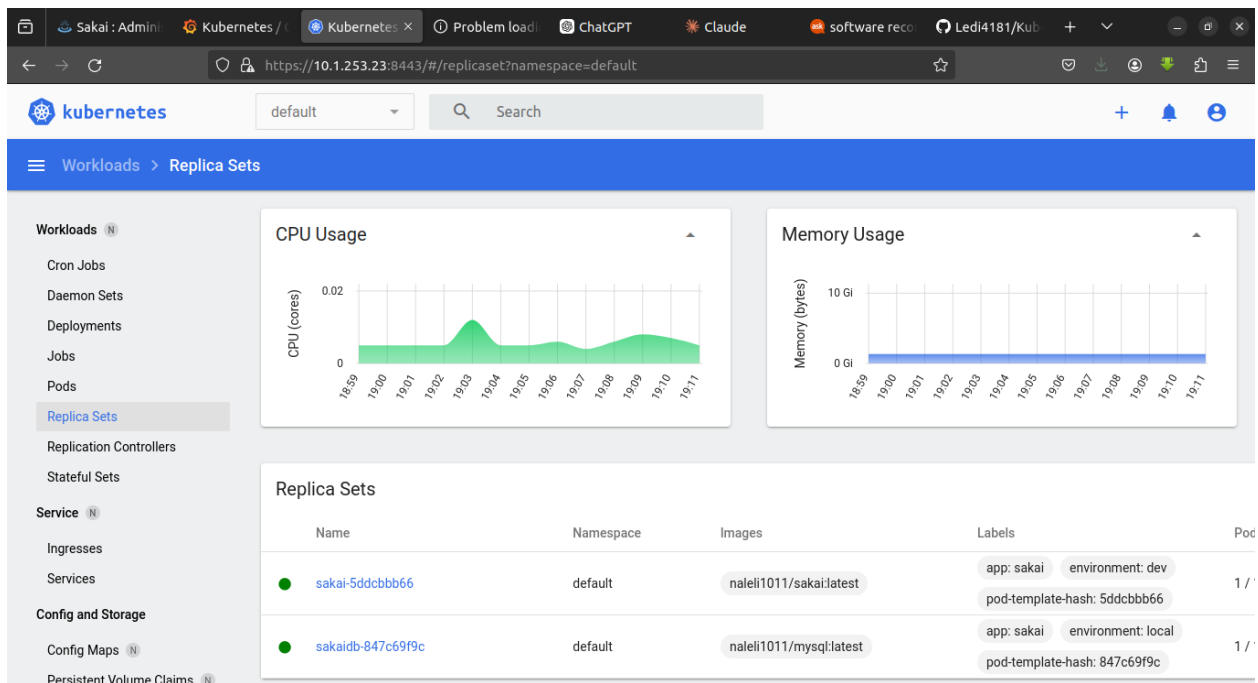


Figure 32 - The defined thresholds for the Thuto LMS metrics (CPU, Memory)

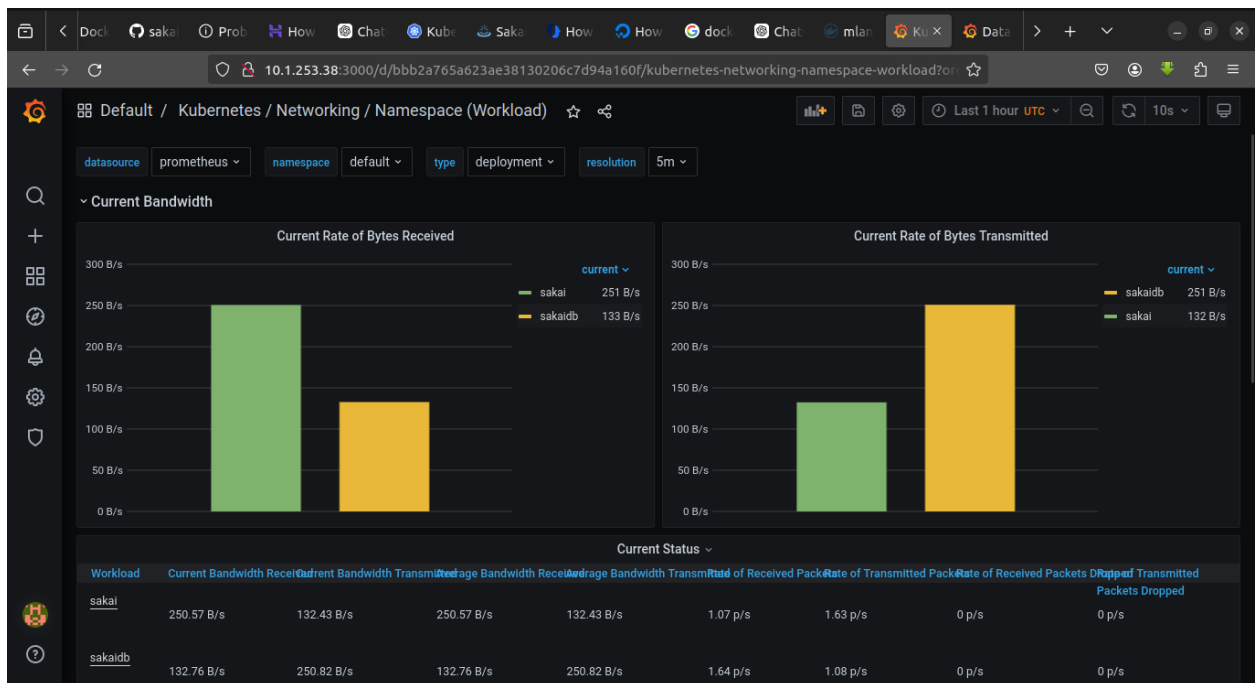


Figure 33 - The visualization of the workloads for the deployment using Grafana (1/2)

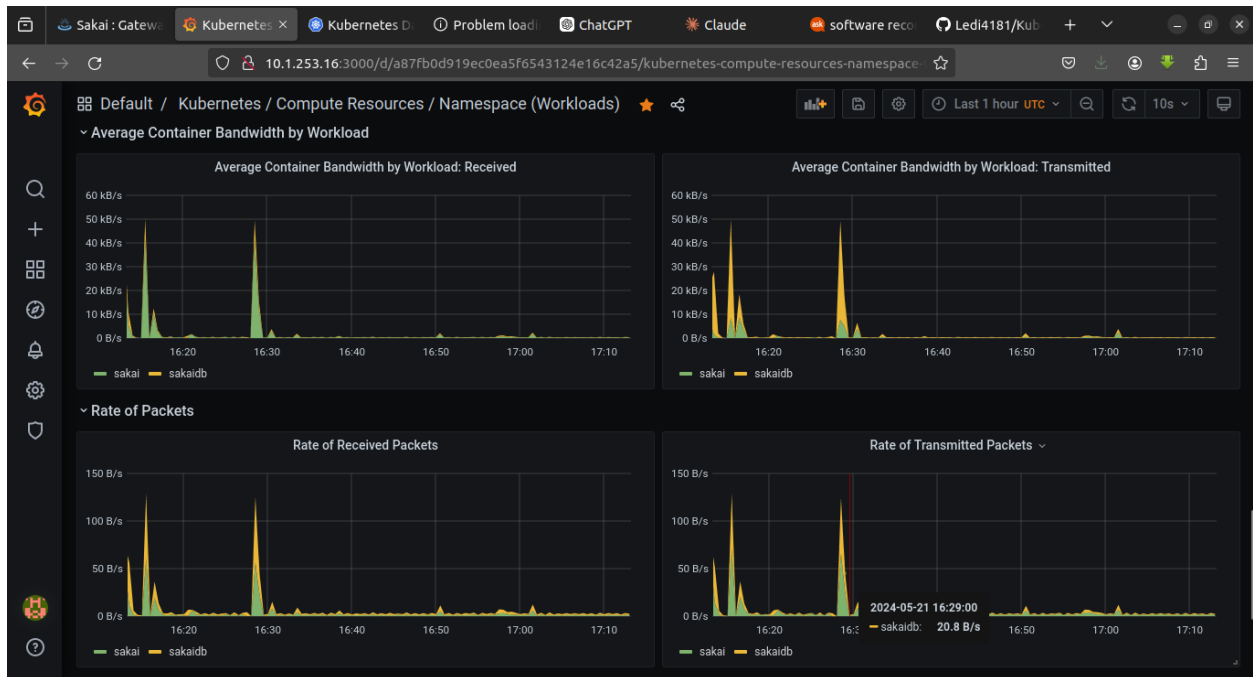


Figure 34 - The visualization of the workloads for the deployment using Grafana (2/2)

4.3.2 Testing non-functional requirements

Upon testing the functionality of automatically restarting a pod when it's deleted, we realized that it took some time for the pod to be up. During that restarting of a pod, the system was unavailable and this was due to the fact that initially, we had only one replica (pod) of the sakai application, so we decided to overcome this by having at least two instances (pods) of sakai to mitigate that. This will enhance the availability of the sakai application so that we have zero to none down time.

CHAPTER 5

Conclusion, Limitations, and Recommendations

5.1 Conclusion

The KubeGuard project, led by Tatolo Matlali and Naleli Seboka Mokake, has culminated in a comprehensive solution for enhancing the Thuto LMS deployment at the National University of Lesotho. By containerizing the Thuto web application and MySQL database and deploying them on a Kubernetes cluster, the project aims to address critical challenges related to scalability, resource management, and high availability within the educational ecosystem. The project's focus on anomaly detection and auto-remediation mechanisms signifies a proactive approach to system maintenance, ensuring a seamless and reliable learning experience for students and faculty. While the project faced some limitations, such as the unavailability of the actual Thuto LMS source code and resource constraints during development, the team was able to successfully implement a proof-of-concept solution that showcases the benefits of containerization, Kubernetes, and advanced monitoring and auto-remediation capabilities for the Thuto LMS. The successful deployment of the Sakai 20 codebase on Docker and Kubernetes, along with the implementation of key features like anomaly detection, alerting, auto-remediation, and monitoring dashboards, demonstrates the project's overall success in enhancing the Thuto LMS infrastructure.

In conclusion, the KubeGuard project represents a significant step towards modernizing and optimizing the Thuto LMS deployment, paving the way for a more robust and adaptive educational platform. The dedication, expertise, and collaborative efforts of the project team, under the guidance of Supervisor Mr. Lebajoa Mphatsi, have been instrumental in shaping a solution that not only addresses current challenges but also sets the stage for future growth and innovation in online education delivery. The project's holistic approach to system design, monitoring, and maintenance underscores a commitment to excellence and continuous improvement, ensuring that the Thuto LMS remains at the forefront of technology-enhanced learning experiences at the National University of Lesotho.

5.2 Limitations

- The project utilized the Sakai open-source LMS codebase instead of the actual Thuto LMS source code, as the latter was not readily available.
- The current Thuto LMS deployment follows a monolithic architecture, which may limit its scalability and flexibility as the system grows. A microservices-based architecture could be more suitable in the long run.
- The project was constrained by the available computing resources, as the team faced limitations in upgrading their personal computers' RAM to handle the resource-intensive Kubernetes deployment.

5.3 Recommendations (Future Work)

- Migrate the Thuto LMS to a microservices-based architecture, which would allow for more granular scaling, independent deployment, and better overall system resilience.
- Explore the feasibility of deploying the Thuto LMS on a cloud-based Kubernetes infrastructure, which would provide auto-scaling capabilities and access to a wider range of cloud-native services and tools.
- Enhance the anomaly detection and auto-remediation mechanisms by incorporating more advanced machine learning techniques and expanding the scope of monitored metrics and events.
- Integrate the KubeGuard solution with the existing Thuto LMS deployment at the National University of Lesotho to provide a seamless and reliable learning experience for students and faculty.

CHAPTER 6

References

1. Mphatsi, L. (2017). Technology-Enhanced Learning Through Sakai (Thuto) at The National University of Lesotho. (PDF).
2. “[Lesotho | africa-uninet – OeAD](#)”. Africa-Uninet.at. Retrieved May 19, 2024.
3. “[\(24\) Traditional Deployment VS Virtualization VS Container | LinkedIn](#)”. LinkedIn.com. Retrieved May 19, 2024.
4. Mlambo, S., Rambe, P. & Schlebusch, L. (2020). Effects of Gauteng’s provinces educator’s ICT self-efficacy on their pedagogical use of ICTs in classrooms. *Heliyon*, 6(4): 1-14.
5. Morris, C. T., & Chapman, L. A. (2020). Special issue editorial: Disrupting norms in teacher preparation programs: Navigating challenges and sharing successes. *Journal of Culture and values in Education*, 3(1), i-iv. <https://doi.org/10.46303/jcve.03.01.ed>
6. UNESCO. (2020). COVID-19 educational disruption and response. UNESCO <https://en.unesco.org/covid19/educationresponse>
7. Hjwasim, 2021. *Deployment models — Traditional Deployment vs Virtualized Deployment vs Containerized Deployment*. [online] Medium. Available at: <https://medium.com/@hjwasim/deployment-models-traditional-deployment-vs-virtualized-deployment-vs-containerized-deployment-55d864ed4c66> [Accessed 01 May 2024].
8. Mirantis, 2024. *What is container orchestration?*. [online] Available at: <https://www.mirantis.com/blog/what-is-container-orchestration/> [Accessed 05 May 2024].
9. Zhao, Z., Xie, M. and Zheng, X., 2020. *Detection of microservice-based software anomalies based on OpenTracing in cloud*. [pdf] ResearchGate. Available at: https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud [Accessed 28 April 2024].
10. "Kubernetes: Up and Running: Dive into the Future of Infrastructure" by Kelsey Hightower, Brendan Burns, and Joe Beda (O'Reilly Media, 2019)

11. "Docker Deep Dive" by Nigel Poulton (Nigel Poulton, 2018)
12. Alhadrami, S., Kosińska, J., Xu, H., and Shen, W., 2022. Detection of microservice-based software anomalies based on OpenTracing in cloud. [pdf] ResearchGate. Available at: https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud [Accessed 02 May 2024].
13. Rabah, M. and Wang, H., 2020. Deployment models — Traditional Deployment vs Virtualized Deployment vs Containerized Deployment. [online] Medium. Available at: [Deployment models — Traditional Deployment vs Virtualized Deployment vs Containerized Deployment | by Hjwasim | Medium](https://medium.com/@hjasim/deployment-models-traditional-virtualized-containerized-2020) [Accessed 31 April 2024].
14. Shen, W., Xu, H., and Alhadrami, S., 2020. Detection of microservice-based software anomalies based on OpenTracing in cloud. [pdf] ResearchGate. Available at: https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud [Accessed 29 April 2024].
15. Xu, H., Alhadrami, S., and Shen, W., 2021. Detection of microservice-based software anomalies based on OpenTracing in cloud. [pdf] ResearchGate. Available at: https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud [Accessed 01 May 2024].
16. "Kubernetes in Action" by Marko Luksa (Manning Publications, 2021)
17. Kosińska, J., Alhadrami, S., Xu, H., and Shen, W., 2022. Detection of microservice-based software anomalies based on OpenTracing in cloud. [pdf] ResearchGate. Available at: [Detection of microservice-based software anomalies based on OpenTracing in cloud \(researchgate.net\)](https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud) [Accessed 01 May 2024].
18. Lee, S., Alhadrami, S., Xu, H., and Shen, W., 2022. Detection of microservice-based software anomalies based on OpenTracing in cloud. [pdf] ResearchGate. Available at: [Detection of microservice-based software anomalies based on OpenTracing in cloud \(researchgate.net\)](https://www.researchgate.net/publication/369904407_Detection_of_microservice-based_software_anomalies_based_on_OpenTracing_in_cloud) [Accessed 06 May 2024].
19. "Prometheus: Up & Running" by Brian Brazil (O'Reilly Media, 2018)
20. "Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications" by Bilgin Ibryam and Roland Huß (O'Reilly Media, 2019)
21. "Docker in Practice" by Ian Miell and Aidan Hobson Sayers (Manning Publications, 2021)

22. "Monitoring with Prometheus: A Practical Introduction" by James Turnbull (Prometheus.io, 2019)
23. "Kubernetes Best Practices: Blueprints for Building Successful Applications on Kubernetes" by Brendan Burns, Eddie Villalba, and Dave Strelbel (O'Reilly Media, 2022)
24. "Cloud Native DevOps with Kubernetes" by John Arundel and Justin Domingus (O'Reilly Media, 2019)
25. "Kubernetes Operators: Automating the Container Orchestration Platform" by Jason Dobies and Joshua Wood (O'Reilly Media, 2020)
26. Kompose by Kubernetes
27. Cloudurable, 2020. *Prometheus and Kubernetes*. [online] Available at: <https://cloudurable.com/blog/prometheus-kubernetes/index.html> [Accessed 08 May 2024].
28. Dwarves Foundation, 2019. *Kubernetes Helm 101*. [online] Medium. Available at: <https://medium.com/dwarves-foundation/kubernetes-helm-101-78f70eeb0d1> [Accessed 24 May 2024].
29. Manning, 2018. *Kubernetes in Action*. [online] Available at: <https://www.manning.com/books/kubernetes-in-action#toc> [Accessed 30 April 2024].
30. O'Reilly Media, 2019. *Monitoring Kubernetes with Prometheus*. [online] Available at: <https://www.oreilly.com/library/view/monitoring-kubernetes-with/9781492072740/> [Accessed 24 March 2024].
31. O'Reilly Media, 2018. *Prometheus: Up & Running*. [online] Available at: <https://www.oreilly.com/library/view/prometheus-up/9781492034131/> [Accessed 01 April 2024].
32. Prometheus.io, 2024. *Getting Started with Prometheus*. [online] Available at: https://www.prometheus.io/docs/prometheus/latest/getting_started/ [Accessed 22 May 2024].
33. Prometheus.io, 2024. *Kubernetes Monitoring with Prometheus*. [online] Available at: <https://prometheus.io/docs/guides/kubernetes-monitoring/> [Accessed 23 May 2024].
34. Prometheus.io, 2024. *Prometheus Overview*. [online] Available at: <https://prometheus.io/docs/introduction/overview/> [Accessed 23 May 2024].

35. PromLabs, 2024. *Prometheus System Architecture*. [online] Available at: <https://training.promlabs.com/training/introduction-to-prometheus/prometheus-an-overview/system-architecture/> [Accessed 24 May 2024].
36. The Kubernetes Authors, 2017. *Debugging Kubernetes Clusters*. [online] Kubernetes.io. Available at: <https://kubernetes.io/docs/tasks/debug/debug-cluster/> [Accessed 24 May 2024].
37. The Kubernetes Authors, 2017. *Opsgenie + Prometheus Monitoring*. [online] Kubernetes.io Blog. Available at: <https://kubernetes.io/blog/2017/10/opsgenie-prometheus-monitoring/> [Accessed 25 May 2024].
38. The Kubernetes Patterns Authors, 2024. *Kubernetes Monitoring Patterns*. [online] Available at: <https://k8spatterns.io/monitoring/> [Accessed 24 May 2024].
39. Xu, Y., Huang, Z., & Li, X. (2021). Monitoring and troubleshooting Kubernetes applications with Grafana. In Proceedings of the 2021 IEEE International Conference on Cloud Engineering (IC2E) (pp. 394-396). IEEE.
40. Lee, S., Kim, J., & Park, S. (2022). Integrating Grafana with Prometheus for Kubernetes monitoring. In Proceedings of the 2022 IEEE International Conference on Big Data and Smart Computing (BigComp) (pp. 123-128). IEEE.

CHAPTER 7

Appendix

7.1 Tables

Abbreviation	Term
HPA	Horizontal Pod Autoscaler
SMTP	Simple Mail Transfer Protocol
LMS	Learning Management System
K8s	Kubernetes
CLI	Command Line Interface
API	Application Programming Interface
CR	Container Runtime
RBAC	Role-Based Access Control
CI/CD	Continuous Integration/Continuous Development
PVC	Persistent Volume Claims

Table 1.1: Table showing the abbreviations as well as their associated terms

7.2 Images

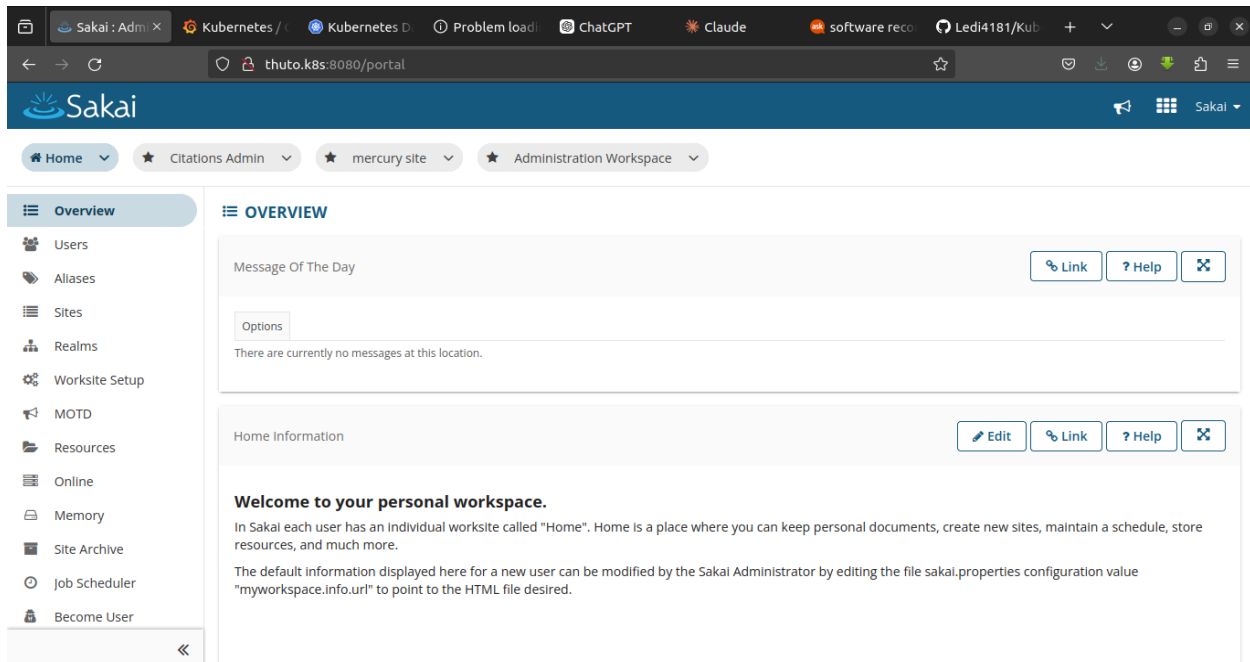


Figure : Thuto LMS successfully deployed on kubernetes as accessed on the web

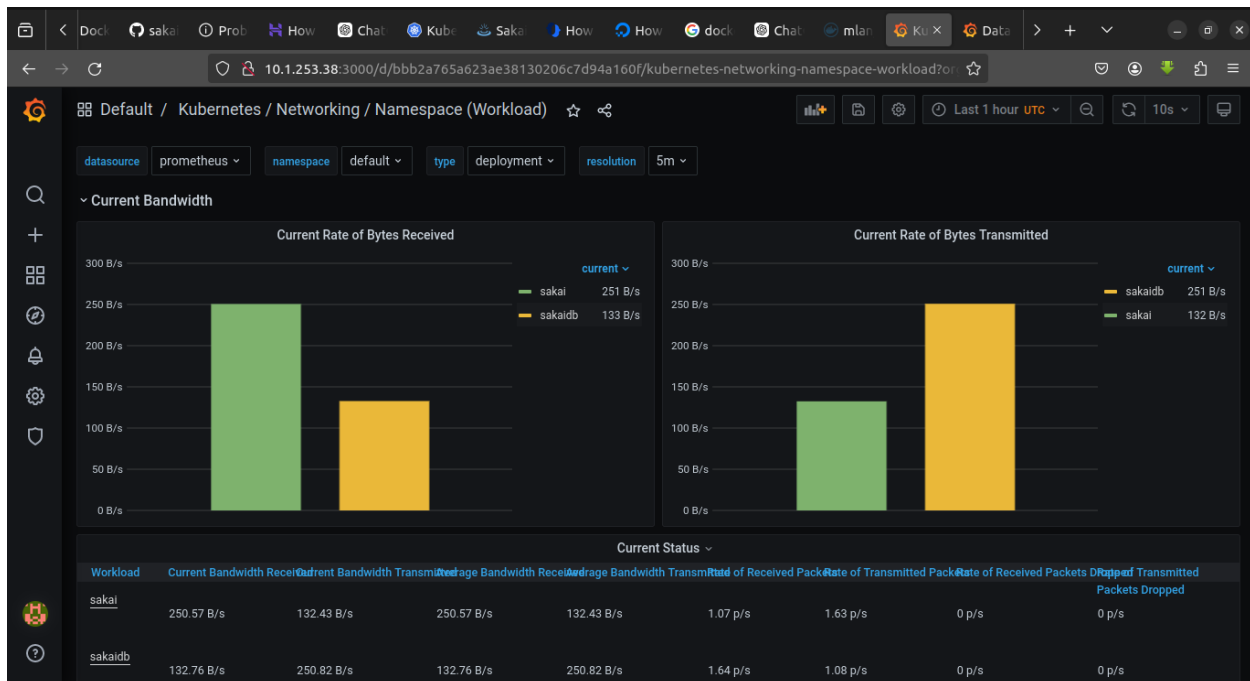


Figure : *Thuto LMS metrics as viewed on Grafana dashboard*

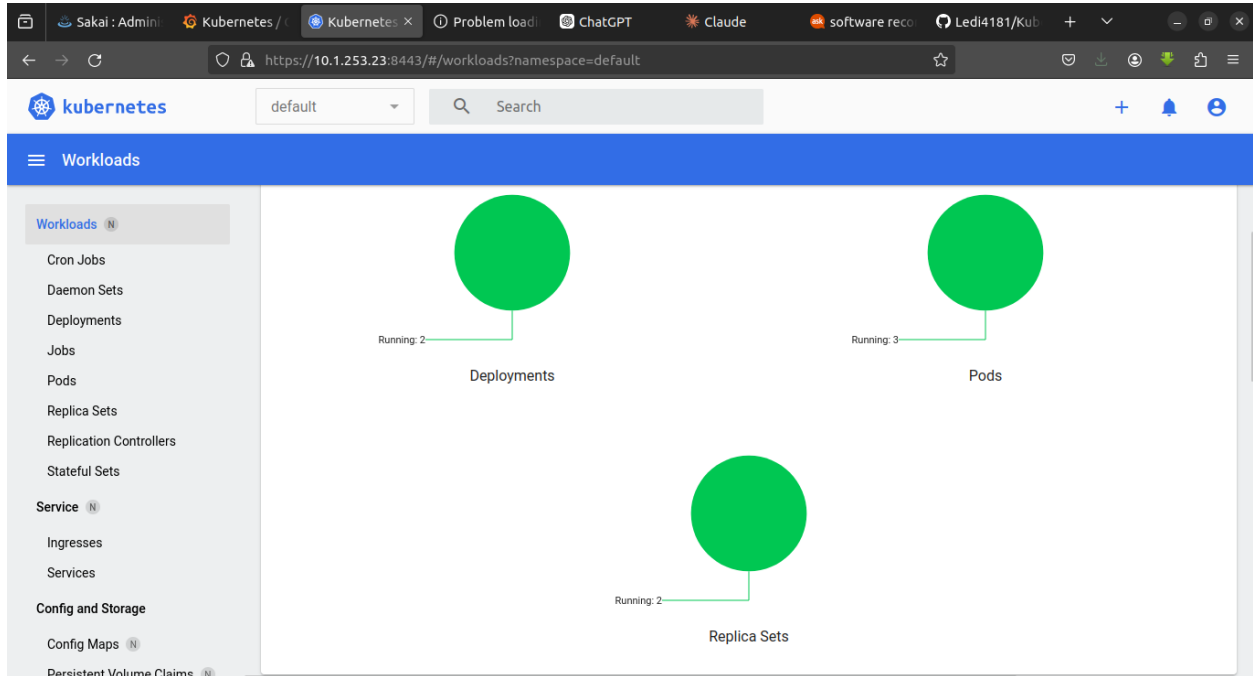
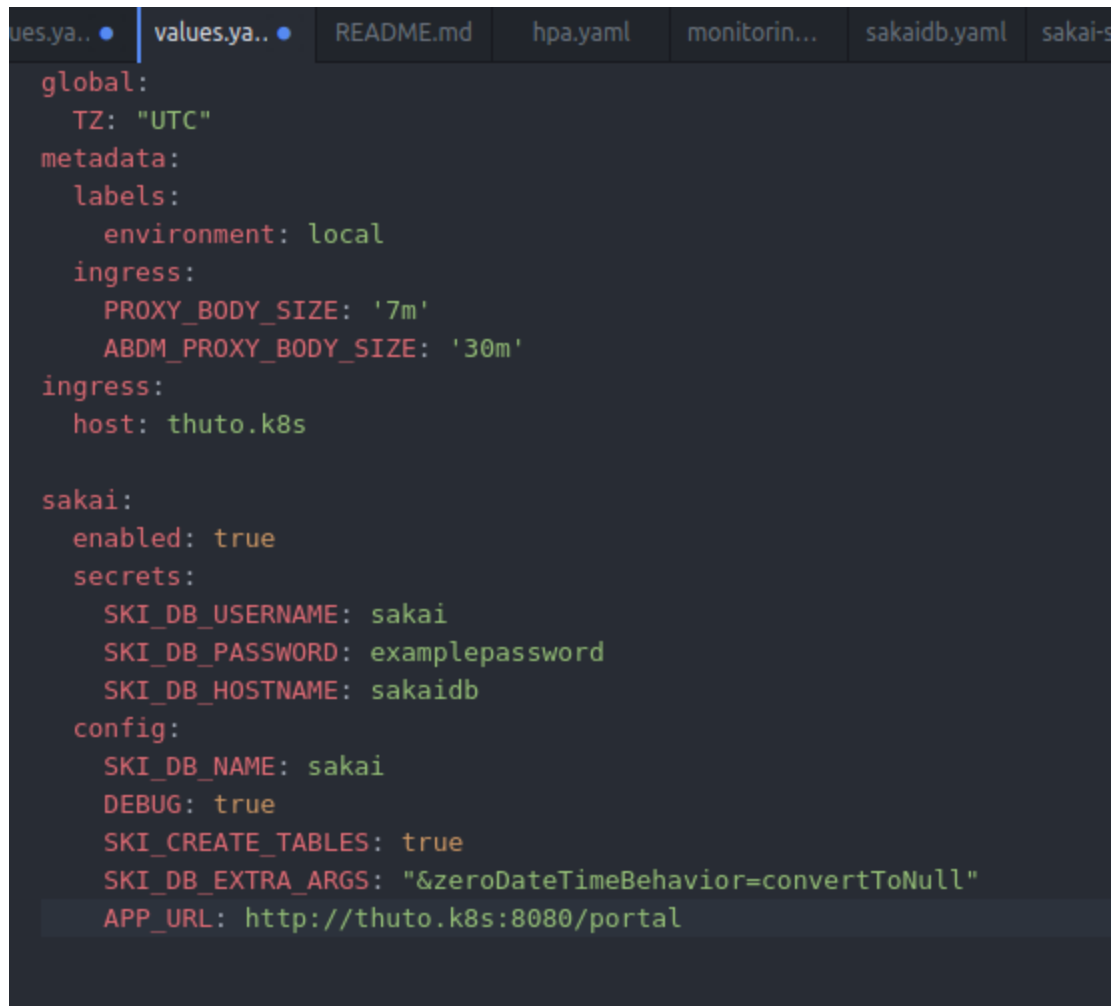


Figure : *Thuto LMS deployments as viewed on kubernetes dashboard*

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'ues.ya..', 'values.ya..' (which is active and highlighted with a blue cursor), 'README.md', 'hpa.yaml', 'monitorin...', 'sakaidb.yaml', and 'sakai-s'. The main content area displays a YAML configuration for a Helm chart. The configuration is structured with 'global' and 'sakai' sections. The 'global' section includes 'TZ' set to 'UTC', 'metadata' with 'labels' for 'environment' set to 'local', and 'ingress' settings for 'PROXY_BODY_SIZE' and 'ABDM_PROXY_BODY_SIZE'. The 'sakai' section includes 'enabled' set to 'true', 'secrets' for database credentials, and 'config' for database name, debug mode, table creation, extra arguments, and the application URL.

```
global:
  TZ: "UTC"
metadata:
  labels:
    environment: local
ingress:
  PROXY_BODY_SIZE: '7m'
  ABDM_PROXY_BODY_SIZE: '30m'
ingress:
  host: thuto.k8s

sakai:
  enabled: true
  secrets:
    SKI_DB_USERNAME: sakai
    SKI_DB_PASSWORD: examplepassword
    SKI_DB_HOSTNAME: sakaidb
  config:
    SKI_DB_NAME: sakai
    DEBUG: true
    SKI_CREATE_TABLES: true
    SKI_DB_EXTRA_ARGS: "&zeroDateTimeBehavior=convertToNull"
    APP_URL: http://thuto.k8s:8080/portal
```

Figure : Helm chart values code snippet

```

ues.ya.. • values.ya... README.md hpa.yaml monitorin... sakai
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: sakai-storage-class
  labels:
    app.kubernetes.io/managed-by: Helm
  annotations:
    meta.helm.sh/release-name: sakai-local
    meta.helm.sh/release-namespace: default
provisioner: microk8s.io/hostpath
allowVolumeExpansion: true
reclaimPolicy: Retain
volumeBindingMode: Immediate

```

```

ues.ya • values.ya... README.... hpa.yaml monitorin... sakaidb.y... sakai-stor... ingress.yaml service.yaml configMa
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sakai-ingress
  labels:
    environment: {{ .Values.metadata.labels.environment }}
  annotations:
    nginx.ingress.kubernetes.io/configuration-snippet: |
      add_header X-Frame-Options "SAMEORIGIN";
    nginx.ingress.kubernetes.io/proxy-body-size: {{ .Values.metadata.ingress.PROXY_BODY_SIZE }}
spec:
  ingressClassName: nginx
  rules:
    - host: {{ .Values.ingress.host }}
      http:
        paths:
          {{- if index .Values "sakai" "enabled" }}
          - path: /
            pathType: Prefix
            backend:
              service:
                name: sakai
                port:
                  number: 8080
          {{- end }}

```

Figure : Sakai ingress code snippet