Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Brandvold, Tom Arne Haugen
Damhaug, Alexander Kristian
Holhjem, Benjamin Knutsen
Lie, Kristoffer

Setup and operation of an e-learning platform

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Erik Hjelmås
May 2023

NTNU
Norwegian University of
Science and Technology

Brandvold, Tom Arne Haugen
Damhaug, Alexander Kristian
Holhjem, Benjamin Knutsen
Lie, Kristoffer

Setup and operation of an e-learning platform

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Erik Hjelmås
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Preface

This bachelor's thesis is written at the department of Information Security and Communication Technology at NTNU in Gjøvik by Tom Arne Haugen Brandvold, Alexander Kristian Damhaug, Benjamin Knutsen Holhjem and Kristoffer Lie.

We would like to thank our supervisor Erik Hjelmås for his support and guidance.

We would also like to thank Orange Business Services, and more specifically Truls Enstad for an exiting and challenging assignment, and his good collaboration.

Thanks to k6 for granting our request for a premium subscription of their service used for testing in the project.

Lastly we want to thank each other for a good and constructive collaboration over the last semester. This has been an educational and rewarding process. Sammen er vi dynamitt!

Abstract

Orange Business Services aims to deploy an e-learning platform utilizing Moodle for training their technicians, offering enhanced flexibility, scalability, and cost-effectiveness compared to conventional training methods. The project involves implementing different infrastructure architectures, ranging from a single server setup to advanced containerized Docker Swarm architecture. Furthermore, load testing is conducted to assess response times, while monitoring the systems load using the TICK-stack, with the ultimate goal of ensuring a robust, scalable, and secure e-learning platform.

Sammendrag

Orange Business Services har som mål å implementere en e-læringsplattform ved hjelp av Moodle for opplæring av teknikerne deres. Dette vil gi forbedret fleksibilitet, skalerbarhet og kostnadseffektivitet sammenlignet med konvensjonelle opplæringsmetoder. Prosjektet innebærer implementering av ulike infrastrukturer, fra en enkelt server til avansert containerisert Docker Swarm-arkitektur. Videre gjennomføres ytelsestesting for å vurdere responstider, samtidig som systembelastningen overvåkes ved hjelp av TICK-stack. Hovedmålet er å sikre en robust, skalerbar og sikker e-læringsplattform.

Contents

Figures

Tables

Code Listings

Acronyms

AI Artificial Intelligence. 26

IaaS Infrastructure as a Service. 32

IDS Intrusion Detection System. 17

KISS Keep It Simple Stupid. 26

LDAP Lightweight Directory Access Protocol. 3, 16, 21, 34

LMS Learning Management System. 3, 5, 14, 30, 44

MVP Minimum Viable Product. 54

NFS Network File System. 31, 55

NTNU Norwegian University of Science and Technology. 3

SSL Secure Sockets Layer. 16

TLS Transport Layer Security. 16

VPN Virtual Private Network. 3

Glossary

LaTeX Is a mark up language specially suited for scientific documents. 9

bash Bash is the shell, or command language interpreter, for the GNU operating system. 30

Cloud Infrastructure Is a term used to describe the components needed for cloud computing, which includes hardware, storage, and network resources . 5

Gantt-chart is a visual representation of a project schedule. 24

GlusterFS is a free opensource distributed file system software. 31, 43

HAProxy is a free opensource load balancing software. 43

kanban board is an agile project management methodology. 24

LAMP Stack of software components, specifically; Linux, Apache, MySQL and PHP. 30, 41

Moodle is a free online learning management system. 3, 5, 19–21, 29, 30

on-prem on-premises also known as on-prem refers to IT infrastructure hardware and software applications that are hosted on-site. 8

OpenStack tool for managing and provisioning infrastructure resources . 32

Overleaf is an online collaborative writing and publishing platform. 9

SkyHiGh NTNU's private cloud solution running OpenStack. 3, 32

Terraform is a vendor free infrastructure as code tool that lets you define both cloud and on-prem resources. 31, 32, 43

TICK-stack is a collection of open-source software tools developed by InfluxData for managing time-series data and monitoring. 4

Toggl is a time tracking and time management platform. 24

Trello Trello is a web-based project management tool that simplifies task organization and collaboration.. 24

Chapter 1

Introduction

This chapter provides an introduction to the project, outlining the background and problem area. It highlights the need for a dedicated e-learning platform for Orange Business Services and the advantages of using an e-learning platform over traditional training methods. The chapter also defines the problem statement and delimits the scope of the project. It presents the project goals and limitations and introduces the target audience and academic background of the project team. The purpose of the project and the structure of the report are also outlined in this chapter.

1.1 Background

Orange Business Services is experiencing rapid growth and sees an increasing need to use a dedicated e-learning platform to train technicians. To meet this need, Orange Business Services wants to use the learning platform Moodle, and the service and underlying platform should meet requirements for up-time, security, and capacity.

The e-learning platform will be designed to provide technicians with a comprehensive training program that will enable them to develop the necessary skills to support Orange Business Services' critical IT solutions. The platform will be used to provide online courses and interactive activities, quizzes, and assessments to measure the progress of the trainees.

In today's fast-paced business environment, traditional training methods can be time-consuming and costly. Employees often need to take time off work to attend training sessions, and the cost of travel, accommodation, and other expenses can quickly add up. Additionally, traditional training methods may not be able to keep up with the rapidly changing technology landscape, making it challenging to ensure that employees have the latest skills and knowledge required for their jobs.

An e-learning platform, on the other hand, offers several advantages over traditional training methods. Primarily, e-learning is accessible from anywhere, at any time, as long as there is an internet connection. This means that employees can learn at their own pace, at a time and place that is convenient for them, without needing to take time off work or travel to a training location. E-learning platforms are also highly scalable, allowing companies to train a large number of employees at once without the need for additional resources. The use of multimedia, such as videos, animations, and interactive activities, can also make learning more engaging and effective. The e-learning platforms can be updated quickly and easily to reflect changes in technology, regulations, or best practices.

This project is an exciting opportunity to provide Orange Business Services with a comprehensive and scalable training platform for their technicians. The platform will be built with a complete infrastructure, based on careful consideration of the appropriate technologies, and designed to meet the client's specific requirements. By providing a reliable e-learning platform, Orange Business Services can increase the efficiency and effectiveness of its workforce, resulting in improved outcomes for its customers. The use of an e-learning platform, such as Moodle, can provide Orange Business Services with a more cost-effective, scalable, and engaging training solution for their technicians.

1.1.1 Problem area

Setting up and operating an e-learning platform for Orange Business Services with a focus on meeting the company's requirements for up-time, security, and capacity while using Moodle as the training platform. The thesis will cover the complete infrastructure setup, including , servers, database, web server, LDAP, and necessary plugins. The solution should be secure, redundant, and scalable. The thesis will also cover testing of the different infrastructure architectures, redundancy and load balancing of web server and database. Additionally, the thesis will provide a justification for the chosen technologies and services. Finally, the thesis will document all configurations to ensure the solution can be deployed in a production environment.

Adopting a LMS for training delivery offers several advantages over traditional training methods [1]. Firstly, a LMS provides a centralized platform for managing and delivering training content to learners, which can enhance the consistency and quality of the training program. Secondly, a LMS can be customized to meet the specific needs of the organization and the learners, improving the effectiveness of the training program. Thirdly, a LMS provides accessibility and flexibility for learners, allowing them to access the training content anytime, anywhere, and on any device. This can make training more convenient and accessible for learners with busy schedules or those who work remotely. Fourthly, a LMS can increase the efficiency of training delivery and administration, saving time and effort for trainers and administrators. Finally, a LMS can generate data-driven insights that can help the organization measure the impact of the training program and make informed decisions about future training investments and improvements. Therefore, the implementation of a LMS can have significant benefits for Orange Business Services in terms of improving the quality, effectiveness, and efficiency of its training program.

1.1.2 Delimitation

Given the scope of the task assigned by Orange Business Services, it is essential to establish clear boundaries for the project. In line with the Keep It Simple Stupid (KISS) principle [2, p.595], the project will follow an iterative approach with a focus on developing small and simple components that can be easily revised as necessary. While Orange Business Services will be responsible for the logical and technical content of the Moodle platform, our team will facilitate its use by integrating the necessary plugins and additional software. To this end, our work will be conducted and tested on SkyHiGh, NTNU's on-premise cloud solution running OpenStack, which differs from the client's preferred cloud platform.

Access to SkyHiGh is limited to NTNU's internal network or through VPN for authorized users. External access is not available due to the solution being hosted on the internal network. In view of the fact that Orange Business Services does not require direct access to the solution, access restrictions will not be a significant

issue.

1.1.3 Project definition

The purpose of this project is to implement Moodle on a complete infrastructure platform with full resource access to manage the service. These are the requirements set by Orange Business Services in the project description:

1. There will be configured an infrastructure platform with reasonable choices of services and technologies.
2. The platform must be secured against unauthorized users, and it is expected to run security tests on common vulnerability continuously.
3. The web-server and database must be configured with redundancy and load-balancing to ensure up-time.
4. Simulated problems will also be tested, to make sure the system is able to be shielded from failure, or potential attacks.
5. TICK-stack will be configured to any operational issues that may appear.
6. Necessary operation tasks, and how it will affect the service's up-time will be documented.
7. Choice of services and technologies should be justified, and all configurations should be documented thoroughly, so the solution later can be used in a production environment.

These requirements can be effectively summarized by the following problem statement:

Develop and configure a complete Moodle infrastructure that ensures reliability, scalability and security using cloud technologies.

1.2 Target audience

1.2.1 Report

The target audience of this report can be segmented into two distinct groups: NTNU and Orange Business Services. NTNU will evaluate the report based on the quality of decision-making and the overall report quality. On the other hand, Orange Business Services serves as the intended client for the report, focusing on the technological findings, in-depth discussions, and recommendations outlined.

The report also caters to additional target audiences, including students in relevant fields of study and academic researchers. Our objective is to ensure that the research findings and recommendations presented in this report serve as valuable and practical resources for these audiences.

1.2.2 Technical documentation and code

The target audience for the technical documentation and code is primarily our client, Orange Business Services, who is particularly interested in a recommended infrastructure solution that can be deployed on their private cloud. Nevertheless, other interested parties, such as the students and faculty at NTNU, may also find the documentation and code to be of practical value. Additionally, any individuals who are seeking guidance on how to set up Moodle as a LMS may also benefit from the documentation and code, serving as a source of inspiration.

1.3 Academic background

This bachelor thesis is written by a team of four students in the final semester of their Bachelor's degree in Digital Infrastructure and Cybersecurity. The team members possess a shared passion for technology, with each individual exhibiting unique interests in various areas, including programming, networking, and infrastructure development. The team members have chosen slightly different elective courses that collectively enhance the application of comprehensive professional knowledge throughout the project.

The bachelor program comprises four courses per semester that cover programming, infrastructure, network, and cybersecurity. Most courses emphasize teamwork through group projects with a focus on work methodology, such as cybersecurity, teamwork, and software engineering. The focus on teamwork is what makes a project like this bachelor project possible. Notably, courses like "DCSG2003 - Robust and scalable services" and "IIKG3005 - Infrastructure as Code" have laid the foundation for building and configuring infrastructures.

While the study program covers most of the essential topics for this project, areas such as databases, statistics, testing, the Moodle platform, and academic report writing are areas that needs to be researched. Most topics are often covered superficially in courses due to the rapid advancements in the field. This project provides us with the opportunity to delve deeper into these theories and technologies.

1.4 Purpose

As mentioned in the background 1.1, Orange Business Services seek to integrate the learning platform Moodle onto their Cloud Infrastructure, to train their technicians. It is relevant to understand how the Moodle platform works as a Learning Management System, how to deploy it on existing infrastructure, how to model the infrastructure to fit the needs for running the platform, and the specific needs to run it efficiently.

Orange Business Services commissioned this project to conduct extensive research and evaluate alternative possibilities for implementing Moodle as a learning plat-

form. They want to use the results of this project to aid in taking decisions about the software and services to use, and how to configure the infrastructure in the best possible way. The solution of this project presents how to successfully configure a LMS on different infrastructure architectures and gives the client perspective and guidance on how to do so themselves. By doing their research through the work of a bachelor's project, Orange Business Services saves cost and resources, and the project gives graduate students relevant industry experience.

Due to the broad nature of the problem, it is possible to redefine the problem statement quite freely. To address the challenges posed by the tasks effectively, justifications for the chosen approaches must align with the established objectives. The problem statement introduced in section 1.1.3; *Develop and configure a complete Moodle infrastructure that ensures reliability, scalability and security using cloud technologies*, Is chosen for a number of reasons. While the original assignment contains most of the required components, some limitations have to be implemented due to the broad nature of the project. These issues are prioritized as essential for developing a functional solution for Orange Business Services.

Despite the extensive theoretical discussions and technological implementations involved in setting up the Moodle platform and achieving quality improvements, the project will not end with the completion of this stage. Rather, it will provide Orange Business Services with a solid foundation that they can build upon and further develop in the future. This project can also assist future graduate students, to further investigate the issues provided.

1.5 Project goals

Result goals

1. Setup a robust, scalable and secure infrastructure that is able to host Moodle for use as an internal learning platform for Orange Business Services.
2. Develop documentation at a quality that makes replicating the work easier for Orange Business Services when implementing the solution.

Business goals

3. Implementing our solution should greatly reduce the time and resources needed to train technicians.
4. The solution should reduce the resources needed to get the learning platform up and running for Orange Business Services.
5. The solution should be modifiable by Orange Business Services so that they can make the necessary changes to get the platform ready for implementation and use in their workplace.

Learning outcome

6. Take a deeper dive into setting up and configuring a robust and scalable infrastructure, and at the same time make sure that the infrastructure is secure.
7. Get broader knowledge with Moodle, web-hosting in general, Terraform, Tick Stack and other tools and services we will touch on during the project work.
8. Together with the technical learning outcomes we get real life experience with a bigger project, and the collaboration within a group and with the client.
9. Insights that will help us reflect on the work during the project, and to see potential improvements.

1.6 Project limitations

The project is limited by multiple factors, and while some limitations are beyond the group members control, like project deadline, others are set by the group. The project is limited by the goals set in the project plan (appendix E), time constraints and set deadlines, physical and practical constraints, project organization, and also the project definition listed in chapter 1.1.3.

Time constrains

1. Deadlines set by supervisor/client
2. Planning defined in our Gantt-chart
3. Internal deadlines set by the group
4. The project deadline is 22.05.23

Physical and practical constrains

5. The use of SkyHiGh vs. other cloud services
6. Client based in Oslo
7. Budget
8. Desired quality standard

Project organization

9. Group roles
10. Collaboration
11. Group routines

Time, physical, and practical constraints, as well as project organization, are frequently encountered limitations during the development of a bachelor's thesis.

These limitations can limit the project's scope, affect the level of detail and complexity achievable, and decrease the quality of the final product. Therefore, it is crucial for the project team to recognize these limitations early on and make necessary adjustments to ensure the successful completion of the project within the given constraints.

1.7 Roles

Supervisor Erik Hjelmås and client point of contact Truls Enstad play central roles in the bachelor project.

Erik Hjelmås is an Associate Professor at NTNU and has broad knowledge in the professional field. His recommendations and discussions at the weekly meetings ensure consistency and professionalism throughout the project.

Truls Enstad is a passionate Technical Trainer at Orange Business Services and is our point of contact with the client. He knows how Orange Business Services' on-prem cloud solution works. His contributions are ensuring that the work aligns with the client's interests. His experience is helping the team to gain a different perspective on the project, and to identify opportunities from a fresh point of view.

The project plan in appendix E explains the administrative background of this project more thoroughly, and explains risks, tools and roles in depth.

1.8 Report structure

1.8.1 Contents

The report is divided into the following chapters with several sub-chapters:

Introduction: This chapter is a short introduction to the thesis and the project.

Theory: This chapter introduces the theory necessary to know to be able to follow the rest of the thesis.

Requirements specification: This chapter includes an in-depth specification of the requirements of the project.

Working method: This chapter describes how we have worked with this thesis and project, and how this method has been applied throughout the project.

Implementation: This chapter goes through the practical implementation of code and how we have developed the different infrastructure architectures.

Discussion: This chapter includes explanation of all choices and goes through the results and alternatives found in the project.

Assignment critique: This chapter discusses our critique of the assignment.

Further work: This chapter identifies potential areas for future research and development that could build upon the current study and address unresolved questions or limitations.

Project evaluation: This chapter is an evaluation of the project.

Conclusion: This chapter is a conclusion of the thesis.

Appendices: This chapter includes all the appendices to our report. This includes our progress plan, all our technical documentation and code, test result summaries, project plan, contract, meeting minutes, and timesheet.

1.8.2 Report layout

The report is written in Overleaf using the mark up language LaTeX. We use a template developed by the Community of Practice in Computer Science Education at NTNU [3].

Chapter 2

Theory

The professional field surrounding this project is extensive and intricate, with frequent developments of new tools aimed at enhancing performance, security, and problem-solving. Throughout the project, we gain theoretical knowledge of various technologies, followed by practical application of one or multiple tools that use these technologies. This is providing us with a understanding of the professional field IT infrastructure [4] and system administration [5].

2.1 Background and professional field

2.1.1 Iron vs Cloud age

Iron age refers to how we used static systems and often worked directly on physical hardware with manual processes. The cloud age facilitates rapid access to virtualized resources via the internet, thereby enabling automated processes that reduce the likelihood of human error. These virtualized resources are as IBM states "the foundation of cloud computing" [6]. Software is used to create an abstraction layer of hardware resources that is made available through a cloud provider. This technological advancement allows for the development of new, faster processes that leverage the advantages of cloud computing. The comparison of iron vs cloud age are illustrated in table 2.1 from the book *Infrastructure as Code* by Kief Morris [7].

Iron Age	**Cloud Age**
Cost of change is high	Cost of change is low
Changes represent failure (changes must be “managed,” “controlled”)	Changes represent learning and improvement
Reduce opportunities to fail	Maximize speed of improvement
Deliver in large batches, test at the end	Deliver small changes, test continuously
Long release cycles	Short release cycles
Monolithic architectures (fewer, larger moving parts)	Micro-services architectures (more, smaller parts)
GUI-driven or physical configuration	Configuration as Code

Table 2.1: Iron vs Cloud Age

2.1.2 Deployment Options (Public, Private, Hybrid)

In brief, the public cloud utilizes services offered by external providers, while in a private cloud model, infrastructure must be built and maintained internally. A hybrid cloud is a combination of building internal infrastructure and utilizing external providers.

Public Cloud

When a business uses a public cloud service model, it "rents" existing resources from a service provider [8]. The customer owns the data itself, and the provider is responsible for preparing, delivering, and maintaining the application and resources. The user is not responsible for significant capital expenses, but instead pays a subscription fee proportional to the company’s usage. This model ensures scalability and focus on the company’s core operations.

Private Cloud

In the private cloud service model, the user connects to the cloud exclusively, often on a secure private network where only selected individuals have access to the services [8]. This model suits companies that require better security, control over resources, and avoids sharing resources with other cloud costumers. These features is often ensured by hosting the cloud model on-premise. The advantages of this service model can also be seen as disadvantages. Owning infrastructure for better security and control, entails greater capital expenses, operating expenses, and complexity.

Hybrid Cloud

Hybrid cloud is the combination of private and public cloud service models [8]. A well-designed hybrid cloud service model utilizes the advantages of the models above, resulting in: a seamless user experience, good flow of data between services, and compliance with legislation and service agreements. This is achieved by placing non-sensitive data in public cloud services, while sensitive and legally required data is stored securely in private cloud. The use of hybrid cloud will however increase complexity, and good design crucial to ensure these benefits.

2.1.3 Cloud service models

IaaS - Infrastructure as a Service

Infrastructure as a Service is the service model where the customer gains access to raw hardware resources on demand using a service provider such as Microsoft Azure or Amazon Web Services [9]. The user can easily provision and configure these resources as if they were on-premises hardware. The cloud service provider is responsible for hosting, managing and maintaining these hardware resources in their data centers.

PaaS - Platform as a Service

Platform as a Service is a service where the development and distribution of cloud-based applications is at the center. The cloud service provider is responsible for hosting, managing and maintaining the hardware and software included in the platform. This gives value for developers as they get a quickly and cost effective way to run and test applications in the cloud [8].

SaaS - Software as a Service

Software as a Service is service model that delivers a fully distributed cloud application that is updated and managed by the SaaS vendor [9]. The service minimizes the time customers spend on configuring and setting up the application while offering services that support the customers business goals. Services in this scope require internet connectivity and are often accessed through a web browser.

A user connects to the service without dealing with underlying infrastructure or configuration.

2.2 Topic theory

The use of an e-learning platform is a contemporary method adopted by Orange Business Services to provide training for their technicians. This platform offers technicians the flexibility to access course materials from any device, at any time. Moodle is the chosen platform that supports the clients needs in this regard. It is imperative to examine some essential theoretical considerations in the underlying infrastructure of the platform.

2.2.1 Redundancy and Load Balancing

In order to support a robust running service, it is necessary to understand redundancy and load balancing. Redundancy, referred as *Fault tolerance* by Schlossnagle [10, p.37], refers to the use of backup resources or systems to ensure that the e-learning platform remains operational even if there are hardware failures or other issues with the primary systems. Load balancing involves distributing the workload across multiple servers to ensure that no single server is overwhelmed and that the platform can handle high levels of traffic or user activity [10, p.61]. These theoretical aspects are achieved using various technologies.

2.2.2 Database and database clusters

A database [11] is information set up for easy access, management and updating. Computer databases store aggregations of data records and files that contain information produced in an environment. A Learning Management System such as Moodle, produce tremendous amounts of data once active in production. Moodle use the type relational databases, which are comprised of tables with unique predefined categories. Each column within a table has at least one data category, and rows with certain data instances that match a columns category specification. These tables, are indexed using SQL queries to retrieve, update, and delete data located in a table.

Database clustering [12] is the process of connecting more than one single database instance or server to your system with the same purpose. In a database cluster, it is usual that one or a group of servers are managing the cluster. The benefits of this structure are system redundancy, load balancing, and performance.

2.2.3 Scalability

In the context of e-learning platforms, a scalable service or system is of high importance to accommodate the growth of the user base and changing demands over time. The ability to scale vertically, by increasing resources like RAM, CPU

on the instance running the service, and horizontally, by distributing the load across multiple instances, is made possible by the advancements in cloud technology [10, p.5-6]. Horizontal scaling is the more modern approach and involves the use of load balancers to distribute the workload across multiple servers or instances. This results in increased availability, reliability, and performance of the e-learning platform, allowing it to handle high levels of traffic or user activity. However, it's crucial to note that proper planning and implementation of scalability strategies, along with ongoing monitoring and testing, are necessary to ensure optimal performance and to identify and address any issues that may arise.

2.2.4 Automation

IBM defines automation as the utilization of technology to execute tasks while minimizing human involvement [13]. The use of cloud computing and automation enhances productivity by enabling the delivery of greater value within reduced time frames [7]. When automation is effectively utilized, these technologies offer convenience, safety, efficiency, and accountability required to implement changes successfully. However, it is important to note that the benefits of utilizing cloud automation depend upon the proper implementation.

Without automation, systems can experience configuration drift, where similar systems end up with different configurations due to manual changes. Kief Morris suggests that this fear of configuration drift hinders the adoption of automation [7]. In essence, a lack of confidence in automating processes results in manual adjustments, ultimately leading to inconsistencies across servers. Therefore, it is crucial to build in automation right from the beginning of the development process. This will reduce the chance of human error and minimize variation in configuration.

2.2.5 Monitoring

Schlossnagle puts monitoring on the list of mission-critical environment in his book about Scalable Internet Architectures [10, p.25-30]. It is an essential aspect of maintaining a reliable and secure e-learning platform. Monitoring involves the continuous tracking of system performance, resource usage, and user activity to detect and address any issues that may arise. Monitoring can be achieved through various tools and techniques, including log analysis and network monitoring. Monitoring helps to ensure high availability, performance, and security of the Moodle service, thereby providing a positive user experience. Regular monitoring can also provide valuable insights into system usage patterns and help to identify opportunities for optimization and improvement.

2.2.6 Containerization

Containerization represents a paradigm wherein a cloud-native application is packed within a compact and executable entity. This entity encompasses the software code itself, alongside the essential operating system libraries and dependencies required for its proper execution [8]. The "lightweight" attribute of containers refers to their ability to share the host machine's underlying operating system, thereby avoids the need to allocate a separate operating system to each individual application. Consequently, this decreases the start-up time and enables the concurrent operation of multiple containers on a single host, whitch leads to reductions in server and licensing expenses.

Containerization gives developers the advantage of deploying their code across diverse host environments with ease. This inherent portability facilitates accelerated development processes, mitigates the risk of vendor lock-in associated with specific cloud providers, ensures fault isolation, simplifies management procedures, and enhances overall security [14].

2.2.7 Security

Security is fundamental in all modern web applications, and it is important to develop and deploy secure infrastructure to host the application. With the ever-increasing reliance on web-based services, ensuring the protection of sensitive data, user privacy, and system integrity is crucial.

Securing the architecture represents a vital element. Adhering to secure architecture principles, such as the principle of least privilege [15], is of great importance. Implementing stringent access controls, segregating components into isolated layers, and regularly updating and patching all software and frameworks are essential measures. Password management is important, and it is important to enforce robust password policies to minimize the risk of unauthorized access [16]. Administrator access to infrastructure and software should be limited to specific needs, and all services should be configured with strong passwords saved in a secure location.

The storage of sensitive data needs special attention, encrypting data both at rest and in transit is crucial [17]. Utilizing robust encryption algorithms and securely managing encryption keys are vital steps to ensure data integrity and confidentiality. Regular audits and reviews of access controls are necessary to maintain good security.

When communicating with services such as Lightweight Directory Access Protocol (LDAP), secure transport protocols such as Secure Sockets Layer (SSL)/Transport Layer Security (TLS) should be utilized [18]. Ensuring the encryption of communication channels and verifying the authenticity of server certificates is essential for secure communication.

Regularly backing up web application data and securely storing backups are crucial for disaster recovery [19]. Implementing encryption and access controls for backup files and regularly testing the restoration process are necessary to ensure data integrity.

Conducting regular security audits and penetration testing [20] aids in the identification of vulnerabilities and weaknesses in the web application and infrastructure. Promptly addressing any findings and keeping security measures up to date is crucial.

It is crucial to remember that security is a process that one never finishes. Regularly monitoring logs and implementing Intrusion Detection System (IDS) help identify any suspicious activities. Staying informed about emerging threats and adapting security measures accordingly is vital. Prioritizing security in web applications and their underlying infrastructure not only safeguards sensitive data and protects user privacy but also ensures the trust of users.

2.2.8 Open-source software

Open-source software, such as Moodle, TICK stack, Ubuntu, and Terraform, provides users with the liberty to modify, distribute, and copy its code, as they desire [21]. These features sets open-source software apart from closed-source software, where the code is proprietary, and the exclusive legal right to modify, copy, and inspect the code belongs to the original author or licensed users [22]. In most cases, open-source software is available free of charge due to the contributions of the community, in contrast to the fees associated with the use of proprietary software such as Canvas [23].

Chapter 3

Requirements specification

The primary objective of this project is to establish a fully operational infrastructure that can support a functional Moodle setup, configured according to the specifications provided by Orange Business Services that is listed in chapter 1.1.3. The project must ensure that the user experience is seamless, with an emphasis on service response time. Additionally, it is imperative that the developed product can be easily deployed for the client. The deployment process should be automated and in compliance with the best practices of the relevant professional field.

3.1 Functional

The functional requirement for the bachelor's project is to enable connectivity to a functional Moodle site over the internet in a testing environment. Upon the successful completion of the setup a web browser should display figure 3.1.

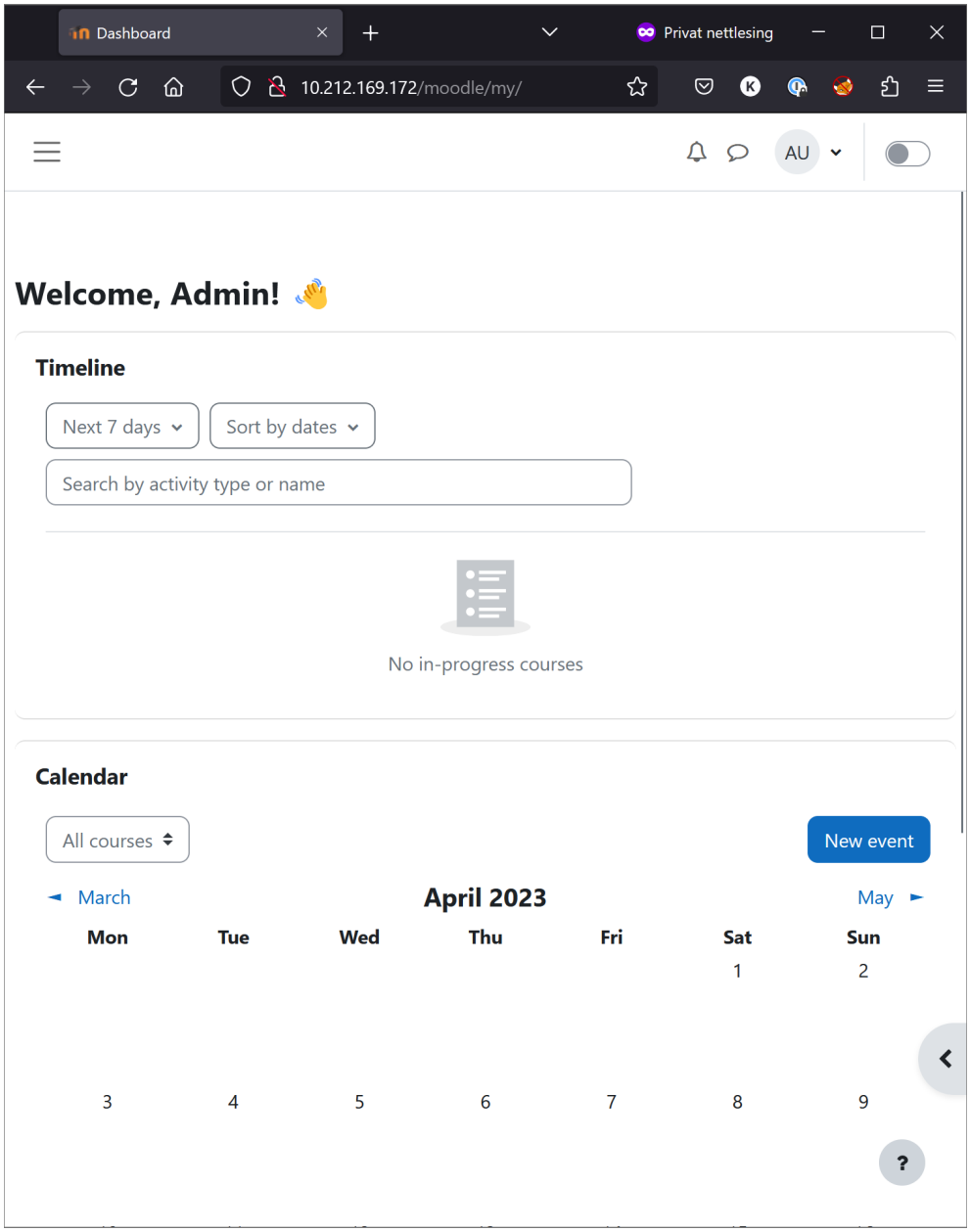


Figure 3.1: Moodle home page.

While the functional requirement specification is comprehensive, certain aspects are beyond the scope of this project, such as the setup of Moodle, except for minor plugins like Lightweight Directory Access Protocol (LDAP), and the testing- and installation processes.

3.2 Operational

The operational requirements for the project focus on the necessary components and considerations to effectively run Moodle. These requirements encompass response time, security, scalability, availability, and usability. The intention is to align with Moodle's documentation for the specific hardware and software requirements and adhere to best practices for operational functionality.

Response Time: The final product should provide an average response time that is comparable to other similar services, ensuring a smooth user experience.

Security: The service must guarantee the security of the hosted service on the company's internal network and its integration with their LDAP-server. It is essential to adhere to industry best practices for security.

Scalability and Availability: The service must be scalable and capable of accommodating at least 100 concurrent users. Additionally, it should be designed to ensure high availability to minimize downtime and maximize accessibility.

Usability: The usability of the final product is crucial. It should be user-friendly and intuitive, catering to the intended purpose of facilitating effective learning management.

Monitoring and Compliance: To ensure that the service meets all the requirements, it should be easily monitored using the TICK-stack monitoring system. This will allow for efficient tracking and assessment of the system's performance.

The hardware requirements for Moodle, as specified in Moodle's documentation [24], will be followed to set up and configure the foundational infrastructure. The focus will be on obtaining the necessary hardware resources through a cloud service provider like SkyHiGh (OpenStack) that offers access to virtual central processing units (vCPUs), virtual machines (VMs), random access memory (RAM), disk space, and other networking components.

The software requirements for hosting Moodle involve configuring specific software on each server instance. Moodle is hosted on the LAMP infrastructure, which includes Linux, Apache, MySQL, and PHP. The selected software should not only meet the project requirements but also ensure secure and efficient application performance. It is vital to use up-to-date software and avoid any potential security risks.

Chapter 4

Working method

This bachelor's thesis follows a structured working method that combines a flexible Kanban framework, effective time management practices, and efficient communication channels. The project is divided into distinct phases, including planning, research and development, improvements and documentation, and finalization. Various infrastructure architectures are explored using a combination of reliable resources and expert advice. Practical and experimental implementation are conducted to integrate technologies smoothly into the project.

4.1 Working method

The adopted work management framework involves utilizing a kanban board, specifically Trello. Kanban is chosen over other frameworks due to its flexibility, adaptability, visual workflow management, focus on continuous delivery, minimal overhead, reduced work in progress, and emphasis on continuous improvement [25]. It offers teams the ability to work at their own pace, visualize and manage workflows effectively, deliver value continuously, and make incremental improvements. The choice of this framework is important to enable the tracking of task progress and to simplify task delegation to team members.

Time management is accomplished through the utilization of a Gantt-chart, which is prepared during the planning phase in appendix E. Actual time usage is measured and analyzed using the time-tracking software Toggl, based on a recommendation from our supervisor.

The project is structured into four phases:

1. Planning
2. Research, development and testing
3. Improvements and documentation
4. Writing report and finishing touches

The planning phase is structured around the project-plan and contains extensive discussions about the project's operational framework. The planning phase is a critical component of any project as it lays the foundation for the project's success. This phase involves establishing the project's goals, objectives, and deliverables, and defining the strategies and tactics required to achieve them.

The second phase of the project involves research, development, and testing. The research part is focused on investigating the required and desired technologies, like Terraform, Moodle and monitoring with TICK-stack. Development is the next part of the phase in which the findings from the research are used to develop solutions. Lastly, the solutions are tested to ensure it meets the desired quality standards.

The third phase is dedicated to implementing improvements and perfecting the solution, while also ensuring that everything is adequately documented. This phase is a crucial part of the project as it focuses on ensuring that the project's solution is optimized and adequately documented. This phase involves implementing changes and improvements to the project solution, based on feedback from Orange.

The final phase involves the creation of a comprehensive report and performing finishing touches. This stage is crucial for ensuring that the quality of the report and accompanying documentation that explains the implementations and results in detail.

4.1.1 Communication and meetings

The project process involves consistent communication and planning, primarily facilitated through a private Discord-server, with team members engaging in both digital and physical discussions regularly. A dedicated Discord-server for the project makes it easier to communicate and collaborate, and all information related to the project is kept in one place. The Discord-server is chosen because it is the group members preferred method for digital communication in larger projects, and the benefits over other communication platforms greatly improves the time it takes to find information. With a solution based primarily on text chat, like Facebook Messenger, it is not as easy to share documents and hold digital meetings or working sessions.

The project process also consists of weekly group and supervisor meetings, along with client meetings every other week. These meetings are essential components of the process, and ensures that the project is on track, and meeting the supervisor's expectations and the client's requirements. All meetings are also documented in meeting minutes, to make sure that the discussed topics are available to all team members and nothing is forgotten. Some meeting minutes is appended in appendix G.

Client meetings play a crucial role in the project's success, as they provide an opportunity for the project team to receive feedback from the client and ensure that their requirements are met. During these meetings, the project team can discuss progress updates, address any issues or concerns, and receive guidance from the client. The team can also use these meetings to provide the client with updates on the project's status and gather any additional information necessary to ensure that the project's outcomes meet the client's expectations.

Continuous client feedback enables the project team to make any necessary adjustments to the project and ensure that the project work is continuously perfected to meet client requirements. Through regular client meetings and open communication, the project team can establish a strong working relationship with the client, which can contribute to the project's overall success.

4.2 Application of working method

The table 4.1 presents a brief outline of the architectures planned to develop in the project, while the actual implementation details and guidelines is explained in chapters 5 and 6. A range of infrastructure architectures are determined in collaboration with the supervisor, progressing in chronological order from the simplest to the most advanced. Each infrastructure architecture presents benefits over the others:

Architecture	Advantage	Disadvantage
Single instance	Easy to set up and test	Not redundant, low performance
3-layer	Separate services	"Old fashioned"
Docker	Disposable containers	Could be more scalable
Docker Swarm	Container cluster	Relatively complex

Table 4.1: Infrastructure architectures

The infrastructure architectures are systematically reviewed in chronological order, all of which are running the Moodle platform as a service. Throughout the development process, a variety of resources such as textbooks, literature, Google searches, YouTube videos, and ChatGPT are utilized. These resources serve as vital tools for knowledge acquisition and problem-solving.

Text books and articles offer comprehensive information on specific topics, while Google searches enable efficient identification of relevant online resources, tutorials, and best practices. Similarly, YouTube videos provide practical demonstrations of complex technical processes, aiding in their comprehension.

Furthermore, ChatGPT is an Artificial Intelligence (AI) tool [26] that responds to input it receives from the user, and returns a response to imitate a conversation. Unlike search engines, the ChatGPT algorithm, takes a user input and returns a response that is the most statistically accurate in the current data set. The benefits in a project, is it can quickly provide information in any field of area. For example queries about resolving technical issues, or recommendation of tools or technologies. It can be used as an assistance tool, to help the user to rephrase poor language, and improve the quality of what's originally written. The group understand the benefits of ChatGPT, but are closely aware of the flaws the technology contains. ChatGPT is never used as a reference for scientifically accurate information, as it often writes factually incorrect answers. In summary, the incorporation of these resources can significantly enhance the development process, by providing a wealth of information and expertise to address technical challenges and ensure that the project meets its objectives.

The typical infrastructure development process begins with researching the most suitable tools, how-to instructions, and best practices, followed by the actual implementation. The outcome is aligned with the KISS principle emphasizing the importance of learning by doing (or failing). This approach results in a significant amount of troubleshooting being resolved through communication with technical experts within and beyond NTNU, as well as through discussions during internal group meetings.

4.2.1 Practical and experimental implementation

Practical implementation refers to the final working result of experimental implementations. Practical implementation in this context entail well documented and easy to follow recipes, with well defined prerequisites, and simple understandable code. The Implementation chapter will introduce all of the working dependencies, infrastructure models, tools and technologies used in this project.

Experimental implementation reefers to the working method during this assignment. An experiment is a system of scientific investigation, usually based on a design to be carried out under controlled conditions, that is intended to test a hypothesis and establish a causal relationship between independent and dependent variables [27]. In this project, existing documentation is being utilized to practically implement technologies as support systems within the architectures. The objective is to identify functional components that can be seamlessly integrated. By the use of trial and error methodology during implementation, it is possible to decide if a component can be integrated with the existing architecture or not.

Chapter 5

Implementation

The implementation chapter holds great significance in this bachelor's thesis, as the project primarily comprises of a practical assignment. As mentioned earlier, this project consist of researching potential architecture solutions, implementing these to fit the Moodle platform, and to implement functional and operational specifications. To reach these milestones, The workflow during this project involves experimental and practical implementation of the proposed solutions. Specifically, this entails configuring infrastructure using different technologies, and employing a trial and error approach to reach functional and stable solutions.

5.1 Technologies, tools and software used

Moodle

Moodle is a free online Learning Management System (LMS). It is a learning platform designed to provide educators, administrators and learners with a single robust, secure and integrated system to create personalized learning environments [28].

Linux

Linux, particularly Ubuntu as the distribution, serves as the primary operating system for deploying instances and services on the infrastructure. The group's extensive understanding of Linux's command line interface is a crucial aspect of this, but it is the comprehensive benefits of Linux as an operating system and its numerous advantageous features that make it the obvious choice for primary usage.

Bash

Bash is the shell, or command language interpreter, for the GNU operating system, the name is an acronym for the 'Bourne-Again SHell' [29]. It is built in the Ubuntu operating system, and is a powerful scripting language that opens thousand of possibilities to automating a sequence of commands, without needing to apply themselves [30].

Bash is utilized as the main command language due to its widespread use as the main script method for Linux operating systems or distributions like Ubuntu. It proves invaluable in automating software installation on virtual machine resources during startup when provisioned in Terraform. This approach allows for the semi-automation of services in the infrastructure. Terraform supports the merging of startup scripts as user data for machine resources, enabling more efficient provisioning.

LAMP

The LAMP Stack is a stack of four different software technologies. It is an acronym for Linux, Apache, MySQL, and PHP. It includes an operating system, Linux; a web server, Apache; a database, MySQL; and a versatile programming language, PHP. All components are available through the default Ubuntu software repositories and their combination is used for running web-services [31].

During the research phase, the LAMP stack emerged as the primary set of technologies encountered. This stack proved to be compatible for implementing a functional solution on a single instance, representing a significant achievement. The successful integration of the LAMP stack with the Moodle software marked the initial milestone in the project. Following a comprehensive evaluation of various infrastructure models, the decision was made to persist with the LAMP stack, owing to its extensive adoption within the Moodle platform [32].

MySQL

MySQL is a relational database management system (DBMS), used to structure data within a database. A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. It is developed to use a logical model with objects such as, databases, tables, view, columns and rows to offer a flexible programming environment [33].

MySQL database software is primarily utilized for storing and managing Moodle data and information. Moodle supports various types of MySQL databases such as MariaDB and PostgreSQL in addition to traditional MySQL software.

MariaDB-Cluster - Galera

MariaDB is a free and open-source database management system, and is one of the most widely used databases today, besides MySQL and Oracle [34].

Galera Cluster is a software package that allows users to set up and configure MariaDB and MySQL clusters. Galera clusters use synchronous replication. A replication process that allows change in any cluster node, and simultaneously update data on the remaining nodes, so no divergence can be carried out [35].

HAProxy

HAProxy is a free loadbalacning software that delivers a fast and reliable reverse-proxy offering high availability and proxying for TCP and HTTP-based applications. It is particularly suited for high traffic, and is the most commonly used load balancing software [36]. HAProxy is a software that evenly distributes traffic between components in a infrastructure.

While there are various types of load balancing tools available, the group decided to utilize HAProxy due to prior experience with the software, as well as it is one of the most commonly used load balancing software [36].

GlusterFS

The Gluster File System (GlusterFS) is a Network File System (NFS), and an open-source solution that offers a low cost, highly scalable distributed file system to meet the storage requirements of a range of environments. In GlusterFS, all entities are treated as volumes which are then assembled to form a specific file system setup [37]. This NFS technology is used particularly in architectures where the load is distributed horizontally over multiple instances.

Terraform

HashiCorp Terraform [38] is an infrastructure as code tool that provides the opportunity to define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. Terraform creates and manages

resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

At the beginning of the project, the team reached a consensus to utilize Terraform as the principal infrastructure as code tool for constructing the physical infrastructure necessary for the Moodle platform. This decision was made, because Terraform is a vendor-free technology that supports integration on virtually any cloud platform. Subsequently, a considerable amount of time was allocated to acquiring proficiency with the tool, as well as provisioning tangible infrastructure components, such as compute, storage, and networking.

Alternatives for Terraform are orchestration tools specifically OpenStack Heat. Heat [39] is the main project in the OpenStack Orchestration program, and it implements an orchestration engine to launch multiple cloud applications based on templates in the forms of text file that can be treated as code. This orchestration tool, could provide the same benefits to launch infrastructure stacks, the same way Terraform can. The difference is that Terraform is vendor neutral, and OpenStack Heat can only be used on infrastructure using the OpenStack software. This is why Terraform is the preferred technology used in this assignment.

SkyHiGh and OpenStack

SkyHiGh is NTNU's private on-prem cloud solution and is running the OpenStack software, which is a multi-tenant virtualization-platform. Through the OpenStack platform at NTNU students and faculty can setup the following resources: single instance machine with an operating system of the user's choice, or a full OpenStack project it is possible to create networks, routers and virtual machines based on specific needs [40]. OpenStack is therefore used as an Infrastructure as a Service (IaaS) tool for managing and provisioning infrastructure resources, and the software can control large pools of compute, storage, and networking resources throughout a data-center, all managed and provisioned through APIs with common authentication mechanisms [41].

Students and faculty at NTNU can access the SkyHiGh's infrastructure if virtualization resources are needed through the OpenStack software [40].This approach presents the opportunity to provision infrastructure resources in real-time through coding, making it the most suitable option for this bachelor's assignment. That is why it is the cloud service used in this project.

Docker

Docker is a orchestration software engine [42] that allow services and applications to be run in containers. With Docker installed on the host one can instantly run the container of the application or service without the need to install or download the software and applications used by the application. The container works by running an image that is built with the needed software, services and files needed for the container to perform its task. To run a container one will need to either download

or build an image. Since images can be downloaded and then run instantly in a container, this makes applications hosted in Docker containers highly portable and quick to deploy.

Docker Compose

Docker Compose is a tool that allow the use of a single YAML-file to define the required Docker containers, and resources to deploy the application or service [43]. This supports the use of declarative code which handles the underlying logic of how to achieve the desired state [7].

Docker Swarm

Docker Swarm allow several Docker host machines to connect and share their available resources to deploy the defined containers. The defined containers will be distributed among all the connected Docker hosts. Docker Swarm provide increased robustness by re-deploying containers if they disappear. When one of the hosts disappear, the defined containers on the specific host will be distributed between the physical hosts available [44].

Docker Registry

Similar to files in 5.1, docker images can also be committed, pushed and pulled in repositories. Instead of using a git repository, one can use a Docker Registry to host images for Docker [45]. Doing so allows the Docker images to be easily accessible across multiple instances or even across multiple networks. Registries can be either self hosted or one can use a public registry server. By using a public registry server, the images can be made publicly available. A commonly used public registry server is the "Docker Hub", which is the official registry server provided by Docker. The use of Docker registry allow the images to be easily accessible without the need for building the images.

TICK-stack

The TICK-stack comprises of Telegraf (T), InfluxDB (I), Chronograf (C), and Kapacitor (K), making it a popular open-source monitoring platform. Chakraborty and Kundan highlight that Chronograf and Kapacitor from the stack has been merged into InfluxDB 2.0 [46, p.133]. To gather information on a system/software, it is necessary to configure a scraper (Telegraf), a software tool or program that automatically extracts data, to collect and transfer it to InfluxDB 2.0, which manages the remaining operations.

k6

k6 is a web testing tool developed by Grafana [47], with wide usage in enterprise companies such as Amazon and Microsoft. The functionality aids in preventing failures and improving system reliability. k6 operates by writing tests in JavaScript, similarly to Postman, and presents various packages and modules that enhance testing procedures. Being an open-source tool, k6 facilitates productive

and straightforward testing processes that may be preformed locally or via cloud computing. Consequently, the results are easily exportable as JSON, CSV, or to a cloud API to enable comprehensive analysis.

OpenLDAP

OpenLDAP is an open-source implementation of Lightweight Directory Access Protocol (LDAP) used to store and manage user and group information in large-scale networks. It supports various authentication mechanisms, and includes robust access control features for defining fine-grained permissions. Its flexibility and power make it a popular choice for managing user and group information in many environments [48]. OpenLDAP is the authentication service used by Orange Business Services.

GIT

GIT [49] is an open source distributed version control system. It is a system used to store code developed, and to run services using the same code. The benefits of it being a version control, gives the opportunity for developers to change code independently, without damaging previous work. If something breaks, there is no issue to revert to a previous version. It is a distributed software, which means it operates on a local repository (a developers computer), and a remote repository, the main repository, stored on a central server. This solution provides flexibility between developers across the same project.

As an essential component for the project collaboration, particularly in development projects, the establishment of a shared repository was necessary. GitLab was chosen as the platform for this purpose, where the repository was created comprising with a main branch along with four distinct branches assigned to each group member. These individual branches granted each member unrestricted autonomy in their approach to project work. Consequently, this repository fostered a conducive environment for the exchange of technological advancements across various branches, thus facilitating an efficient workflow throughout the project.

5.1.1 Detailed description

Moodle

The installation of Moodle software involves a number of configuration and implementation steps to ensure full integration on the provided infrastructure. Initially, the software dependencies must be installed and pre-configured to enable the platform to run and be managed. It is also necessary to modify the

`max_input_vars`

variable in the configuration of the php libapache2 to 5000 [50]. The Moodle repository should then be cloned, and the appropriate stable branch version tracked. Appropriate permissions must be set for the directory that runs Moodle, as well

as the directory that locally saves data on the web-server. Afterwards, the Moodle database must be created, with an admin user possessing the necessary permissions. Finally, the appropriate Moodle configurations must be applied either through the GUI configuration or the CLI configuration script, with the pre-configured config.php file.

Although this is the simplest way to have a functional Moodle platform, it is not sufficient to ensure a reliable and secure platform. The Moodle platform is a highly complex and critical service once used in production, so important security measures must be implemented to ensure reliability in both functional and operational sections, to avoid unforeseen events [51].

GlusterFS on distributed architectures

The Moodle platform stores some information locally on the file system in a directory called moodledata [52], this includes the hash value of pictures, user sessions, together with other temp files, logs and user info. The reason for this is to improve performance optimization by retrieving this information locally on the file system, instead of retrieving it from the database. For this reason, NFS must be designed for distributed architectures with multiple web servers that share the same moodledata folder. Here we use GlusterFS as a popular NFS technology to resolve this issue, because it is a software recommended by the Moodle developers, and it is a familiar software to use by the group.

MariaDB - Galera

Upon increasing the complexity of the infrastructure, it is relevant to implement a database cluster. There are several different cluster models supported by Moodle, such as MySQL, MariaDB, and PostgreSQL. Through experimental implementation and research, the group reached a consensus to implement the MariaDB-based Galera Cluster. Alternative systems, such as CockroachDB, which is part of the course material is not supported by Moodle, and therefore not used.

5.2 Additional components

5.2.1 TI(CK)-Stack

An essential aspect of this project is to monitor the underlying infrastructure for operational disruptions using the TICK-stack. Monitoring is critical for efficient service operation and effective resource utilization. Historical resource utilization provides fundamental values for the architecture design and provides developers with quantitative data to utilize. Monitoring valuable data is essential to preemptively detecting anomalous behavior [46, p.8].

Use of TI-stack

In this project, the TI-stack is utilized to monitor the running e-learning platform. To achieve this, a separate InfluxDB server is required, which functions as the

database for storing and processing data. Additionally, InfluxDB offers a variety of methods for displaying processed data, including dashboard templates.

The monitoring documentation in appendix B uses dashboard templates supported by Telegraf plugins, which offer a variety of tools to aid in monitoring [53]. Specifically, the Linux Monitoring Template dashboard [54] is utilized to monitor crucial metrics of an operating system including but not limited to disk usage, system load, memory, and CPU utilization.

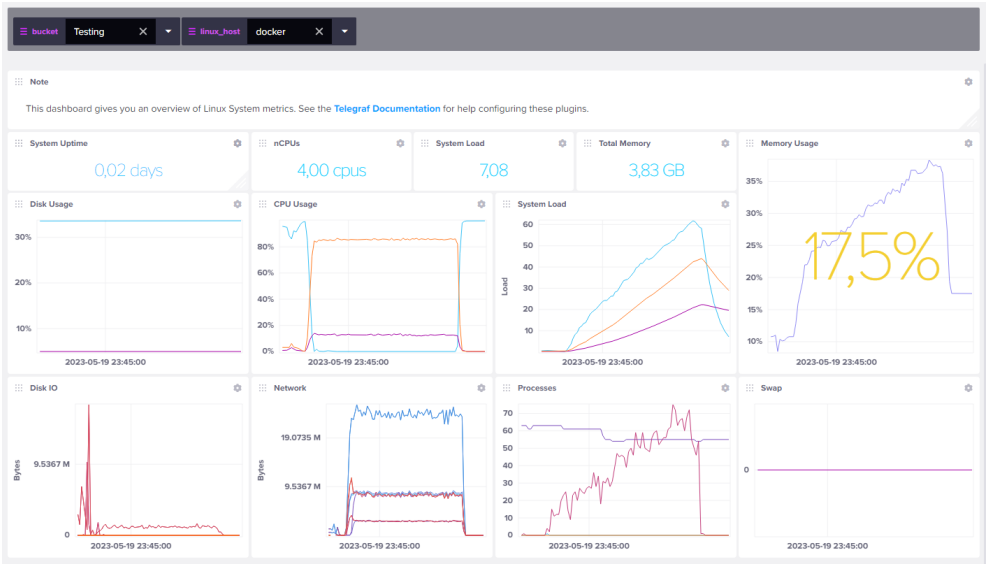


Figure 5.1: InfluxDB dashboard

To send data to the InfluxDB host, the utilization of Telegraf is employed. Telegraf is installed and configured on each server that require monitoring. The processed data can be viewed on the Linux monitoring dashboard 5.1. A predefined Telegraf configuration is implemented, making use of environment variables. The process outlined in the provided documentation offers a explanation of the steps and includes code listings for further guidance.

5.2.2 Testing

Postman response time test

In order to validate the response time of different architectures in a consecutive manner, Postman is used along with a simple JavaScript test script. The script checks the response time of the web service running on the architecture to ensure that they meet the requirements specified by the client. This approach helps to validate the performance of different architectures, and facilitates the selection of an architecture that provides optimal response time while satisfying the client's requirements.

Postman test

Code listing 5.1: Postman script

```
pm.test("Status␣code␣is␣200", function () {
   pm.response.to.have.status(200);
});

let responsTime = pm.response.responseTime;

console.log(responseTime);

pm.test(‘Response time is ${responseTime} ms
    and should be less than 5000 ms‘, function () {

   pm.expect(responseTime).to.be.below(5000);
});
```

Upon execution in Postman, script 5.1 evaluates two criteria for passing. The first condition checks that the status code of the HTML response is 200 - OK. The second criterion validates that the actual response time is less than 5000ms. A successful execution of the script is depicted in figure 5.2.

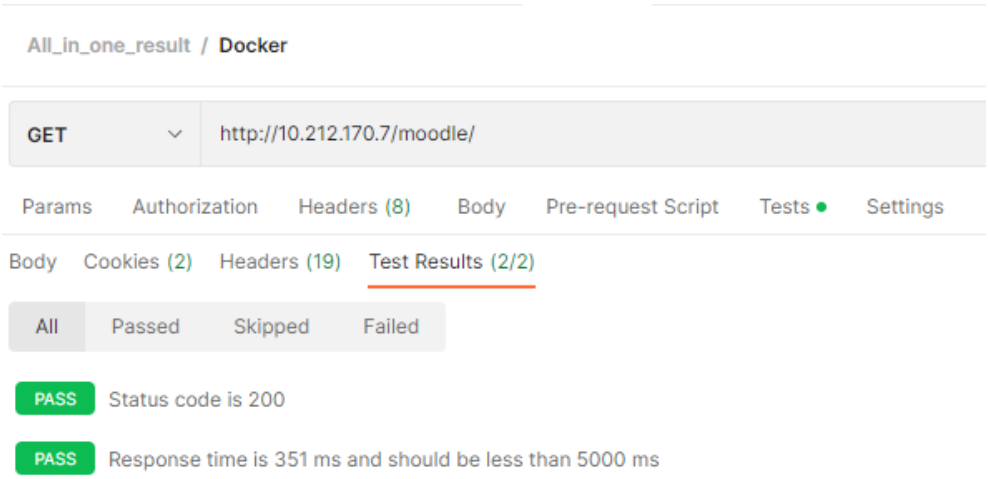


Figure 5.2: Single postman test

k6 testing

The k6 testing tool [47] can capture actions on web browsers, perform analyses, and generate test scripts that may be run on a local or cloud platform. Nonetheless, the free trial version of k6 is subject to limitations, such as a cap on virtual users, duration of run, and other parameters.

k6 has a diverse range of tests available and possesses an extensive community and documentation, which makes it accessible and user-friendly. K6 has a flexible and scalable architecture, which enables it to manage large-scale testing scenarios. Furthermore, K6 provides real-time analytics for test runners.

The script 5.2 was generated using the k6 browser extension, which captured a user navigating to the first Apache site of the Docker Swarm load balancer. The parameters used were default and not modified.

Simple k6 test script

Code listing 5.2: Test script k6

```
import { sleep, group } from 'k6'
import http from 'k6/http'

export const options = {
  ext: {
    loadimpact: {
      projectID: 3638175,
      // Test runs with the same name groups test runs together
      name: "Docker Swarm 100VUs"
    }
  },
  ext: {
    loadimpact: {
      distribution: { 'amazon:us:ashburn':
        { loadZone: 'amazon:us:ashburn', percent: 100 } },
      apm: [],
    },
  },
  thresholds: {},
  scenarios: {
    Scenario_1: {
      executor: 'ramping-vus',
      gracefulStop: '30s',
      stages: [
        { target: 20, duration: '1m' },
        { target: 20, duration: '3m30s' },
        { target: 0, duration: '1m' },
      ],
      gracefulRampDown: '30s',
      exec: 'scenario_1',
    },
  },
}

export function scenario_1() {
  let response

  group('page_1 - http://10.212.169.172/', function () {
    response = http.get('http://10.212.169.172/', {
      headers: {
        host: '10.212.169.172',
        accept:
          'text/html,application/xhtml+xml,application/xml;
                q=0.9,image/avif,image/webp,*/*;q=0.8', 'accept-language':
            'nb-NO,nb;q=0.9,no-NO;q=0.8,no;q=0.6,
```

```
                nn-NO;q=0.5,nn;q=0.4,en-US;q=0.3,en;q=0.1',
        'accept-encoding': 'gzip, deflate',
        dnt: '1',
        connection: 'keep-alive',
        'upgrade-insecure-requests': '1',
      },
    })
  })

  // Automatically added sleep
  sleep(1)
}
```

Script 5.2 executes a single scenario named "Scenario 1". The script gradually increases the number of virtual users up to 20 for one minute and maintains this level for three minutes and thirty seconds before decreasing the load to zero users for one minute. During the execution, the script utilizes HTTP GET requests to the Apache site of the Docker Swarm load balancer.

The test is executed from an Ubuntu instance that is running on SkyHiGh to guarantee connection with the tested site on NTNU's internal network. The instance is equipped with k6, and the test script provided above is copied into a file named "simpleHTTPtest.js." As illustrated in figure 5.3, the script is executed using the "-o cloud" parameter, instructing k6 to transmit the test results to k6 Cloud for analysis. As demonstrated in figure 5.4, it is easy to produce an executive summary of the test execution. The test report comprises the essential details of the test run, such as max throughput, HTTP failures and the average response time.

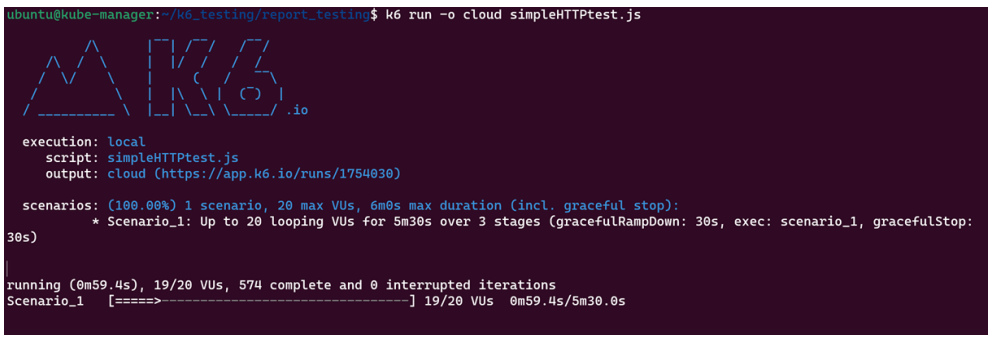


Figure 5.3: k6 simple run

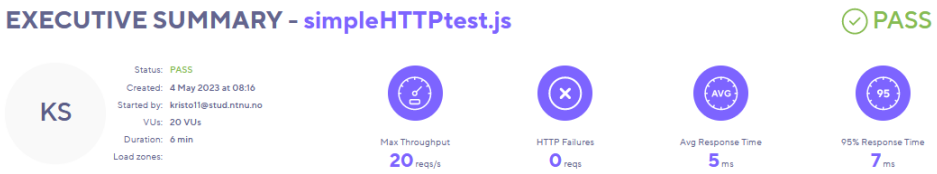


Figure 5.4: k6 executive summary

5.2.3 LDAP server

The setup of an LDAP server is elemental to this project, because that is the authentication solution in use at Orange Business Services. The setup and configuration of the LDAP server used in this project is simple, and only the bare minimum to be able to test LDAP as an authentication method for Moodle. This is because Orange Business Services already have their LDAP server configured and they will continue to use that. The setup of the LDAP server is detailed in appendix B, and details the installations of OpenLDAP, creating users, adjusting Moodle to use LDAP for authentication, and the scheduling of cron-jobs to make sure the user-database is up to date with the LDAP server.

5.2.4 Docker Registry server

To support the architectures where Docker is used, there is a Docker Registry server where images that will be used are pushed to. The server is made available to the rest of the network through a floating IP address in the network where the architectures are connected to. The server is set up following the documentation in appendix B.

5.3 Implementation of infrastructures and components

This section will cover how to implement the different infrastructure models, and their respective components to successfully implement the Moodle platform on a completely new environment. The following content will display context for the different technologies and terminology used cross-platform, and refer to the existing documentation and code attachments in this report. This section will also cover independent technologies, that operates on the following architectures, making them more reliable and predictable.

5.3.1 Single server architecture

This section will cover how to implement the Moodle platform more in depth on a LAMP stack. A LAMP stack is the simplest form of infrastructure resource configuration, and is the easiest way to run the Moodle platform on-premise. The figure 5.5, shows a visual representation of how this infrastructure model is built.

SkyHiGh is NTNU's cloud on-prem infrastructure platform, and all the physical resources are connected through the network called "*ntnu-internal*". The OpenStack project assigned contains a router that connects to a created private subnet. Lastly there is a single virtual machine that contains the necessary software dependencies, including the Moodle platform. The necessary software dependencies and technologies used, are defined in section 5.1.

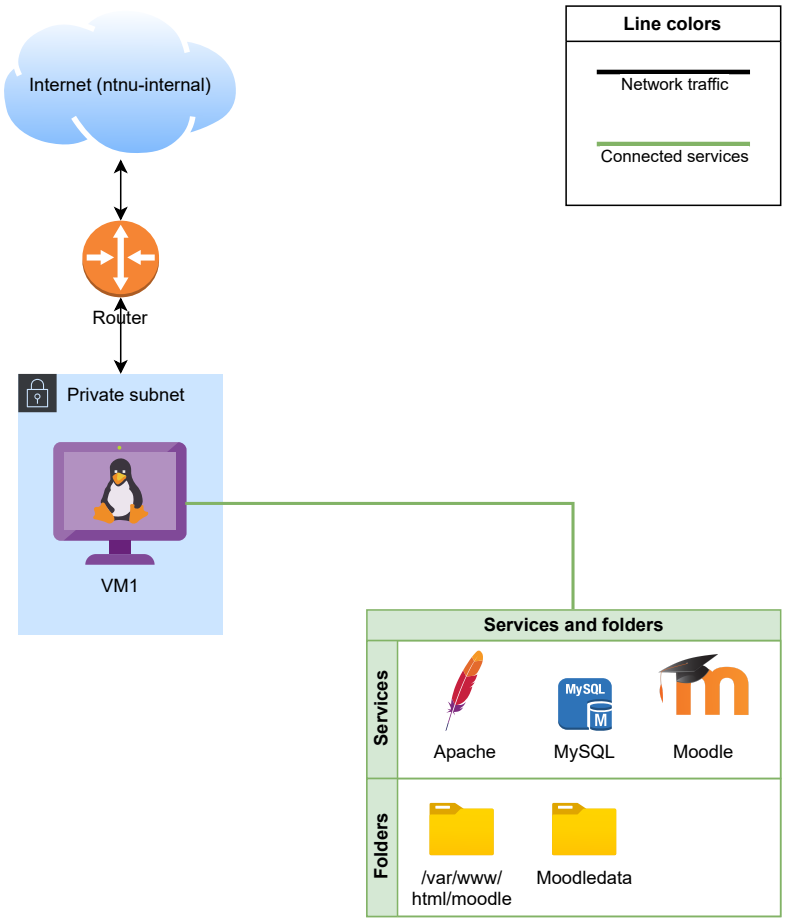


Figure 5.5: LAMP-Stack Moodle.

How to configure Moodle on a single instance

The appended documentation in appendix B is a comprehensive implementation manual, that offers a complete set of commands and guidelines to facilitate successful development of a LAMP-stack model running Moodle. The documentation covers further implementation steps after the needed code has been run. The following code is available in appendix C. The architecture is relatively straightforward, comprising of infrastructure resources such as; a single Ubuntu 22.04 virtual machine for compute, and a router for networking. This specific environment connects to the outside world (internet) via "*ntnu-internal*", which is a vendor-specific infrastructure network for this project. Finally, the Moodle platform and associated software are installed on the Ubuntu machine to enable a working Moodle platform.

To successfully follow the documentation, there are some prerequisite steps that must be pre-configured. The user should have a fresh Ubuntu instance with the latest image, equipped with a floating-IP and internet connectivity. This can be accomplished in an OpenStack environment as shown in appendix B.

Once implemented, the user will have a fully functioning and independent Moodle platform running on a single instance, which can be utilized as a working LMS. It is important to note that the dependent software and configuration are applied into a single instance, and therefore the solution lack redundancy, and scalability. Conclusively this is a solution that shouldn't be applied in a real production environment, it should be considered as a foundation for acquiring knowledge on how to establish a working Moodle platform.

5.3.2 3-Layer architecture

This section covers the initial setup and implementation of the Moodle platform similar to the Single Server architecture. The difference here, is that this architecture is evolved to feature a standard 3-layer architecture, and is a more sophisticated infrastructure model for integrating Moodle compared to the Single Server architecture. The figure 5.6 is a visual representation on the architecture, along with the dependent technologies and software used to build the infrastructure architecture.

The underlying network infrastructure is similar to the Single Server architecture, but the components are quite different in this solution. Instead of a single virtual machine doing the work of the whole service, the components are split into 5 physical components instead. This includes a load-balancer, at least two web-servers, and a database. The necessary software dependencies and technologies used, are defined in section 5.1 similar to the Single Server architecture.

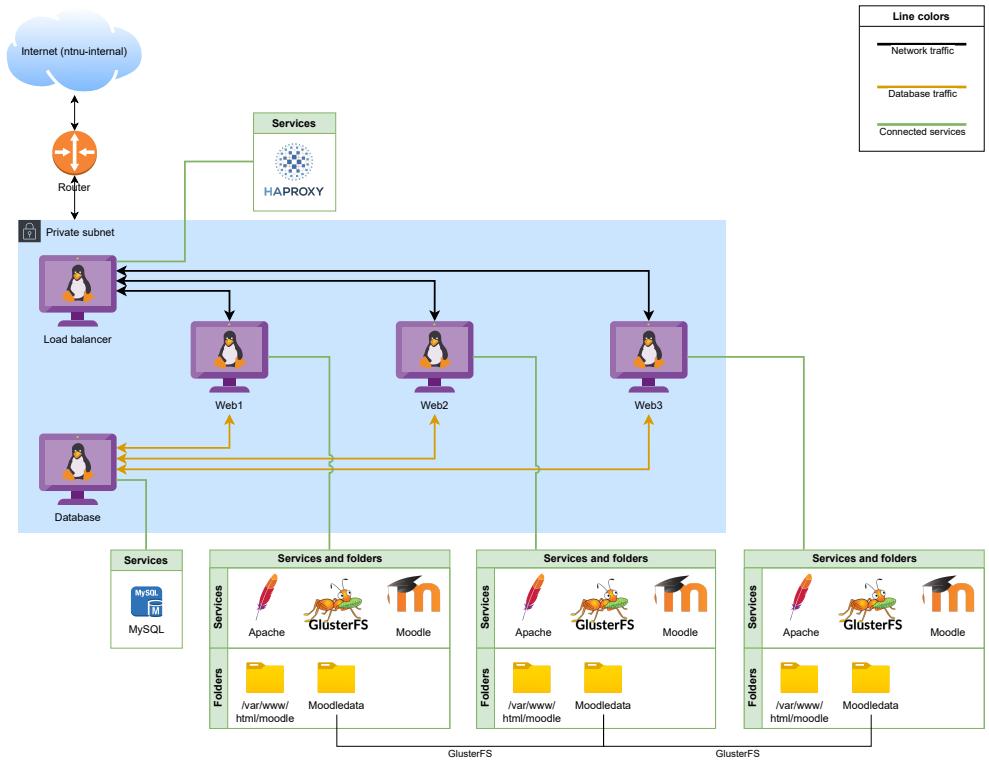


Figure 5.6: 3-layer architecture with Moodle.

How to configure Moodle on a 3-layer architecture

Similar to the LAMP-stack configuration, the appended documentation in appendix B provides specific details on how to successfully implement the Moodle-platform on a 3-layer architecture combined with the code in appendix C. This architecture has a greater complexity compared to the single server architecture explained in the previous section. The 3-layer architecture contains five Ubuntu 22.04 virtual machines that runs different services and are unique components in the infrastructure. Three of the machines are web-servers running the Moodle-platform and the Apache services, along with the PHP dependencies, managing the front-end and back-end operations. One machine is running the HAProxy software, used for load-balancing between the web servers, and the last machine in the architecture is the database that the web-servers communicate with. The networking is the same as the previous solution, and is a pre-configured component. In this architecture GlusterFS is introduced as a new logical component. It is used to mount a shared volume onto the Moodledata directory.

To configure the following architecture using the documentation provided, there are some prerequisite steps that needs to be implemented beforehand. These are defined in the documentation, and are used to run the architecture. The most important step is to have an Ubuntu machine with Terraform installed and con-

figured to provision the cloud resources on the cloud platform.

After implementation of the architecture, the user will have a functioning Moodle platform on a simple 3-layer architecture, which can be suitable for a small-scale Learning Management System (LMS). The architecture itself is a more redundant solution, as one of the web servers can turn off and the service will still be available. Although it's an improvement, there are still single point of failures in the infrastructure including the database and load-balancer.

5.3.3 Docker architecture

To effectively utilize Docker, certain prerequisites must be met. Initially, the installation of Docker software is essential on all servers intended for Docker usage. Additionally, the configuration involves the provision of a Docker image capable of executing the desired containers.

In certain scenarios, it is possible to utilize pre-existing images developed by external entities, obtained from the Docker Hub. Typically, the official Docker image corresponding to a specific software can be found there [55]. Alternatively, one may choose to construct the image independently. This can be accomplished by generating a Dockerfile, which specifies the characteristics and composition of the image. Subsequently, the Docker software can be used to build an image based on this file.

For Moodle to run in a container the following dependencies are needed:

- The Moodle Source code
- A database
- Apache
- Moodle config file

The Moodle source code can be obtained from the Moodle GitHub repository [56]. Additionally, Moodle requires configuration to function with a database. To set up the database, it is necessary to execute a script included in the Moodle source code. This script only needs to be run during the initial use of the database. Hence, it can be beneficial to create an image specifically designed to execute the database setup script and then automatically stop once the task is finished.

Furthermore, it is necessary to have an additional image that runs the web server service, such as Apache. This container is responsible for hosting the Moodle web service's source code and making it available to users. To achieve this, the image should install Apache, retrieve the Moodle source code and configuration file, and then launch the web server service. Pre-configuring the Moodle configuration file beforehand is important to avoid the need for manual configuration after the web service has been started.

Image	**Task**	**Duration**
Database setup	Run the Moodle install script	Until the script is completed
Moodle web server	Run Apache with the Moodle code	Until stopped

Table 5.1: Docker container images used with Moodle

Table 5.1 lists Docker container images used with Moodle, along with their corresponding tasks and durations. The images include "Database setup" which runs the Moodle install script, and "Moodle web server" which runs Apache with the Moodle code. The duration for the database setup is until the install script is completed, while the duration for the Moodle web server is until it is manually stopped. Database setup has to complete running before Moodle web server can be started.

The images used in the architecture is hosted in the registry server 5.2.4, making it easy and effective to run the required containers. The use of the registry server eliminate the need for building the images on the Docker host machine.

The commands and code needed to be able to replicate this implementation of Moodle on a Docker based architecture can be found in the appended documentation in appendix B. This documentation is a draft of Docker documentation represented in the public GitLab repository.

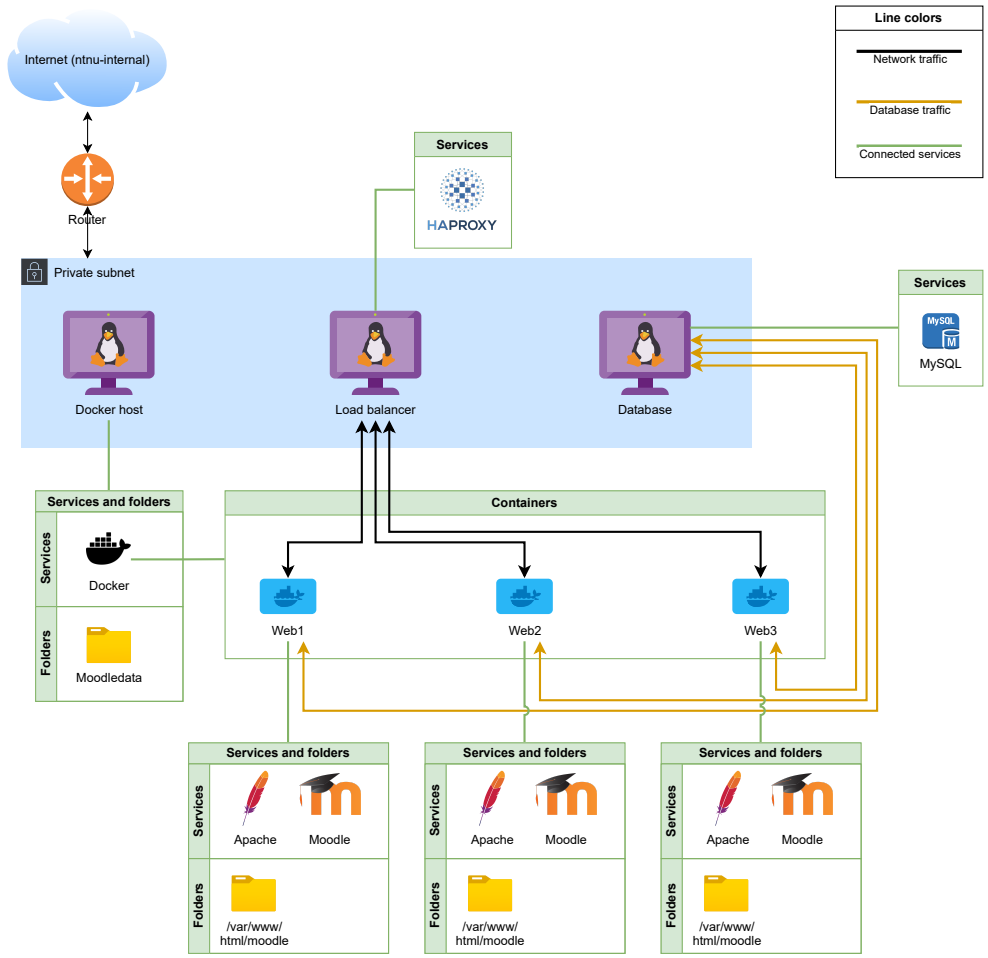


Figure 5.7: Docker with Moodle

Figure 5.7 illustrates the Docker-based architecture comprising a Docker host, a load balancer, and a database server. The Docker server is equipped with Docker and runs the container hosting the Moodle web service. Within the Docker server, three containers are operational, all utilizing the Moodle web service image. These containers establish connections to the database and are accessible externally through the load balancer.

The Docker architecture's implementation utilizes an image running Ubuntu, which subsequently installs Apache and other essential packages to facilitate Moodle's operation. The image also incorporates the Moodle source code. To obtain the necessary configuration file and initiate Apache, the image is using "init.sh" from listing 5.3 as its entry point.

Code listing 5.3: Web service Dockerfile

```
FROM ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive

RUN mkdir /moodledata
RUN chmod 777 /moodledata

RUN apt update
RUN apt install -y php-mbstring php-curl php-zip php-gd php-intl
RUN apt install -y apache2 libapache2-mod-php
RUN apt install -y php-xml php-mysql php-cli
RUN apt install -y git curl
RUN apt install -y git curl
RUN apt install -y php-ldap

RUN rm -rf /var/www/html/*
COPY moodle /var/www/html/moodle

RUN echo "max_input_vars = 5000" >> /etc/php/8.1/cli/php.ini

COPY docker_www/init.sh /
EXPOSE 80
ENTRYPOINT ["/init.sh"]
```

The Dockerimage in listing 5.3 will run the script in listing 5.4.

Code listing 5.4: Init.sh

```
#!/bin/bash -x
curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-api> >> \
/var/www/html/moodle/config.php

# Run crontab
/usr/bin/php /var/www/html/moodle/admin/cli/cron.php
echo "*/1 * * * * /usr/bin/php /var/www/html/moodle/admin/cli/cron.php \
>/dev/null" | crontab -u root -

/usr/sbin/apache2ctl -D FOREGROUND -k start
```

The script 5.4 use the curl command together with GitLab's API to download the latest version of the Moodle config file without needing to clone the whole git repository. This approach allow the admin to push a new version of the config.php file without needing to build a new image to use the new configuration. Every time a container with this image is started the most recent version of the configuration is used. The script's last line runs Apache.

To sum up, the Moodle platform will require two images to be able to run. One that has the purpose of running the database setup script, and one that has the purpose of hosting the web service.

5.3.4 Docker Swarm architecture

The Docker Swarm architecture is an extension to the Docker architecture where the Moodle web containers are distributed among several servers and connected using the Docker Swarm technology. In order to have Moodle work with this setup, it is also required to have a shared storage solution between the machines where the docker containers are running. This is accomplished using the network file system (NFS) GlusterFS. To further improve the robustness of the architecture, the database in this architecture is implemented in the form of a database cluster using the same virtual machines as Docker as its host.

For the database cluster to work together with the web-containers, there is a need for a load-balacer to allow the containers to communicate with the database. When using Docker, this can be solved by introducing an additional container for load-balancing. In this architecture the haproxy container from Docker Hub [57] is used as part of the Docker Swarm.

The swarm will be running on the three servers and will, like the Docker architecture, run three containers of the Moodle-web image. In addition to one container running the "haproxy" image. The web-containers are configured to go through the haproxy-container in order to reach the database.

To further improve the robustness of the Moodle service this architecture will use the built in load-balacing functionality of OpenStack. Like the load-balancer previously used, this setup will also balance between the three servers using the round-robin algorithm.

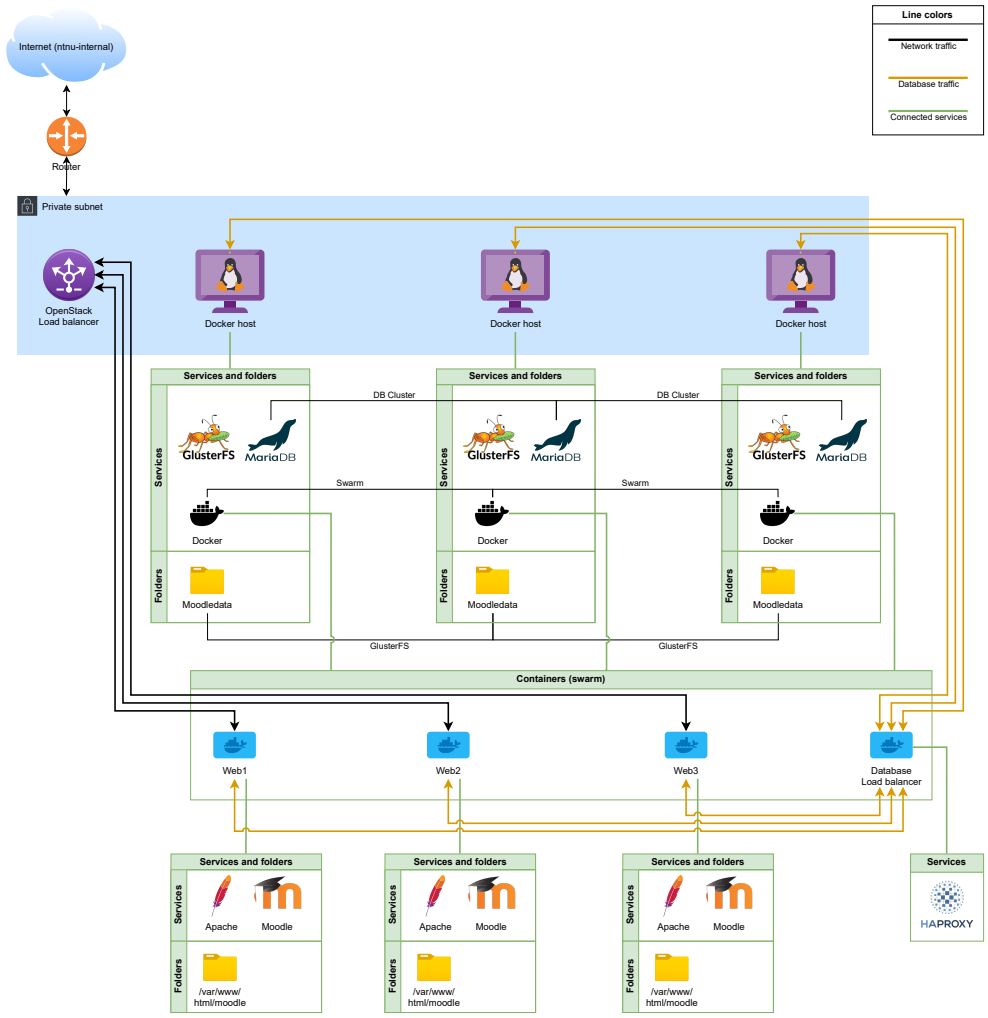


Figure 5.8: Docker swarm with Moodle.

As seen in figure 5.8, traffic from the outside trying to reach the Moodle service will first reach the OpenStack load-balancer and then be routed to the Moodle web containers. The container will then reach out to the HAProxy container, which will then further direct to one of the databases. This setup ensures that traffic to both the web server and the database is equally distributed among the hosts.

The figure 5.8 also show that, similar to the 3-layer architecture, the host machines have their databases connected in a database cluster as well as a shared storage using GlusterFS. To allow the Docker hosts to dynamically distribute the Docker containers the hosts are connected in a Docker Swarm. The "Containers (swarm)" box in the figure illustrate the resources provided by the Docker Swarm.

To set up the Docker Swarm architecture with Moodle it is first required to install Docker and then install and set up the database cluster. Then the GlusterFS NFS

need to be created with the needed folders for Moodle mounted. When the NFS and the database cluster is running the next step is to run the Docker container with a similar database startup script as the one used in chapter 5.3.3. After this script have been run, the Docker Swarm can be set up to allow the servers to work together as a swarm. When the swarm is ready, the Moodle web containers can be started and distributed among the servers in the swarm.

To begin with there will be three of these containers distributed across the three servers. The Moodle service is with this setup reachable through the floating IP address associated with the OpenStack load-balancer. The swarm containers are deployed using Docker Compose. The compose file in code listing 5.5 is used to deploy the web server containers.

Code listing 5.5: Docker compose file

```
version: '3.3'

configs:
  haproxy_config:
    file: ./haproxy.cfg

networks:
  moodle:
    attachable: true

services:
  web:
    image: 10.212.170.44:5000/moodle_www

    ports:
      - "80:80"

    networks:
        - moodle
    volumes:
        - type: bind
          source: /moodledata
          target: /moodledata
    environment:
        MOODLE_DB_HOST: db_balance

    deploy:
      replicas: 3

  db_balance:
    image: haproxy

    restart: always

    configs:
      - source: haproxy_config
        target: /usr/local/etc/haproxy/haproxy.cfg

    networks:
      - moodle

    ports:
```

```
      - "1936:1936"
```

In the code listing 5.5, the IP or URL of the registry server will be used when specifying the image. The HAProxy image is also deployed using the Compose file. For this image the IP is not specified because we in this case one should use the official HAProxy image from Docker Hub.

Upon successful implementation, the Docker Swarm architecture will be able to host a comprehensive Moodle service, while ensuring redundancy through a database cluster.

Chapter 6

Discussion

This chapter provides a comprehensive analysis and evaluation of the study's findings, aiming to shed light on the implications and significance of the research. This chapter delves into a comparison of various infrastructure architectures, and the strengths and weaknesses of each architecture are examined, considering factors such as robustness, scalability, and the presence of single points of failure. Additionally, the chapter presents the test results offering insights into the performance and responsiveness of the infrastructures under load.

6.1 Comparison of the infrastructure architectures

6.1.1 Single server architecture

The single server setup is mainly meant as a suggestion for a Minimum Viable Product (MVP) and does not have the characteristics of robustness and scalability required for this architecture to be recommended for use in a production environment. Although the architecture does have the components required to be able to run in production, it has several weak-points making it unfit for production. Since everything is running on a single server this means there is no opportunity for load balancing between several web servers. The database is also running on the same single host machine. With this architecture a single error in either Apache, MySQL or a general error in the single host machine will make Moodle instantly unreachable meaning that the host machine will be a single point of failure for Moodle. The failure could occur either as a result of a high load or from some of the services on the host machine crashing.

6.1.2 3-Layer architecture

The 3-Layer architecture improves some of the aspects discussed in 6.1.1. In this improved architecture the components are distributed over several host machines. The database is running on a dedicated machine in this architecture and Apache is running on three different web servers allowing for load balancing between them for the web traffic. This architecture also introduce a load balancing server as a new component. One challenge with this architecture is that both the load-balancer and the database still can be single points of failures. With the three web servers, this architecture can be made more robust by having the load-balancer only route traffic to the web server while it is responding. This means that if one or two of the web servers stop working it will still be possible to reach Moodle. However if the load-balancer stops working, Moodle will not be reachable from the internet because no route to the web servers will be found. If the database stop working, Moodle will stop functioning. Even though this architecture comes with some improvements from the single server architecture it is still having single points of failures that can cause problems in an production environment.

6.1.3 Docker architecture

The 3-layer architecture improves robustness, however there is also ways to improve on the scalability of the architectures. By moving Apache from a dedicated server to a Docker container, the architecture can be made more flexible. With a Docker image for the Moodle web server, the web server can be instantly started on a host with Docker installed and access to the Moodle web server image. In this architecture three web containers are running on a single Docker host machine. This means that if Apache stops working in one of the containers or the container itself stops working, Moodle can still be accessed on the two other containers. However

if Docker stops working or the Docker host machine stops working Moodle will stop working completely until the problem is resolved. The Docker architecture also comes with the disadvantages seen in the 3-layer and single server architectures, where the database and load-balancer are single points of failure. To sum up, the absence of either the database, the load-balancer or the Docker host will lead to Moodle being unavailable.

6.1.4 Docker Swarm architecture

In order to avoid any single point of failure there are several steps that can be taken. The Docker Swarm architecture seeks to solve these. By using several databases and connecting them in a database cluster the architecture can handle the loss of one of the databases. As seen in 6.1.3 having Docker containers running on one host machine only, leads to lack of robustness. By having several host machines and joining them together in a Docker Swarm, their resources can be shared, and the provision of containers are decentralized across the machines. With Docker Swarm the defined set of containers will always be available as long as one of the Docker hosts in the swarm is available. If one of the hosts disappear the other hosts will instantly spin up the missing containers.

To achieve robustness one will need to perform load-balancing between the machines. However, if load balancing is performed using a single machine, the loss of this machine will result in Moodle not being accessible. To solve this problem, the Docker Swarm architecture makes use of OpenStack's built in load-balancer. With this, the architecture have web servers, databases and load balancing without them acting as single points of failure. By using the swarm technology one can easily scale up horizontally by joining additional servers to the swarm and connecting them to the Network File System (NFS).

One could argue that the 3-layer architecture could be a good competitor to the Docker Swarm architecture if it also had implemented a database cluster and the OpenStack built in load-balancer. However there is a fundamental difference between them as Docker Swarm will take care of the load-balancing between the containers internally by itself. When scaling up the 3-layer architecture one would need to also take care of configuration of the load balancing as part of the process, however with the swarm architecture the Docker Swarm will take care of the load balancing itself when the additional Docker host have joined the swarm.

Unless the configuration of the load-balancer is modified, any additional Docker hosts that join the swarm after the initial setup will not act as an entry point for the swarm for traffic from the outside. However, they will still receive traffic from the internal Docker Swarm load balancing, making their resources available for the swarm. Adding hosts to the swarm will increase the capacity of the swarm. With increasing traffic it may also be necessary to include more hosts in the OpenStack load-balancer configuration to distribute the load related to handling and directing incoming traffic for the entry points of the swarm.

It is important to acknowledge that the act of scaling up the swarm does not automatically result in the scaling up of the database. In the event of a bottleneck within the database, it becomes necessary to scale up the corresponding database cluster. The incorporation of both exceptional robustness, achieved through the elimination of single points of failure, and remarkable scalability facilitated by the straightforward addition of supplementary machines to the swarm, renders this infrastructure highly appealing for utilization within a production environment.

6.2 Test results

In this thesis, the postman test described in chapter 5 is used to obtain an initial assessment of the infrastructure architectures and their response time. The data collected from these tests are utilized in the k6 load tester, which generates a more realistic load on the infrastructures.

All architectures are given the same standardized test course, consisting of 100MB of content, including 100 assignments, 1000 pages, and 1000 users. This is commonly referred to as the medium test course by Moodle [58] and satisfies the specified operational requirement of availability and scalability in chapter 3.

Initially, a test run with Postman was conducted with the Large test course with 1GB of content. However, after executing the Postman test, it was evident that the infrastructures did not handle that much data well, particularly in cases where the load was distributed across multiple servers that utilized GlusterFS as shown in figure 6.1.

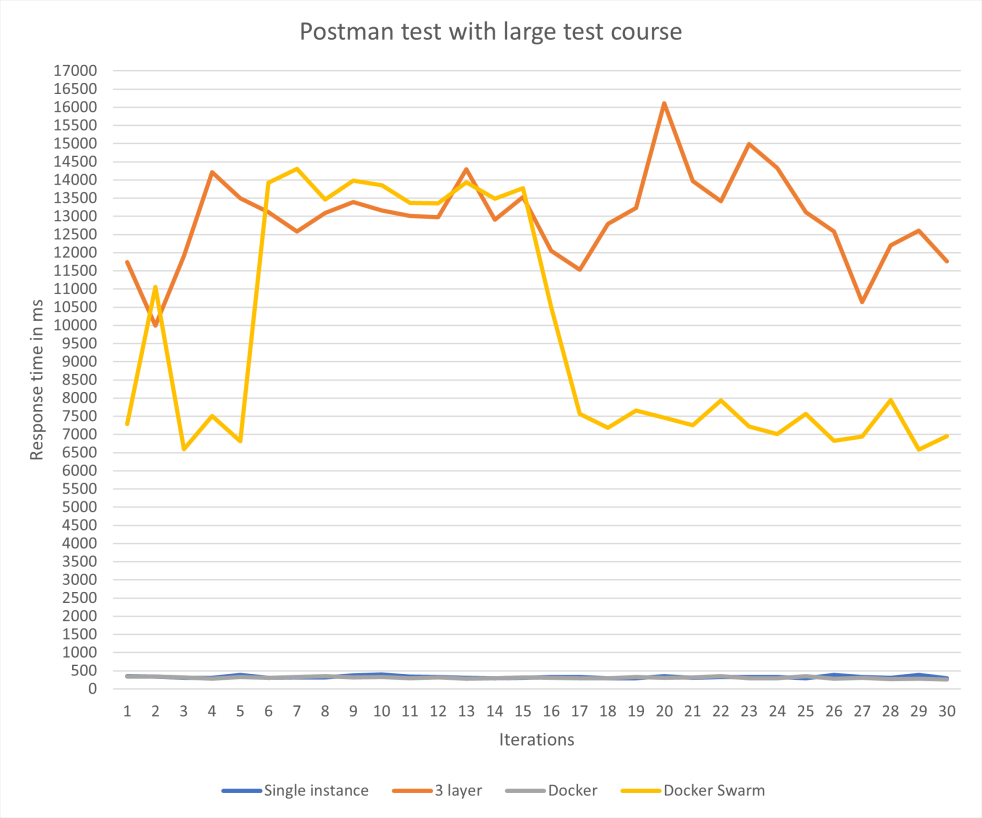


Figure 6.1: Large test course

Figure 6.2 presents the impact of unmounting GluserFS on the 3-layer and Docker Swarm architectures. However, these architectures do not work properly without a shared folder.

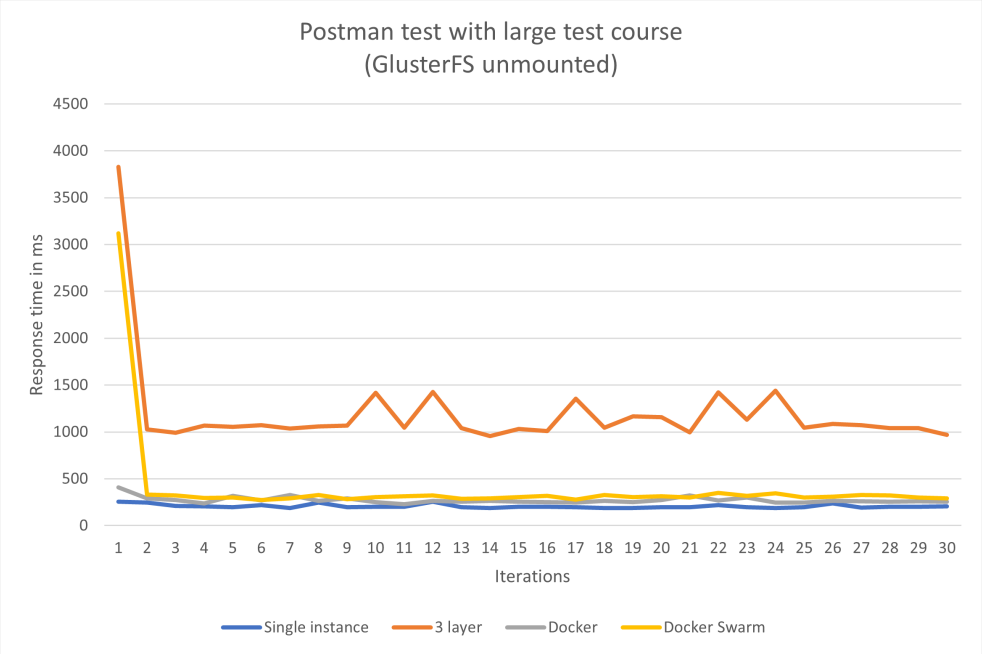


Figure 6.2: Large test course without GlusterFS

By decreasing the size of the course, the average response time of the two architectures using GlusterFS is reduced to approximately 4 seconds as seen in figure 6.3.

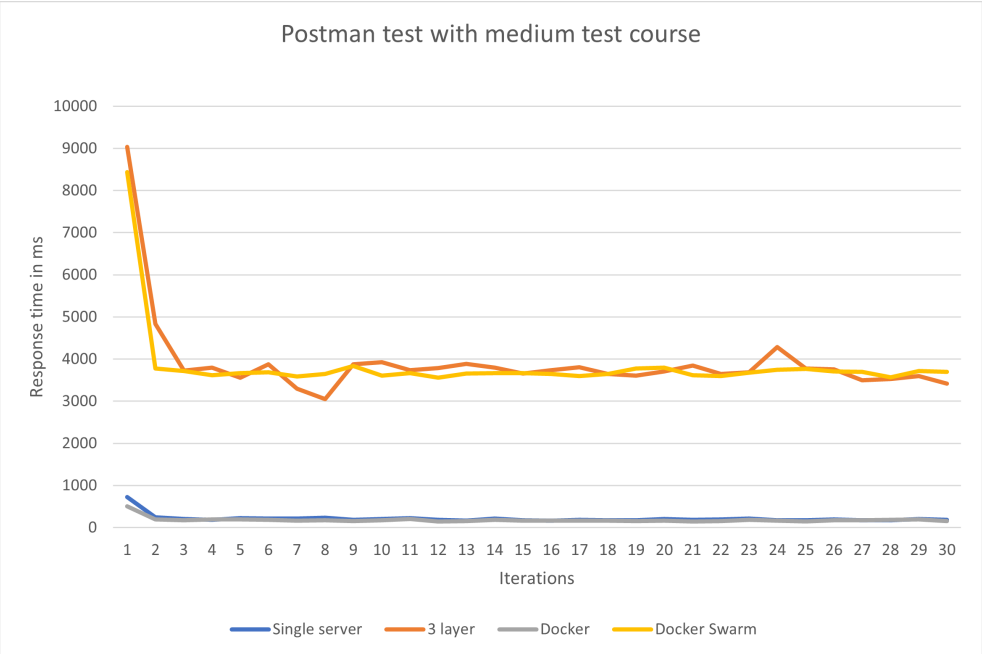


Figure 6.3: Medium test course

6.2.1 Architecture hardware resources

Table 6.1 presents the allocation of infrastructure resources for each architecture during testing. The resource allocation has been carefully set to align with the requirements presented in chapter 3, as well as current understanding and expertise in the team. It is important to acknowledge that the hardware resources allocated to the architectures may potentially influence the outcomes of the tests.

Architecture	Instances	vCPUs and RAM
Single Server Setup	1 instance	4 vCPUS, 4Gb RAM
3-Layer Architecture	4 Instances	8 vCPUs, 13GB RAM
Docker	2 instances	6 vCPU, 12GB RAM
Docker Swarm	3 Instances	6 vCPU, 12GB RAM

Table 6.1: Architecture resources

Each architecture is set up as described in chapter 5.

6.2.2 k6 Test results

During the k6 test, the test course is made accessible to everyone in order to ease the creation of test scripts. The script itself involves the following actions: Accessing the Moodle home page, load the test course, access a small file, return to the previous page, and access a discussion forum.

The k6 test is designed to perform the above actions while generating load of 100 concurrent users for a duration of 12 minutes. The executive summary for each test is documented in appendix D.

Summary of the test results

Architecture	Max Throughput	HTTP Failures	Avg Response Time	95% Response Time
Single server	54 reqs/s	0 reqs	1231ms	6079ms
3 layer	25 reqs/s	4012 reqs	1797ms	1306ms
Docker	36 reqs/s	0 reqs	1393ms	4927ms
Docker Swarm (50VUs)	4 req/s	357 reqs	31642ms	50020ms

Table 6.2: Infrastructure test results

Architecture: Table 6.2 compares different architectures, namely Single server, 3-layer, Docker, and Docker Swarm (50VUs). Each architecture represents a different setup or configuration being tested.

Max Throughput: This column indicates the maximum throughput achieved by each architecture during the test. The higher the value, the more requests per second (reqs/s) the architecture can handle. The Single server architecture achieved the highest throughput of 54 reqs/s. The 3-Layer architecture had a lower through-

put of 25 reqs/s. The Docker architecture achieved a slightly higher throughput of 36 reqs/s. The Docker Swarm architecture had the lowest throughput of 4 req/s.

HTTP Failures: This column represents the number of HTTP failures or requests that failed during the test. A HTTP request failure occurs when the HTTP response is a status code other than 200 OK. The Single server architecture and "Docker" architectures had 0 failed requests. The 3-Layer architecture experienced 4012 failed requests. The Docker Swarm architecture had 357 failed requests.

Avg Response Time: This column indicates the average response time (in milliseconds) for each architecture. It represents the time it takes for the system to respond to a request on average. The Single server architecture had an average response time of 1231ms. The 3-Layer architecture had a slightly higher average response time of 1797ms. The Docker architecture had a lower average response time of 1393ms. The Docker Swarm architecture had a significantly higher average response time of 31642ms.

95% Response Time: This column represents the 95th percentile response time, which indicates the response time for the 95% slowest requests. The Single server architecture had a 95th percentile response time of 6079 ms. The 3-Layer architecture had a lower 95th percentile response time of 1306 ms. The Docker architecture had a higher 95th percentile response time of 4927 ms. The Docker Swarm architecture had the highest 95th percentile response time of 50020 ms.

Load on instances during test

During the load test, the TI-stack is used to monitor the crucial system metrics of the instances, specifically CPU, memory, disk I/O, and network.

Disk I/O: The graphs bellow displays the disk I/O in bytes per second, reaching a maximum of approximately 30 MB/s on the Docker database during the test run (see Figure 6.10). This value is well below the limit of an SSD drive.

Network: Among the architectures, the highest network traffic is observed in the transmission and reception of data in the docker architecture. It is notable that the database and Docker host exhibit complementary patterns, with the blue graph representing bytes sent and the purple graph representing bytes received. Although other architectures show lower network traffic, the measured traffic is as expected. It's important to consider that network traffic can be influenced by factors such as available network bandwidth and the nature of the transmitted data, making it difficult to track accurately.

Single Instance

CPU usage: The graphs in figure 6.4 illustrates that the majority of the CPU is actively engaged, with approximately 83% of the CPU utilization attributed to user processes (indicated by the orange line). The remaining usage is allocated to

system operations (kernel). This indicates that a substantial portion of the CPU resources is being utilized by user processes, specifically the Moodle application.

Memory: Throughout the test, the memory utilization reaches a maximum of 54% of the available memory, indicating a moderate level of usage.

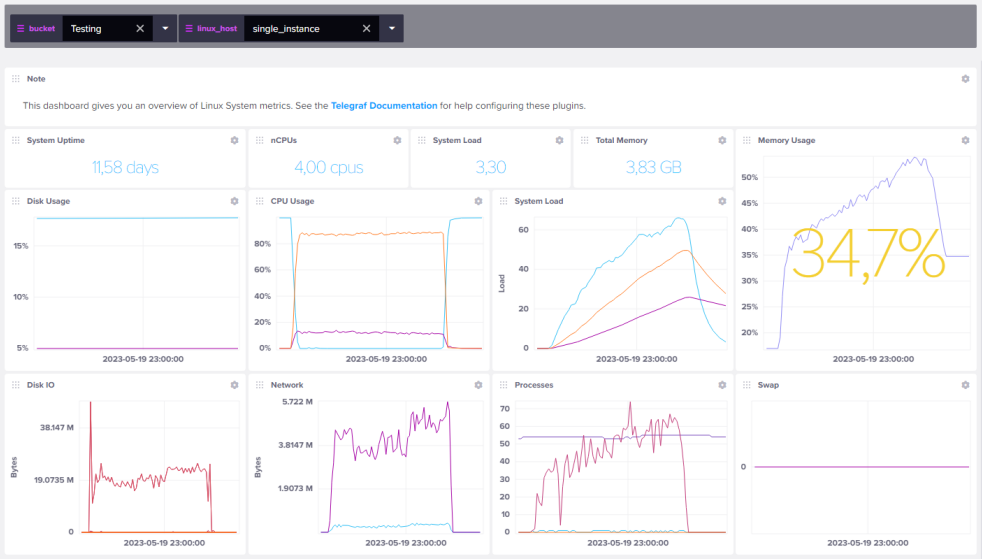


Figure 6.4: Monitoring of single instance architecture during test run

3 layer

CPU usage: The web servers exhibit an average CPU utilization of approximately 40% for user data, while the system operations account for slightly over 23%. Additionally, it can be observed that some of the CPU cores remain idle during the load tests. The database CPU has an average idle rate of approximately 85%. This indicates that a large portion of the database's CPU resources is not being actively utilized.

Memory: All web servers reach a maximum memory usage of 26%. This indicates that the memory resources on the web servers are not fully utilized and there is still a significant amount of available memory.

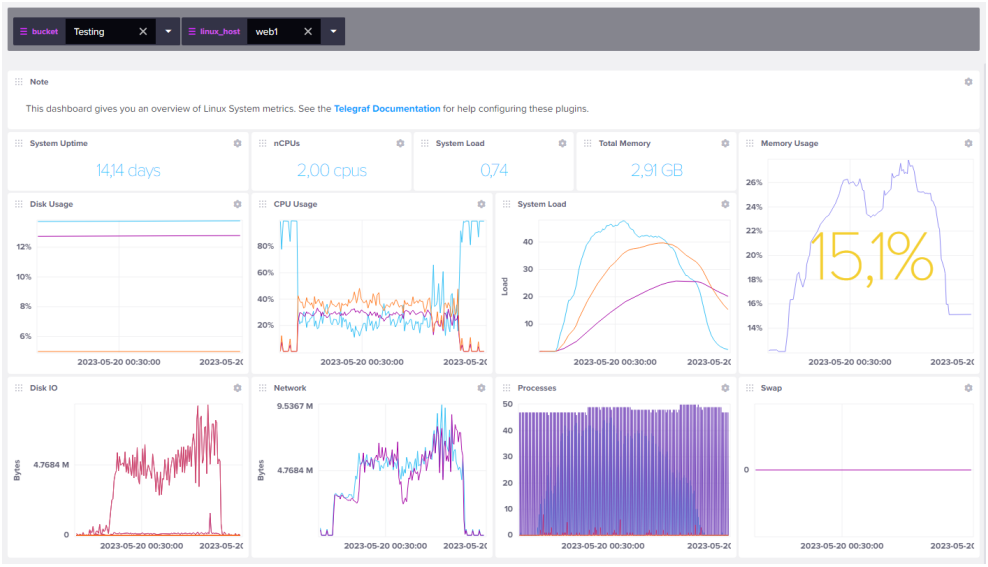


Figure 6.5: Monitoring of 3 layer web server 1 during test run

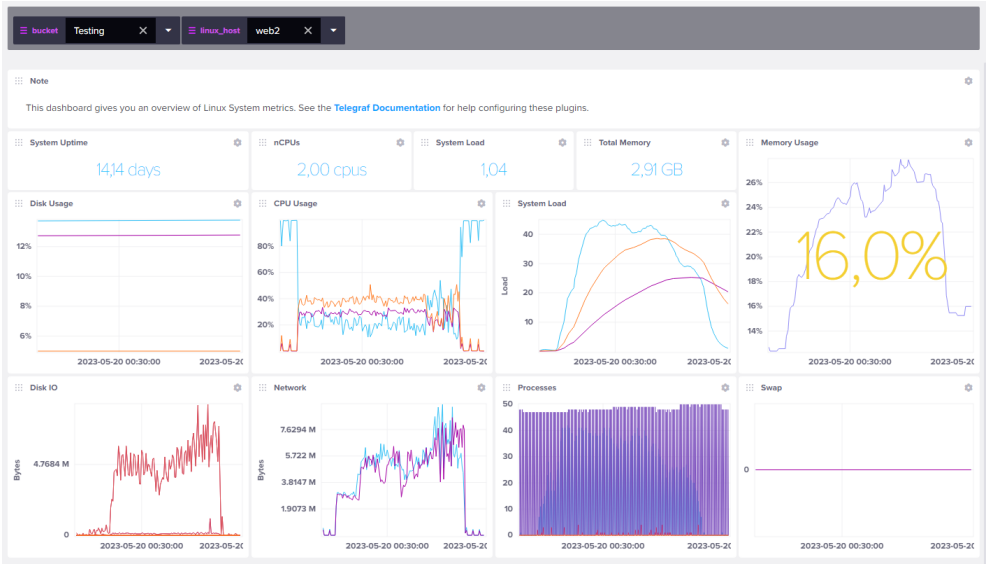


Figure 6.6: Monitoring of 3 layer web server 2 during test run

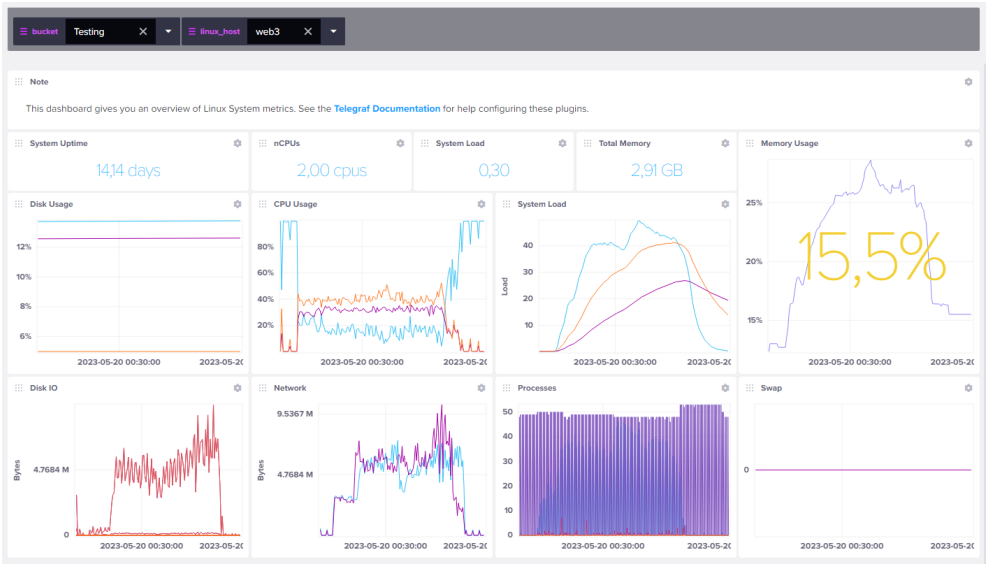


Figure 6.7: Monitoring of 3 layer web server 3 during test run

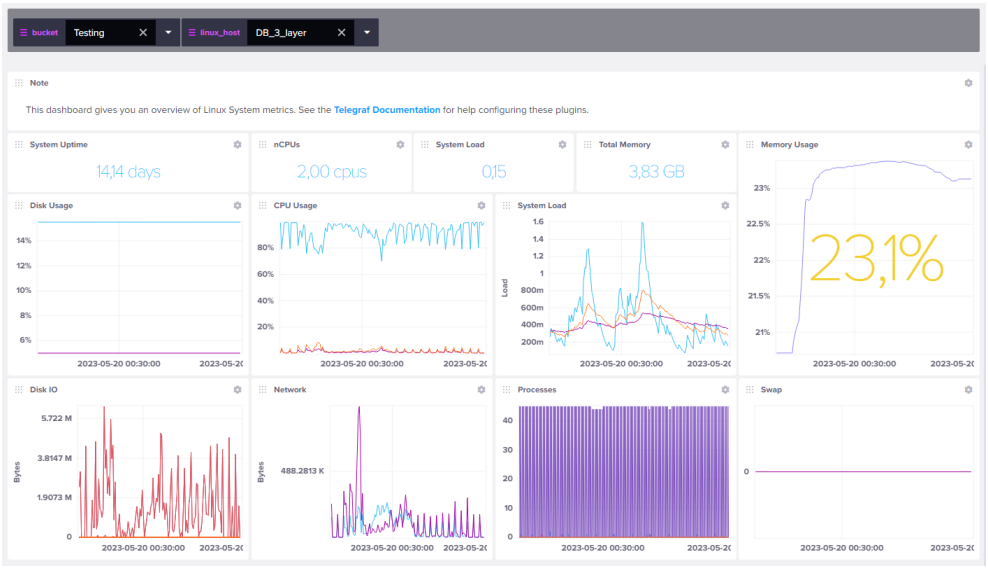


Figure 6.8: Monitoring of 3 layer database during test run

Docker

CPU usage: The Docker host, as shown in figure 6.9, exhibits CPU utilization similar to a single instance. Approximately 80% of the CPU is utilized by user processes, while the remaining CPU is allocated to system operations. This distribution of CPU usage indicates that a significant portion of the CPU resources on the Docker host is consumed by user applications or containers. The high user

CPU utilization suggests that the workload primarily consists of user processes, while the system operations utilize the remaining CPU capacity. Around 23% of the CPU resources in the database 6.10 are not actively utilized.

Memory: The Docker host reaches a peak memory usage of just above 35%, while the database host has a peak memory usage of 15%.

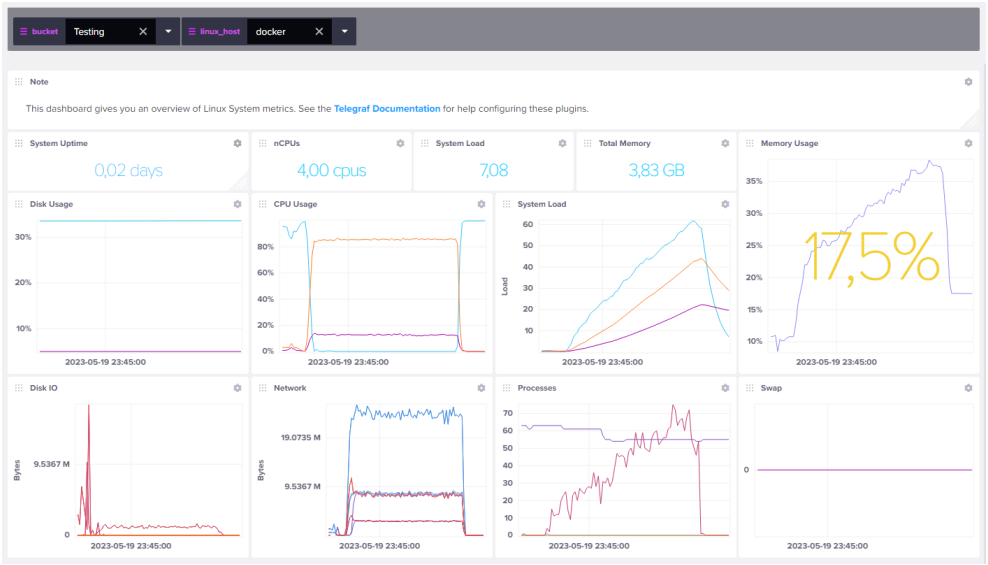


Figure 6.9: Monitoring of Docker host during test run

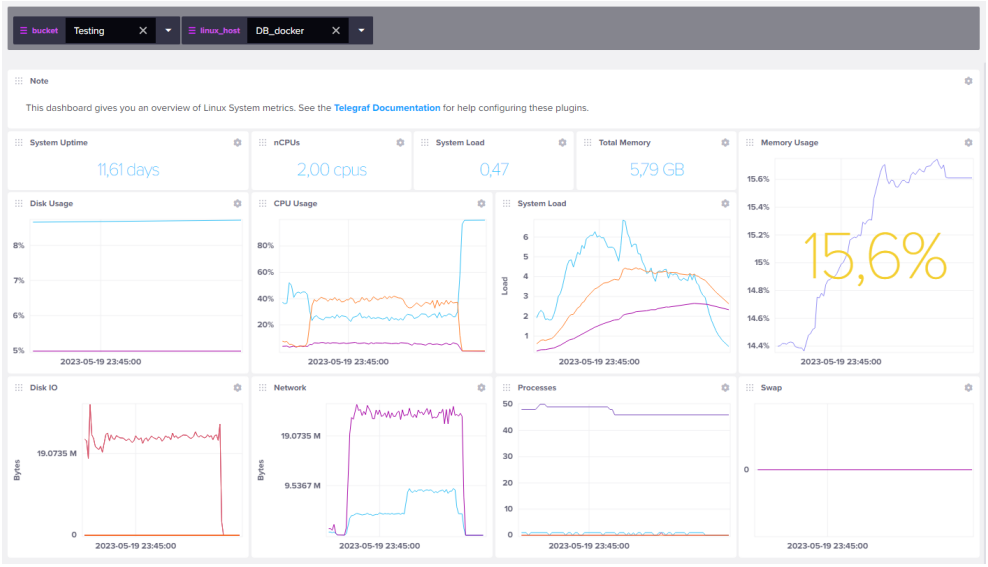


Figure 6.10: Monitoring of Docker database during test run

Docker Swarm

The three nodes in the swarm exhibit a well-distributed traffic pattern as their system loads appear to be quite similar.

CPU usage: The CPU usage in the Docker swarm architecture resembles the Three-layer architecture, with approximately 40% utilization by user processes and 20% utilization by system operations. Additionally, around 20% of the CPUs remain idle. The presence of idle CPUs implies that there is available processing capacity within the Docker swarm architecture.

Memory: This indicates that the memory resources in the architecture are utilized up to 40% of their capacity during peak usage. The memory usage level suggests that the architecture operates with a moderate level of memory consumption.

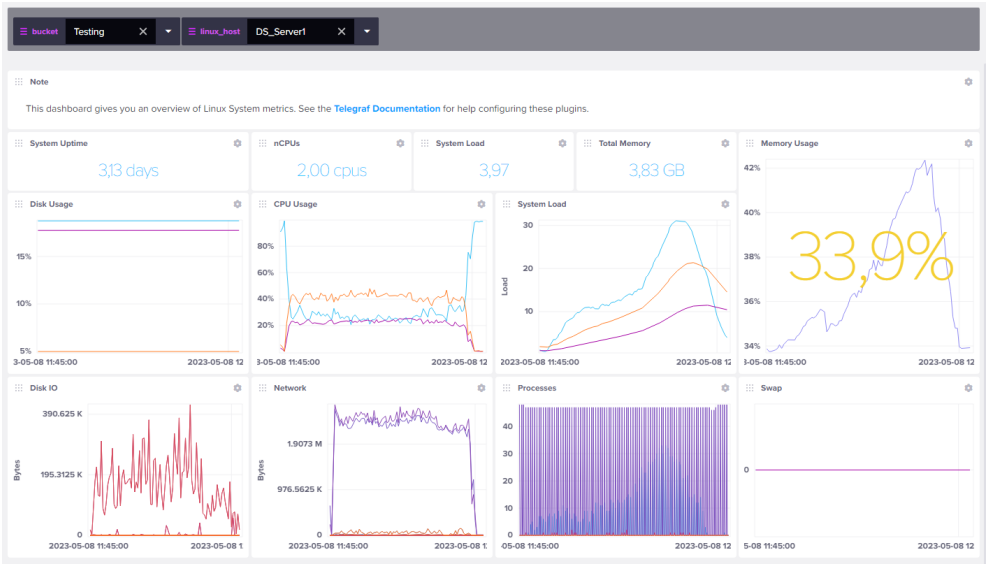


Figure 6.11: Monitoring of Docker Swarm container 1 during test run

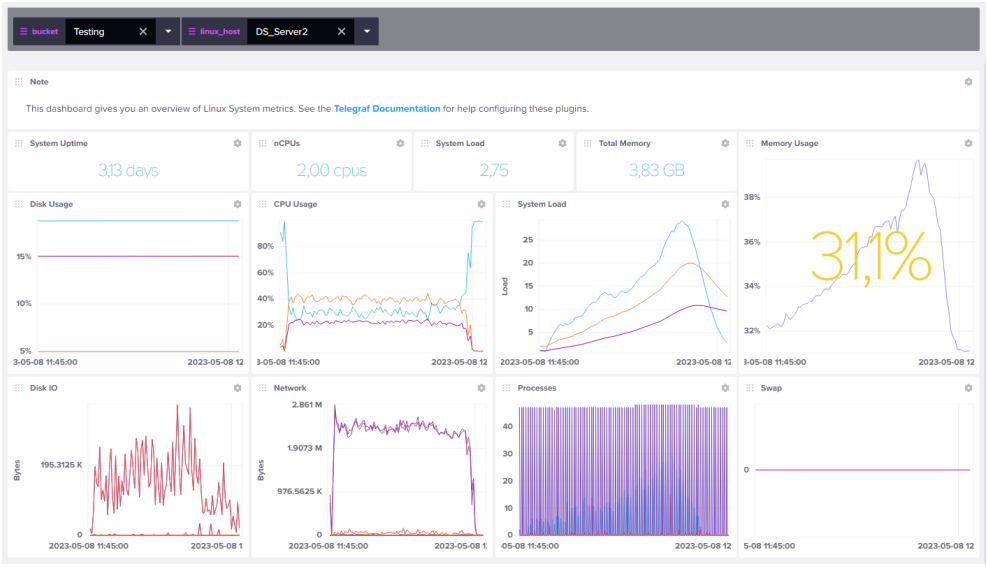


Figure 6.12: Monitoring of Docker Swarm container 2 during test run

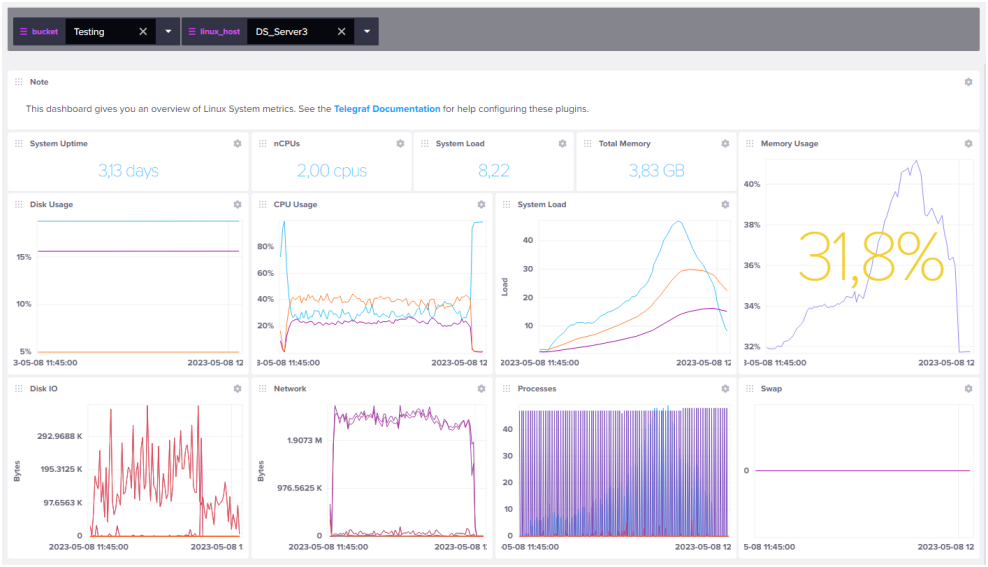


Figure 6.13: Monitoring of Docker Swarm container 3 during test run

The architectures exhibit varying CPU load and memory usage during the load test. Both the 3-layer architecture and the Docker Swarm architecture demonstrate similarities in system load. This can be attributed to the shared load distribution among different instances and the potential impact of GlusterFS as explained earlier 6.2.

6.2.3 Analysis of architecture performance and requirements

The Docker Swarm architecture, as discussed in section 6.1, offers a combination of robustness by addressing all single points of failure and scalability through the easy addition of machines to the swarm. This theoretically positions it as the recommended infrastructure architecture. However, the results presented indicate that the architecture did not perform as expected. This can be attributed to an incomplete setup, which highlights the need for further improvements, as discussed in section 8.

Based on the obtained results, it is noticeable that the Single Server architecture preformed better that the other architectures in terms of throughput, average response time, and 95th percentile response time. On the other hand, the Docker Swarm architecture exhibited the lowest performance across all metrics, despite having only 50 concurrent users during the test. An analysis of the monitored data suggests that both the 3-layer and Docker Swarm architectures were affected by the slow response of GlusterFS, which resulted in idle CPU usage as they awaited files from the NFS GlusterFS. In contrast, the Docker and Single Instance architectures showed 100% CPU utilization, primarily by the www-data (Moodle application), leading to better performance. It is worth noting that high CPU utilization implies that a new CPU process will have to wait for its turn, potentially resulting in increased response times for tasks.

Clients requirements:

No	Requirement	Status
1	Configurable infrastructure platform.	Implemented
2	Secure, tested platform.	Theoretical, not implemented
3	Redundant, load-balanced infrastructure.	Implemented
4	Robustness testing conducted.	Not yet addressed
5	TICK-stack for monitoring.	Implemented
6	Documented operation tasks.	Not yet implementation
7	Justified, documented configurations.	Implemented

Table 6.3: Client requirements

Table 6.3 presents the client requirements in short together with status.

Requirement **No. 1** has been fulfilled by establishing the infrastructure platform based on thorough research and discussions on technologies and services. All infrastructure architectures and components have been extensively documented and explained. However, there is still potential for further improvement in the solution.

Requirement **No. 2** has been theoretically implemented in this report, but practical implementation of the security is pending, as outlined in chapter 8. It is worth mentioning that the client follows security practices on their end.

Requirement **No. 3** has been implemented in this project. The architectures implemented aim to incorporate load balancing and redundancy. Furthermore, these architectures can be combined in various ways beyond the scope of this project, such as integrating the Docker or 3-layer architecture with a database cluster, ultimately ensuring high up-time for the service.

Requirement **No. 4** has not been addressed yet. However, the project team has encountered issues such as machines being turned off and database failures, although these occurred during development and were not properly documented. The importance of this requirement is acknowledged, and security features such as backup and restoration are discussed in chapter 8.

Requirement **No. 5** has been implemented using the TI-stack, which was used to monitor the services during the load test. TI-stack offers additional convenient services that can be implemented, including alerts and service monitoring. It can also be integrated with tools like Grafana.

Requirement **No. 6** is yet to be implemented. This includes documenting operational tasks such as upgrading the version of Moodle, performing backup and restoration procedures, and scaling the service. These aspects will be addressed in chapter 8.

Requirement **No. 7** has been fulfilled through the documentation provided in the public GitLab repository and the justification of technologies and architectures in this thesis. The documentation is designed to be easy to follow and adopt to the client's environment.

Operational requirements:

No	Requirement	Status
1	Response Time	Implemented, Potential for improvement
2	Security	Theoretical implementation, yet to be implemented
3	Scalability and Availability	Implemented, Potential for improvement
4	Usability	Implemented
5	Monitoring and Compliance	Implemented

Table 6.4: Operational requirements

Table 6.4 present in short the operational requirements defined in chapter 1.1.3.

Requirement **No 1** regarding response time has been fulfilled in two out of four architectures, the single instance and Docker architectures. The average response time for a single request in both architectures is below 500 ms, as one can see in

figure 6.3. However, it is important to note that the Docker Swarm architecture reveal low response time when GlusterFS service is not running, as shown in figure 6.1. It should be highlighted that the Moodle application does not function properly without the Moodledata shared between instances. During the load test with 100 concurrent users, the Docker and single server architectures demonstrate acceptable average response times. Conversely, the Docker Swarm and 3-layer architectures struggle when the load is increased, as indicated in table 6.2.

Requirement **No 2** has been discussed and analyzed in this report; However, the practical implementation is still pending and is outlined in section 8. It is crucial to highlight that the client follows their own security practices and guidelines.

Requirement **No 3** has been implemented and discussed. As observed in the load test, two of the architectures, Docker and single instance, are capable of accommodating 100 concurrent users. The Docker architecture leverages the benefits of containerization technology, resulting in reduced startup time, concurrent operation, and lower server and licensing costs. Additionally, the Docker host eliminates single points of failure in the web service.

Requirement **No 4** has been fulfilled through comprehensive documentation and explanation of the architectures in this report. The documentation and actual code are stored in a publicly accessible GitLab repository, with well-structured development practices. The prioritization lies in the development of code and documentation that is clear and comprehensible, aiming to enhance understandability and facilitate efficient collaboration and maintenance.

Requirement **No 5** has been achieved by utilizing the TI-stack, which effectively monitored the services during the load test. The TI-stack offers additional convenient services, including alerts and service monitoring, which have been utilized. Additionally, the k6 tool has been employed to test the load on the architectures.

6.3 Knowledge and technological familiarity's influence on choices and technologies

The project team faces a significant bias originating from limited experience beyond the concepts learned exclusively within the study program's courses. The bias centers around the course DCSG2003 - Robust and scalable services. It can be argued that the implementation closely adhered to the course curriculum, particularly with regards to Docker Swarm and the 3-layer architecture. The successful utilization of GlusterFS in the course led to a biased assumption that it would seamlessly adapt to accommodate the Moodle platform. However, as indicated by the test results, the bottleneck in the Docker Swarm and 3-layer architectures lies in GlusterFS, resulting in slow response times.

Chapter 7

Assignment critique

This chapter goes trough assignment critique based on the experience throughout the project. It discusses the strength and weaknesses of the assignment as a whole together with areas of improvement.

7.1 Short overview of the assignment

To accommodate the growth of Orange Business Services, it is necessary to utilize an e-learning platform for technician training. This platform offers multiple advantages compared to traditional learning methods, such as improved accessibility, scalability to handle 100 simultaneous users, and the ability to provide multimedia content. By leveraging this platform, training becomes more engaging and effective, ensuring that the content remains up-to-date with current and future best practices.

The use of specific technologies is not assigned, however, the students are required to utilize the open-source e-learning platform Moodle and establish the necessary infrastructure [59]. Orange Business Services has assigned this task to evaluate how bachelor students approach and address the challenges and opportunities presented in this assignment. Additionally, the use of TICK-stack [60] for monitoring is expected. The report and documentation includes a explanation and justification for the technologies employed.

7.2 Critique of the assignment

The assignment is quite extensive and require effective collaboration among the team members. The development process can follow various paths that ultimately lead to the same outcome: an e-learning platform that fulfills the requirements for up-time, security, and capacity. This includes understanding of the professional fields IT infrastructure [4] and system administration [5]. It is essential to become acquainted with a range of tools, thereby providing an explanation for the rationale behind their usage.

It is crucial to have a clear plan and effective working method to use in the project together with a plan of how to choose technologies. It is easy to get lost in the possible technologies that all support the same purpose, specially for students that only have a few guidelines. The task at hand is clear on what to expect as the final product, but not the way to get there or what techniques to use. This gives the group the freedom of choice when it comes to technologies.

The project team is primarily responsible for managing their own time usage, but they also have regular meetings with both the supervisor and the client. In addition, Orange Business Services offers dedicated resources that can be utilized if needed, while the supervisor provides guidance throughout the entire project, from beginning to end.

When it comes to complexity, the project group itself sets the complexity level based on knowledge and prior experience. The group can choose between a simple 3-layer architecture or a more complex containerized solution for the underlying infrastructure. The project group is responsible for administration like the level of

automation used, and allocated time for each part of the project in communication with supervisor.

Despite the assignment being broad and allowing for interpretation, it does provide clear operational requirements. This gives the project group the opportunity to explore and apply what they see appropriate based on their learning in the study program. The assignment can be viewed as a typical task presented by a company seeking assistance from consultants.

7.3 Student Performance

The students have very different personalities, which lead to well developed project solving skills. Though this also lead to various plans and ways to solve a problem, some are happy to read up on new literature and apply this to the project, while others are more likely to use prior knowledge and working solutions. This lead to diversion of same product and sometimes discussion on what to implement.

The cooperation in the group have lead to a well developed solution, although there have been some obstacles on the way. As described in chapter 8, there is not a lack of technologies to develop this product further. The students could be better at giving specific tasks to one another, set deadlines and follow these. Despite the students knowledge trough the study program, the use of resources at Orange Business Services could be utilized to more effectively solve problems encountered.

7.4 Reflection on Learning

This assignment has provided the students with a real-life project in which the requirements are defined, but the choice of technologies and how to fulfill those requirements is open. As a result, extensive research, trial and error, and iterative development approach have taken place. The project team has faced challenges in terms of working methods and collaboration due to their differences. All in all, it can be stated that this assignment has equipped the students with knowledge of new technologies and a profound understanding of both new and existing technologies and tools.

Chapter 8

Further work

During this project, the group found different alternatives for solutions, technologies and software. The options available could have been used to improve the quality of the architectures. This bachelor's project gave the group the opportunity to define the working methods freely, and made it difficult to decide the specific approaches to use to solve the problem given. This chapter outlines examples on further work the group would focus on if the project was extended.

8.1 Implementation improvements

The code implementation of the infrastructure as it stands, use Terraform modules for provisioning infrastructure components as a stack. Each component are using start-up scripts for running specific services on the components. Start-up scripts are imperative coding languages, and is a static, unreliable method for configuring services, as Bash scripts deprecate quickly. If the Moodle platform is updated, or any of the infrastructure components changes in the future, it is probable that the existing implementation of code no longer will work, and therefore needs to be frequently updated to fit the new software.

In the future, it is necessary to use declarative languages like Puppet or Ansible as well as Terraform, to handle the service configuration for the infrastructure components provisioned efficiently.

CI/CD

Continuous Integration (CI), is a software development practice in which all developers merge code changes in a central repository multiple times a day. CD, stands for Continuous Delivery, and is automating the release of code into production [61]. Although we are not developing the Moodle Platform ourselves, the group wants to secure a pipeline from when the Moodle developers release a new version of the Moodle platform, and when the new version is integrated on our service. This is to add an extra layer of security and to check that the new version does not include any security vulnerabilities or malicious code that could harm our system.

It is also desired by the group to have a more flexible and automated implementation system, to have our platform stay up to date with the latest software. This pipeline allows for reduced manual work and human error, and faster delivery of new functionality and improved security, which again allows for improved flexibility.

Consul

Consul [62] is a micro service networking solution with service mesh capabilities. Micro services are useful for many reasons. They support scalability, and can flexibly scale independently from each other, they are isolated systems meaning if a micro service fails, the rest of the services can still run. It allows for flexibility and improved integration and delivery because they are independent from each other. Although micro services greatly improve an application in numerous ways, the architecture's complexity will increase considerably. This issue can be greatly solved by Consul. Consul gathers information about every micro service developed, and monitors health and location of all services in real-time. The four core functionalities from Consul are service discovery, automated networking, secured networking, and controlled access. These functionalities can greatly improve the automation process of different services in the Moodle platform, but also enhance

security on the network through encryption and authentication. All these functionalities are worth to explore and integrate in the future.

8.2 Performance improvements

Nginx

Nginx [63] is one of the most reliable servers for scalability and speed. It is one of the fastest growing web servers in the industry, number one after Apache. The main difference between Apache and Nginx is that Nginx has event-driven architecture, handling multiple requests simultaneously within a single thread, while Apache is process-driven creating one thread per request. This allows for generally greater performance, compared to Apache.

Since the Moodle platform is a complex platform to host, the group wants to further explore the possibilities to use Nginx as a future web server. We also wish to explore the possibilities to combine both services for performance optimization, as they both serve great purposes.

Caching

Caching [64] is a technique to reduce peak traffic rates by prefetching popular content into memory at the end-user. These techniques are able to significantly improve response time for the end-user and is one the most efficient methods to increase performance. The groups current solutions have weak response times, and therefore it is highly desired for the group to focus on this further.

Docker

The orchestration technology Docker is intended to run lightweight containers for maximum performance. In this bachelor's project the group has used Ubuntu as the distro for running the Docker and other technologies/services. To maximize the performance of the physical resources, the group wants to further explore the use of other Linux distributions that are better supported and optimized to run docker containers. Distributions the group want to explore are Alpine and Fedora.

8.3 Security improvements

HTTP encryption (HTTPS)

An important step to secure the platform is to implement encrypted communication on the service. When requesting and responding to the HTTP protocol, data should be secured on the path to the destination or back to source, so the data can not be compromised. This is especially important for passwords and securing sessions for users.

Vulnerability testing

Testing the running platform for vulnerabilities is a time consuming job, but an important practice to secure the platform against unforeseen events or attacks. These measures helps an administrator understand the security applied by the Moodle developers, and gives perspective on security measures that can be taken on a self-hosted platform.

Sensitive data handling

To improve storage and handling of sensitive data like passwords, we wish to expand the current solutions overview on the use of the LDAP-server. In future architectures improvements, we wish to implement a LDAP-server where we can test the authentication process directly on the solution, and test reliability and performance of the additional component.

Backup and restoration

The most important security measure to secure data at rest from threat actors has always been backup and restoration. 0-days and other vulnerabilities are constantly being discovered and exploited. A business can never guarantee to be 100% secure from external or internal attacks. Therefore everyone should assume that their business can be compromised in the future. The best security measure to protect data is to always have a backup available. An active business consistently updates their data, so to counter this issue, it is important to backup frequently and preferably automatically. This will significantly reduce the risk of losing critical data when recovering from outdated files.

It is also worth mentioning that frequent backups are unhelpful unless there are reliable restoration and recovery systems. The group wants to explore the possibilities for a reliable, secure, and efficient backup/restoration system for the Moodle platform and its data at rest. It would also be relevant to test the backup/restoration system in a test environment before setting it in production, to test the security and reliability of the system, and to measure how quickly and painless the restore system is.

8.4 Monitoring

Continuous monitoring

Monitoring is an essential part of operating a running platform in production. The purpose is to always have knowledge of any incidents that may occur on the running platform. The group wants to explore the possibility of further developing a system that can continuous monitor the Moodle platform, so any administrator can reliably assure up-time of the service, and how to use this system for continuous monitoring.

Grafana is an open-source solution for running data analytics with the help of metrics which gives an insight on complex infrastructure and a massive amount of data that our services have to process. Grafana [65] compiles all data analyzed into customizable dashboards. Grafane can easily be integrated with the already set up monitoring.

The group also wants to explore the possibilities of running automated alerts in case of any incident that may happen. The goal is to eventually have a reliable system, that can automatically resolve issues consistently. These opportunities can be resolved by the use of Kubernetes, which is also an orchestration technology that might be an infrastructure architecture option for the Moodle platform in the future.

Automated testing

Load testing an architecture is usually a superior method for understanding how much traffic a platform can handle, before the platform suffers from latency, or even reduction of service reliability. The group wishes to efficiently improve the automation of load testing tools, to retrieve and analyze data after a load testing session has occurred. This is to efficiently test how much traffic an architecture can handle, and to apply knowledge on how to improve the performance under different circumstances.

The end goal is to implement a reliable system that can efficiently scale up and down infrastructure resources, depending on platform traffic. This will relieve stress on infrastructure components where traffic is low, but uphold the expectations for response time when the traffic is high. This method can reduce cost and extend the life cycle of physical infrastructure components.

8.5 Kubernetes

Kubernetes, commonly referred as K8s, is an open source solution that helps administrate containers in a cluster [66]. K8s a is widely used solution for managing containers in the cloud, and serves as a container orchestration engine. It manages the life-cycle of containers with ease.

Built upon docker as the Container Runtime Environment(CRE), Kubernetes act as the mastermind of the cluster orchestrating the containers. As developers, one only need to build the app image of the application and push it to the registry, and Kubernetes can deploy the app anywhere, thus making it homogeneous [66]. When configured properly, K8s takes care of activities such as restart on failure, auto scaling based on rules, and redeploying containers (pods) to different nodes in case of node failure.

Kubernetes can be seen as a further advancement of the Docker Swarm architecture that is implemented in the project. By setting up the Kubernetes cluster in the cloud provider one should have an even more robust and scalable service due

to Kubernetes' features. It is however a stiff learning curve with K8s, where as Docker Swarm serves as a simpler container orchestration tool.

Gonzalez's book contains a compelling and noteworthy point [67], that Docker Swarm and Kubernetes is not directly competitors, but rather servers different purposes. While Docker Swarm is seen as Docker on steroids, mostly straightforward to deploy, Kubernetes is suited for more advances applications and aids in areas like security and monitoring, backed up by research by the community and Google.

It is worth it to use time to implement Kubernetes, and look at the performance in context with the performance of the already developed architectures.

Chapter 9

Project evaluation

This section includes an evaluation of the groups work throughout the project. This is divided into how the work was organized and delegated, and the groups experience working with the project.

9.1 Organization

The organization of the group has been working as planned. The weekly meetings between the group members has been a successful event that has consistently occurred throughout the semester. It has been essential to have these group meetings, to understand the current workflow in the group, and to stay up to date with the progress on current work assignments delegated. The group meetings has helped us gain perspective on issues encountered, resolving of earlier issues, discussions on how we should generally move forward with the project from week to week, and further delegations of work.

The meetings with the supervisor and client has also been essential to get inputs from other viewpoints. These meetings has been important for guiding the project in the right direction and to get answers on questions throughout the project.

9.2 Work delegation

During the weekly meetings, each group member explains to the rest of the group what work they are planning to do the upcoming week. This has given the opportunity for each group member to focus on where they can utilize their work best, and has been conclusively a successful work methodology, although this could have been discussed a lot clearer during the group meetings.

The use of a Kanban board has been a useful framework tool that has given the group members perspective on what each group member is working on, and has efficiently improved the work flow during the project. This framework combined with the KISS principle, has helped the group members acquired basic knowledge that had been further developed to design more advanced solutions. This learning method improved each members knowledge, and has been successful throughout the project.

9.3 Progress plan

Throughout the project, the group had a Gantt-chart which is appended in appendix E, which prepared the different phases the project would follow. Due to frequent issues encountered throughout the research, development, and testing phase, the group had to postpone and alter the subsequent phases of the project. The amount of issues encountered from developing the architecture solutions, led the time management planned on the later phases to be spent on troubleshooting problems in the research phase. It was expected to have some problems and delayed activities during the research phase, but it was a significantly underestimated prediction prior to the project start.

During the planning phase, the group designed a risk analysis that would explain following risks and mitigations. Unfortunately the group didn't have a specific risk

associated with troubleshooting problems. If this was noted in the risk analysis with specific mitigations, perhaps the time management could've been spent more efficiently.

Appendix H presents a summary of the hours tracked in the Toggl time-tracking software. It displays the distribution of work hours throughout the bachelor's thesis. The three most frequently tracked activities are project reporting, development, and research.

Chapter 10

Conclusion

This project implements various infrastructure architectures to host the e-learning platform Moodle for Orange Business Services, aiming to meet the need for a training platform to train their technicians.

Chapter 1 presents the project's goals categorized as results, business, and learning outcomes. The **result goals** encompass the development of a robust, scalable, and secure infrastructure, as well as high-quality documentation. The test results discussed in chapter 6 indicate that Docker can effectively handle the concurrent load of 100 users which is one of the requirements by Orange Business Services. If this was the only requirement in the infrastructure, it would be a sufficient solution. Nevertheless, Docker itself lacks scalability, and especially redundancy. Therefore it is recommended that Orange Business Services, further develop these architectures, to sufficiently satisfy the specifications needed to meet the requirements. Explained in chapter 8, Kubernetes and other performance-oriented measures, are described as further steps that can satisfy these requirements.

The **business goals** focus on the client's ability to adopt the developed solution, and utilize the platform for technician training. On our end, these goals have been achieved through well-documented infrastructure architectures that successfully support a user load of 100. However, the assessment of the time and resources required for technician training is yet to be conducted.

Regarding the **learning outcome**, the project team has extensively explored different infrastructure architectures, paying attention to security, robustness, and scalability. The research and implementation have specifically revolved around Moodle, complemented by monitoring using the TI-stack. Furthermore, potential areas for further work beyond this thesis have been identified.

Bibliography

[1] WahooLearning. "What are the Benefits of a Learning Management System (LMS)?" (2023), [Online]. Available: https://wahoolearning.com/blog/learning-management-systems/benefits-customised-lms/ (visited on 05/19/2023).

[2] T. Dalzell, *The Routledge Dictionary of Modern American Slang and Unconventional English*. Routledge, 2009, ISBN: 9780415371827. [Online]. Available: https://books.google.no/books?id=5F-YNZRv-VMC.

[3] NTNU. "Community of Practice in Computer Science Education Home." (2016), [Online]. Available: https://www.ntnu.no/wiki/display/copcse/Community+of+Practice+in+Computer+Science+Education+Home (visited on 04/17/2023).

[4] Rahul Awati. "Getting started with a career in IT infrastructure: A brief guide." (2021), [Online]. Available: https://www.techtarget.com/whatis/feature/Getting-started-with-a-career-in-IT-infrastructure-A-brief-guide (visited on 04/25/2023).

[5] Coursera. "What Does a System Administrator Do? Career Guide." (2022), [Online]. Available: https://www.coursera.org/articles/what-is-a-system-administrator-a-career-guide (visited on 04/25/2023).

[6] IBM. "What is virtualization?" (n.d.), [Online]. Available: https://www.ibm.com/topics/virtualization?mhsrc=ibmsearch_a&mhq=virtualization (visited on 05/14/2023).

[7] Kief Morris, *Infrastructure as Code: Dynamic Systems for the Cloud Age*. O'reilly, 2020, ISBN: 9781098114671.

[8] IBM. "Cloud computing: A complete guide." (n.d.), [Online]. Available: https://www.ibm.com/cloud/learn/cloud-computing-gbl?mhsrc=ibmsearch_a&mhq=public%5C%26comma%5C%3B%5C%20private%5C%20hybrid%5C%20cloud (visited on 05/14/2023).

[9] IBM. "What are Iaas, Paas and Saas?" (n.d.), [Online]. Available: https://www.ibm.com/topics/iaas-paas-saas?mhsrc=ibmsearch_a&mhq=cloud%5C%20service%5C%20models (visited on 05/14/2023).

[10] Theo Schlossnagle, *Scalable Internet Architectures*. Sams Publishing, 2007, ISBN: 978-0-672-32699-8.

[11] Ben Lutkevich. "database (DB)." (), [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/database (visited on 05/21/2023).

[12] Mostafa Ibrahim. "What is Database Clustering?" (2022), [Online]. Available: https://www.harperdb.io/post/what-is-database-clustering (visited on 05/21/2023).

[13] IBM. "What is automation?" (n.d.), [Online]. Available: https://www.ibm.com/topics/automation?mhsrc=ibmsearch_a&mhq=Automation (visited on 05/14/2023).

[14] IBM. "What is containerization?" (n.d.), [Online]. Available: https://www.ibm.com/topics/containerization (visited on 05/14/2023).

[15] Xiaopu Ma, Ruixuan Li, Zhengding Lu, Jianfeng Lu, and Meng Dong, "Specifying and enforcing the principle of least privilege in role-based access control," eng, *Concurrency and computation*, vol. 23, no. 12, pp. 1313–1331, 2011, ISSN: 1532-0626.

[16] Ding Wang and Ping Wang, "The emperor's new password creation policies: An evaluation of leading web services and the effect of role in resisting against online guessing," eng, in *Computer Security – ESORICS 2015*, ser. Lecture Notes in Computer Science, vol. 9327, Cham: Springer International Publishing, 2015, pp. 456–477, ISBN: 9783319241760.

[17] Chun-Ting Huang, Lei Huang, Zhongyuan Qin, Hang Yuan, Lan Zhou, Vijay Varadharajan, and C.-C. Jay Kuo, "Survey on securing data storage in the cloud," eng, *APSIPA transactions on signal and information processing*, vol. 3, no. 1, 2014, ISSN: 2048-7703.

[18] Microsoft. "Enable LDAP over SSL with a third-party certification authority." (2023), [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-server/identity/enable-ldap-over-ssl-3rd-certification-authority (visited on 05/21/2023).

[19] Google. "What is a Disaster Recovery Plan?" (2023), [Online]. Available: https://cloud.google.com/learn/what-is-disaster-recovery (visited on 05/21/2023).

[20] Cloudflare Inc. "What is penetration testing? | What is pen testing?" (2023), [Online]. Available: https://www.cloudflare.com/learning/security/glossary/what-is-penetration-testing/ (visited on 05/21/2023).

[21] Noviantika G. "What Is Ubuntu? A Quick Beginner's Guide." (2023), [Online]. Available: https://www.hostinger.com/tutorials/what-is-ubuntu (visited on 04/13/2023).

[22] Opensource.com. "What is open source?" (2023), [Online]. Available: https://opensource.com/resources/what-open-source (visited on 04/13/2023).

[23] Instructure. “Yes, you can with Canvas.” (2023), [Online]. Available: https://www.instructure.com/canvas (visited on 05/21/2023).

[24] Moodle. “Installing Moodle.” (2022), [Online]. Available: https://docs.moodle.org/401/en/Installing_Moodle (visited on 04/17/2023).

[25] Howard Lei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran, and Pinar Ozcan, “A statistical analysis of the effects of Scrum and Kanban on software development projects,” eng, *Robotics and computer-integrated manufacturing*, vol. 43, pp. 59–67, 2017, ISSN: 0736-5845.

[26] David Gewirtz. “How does ChatGPT work?” (2023), [Online]. Available: https://www.zdnet.com/article/how-does-chatgpt-work/ (visited on 05/14/2023).

[27] American Psychological Association. “experimental method.” (n.d), [Online]. Available: https://dictionary.apa.org/experimental-method (visited on 04/17/2023).

[28] Moodle. “About Moodle.” (2014), [Online]. Available: https://docs.moodle.org/401/en/About_Moodle (visited on 04/24/2023).

[29] GNU. “What is Bash?” (2022), [Online]. Available: https://www.gnu.org/software/bash/manual/html_node/What-is-Bash_003f.html (visited on 04/13/2023).

[30] Opensource.com. “What is Bash?” (2022), [Online]. Available: https://opensource.com/resources/what-bash (visited on 04/13/2027).

[31] Jeff Novotny. “Install a LAMP Stack on Ubuntu 22.04.” (2022), [Online]. Available: https://www.linode.com/docs/guides/how-to-install-a-lamp-stack-on-ubuntu-22-04/ (visited on 04/13/2023).

[32] Moodle. “Installing AMP.” (2011), [Online]. Available: https://docs.moodle.org/401/en/Installing_AMP (visited on 04/24/2023).

[33] MySQL. “What is MySQL?” (n.d.), [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html (visited on 04/29/2023).

[34] U. Hairah and E. Budiman, *J. Phys.: Conf. Ser.*, vol. 1844, 012021, 2021. DOI: 10.1088/1742-6596/1844/1/012021.

[35] Ionos. “What is Galera Cluster?” (2020), [Online]. Available: https://www.ionos.com/digitalguide/hosting/technical-matters/mariadb-galera-clusters/ (visited on 05/01/2023).

[36] HAProxy. “HAProxy.” (2002), [Online]. Available: https://www.haproxy.org/ (visited on 04/27/2023).

[37] S. Toor *et al.*, *J. Phys.: Conf. Ser.*, vol. 513, 062047, 2014. DOI: 10.1088/1742-6596/513/6/062047.

[38] HashiCorp. “What is Terraform?” (2014), [Online]. Available: https://developer.hashicorp.com/terraform/intro (visited on 04/13/2023).

[39] OpenStack. “OpenStack Orchestration.” (n.d), [Online]. Available: https://wiki.openstack.org/wiki/Heat (visited on 05/15/2023).

[40] Eigil Obrestad. “Openstack at NTNU.” (2016), [Online]. Available: https://www.ntnu.no/wiki/display/skyhigh/Openstack+at+NTNU (visited on 04/24/2023).

[41] OpenStack. “What is OpenStack?” (n.d), [Online]. Available: https://www.openstack.org/software/ (visited on 04/18/2023).

[42] IBM. “What is Docker?” (n.d.), [Online]. Available: https://www.ibm.com/topics/docker?mhsrc=ibmsearch_a&mhq=docker (visited on 05/21/2023).

[43] Docker Inc. “Docker Compose overview.” (n.d.), [Online]. Available: https://docs.docker.com/compose/ (visited on 05/21/2023).

[44] IBM. “Docker Swarm vs. Kubernetes: A Comparison.” (2022), [Online]. Available: https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison?mhsrc=ibmsearch_a&mhq=docker%5C%20swarm (visited on 05/21/2023).

[45] Docker Inc. “Docker Registry.” (n.d.), [Online]. Available: https://docs.docker.com/registry/ (visited on 05/21/2023).

[46] Mainak Chakraborty and Ajit Pratap Kundan, *Monitoring Cloud-Native Applications*. Apress, 2021, ISBN: 978-1-4842-6888-9.

[47] Grafana Labs. “The best developer experience for load testing.” (2023), [Online]. Available: https://k6.io/ (visited on 05/14/2023).

[48] Wikipedia contributors. “OpenLDAP — Wikipedia, The Free Encyclopedia.” (2023), [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenLDAP&oldid=1139266868 (visited on 05/06/2023).

[49] Aditya Sridhar. “An introduction to Git: what it is, and how to use it.” (2018), [Online]. Available: https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/ (visited on 05/15/2023).

[50] Moodle. “Environment - max input vars.” (2021), [Online]. Available: https://docs.moodle.org/402/en/Environment_-_max_input_vars (visited on 05/03/2023).

[51] Moodle. “Security Recommendations.” (2023), [Online]. Available: https://docs.moodle.org/402/en/index.php?title=Security_recommendations&oldid=83534 (visited on 04/27/2023).

[52] Moodle. “Moodledata directory.” (2010), [Online]. Available: https://docs.moodle.org/19/en/index.php?title=Moodledata_directory&oldid=77966 (visited on 04/27/2023).

[53] Influxdata. “Dashboards.” (2023), [Online]. Available: https://www.influxdata.com/products/influxdb-templates/gallery/ (visited on 04/25/2023).

[54] Russ Savage. “Linux System Monitoring Template.” (2020), [Online]. Available: https://github.com/influxdata/community-templates/tree/master/linux_system (visited on 05/14/2023).

[55] Docker Inc. “httpd.” (2023), [Online]. Available: https://hub.docker.com/_/httpd (visited on 05/21/2023).

[56] Moodle. “github moodle/moodle.” (2023), [Online]. Available: https://github.com/moodle/moodle (visited on 05/21/2023).

[57] Docker Inc. “haproxy.” (2023), [Online]. Available: https://hub.docker.com/_/haproxy (visited on 05/21/2023).

[58] Moodle. “Test course generator.” (2021), [Online]. Available: https://docs.moodle.org/39/en/Test_course_generator (visited on 05/06/2023).

[59] Moodle. “About Moodle FAQ.” (2020), [Online]. Available: https://docs.moodle.org/401/en/About_Moodle_FAQ#What_is_Moodle.3F (visited on 04/13/2023).

[60] InfluxData Inc. “InfluxDB 1.x.” (2023), [Online]. Available: https://www.influxdata.com/time-series-platform/ (visited on 05/15/2023).

[61] Marko Anastasov. “What is a CI/CD pipeline?” (2022), [Online]. Available: https://semaphoreci.com/blog/cicd-pipeline (visited on 05/20/2023).

[62] Philippe Bournhonesque. “HashiCorp Consul: Because Dynamic Workloads Call for Dynamic Service Networking.” (), [Online]. Available: https://www.devoteam.com/expert-view/hashicorp-consul-because-dynamic-workloads-call-for-dynamic-service-networking/ (visited on 05/20/2023).

[63] Richard B. “NGINX vs Apache – Choosing the Best Web Server in 2023.” (2022), [Online]. Available: https://www.hostinger.com/tutorials/nginx-vs-apache-what-to-use/ (visited on 05/19/2023).

[64] Mohammad Ali Maddah-Ali and Urs Niesen, “Fundamental Limits of Caching,” English, *IEEE Transactions on Information Theory*, vol. 60, no. 5, pp. 2856–2867, 2014, ISSN: 0018-9448. DOI: 10.1109/TIT.2014.2306938.

[65] Shivang. “What is Grafana and what is it used for?” (), [Online]. Available: https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/ (visited on 05/20/2023).

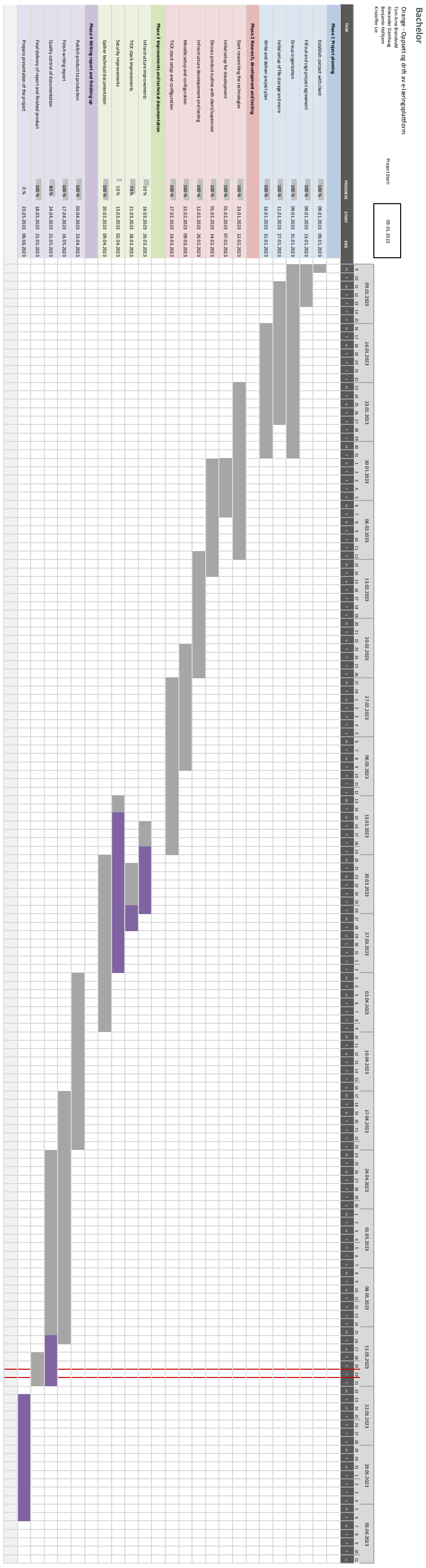
[66] Prateek Khushalani, *Kubernetes Application Developer*, eng. Apress Berkeley, CA, 2022, ISBN: 978-1-4842-8031-7.

[67] David Gonzalez, “Docker Swarm and Kubernetes - Clustering Infrastructure,” eng, in *Implementing Modern DevOps*, United Kingdom: Packt Publishing, Limited, 2017, ISBN: 1786466872.

Appendix A

Progress plan

The progress plan throughout the project is summed up in our Gantt chart, which is appended in the following appendix. In this project management tool there is an overview on all the different phases throughout the project.



Appendix B

Documentation

This appendix contains the technical documentation on every solution produced in the project. The documentation consists of the architecture that we consider to be a possible solution for developing the Moodle platform. Along with the architectures, there are additional components documented as well. These additional components are TI-Stack, LDAP server, and other documentation such as OpenStack GUI configuration. The documentation is available in the following public Git-repo: `https://gitlab.com/dcsg2900_2023_orange/bachelor_orange/-/tree/main`.

B.1 Architectures

B.1.1 Single server setup

How to manually configure and run Moodle using the LAMP method.

Author: Alexander Damhaug, Date: 04.02.2023

Following document is a thorough documentation on how to sucessfully run the Moodle Service on a Ubuntu Server 22.04 LTS (Jammy Jellyfish amd64) Virtual Machine. It is a simplification of Linode's solution, and includes both the installation of a LAMP Stack (Virtual Machine with: Linux, Apache, MySQL, and PhP) And a manual on how to sucessfully run Moodle on the stack.

NOTE: This is the simplest form of running a Moodle platform, and it's purpose is to build a learning curve on understanding the service, and should not be used as a solution itself.

The Sources of this documentation:

- Install a LAMP Stack on Ubuntu 22.04: https://www.linode.com/docs/guides/how-to-install-a-lamp-stack-on-ubuntu-22-04/
- Install Moodle on Ubuntu 22.04: https://www.linode.com/docs/guides/how-to-install-moodle-on-ubuntu-22-04/

Prerequisite

Following steps is expected to be preconfigured:

- Virtual machine with Ubuntu 22.04 image (manager)
- Terraform installed
- On-prem cloud solution with OpenStack as opensource IaaS tool
- Existing virtual network, where manager is a part of

See: `1_doc_openstack_setup.md` in `5_doc_other_components`

Setting up the environment

1. Create a directory and move into the empty environment

```
mkdir moodle_Infra
cd moodle_Infra
```

2. Clone the repo with the terraform code: Use own access token below with minimum read-repo rights

```
git clone https://private-token:<access-token>@<link-to-gitlab>
```

3. Copy the directory with the terraform code used and navigate into the dir

```
cp -r bachelor_orange/code/1_code_single_instance/ .
cd 1_code_single_instance/
```

4. Modify the following variables in the terraform file on main.tf to fit your environment in OpenStack
    - `keypair`
    - `network`
    - `security_groups`

Run the terraform code and configure the remaining steps

```
terraform init
terraform plan
terraform apply
```

How to install LAMP (Linux, Apache, MySQL, PhP)

1. Update the ubuntu packages using `apt`

```
sudo apt update && sudo apt upgrade
```

2. Install apache server using `apt`

```
sudo apt install apache2
```

3. Install MySQL web server using `apt`

```
sudo apt install mysql-server
```

4. Install PHP, along with php modules compatibale with Apache and MySQL using `apt`

```
sudo apt install php libapache2-mod-php php-mysql
```

How to install Moodle on Ubuntu 22.04

1. Ensure Ubuntu system is up to date

```
sudo apt update && sudo apt upgrade
```

2. Confirm release of PHP

```
php -v
```

3. Install the remaining php packages using `apt`

```
sudo apt-get install graphviz aspell ghostscript clamav php-pspell php-curl php-gd
php-intl php-mysql php-xml php-xmlrpc php-ldap php-zip php-soap php-mbstring git
```

4. Remove ; and change setting max_input_vars to atleast 5000 and restart the web server

```
sudo nano /etc/php/8.1/apache2/php.ini
sudo service apache2 restart
```

5. Move to `/opt` directory, and clone the Moodle Git repo

```
cd /opt
sudo git clone git://git.moodle.org/moodle.git
```

6. Change to the Moodle directory and list the branches in the Moodle repository. Review the list choose the latest stable release

```
cd moodle
sudo git branch -a
```

7. Track, and choose the desired git-branch

```
sudo git branch --track MOODLE_401_STABLE origin/MOODLE_401_STABLE
sudo git checkout MOODLE_401_STABLE
```

8. Copy the contents of the Moodle repo to the running Apache service, and change the moderation of the moodle directory

```
sudo cp -R /opt/moodle /var/www/html/
sudo chmod -R 0777 /var/www/html/moodle
```

NOTE: For security reasons, consider changing the directory permissions

9. Create the `/var/www/moodledata` directory and change owner and permissions

```
sudo mkdir /var/www/moodledata
sudo chown -R www-data /var/www/moodledata
sudo chmod -R 0777 /var/www/moodledata
```

How to configure the MySQL Server for Moodle

1. Create a password generated for the admin user

```
sudo apt install pwgen

pw=$(pwgen -s 24 1)
sudo bash -c "echo $pw >> password.txt"
pw=""
sudo chmod 600 password.txt
```

2. Log into mySQL using `root` privileges and create a password

```
sudo mysql -u root -p
```

3. Create a database for Moodle

```
CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

4. Create a Moodle MySQL user, and grant them permissions for the database.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY '$(cat passord.txt)';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE TEMPORARY TABLES,DROP,INDEX,ALTER
ON moodle.* TO 'admin'@'localhost';
```

5. Exit database

```
quit
```

How to configure the Moodle service to finish installation GUI

1. Go in the browser and put the following in the URL: `<floating_IP>/moodle`

2. Finish the installation, by following the steps provided in the browser

```
Choose Language -> Next -> Confirm paths -> Choose database driver: Improved MySQL
database driver -> Database settings:

Database host:     Localhost
Database name:     Moodle
Database user:     admin
Database password: <generated-password>

Review the licensing agreement and continue
```

3. Moodle verifies installations, Ensure all `server checks` indicate `ok`. If the installation is successful, Moodle displays the message `Your server environment meets all minimum requirements` select **Continue**

4. Moodle will now install every service it provides in the background, this may take some time, so just be patient.

5. Press **continue** and configure the main administrator account which will have complete control over the platform.

```
Username:      <username>
Password:      <Password>
First Name:    <first_name>
Last Name:     <last_name>
Email Address: <Email_Address>
City/Town:     <City/Town>
Country:       <Country>
Timezone:      <Timezone>
Description:   <Description>
```

1. Installation of home web page

```
Full site name: <site_name>
Short name for site: <short_name>
Support Email: <support_email>
Noreply address: <noreply_address>
```

Now you have a fully working Moodle platform!

B.1.2 3-Layer architecture

How to configure a complete infrastructure with the moodle-platform using terraform

Author: Alexander Damhaug, Date: 28.02.2023

Following document is a thorough documentation on how to successfully run the Moodle platform on a complete self-provisioned infrastructure with a separate load-balancer, database and an optional amount of web-servers for configuration. In this document, we use the Ubuntu Server 22.04 LTS (Jammy Jellyfish amd64) Virtual Machine for the compute resource.

NOTE: This documentation, uses preconfigured "terraform code and bash script" for provisioning the infrastructure and the Moodle platform itself.

Sources:

- How to implement sticky sessions with haproxy: https://www.haproxy.com/blog/enable-sticky-sessions-in-haproxy/
- How to cluster multiple servers: https://severalnines.com/blog/clustering-moodle-multiple-servers-high-availability-and-scalability/
- ChatGPT - specifically: Bash script-syntax
- Lecture from DCSG2003, Uke 10: "Nettverkslagring"
- Lecture from DCSG2003, Uke 3: "Arkitekturer og lastbalansering"

Prerequisite

Following steps is expected to be preconfigured:

- Virtual machine with Ubuntu 22.04 image (manager)
- Terraform installed
- On-prem cloud solution with OpenStack as opensource IaaS tool
- Existing virtual network, where manager is a part of

See: `1_doc_openstack_setup.md`

Setting up the environment

1. Create a directory and move into the empty environment

```
mkdir moodle_Infra
cd moodle_Infra
```

2. Clone the research repo with the terraform code: Use own access token with read-repo rights

```
git clone https://private-token:<access-token>@<link-to-gitlab>
```

3. Copy the directory with the terraform code used and navigate into the dir

```
cp -r bachelor_orange/code/1_code_single_instance/ .
cd 1_code_single_instance/
```

4. Modify the following variables in the terraform files in the module folder to fit your environment in OpenStack
    - `instance_count (Amount of webservers)`
    - `instance_name`
    - `instance_flavor`
    - `instance_key_pair (Manager public key)`
    - `network (Same as manager)`
    - `security_groups`

```
database -> variables.tf
loadbalancer -> variables.tf
webservers -> variables.tf
```

5. Make sure to also modify the loadbalancer startup script with the correct repo location

```
loadbalancer -> startupscript.sh
```

Run the terraform code and configure the remaining steps

```
terraform init
terraform plan
terraform apply
```

NOTE: This configuration can take several minutes, be patient!

Configure Gluster FS between the **webservers**

NOTE: Replace <"server1-x"> section with the ip-addresses of applicable webserver

On every server

1. Open a secure connection between all the provisioned webservers through ssh from main. Change the hostname to web1-x to differentiate the webservers

```
ssh <ip-address>
sudo hostname <webX>
sudo su
```

On one server (server1)

1. Run the following command(s), but only use the ip-addresses of the remaining webservers

```
gluster peer probe <server2>
gluster peer probe <server3>
...
gluster peer probe <serverX>
```

NOTE: If 'gluster' not found is the output, the startup installation is not finished yet, wait some minutes and try again. Check the status.txt: `cat ~/status.txt`

2. Check if every webserver is within the same cluster

```
gluster peer status
```

3. Create and run the volume (Change number to amount of webservers provisioned)

```
gluster volume create moodle replica 2 <server1>:/moodle_brick
<server2>:/moodle_brick <server3>:/moodle_brick  force

gluster volume start moodle
```

On every server

1. Mount the the volume on the existing moodledata directory: serverX is the ip-address of the server you are running the command on

```
mount -t glusterfs <serverX>:moodle /var/www/moodledata
```

2. Check if the mount was successful

```
df -h
```

On one server

1. Change the owner and permissions for the /moodledata directory

```
sudo chown -R www-data /var/www/moodledata
sudo chmod -R 0777 /var/www/moodledata
```

Configure the config.php, start CLI installation, and finish configuration in GUI

- Go back to manager, and retrieve the config-file in the research-repo

```
cp /home/ubuntu/moodle_Infra/research/config/config.php .
```

- Modify the `config.php` to your specification, in our solution you only need to change the following arguments:

```
$CFG->dbhost    = '<DB_ip-address>';
$CFG->wwwroot   = 'http://<LB_floating-IP>/moodle';
```

- Create config.php file under /var/www/html/moodle directory in every webserver and copy the config contents earlier and paste it in the same config.php file

```
ssh <ip-address>
sudo su
nano /var/www/html/moodle/config.php
^U
```

- On Server1: Change the password below to the Admin password generated for Moodle in the startupscript, and run the command to start the installation process: (Can take several minutes to finish installation)

```
php /var/www/html/moodle/admin/cli/install_database.php --agree-license --
adminpass=<generated_password>
```

- Copy the floating-IP and paste it in the browser
- Log in as the admin you've just created, and finish and update the profile
- Finish the site settings configuration and save changes

Now you have a fully working Moodle platform on a provisioned infrastructure, with Load-Balancing!

B.1.3 Docker

Docker initial setup

Docker

This document goes through the setup and configuration of docker to set up moodle

Short description:

The docker architecture is set up to run multiple containers with the self-made docker image.

Prerequisite:

Following steps is expected to be preconfigured:

- Virtual machine with Ubuntu 22.04 image (manager)
- Terraform installed
- On-prem cloud solution with OpenStack as opensource IaaS tool
- Connection to the openstack provider.
- Existing virtual network, where manager is a part of

See `/doc/5_doc_other_components/1_doc_openstack_setup.md` for manual prosess to set up these prerequisite.

Use terraform to set the infrastructure.

Clone down the needed code in the folder '/code/3_code_docker/infra_code/'

```
git clone https://private-token:<access-token>@<link-to-gitlab>

cp -r  ./bachelor_orange/3_code_docker/docker_code .
```

Edit these variables before you apply the code:

```
/modules/<module_name>/variables.tf

    instance_key_pair
    network
    security_groups

/modules/docker/startupscript.sh

    docker registry ip
```

Modules (database, docker and docker_loadbalancer)

Apply the code

```
Terraform init
terraform apply
```

This code sets up three servers in openstack. These are a docker instance, a database and a loadbalancer for docker containers. Software is installed with startup scripts.

After a few minutes the three instances are created. Both the database and loadbalancer does the software configuration for you.

If you have prebuilt images pushed to the private registry, follow this guide docker_registry.md. To build your own images localy, see this guide docker_build.md. Continue on the newly created docker instance.

Docker registry

Using images from private docker registry

This document guides you trough how to use images stored on a private docker registry server.

Before running: Edit `dbhost` to database IP and `www-root` to "http://balancer-floating-ip/moodle" in `bachelor_orange/config/config.php` and push changes. Ensure that the database type is set to `mysqli`

Pull images:

```
sudo docker pull <registry_IP>:5000/moodle_install_db_script
sudo docker pull <registry_IP>:5000/moodle_www
```

Running the containers:

If the db is new, make sure the moodle database is created and run the install database container:

```
sudo docker run -P <registry_IP>:5000/moodle_install_db_script:latest
```

Note that this may take some time to run trough the database initialzation.

Start the webserver container:

To start webserver container run

```
sudo docker run -d -p 22222:80 -v moodledata:/moodledata
10.212.170.44:5000/moodle:latest
sudo docker run -d -p 33333:80 -v moodledata:/moodledata
10.212.170.44:5000/moodle:latest
sudo docker run -d -p 44444:80 -v moodledata:/moodledata
10.212.170.44:5000/moodle:latest
```

Remember to open for ports/portrange used for containers in sec-group.

You can now visit the running service at http:///moodle

Docker build

Setup to build images on a docker instance:

In home directory: clone `bachelor_orange` repo

```
git clone https://private-token:<access-token>@<link-to-gitlab>
```

Copy the files to your location

```
cp -r  ./bachelor_orange/3_code_docker/docker_code .
```

Go into the folder:

```
cd docker_code/
```

Clone the moodle code from moodle github:

```
git clone -b MOODLE_401_STABLE git://git.moodle.org/moodle.git
```

Build the two images needed, moodle_www:

```
cd docker_www

sudo docker build -t moodle_www:<version-tag> -f Dockerfile ..
```

And detabase_script image:

```
cd ../docker_moodle_db-setup

sudo docker build -t moodle_www:<version-tag> -f Dockerfile ..
```

How to run

Before running: Edit `dbhost` to database ip and `www-root` to "http://balancer-floating-ip/moodle" in `bachelor_orange/config/config.php` and push changes. Ensure that the database type is set to `mysqli`

Running the containers:

If the db is new, make sure the moodle database is created and run the install database container:

```
sudo docker run -P moodle_install_db_script:latest
```

Note that this may take some time to run trough the database initialzation.

Start the webserver container:

To start webserver container run

```
sudo docker run -d -p 22222:80 -v moodledata:/moodledata moodle:latest
sudo docker run -d -p 33333:80 -v moodledata:/moodledata moodle_www:latest
sudo docker run -d -p 44444:80 -v moodledata:/moodledata moodle_www:latest
```

Remember to open for ports/portrange used for containers in sec-group.

You can now visit the running service at http:///moodle

B.1.4 Docker swarm

This document goes through the setup and configuration of docker swarm running Moodle.

Author: Kristoffer Lie, Date: 08.05.2023

Short description:

The docker swarm architecture is a combination of all our previous infrastructures and knowledge. Docker swarm is a container orchestration tool that allows us to run up redundant amount of services that all runs the same docker image. Docker itself handles what container runs where and load balancing internal.

Following steps in the documentation:

1. 3 ubuntu instances
2. Installed and setup MariaDB cluster
3. Set up shared network folder using glusterFS
4. Install and setup docker, including docker swarm
5. Run database container (install DB)
6. Set up openstack loadbalancer (web and db)
7. Deploy the stack

Prerequisite:

Following steps is expected to be preconfigured:

- Virtual machine with Ubuntu 22.04 image (manager)
- Terraform installed
- On-prem cloud solution with OpenStack as opensource IaaS tool
- Existing virtual network, where manager is a part of

Use terraform to set up the three servers:

The code needed is stored in the folder '/code/4_code_docker_swarm'. This sets up three servers in openstack. Software is installed with a startup script.

Edit these variables befor you apply the code:`

```
/modules/server/variables.tf

    instance_key_pair
    network
    security_groups

/modules/server/startupscript.sh

    docker registry ip on line (at the end)
```

Apply the code

```
Terraform init
terraform apply
```

Note that you need terraform installed and connect to the openstack provider.

Note down the ip addresses for each serves:

```
server1: <ipadd>
server2: <ipadd>
server3: <ipadd>
```

Configure the db-servers

On every DB server

1. Open a secure connection between all the provisioned servers through ssh from main.

```
ssh <ip-address>
sudo su
```

2. Configure local DNS to map the ip-addresses of the 3 DB-servers in /etc/hosts 'db1-db2-db3'

serverX: `nano /etc/hosts`:

```
<server1> db1
<server2> db2
<server3> db3
```

3. Ping domain-names to assure connectivity on the network `ping dbX`

4. Copy the contents of the following DB-cluster configuration file

- Source: - MariaDB cluster setup:

```
[mysqld]
# Cluster node configurations
wsrep_cluster_address="gcomm://db1,db2,db3"
# Make sure this is corresponds to hostname:
wsrep_node_address="dbx"
innodb_buffer_pool_size=600M

# Mandatory settings to enable Galera
```

```
wsrep_on=ON
wsrep_provider=/usr/lib/galera/libgalera_smm.so
binlog_format=ROW
default-storage-engine=InnoDB
innodb_autoinc_lock_mode=2
innodb_doublewrite=1
query_cache_size=0
bind-address=0.0.0.0

# Galera synchronization configuration
wsrep_sst_method=rsync
```

6. Create and modify `cluster.cnf` to the servers specification, so it can join the cluster. Change the following argument "dbx" to db1-3 on the respective server you are on. server1: `nano /etc/mysql/conf.d/cluster.cnf`:

```
wsrep_node_address="db1"
```

server2: `nano /etc/mysql/conf.d/cluster.cnf`:

```
wsrep_node_address="db2"
```

server3: `nano /etc/mysql/conf.d/cluster.cnf`:

```
wsrep_node_address="db3"
```

On one DB-server (server1)

1. Start server1 in bootstrap mode, this server will be the master of this database-cluster: `galera_new_cluster`

On the remaining servers

1. Start their database and confirm connectivity to the cluster. Complete synchronization can take a few minutes.

```
service mysql restart
service mysql status
```

2. Check cluster size

```
mysql -u root -p"" -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
```

One DB-server:

1. Create the Moodle database, the admin user with the generated password and restart the DB

```
pw=$(pwgen -s 24 1)
sudo bash -c "echo $pw >> password.txt"
sudo chmod 600 password.txt


sudo mysql -u root -p"" -e "CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;"

sudo mysql -u root -p"" -e "CREATE USER 'admin'@'%' IDENTIFIED BY ‘$pw’;"

sudo mysql -u root -p"" -e "GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE
TEMPORARY TABLES,DROP,INDEX,ALTER ON moodle.* TO 'admin'@'%';"

sudo sed -i 's/#bind-address.*/bind-address = 0.0.0.0/'
/etc/mysql/mariadb.conf.d/60-galera.cnf

sudo systemctl restart mysql

pw=""
```

2. Connect to another server in the cluster and check if the database is available

```
mysql -u root -p"" -e "SHOW DATABASES"
```

Set up glusterFS

Probe the the servers from server1

```
gluster peer probe <server2_ip>
gluster peer probe <server3_ip>
```

Create the gluster volume

```
gluster volume create moodle replica 3 <server1>:/moodle_brick
<server2>:/moodle_brick <server3>:/moodle_brick force
```

Start the volume:

```
gluster volume start moodle
```

Mount the volume on each server:

```
mount -t glusterfs <serverx_ip>:moodle /moodledata
```

Verify connection by creating a file on server1:

```
touch /moodledata/I_AM_MOUNTED
```

On server2 and server3:

```
ls /moodledata
```

give the proper access to moodledata:

```
sudo chown -R www-data /moodledata
sudo chmod -R 0777 /moodledata
```

You should now have a shared network folder mounted on /moodledata

Create a loadbalancer in GUI for HTTP

Create a loadbalancer in openstack GUI.

Update the floating IP of wwwroot in config.php in the gitlab repo. Set the dbhost to be server1 private IP.

Set up the database:

First, we need to pull down the image from our local docker registry.

To allow http pull, add this registry:

```
echo "{
\"insecure-registries\" : [\"http://<registry_ip>:5000\"]
}" >> /etc/docker/daemon.json

sudo systemctl restart docker
```

Pull and run the install database installation script:

```
sudo docker pull <registry_ip>:5000/moodle_install_db_script

sudo docker run -P <registro_ip>:5000/moodle_install_db_script:latest
```

Set up the swarm:

Initialize the swarm on server1

```
docker swarm init
```

Take note of the join command outputted and paste it in the remaining servers.

Get the docker-compose file and start the moodle service!

Get the docker compose file and haproxy config in the /code/config folder:

```
curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-
api>/config/docker_compose.yaml" >> docker_compose.yaml

curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-
api>/config/haproxy.cfg" >> haproxy.cfg
```

Start the moodle service!

In order for the LB container to work, edit the dbhost to 'getenv("MOODLE_DB_HOST")' before you start up the stack.

Pull down the web service image:

```
sudo docker pull <registry_ip>:5000/moodle_www_krisern
```

Run the stack

```
sudo docker stack deploy -c docker_compose.yaml moodle
```

Use the set_premissions_moodledata.sh to set the preferd premissions on dataroot (/moodledata).

```
 curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-api>" >>
set_premissions_moodledata.sh
```

make the file executable, and run the set premissions script:

```
./set_premissions_moodledata.sh /moodledata
```

B.2 Additional infrastructure

B.2.1 TI-Stack

Set up monitoring of services (linux monitoring):

@ Author Kristoffer Lie

First, we need to provision a influxdb server, witch is done using terraform.

Get the code:

git clone https://private-token:@

Extract the folder needed and move into it:

```
cp -r code/5_code_other_components/monitoring/ .

cd monitoring.
```

Some of the variables in variables.tf should be changed:

- `instance_key_pair`,
- `network1`
- `network_pool`
- `security_groups`
- `instance_flavor`

```
nano variables.tf
```

Running `terraform init` followed by `terraform apply` will provision a instance with influxdb installed and started, together with a floating IP to access the servcer.

Head to the server floating ip and port 8086 to get started:

```
http://<floating_ip>:8086
```

Log in with the cridentials provided in the startupscript.

Add a template dashboard:

Head to the dashboard tab on the left --> create dashboard --> add template --> browse comunity templates.

Choose the template you prefer, we use the linux system template:

```
https://github.com/influxdata/community-templates/tree/master/linux_system
```

Copy the .yaml url into the influxdb site and click `Lookup Template` --> install template

If you head back to your dashboard you should now see a linux system dashboard.

Generate a API token to be used to connect to the influxDB in the API token tab. Take note of the Token generated and store it somewhere safe!

Setup telegraf to run (on evry host)

Note: inspierd by this site: https://serverfault.com/questions/1057001/telegraf-works-manually-but-not-the-service-run-telegraf-in-background

Install telegraf:

```
sudo apt install telegraf
```

Verify that telegraf is installed:

```
telegraf version
```

Curl down the config files:

```
 curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-
api>"/bachelor_orange/config/env_telegraf >> env_telegraf

 curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-
api>"/bachelor_orange/config/linux_montr.conf >>
/var/www/html/moodle/linux_montr.conf
```

Fill out the env variables in the env_telegraf file to fit your influx setup and send it to telegraf

```
nano env_telegraf

sudo cp env_telegraf /etc/default/telegraf
```

Edit the hostname to match the instance.

```
nano linux_montr.conf

``
```

Copy the telegraf configuration file to let telegraf service read from it:

```
sudo cp linux_montr.conf /etc/telegraf/telegraf.conf
```

Restart the telegraf service and check the status

```
sudo systemctl restart telegraf

sudo systemctl status telegraf
```

After 5 minutes, head to the linux system dashboard! Spesify the bucket and host.

B.2.2 LDAP server

OpenLDAP

Config/installation

Installation:

```
sudo apt install slapd ldap-utils -y

sudo dpkg-reconfigure slapd
#  - Omit OpenLDAP server configuration? No
#  - DNS domain name? orange.ba
#  - Organization name? Bachelor Orange
#  - Administrator password?
#  - Do you want the database to be removed when slapd is purged? No
#  - Move old database? Yes
```

Create users

```
# Setting variable for generated passwords for user
sudo apt install pwgen

password=$(pwgen -s 24 1)
sudo bash -c "echo $password >> password.txt"
sudo chmod 600 password.txt

# Create OU
sudo ldapadd -x -D cn=admin,dc=orange,dc=ba -W << EOF
dn: ou=People,dc=orange,dc=ba
objectClass: organizationalUnit
ou: People
EOF

# Create users
sudo ldapadd -x -D cn=admin,dc=orange,dc=ba -W << EOF
dn: uid=User,ou=People,dc=orange,dc=ba
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: User
givenName: User
sn: User
```

```
cn: User User
displayName: User User
mail: user@user.user
uidNumber: 1001
gidNumber: 1001
userPassword: $password
gecos: User User
loginShell: /bin/bash
homeDirectory: /home/user

dn: uid=User2,ou=People,dc=orange,dc=ba
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: User2
givenName: User2
sn: User
cn: User2 User
displayName: User2 User
mail: user2@user.user
uidNumber: 1002
gidNumber: 1001
userPassword: $password
gecos: User2 User
loginShell: /bin/bash
homeDirectory: /home/user2

dn: uid=User3,ou=People,dc=orange,dc=ba
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: User3
givenName: User3
sn: User
cn: User3 User
displayName: User3 User
mail: user3@user.user
uidNumber: 1003
gidNumber: 1001
userPassword: $password
gecos: User3 User
loginShell: /bin/bash
homeDirectory: /home/user3

dn: uid=User4,ou=People,dc=orange,dc=ba
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: user4
givenName: User4
sn: User
cn: User4 User
displayName: User4 User
mail: user4@user.user
```

```
uidNumber: 1004
gidNumber: 1001
userPassword: $password
gecos: User4 User
loginShell: /bin/bash
homeDirectory: /home/user4
EOF

# Clearing password variable
password=""
```

This creates the following structure:

- orange.ba
  - People
    - User
    - User2
    - User3
    - User4

User passwords are set using a saved password file. The file with the password is only accessible as root, and the file is deleted as soon as it's not needed anymore.

Setup Moodle to use LDAP server

https://docs.moodle.org/401/en/LDAP_authentication#Enabling_LDAP_authentication

1. Go to Site administration -> Plugins -> Authentication -> Manage authentication and click the eye icon opposite LDAP Server. When enabled, it will no longer be greyed out.
2. Click the settings link, configure as required (see information below), then click the 'Save changes' button.

Configure the following settings:

Setting	New value
auth_ldap | host_url	ldap://10.212.174.170
auth_ldap | preventpassindb	Yes
auth_ldap | bind_dn	cn=admin,dc=orange,dc=ba
auth_ldap | bind_pw	YOUR LDAP ADMIN PASSWORD
auth_ldap | user_type	posixAccount (rfc2307)
auth_ldap | contexts	ou=people,dc=orange,dc=ba
auth_ldap | field_map_firstname	givenName
auth_ldap | field_map_lastname	sn
auth_ldap | field_map_email	mail

Configure cron job

https://docs.moodle.org/401/en/LDAP_authentication#Enabling_the_LDAP_users_sync_job

1. Go to Site administration -> Server -> Scheduled tasks and click the gear icon on LDAP users sync job.
2. Select the desired frequency of running and enable the task by un-ticking the disabled checkbox.

Run the following command on the Moodle server to check if the cron jobs work:

```
sudo /usr/bin/php /var/www/html/moodle/admin/cli/cron.php
```

Next configure cron on the server to run the previous command automatically:

https://docs.moodle.org/36/en/Cron_with_Unix_or_Linux

```
sudo crontab -u www-data -e

# Add this line:
*/1 * * * * /usr/bin/php /var/www/html/moodle/admin/cli/cron.php >/dev/null
```

B.3 Other documentation

B.3.1 OpenStack GUI-Setup

How to create a stack of components in Openstack.

Author: Alexander Damhaug, Date: 30.01.2023

Following document is a thorough documentation on how to successfully build a stack of components in GUI for Openstack.

Create a Router

- A router is a virtual component that connects multiple networks within the Openstack environment. This allows the user to connect their virtual network with the gateway of the on-prem solution.
- 'ntnu-internal' refers to the standarized internal network in SkyHiGH (NTNU's on-prem cloud platform that runs the Openstack software). An external user, typically has a similar option, with a different name.

```
Network     -> Routers        -> Create Router -> Router Name: "default" -> create
router ->
Set Gateway -> Select network -> ntnu-internal -> submit
```

Create a Network

- In this step, you have to create your own private network, an environment your instances can launch on.

```
Network -> Networks        -> Create Network -> Network Name: "default" -> Subnet
->
Subnet Name: "default"     -> Network Address: 192.168.X.X/24 -> Create
```

Create Security Groups

- A security group is essentially a simple firewall, to block or allow specific types of traffic or ip-addresses.

```
Security Group -> Name: "default"  -> Create Security Group ->
Add rules      -> Rule: HTTP, HTTPS, SSH -> Ingress -> Add
```

NOTE: Ensure that all ip protocols and any port range is open for traffic within the network

Create key pair / Import public key

- Creating a public key for your openstack project is necessary to be able connect to instances created.

```
Compute -> Key Pairs -> Import Public Key -> Key Pair Name: "default"
                     -> Key Type: SSH Key -> Browse: default.pub -> Import Public
Key
```

Create Main Instance (Manager)

- Use the latest available image.
- Following section is an example of compute resources, that can be applied.

```
Instances -> Launch Instance ->
Instance Name: "Test_Manager"
Source: 'Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64'
Flavor: 'gx1.2c4r' (2 VCPUS, 4GB, 40 GB Total Disk)
Networks: "default"
Security Groups: "default" -> Security Groups -> Key Pair -> default -> Launch
Instance
```

Associate Floating IP

- Associate a floating IP to the instance created, to reach it outside the network.

```
Instances -> "Test_Manager" -> Dropdown    -> Associate Floating IP -> +
                            -> Allocate IP -> Select a port: Test_Manager ->
Associate
```

SSH into the VM

- Open a secure connection the instances created.

```
ssh -i .\default ubuntu@<floating_IP>
```

What's next?

- Run Moodle: See Documentations -> doc/1_doc_single-instance/single_instance.md

NOTE: You can start at "How to install LAMP (Linux, Apache, MySQL, PhP)" if you've finished `1_doc_openstack_setup.md`

Appendix C

Code

In this appendix all the code used to build the infrastructure components including the semi-automated startup-scripts for the solutions are included. Similar to appendix B, all the following code are used in all the architectures considered to be potential solutions for running the Moodle platform. The code of additional components like TI-stack is also included. The code is available in the following public Git-repo: `https://gitlab.com/dcsg2900_2023_orange/bachelor_orange/-/tree/main`.

C.1 Infrastructure 1: Single server setup

```
1_code_single_instance
└── main.tf
```

main.tf:

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Configure the OpenStack Provider
provider "openstack" {
    cloud = "openstack"
}


# Variables
variable "keypair" {
  type    = string
  default = "default"   # name of keypair created
}


variable "network" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}

# Data sources
## Get Image ID
data "openstack_images_image_v2" "image" {
  name        = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64" # Name of image
    to be used
  most_recent = true
}

## Get flavor id
data "openstack_compute_flavor_v2" "flavor" {
  name = "gx3.4c4r" # flavor to be used
}


```

```
# Get a floating IP
resource "openstack_networking_floatingip_v2" "fip_1" {
  pool = "ntnu-internal"
}

# Create a terraVM
resource "openstack_compute_instance_v2" "TerraVM_Alex" {
  name            = "single_instance"
  image_id        = data.openstack_images_image_v2.image.id
  flavor_id       = data.openstack_compute_flavor_v2.flavor.id
  key_pair        = var.keypair
  security_groups = var.security_groups

  network {
    name = var.network
  }

  user_data = file(var.startup_script)
}

resource "openstack_compute_floatingip_associate_v2" "fip_1" {
  floating_ip = "${openstack_networking_floatingip_v2.fip_1.address}"
  instance_id = "${openstack_compute_instance_v2.TerraVM_Alex.id}"
}
```

Code listing C.1: Infrastructure 1: Single server setup

C.2 Infrastructure 2: 3-Layer Architecture

```
2_code_3_layer
├── modules
│   ├── database
│   │   ├── main.tf
│   │   ├── startupscript.sh
│   │   └── variables.tf
│   ├── loadbalancer
│   │   ├── main.tf
│   │   ├── startupscript.sh
│   │   └── variables.tf
│   └── webservers
│       ├── main.tf
│       ├── outputs.tf
│       ├── startupscript.sh
│       └── variables.tf
└── main.tf
```

main.tf:

```
# Define required providers
terraform {
```

```
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Configure the OpenStack Provider
# Note that the clouds.yaml from openstack is located inside ‘‘.config/openstack/‘‘
     folder
provider "openstack" {
    cloud = "openstack"
}


module "database" {
  source = "./modules/database/"
}


module "loadbalancer" {
  source = "./modules/loadbalancer/"
  depends_on = [module.database]
}
```

Code listing C.2: Infrastructure 2: 3-Layer Architecture - "main.tf"

`/modules/database/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "database" {
  name = var.instance_name
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }

  user_data = file(var.startup_script)
```

```
}
```

Code listing C.3: Infrastructure 2: 3-Layer Architecture - Database "main.tf"

`/modules/database/variables.tf:`

```

# The file contains variable declaration and some default values.
# This file together with main.tf and terraform.tfvars provision and instantiate a
    infrastructure stack
# See teraform documentation for more details of syntax/code

# @ File variables.tf
# @ Author Alexander Damhaug, NTNU

# Define variables:
variable "instance_name" {
 type = string
 default = "database"
 description = "Name of the instance"

}

variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx1.2c4r"
 description = "Flavor for the server"
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "startup_script" {
   type = string
   default = "./modules/database/startupscript.sh"
   description = "startup script to be run on the instance"
}

variable "network" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
```

```
}
```

Code listing C.4: Infrastructure 2: 3-Layer Architecture - Database "variables.tf"

`/modules/database/startupscript.sh:`

```
#!/bin/bash

echo "startupscript started" >>~/status.txt

# Update ubuntu packages
sudo apt update && sudo apt upgrade

# Install password generator
sudo apt install pwgen

# Install Mysql web server
sudo apt install mysql-server -y

echo "Installed mysql" >>~/status.txt

# Generate a random password only root can access
pw=$(pwgen -s 24 1)
sudo bash -c "echo $pw >> password.txt"
sudo chmod 600 password.txt



# log in to the database as root
sudo mysql -u root

sudo mysql -u root -p"" -e "CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;"
sudo mysql -u root -p"" -e "CREATE USER 'admin'@'%' IDENTIFIED BY '$pw';" # Use
    relevant the ip-address instead of %
sudo mysql -u root -p"" -e "GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE
    TEMPORARY TABLES,DROP,INDEX,ALTER ON moodle.* TO 'admin'@'%';"
sudo sed -i 's/bind-address.*/bind-address = 0.0.0.0/' /etc/mysql/mysql.conf.d/
    mysqld.cnf
sudo systemctl restart mysql

pw=""

quit
```

Code listing C.5: Infrastructure 2: 3-Layer Architecture - Database "startupscript.sh"

`/modules/loadbalancer/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
```

```
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Reference to the webserver module
module "webserver" {
  source = "../webservers"
}

# Get a floating IP
resource "openstack_networking_floatingip_v2" "fip_1" {
  pool = "ntnu-internal"
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "loadbalancer" {
  name = var.instance_name
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }

  user_data = "${data.template_file.script.rendered}"
  depends_on = [module.webserver]
}

output "template_contents" {
  value = data.template_file.script.rendered
}

data "template_file" "script" {
  template = "${file(var.startup_script)}"
  vars = {
    IP = join(",", [for ip in module.webserver.instance_private_ip: ip])
  }
}

# Connect Floating IP to instance
resource "openstack_compute_floatingip_associate_v2" "fip_1" {
  floating_ip = "${openstack_networking_floatingip_v2.fip_1.address}"
  instance_id = "${openstack_compute_instance_v2.loadbalancer.id}"
}
```

Code listing C.6: Infrastructure 2: 3-Layer Architecture - Loadbalancer "main.tf"

`/modules/loadbalancer/variables.tf:`

```

# The file contains variable declaration and some default values.
```

```
# This file together with main.tf and terraform.tfvars provision and instantiate a
    infrastructure stack
# See teraform documentation for more details of syntax/code

# @ File variables.tf
# @ Author Alexander Damhaug, NTNU

# Define variables:


variable "instance_name" {
 type = string
 default = "loadbalancer"
 description = "Name of the instance"
}

variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx1.2c4r"
 description = "Flavor for the server"
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "startup_script" {
   type = string
   default = "./modules/loadbalancer/startupscript.sh"
   description = "startup script to be run on the instance"
}

variable "network" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}
```

Code listing C.7: Infrastructure 2: 3-Layer Architecture - Loadbalancer "variables.tf"

`/modules/loadbalancer/startupscript.sh:`

```
#!/bin/bash

# Retrieve the IP-addresses of the webservers
echo ${IP} | tr ',' '\n' >>~/ipaddresses.txt
IP_ADDRESSES=($(cat ~/ipaddresses.txt))

echo "startupscript started" >>~/status.txt

# pull the loadbalancing script from working directory
cd ~/
curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-api>" >> ~/

echo "Curled haproxy script" >>~/status.txt

# Run the host configuration script
~/bachelor-orange/code/scripts/hosts.sh

# Run the haproxy script
~/bachelor-orange/code/scripts/haproxy.sh

echo "Script sucessfully ran" >>~/status.txt
```

Code listing C.8: Infrastructure 2: 3-Layer Architecture - Loadbalancer "startupscript.sh"

`/modules/webservers/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "webserver" {
  count = var.instance_count
  name = var.instance_name
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }

  user_data = file(var.startup_script)
}
```

Code listing C.9: Infrastructure 2: 3-Layer Architecture - Webservers "main.tf"

/modules/webservers/outputs.tf:

```
# main.tf

output "instance_private_ip" {
  value = openstack_compute_instance_v2.webserver.*.access_ip_v4
}
```

Code listing C.10: Infrastructure 2: 3-Layer Architecture - Webservers "outputs.tf"

/modules/webservers/variables.tf:

```

# The file contains variable declaration and some default values.
# This file together with main.tf and terraform.tfvars provision and instantiate a
    infrastructure stack
# See teraform documentation for more details of syntax/code

# @ File variables.tf
# @ Author Alexander, NTNU

# Define variables:

variable "instance_count" {
  type = number
  default = 3
}

variable "instance_name" {
 type = string
 default = "webserver"
 description = "Name of the instance"
}

variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx1.2c4r"
 description = "Flavor for the server"
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}
```

```
39
40 variable "startup_script" {
41    type = string
42    default = "./modules/webservers/startupscript.sh"
43    description = "startup script to be run on the instance"
44 }
45
46 variable "network" {
47   type    = string
48   default = "default" # default network to be used
49 }
50
51 variable "security_groups" {
52   type    = list(string)
53   default = ["default"]  # Name of default security group
54 }
```

Code listing C.11: Infrastructure 2: 3-Layer Architecture - Webservers "variables.tf"

`/modules/webservers/startupscript.sh:`

```
1  #!/bin/bash
2
3  echo "startupscript started" >>~/status.txt
4
5  # Update ubuntu packages
6  sudo apt update && sudo apt upgrade
7
8  # install apache
9  sudo apt install apache2 -y
10
11 echo "Installed apache2" >>~/status.txt
12
13 # Install php with some modules:
14 sudo apt install php libapache2-mod-php php-mysql -y
15
16 # Install the remaining php packages using apt
17 sudo apt-get install graphviz aspell ghostscript clamav php-pspell php-curl php-gd
       php-intl php-mysql php-xml php-xmlrpc php-ldap php-zip php-soap php-mbstring
       git -y
18 echo "Installed php" >>~/status.txt
19
20 # Change max_input_vars to 5000 for the installation capabilities, in both GUI and
       CLI
21 echo "max_input_vars = 5000" >> /etc/php/8.1/cli/php.ini
22 echo "max_input_vars = 5000" >> /etc/php/8.1/apache2/php.ini
23
24 # restart apache:
25 sudo service apache2 restart
26 echo "Apache sucsessfully started" >>~/status.txt
27
28 # Move to /opt directory, and clone the Moodle Git repo
29 cd /opt
30 sudo git clone git://git.moodle.org/moodle.git
```

```
31
32 # Change to the Moodle directory and list the branches in the Moodle repository.
       Review the list choose the latest stable release
33 cd moodle
34 sudo git branch -a
35
36 # Track, and choose the desired git-branch
37 sudo git branch --track MOODLE_401_STABLE origin/MOODLE_401_STABLE
38 sudo git checkout MOODLE_401_STABLE
39
40 # Copy the contents of the Moodle repo to the running Apache service, and change
       the moderation of the moodle directory
41 sudo cp -R /opt/moodle /var/www/html/
42 sudo chmod -R 0777 /var/www/html/moodle
43 sudo mkdir /var/www/moodledata/
44
45 # Create the /var/www/moodledata directory and change owner and permissions
46 sudo chown -R www-data /var/www/moodledata
47 sudo chmod -R 0777 /var/www/moodledata
48
49 # Installing gluster FS
50
51 apt-get -y install glusterfs-server glusterfs-client
52 systemctl enable glusterd
53 systemctl start glusterd
54 echo "GlusterFS sucsessfully installed and started" >>~/status.txt
55
56 mkdir /moodle_brick
```

Code listing C.12: Infrastructure 2: 3-Layer Architecture - Webservers "startupscript.sh"

C.3 Infrastructure 3: Docker

```
3_code_docker
├── docker_code
│   ├── docker_moodle_db-setup
│   │   ├── Dockerfile
│   │   ├── README.md
│   │   └── install_database_php_script.sh
│   └── docker_www
│       ├── Dockerfile
│       ├── README.md
│       └── init.sh
└── infra_code
    ├── modules
    │   ├── database
    │   │   ├── main.tf
    │   │   ├── startupscript.sh
    │   │   └── variables.tf
    │   ├── docker
    │   │   ├── main.tf
    │   │   ├── outputs.tf
    │   │   ├── startupscript.sh
    │   │   └── variables.tf
    │   └── docker_loadbalancer
    │       ├── main.tf
    │       ├── startupscript.sh
    │       └── variables.tf
    └── main.tf
```

C.3.1 docker_code

`docker_code/docker_moodle_db-setup/Dockerfile:`

```
FROM ubuntu:22.04
#MAINTAINER TomArne
ENV DEBIAN_FRONTEND=noninteractive

RUN mkdir /moodledata
RUN chmod 777 /moodledata

RUN apt update
RUN apt install -y php-mbstring php-curl php-zip php-gd php-intl
RUN apt install -y apache2 libapache2-mod-php
RUN apt install -y php-xml php-mysql php-cli
RUN apt install -y git curl
RUN apt install -y php-ldap

```

```
RUN rm -rf /var/www/html/*
COPY ../moodle/ /var/www/html/moodle


RUN echo "max_input_vars = 5000" >> /etc/php/8.1/cli/php.ini

COPY docker_moodle_db-setup/install_database_php_script.sh /
EXPOSE 80
ENTRYPOINT ["/install_database_php_script.sh"]
```

Code listing C.13: Infrastructure 3: Docker - db-setup "Dockerfile"

`docker_code/docker_moodle_db-setup/install_database_php_script.sh:`

```
#!/bin/bash -x

curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-api> >> /var/
    www/html/moodle/config.php"

# Change --adminpass to the password created for the Admin user
/usr/bin/php /var/www/html/moodle/admin/cli/install_database.php --agree-license --
    adminpass='<your_generated_password>'
```

Code listing C.14: Infrastructure 3: Docker - db-setup "install_database_php_script.sh"

`docker_code/docker_moodle_db-setup/README.md:`

```
sudo docker build -t moodle_install_db_script:version-tag -f Dockerfile ..
```

Code listing C.15: Infrastructure 3: Docker - db-setup "README.md"

`docker_code/docker_www/Dockerfile:`

```
FROM ubuntu:22.04
ENV DEBIAN_FRONTEND=noninteractive

RUN mkdir /moodledata
RUN chmod 777 /moodledata

RUN apt update
RUN apt install -y php-mbstring php-curl php-zip php-gd php-intl
RUN apt install -y apache2 libapache2-mod-php
RUN apt install -y php-xml php-mysql php-cli
RUN apt install -y git curl
RUN apt install -y git curl
RUN apt install -y php-ldap

RUN rm -rf /var/www/html/*
COPY moodle /var/www/html/moodle

```

```
RUN echo "max_input_vars = 5000" >> /etc/php/8.1/cli/php.ini

COPY docker_www/init.sh /
EXPOSE 80
ENTRYPOINT ["/init.sh"]
```

Code listing C.16: Infrastructure 3: Docker - www-setup "Dockerfile"

`docker_code/docker_www/init.sh:`

```
#!/bin/bash -x

# get the php config file from repostory
curl --header "PRIVATE-TOKEN: <access-token>" "<link-to-gitlab-file-api>" >> /var/
    www/html/moodle/config.php


# Run crontab
/usr/bin/php /var/www/html/moodle/admin/cli/cron.php

echo "*/1 * * * * /usr/bin/php /var/www/html/moodle/admin/cli/cron.php >/dev/null"
    | crontab -u root -

#/usr/bin/php /var/www/html/moodle/admin/cli/install_database.php
/usr/sbin/apache2ctl -D FOREGROUND -k start
```

Code listing C.17: Infrastructure 3: Docker - www-setup "init.sh"

`docker_code/docker_www/README.md:`

```
sudo docker build -t moodle_www:version-tag -f Dockerfile ..
```

Code listing C.18: Infrastructure 3: Docker - www-setup "README.md"

C.3.2 docker_Infra

`Infra_code/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Configure the OpenStack Provider
```

```
# Note that the clouds.yaml from openstack is located inside ‘‘.config/openstack/‘‘
    folder
provider "openstack" {
    cloud = "openstack"
}
# initialize the database module
module "database" {
  source = "./modules/database/"
}
# initialize the loadbalancer module
module "loadbalancer" {
  source = "./modules/docker_loadbalancer/"
}
```

Code listing C.19: Infrastructure 3: Docker - "main.tf"

`Infra_code/modules/database/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "database" {
  name = var.instance_name
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }
  # script to be run upon creation
  user_data = file(var.startup_script)
}
```

Code listing C.20: Infrastructure 3: Docker - Database "main.tf"

`Infra_code/modules/database/startupscript.sh:`

```
#!/bin/bash

# inspierd by: https://www.linode.com/docs/guides/how-to-install-a-lamp-stack-on-
    ubuntu-22-04/
```

```
4  echo "startupscript started" >>~/status.txt
5
6  # Update ubuntu packages
7  sudo apt update && sudo apt upgrade
8
9  # Install Mysql web server
10 sudo apt install mysql-server -y
11
12 echo "Installed mysql" >>~/status.txt
13
14 # Generate a random password only root can access
15 pw=$(pwgen -s 24 1)
16 sudo bash -c "echo $pw >> password.txt"
17 sudo chmod 600 password.txt
18
19 # log in to the database as root
20 sudo mysql -u root
21
22 sudo mysql -u root -p"" -e "CREATE DATABASE moodle DEFAULT CHARACTER SET utf8mb4
       COLLATE utf8mb4_unicode_ci;"
23 sudo mysql -u root -p"" -e "CREATE USER 'admin'@'%' IDENTIFIED BY '$pw';" #Remember
        to change password!
24 sudo mysql -u root -p"" -e "GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE
       TEMPORARY TABLES,DROP,INDEX,ALTER ON moodle.* TO 'admin'@'%';"
25 sudo sed -i 's/bind-address.*/bind-address = 0.0.0.0/' /etc/mysql/mysql.conf.d/
       mysqld.cnf
26 sudo systemctl restart mysql
27
28 quit
29
30 pw=""
```

Code listing C.21: Infrastructure 3: Docker - Database "startupscript.sh"

`Infra_code/modules/database/variables.tf:`

```
1
2  # The file contains variable declaration and some default values.
3  # This file together with main.tf and terraform.tfvars provision and instantiate a
       infrastructure stack
4  # See teraform documentation for more details of syntax/code
5
6  # @ File variables.tf
7  # @ Author Alexander Damhaug, NTNU
8
9  # Define variables:
10 variable "instance_name" {
11  type = string
12  default = "database"
13  description = "Name of the instance"
14
15 }
16
```

```
variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx1.2c2r"
 description = "Flavor for the server"
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "startup_script" {
   type = string
   default = "./modules/database/startupscript.sh"
   description = "startup script to be run on the instance"
}

variable "network" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}
```

Code listing C.22: Infrastructure 3: Docker - Database "variables.tf"

`Infra_code/modules/docker/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "docker" {
  name = var.instance_name
  image_name = var.instance_image
```

```
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }
  # Script to run at creation
  user_data = file(var.startup_script)
}
```

Code listing C.23: Infrastructure 3: Docker - Server "main.tf"

`Infra_code/modules/docker/outputs.tf:`

```
# main.tf

output "instance_private_ip" {
  value = openstack_compute_instance_v2.docker.*.access_ip_v4
}
```

Code listing C.24: Infrastructure 3: Docker - Server "outputs.tf"

`Infra_code/modules/docker/startupscript.sh:`

```
#!/bin/bash

echo "startupscript started" >>~/status.txt

# Update ubuntu packages
sudo apt update && sudo apt upgrade

# Install docker and dependencies from https://docs.docker.com/engine/install/
    ubuntu/
sudo apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -m 0755 -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /
    etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
    https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
    null

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
    docker-compose-plugin
```

```
24
25 echo "Installed docker and dependencies" >>~/status.txt
26
27 sudo service docker start
28
29 # Allow pull from http docker registry
30 echo "{
31 \"insecure-registries\" : [\"http://<registry_ip>:5000\"]
32 }" >> /etc/docker/daemon.json
33
34 sudo service docker restart
35
36 # Create the moodledata directory
37 mkdir /moodledata
38 sudo chown -R www-data //moodledata
39 sudo chmod -R 0777 /moodledata
40
41 # note, the accsess to the moodledata directory and moodle folder should be
       restricted and follow Moodle's security recomendations:
42 #     https://docs.moodle.org/402/en/Security_recommendations
43
44 echo "startupscript done" >>~/status.txt
```

Code listing C.25: Infrastructure 3: Docker - Server "startupscript.sh"

`Infra_code/modules/docker/variables.tf:`

```
1  # The file contains variable declaration and some default values.
2  # This file together with main.tf and terraform.tfvars provision and instantiate a
       infrastructure stack
3  # See teraform documentation for more details of syntax/code
4
5  # @ File variables.tf
6  # @ Author Alexander, NTNU
7
8  # Define variables:
9
10 variable "instance_name" {
11  type = string
12  default = "docker"
13  description = "Name of the instance"
14 }
15
16 variable "instance_image" {
17  type = string
18  default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
19  description = "Image for the server"
20 }
21
22 variable "instance_flavor" {
23  type = string
24  default = "gx1.2c2r"
25  description = "Flavor for the server"
```

```
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "startup_script" {
   type = string
   default = "./modules/docker/startupscript.sh"
   description = "startup script to be run on the instance"
}

variable "network" {
  type    = string
  default = "default" # default network to be used
  description = "default network to be used"
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}
```

Code listing C.26: Infrastructure 3: Docker - Server "variables.tf"

`Infra_code/modules/docker_loadbalancer/main.tf:`

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Reference to the docker module
module "docker" {
  source = "../docker"
}

# Get a floating IP
resource "openstack_networking_floatingip_v2" "fip_1" {
  pool = "ntnu-internal"
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "docker_loadbalancer" {
  name = var.instance_name
```

```
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }

  user_data = "${data.template_file.script.rendered}"
  depends_on = [module.docker]
}

output "template_contents" {
  value = data.template_file.script.rendered
}

data "template_file" "script" {
  template = "${file(var.startup_script)}"
  vars = {
    IP = join(",", [for ip in module.docker.instance_private_ip: ip])
  }
}

# Connect Floating IP to instance
resource "openstack_compute_floatingip_associate_v2" "fip_1" {
  floating_ip = "${openstack_networking_floatingip_v2.fip_1.address}"
  instance_id = "${openstack_compute_instance_v2.docker_loadbalancer.id}"
}
```

Code listing C.27: Infrastructure 3: Docker - Loadbalancer "main.tf"

`Infra_code/modules/docker_loadbalancer/startupscript.sh:`

```
#!/bin/bash

# Retrieve the IP-addresses of the webservers
echo ${IP} | tr ',' '\n' >>~/ipaddresses.txt
IP_ADDRESSES=($(cat ~/ipaddresses.txt))

echo "startupscript started" >>~/status.txt

# clone working directory
cd ~/
git clone https://private-token:<access-token>@<link-to-gitlab>

echo "cloned git-repo" >>~/status.txt

# Run the host configuration script
~/bachelor_orange/code/scripts/hosts.sh

# Run the haproxy script
~/bachelor_orange/code/scripts/docker_haproxy.sh
```

```
20
21
22 echo "Script sucessfully ran" >>~/status.txt
```

Code listing C.28: Infrastructure 3: Docker - Loadbalancer "startupscript.sh

`Infra_code/modules/docker_loadbalancer/variables.tf:`

```
1  # The file contains variable declaration and some default values.
2  # This file together with main.tf and terraform.tfvars provision and instantiate a
       infrastructure stack
3  # See teraform documentation for more details of syntax/code
4
5  # @ File variables.tf
6  # @ Author Alexander Damhaug, NTNU
7
8  # Define variables:
9
10
11 variable "instance_name" {
12  type = string
13  default = "docker_Loadbalancer"
14  description = "Name of the instance"
15 }
16
17 variable "instance_image" {
18  type = string
19  default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
20  description = "Image for the server"
21 }
22
23 variable "instance_flavor" {
24  type = string
25  default = "gx1.2c2r"
26  description = "Flavor for the server"
27 }
28
29 variable "instance_key_pair" {
30  type = string
31  default = "default"
32  description = "SSH key to be used to connect to the instance"
33 }
34
35 variable "startup_script" {
36    type = string
37    default = "./modules/docker_loadbalancer/startupscript.sh"
38    description = "startup script to be run on the instance"
39 }
40
41 variable "network" {
42   type    = string
43   default = "default" # default network to be used
44 }
```

```
45
46 variable "security_groups" {
47   type    = list(string)
48   default = ["default"]  # Name of default security group
49 }
```

Code listing C.29: Infrastructure 3: Docker - Loadbalancer "variables.tf

C.4 Infrastructure 4: Docker Swarm

```
4_code_docker_swarm
├── modules
│   └── servers
│       ├── main.tf
│       ├── outputs.tf
│       ├── startupscript.sh
│       └── variables.tf
└── main.tf
```

main.tf:

```
1  # Define required providers
2  terraform {
3  required_version = ">= 0.14.0"
4    required_providers {
5      openstack = {
6        source  = "terraform-provider-openstack/openstack"
7        version = "~> 1.48.0"
8      }
9    }
10 }
11
12 # Configure the OpenStack Provider
13 # Note that the clouds.yaml from openstack is located inside ‘‘.config/openstack/‘‘
       folder
14 provider "openstack" {
15     cloud = "openstack"
16 }
17
18 module "servers" {
19    source = "./modules/servers/"
20 }
```

Code listing C.30: Infrastructure 4: Docker Swarm - "main.tf"

modules/servers/main.tf:

```
1  # Define required providers
2  terraform {
```

```
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Create a ubuntu instance
resource "openstack_compute_instance_v2" "server" {
  count = length(var.instance_name)
  name = var.instance_name[count.index]
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network
  }
  # Startup script
  user_data = file(var.startup_script)
}
```

Code listing C.31: Infrastructure 4: Docker Swarm - Servers "main.tf"

`modules/servers/outputs.tf:`

```
# main.tf

output "instance_private_ip" {
  value = openstack_compute_instance_v2.server.*.access_ip_v4
}
```

Code listing C.32: Infrastructure 4: Docker Swarm - Servers "outputs.tf"

`modules/servers/startupscript.sh:`

```
#!/bin/bash

# @author: Kristofer Lie, date: 19.04.2023
echo "startupscript started" >>~/status.txt

# Update ubuntu packages
sudo apt update && sudo apt upgrade

# use NTP to syncronise the internal server clocks
sudo apt-get install -y ntpdate
sudo ntpdate -b ntp.justervesenet.no

# Use crontab to frequently update the clock to follow NTP to assure accurate
    syncronisation
```

```
14 echo "*/10 * * * * root ntpdate -b ntp.justervesenet.no" >> /etc/crontab
15
16
17 # Install MariaDB Galera Cluster
18 sudo apt-get install mariadb-server mariadb-client galera-4 -y
19 echo "Installed mariadb-cluster galera" >>~/status.txt
20
21
22 # Install gluster FS
23 apt-get -y install glusterfs-server glusterfs-client
24 systemctl enable glusterd
25 systemctl start glusterd
26 echo "GlusterFS sucsessfully installed and started" >>~/status.txt
27
28 # Create folders to be used in gluster:
29 mkdir /moodle_brick
30 mkdir /moodledata
31
32 # From docker docks: https://docs.docker.com/engine/install/ubuntu/
33 # Install docker:
34 sudo apt-get update
35 sudo apt-get install -y \
36     ca-certificates \
37     curl \
38     gnupg \
39     lsb-release
40
41 sudo mkdir -m 0755 -p /etc/apt/keyrings
42 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /
       etc/apt/keyrings/docker.gpg
43
44 echo \
45   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
       https://download.docker.com/linux/ubuntu \
46   $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/
       null
47
48 sudo apt-get update
49 sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
       docker-compose-plugin
```

Code listing C.33: Infrastructure 4: Docker Swarm - Servers "startupscript.sh"

`modules/servers/variables.tf:`

```
1 # The file contains variable declaration and some default values.
2 # This file together with main.tf and terraform.tfvars provision and instantiate a
      infrastructure stack
3 # See teraform documentation for more details of syntax/code
4
5 # @ File variables.tf
6 # @ Author Alexander, NTNU
7
```

```
# Define variables:

variable "instance_name" {
 type = list(string)
 default = ["server1", "server2", "server3"]
 description = "Name of the instance"
}

variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx1.2c4r"
 description = "Flavor for the server"
}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "startup_script" {
   type = string
   default = "./modules/servers/startupscript.sh"
   description = "startup script to be run on the instance"
}

variable "network" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}
```

Code listing C.34: Infrastructure 4: Docker Swarm - Servers "variables.tf"

C.5 Additional Component 1 - Monitoring

```
5_code_other_components
└── Monitoring
    ├── main.tf
    ├── startupscript.sh
    └── variables.tf
```

main.tf:

```
# Define required providers
terraform {
required_version = ">= 0.14.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = "~> 1.48.0"
    }
  }
}

# Configure the OpenStack Provider
# Note that the clouds.yaml from openstack is located inside ‘‘.config/openstack/‘‘
      folder
provider "openstack" {
    cloud = "openstack"
}



### influxdb instance ###
resource "openstack_compute_instance_v2" "influxdb" {
  name = var.instance_name
  image_name = var.instance_image
  flavor_name = var.instance_flavor
  key_pair = var.instance_key_pair
  security_groups = var.security_groups

  network {
    name = var.network1
  }

  user_data = file(var.startup_script)
}




 # Get a floating IP
 resource "openstack_networking_floatingip_v2" "fip_1" {
   pool = var.network_pool_ntnu
 }


 # Connect Floating IP to instance
 resource "openstack_compute_floatingip_associate_v2" "fip_1" {
   floating_ip = "${openstack_networking_floatingip_v2.fip_1.address}"
   instance_id = "${openstack_compute_instance_v2.influxdb.id}"
 }



#output ip influxdb
```

```
output "influxdb_IP" {
  value = openstack_compute_instance_v2.influxdb.network[0].fixed_ip_v4
}
#output ip influxdb
output "influxdb_floating_IP" {
  value = openstack_networking_floatingip_v2.fip_1.address
}
```

Code listing C.35: Additional Component 1 - Monitoring

`startupscript.sh:`

```
#!/bin/bash

apt-get update
echo "Startupscript started" >> /home/ubuntu/status.txt


# Install influxdb, from: https://portal.influxdata.com/downloads/
wget -q https://repos.influxdata.com/influxdata-archive_compat.key

echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c influxdata-archive_compat.key' | sha256sum -c && cat influxdata-archive_compat.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg > /dev/null

echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sources.list.d/influxdata.list

apt-get update && sudo apt-get install influxdb2 -y

#start influxdb:
systemctl start influxdb


rm influxdata-archive_compat.key

echo "Influxdb installed and started" >> /home/ubuntu/status.txt


echo "check status with: sudo systemctl status influxdb" >> /home/ubuntu/status.txt

# create default user, org and bucket:
influx setup -u orange -p Orange2023& -b bucket -o Orange -r 0 -f # These should be env. variables.

echo "setup of influx compleate!" >> /home/ubuntu/status.txt
```

Code listing C.36: Additional Component 1 - Monitoring

`variables.tf:`

```

# The file contains variable declaration and some default values.
```

```
# This file together with main.tf and terraform.tfvars provision and instantiate a
    infrastructure stack
# See teraform documentation for more details of syntax/code

# @ File variables.tf
# @ Author Kristoffer Lie, NTNU


variable "network_pool" {
   type = string

   default = "default"

   description = "Public/floating netowrk pool"

}

variable "instance_key_pair" {
 type = string
 default = "default"
 description = "SSH key to be used to connect to the instance"
}

variable "network1" {
  type    = string
  default = "default" # default network to be used
}

variable "security_groups" {
  type    = list(string)
  default = ["default"]  # Name of default security group
}


variable "instance_image" {
 type = string
 default = "Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64"
 description = "Image for the server"
}

variable "instance_flavor" {
 type = string
 default = "gx3.4c4r"
 description = "Flavor for the server"
}

#### Influxdb ####

# Define variables:
variable "instance_name" {
 type = string
 default = "influxdb"
 description = "Name of the instance"

```

```
}

variable "startup_script" {
   type = string
   default = "./startupscript.sh"
   description = "startup script to be run on the instance"
}
```

Code listing C.37: Additional Component 1 - Monitoring

Appendix D

Testing result summaries

In this appendix, all the documents and reports from testing the different architectures with k6 are included. The reports are executive summaries of the the test results for the different architectures under load.

D.1 Single instance

EXECUTIVE SUMMARY - Single_instance 100VUs

PASS

KS

Status: PASS
Created: 19 May 2023 at 20:52
Started by: kristo11@stud.ntnu.no
VUs: 100 VUs
Duration: 11 min 30 sec
Load zones:

Max Throughput	HTTP Failures	Avg Response Time	95% Response Time
54 reqs/s	0 reqs	1231 ms	6079 ms

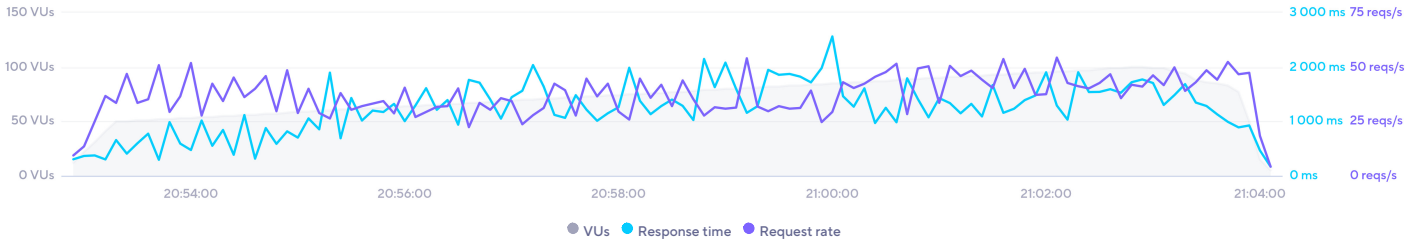
SUMMARY

This report summarizes a test run of the test "Single_instance 100VUs". It was performed on May 19, 2023 and is considered to be successful.

The test was configured to run up to **100 VUs** for 11 minutes 30 seconds. A total of **25 220 requests** were made with a max throughput of **54 reqs/s**. The sections below give a more detailed breakdown.

PERFORMANCE OVERVIEW

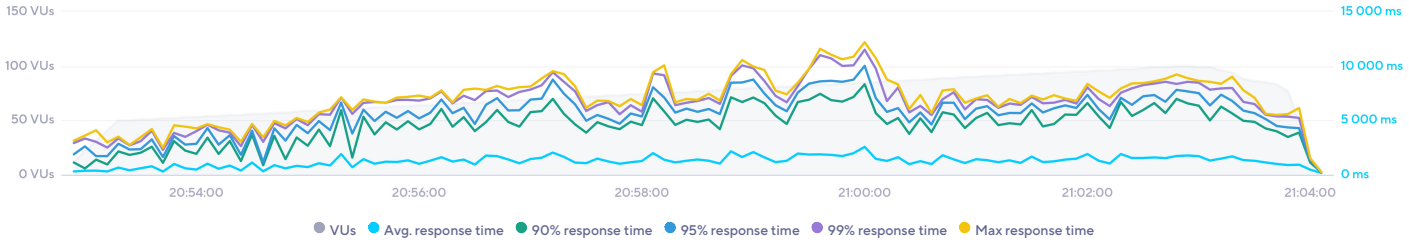
The average response time of the system being tested was **1 231 ms** and **25 220 requests** were made at an average request rate of **37 requests per second**.



TEST OVERVIEW

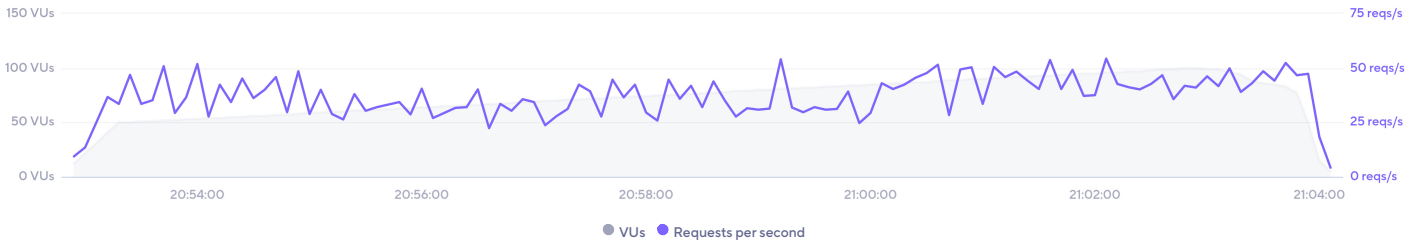
RESPONSE TIME

The maximum response time was **12 189 ms** at **85 VUs**. The average response time at the same point in time was **2 562 ms**, with 95% of requests taking less than **10 015 ms**.



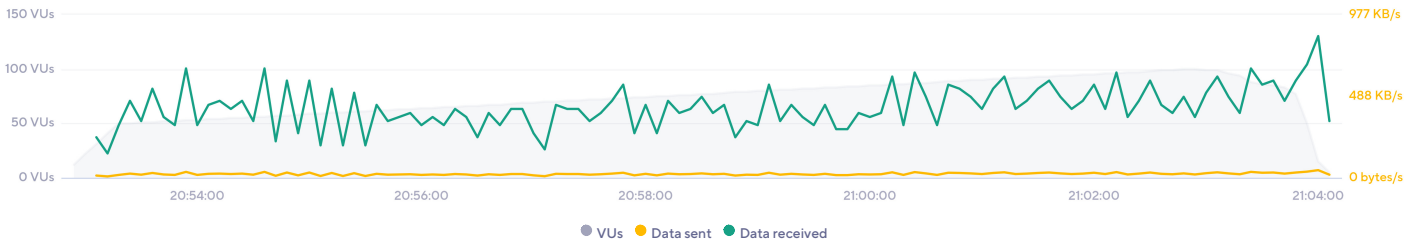
THROUGHPUT

The test had an overall average request rate of **37 reqs/s** peaking at **54 reqs/s** while running **96 VUs**.



BANDWIDTH

The amount of data sent peaked at **15 VUs**, sending **45.4 KB/s** of data. Data received had its peak at **15 VUs** with **848 KB/s** being received.



SLOWEST REQUESTS

There were requests to **12** unique URLs, with **25 220** different responses received. The slowest response had an average response time of **4 905 ms**.

URL	METHOD	STATUS	COUNT	MIN	AVG	95%	99%	MAX
http://10.212.172.53/moodle/course/view.php?id=2	GET	200	3880	389 ms	**4 905 ms**	8 319 ms	9 791 ms	12 189 ms
http://10.212.172.53/moodle/mod/forum/view.php?id=307	GET	200	1940	234 ms	**3 360 ms**	5 887 ms	7 039 ms	8 650 ms
http://10.212.172.53/moodle/mod/forum/discuss.php?d=89	GET	200	1940	137 ms	**1 207 ms**	2 095 ms	2 505 ms	3 192 ms
http://10.212.172.53/moodle/	GET	200	1940	85 ms	**741 ms**	1 335 ms	1 684 ms	2 145 ms
http://10.212.172.53/moodle/lib/ajax/service-nologin.php?info=c	GET	200	1940	14 ms	**188 ms**	369 ms	471 ms	664 ms
http://10.212.172.53/moodle/pluginfile.php/315/mod_resource/c	GET	200	1940	12 ms	**152 ms**	325 ms	411 ms	591 ms
http://10.212.172.53/moodle/lib/ajax/service-nologin.php?info=c	GET	200	1940	8 ms	**124 ms**	255 ms	326 ms	415 ms
http://10.212.172.53/moodle/lib/ajax/service-nologin.php?info=c	GET	200	1940	9 ms	**122 ms**	265 ms	357 ms	477 ms
http://10.212.172.53/moodle/mod/resource/view.php?id=301	GET	303	1940	34 ms	**117 ms**	225 ms	308 ms	449 ms
http://10.212.172.53/moodle/lib/ajax/service-nologin.php?info=c	GET	200	1940	8 ms	**113 ms**	232 ms	285 ms	376 ms

VOCABULARY

VUs
A Virtual User is a simulation of a real user making requests to the system. Multiple VUs are executed concurrently to simulate traffic to the website or API.

Response Time
The time from sending the request, processing it on the server side, to the time the client received the first byte.

Throughput
The amount of transactions the system under test can process, showing the capacity of the website or application.

Latency
The time that data sent or received spends on the wire, i.e. from the start of data being transmitted until all the data has been sent.

Checks
A check is an assertion that the system under test behaves correctly, e.g. that it returns the correct status code. They do not halt the execution of the test, but acts as a pass/fail metric.

Thresholds
Thresholds are a pass/fail criteria used to specify the performance expectations of the system under test.

ABOUT k6 CLOUD

k6 helps engineering teams prevent system failures and quickly deliver best-of-class applications. Our cutting-edge load testing platform brings cross-functional teams together to prevent reliability and scalability issues so that every application performs well. Developers, operations, and QA teams use our tools to automate testing and test earlier in the development process to bring high-quality products to market faster.

For more than 20 years, we have consulted businesses about load testing. We have spent the past 12 years developing state-of-the-art load and performance testing tools. 6,000+ customers — including Grafana, Microsoft, Carvana, and Olo — run millions of k6 tests every month. For more information, visit https://k6.io.

D.2 3 layer

EXECUTIVE SUMMARY - 3 layer

PASS

KS

Status: PASS
Created: 19 May 2023 at 22:22
Started by: kristo11@stud.ntnu.no
VUs: 100 VUs
Duration: 11 min 30 sec
Load zones:

Max Throughput	HTTP Failures	Avg Response Time	95% Response Time
25 reqs/s	4 012 reqs	1797 ms	1306 ms

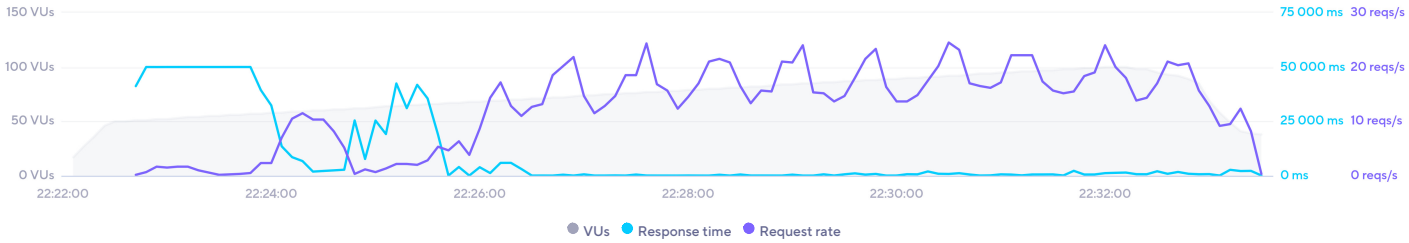
SUMMARY

This report summarizes a test run of the test "3 layer". It was performed on May 19, 2023 and is considered to be successful.

The test was configured to run up to **100 VUs** for 11 minutes 30 seconds. A total of **8 272 requests** were made with a max throughput of **25 reqs/s**. The sections below give a more detailed breakdown.

PERFORMANCE OVERVIEW

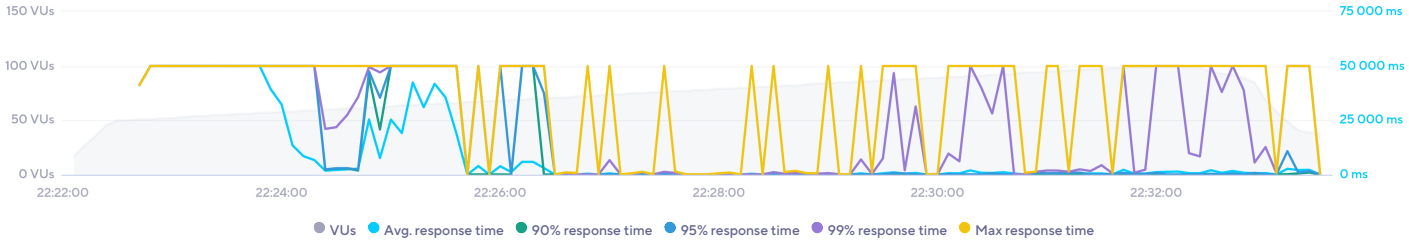
The average response time of the system being tested was **1 798 ms** and **8 272 requests** were made at an average request rate of **12 requests per second**.



TEST OVERVIEW

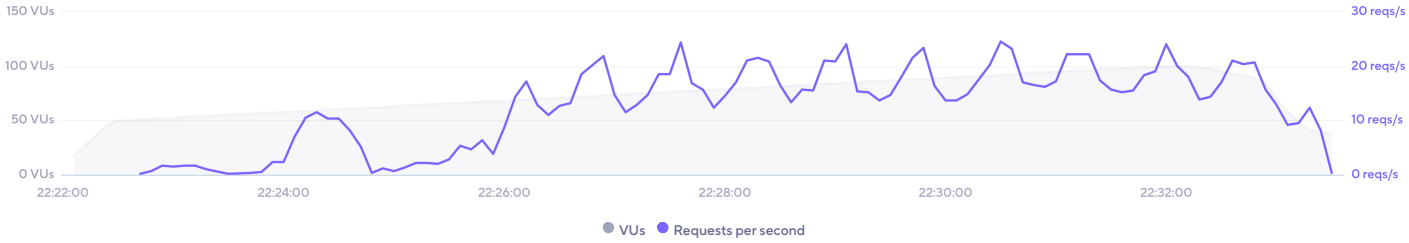
RESPONSE TIME

The maximum response time was **50 015 ms** at **64 VUs**. The average response time at the same point in time was **42 429 ms**, with 95% of requests taking less than **50 015 ms**.



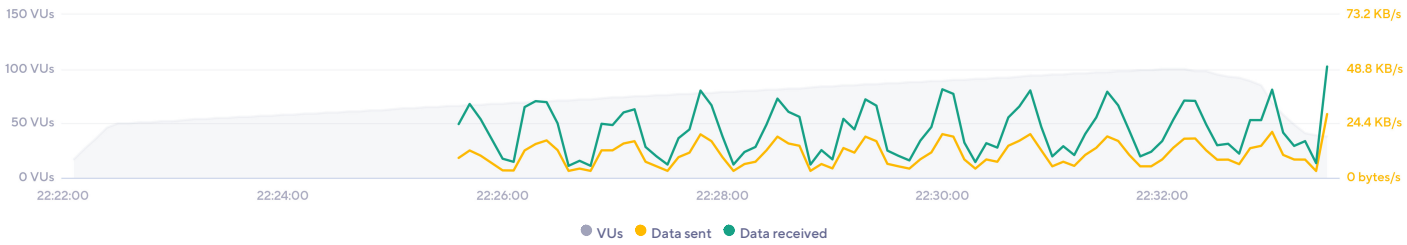
THROUGHPUT

The test had an overall average request rate of **12 reqs/s** peaking at **25 reqs/s** while running **92 VUs**.



BANDWIDTH

The amount of data sent peaked at **38 VUs**, sending **28.6 KB/s** of data. Data received had its peak at **38 VUs** with **49.9 KB/s** being received.



SLOWEST REQUESTS

There were requests to **16** unique URLs, with **8 272** different responses received. The slowest response had an average response time of **50 015 ms**.

URL	METHOD	STATUS	COUNT	MIN	AVG	95%	99%	MAX
http://10.212.175.203/moodle/course/view.php?id=2	GET	504	158	50 002 ms	**50 015 ms**	50 015 ms	50 015 ms	50 015 ms
http://10.212.175.203/moodle/mod/page/view.php?id=131	GET	504	39	50 004 ms	**50 008 ms**	50 008 ms	50 008 ms	50 008 ms
http://10.212.175.203/moodle/	GET	504	74	50 003 ms	**50 008 ms**	50 008 ms	50 008 ms	50 008 ms
http://10.212.175.203/moodle/	GET	200	3	41 205 ms	**46 938 ms**	49 800 ms	49 800 ms	49 800 ms
http://10.212.175.203/moodle/mod/resource/view.php?id=301	GET	303	77	1 359 ms	**2 261 ms**	3 007 ms	3 328 ms	3 840 ms
http://10.212.175.203/moodle/pluginfile.php/315/mod_resource/	GET	200	62	629 ms	**1 243 ms**	1 765 ms	1 939 ms	2 021 ms
http://10.212.175.203/moodle/lib/ajax/service-nologin.php?info:	GET	200	824	22 ms	**157 ms**	980 ms	1 479 ms	2 002 ms
http://10.212.175.203/moodle/lib/ajax/service-nologin.php?info:	GET	200	823	27 ms	**157 ms**	910 ms	1 290 ms	1 825 ms
http://10.212.175.203/moodle/lib/ajax/service-nologin.php?info:	GET	200	823	24 ms	**150 ms**	819 ms	1 291 ms	1 541 ms
http://10.212.175.203/moodle/lib/ajax/service-nologin.php?info:	GET	200	824	29 ms	**125 ms**	545 ms	760 ms	1 149 ms

VOCABULARY

VUs
A Virtual User is a simulation of a real user making requests to the system. Multiple VUs are executed concurrently to simulate traffic to the website or API.

Response Time
The time from sending the request, processing it on the server side, to the time the client received the first byte.

Throughput
The amount of transactions the system under test can process, showing the capacity of the website or application.

Latency
The time that data sent or received spends on the wire, i.e. from the start of data being transmitted until all the data has been sent.

Checks
A check is an assertion that the system under test behaves correctly, e.g. that it returns the correct status code. They do not halt the execution of the test, but acts as a pass/fail metric.

Thresholds
Thresholds are a pass/fail criteria used to specify the performance expectations of the system under test.

ABOUT k6 CLOUD

k6 helps engineering teams prevent system failures and quickly deliver best-of-class applications. Our cutting-edge load testing platform brings cross-functional teams together to prevent reliability and scalability issues so that every application performs well. Developers, operations, and QA teams use our tools to automate testing and test earlier in the development process to bring high-quality products to market faster.

For more than 20 years, we have consulted businesses about load testing. We have spent the past 12 years developing state-of-the-art load and performance testing tools. 6,000+ customers — including Grafana, Microsoft, Carvana, and Olo — run millions of k6 tests every month. For more information, visit https://k6.io.

D.3 Docker

EXECUTIVE SUMMARY - Docker 100VUs

PASS

KS

Status: PASS
Created: 19 May 2023 at 21:41
Started by: kristo11@stud.ntnu.no
VUs: 100 VUs
Duration: 11 min 30 sec
Load zones:

Max Throughput	HTTP Failures	Avg Response Time	95% Response Time
36 reqs/s	0 reqs	1393 ms	4927 ms

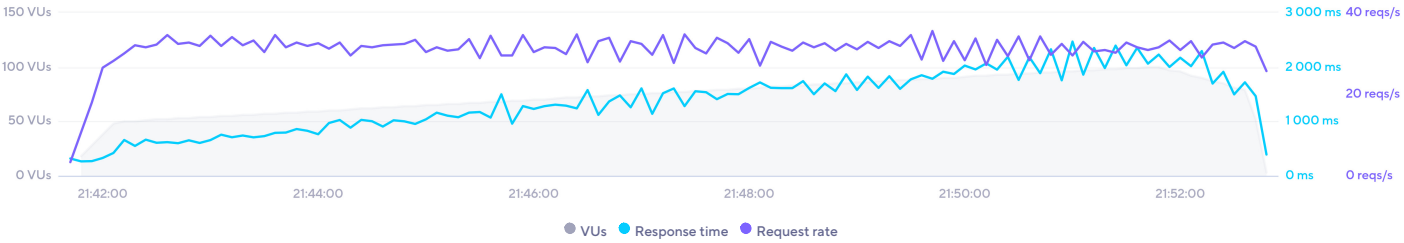
SUMMARY

This report summarizes a test run of the test "Docker 100VUs". It was performed on May 19, 2023 and is considered to be successful.

The test was configured to run up to **100 VUs** for 11 minutes 30 seconds. A total of **20 825 requests** were made with a max throughput of **36 reqs/s**. The sections below give a more detailed breakdown.

PERFORMANCE OVERVIEW

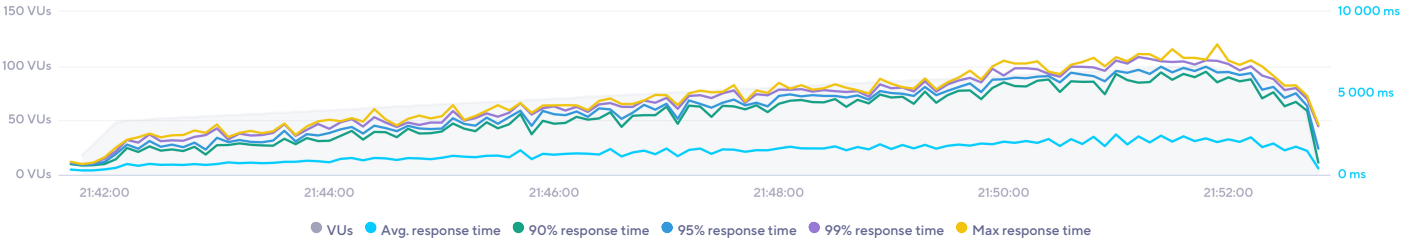
The average response time of the system being tested was **1 393 ms** and **20 825 requests** were made at an average request rate of **31 requests per second**.



TEST OVERVIEW

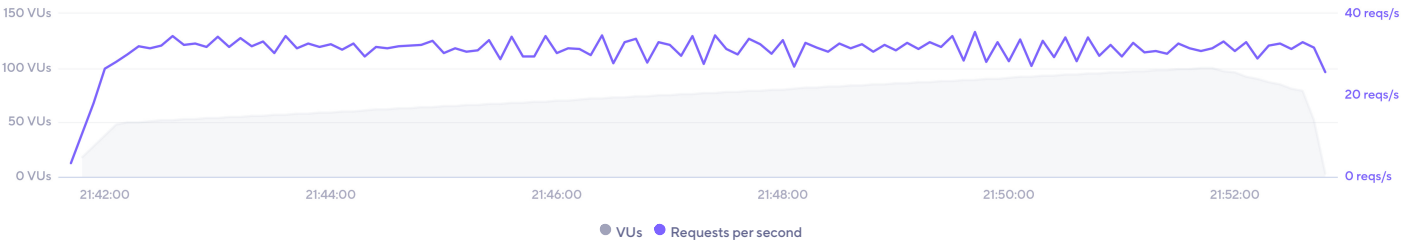
RESPONSE TIME

The maximum response time was **7 989 ms** at **97 VUs**. The average response time at the same point in time was **2 001 ms**, with 95% of requests taking less than **6 274 ms**.



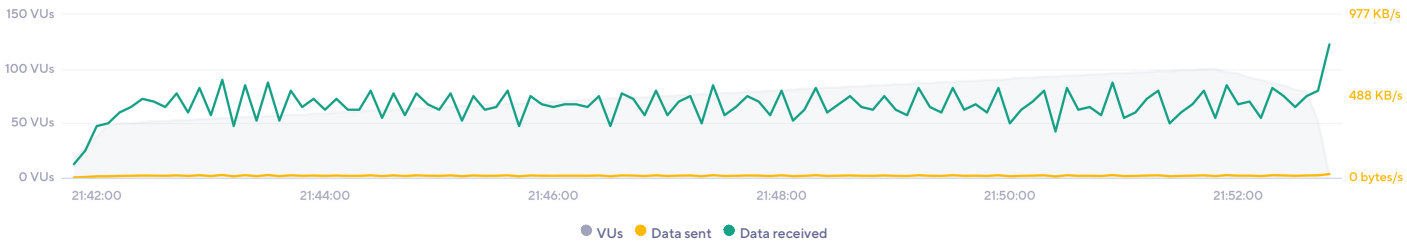
THROUGHPUT

The test had an overall average request rate of **31 reqs/s** peaking at **36 reqs/s** while running **89 VUs**.



BANDWIDTH

The amount of data sent peaked at **3 VUs**, sending **21.9 KB/s** of data. Data received had its peak at **3 VUs** with **798 KB/s** being received.



SLOWEST REQUESTS

There were requests to **6** unique URLs, with **20 825** different responses received. The slowest response had an average response time of **3 970 ms**.

URL	METHOD	STATUS	COUNT	MIN	AVG	95%	99%	MAX
http://10.212.170.7/moodle/course/view.php?id=2	GET	200	2975	493 ms	**3 970 ms**	6 399 ms	6 951 ms	7 989 ms
http://10.212.170.7/moodle/mod/forum/view.php?id=307	GET	200	2975	342 ms	**2 842 ms**	4 767 ms	5 183 ms	5 848 ms
http://10.212.170.7/moodle/mod/forum/discuss.php?d=80	GET	200	2975	182 ms	**1 133 ms**	1 895 ms	2 175 ms	2 665 ms
http://10.212.170.7/moodle/	GET	200	5950	103 ms	**726 ms**	1 199 ms	1 375 ms	1 631 ms
http://10.212.170.7/moodle/mod/resource/view.php?id=301	GET	303	2975	83 ms	**231 ms**	343 ms	406 ms	551 ms
http://10.212.170.7/moodle/pluginfile.php/315/mod_resource/co	GET	200	2975	23 ms	**125 ms**	228 ms	298 ms	441 ms

VOCABULARY

VUs
A Virtual User is a simulation of a real user making requests to the system. Multiple VUs are executed concurrently to simulate traffic to the website or API.

Response Time
The time from sending the request, processing it on the server side, to the time the client received the first byte.

Throughput
The amount of transactions the system under test can process, showing the capacity of the website or application.

Latency
The time that data sent or received spends on the wire, i.e. from the start of data being transmitted until all the data has been sent.

Checks
A check is an assertion that the system under test behaves correctly, e.g. that it returns the correct status code. They do not halt the execution of the test, but acts as a pass/fail metric.

Thresholds
Thresholds are a pass/fail criteria used to specify the performance expectations of the system under test.

ABOUT k6 CLOUD

k6 helps engineering teams prevent system failures and quickly deliver best-of-class applications. Our cutting-edge load testing platform brings cross-functional teams together to prevent reliability and scalability issues so that every application performs well. Developers, operations, and QA teams use our tools to automate testing and test earlier in the development process to bring high-quality products to market faster.

For more than 20 years, we have consulted businesses about load testing. We have spent the past 12 years developing state-of-the-art load and performance testing tools. 6,000+ customers — including Grafana, Microsoft, Carvana, and Olo — run millions of k6 tests every month. For more information, visit https://k6.io.

D.4 Docker Swarm

EXECUTIVE SUMMARY - Docker Swarm 50VUs

PASS

KS

Status: PASS
Created: 8 May 2023 at 09:45
Started by: kristo11@stud.ntnu.no
VUs: 50 VUs
Duration: 11 min 30 sec
Load zones:

Max Throughput	HTTP Failures	Avg Response Time	95% Response Time
4 reqs/s	357 reqs	31642 ms	50020 ms

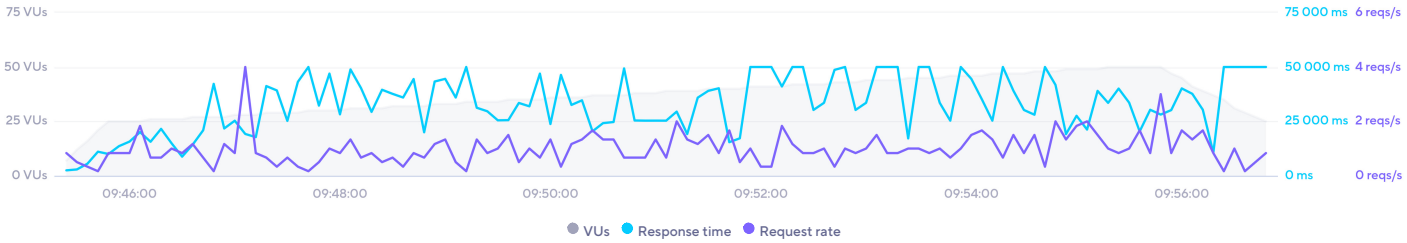
SUMMARY

This report summarizes a test run of the test "Docker Swarm 50VUs". It was performed on May 8, 2023 and is considered to be successful.

The test was configured to run up to **50 VUs** for 11 minutes 30 seconds. A total of **687 requests** were made with a max throughput of **4 reqs/s**. The sections below give a more detailed breakdown.

PERFORMANCE OVERVIEW

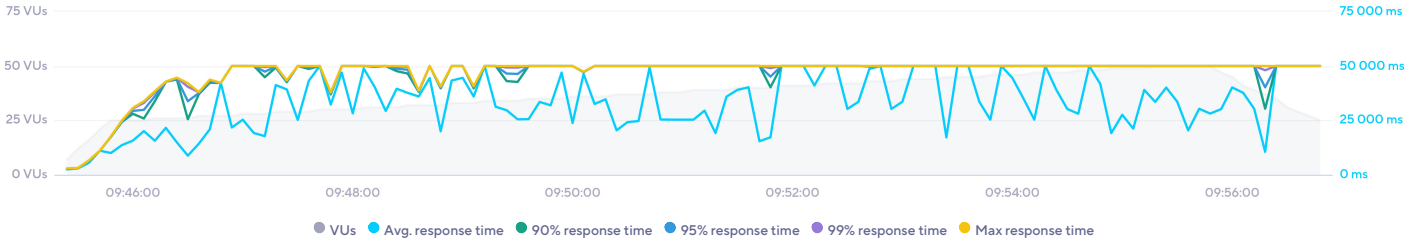
The average response time of the system being tested was **31 642 ms** and **687 requests** were made at an average request rate of **1 requests per second**.



TEST OVERVIEW

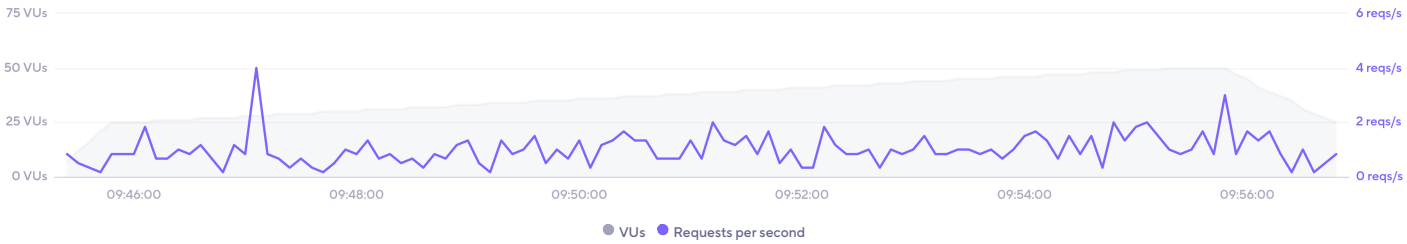
RESPONSE TIME

The maximum response time was **50 021 ms** at **34 VUs**. The average response time at the same point in time was **25 485 ms**, with 95% of requests taking less than **46 347 ms**.



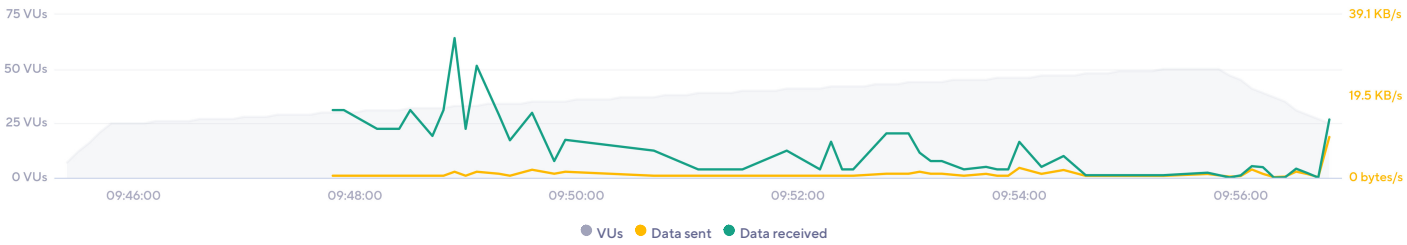
THROUGHPUT

The test had an overall average request rate of **1 reqs/s** peaking at **4 reqs/s** while running **28 VUs**.



BANDWIDTH

The amount of data sent peaked at **25 VUs**, sending **9.76 KB/s** of data. Data received had its peak at **33 VUs** with **33.4 KB/s** being received.



SLOWEST REQUESTS

There were requests to **11** unique URLs, with **687** different responses received. The slowest response had an average response time of **50 015 ms**.

URL	METHOD	STATUS	COUNT	MIN	AVG	95%	99%	MAX
http://10.212.169.172/moodle/mod/forum/discuss.php?d=255	GET	504	66	50 004 ms	**50 015 ms**	50 015 ms	50 015 ms	50 015 ms
http://10.212.169.172/moodle/course/view.php?id=3	GET	504	91	50 003 ms	**50 014 ms**	50 014 ms	50 014 ms	50 014 ms
http://10.212.169.172/moodle/	GET	504	108	50 004 ms	**50 010 ms**	50 010 ms	50 010 ms	50 010 ms
http://10.212.169.172/moodle/mod/forum/view.php?id=921	GET	504	91	50 003 ms	**50 009 ms**	50 009 ms	50 009 ms	50 009 ms
http://10.212.169.172/moodle/mod/forum/view.php?id=921	GET	200	6	36 307 ms	**43 098 ms**	46 870 ms	47 286 ms	47 390 ms
http://10.212.169.172/moodle/mod/forum/discuss.php?d=255	GET	200	15	34 757 ms	**41 344 ms**	49 349 ms	49 720 ms	49 813 ms
http://10.212.169.172/moodle/course/view.php?id=3	GET	200	18	17 620 ms	**38 817 ms**	48 629 ms	49 742 ms	50 021 ms
http://10.212.169.172/moodle/	GET	200	77	2 226 ms	**28 798 ms**	46 949 ms	47 585 ms	47 585 ms
http://10.212.169.172/moodle/	GET	500	1	966 ms	**966 ms**	966 ms	966 ms	966 ms
http://10.212.169.172/moodle/mod/resource/view.php?id=903	GET	303	107	347 ms	**876 ms**	1 133 ms	1 204 ms	1 248 ms

VOCABULARY

VUs
A Virtual User is a simulation of a real user making requests to the system. Multiple VUs are executed concurrently to simulate traffic to the website or API.

Response Time
The time from sending the request, processing it on the server side, to the time the client received the first byte.

Throughput
The amount of transactions the system under test can process, showing the capacity of the website or application.

Latency
The time that data sent or received spends on the wire, i.e. from the start of data being transmitted until all the data has been sent.

Checks
A check is an assertion that the system under test behaves correctly, e.g. that it returns the correct status code. They do not halt the execution of the test, but acts as a pass/fail metric.

Thresholds
Thresholds are a pass/fail criteria used to specify the performance expectations of the system under test.

ABOUT k6 CLOUD

k6 helps engineering teams prevent system failures and quickly deliver best-of-class applications. Our cutting-edge load testing platform brings cross-functional teams together to prevent reliability and scalability issues so that every application performs well. Developers, operations, and QA teams use our tools to automate testing and test earlier in the development process to bring high-quality products to market faster.

For more than 20 years, we have consulted businesses about load testing. We have spent the past 12 years developing state-of-the-art load and performance testing tools. 6,000+ customers — including Grafana, Microsoft, Carvana, and Olo — run millions of k6 tests every month. For more information, visit https://k6.io.

Appendix E

Project plan

In this appendix the project plan written in the planning phase of the project is included. All the planning prior to the project start, including goals and limitations, scope, and project organization is described.

NTNU
Institutt for informasjonssikkerhet
og kommunikasjonsteknologi

DCSG2900 - Bachelor thesis, Bachelor in digital infrastructure and cybersecurity

Project plan

Authors:
Tom Arne Brandvold
Alexander Damhaug
Kristoffer Lie
Benjamin Holhjem

Date: 01.02.2023

Table of Contents

List of Figures

List of Tables

1 Goals and limitations

1.1 Background

Orange Business Services is a local and global managed service provider, providing cloud services, digital workplace and data-driven solutions. They consist of over 28 500 employees globally with over 2 600 cloud experts and over 70 data-centers on 5 continents. The previous company Basefarm was bought by the french telecom provider Orange in 2018, and now exists as Digital Services Europe, a division of Orange Business Services covering the Nordic region, where this project was initiated [5].

Orange Business Services is growing and seeing a raising demand for a dedicated learning platform for training their technicians. The ultimate goal of this project is to setup and configure the infrastructure required to host Moodle (a free and open-source learning management system [6]). The underlying infrastructure must meet the company's requirements for uptime, security and capacity.

1.2 Project goals

Result goals

1. The main goal of the project is to setup a robust, scalable and secure infrastructure that is able to host Moodle for use as an internal learning platform for Orange Business Services.
2. Another goal is to develop documentation at a quality that makes the further job for Orange Business Services easier when they want to implement our solution.

Business goals

3. The business goals of the project is that implementing our solution should greatly reduce the time and resources needed to train technicians.
4. Our work should also make it easier for Orange Business Services to implement our solution, and reduce the resources needed to get the learning platform up and running.
5. Our solution should also be modifiable by Orange Business Services so that they can make the necessary changes to get the platform ready for implementation and use in their workplace.

Learning outcome

6. The learning outcome of the project is to take a deeper dive into setting up and configuring a robust and scalable infrastructure, and at the same time make sure that the infrastructure is secure.
7. We are going to learn a lot about Moodle, and web-hosting in general. And also Terraform, Tick Stack and other tools and services we will touch on during the project work.
8. Together with the technical learning outcomes we will also get real life experience with a bigger project, and the collaboration within a group and with the client. This opportunity allows us to develop good routines for best practices on how to plan, manage and collaborate on a project like this.
9. In the end we will hopefully also have gotten new insights that will help us reflect on our work during the project, and see if we could have improved something or done something in a different way.

1.3 Limitations

Technical limitations

1. Our product should be easily deployable by Orange Business Services
2. Our infrastructure should be based on sensible choices of services and technologies
3. The platform should be secured against unauthorized access, and tested for known vulnerabilities
4. The web server should be configured with redundancy and load balancing
5. The service and underlying infrastructure must be monitored using TICK stack
6. Necessary maintenance tasks and their affect on uptime should be documented
7. All choice of services and technologies should be justified, and configuration should be documented

Formal limitations

8. The project deadline is 22.05.23

2 Scope

2.1 Problem area

Orange Business Services AS, is seeking to integrate the training platform "Moodle" onto their infrastructure, to train technicians. The training platform needs to uphold the company's demands on uptime, security and capacity of a large amount of users.

To make this infrastructure platform, our focus is on writing Infrastructure as Code(IaC) and follow known best practices, including Kief Morris "Infrastructure as code "core practices [3]. Choices on which technologies and supported services will be justified, and all configurations made, will be thoroughly documented so the solution can be put in operation in a production-environment later.

2.2 Problem delimitation

The task given by Orange Business Service AS is fairly broad so it's important to set delimitation to our work. We work with the principle KISS (Keep it simple stupid). It's translated to work iterative with small and simple pieces, so it's easy to go back to previous versions when needed.

It is Orange Business Services AS that are responsible for the logical and technical content on Moodle. The project should however facilitate the use by integrating necessary plugins. Our work will be ran and tested on NTNU's onprem cloud solution SkyHiGh (running Openstack). This will distinguish from the client's preferred cloud platform.

Since SkyHiGh, is ran through NTNU's internal network and secured with a VPN, there will not be possible for external users to access the service. Since it isn't Orange Business Services AS need to have direct access with our solution, the service will be limited to only be accessed from NTNU's internal network.

There is no direct need for manual pen-testing, so the focus will not be directed to this. Although if we at a later time decide that there is more work that can be done, we will put manual pen-testing into consideration. When it comes to the different technologies we can choose from, the necessary choices will be taken after consideration throughout working with the project using the KISS model. Technologies like docker/Kubernetes, Terraform/ubuntu and other technologies will be determined after necessity.

2.3 Problem statement

The purpose with this assignment is to implement Moodle on a complete infrastructure platform with full resource access to manage the service. These are the following, requirements:

1. There will be configured a platform with full infrastructure based on reasonable choices of services and technologies
2. The platform must be secured against unauthorized users, and it is expected to run security tests on common vulnerabilities continuously.
3. The web-server and database, must be configured with redundancy and load-balancing to ensure up-time.
4. Simulated problems will also be tested, to make sure the system is able to be shielded from failure, or potential attacks
5. TICK Stack will be configured to monitor any operation issues that may appear.
6. Necessary operation tasks, and how it will affect the service's up-time will be documented.

7. Choice of services and technologies shall be justified, and all configurations shall be documented thoroughly, so the solution can be later used in production in an environment at a later time.

It's summed up by this problem statement:

Develop and configure a complete Moodle for infrastructure that ensures reliability, scalability and security using modern cloud technologies.

3 Project organization

For this project to be a success, it is important that the project is well organized. In this section we will look at how the group will look like in terms of its members different roles and the responsibilities related to those roles. We will also have a look at the routines and rules that should apply to the group.

3.1 Roles and responsibilities

To ensure productivity, efficiency and good communication for the group it may be beneficial to set a few roles for the group members.

Such roles may include:

- Group leader: Kristoffer
- Assisting Leader: Alexander
- Quality manager: Benjamin, Alexander
- Meeting coordinator: Tom Arne
- Responsible for compliance with deadlines: Kristoffer

The group leader: Will be responsible for the overall functionality of the group and can step in if there happens to be some form of conflict within the group. The group leader delegates work between group members if necessary. They will usually take lead in group meetings between Orange Business Services and our supervisor, but should get assistance from the other group members.

The Assisting Leader: Takes over responsibility from the group leader in case of absence, but will also assist the leader in decision making, conflict resolution, and work delegation if necessary.

The quality manager: Will make sure documents and work provided by the group is within specified requirements as well as the work holding the expected standard of quality for such project.

The meeting coordinator: Will assure that the group stay up to date with each other as well as other interests within the project. They will be responsible for the arrangement of internal meetings as needed throughout the project. They will also make sure that meetings held by the group will have a meeting leader and referent for the given meeting.

The responsible for compliance with deadlines: Will make sure that work on tasks will be started and finished in due time to get the task completed within the deadline. They will also assist the meeting coordinator in making sure meetings are planned and announced in time as needed.

3.2 Routines and group rules

Routines

- Core hours (11-15)
- Weekly meetings with supervisor
- Meetings with Orange every two weeks or as needed
- Log hours worked
- Diary

Group rules

§ 1. Everyone should be available during core hours (11-15) Monday - Friday.

§ 2. The rest of the expected work hours can be freely distributed before/after core hours, unless there is an agreed collaborative session.

§ 3. Everyone should attend all planned meetings unless otherwise agreed upon in advance.

§ 4. If a group member will not be able to attend a planned meeting or work session, the group should be informed in due time, so that if necessary the meeting can be rescheduled.

§ 5. Tasks will be delegated as we go and the delegation will be agreed upon by the whole group.

§ 6. Workload will ideally split equally between group members. If a task require less work than expected the member should try to help with other tasks.

§ 7. Specific tasks will be delegated using the Trello-board, and will give the group a core overview on a specific tasks progression.

§ 8. If the group or any of its members are not satisfied with the efforts of a specific member, they are required to communicate this early on with this person, in order to address and resolve the issue(s) at hand.

§ 9. Communication is always key to progression! Lack of communication leads to misunderstanding, which creates an unhealthy work environment, which can alter the quality of the project significantly. It is in every group-members best interest to manage situations where this might happen as we all are in this together.

Procedures for serious issues or violations of the rules

§ 10. If there is a lack of consensus in the group this should be attempted to be solved over a group meeting. If there is still a disagreement, the group leader gets the final say. If a group member is still unsatisfied, that individual must contact the supervisor (Erik) to consult on a resolvement in the group.

§ 11. For repeated violations, the group will firstly give an oral warning. If problems keep occurring, we will move onto written warnings which will be documented. If the following steps are unable to resolve the issue, § 12. will apply.

§ 12. If rules are not followed to the extend of disturbing the workflow for the rest of the group, the supervisor (Erik) should be involved.

4 Planning, follow-up and reporting

4.1 Main division of the project

Our project is a combination of infrastructure development and science. Our goal is to develop a secure, reliable and scalable infrastructure, based on our research and testing.

For this purpose, we will try to follow the agile framework called kanban. As a kanban board, we use Trello, where all tasks are placed . This process allows shortened cycles, and hopefully decrease the bottlenecks encountered by limiting the amount of "work in progress" [1]. Our modifications to kanban board are the following:

1. Specified own name of "workflow column" to be:
   (a) Todo
   (b) In progress
   (c) Need help
   (d) Open for improvement
   (e) To control and revision
   (f) Done
2. Our backlog is represented in the two columns "need help" and "Open for improvement"
3. The delivery point is the handover of technical documentation/product to Orange Business Services AS.
4. We will deviate from kanbans CI/CD by having a bigger focus on well written documentation.

4.2 Plan for status meetings and decision points

We divide our meetings in three: Meetings with Orange Service AS, Supervisor Erik Hjelmås and the group members. Client's meeting (Orange) will happens every odd week to discuss status of the project and challenges encountered. We have the possibility to book a meeting within short notice when needed. Meetings with the supervisor is set weekly, with the possibility to cancel if we have nothing to go over. The group members are available in "core time" between 11.00 and 15.00 on weekdays. In this core time every group member is available for discussions, meetings and other happenings throughout this project, like preparation for meetings with Orange Service AS. We will use the agile technique called "Pair programming", where two people works together on a topic [2].

When a decision (technical or non-technical) is encountered, the one who encounters it reads up on the "problem at stake". When ready, he calls for a meeting in the core time and discusses the findings with rest of the group This discussion should be well documented, containing both pros and cons. If we can not agree within the group, our supervisor will have a saying.

5 Organization of quality assurance

5.1 Documentation, storage, and source code

Documentation on design and implementation that mainly focus on our solution itself, will be stored in GitLab. Here we will focus on version control, and continuously deliver updates and integration's consecutively throughout the project when needed. Source code, supporting the implementation will also be stored in GitLab, and will be managed equally with the documentation.

Beside the documentation and source code, we will also work on more thoroughly reports and documents, that goes more in depth on the scientific part of our solution. For example the main report. Here we will answer different research questions, regarding the project and everything we've done. We will also have documentations regarding meetings between, supervisor/client, interviews with persons of interest, agreements with client/group-members on the project, project plan, and other potential reports regarding the administrative part of the project. This data will be stored on SharePoint.

5.2 Plan for development and testing

When it comes to the actual development for the solution, we will plan as we go depending on our achievements. As a fundamental standing point, we will develop a few instances and run the service on the machines and get that to work. Since we are provided the resources needed for the development through NTNU's SkyHiGh's onprem solution, we should be quite flexible for building the infrastructure for the project. We will use git for storing any developed solutions, and provide documentations for consecutively milestones hit. The documentations will be written in Markdown, and automation script will be created for working solutions.

Testing will be done continuously beside the actual development of the product, both for error-handling, but also for optimization and reliability. We will continuously deliver and integrate our solution, to create a good product when finished. Git will also be essential, for testing, and version control.

5.3 Tools

When it comes to a project as large as this, we must use lots of different tools, to assure we are able to deliver a good product for our client. This is not only tools directed for our solution, but administrative tools that assure that we are able to work as efficiently as possible, and don't fall behind schedule on our work, or miss out on crucial issues that needs to be resolved.

Administrative tools

For the administrative tools, that helps us coordinate our resources, stay up to date with our work, or meetings with people of interests, will be specified in the following bullet list:

1. Discord: Digital communication platform for formal meetings between the group members, or chat-rooms for discussions/planning regarding the project. It's also used for live voice-communication when working on the project itself.
2. Slack: Similar to discord, but only used for contacting the client with consecutive issues or questions at hand.
3. Trello: A digital Kanban board like framework tool, to give us an efficient overview on what needs to be done. It shows who is responsible for a specific issue, and reminds us to implement the agile issues at hand.
4. Teams: A formal voice communication platform, used for having meetings with our client.

5. Signal: A messaging app, for communicating with group members
6. E-Mail: Used for communicating with Client/Supervisor, etc.
7. Excel (time sheet): Used for documenting a group member's time spent on any type work they've done related to the project.

Development tools

The tools used for developing the solution, and/or supported material for the project, is listed here:

1. Overleaf: A technical tool used to write comprehensive documents in LaTeX e.g project plan/main report.
2. Word: Similar to Overleaf, but is used for smaller and less, complex documents e.g project agreement.
3. Visual Studio Code: Mainly used for developing the solution itself, like documentation, source code, design and implementation. Here it is common for us to use languages like Markdown for documentation, and e.g bash for development.
4. Openstack: Infrastructure as a Service tool used for running instances needed for the project.
5. GitLab: Used for storing source code, and documentation of design and implementation of the solution.
6. OneDrive: Storage for any produced documents regarding the project that won't be published on GitLab.
7. ChatGPT: A highly advanced Artificial Intelligence chat-bot that can assist us with specific parts of the project, e.g code errors etc.

5.4 Risk analysis

This section will first identify the risks we may encounter during this project, then analyze the findings. Based on an assessment of each risk, we set the appropriate actions to reduce/manage the critical risks. To accomplish this, we follow NTNU's ROS assessment [4].

5.4.1 Values

The values in table 5.1 are based on what we as a group see as values during this project. All group members contribute with their knowledge and work. Our supervisor has knowledge of the process and will guide us in the right path. Orange Business Services AS is providing us with two technicians that will be useful for technical desertions/problems encountered. Documentation and source code is a critical part of our project delivery. Securing this is important for the final delivery.

Values	Assessment
Group members	4
Supervisor/client	3
Documentation	3
Source code	3

Table 1: Values in the risk analysis

5.4.2 Risks

In order to place impact and probability on risks, we have clarified different severity's of these measurements.

Grade	Values	Elaboration
4	Very likely	Happens every week
3	Probable	Happens every other week
2	Less likely	Once a month
1	Unlikely	Twice or less

Table 2: Probability

Grade	Values	Description
4	Severe	Events that critically compromise the progression of the project
3	Moderate	Events that hold up the progress of the project
2	Low	Events that has some affect on the progress of the project
1	Insignificant	Events that do not have noticeable affect on the process

Table 3: Impact

Table 4 shows the risk we think we may encounter during this project without any actions taken. Each risk has been given a probability and a impact, witch combined shows a total score. The actions to be takes is focused on the most critical risks at a score of 9 and 8.

No	Risks	Probability	Impact	Total score
1	Disagreements within group	Very likely	Low	8
2	Unclear tasks	Less likely	Moderate	6
3	Bad communication within group	Probable	Low	6
4	Lose communication with client/supervisor	Less likely	moderate	6
5	Absent group member	Less likely	Severe	8
6	Important technology can't be integrated	Probable	Low	6
7	Lose source code and/or documentation	Probable	Moderate	9
8	Delay in planed activities	Probable	Moderate	9

Table 4: Risks documented before actions are taken

Probability	Very likely		1		
	Probable		6	3, 7, 8	
	Less likely			2, 4	5
	Unlikely				
		Insignificant	Low	Moderate	Severe
		Impact			

Table 5: Risk matrix before actions

5.5 Risk management plan

Based on the identifying and analysis done in section 5.4 Risk analysis, we have made a series of actions to be taken (see table 6).

No	Action	Risk it affects
1	Clearly defined group rules	1, 3, 4
2	Weekly meetings (Supervisor, client, group)	1, 2, 3, 4, 5, 6, 8
3	Use Kanban and Trello board	2, 8
4	Keep important documents in SharePoint	7
5	Use a Version Control System (GIT)	7

Table 6: Actions to be taken

The actions in table 6 affects all risks in a beneficial way. With clearly defined rules together with weekly meetings, we lower the probability of risks related to the group. These meetings helps us ensure continuous communication with our supervisor/client. The use of Kanban and Trello makes it easier to know what's our work in progress and track how other's doing. Using a VCS like GIT together with SharePoint ensures that our work is saved in the cloud and gives us the possibility to roll back to a earlier versions.

No	Risks	Probability	Impact	Total score
1	Disagreements within group	Probable	Low	6
2	Unclear tasks	Less likely	Insignificant	2
3	Bad communication within group	Less likely	Low	4
4	Lose communication with client/supervisor	Unlikely	Low	2
5	Absent group member	Less likely	Moderate	6
6	Important technology can't be integrated	Probable	Insignificant	3
7	Lose source code and/or documentation	Unlikely	Moderate	3
8	Delay in planed activities	Less likely	Low	4

Table 7: Risks after actions are taken

Probability	Very likely				
	Probable	6	1		
	Less likely	2	3, 8	5	
	Unlikely		4	7	
		Insignificant	Low	Moderate	Severe
		Impact			

Table 8: Risk matrix after actions

After our actions are implemented, all the risks found are acceptable to us (see table 8).

6 Execution plan

6.1 Gantt chart

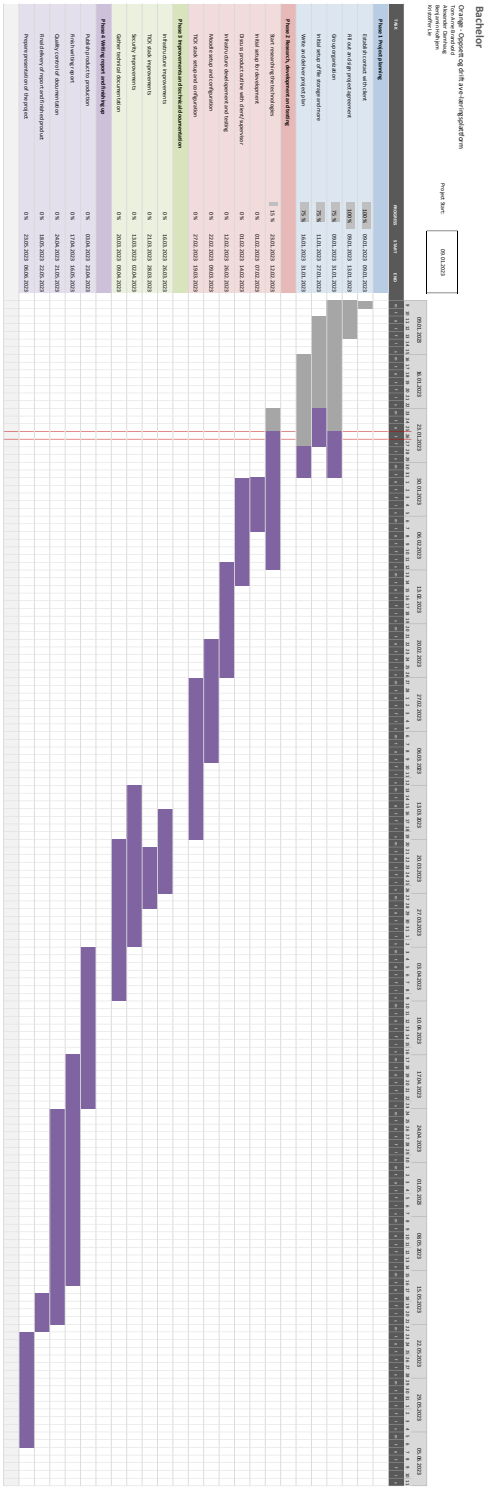


Figure 1: Gantt chart

6.2 Milestones and decision points

The Gantt chart above shows the progress of this project. While our milestones is set at specific dates, these milestones represents when we expect/want parts of the project to be done. By checking the milestones regularly, it's possible to control if we were on schedule or not. The milestones set only covers the solution work, but throughout this project, we will simultaneously document our findings, our working solutions, and write the main report.

No	Milestone	Date
1	Deliver Project plan	31.01.2023
2	Start research	01.02.2023
3	Start development and testing	12.02.2023
4	Our first MVP	24.02.2023
5	Start working on improvements	13.03.2023
6	Set the outline of the final report	01.04.2023
7	Publish to production	23.04.2023
8	Final delivery	22.05.2023

Table 9: Milestones

References

[1] Atlassian. *Kanban: How the kanban methodology applies to software development*. 2023. URL: https://www.atlassian.com/agile/kanban (visited on 18/01/2023).

[2] Alexander S. Gillis. *pair programming*. 2021. URL: https://www.techtarget.com/searchsoftwarequality/definition/Pair-programming (visited on 18/01/2023).

[3] Kief Morris. *Infrastructure as Code: Dynamic Systems for the Cloud Age*. O'reilly, 2020. ISBN: 9781098114671.

[4] NTNU. *Informasjonssikkerhet - risikovurdering*. 2023. URL: https://i.ntnu.no/wiki/-/wiki/Norsk/Informasjonssikkerhet+-+risikovurdering (visited on 23/01/2023).

[5] Orange Business Services. *Om oss*. URL: https://cloud.orange-business.com/no/om-oss-no/ (visited on 23/01/2023).

[6] Wikipedia contributors. *Moodle — Wikipedia, The Free Encyclopedia*. 2023. URL: https://en.wikipedia.org/w/index.php?title=Moodle&oldid=1134790859 (visited on 23/01/2023).

Appendix F

Contract

This appendix contains the standard agreement between the group and the client. The contract specifies copyright and ownership of the results developed in the bachelor's project.

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: Institutt for informasjonssikkerhet og kommunikasjonsteknologi
Veileder ved NTNU: Erik Hjelmås E-post: erik.hjelmas@ntnu.no Tlf: 93034446
Ekstern virksomhet: Orange Business Services Ekstern virksomhet sin kontaktperson, e-post og tlf.: Truls Enstad, truls.enstad@basefarm-orange.com, (+47) 417 60 909 Brage Veiby Reseland, brage.veiby.reseland@basefarm-orange.com, (+47) 934 43 092
Student: Tom Arne Haugen Brandvold Fødselsdato: 25.12.97 Tlf: 98120994 Epost: tabrandv@stud.ntnu.no
Student: Alexander Kristian Damhaug Fødselsdato: 04.07.01 Tlf: 94867404 Epost: alexander.damhaug@ntnu.no
Student: Benjamin Knutsen Holhjem Fødselsdato: 14.03.99 Tlf: 93263131 Epost: benjamin.holhjem@ntnu.no
Student: Kristoffer Lie Fødselsdato: 14.10.00 Tlf: 95406500 Epost: kristoffer.lie@ntnu.no

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	x
Prosjektoppgave	
Annen oppgave	

Startdato: 09.01.2023
Sluttdato: 22.05.2023

Oppgavens arbeidstittel er:
Oppsett og drift av e-læringsplattform

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

x	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss	
x	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

NTNU
NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

Instituttleder: Nils Kalstad Svendsen Dato: 01.0223 Åse Ringlund på vegne av Inst. leder
Veileder ved NTNU: Erik Hjelmås Dato:
Ekstern virksomhet: Dato: 13.01.2023
Student: Alexander Damhaug Dato: 13.01.2023
Student: Benjamin Holhjem Dato: 13/1-23
Student: Kristoffer Lie Dato: 13.01.23
Student: Tom Arne Haugen Brandvold Dato: 13.01.23

Appendix G

Meeting minutes

This appendix contains extracts of meeting minutes from group meetings, supervisor meetings and client meetings held during the work on the bachelor's project.

G.1 Client meetings

Møte 020223 m/Orange

torsdag 2. mars 2023 13:04

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	
Truls	Tilstede	

Referat:

- Info om at vi har satt litt bedre krav til infrastrukturene
  - Truls synes dette er bra
  - Infrastrukturene høres fornuftige ut
  - Viktig å se på problematikk med tidsbruk på automatisering og om det er nødvendig hele tida
- Performance er interessant
  - Øke kompleksiteten for å få hastighet opp, da mister man litt fordelene
- Godt i gang med problemløsning
  - Truls er imponert over at vi faktisk klarer å løse problemene som dukker opp
- LDAP
  - Ingen preferanse til AD eller openLDAP
  - Litt mer mot openLDAP
  - Kun ekstrapoeng å sette opp dette
  - Veldig basic innlogging
  - Litt overkill med full AD for kun en tjeneste
- TICK stack
  - Overvåke:
    - Hardware, bruk av disk ressurser
    - Cpu - men ikke fullt så nyttig om man ikke benytter det til skalering
    - Tjeneste
      - http på moodle forsida - se at den er oppe eller nede
      - Moodle tjenesten på serversiden, systemd tjenesten
    - Varsling om noe går ned
      - Hensiktsmessige alarmer
  - TIG stack
    - Kan være bedre å se på i forhold til at det kan være noe lettere

Møte 270423 m/Orange

torsdag 27. april 2023 14:00

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	
Truls	Tilstede	

Referat:

- Litt snakk rundt swarmen, og UDP åpninga
    - Ekspert hos Orange ville egentlig brukt Kubernetes i stedet for swarm
    - Tida strakk ikke til for å få til Kubernetes
    - Det er ingen problem fra Truls sin side, vi gjør det vi har fått tid til
        - Men kan nevne litt i rapporten at vi heller ville brukt Kubernetes
- Krissærn viser testinga av arkitekturene
    - Viser frem forskjellen i responstid
    - Truls synes responstid er relevant
    - Truls har ikke så mye spesifikke parametere å gi oss i forhold til testing
        - Tilbake til JMeter???
        - Kan bruke noe av dokumentasjonen rundt JMeter og Moodle, og ta med det inn i k6
- Informert om status på rapporten
    - Hovedfokuset vårt ligger der nå
    - Truls med tips til rapporten:
        - Orange ønsker løsning og funn vi har kommet frem til. Ønsker dette som vedlegg, teknisk dokumentasjon
        - Lettere for Orange å se på tekniske vedlegg enn å replikere tekniske løsninger ved å måtte lese hele rapporten
        - Burde snakke med Erik om dette, og spørre om det er greit at det er et vedlegg og ikke er bakt inn i rapporten
        - Det vi har i git repoet nå ser bra ut
- Tror vi bør bli bedre på å avslutte møter.. Det blir plutselig bare helt stille og så må Truls avslutte 🤦

G.2 Supervisor meetings

Møte med Erik 220223

onsdag 22. februar 2023 11:46

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	
Erik	Tilstede	

Gruppas status:

- Kommet igang med 2 nye arkitektuerer + Lastbalansering
    - Dog nye problemer stadig vis.
- Progresjon når deg gjelder helautomatisering
- Hatt møte med Eigil om Kubernetes
- Dine tanker om HELM chart?
- Bruke NB tech support!
- Manuelle private IP adresser?

Referat:

- Adressere hvor programvare kommer fra, i forhold til sikkerhet. Rapport.
- Ikke feil å hente inspirasjon fra kode hos uverifiserte kilder, men unngå å laste ned data fra uverifiserte kilder
- Se på om ansible kan bidra til at vi kan unngå startup skript
- Sikkerhetstesting av programvare
- Få konteinerne så små som mulig
    - Bruke image på konteinerne som kun har det nødvendige, slik at de blir kompakte
        - Alpine linux?
- Lage figurer og begynne å strukturere rapporten - kanskje se på dette neste
- Rapport - erfaringer i forhold til fremdriftsplanen

Møte med Erik 110523

torsdag 11. mai 2023 09:31

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	
Erik	Tilstede	

Agenda:

- Gå gjennom rapporten

Referat:

- Ikke sammenheng, må skrive ut hver vår kopi og lese gjennom så ser vi det
- Mangler litt referanser. Bruk referanselista til wikipedia - kiss en bok fra en amerikansk general
- Kutte noe i purpose, noe skrevet av gpt der 🤡
- 1.5 punktene er de samme som er definert tidligere - unødvendig. Kan formulere som prosjektmål, effektmål og læringsmål (se i prosjektplanen om vi har det fra før)
- Henvise til forprosjektet ihht verktøy, risiko, roller ++ (appendix B)
- Ikke "in the table under" men referer til figuren med nummer
- \clearpage - skyver resten av teksten over på neste side
- Ikke henvis til Erik i kap 2, bruke referanser til bøker eller vitenskapelige artikler. Wikipedia for virtual machine ligger en referanse til ieee
- Mangler en overskrift i kap 2
- Kutte første avsnitt i topic theory
- Kap 3: Holder kanskje med 1 skjermbilde. Beskriv tekstlig.
- Kap 3 virker litt rart. Sammenheng. Funksjonelle og ikke-funksjonelle krav (ytelse og sikkerhet, men vi sier noe om hardware og software krav). Hva vi skal nå, hvilke krav skal vi oppnå etter testing. Responstid skal være maks 5000 ms. Hente inn krav fra prosjektplanen. Liste med 7 punkt med krav burde kanskje vært operasjonelle krav.
- Kap 4 mangler refleksjon, hvorfor brukte vi kanban og hvilke alternativer hadde vi.
    - Gjennomføringsmetode - literature survey, eksperimenter bruk av gpt (eller enklere bruk av ai), dette er metoder.
    - Gpt må skrives så det fremkommer tydelig at vi får ut sannsynlig svar på den teksten vi skriver inn. Kun statistisk ikke noen kunnskapsbase, svarer med ord som det er statisk sannsynlig at er et svar på det man putter inn.
    - Ingen referanser i kapittelet
- Kap 5 mangler noe i forhold til det man eksakt skal teste. Hva vi skal teste burde stå i kravspec kapittelet.
    - 5.1 flytt til forrige kapittel
    - Hva linux er trenger vi ikke forklare
    - Blir plutselig veldig detaljert om moodle, brå overgang, bør se om vi kan omstrukturerer det
    - Mangler henvisning til figurer i teksten
    - Rettskrive noe
    - Drøfte et sted hvorfor vi bruker terraform og alternativene (burde stå tidligere enn kap 5 kanskje)
    - Drøfte verktøy i alle arkitekturer først, så kommer arkitekturene. Man vil ikke lese verktøyene brukt i mellom hver arkitektur, men vil ha mer sammenheng i arkitekturene.
        - Kort om alle relevante teknologier man trenger kjennskap til
        - Hver arkitektur, hvorfor vi har valgt den og hvordan den blir implementert, ved

hjelp av disse verktøyene…
    - Skrive tall 1 til 10 med bokstaver (ikke 3-layer for det skal stå slik)
    - Docker er kanskje litt for detaljert, men ikke noe stort poeng
    - Hvilke avhengigheter har man, hvilke må kjøre i sekvens. Tabellen i docker sier ikke noe om det. Avhengighetene mellom må komme tydeligere frem
    - Må ikke stå et eneste forslag til passord i rapporten, mulig løsning tok Tomærn bilde av når Erik skrev det på tavla.
    - Liten bommert med ei pil på swarm figuren
    - Skjermbilder uten mørk bakgrunn - prøve å bytte til hvit bakgrunn der det er mulig
    - Leseflyt: plutselig kommer det mye k6 kode
- Kap 8 alle endringer og problemer vi har møtt må ikke fremstå som klaging. Henvis til risikoanalysen og hvorfor vi ikke har det med i risikoanalysen. Må spesifisere helt spesifikt hvorfor enkelte steder, hvorfor vi ikke hadde tid, og hvorfor vi ikke hadde planlagt mitigation for et slikt problem. Også flytte noe fra further work til senere i rapporten.

G.3 Group meetings

Møte 270223

mandag 27. februar 2023 11:54

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	

Agenda:

- Møterom
- Status
- Hva er vårt felles mål?
- Hvem jobber med hva?
- Plan for uka
- Hva må gjøres?
- Div
    - Git
    - TICK stack
    - Toggl

Referat:

- Har booka 3 uker fram. Neste uke blir det et møte på tirsdag, de to neste ukene er T117 reservert hver dag fra 08 til 15.
- Folk er litt stuck, Alexander og Kristoffer har kjørt seg litt fast. Alexander har fått til ip adresser og flere webservere, er stuck på glusterfs, men vi burde se om vi kan unngå å bruke det. Tom Arne får ikke til db, så får heller ikke testa om ting funker på docker. Benjamin hadde dårlig tid forrige uke, skal få sett på rapport og figurer. Kristoffer har fungerende lamp-stack på kub, noe feil som må rettes.
- Motivasjonen er litt laber hos noen, vi må prøve å få satt litt mer klare målsettinger og retningslinjer.
- Automatisering er ikke helt nødvendig, men ønskelig i konfigurasjon av infrastrukturen. Slik som Alex holder på nå er kanskje ikke helt nødvendig, men det er viktige erfaringer i forhold til rapport og anbefaling av en type infrastruktur over en annen.
- Må forholde oss enda mer til KISS, slik at vi ikke overkompliserer ting.
- Prioriteringene i Alex sitt hode:
    1. Robust og skalerbar
    2. Sikkerhet
    3. Automatisering
- Alle er enig om at vi må begrense automatisering nå og sørge for å få infrastrukturen robust og skalerbar.
- Mål: sette oppe en robust og skalerbar infrastruktur.
    - 2 skalerbare infrastrukturer - Docker Swarm og Kubernetes
    - (Ikke skalerbare infrastrukturer også)
    - Veien mot målet inneholder også monolittisk infrastruktur og docker på en monolittisk infrastruktur.
    - Krav til infrastrukturene:
        - (Monolittisk):
            - 1 vm som kjører hele tjenesten, webserver og database.
            - Allerede oppnådd.
        - Ikke monolittisk infrastruktur med flere servere (klassisk infrastruktur):
            - Utvidelse av den monolittiske
            - 1 vm som kjører hver tjeneste, 1 eller flere vm-er for webserver og 1 for db + load balancer
            - Alex har løsningen på denne, automatisering funker kanskje ikke helt
            - Prøve å sette opp et delt volum for å slippe glusterfs, dette kan bidra til å få automatisert helt
            - Lage god dokumentasjon
        - Docker:
            - 1 vm som kjører Docker, og så konteinere for tjenestene
            - Kjøre databasen utenfor docker
            - Konteinerisere webserverne
            - LAMP stack med moodle
        - Docker Swarm:
            - 3, 5, 7 … vm-er med docker som kjører konteinere i swarm
            - Glusterfs
            - Database-cluster

            - Gjør som Kyrre, men bare med moodle
        - Kubernetes:
            - Har master og noder
            - Veldig likt docker swarm, pods i stedet for konteinere
            - Kan kjøre opp docker image på pods
    - Krav til dokumentasjon:
        - Ingenting er ferdig før dokumentasjonen også er ferdig
        - Alt bygger på hverandre, så det er viktig å dokumentere godt så andre kan forstå det
        - Passe på å ha forståelige filer
        - Hvis man benytter moduler i terraform så starter man i main og man kaller der alle andre funksjoner
        - Passe på at filnavn gir mening, ikke ha flere main.tf
- Ting som må gjøres:
    - Alexander og Kristoffer fikser klassisk
    - Kristoffer og Tom Arne ser på database og docker image
    - Benjamin skriver rapport og lager figurer
    - Vi må prøve å få til TICK stack snart, men da trenger vi infrastruktur å overvåke
- Fortsatt vanskelig å finne ting i git. Hvilket repo jobber folk i og hvordan skal vi jobbe i git. Noe å tenke på til neste møte.
- Vi må passe på å ha tid til TICK stack
- Alle må passe på å bruke toggl slik at vi får en så nøyaktig representasjon av dataen som mulig.
- Neste møte: tirsdag 07.03 kl 15

Møte 020523

tirsdag 2. mai 2023 10:27

Deltakere:

Alexander	Tilstede	
Benjamin	Tilstede	Referent
Kristoffer	Tilstede	Møteleder
Tom Arne	Tilstede	

Agenda:

- Status/hvordan går det?
- Gå gjennom hvert kap.
    - Vurdere hva som kan forbedres/skal skrives
- Hvem skriver hva? (annsvarsområder)
- neste møte fredag kl. 11?
- felles betegninger
    - moodle platform, web service, e-learning platform/service etc..
    - Webservice

Referat:

- Status: Implementation begynner å ta form, docker er brukbar, swarm må revideres. Begrensninger på trial av k6, som ødelegger litt for testinga, så må finne en annen løsning der.
- Fellesbetegnelser: Få orden på bruk av forskjellige ord, eks. software, service, platform
- Møte: fredag 11:15

Rapportgjennomgang:

1. Kap 1. Background - må skrives bedre ut fra tilbakemelding fra Erik
2. Kap 2. Er for kort, det må skrives mer.
    a. Mer på professional field, kan kanskje skrive mer om webtjenester, lms?, standarder og best practices. IaaS og skytjenester burde være teori her.
    b. Topic theory kan ha mer underoverskrifter, litt lite oversiktlig med så mye tekst. Kan pushe inn mer om konteinerisering, automatisering, sikkerhet, database, testing. Kan se på forelesninga til Kyrre om arkitekturer og så om det er noen nyttige temaer der.
3. Kap 3. Må få inn de skjermbildene som mangler der. Noen kilder som mangler også.
4. Kap 4. Ser good ut foreløpig, får høre hva Erik sier om den.
5. Kap 5. Står masse x-er der, de må bort. Må skrive i nåtid og uten personlig pronomen. Mangler figur for swarm. Burde kanskje være figur for db-cluster (ikke kritisk). Kan kutte eksperimentell implementasjon, og kanskje putta den i kap 4 i stedet… Kanskje?
    a. 5.1.1 docker og docker swarm mangler + LDAP. Mer git enn gitlab..
    b. 5.1.2 Skal ikke være der, tar det i diskusjon i stedet
    c. 5.2.5 Mangler en del. Burde vært 5.3. Database-cluster er irrelevant for implementasjonen. Kan se om vi kan flytte den et annet sted kanskje..(diskusjon?) Burde nok ikke fjernes, men passer ikke inn der. Overskrifter i dette kapittelet må ryddes opp i.

Generelt kan vi prøve å lage litt tabeller og få inn noen flere figurer.

Arbeidsfordeling:

Dette forsøkes å gjøres ferdig innen fredag. På fredag går vi i gang med de neste kapitlene.

Hvem?	Hva?
Kristoffer	Kap. 2
Alexander	Rettskrive på kap 5.

Benjamin	Kap. 1, 3 + figurer og LDAP i 5.1.1
Tom Arne	Kap 5 - docker og docker swarm + docker i 5.1.1

Til møte med Erik:
Diskutere med Erik databasearkitektur (implementation) og eksperimentell implementasjon.

Appendix H

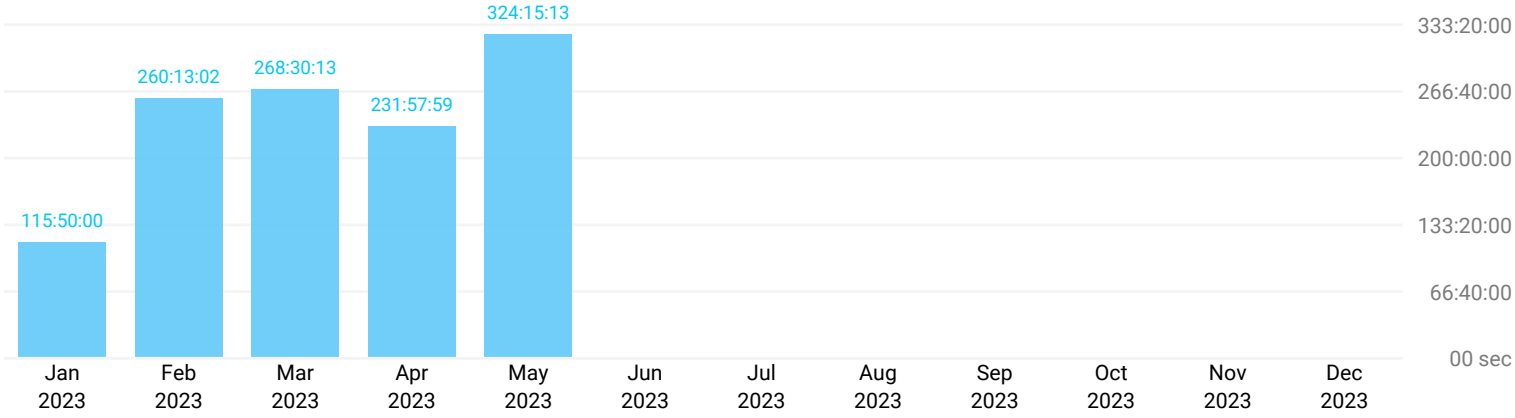
Timesheet

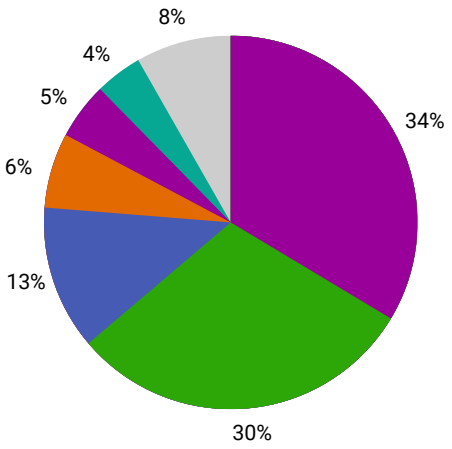
This appendix includes an export from the time tracking software used throughout the project.

Summary Report

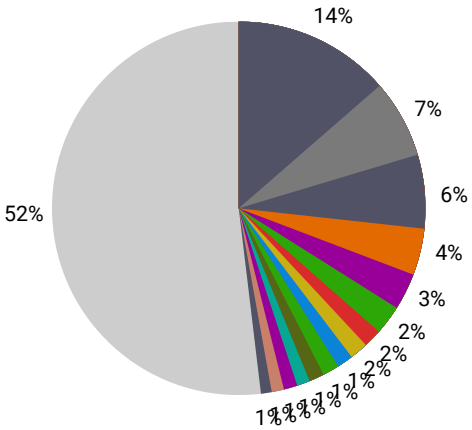
01.01.2023 – 31.12.2023

TOTAL HOURS: 1200:46:27

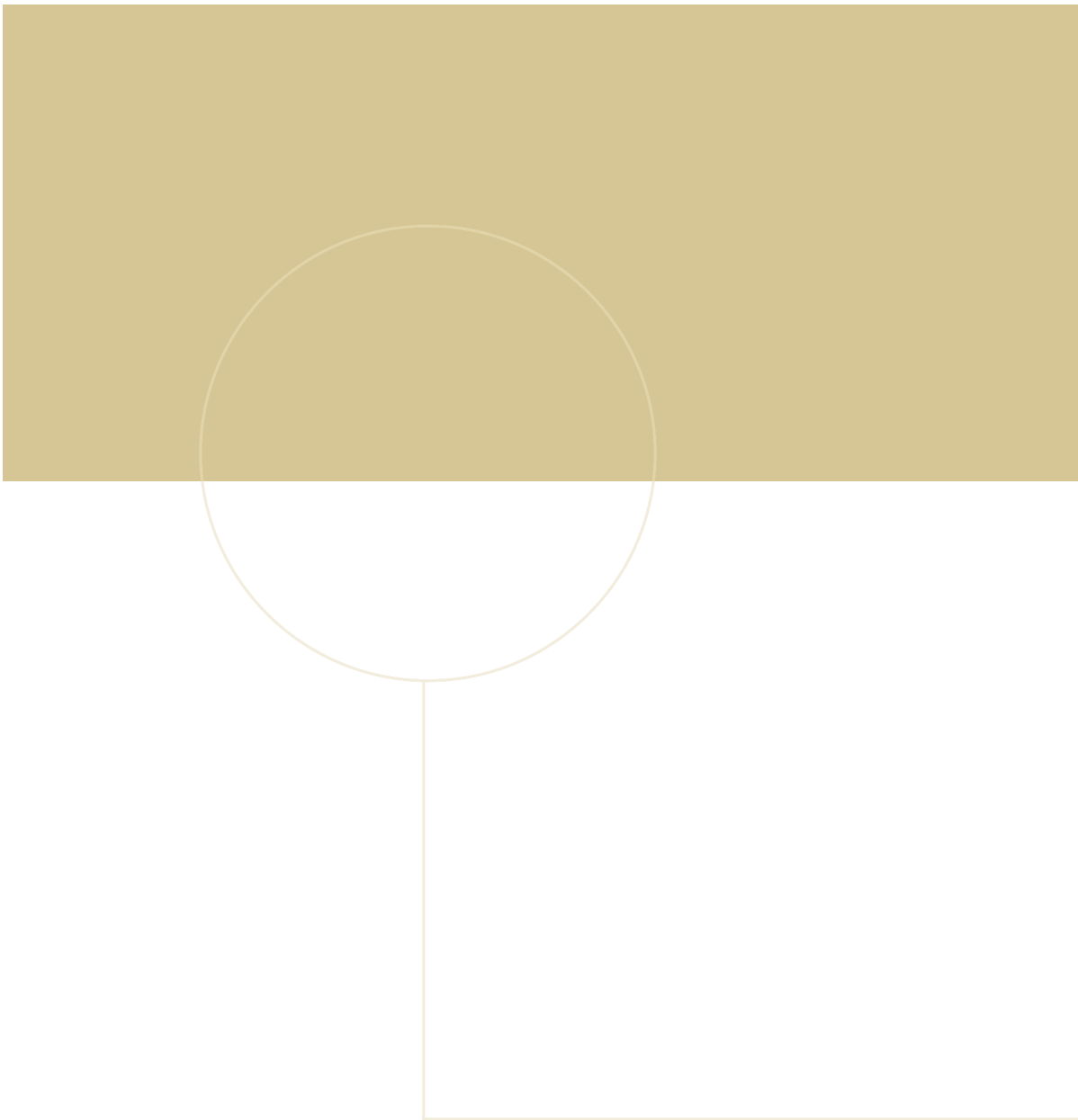




PROJECT	DURATION
08 - Project report	403:39:04
06 - Development	362:35:10
05 - Research	150:22:43
01 - Group meeting	77:45:04
07 - Documentation	58:24:31
09 - Planning	48:22:18
Other projects	99:37:37



TIME ENTRY	DURATION
Writing report	163:46:05
Without description	80:53:32
Project report	76:15:31
Group meeting	49:21:37
docker swarm debug	37:14:16
docker	29:57:59
Docker swarm	19:55:12
ti stack	18:12:41
planing my day/week	17:39:44
k6 testing	16:53:14
Worked with report	15:45:31
Worked with Report	13:52:27
Writing project plan	13:00:00
Writing Report	13:00:00
Meeting with supervisor	12:42:07
Other time entries	622:16:31



NTNU

Norwegian University of
Science and Technology