Theodor Alexander Holme, Torjei Emil Reime, Ola Sagberg

Setup and Management of an E-Learning Platform

Bachelor's thesis in Digital Infrastructure and Cybersecurity
Supervisor: Olav Skundberg
May 2023

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Theodor Alexander Holme, Torjei Emil Reime, Ola Sagberg

Setup and Management of an E-Learning Platform

Bachelor’s thesis in Digital Infrastructure and Cybersecurity
Supervisor: Olav Skundberg
May 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Setup and Management of an E-Learning Platform

Theodor Alexander Holme, Torjei Emil Reime, Ola Sagberg

Abstract

In this thesis, we explore the complexities with scaling, securing, and maintaining an e-learning platform, using Moodle as a case study. We identify potential bottlenecks when scaling and discuss the trade-offs between cost, performance, and complexity in different architectures. The study also examines redundancy and security considerations, highlighting the importance of thorough documentation, diligent monitoring, and the need for a tailored approach based on the size and requirements of the institution. The thesis concludes with recommendations for various scenarios, including user size and security concerns, along with potential future research areas.

The thesis serves as a practical guide for organisations aiming to implement a scalable, secure, and efficiently maintained e-learning platform.

Our findings suggest that a monolithic LAMP (Linux Apache MySQL PHP) stack is suitable for fixed user size and minimal security concerns. For growing user bases or heightened security needs, LAMP or LNMP (Linux Nginx MySQL PHP) microservice stacks are recommended. We also propose that with proper implementation, a highly available or redundant, distributed LAMP or LNMP stack can perform comparably to non-redundant counterparts.

Sammendrag

I denne avhandlingen utforsker vi kompleksitetene ved skalering, sikring og vedlikehold av en e-læringsplattform, med Moodle som case-studie. Vi identifiserer potensielle flaskehalser ved skalering og diskuterer avveiningene mellom kostnad, ytelse og kompleksitet i ulike arkitekturer. Studien undersøker også redundans og sikkerhetsvurderinger, fremhever viktigheten av grundig dokumentasjon, grundig overvåking og behovet for en skreddersydd tilnærming basert på størrelsen og kravene til institusjonen. Avhandlingen konkluderer med anbefalinger for ulike scenarier, inkludert brukerstørrelse og sikkerhetsvurderinger, sammen med potensielle fremtidige forskningsområder.

Avhandlingen fungerer som en praktisk veiledning for organisasjoner som ønsker å implementere en skalerbar, sikker og effektivt vedlikeholdt e-læringsplattform.

Våre funn tyder på at en monolittisk LAMP- (Linux Apache MySQL PHP) stakk er egnet for fast brukerstørrelse og minimale sikkerhetsbekymringer. For voksende brukerbaser eller økte sikkerhetsbehov anbefales LAMP- eller LNMP- (Linux Nginx MySQL PHP) mikrotjeneste-stakker. Vi foreslår også at med riktig implementering kan en høyt tilgjengelig eller redundant, distribuert LAMP- eller LNMP-stakk prestere sammenlignbart med ikke-redundante motparter.

Contents

Figures

Tables

Code Listings

Acronyms

AWS Amazon Web Services. xiii, 31, 47, 51, 52, 60

CA Certificate Authority. xxi, 9, 56

CD Continuous Delivery. 8

CI Continuous Integration. 8

CIA Confidentiality, Integrity & Availability. 17

CLI Command Line Interface. 60

CPU Central Processing Unit. 11, 20, 21, 32, 59, 62, 67, 68

csv Comma separated value. 28

DB Database. 1

DBMS Database Management System. xxi, 9

ETL Extract, Transform & Load. 16

GCP Google Cloud Platform. xiii, 31, 47, 51, 52

GDPR General Data Protection Regulation. 1

HOT Heat Orchestration Template. 23, 61

HTTP Hypertext Transfer Protocol. 18, 28, 29

HTTPS Hypertext Transfer Protocol Secure. 22, 28, 56, 57, 66

IaaS Infrastructure as a Service. 61

IaC Infrastructure as Code. 8, 61

IIS Internet Information Services. 9

IP Internet Protocol. 12, 28

ISP Internet Service Provider. 11

jmx JMeter test plan. 28

LAMP Linux Apache MariaDB PHP. xi, xiii, 43, 44, 52, 54

LAMP Linux Apache MySQL PHP. iii, v, xi, xiii, 19, 23–27, 31–40, 42–45, 47–49, 51–55, 60, 61, 65, 67

LAPP Linux Apache PostgreSQL PHP. xi, xiii, 19, 45, 46, 52, 63

LDAP Lightweight Directory Access Protocol. xi, 26, 27, 68

LMS Learning Management System. 1, 2, 4, 20, 30, 60

LNMP Linux Nginx MySQL PHP. iii, v, xi, xiii, 19, 41, 42, 52, 67

MITM Man-in-the-middle. 22

NTFS New Technology File System. 10

NTNU Norwegian University of Science and Technology. 1, 56, 60

OS Operating System. 8–10, 18, 21

PHP Hypertext Preprocessor. 17, 18, 23, 24, 56

PSU Power Supply Unit. 12

RAM Random Access Memory. 20, 21, 32, 68

SPOF Single Point of Failure. 11, 13, 57

SQL Structured Query Language. 9, 22

SSH Secure Shell Protocol. 22

SSL Secure Sockets Layer. 22, 56, 57

SSO Single Sign-On. xi, 26, 27, 68

TDD Test-Driven Development. 8

TI Telegraf, InfluxDB. 4, 62

TICK Telegraf, InfluxDB, Chronograf, Kapacitor. xi, 4, 15, 18

TLS Transport Layer Security. xxi, 9, 18, 25

VCS	Version Control System. 8
VM	Virtual Machine. 21, 22, 24, 26, 31, 41, 42, 47, 52, 54, 55, 63, 67, 68
XSS	Cross-site scripting. 22

Glossary

Application Programming Interface An API is a set of rules and protocols that allows different software applications to communicate with each other, enabling developers to access and use functionalities provided by another software or service. 16

Certificate Authority A Certificate Authority (CA) is an entity that stores, signs, and issues digital certificates [1] . 9

Database management system A Database Management System (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database [2, p. 8]. 9

General Data Protection Regulation The General Data Protection Regulation is a EU law on data protection and privacy in the EU and the EEA[3] . xvii

Lightweight Directory Access Protocol The Lightweight Directory Access Protocol is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over a network[4]. xviii

Load balancer A load balancer distributes incoming traffic between nodes with the purpose of maximizing throughput, minimizing response time and preventing single-resource overload. 12

Metadata Metadata is data about data, and provides a description of the data characteristics and the set of relationships that link the data found within the database [2, p. 8]. 9

Transport Layer Security A Transport Layer Security (TLS) is a cryptographic protocol designed to provide communications security over a computer network [5] . xviii, xxi, 9

Chapter 1

Introduction

1.1 Background

There are multiple choices of platforms within the field of e-learning, also known as a Learning Management System (LMS), such as Blackboard Learn, Canvas LMS, TalentLMS, Moodle LMS and 360Learning. Most of the e-learning platforms that are in large use are proprietary and cloud-based, which limits the ease of use as an internal platform and its customisation, especially in regards to choices for the infrastructure stack used to host the platform. From this it is clearly a lot of different choices when it comes to provider of a LMS.

Orange Business Services see the need for a LMS for internal use, and have found the Open Source platform Moodle as an appropriate solution. Moodle as a LMS has been implemented in different scenarios and use-cases such as Stack at NTNU and as the Virtual Learning Environment at University of Greenwich[1].

Moodle's documentation provides recommendations on which technologies to use and how to implement the platform. However, the actual implementation and choice of technology are ultimately left to the system administrator[6]. Certain technologies, such as Oracle DB as a database, are highlighted in the documentation due to limited support and are therefore not recommended for use[7]. The documentation also includes a page on server clusters and a link to a high-availability implementation featuring specific technology choices. The challenge with these pages is that they do not justify the choice of technology, leaving it unclear whether these choices are optimal[8].

There are a handful of research papers on Moodle, but they mostly pertain to the usage of Moodle for learning, and not setup and maintenance of the infrastructure [9–11]. There are some papers on the security of Moodle [12–15], however they mostly research the security of Moodle as an isolated application instead of the underlying infrastructure or application of GDPR. Therefore, we aim for our paper to contribute with thoughts around the underlying infrastructure when implementing a LMS or become a foundation for future research.

[1]See: Page on Moodle at IT and Library services at University of Greenwich `https://www.gre.ac.uk/it-and-library/teach/moodle`

1.2 Motivation

As technology advances, topics such as scalable infrastructure, redundancy, security and monitoring will become increasingly crucial for businesses. Understanding the most efficient ways of implementing these technologies could be vital for staying competitive, as well as for ensuring a continuous operation in the future. By addressing a range of research questions in-depth, we aim to provide valuable insights and findings that could serve as a foundation for further research or implementation of similar technologies.

Our motivation for writing this thesis comes from our enthusiasm for examining emerging technologies in the context of digital infrastructure and e-learning platforms. We find it rewarding to delve into the latest technological advancements and explore their potential applications and impacts on various aspects of business operations. By documenting our findings in this thesis, we intend to make it easier for others to replicate a given infrastructure.

1.3 Thesis topic

There exists an overwhelming amount of technologies and approaches for implementing digital services, each with its own trade-offs. Businesses have distinct criteria and requirements for operations, which necessitates making informed choices concerning scalability, availability, security, and monitoring. Navigating this vast landscape of technologies and evaluating the available options can be a challenge for any business, both big and small. Identifying the most suitable choices that fulfils an organisation's specific needs and requirements is a critical aspect of successful technology adoption, and it is therefore the basis of our thesis topic.

The choice of thesis topic is based around the assignment presented by Orange Business Services. The platform must be able to meet the needs of the business in regards to factors like scalability, availability, security and monitoring. Identifying services and technologies for the LMS implementation that covers these criteria is the basis for our thesis. Based on this, we have defined our thesis topic as the following:

What are sensible choices of services and technologies for implementing a scalable and secure e-learning platform?

1.3.1 Research questions (RQ)

From this thesis topic, we have derived four research questions, each of them addressing the crucial factors of scalability, availability, security, and monitoring. While the thesis topic explicitly mentions scalability and security, we consider availability and monitoring as closely related aspects that are essential to the implementation of a successful e-learning platform. Therefore, including these factors in our research questions is well justified.

The following research questions will be used throughout our thesis, and we provide an explanation for the inclusion of each question:

RQ1: What bottlenecks occur when scaling the e-learning platform?

As e-learning platforms experience an increasing number of users and resources, it is essential to identify and address potential bottlenecks that could impact the system's performance and stability. Understanding these bottlenecks will guide the selection of suitable technologies and services that ensure the platform can handle the growing demand.

RQ2: What considerations must be made to ensure redundancy for the e-learning platform?

Availability is a critical aspect of any digital service, especially in e-learning platforms where users rely on uninterrupted access to resources and activities. This research question explores the necessary considerations to ensure that the platform remains accessible and resilient in the face of component failures or other disruptions.

RQ3: How to secure the e-learning platform from unauthorised access and other vulnerabilities?

Security is a top priority for any online service, and e-learning platforms are no exception. With the rising number of cyber threats, it is crucial to ensure that the platform is protected from unauthorised access and vulnerabilities. This research question focuses on identifying the best practices and technologies to safeguard the e-learning platform from potential security risks.

RQ4: What should be documented and monitored to maintain and service the e-learning platform?

Effective monitoring and documentation play a significant role in ensuring the smooth operation and maintenance of an e-learning platform. This research question aims to identify the critical aspects that need to be documented and monitored to facilitate efficient system management, troubleshoot issues, and optimise performance.

By answering these research questions, we aim to provide a comprehensive understanding of the essential factors to consider when implementing a scalable and secure e-learning platform.

1.3.2 Partner organisation

This project was conducted in cooperation with Orange Business Services AS, formerly known as Basefarm[2] AS before the 12th of September 2022. Orange Business Services is a global and local cloud services provider. In addition, they work with digitising older IT-solutions to modern cloud based solutions through consulting. As a company in growth, Orange Business Services have seen an increased need for a dedicated learning platform for internal training of technicians.

[2]See: `https://cloud.orange-business.com/no/basefarm-er-blitt-orange-business-services/`

1.4 Scope and delimitation

1.4.1 Scope

The project is focused on the implementation of Moodle as an infrastructure. The choice of relevant technologies based on theory and analysis is essential for finding the most efficient infrastructure solution. The project delves into technologies that could affect the small- and medium scale deployment of Moodle, such as web servers, databases, load balancing, monitoring, backup and security.

Furthermore, we explore the various ways to automate the set up process of infrastructures, and how to test their performance.

1.4.2 Delimitation

Due to our resources tied to the project, all virtualisation will happen within NTNU Gjøvik[3]'s OpenStack implementation, SkyHiGh[4]. Therefore, we will discuss differences between virtualisation platforms only in theory.

We have little control over the network layer on this platform, so the network will primarily be covered in theory and in discussions.

The e-learning platform we will be using throughout the project is Moodle LMS as that was a part of the specifications set out by the project commissioner, Orange Business Services. Delving into other potential platforms would not only take too much time, but it would also not make it possible to test out as many different technologies and configurations as needed to answer our research questions.

Additionally, every Moodle implementation requires both an operating system and the use of PHP. In this context, we will focus exclusively on Linux as the operating system. This decision is driven by Linux's open-source nature and its status as a standard within development platforms. Since Moodle is coded in PHP, it is essential to utilise web servers that offer PHP support.

Furthermore, Orange Business Services encouraged us to explore and utilise the TICK-stack for monitoring as this was previously a bachelor project they issued[5].

Lastly, we will limit ourselves to the use of TICK or TI stack, and will therefore not explore other monitoring or data visualisation solutions such as Prometheus with Grafana or Splunk.

1.5 Thesis outline

Chapter 1: Introduction In the introduction we will provide the background, topic, research questions and scope for our report.

[3]See: `https://www.ntnu.no/gjovik`

[4]See: `https://www.ntnu.no/wiki/display/skyhigh`

[5]See: previous bachelor thesis for Orange Business Services, TICK-stack by Adrian Lund-Lange and Vetle Tangen Moen, `https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2617774`

Chapter 2: Theory Covers the theory behind the technologies we use and serves as an introduction to the different technologies.

Chapter 3: Method Lays out our analysis criteria, infrastructure setups and covers the steps we take to conduct our research.

Chapter 4: Results Covers the performance, cost and reliability results we gathered from our tests.

Chapter 5: Discussion Covers a reflection and general discussion of our results in regards to both theory and method, in order to see what the result say and what we interpret from them.

Chapter 6: Conclusion Concludes our research and findings, aims to answer the thesis- and research questions, and share our thoughts on future work that can build upon our paper.

Chapter 2

Theory

This chapter discusses the theoretical groundwork for deciding the relevant technologies used in our infrastructure solutions. The chapter begins with essential definitions and concepts, including descriptions of technologies used in our solutions. We also summarise the central topics presented in our research questions; including Scalability, Security, Availability and Monitoring.

2.1 General definitions and concepts

2.1.1 Virtualisation

Virtualisation creates a simulated, or virtual, computing environment as opposed to a physical environment [16]. This allows for multiple virtual instances to run on a single physical machine, and maximises resource utilisation and reduces hardware costs, as fewer physical machines are required to host a larger number of virtual instances. Virtualisation enables resources to be easily allocated or deallocated as needed, providing flexibility and scalability to the IT infrastructure. Additionally, virtualisation technology facilitates the use of cloud based computing, allowing organisations to access computing resources on demand over the internet.

As the concept of virtual memory started to emerge, it allowed the programmer to free themselves from physical memory constraints of the past. Conversely, the emergence of virtual machines gave users the illusion of a dedicated machine. In reality, users were sharing a single real machine with other users, creating a layer of abstraction between the users and the underlying processors [17, p. 686-687]. Virtual machines allow the continued use of existing applications running under the control of different operating systems on a new platform. As several systems can be supported simultaneously and isolated from each other, this creates a level of fault isolation, where a failure in one virtual machine does not affect other instances [17, p. 686-687].

2.1.2 Infrastructure as Code

Infrastructure as Code (IaC) is an approach of managing and automating infrastructure through definition files and thorough validation [18, p. 5]. This approach to infrastructure automation minimises the need for physical hardware configuration; conversely, consistency and repeatable routines are emphasised. Infrastructure as Code introduces key tools normally used in data management and software development. This includes a Version Control System (VCS), automated testing libraries, Test-Driven Development (TDD), Continuous Integration (CI), and Continuous Delivery (CD) [18, p. 5].

2.1.3 Containerisation

The next step from virtualisation is a more application focused and centred approach to IT, a more containerised way of looking at the world of applications, where applications have their run time environments packaged together with the application data and deployed as a containerised object [19, p. 313]. Containers are in many ways lightweight or reduced virtual machines. They no longer rely on an entire machine being simulated and by extension it is not fully possible to run an entire OS as a container. This also has its draw backs: They will be less feature heavy since most containers rely on calls to the underlying OS to function.

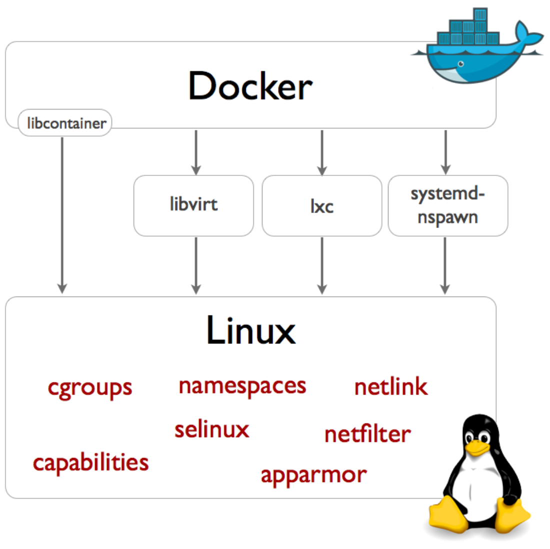


Figure 2.1: The execution drivers and kernel features used by Docker (docker.com/docker-execdriver-diagram.png)

Until we got Docker, one of the most popular framework for containers, they were considered unwieldy and less than optimal solutions. Docker itself relies on already existing technology that exists in Linux's kernel for containerisation and looking at figure 2.1 one can see how drivers and kernel features as utilised by docker [19, p.313-314]. Another popular framework which started as an orchestrator built on top of Docker is Kubernetes.

2.1.4 Database

Databases can be defined as a shared, integrated computer structure that stores a collection of end-user data, as well as Metadata [2, p. 8]. Through the usage of databases, this collection of data can be stored, accessed, managed and updated through the use of a Database management system.

The data can be used in a variety of applications, ranging from small-scale personal projects to large enterprise-level systems. There are a wide range of database management systems that fit the needs of both small- and large businesses. For example, open source DBMS such as MySQL and PostgreSQL are free and easier to use than large-scale vendor DBMS products as they stick to the basic fundamental database principles [2, p. 12]. Likewise, a more robust, durable and expensive solution would be to use Microsoft SQL server or Oracle. In conclusion, it is important to chose a DBMS that supports the scale and functionality of a specific project.

2.1.5 Web server

Web servers are pieces of software that are usually run on servers to provide a framework and method to serve clients with content, such as HTML-based web pages, over the internet by way of requests via HTTP [17, p. 227-230]. Most modern web servers lets you encrypt connections through the use of Transport Layer Security (TLS) where you use a tool such as Let's Encrypt, which is a nonprofit Certificate Authority (CA), to provide the necessary certificate for the TLS encryption [20].

Some examples of modern web servers are Microsoft Internet Information Services (IIS), Apache2 and Nginx. Of these Microsoft IIS is created by Microsoft for running on Windows OS, server or client, while Apache2 and Nginx are open-source servers that run on most OSes be it Windows, Linux-based or Mac. In terms of popularity Nginx is currently the most popular web server, trailed by Apache and followed after that by Cloudflare server, with the market share of Nginx somewhere between $26\% - 35\%$ according to statistics on W3Techs and Netcraft [21, 22].

2.1.6 File storage

File storage is in general the process of storing files on a device be that a server, computer or external media like a DVD. In regards to the topics discussed in this thesis file storage more relates to the storage of files and media in a central repository on a server or multiple servers where it can be accessed either directly or by an application. This storage uses a file system to define how the data is stored and if it has any form of encryption inherent to itself [23].

Further file storage can be expanded to go across multiple devices or servers creating a file cluster. Some file systems have clustering as an inherent property whilst most require some form of third-party application or method to be connected as a cluster. There are also some differences when it comes to a file system functioning on or with different OSes as they might have some form of proprietary way of being implemented. An example of a proprietary file system would be New Technology File System (NTFS) that was first introduced in Windows NT as a replacement for the FAT file system used on earlier Windows systems built on DOS, such as Windows 95 [24, p. 163-197].

The alternative to using a proprietary system would be an open source system, which would be the standard case when working with a open source OS such as a Linux-based OS. An example of an open source file system that would be regularly found on Linux-based OSes would be EXT4 which is the newest iteration of a series of journaling file systems for Linux [25].

2.1.7 Bastion/Entrypoint

A bastion host is defined by R. Shirey as: "*A strongly protected computer that is in a network protected by a firewall (or is part of a firewall) and is the only host (or one of only a few) in the network that can be directly accessed from networks on the other side of the firewall.*" [26, p. 33]. The primary objective of a bastion setup is to safeguard sensitive resources from unauthorised access, which is accomplished by enforcing controlled access to the network or system. As such, bastion setups are utilised in a variety of applications and other use-cases, including the protection of internal networks from external threats, and controlling access to cloud-based resources. A bastion can be configured to provide a variety of security features, such as multi-factor authentication, encryption, and intrusion detection.

2.2 Scalability

Scalability refers to the ability of an IT infrastructure to accommodate growth or increased demand without experiencing performance degradation or downtime. In other words: scalability is an indicator of how well a system can handle increased load after modification or addition of components [27, p. 83]. A scalable infrastructure can handle increased traffic, users, or applications, and can be easily expanded to meet changing business requirements. Scalability can be achieved through the use of technologies such as load balancing, clustering, and virtualisation. In general, there are two ways of scaling a system: vertically, and horizontally [27, p. 83].

2.2.1 Vertical scaling

Vertical scaling, or scaling up, is when you add resources to a single component in a system, for example additional CPUs or memory to a server [27, p. 83]. Vertical scaling is usually done when a quick fix is needed to handle a performance issue, or when spikes in workload is not satisfied by the current performance levels of a system [28]. If the additional resources are not needed long term, it is sensible to scale down for the sake of cost saving.

2.2.2 Horizontal scaling

In contrast to vertical scaling, where resources are added to a single component, horizontal scaling focuses on adding additional components to the infrastructure. As a result, the complexity and upper limit of the infrastructure capacity increases [27, p. 83]. With horizontal scaling, data is partitioned such that each partition can be scaled to fit the needs of the infrastructure, therefore avoiding the physical hardware limits of vertical scaling. In theory, this results in a system that can be scaled almost without any limit [28].

2.2.3 Autoscaling

Autoscaling is the process of automatically and dynamically matching resources to meet the performance requirements of a system, so that performance levels can be continuously maintained [28]. Autoscaling removes the need for a system operator to make frequent decisions about adding resources to increase performance when needed, or removing unused resources to reduce costs when additional performance is no longer needed. Autoscaling is most frequently used when scaling horizontally, because adding or removing resources can be done without application downtime. Contrary to this, autoscaling vertically would make the system temporarily unavailable while it is being redeployed [28].

2.3 Availability

In today's day and age, highly integrated systems are expected to have low downtime. Outages are quickly noticed and may have business-impacting consequences. However, 100% availability is not possible [27, p. 50]. Therefore, we want to get as close as possible to perfect availability. This can be accomplished by employing multiple different techniques such as running backups of data and creating redundancy in power, networking and web servers.

2.3.1 Redundancy

Redundancy is the duplication of critical components in a single system, to avoid a SPOF [27, p. 61-64]. For instance, a company that wants high availability through redundancy may utilise two independent power lines, independent ISPs, on-premise power generator, double or triple PSU servers and similar measures to duplicate infrastructure. For some large companies such as Google, Microsoft, Apple or Amazon it may be necessary to have large data centres on different continents in order to account for local disasters, such as power outages, typhoons or earthquakes, impacting their operations.

2.3.2 Load balancing

A Load balancer can be utilised to spread incoming traffic across multiple web servers to equalise the load. In addition, it can analyse information about the web traffic source to identify if the traffic should be redirected to a specific place. For instance, if the load balancer notices incoming traffic from a known IP address it can route the traffic to an intranet that is reserved for employees. Furthermore, new capabilities and updates can be tested through the use of canary releases which sends a small portion of users over to a testing environment with new features to gather data on a smaller size of customers to gather data and identify bugs and issues.

2.3.3 Backup

Backups refer to the process of creating and storing duplicate copies of data or information that can be used to restore or recover the original data in case of data loss or damage. The purpose of backups is to protect against accidental or deliberate data loss, such as due to hardware failures, software errors, hacking, or natural disasters [17, p. 448].

Backups can be performed in different ways, such as full backups (where all data is copied), incremental backups (where only changes made since the last backup are copied), or differential backups (where only changes made since the last full backup are copied) [17, p. 450-452]. Backups can be stored in various media, such as external hard drives, cloud storage, or tape backups.

Backups are important for several reasons, including:

- Disaster recovery: Backups enable organisations and individuals to recover from data loss due to natural disasters, cyber-attacks, hardware failures, or human errors.
- Business continuity: Backups help ensure that critical data is available and accessible, which is essential for the smooth functioning of business operations.
- Compliance: Many industries are required by law to keep backups of their data, such as healthcare organisations, financial institutions, and government agencies.
- Reassurance: Backups provide reassurance to individuals and organisations by offering a sense of security, knowing that their valuable data is safeguarded and can be recovered in the event of data loss.

2.3.4 High availability

High availability is a term to describe a system or infrastructure that is designed to minimise downtime and ensure that services and applications are accessible to users at all times. This is achieved through implementing redundancy, fault tolerance, and fail-over mechanisms that help eliminate Single Point of Failure and ensure service continuity. By having high availability, critical business applications and services are always available to users, which is essential for business operations and customer satisfaction. In addition, high availability infrastructure is often designed to be scalable, which means that additional resources can be added without disrupting the existing infrastructure or services [17, p. 588-591].

High availability is important for minimising downtime and associated costs, including lost revenue, productivity, and customer loyalty. It is also often used as part of a disaster recovery strategy to ensure that services can be quickly and seamlessly recovered in the event of a disaster or outage. In addition, high availability provides peace of mind to both individuals and organisations, knowing that their critical data and services are always available and can be recovered quickly in the event of an issue. Overall, high availability is crucial for maintaining business continuity, minimising downtime and costs, and providing a positive user experience.

Canary releases

Canary releases or canary deployment is a popular software development practice that allows teams to test new features or changes on a small subset of users before rolling them out to the entire user base [29]. This approach provides valuable feedback on the performance, functionality, and user satisfaction of new features or changes, which helps ensure that they meet user needs and are bug-free. A simple visualisation of how canary releases function can be seen in figure 2.2.

The success of canary releases largely depends on the selection of users for the initial test group. Teams should consider selecting users who are representative of the broader user base and who are willing to provide feedback on the new feature or change. The feedback gathered during the canary release can be used to fine-tune the new feature or change before it is rolled out to a wider audience.

Furthermore, to ensure that canary releases are efficient, teams can introduce additional testing environments and testers. For instance, a development team can create multiple testing environments to ensure that the new feature or change is tested in a variety of conditions. Additionally, testers can be brought on board to provide valuable feedback on the new feature or change, which can help improve the quality of the product. This can be seen in figure 2.3.

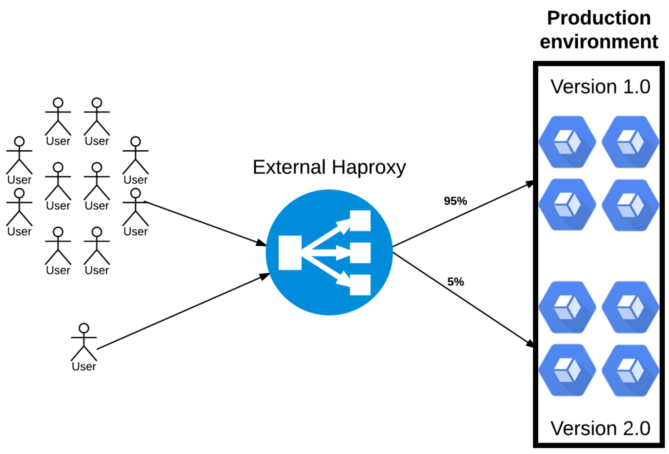


Figure 2.2: Traffic distribution with canary releases [29]

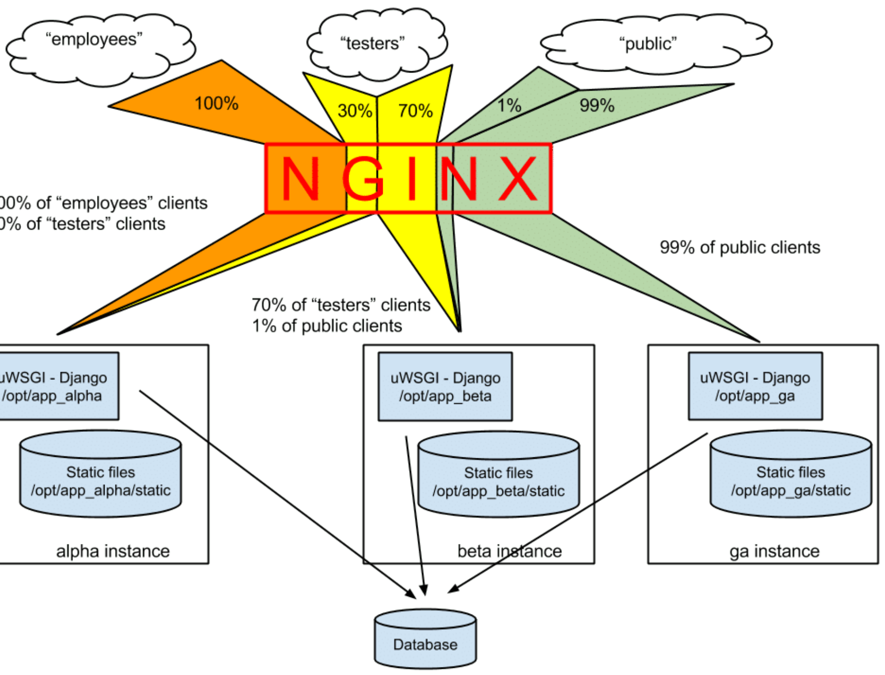

"employees"
"testers"
"public"
100%
30%
70%
1%
99%
NGINX
100% of "employees" clients
30% of "testers" clients
70% of "testers" clients
1% of public clients
99% of public clients
uWSGI - Django /opt/app_alpha
Static files /opt/app_alpha/static
alpha instance
uWSGI - Django /opt/app_beta
Static files /opt/app_beta/static
beta instance
uWSGI - Django /opt/app_ga
Static files /opt/app_ga/static
ga instance
Database

Figure 2.3: Canary deployment with Nginx with three user groups [30]

2.4 Monitoring

Monitoring refers to the continuous and automated collection, analysis, and reporting of performance metrics and other relevant data from various system components, applications, and services. The main purpose of monitoring is to detect and diagnose any potential issues or anomalies in real-time or near real-time, allowing IT teams to respond promptly and proactively before they turn into critical problems.

2.4.1 TICK stack

TICK stack, which stands for Telegraf, InfluxDB, Chronograf, and Kapacitor, is a popular open-source monitoring platform that offers a comprehensive set of tools and functionalities for data collection, storage, monitoring, visualisation, and alerting[31]. The figure 2.4 presents an overview of the TICK components and how they interact.

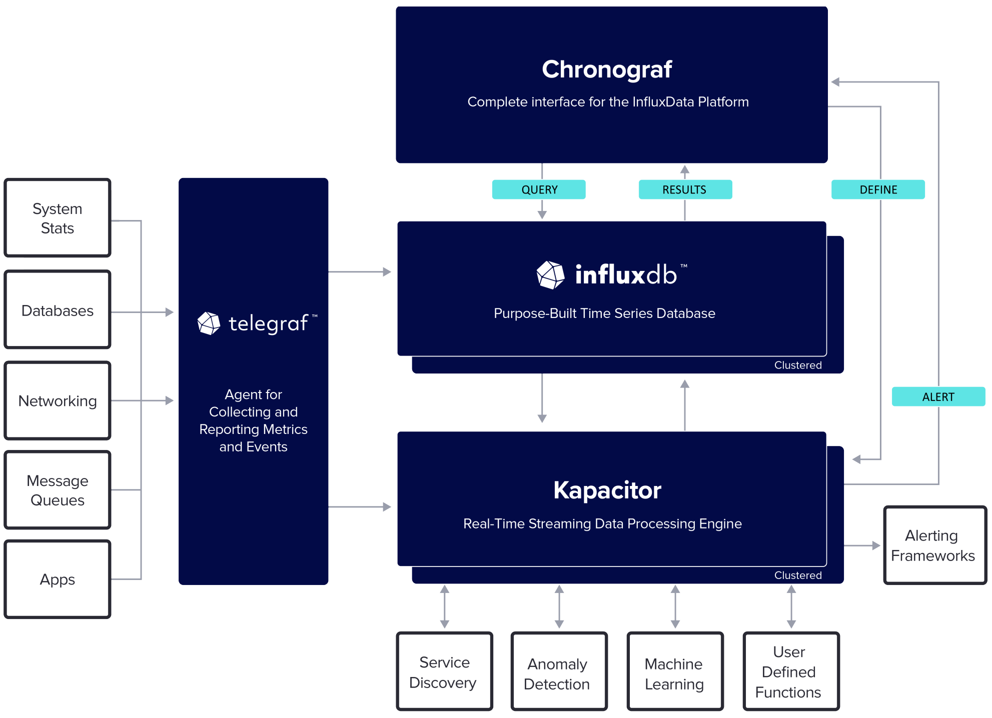


Figure 2.4: Flow diagram of the TICK-stack components [32]

Telegraf

Telegraf is a server-based agent for data collection, designed to gather data from a variety of sources, including system- and application metrics, logs, and sensors

[33]. Telegrafs modular architecture supports a wide range of plugins to allow data collection from various sources, including the following:

- **Input** - Metrics collection from systems, services and 3rd party s
- **Process** - Sanitizes data by transforming, decorating, and filtering metrics
- **Aggregate** - Creates aggregates, for example mean, minimum and maximum from collected metrics
- **Output** - Write to a variety of services, datastores, and message queries

Telegraf is used in conjunction with InfluxDB to collect and store time-series data for the purposes of monitoring, analytics, and alerting [33].

InfluxDB

InfluxDB is a database for collecting, storing, processing and visualising time series data, more specifically successive measurements from the same source to track changes over time. Purpose-built from the ground up with a focus on performance; InfluxDB is capable of handling a high amount of query load [34].

Chronograf

Chronograf is the open source web application of the stack, and is used to visualize monitoring data and create automation and alerting rules. Key features of Chronograph include: Infrastructure monitoring, data virtualisation, alert management, database management and query management [35].

Kapacitor

Kapacitor is an open source data processing framework that makes it easy to create alerts, run Extract, Transform & Load (ETL) jobs and detect anomalies [36]. With Kapacitor, you can use custom logic or user-defined functions to process alerts with dynamic thresholds, match metrics for patterns, compute statistical anomalies, and perform specific actions based on these alerts, like dynamic load rebalancing [37].

2.5 Security

Security is a critical aspect of IT infrastructure, encompassing a range of measures to protect systems, data, and networks from unauthorised access, theft, or damage. Security measures can include access controls, authentication, encryption, firewalls, intrusion detection and prevention systems, and security monitoring. Security must be designed into the IT infrastructure from the ground up, and must be continually reviewed and updated to ensure that both new and existing threats are addressed.

In the field of cyber security; a set of fundamental attributes, namely the desirable characteristics of a secure system, has been defined [38, p. 37]. Originally, the CIA triad consisting of confidentiality, integrity and availability were discussed as core attributes of cyber security. Although the CIA triad is still relevant to this day, the advancements in technology and evolution of security threats has led to additional attributes being added retrospectively, namely authenticity and non-repudiation.

- **Confidentiality** is about preventing unauthorised access to sensitive information, such that only parties with sufficient authorisation can access certain data[38, p. 37]. This includes protecting data at rest, in transit and in use. Confidentiality can for example be achieved by making use of symmetric and asymmetric encryption, access control, and data masking.
- **Integrity** means that the exchanged data must remain unchanged between sender and receiver, neither by accident or by malicious intent[38, p. 38]. Integrity can for example be achieved by making use of digital signatures or hashing.
- **Availability** refers to the balance of restrictions put on the system measured against the utility of the system. In essence, availability is achieved when resources and information are accessible in a timely and reliable manner [38, p. 38]. Additionally, configuring the systems to withstand failures is also a key component of availability. Availability can be achieved by making use of load balancing, redundancy, and disaster recovery planning.
- **Authenticity** assures that the identities of the involved parties in an exchange are who they claim to be, ensuring that the cyber space actors matches their claimed physical reality. [38, p. 38]. This includes protecting against impersonation, spoofing and other types of identity fraud. A common technique used to establish authenticity is two-factor authentication.
- **Non-repudiation** relates to attributing actions to the actors who perform them, including auditable logging of actions, so that the actors would find it difficult to refute their actions or activities [38, p. 38]. Non-repudiation helps protect against for example replay attacks and message tampering.

2.5.1 Securing Moodle

When it comes to the security of the Moodle application, it is imperative to consider all the components included in the system. These components, namely the operating system, web server, PHP, database server, and Moodle application, can be exploited by malicious actors seeking unauthorised access to sensitive data [12, p. 8]. Consequently, it is crucial to implement various robust security measures for each component in order to prevent security breaches and minimising security risks in general.

By configuring the following basic elements, you can add crucial security to one or more servers that run the various components:

- **Firewall**, by blocking access to all ports except the ones we want to expose to the public [12, p. 24]

- **Passwords**, by making use of complex passwords to avoid dictionary attacks [12, p. 28-29].
- **Patching**, to keep the OS and packages up to date to minimise the risk of security threats [12, p. 27-28].
- **Apache configuration**, for example by configuring ServerTokens so that HTTP request/response headers does not expose additional information about what software it uses [12, p. 28-29].
- **MySQL configuration**, for example changing default password of super-user, removing the sample database or restricting remote access to the database [12, p. 32].
- **PHP configuration**, by enabling error logging, disabling exposure of PHP information in server headers, and disabling errors displayed in the web browser [12, p. 34].

2.5.2 Securing TICK stack

When securing TICK stack against potential threats, encryption should be enabled to protect sensitive data in transit between clients and the TICK stack components. With a TLS certificate, clients will be able to verify the authenticity of the InfluxDB server [39]. Furthermore, enabling TLS, authentication and authorization on the Kapacitor- and Chronograf server will add a further layer of security for the TICK stack [40] [41]. Chronograf authentication and role-based access controls can also easily be configured with for example OAuth 2.0 [41].

Chapter 3

Method

In this chapter, we delve into our research approach for this thesis. The process begins with gathering relevant literature, which we use to establish analysis criteria for collected data. Following this, we set up various infrastructure models, ensuring we adjust only one component at a time. We will run performance tests on all models, monitor and collect relevant data for our research, and analyse and discuss these finding. Finally, we answer our research questions and conclude the thesis topic.

Step-by-step method process

1. Literature and criteria for analysis
2. Setup of infrastructure models
3. Run performance tests and gather data
4. Experiment findings and analysis
5. Discuss findings and validity
6. Conclude

3.1 Research methodology

We will configure multiple setups of Moodle with varying combinations of web servers, databases, load balancers and server architecture. The different stack configurations to be explored include monolithic LAMP, microservice LAMP, microservice LNMP, microservice LAMP with MariaDB, microservice LAPP with PostgreSQL, and a redundant microservice LAMP setup. Once these configurations are in place, we will run stress tests and security tests to gather data on their performance, security, cost, and ease of setup and usage. While some metrics may not be quantifiable, such as ease of use or setup, they can still be discussed and evaluated in our overall analysis.

To conduct stress tests, we will simulate various levels of user activity, ranging from light to heavy workloads, and measure the response times, throughput, and resource utilisation for each configuration. For security evaluation, we will note down the issues we encounter, list possible weaknesses, and highlight potential measures that can be implemented to counteract the vulnerabilities.

3.2 Analysis criteria

In this section, we outline the criteria that we will use to evaluate the various components that are part of the Moodle configurations we produce. We will have a look at an outline for each of the research questions where some criteria will come again in other questions whilst some are unique to a question.

3.2.1 Scalability

The first research question we have, as explained in section 1.3.1, is "What bottlenecks occur when scaling the e-learning platform?". We will primarily look into two criteria:

Firstly, we will investigate the cost of scaling the infrastructure up or down to meet various demands. Secondly, we will focus on the observed performance at different scales. We will examine how the various components and configurations of Moodle LMS can handle increasing workloads, the components' capacity for horizontal and vertical scaling, as well as the efficiency of resource utilisation.

Cost

The cost of scalability often comes from expansion or upgrades of hardware. This can come from increasing the number of CPUs, increasing RAM or increasing storage space. Whether this be physical or virtual resources it still impacts the utilisation of other resources. Therefore, if a service runs with a lot of overhead you have unspent resources that might be better used elsewhere.

In addition, the costs associated with the choice of cloud hosting environment, as well as external consulting services for initial setup, are factors that we explore. We estimate and compare the associated costs. Would a higher initial or long-term cost have a noticeable improvement in regards to performance and security?

Performance metrics

In regards to scalability, the performance metrics that we are primarily interested in gathering are usage of CPU, RAM and storage. We want to track these components and associated values over time to get a representative average and compare resource usage under heavier load. Furthermore, the number of requests can be used to identify heavy traffic in order to automatically scale the infrastructure accordingly.

We recognise that network throughput might become a limiting factor when scaling the infrastructure. In theory it's possible to throttle the network throughput and compare the results to non-throttled performance. However, we are not in control of the underlying network infrastructure of SkyHiGh, so any network impact between different hosts wouldn't be affected. As we are not in control of the underlying network we would have to artificially limit the network interfaces in the OS of each VM we use.

3.2.2 Availability

The second research question from section 1.3.1 is "What considerations must be made to ensure redundancy for the e-learning platform?". To answer this question the criteria that needs to be looked at are the general cost to ensure this redundancy, what impact it has on performance and its availability.

We will assess the ability of the selected configurations to provide high availability and minimise downtime, considering factors such as redundancy, fail-over support, and load balancing.

Cost

The cost of ensuring availability is a critical factor to consider. Redundancy, which involves creating backup systems or duplicating services, can significantly increase the overall cost. For example, the increase in cost can depend on the number of redundant components in the setup, as well as the chosen service provider for the redundant setup. The trade-off between the desired level of redundancy and the associated expenses must be considered.

Performance metrics

In regards to availability the performance metrics, we are interested in measuring and logging up- and downtime, number of database requests and number of web server requests. Furthermore, we are interested in finding threshold values for our system so that alerts can be triggered and scaling can be automated in order to maintain high availability. This might for instance be prolonged load at 90% capacity for CPU, RAM or storage.

Redundancy

Redundancy leads to reliability in the case of a failure, however not all components of an infrastructure has the same need for redundancy. Therefore, it is important to investigate if the redundancy of a setup will lead to a sufficient return on investment. Alternatively, would a lower level of redundancy lead to more performance without sacrificing the availability of the system?

3.2.3 Security

The third research question from section 1.3.1 is "How to secure the e-learning platform from unauthorised access and other vulnerabilities?". We will not gather data in order to answer this question, but we will discuss our experience with setup, differences in safety measures between monolithic, distributed and redundant stacks, and create a safety evaluation based on this.

Accessibility

Due to our bastion VM setup, there will only be a singe point of entry for accessing the web servers, databases, and load balancers. The connections in-between the Virtual Machines utilise SSH, where the storage of public and private key pairs are integral for the systems security. Furthermore, password security is important since services such as databases, InfluxDB and Moodle give complete administrator privileges. Moodle will only be reachable through the reverse proxy on Port 443 (SSL) and the reverse proxy has no other connections except for hardened SSH connection from Bastion host.

Although we mention accessibility as an analysis criteria for the security of the system, it is more a matter of explaining and evaluating the level of accessibility we have chosen for our setup, as mentioned above. A stricter level of accessibility would positively impact the security of the system at the cost of user experience. Conversely, a lower level of accessibility would have the opposite effect.

Vulnerability

To assess the security of an implementation of Moodle and the surrounding infrastructure, several approaches can be taken. One common method is to conduct vulnerability scanning and penetration testing to identify weaknesses in the system and potential attack vectors. This can include testing for common vulnerabilities such as SQL injection, XSS scripting, and session hijacking, as well as testing for more advanced attacks such as MITM attacks and HTTPS downgrade attacks.

Another approach is to conduct a code review of the Moodle codebase and associated plugins, looking for any potential security flaws that may have been introduced during development. This is beyond the scope of this bachelor, but similar research has been done back in 2017. Their findings concluded that "By analyzing all the results that were discovered during the security study of Moodle, Moodle 2.6 is not secure as an out of the box product." [42]. There have since been new updates to Moodle so some threats or attacks might have been patched, however this emphasises the importance of keeping software up to date.

3.2.4 Monitoring and documentation

The final research question from section 1.3.1 is "What should be documented and monitored to maintain and service the e-learning platform?". Here we will be looking more closely at what documentation already exists, how good it is, what should be added to make it better, and what can be automated in terms of monitoring.

Ease of setup and use

Ease of setup and use will be split into two sections. Firstly, we will evaluate the complexity of the initial setup process, including the installation and configuration of necessary components. We will consider factors such as required technical expertise, the need for manual intervention, and the availability of pre-built configurations or templates. Secondly, we will assess the user experience and the simplicity of managing the system, including configuration, deployment, and ongoing administration. This will involve evaluating the intuitiveness of the availability of documentation and the level of community support.

Automation

We will investigate the extent to which automation can be used to simplify and streamline set up and management tasks in order to improve efficiency. We will explore the availability of automated tools for tasks such as deployment, monitoring, backups, and scaling. Since some technologies might sacrifice simplicity for performance, automation can contribute to reducing the difficulty of setup and other tasks whilst reaping the performance rewards. The technologies we will utilise and look at are OpenStack Heat Orchestration Template (HOT), Ansible, Puppet, Docker, Kubernetes and native shell scripts. Furthermore, we will look at using these in a combination with each other.

All our scripts are gathered on a GitLab project and can be accessed through this url `https://gitlab.stud.idi.ntnu.no/obs-bachelor/obs-bachelor`

3.3 Outline of Stacks for Data Collection

When answering the research questions, our research topic and implementing our method we will be creating a multitude of different implementations using different technologies and services, or stacks, which we can record data from and detail our experience with. We will primarily change one component to keep a consistent basis for comparison. We will implement one monolithic stack which will act as a baseline. By changing one component at a time we can isolate the differences in performance to the single change that was made. If we were to change multiple components between the implementations it would be difficult to attribute performance changes to one choice or technology.

3.3.1 Monolithic LAMP from Moodle Documentation

The most basic monolithic stack that we will be testing is derived from Moodle's own documentation and will serve as our baseline. This stack is based on the standard LAMP structure, which utilises Linux as the operating system, Apache as the web server, MySQL as the database provider, and PHP for running server-side code. As discussed earlier in the delimitation chapter, the components Linux and PHP will remain constant. Linux will not be changed, as it is generally the most used operating system for web service stacks, and PHP will not be changed, as Moodle is built using PHP and would require re-coding for deployment using an alternative language, such as JavaScript or Python.

A monolithic implementation involves running all services together on the same physical or Virtual Machine. This approach has its advantages, such as being easier to set up and only requiring a single VM or computer. However, there are also disadvantages, including the difficulty in upgrading a service or component without restarting the entire system. Additionally, it may be challenging to modify parts of the stack, as everything is integrated into one solution, as opposed to having the abstraction that microservices provide.

Table 3.1: Monolithic LAMP Virtual Machine Specifications

VM	vCPU	vRAM [GB]	Storage [GB]
monolithic	8	16	40

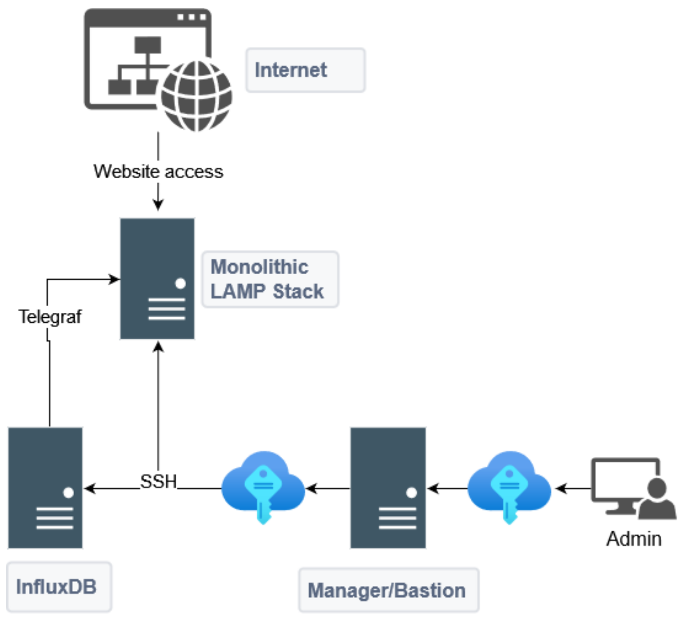


Figure 3.1: Figure of monolithic LAMP configuration

3.3.2 Microservice LAMP

This implementation adopts the same basic stack from the previous subsection on monolithic stacks, but applies it as a microservice architecture rather than a monolithic one. A general point to make when creating this basic LAMP stack is that we use MySQL which is second most popular database, according to DB-Engines, closely following Oracle Database which has been the most popular for multiple years[43, 44].

To achieve this, we will have a reverse proxy as our first server. This server will be the only one directly exposed to the internet on ports 443 and 80. Port 80 will redirect to 443 to ensure secure connections running over TLS. We will be using HAProxy for our reverse proxy as it implements this in the same way as the main alternative, Nginx. Following this, we will have a web server running Apache2 and hosting the Moodle installation. These servers will be accessed by the reverse proxy where it redirects the traffic internally over whatever port we choose to run the Apache service on, but the default would be port 80. Lastly, there will be a database server running MySQL.

Table 3.2: LAMP Microservices Virtual Machines Specifications

VMs	vCPU	vRAM [GB]	Storage [GB]
web server	4	4	40
database	2	8	40
load balancer	2	4	40
bastion	2	4	40

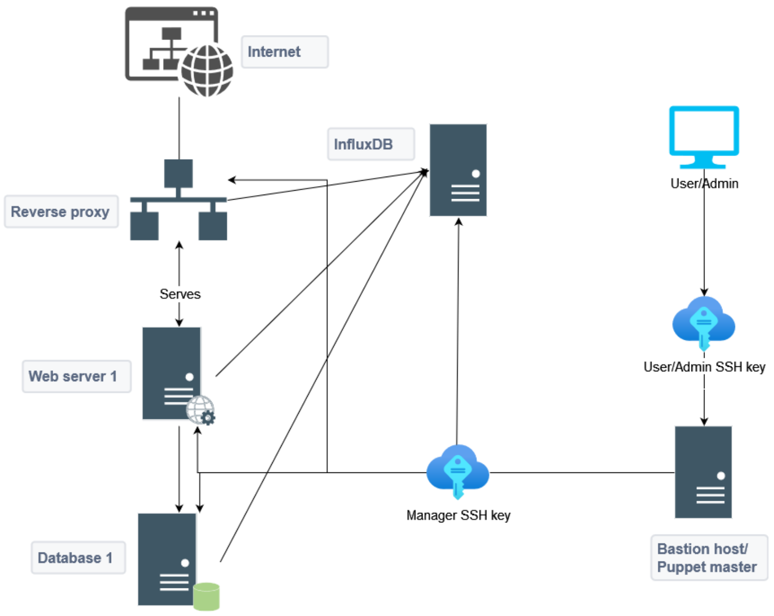


Figure 3.2: Figure of microservice LAMP configuration

3.3.3 Microservice LNMP

In this implementation, we have exchanged the Apache web server for a Nginx web server, which passed Apache as the most popular web server in 2019 and is the most popular web server at the time of this writing[45]. This is to keep as many variables as possible constant, while exploring the performance difference between the web servers. Furthermore, we will look at the built-in security features and also evaluate the ease of setup and use. This stack requires the same amount of resources and will therefore use the same VM setup as the previous stack 3.2 and the same infrastructure configuration 3.2.

3.3.4 Microservice LAMP (MariaDB)

In this implementation, we return to our baseline stack with Apache, but exchanged the MySQL database for a MariaDB database. This is done to explore the differences between the database technologies, and highlight changes in performance, security and simplicity of setup and use, while keeping other variables unchanged. Although MySQL is still the most known database language by developers[46], it is on a slow downwards trend[44].

3.3.5 Microservice LAPP (PostgreSQL)

In this implementation, we again use the LAMP stack as our basis, but we substitute the MySQL database for a PostgreSQL database. Similarly to MariaDB, PostgreSQL is also on the rise[44] and was in 2022 ranked as the second most known database technology in a survey from Stack Overflow only 3.26% behind[46].

3.3.6 Microservice LAMP Redundant

In this implementation, we make the basic LAMP stack redundant by including another set of web servers and databases. Moreover, we are interested in finding the differences in performance when doubling the working components. Conversely, due to the increase of Virtual Machines, the infrastructure becomes more complex, and we will examine how this impacts both security and setup complexity.

We hypothesise that this implementation will work better under heavy load since the load balancer can work in a round-robin fashion to equalise the load between the web servers. However this comes at a financial and resource cost which we will explore further in the discussion chapter.

As we want to implement a SSO, we developed a diagram representing the redundant microservice stack, as seen in figure 3.3. This combined illustration is essential as we could potentially utilise this redundant stack to deploy the SSO, employing LDAP via FreeIPA.

Table 3.3: Redundant LAMP Microservices Virtual Machines Specifications

VMs	vCPU	vRAM [GB]	Storage [GB]
web server 1	4	4	40
web server 2	4	4	40
database 1	2	8	40
database 2	2	8	40
load balancer	2	4	40
bastion	2	4	40

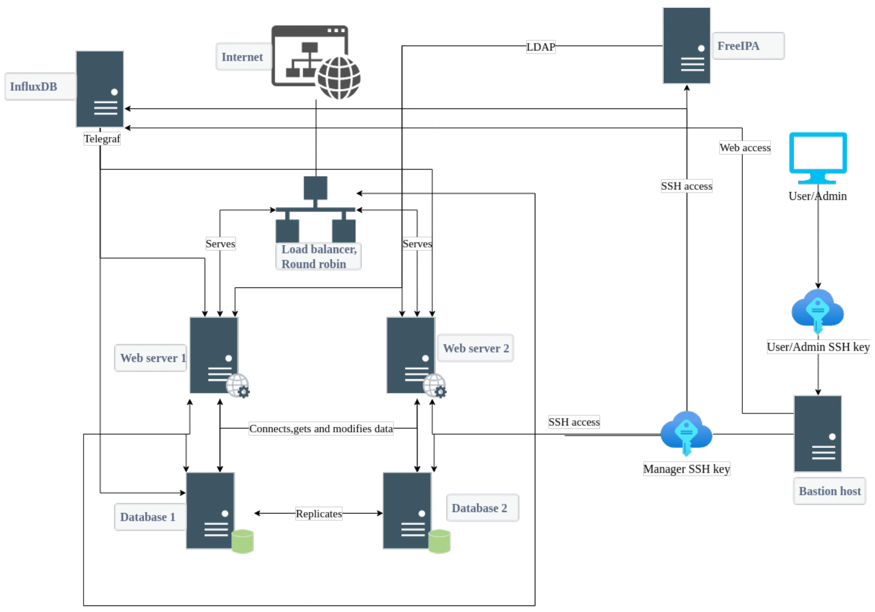


Figure 3.3: Figure of redundant microservice LAMP configuration including LDAP server running FreeIPA to illustrate potential SSO method

3.4 Data collection

We follow the Moodle documentation for tests with JMeter [47]. Within the Moodle admin pages under the developer tab there is a Make test course page [48]. From here there are several options of sizes of the test course ranging from extra small (XS) to extra extra large (XXL). For our purposes the most relevant and interesting sizes are the small (S) and medium (M) sizes. The small course size simulates 30 users and runs through each section 5 times per user, resulting in 150 calls. In addition it fills up the cache and primes the system by running the first test an additional round in the beginning. The Medium course size simulates 100 simultaneous users for 5 rounds and is similar to a typical course at a large institution. This is useful for testing performance under normal conditions [48].

With a test course created, we have to create a test plan. For that we use the JMeter test plan generator within Moodle [49]. It can be found in the Site Administration block under the Development tab by clicking on the Make JMeter test plan page. From here we select our test course and decide the size of the test plan. This choice decides the number of queries run and simulated users. This choice ranges from extra small (XS) to extra extra large (XXL), however for our purposes small (S) or medium (M) suffices. Furthermore, we want to check the "update users password" checkbox in order to generate new passwords for the simulated users. This will generate an additional csv file with the new passwords which will be added into JMeter when running the tests.

After the JMeter test plan generator has completer we are left with two files a "testplan_xxx_xxx.jmx" and a "users_xxx_xxx.csv" file. To start JMeter make sure you have Java installed and run the ApacheJMeter.jar file from within the bin folder. This will open the JMeter GUI. Then you go to file and click open and locate the jmx file generated from Moodle. When that is done you will get content in the left side menu. From here you want to open up the Warm-up site, click on the default site request and enter the protocol as HTTP or HTTPS, and server name or IP. Then you want to select the generated users csv file under "csv users data". You want to repeat this step for the Moodle Test. The final step before you run is to add Listeners by right clicking on the Test Plan, going to Add and selecting the Aggregate and Summary Report. Then you just press the green play icon on the top toolbar and the test will run. You can monitor the results as they are running by clicking the aggregate or summary report in the left side menu [47].

3.5 Data analysis

In order to analyse the data we need an understanding of the information we gather. From the summary test in JMeter we get the following fields:

- Label
- # Samples
- Average
- Min
- Max
- Std. Dev.
- Error %
- Throughput
- Received KB/sec
- Sent KB/sec
- Avg. Bytes

And from the aggregate test we get these fields:

- Label
- # Samples
- Average

- Median
- 90% Line
- 95% Line
- 99% Line
- Min
- Max
- Error %
- Throughput
- Received KB/sec
- Sent KB/sec

As we can see the fields "Label,# Samples,Average,Min,Max,Error %,Throughput,Received KB/sec,Sent KB/sec" overlap between the tests and since both tests run in parallel the overlapping fields give the same results.

Label consists of the names of the simulated actions that are run during the tests. These are the following labels present in both the summary and aggregate tests:

- Frontpage not logged
- View login page
- Login
- Frontpage logged
- View course
- Logout
- View a page activity
- View course again
- View a forum activity
- View a forum discussion
- Fill a form to reply a forum discussion
- Send the forum discussion reply
- View course once more
- View course participants
- TOTAL

Most of these actions are quite straight forward, the first actions have to do with the process of loading the login screen and logging in. When logged in it goes through the actions that become available like viewing courses, forum activities, creating a forum reply, sending said forum reply and viewing the course again after it has been loaded once since some of it might be cached.

The not so obvious TOTAL field is a label for the data columns, it operates differently depending on the data. It sums the # of Samples, It gives an average of the average values, it gives a median of the median values, it sums the throughput, it averages the Error %, it shows the min and max values, it gives the average standard deviation and averages the 90/95/99% lines.

The # Samples field gives us the number of HTTP requests for each action. This number is usually the number of $users * rounds$, but in order to fill the cache and engage the system, JMeter runs a warmup round on the first 6 actions making the number equal to $users * (rounds + 1)$.

The average provides a general idea of the typical value, the median represents the middle value, the min represents the smallest value, and the max represents the largest value in the dataset.

The 90/95/99% Line provides information about the distribution of data by indicating the value below which a certain percentage of the data falls. The standard deviation quantifies the variability or spread of data points from the mean, giving insights into the overall dispersion of the dataset.

Error % indicates the percentage of failed requests, throughput measures the rate of work processing, Received/Sent KB/sec represent the data transfer rates, and Avg. Bytes gives insights into the average size of data units or payloads in the system.

With a better understanding of the data we will gather we want to analyse the data. This will be done by comparing values, representing the data visually and by contextualising the data. We will visualise the data through Microsoft Excel and highlight key values.

3.6 Reliability and validity of the results

To ensure that our results are generated in a reliable and valid manner, we will use reputable technologies like JMeter for testing, which is widely adopted by industry professionals. To account for potential sources of error, we will consider the impact of testing software on system resources and conduct multiple tests to average out results and minimise the effect of outliers. Furthermore, we will adhere to best practices to ensure that our results are easily reproducible and comparable to existing benchmarks and theories.

There are some factors outside our control that will affect the results, such as running this on SkyHiGh instead of on bare metal. Other factors that affect the results that is outside of our control is the use of a network we do not control so data going to or from the servers can not be closely regulated.

By following this methodology, we aim to provide a comprehensive and reliable comparison of various Moodle LMS and monitoring configurations, enabling organisations to make informed decisions when implementing these solutions.

Chapter 4

Results

In this chapter we are going to present the cost estimates and performance results from our tests and visualise our findings. In places where the results are equivalent we will acknowledge this and move on. For instance, since some of the stacks build upon each other and require the same resources the associated costs stay the same.

JMeter provides two types of test results: Summary and Aggregate Reports. Both reports display test statistics such as response time and error rate, with results measured in milliseconds. The TOTAL field in these reports provides additional statistics such as a calculated average, median, and the extremal points such as maximum and minimum.

In addition, these tests can be simulated for a specified amount of users. For the monolithic LAMP stack and microservice LAMP stack, we have chosen to run both the small- and medium tests, which simulates 30 and 100 consecutive users respectively. For the other stacks, we found it sensible to only run the medium tests.

4.1 Monolithic Stack

4.1.1 Moodle docs LAMP

Cost estimate

Based on the previously mentioned monolithic setup specifications:

Table 4.1: Monolithic LAMP Virtual Machine Specifications

VM	vCPU	vRAM [GB]	Storage [GB]
monolithic	8	16	40

It is not possible to match the exact Monolithic LAMP virtual machine specifications with available virtual machines in AWS, GCP and Azure. Therefore, we approximate the most similar Virtual Machines (VMs) available in the cloud to the mentioned specifications, emphasising the amount of vCPU's over vRAM and storage.

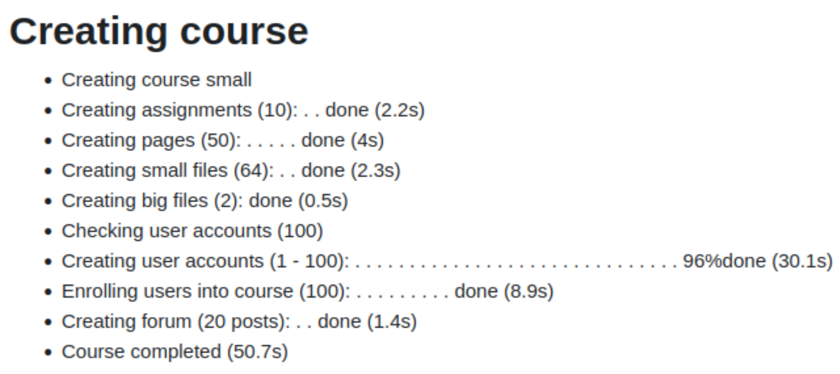
Table 4.2: Monolithic LAMP Cost Estimate

Provider	Instance	Storage [GB]	Estimated monthly cost [USD]
AWS: EC2	c6g.2xlarge	40 EBS	$213 [50]
GCP: Compute Engine	e2-standard-8	40 Persistent Disk	$220 [51]
Azure: Virtual Machine	A8 v2	80 Temporary storage	$680 [52]

Additionally, setting up a monolithic Moodle environment may require professional assistance in the form of consultants. This would depend on what kind of expertise already exists within the organisation. For Orange Business Services, setup and installation of a Monolithic Moodle environment could be handled by their internal operations team. However, a local school or university wishing to set up Moodle for the first time would require external assistance.

The cost of a consultant can vary widely depending on their experience, region, and scope of work, but a reasonable estimate would be between $75 to $150 per hour. The number of hours required would depend on the complexity of the setup. For this approximation, we can assume it would take 20 hours to set up the monolithic infrastructure and Moodle installation. Therefore, the estimated consulting- and setup costs are between $1,500 to $3,000 (one-time cost) [1]

Performance results



Creating course

- Creating course small
- Creating assignments (10): . . done (2.2s)
- Creating pages (50): . . . . . done (4s)
- Creating small files (64): . . done (2.3s)
- Creating big files (2): done (0.5s)
- Checking user accounts (100)
- Creating user accounts (1 - 100): 96%done (30.1s)
- Enrolling users into course (100): . . . . done (8.9s)
- Creating forum (20 posts): . . done (1.4s)
- Course completed (50.7s)

Figure 4.1: Figure of monolithic LAMP stack small size test course creation time

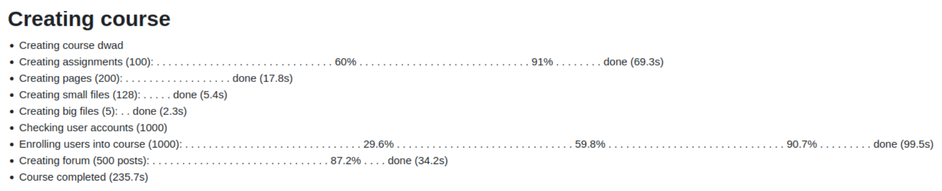
[1]Based on wages for Moodle and infrastructure consultants from upwork: `https://www.upwork.com/search/profiles/?q=moodle`

Table 4.3: Monolithic LAMP JMeter Small Summary (30 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	180	278	249
2	View login page	180	116	17
3	Login	180	1080	289
4	Frontpage logged	180	419	45
5	View course	180	382	81
6	Logout	180	352	46
7	View a page activity	150	334	44
8	View course again	150	334	68
9	View a forum activity	150	497	48
10	View a forum discussion	150	503	181
11	Fill a form to reply a forum discussion	150	510	74
12	Send the forum discussion reply	150	277	45
13	View course once more	150	338	38
14	View course participants	150	314	37
15	TOTAL	2280	412	254

Table 4.4: Monolithic LAMP JMeter Small Aggregate (30 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	923	1302	185	1503	0.19
2	0.0%	140	189	112	235	0.19
3	0.0%	1762	1963	972	2016	0.18
4	0.0%	494	515	420	522	0.19
5	0.0%	521	736	364	873	0.19
6	0.0%	437	459	353	481	0.17
7	0.0%	396	425	332	566	0.17
8	0.0%	399	805	324	839	0.17
9	0.0%	578	625	495	631	0.17
10	0.0%	830	921	467	922	0.17
11	0.0%	628	637	506	696	0.17
12	0.0%	349	378	276	429	0.17
13	0.0%	399	428	340	429	0.17
14	0.0%	376	424	310	449	0.17
15	0.0%	952	1544	354	2016	2.00



```
Creating course

• Creating course dwad
• Creating assignments (100): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60% . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91% . . . . . . . . done (69.3s)
• Creating pages (200): . . . . . . . . . . . . . . . . . . done (17.8s)
• Creating small files (128): . . . . . done (5.4s)
• Creating big files (5): . . done (2.3s)
• Checking user accounts (1000)
• Enrolling users into course (1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29.6% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59.8% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90.7% . . . . . . . . . done (99.5s)
• Creating forum (500 posts): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87.2% . . . . done (34.2s)
• Course completed (235.7s)
```

Figure 4.2: Figure of monolithic LAMP stack medium size test course creation time

Table 4.5: Monolithic LAMP JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	226	46
2	View login page	600	116	12
3	Login	600	1036	234
4	Frontpage logged	600	427	46
5	View course	600	442	94
6	Logout	600	365	37
7	View a page activity	500	316	82
8	View course again	500	403	86
9	View a forum activity	500	504	113
10	View a forum discussion	500	957	452
11	Fill a form to reply a forum discussion	500	729	266
12	Send the forum discussion reply	500	325	38
13	View course once more	500	339	70
14	View course participants	500	342	41
15	TOTAL	7600	464	304

Table 4.6: Monolithic LAMP JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	264	285	223	1217	0.19
2	0.0%	137	151	115	198	0.19
3	0.0%	1648	1843	954	2106	0.18
4	0.0%	507	613	419	728	0.18
5	0.0%	656	786	417	1087	0.18
6	0.0%	421	528	359	603	0.16
7	0.0%	458	630	292	1153	0.17
8	0.0%	526	723	387	1347	0.17
9	0.0%	654	870	472	1495	0.17
10	0.0%	1666	2185	895	2870	0.17
11	0.0%	1214	1374	665	1454	0.17
12	0.0%	392	471	320	601	0.17
13	0.0%	415	566	328	1324	0.17
14	0.0%	397	491	338	828	0.17
15	0.0%	1051	1611	374	2870	2.00

This performance summary contains the results of JMeter stress tests conducted on a Moodle e-learning platform running on a monolithic LAMP stack. The medium test simulates 100 consecutive users and their interactions with the platform, such as viewing the front page, logging in, viewing courses and activities, participating in forums, and logging out. Meanwhile, the small test simulates the same interactions, although only for 30 consecutive users. The datasets provide insight into the performance of the platform under load, specifically focusing on the average and max response times (in milliseconds), and the standard deviation values. The number of samples, average response time and standard deviation values are presented in the summary tables 4.3 and 4.5. Additionally, the datasets contains error percentages, 95 and 99 percentiles, median and max response times, as well as throughput for each interaction. These values are found in the aggregate tables 4.4 and 4.6.

There are several notable performance indicators worth considering in these datasets. For instance, the "course creation" time shows a significant difference between the small test and medium test. As illustrated in figures 4.1 and 4.2, the course creation time in the small test, depicted in Figure 4.1, is considerably faster at 50.7 seconds compared to the 235.7 seconds recorded in the medium test, as shown in Figure 4.2.

When looking at the medium test 4.5, the average response time (in milliseconds) for each interaction ranges from 116ms for viewing the login page to 1036ms for logging in, indicating that the login process takes the longest time to complete. This can also be observed in the small test 4.3, only with a slightly higher average on logging in at 1080ms.

The maximum response times for interactions are noticeable higher for the medium test compared to the small test, as seen in the tables 4.4 and 4.6. These values are also reflected in the standard deviation values, which measure the variability of response times. In this case, the highest standard deviation is observed in the "View a forum discussion" interaction, suggesting that the response times for this interaction are less consistent compared to others.

The error percentage column in the datasets indicates that there were no errors encountered during the tests, which is a positive sign for the stability of Moodle running on a monolithic LAMP infrastructure. Throughput, which is measured in requests per second, is a useful metric for evaluating the platform's capacity to handle user requests. In both the small- and medium tests, the overall throughput was approximately 2.00 requests per second in total.

4.2 Microservice Stacks

4.2.1 LAMP

Cost estimate

Based on the previously mentioned microservices setup specifications:

Table 4.7: LAMP Microservies Virtual Machines Specifications

VMs	vCPU	vRAM [GB]	Storage [GB]
web server	4	4	40
database	2	8	40
load balancer	2	4	40
bastion	2	4	40

Table 4.8: Microservice LAMP Cost Estimate

Provider	Service	Instance	Estimated monthly cost [USD]
AWS	Web server	c6g.xlarge	$106
AWS	Database	t4g.large	$50
AWS	Load balancer	t4g.medium	$25
AWS	Bastion	t4g.medium	$25
AWS	Storage	4 x 40GB EBS	$13
-	-	-	$219 [50]

Table 4.9: Microservice LAMP Cost Estimate

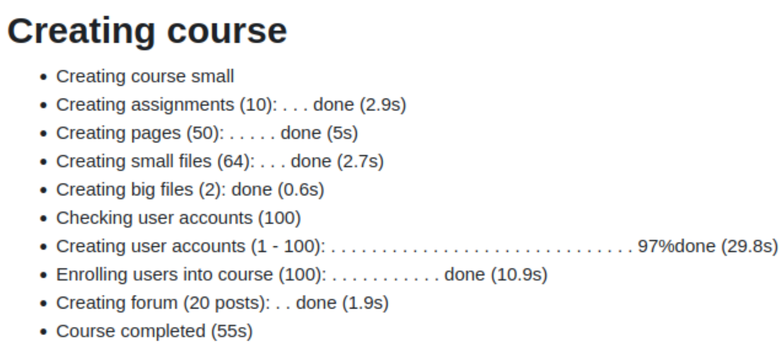
Provider	Service	Instance	Estimated monthly cost [USD]
GCP	Web server	e2-highcpu-4	$80
GCP	Database	e2-standard-2	$54
GCP	Load balancer	e2-medium	$27
GCP	Bastion	e2-medium	$27
GCP	Storage	4 x 40GB Persistent Disk	$18
-	-	-	$206 [51]

Table 4.10: Microservice LAMP Cost Estimate

Provider	Service	Instance	Estimated monthly cost [USD]
Azure	Web server	A4 v2	$282
Azure	Database	D2as	$161
Azure	Load balancer	D2ls v5	$156
Azure	Bastion	D2ls v5	$156
Azure	Storage	4 x 64GB Premium SSD	$64
-	-	-	$819 [52]

A microservice infrastructure would also require help from external consultants. We are assuming the same price as before, between $75 to $150 per hour. We estimate roughly 40 hours to set up the microservice infrastructure, doubling the estimate of the monolithic setup. Consequently, the estimated consulting- and setup costs are between $3,000 to $6,000 (one-time cost).

Performance results



```
Creating course

• Creating course small
• Creating assignments (10): . . . done (2.9s)
• Creating pages (50): . . . . . done (5s)
• Creating small files (64): . . . done (2.7s)
• Creating big files (2): done (0.6s)
• Checking user accounts (100)
• Creating user accounts (1 - 100): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97%done (29.8s)
• Enrolling users into course (100): . . . . . . . . . . . done (10.9s)
• Creating forum (20 posts): . . done (1.9s)
• Course completed (55s)
```

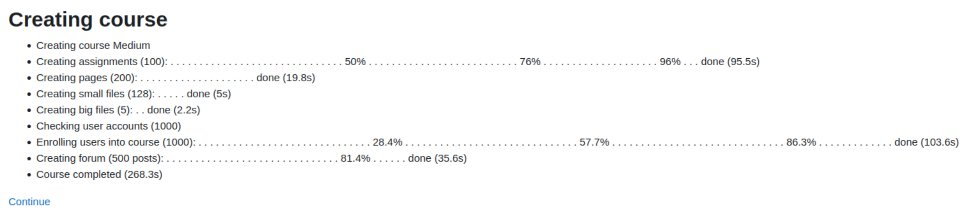
Figure 4.3: Figure of microservice LAMP stack small size test course creation time

Table 4.11: Microservice LAMP JMeter Small Summary (30 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	180	186	16
2	View login page	180	118	9
3	Login	180	1057	225
4	Frontpage logged	180	409	37
5	View course	180	334	61
6	Logout	180	383	47
7	View a page activity	150	325	36
8	View course again	150	262	17
9	View a forum activity	150	477	40
10	View a forum discussion	150	461	176
11	Fill a form to reply a forum discussion	150	485	62
12	Send the forum discussion reply	150	300	35
13	View course once more	150	301	31
14	View course participants	150	295	41
15	TOTAL	2280	387	237

Table 4.12: Microservice LAMP JMeter Small Aggregate (30 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	217	228	183	276	0.19
2	0.0%	135	148	117	155	0.19
3	0.0%	1581	1680	971	1859	0.18
4	0.0%	477	495	400	538	0.19
5	0.0%	479	518	314	569	0.19
6	0.0%	466	487	377	504	0.17
7	0.0%	383	399	324	401	0.17
8	0.0%	296	326	260	351	0.17
9	0.0%	564	586	467	592	0.17
10	0.0%	776	876	435	876	0.17
11	0.0%	590	614	476	668	0.17
12	0.0%	358	371	296	390	0.17
13	0.0%	356	397	291	451	0.17
14	0.0%	354	419	283	570	0.17
15	0.0%	948	1451	331	1859	2.00



```
Creating course

• Creating course Medium
• Creating assignments (100): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50% . . . . . . . . . . . . . . . . . . . . . . . . . 76% . . . . . . . . . . . . . . . . . . . 96% . . . done (95.5s)
• Creating pages (200): . . . . . . . . . . . . . . . . . . . . done (19.8s)
• Creating small files (128): . . . . . done (5s)
• Creating big files (5): . . done (2.2s)
• Checking user accounts (1000)
• Enrolling users into course (1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28.4% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57.7% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86.3% . . . . . . . . . . . . . done (103.6s)
• Creating forum (500 posts): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81.4% . . . . . . done (35.6s)
• Course completed (268.3s)

Continue
```

Figure 4.4: Figure of microservice LAMP stack medium size test course creation time

Table 4.13: Microservice LAMP JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	234	27
2	View login page	600	116	11
3	Login	600	1115	359
4	Frontpage logged	600	500	111
5	View course	600	1811	2804
6	Logout	600	367	143
7	View a page activity	500	372	150
8	View course again	500	839	224
9	View a forum activity	500	1018	239
10	View a forum discussion	500	1127	514
11	Fill a form to reply a forum discussion	500	943	375
12	Send the forum discussion reply	500	388	147
13	View course once more	500	771	224
14	View course participants	500	468	131
15	TOTAL	7600	717	939

Table 4.14: Microservice LAMP JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	265	306	233	642	0.19
2	0.0%	135	153	114	181	0.19
3	0.0%	1921	2150	985	2774	0.18
4	0.0%	640	741	472	1584	0.18
5	9.2%	10280	10391	750	10482	0.18
6	9.2%	527	672	375	1479	0.16
7	0.0%	512	1334	331	1550	0.17
8	0.0%	1235	1662	757	2066	0.17
9	0.0%	1406	1882	917	2377	0.17
10	0.0%	1955	2380	1124	2777	0.17
11	0.0%	1595	1818	926	2381	0.17
12	0.0%	519	1317	358	1405	0.17
13	0.0%	1124	1650	682	2232	0.17
14	0.0%	595	1419	452	1569	0.17
15	1.4%	1542	2381	530	10482	2.00

This performance summary highlights key findings from the JMeter stress tests conducted on a Moodle e-learning platform using a LAMP microservices architecture. Just like with the monolithic LAMP setup, we simulate 100 consecutive users for the medium test, and 30 users for the small test.

The datasets reveal varying response times for each interaction. The slowest interaction for the medium test 4.13, "View course," had an average response time of 1811ms, which is a noticeable outlier. In the small test 4.11, the same interaction only averages 334ms, making "Login" at 1057ms the highest average value. The total average value of the medium test is 717ms, nearly double that of the small test at 387ms. In regards to course creation times, the small test 4.3 is noticeably faster at 55 seconds compared to the 268.3 seconds of the medium test 4.4.

The standard deviation values are much higher for the medium test 4.13, at 939.46, compared to the small test 4.11, at 237.20. This indicates that the medium test response times are more varied and less consistent. This is also reflected in the median and max response times. The median values give a clearer picture of the central tendency, such as 682ms for "View course once more" and 985ms for "Login". Additionally, maximum response times highlight the worst-case scenarios experienced during the test, with notable examples like 10482ms for "View course" and 2777ms for "View a forum discussion". It is worth noting that these values can in some cases be outliers that pull the average in one direction.

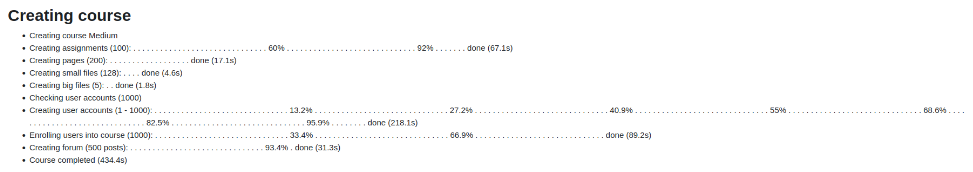
Notably, some errors were encountered during the medium test, specifically in the "View course" and "Logout" interactions, both with error percentages of 9.2%, so investigating the root causes of these errors would be crucial for maintaining platform stability and reliability. The errors could help explain why the max response time value of "View course" is much higher than the other values in the medium test.

4.2.2 LNMP

Cost estimate

Since the LNMP stack requires the same number of VMs at the same resource level, the cost will be kept unaltered. We will therefore refer to the previous cost tables 4.8. We assume that the consulting costs would be similar since Nginx is a standard web server technology that does not require highly specialised expertise.

Performance results



```
Creating course

• Creating course Medium
• Creating assignments (100): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92% . . . . . . . done (67.1s)
• Creating pages (200): . . . . . . . . . . . . . . . . . . . done (17.1s)
• Creating small files (128): . . . . done (4.6s)
• Creating big files (5): . . done (1.8s)
• Checking user accounts (1000)
• Creating user accounts (1 - 1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13.2% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27.2% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40.9% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68.6% . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . 82.5% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95.9% . . . . . . . . done (218.1s)
• Enrolling users into course (1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33.4% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66.9% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . done (89.2s)
• Creating forum (500 posts): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93.4% . done (31.3s)
• Course completed (434.4s)
```

Figure 4.5: Figure of microservice LNMP stack medium size test course creation time

Table 4.15: Microservice LNMP JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	186	15
2	View login page	600	120	93
3	Login	600	1075	445
4	Frontpage logged	600	478	215
5	View course	600	2049	3231
6	Logout	600	1379	3401
7	View a page activity	500	363	144
8	View course again	500	911	356
9	View a forum activity	500	1062	314
10	View a forum discussion	500	1202	608
11	Fill a form to reply a forum discussion	500	932	383
12	Send the forum discussion reply	500	354	113
13	View course once more	500	864	274
14	View course participants	500	455	136
15	TOTAL	7600	822	1447

Table 4.16: Microservice LNMP JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	206	222	185	384	0.19
2	0.0%	129	157	110	1138	0.19
3	0.0%	1666	2102	963	8046	0.18
4	0.0%	663	1454	426	3585	0.18
5	2.3%	11234	14330	948	14806	0.18
6	4.0%	10798	17354	325	19849	0.16
7	0.0%	527	1278	314	1408	0.17
8	0.0%	1470	1882	774	4053	0.17
9	0.0%	1613	2032	947	2578	0.17
10	0.0%	2319	2750	1177	3116	0.17
11	0.0%	1528	2088	877	2670	0.17
12	0.0%	479	608	319	1327	0.17
13	0.0%	1368	1883	817	2059	0.17
14	0.0%	649	1050	415	1479	0.17
15	0.5%	1634	10049	524	19849	2.00

This performance summary highlights key findings from the JMeter stress tests conducted on a Moodle e-learning platform using a LNMP microservices architecture. Unlike the previous stacks, where we tested for both 30 and 100 users, we are only focusing on the medium tests for 100 consecutive users moving forward. This is done to simplify the results for direct comparisons between stacks.

In the test summary table 4.15, the "View course" interaction exhibited the longest average response time at 2049ms, whereas the shortest average response time of 120ms was observed in the "View login page" interaction. The "Logout" interaction demonstrated the greatest variability in response times, as shown by its high standard deviation value. The course creation time of the medium test mas measured at 434.4 seconds according to figure 4.5.

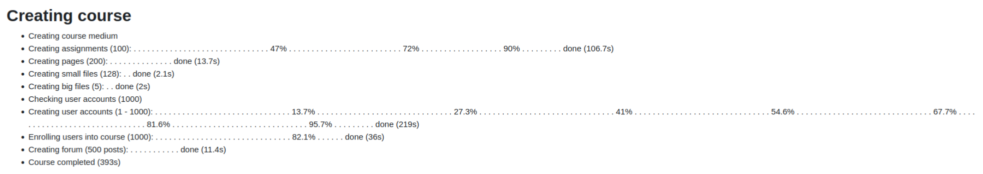
According to the test aggregate table 4.16, errors were detected in both the "View course" and "Logout" interactions, with error percentages of 2.3% and 4.0% respectively. These errors could help explain why the max response times, and therefore other values as well, are much higher than the values of other interactions. In addition, the maximum response times column showed a 8046ms response time for "Login.", which is a sizeable outlier, especially considering there were no errors detected for this interaction.

4.2.3 LAMP (MariaDB)

Cost estimate

The LAMP stack with MariaDB requires the same number of VMs at the same resource level as the previous stacks. Therefore, the cost here will also be kept unaltered. See the previous cost tables 4.8. Furthermore, we assume that the consulting costs do not differ from the cost of a microservice LAMP stack of Moodle since MariaDB does not require additional work compared to MySQL.

Performance results



Creating course

- Creating course medium
- Creating assignments (100): . . . . 47% . . . . 72% . . . . 90% . . . . done (106.7s)
- Creating pages (200): . . . . done (13.7s)
- Creating small files (128): . . done (2.1s)
- Creating big files (5): . . done (2s)
- Checking user accounts (1000)
- Creating user accounts (1 - 1000): . . . . 13.7% . . . . 27.3% . . . . 41% . . . . 54.6% . . . . 67.7% . . . . 81.6% . . . . 95.7% . . . . done (219s)
- Enrolling users into course (1000): . . . . 82.1% . . . . . . done (36s)
- Creating forum (500 posts): . . . . done (11.4s)
- Course completed (393s)

Figure 4.6: Figure of microservice LAMP (MariaDB) stack medium size test course creation time

Table 4.17: Microservice LAMP (MariaDB) JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	221	62
2	View login page	600	112	15
3	Login	600	1269	734
4	Frontpage logged	600	606	398
5	View course	600	3688	8459
6	Logout	600	348	177
7	View a page activity	500	412	207
8	View course again	500	895	274
9	View a forum activity	500	1198	369
10	View a forum discussion	500	1289	644
11	Fill a form to reply a forum discussion	500	1182	468
12	Send the forum discussion reply	500	318	231
13	View course once more	500	859	248
14	View course participants	500	424	155
15	TOTAL	7600	926	2567

Table 4.18: Microservice LAMP (MariaDB) JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	238	254	220	1245	0.19
2	0.0%	124	138	111	429	0.19
3	0.0%	1964	5210	1076	7858	0.18
4	0.0%	757	1664	533	6288	0.18
5	0.0%	30303	34803	824	35245	0.18
6	0.0%	390	1354	320	1871	0.16
7	0.0%	542	1411	359	1605	0.17
8	0.0%	1400	1807	788	2668	0.17
9	0.0%	2067	2567	1085	4343	0.17
10	0.0%	2349	3032	1176	4394	0.17
11	0.0%	1971	2511	1113	4903	0.17
12	0.0%	447	1321	258	3367	0.17
13	0.0%	1202	1890	767	2454	0.17
14	0.0%	531	1388	397	1431	0.17
15	0.0%	1682	4394	574	35245	2.00

This performance summary highlights key findings from the JMeter stress tests conducted on a Moodle e-learning platform using a LAMP microservices architecture, with MariaDB as the database instead of MySQL. The tests were conducted simulating 100 consecutive users to provide a medium load test.

Course creation time for the medium test was recorded at 393 seconds according to figure 4.6.

As shown in Table 4.17, the response times for each interaction varied significantly, with average values between 112ms and 3688ms for the "View login page" and "View course" interactions respectively. The "View course" interaction also had the highest maximum response time at 35245ms and the highest standard deviation of 8459ms, both substantial outliers compared to other response times. The total average response time for all interactions was 926ms, with a standard deviation of 2567ms.

Table 4.18 provides a more detailed view of the distribution of response times. When looking closer at the "View course" interaction, we can see it had the highest 95% Line and 99% Line values, indicating that 95% and 99% of the response times for this interaction were less than or equal to these values respectively. This interaction also had the highest median response time.

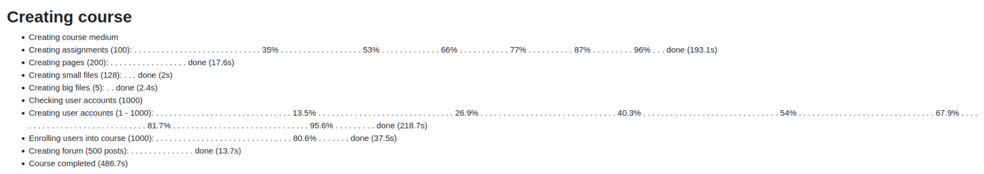
It was expected that the high values of the "View course interaction" could have been caused by errors in the test, just like in previous stacks. However, no errors were reported during the test.

4.2.4 LAPP (PostgreSQL)

Cost estimate

The infrastructure required for LAPP is identical to the previous stacks, therefore the cost of cloud resources remains the same. See the previous cost tables 4.8. In addition, we assume that the consulting costs remain the same as the microservice LAMP stack of Moodle since PostgreSQL does not require much more setup.

Performance results



```
Creating course

• Creating course medium
• Creating assignments (100): . . . . . . . . . . . . . . . . . . . . . . . . . . . 35% . . . . . . . . . . . . . . . . . 53% . . . . . . . . . . . . 66% . . . . . . . . . . . 77% . . . . . . . . . . 87% . . . . . . . . . 96% . . . done (193.1s)
• Creating pages (200): . . . . . . . . . . . . . . . . . done (17.6s)
• Creating small files (128): . . . done (2s)
• Creating big files (5): . . done (2.4s)
• Checking user accounts (1000)
• Creating user accounts (1 - 1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13.5% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26.9% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40.3% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67.9% . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . 81.7% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95.6% . . . . . . . . . done (218.7s)
• Enrolling users into course (1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80.6% . . . . . . . done (37.5s)
• Creating forum (500 posts): . . . . . . . . . . . . . . done (13.7s)
• Course completed (486.7s)
```

Figure 4.7: Figure of microservice LAPP stack medium size test course creation time

Table 4.19: Microservice LAPP JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	329	81
2	View login page	600	192	102
3	Login	600	1407	246
4	Frontpage logged	600	717	152
5	View course	600	2014	2828
6	Logout	600	461	195
7	View a page activity	500	516	235
8	View course again	500	1006	341
9	View a forum activity	500	1244	327
10	View a forum discussion	500	1432	639
11	Fill a form to reply a forum discussion	500	1362	529
12	Send the forum discussion reply	500	416	179
13	View course once more	500	963	255
14	View course participants	500	567	191
15	TOTAL	7600	898	995

Table 4.20: Microservice LAPP JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	421	437	314	1314	0.19
2	0.0%	199	288	182	1206	0.19
3	0.0%	1854	2306	1310	2831	0.18
4	0.0%	864	1692	690	1885	0.18
5	9.5%	10452	10541	976	10744	0.18
6	9.5%	592	799	483	2504	0.16
7	0.0%	665	1531	468	3547	0.17
8	0.0%	1732	2441	885	3525	0.17
9	0.0%	2041	2336	1129	4263	0.17
10	0.0%	2440	3097	1330	3977	0.17
11	0.0%	2187	2999	1242	4603	0.17
12	0.0%	529	1382	363	1550	0.17
13	0.0%	1352	1795	864	3993	0.17
14	0.0%	658	1548	526	1581	0.17
15	1.5%	1854	3105	707	10744	2.00

This performance summary highlights key findings from the JMeter stress tests conducted on a Moodle e-learning platform using a LAPP microservices architecture, with PostgreSQL as the database instead of MySQL. The tests were conducted simulating 100 consecutive users to provide a medium load test.

As illustrated in the summary table 4.19, the interaction "View course" was again a standout, exhibiting the highest average response time of 2014ms and a standard deviation of 2828ms. On the other hand, the "View login page" interaction showed the lowest average response time, which was 192ms. The total average response time across all interactions was 898ms, with a standard deviation of 995ms.

Table 4.20 provides more detailed statistics about the results. As usual, the "View course" interaction stands out, with a max value of 10744ms. Both the "View a forum activity" and "Fill a form to reply a forum discussion" max values were on the high side, measuring at 4263ms and 4603ms respectively. When looking at the 95% and 99% lines for the "View course" interaction, 95% of requests completed within 10452ms, and 99% of requests completed within 10541ms.

As previously observed in other stacks, there were errors detected in both the "View course" and "Logout" interactions, with error percentages at 9.5% for both interactions. Course creation time for the medium test was recorded at 486.7 seconds, as shown in figure 4.7.

4.2.5 Redundant LAMP

Cost estimate

A redundant microservice infrastructure would be more complex and require more VMs. Based on the same consulting cost of $75 to $150 and expect the setup to be more time intensive, it would be logical to estimate around 60 hours of consulting work. Accordingly, the estimated consulting- and setup costs would be between $4,500 to $9,000 (one-time cost).

Considering that the redundant setup would require an increase in VMs, we have recalculated the costs for each provider, factoring in one additional web server and database in the monthly cost.

Table 4.21: Redundant AWS Microservice LAMP Cost Estimate

Provider	Service	Instance	Estimated monthly cost [USD]
AWS	Web server 1	c6g.xlarge	$106
AWS	Web server 2	c6g.xlarge	$106
AWS	Database 1	t4g.large	$50
AWS	Database 2	t4g.large	$50
AWS	Load balancer	t4g.medium	$25
AWS	Bastion	t4g.medium	$25
AWS	Storage	6 x 40GB EBS	$19
-	-	-	$381 [50]

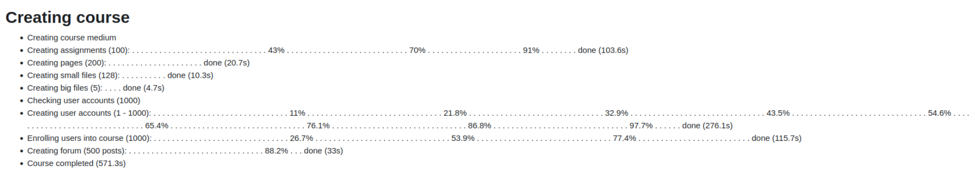
Table 4.22: Redundant GCP Microservice LAMP Cost Estimate

Provider	Service	Instance	Estimated monthly cost [USD]
GCP	Web server 1	e2-highcpu-4	$80
GCP	Web server 2	e2-highcpu-4	$80
GCP	Database 1	e2-standard-2	$54
GCP	Database 2	e2-standard-2	$54
GCP	Load balancer	e2-medium	$27
GCP	Bastion	e2-medium	$27
GCP	Storage	6 x 40GB Persistent Disk	$27
-	-	-	$349 [51]

Table 4.23: Redundant Azure Microservice LAMP Cost Estimate

Provider	Service	Instance	Estimated monthly cost [USD]
Azure	Web server 1	A4 v2	$282
Azure	Web server 2	A4 v2	$282
Azure	Database 1	D2as	$161
Azure	Database 2	D2as	$161
Azure	Load balancer	D2ls v5	$156
Azure	Bastion	D2ls v5	$156
Azure	Storage	6 x 64GB Premium SSD	$96
-	-	-	$1294 [52]

Performance results



```
Creating course

• Creating course medium
• Creating assignments (100): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43% . . . . . . . . . . . . . . . . . . . . . . . . . . . 70% . . . . . . . . . . . . . . . . . . . . . 91% . . . . . . . . done (103.6s)
• Creating pages (200): . . . . . . . . . . . . . . . . . . . . . . done (20.7s)
• Creating small files (128): . . . . . . . . . . done (10.3s)
• Creating big files (5): . . . . done (4.7s)
• Checking user accounts (1000)
• Creating user accounts (1 - 1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21.8% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32.9% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43.5% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54.6% . . . .
  . . . . . . . . . . . . . . . . . . . . . . . . . . 65.4% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76.1% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 86.8% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97.7% . . . . . . done (276.1s)
• Enrolling users into course (1000): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26.7% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53.9% . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77.4% . . . . . . . . . . . . . . . . . . . . . . . . . done (115.7s)
• Creating forum (500 posts): . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 88.2% . . . done (33s)
• Course completed (571.3s)
```

Figure 4.8: Figure of redundant LAMP stack medium size test course creation time

Table 4.24: Redundant LAMP JMeter Medium Summary (100 users, 5 loops)

#	Label (all measurements in ms)	# Samples	Average	Std Dev
1	Frontpage not logged	600	54022	57441
2	View login page	600	307	108
3	Login	600	52271	53251
4	Frontpage logged	600	55891	51487
5	View course	600	90243	54536
6	Logout	600	70669	57697
7	View a page activity	500	111521	47836
8	View course again	500	140999	46374
9	View a forum activity	500	167879	55840
10	View a forum discussion	500	207369	105304
11	Fill a form to reply a forum discussion	500	210005	103252
12	Send the forum discussion reply	500	3524	16163
13	View course once more	500	141934	68441
14	View course participants	500	107081	63326
15	TOTAL	7600	97263	88146

Table 4.25: Redundant LAMP JMeter Medium Aggregate (100 users, 5 loops)

#	Error %	95% Line	99% Line	Median	Max	Throughput [KB/s]
1	0.0%	163375	192393	36392	241471	0.09
2	0.0%	460	529	309	1074	0.09
3	0.0%	147641	188258	33983	210838	0.08
4	0.0%	139489	183029	59991	216679	0.08
5	0.0%	174108	207344	100126	250174	0.08
6	0.0%	173281	197707	72717	230222	0.07
7	0.0%	181138	223291	114072	249146	0.07
8	0.0%	213244	238890	142695	259531	0.07
9	0.0%	252451	273919	172068	304337	0.07
10	4.2%	416070	464923	209312	489972	0.07
11	2.6%	422238	459721	217130	473571	0.06
12	2.6%	1895	112369	1076	145461	0.06
13	0.0%	239854	269715	155522	319092	0.06
14	0.0%	203342	226208	114844	237223	0.06
15	0.6%	250437	364275	93543	489972	0.88

This performance summary highlights key findings from the JMeter stress tests conducted on a Moodle e-learning platform using a LAMP microservices architecture with added redundancy, doubling the working components of the architecture. The tests were conducted simulating 100 consecutive users to provide a medium load test.

The redundant LAMP microservice architecture demonstrated significantly slower performance compared to the previous configurations. A look at the summary table 4.24 shows notably high average response times across all operations, ranging from 307ms for the "View login page" interaction, to 210005ms for the "Fill a form to reply a forum discussion" interaction. The substantially higher values are also reflected in the total average and standard deviation, measuring in at 97263 ms and 88146ms respectively.

The Aggregate table 4.25 reveals that the 95% and 99% lines are high for most interactions, with the exception of "View login page". The "Send the forum discussion reply" interaction also measures a much lower 95% line, at 1895ms, compared to its 99% line, at 112369ms. The "View a forum discussion" interaction measures the highest max response time value, at a substantial 489972ms.

There were also some errors detected during testing. More specifically, the "View a forum discussion" operation showed a high error rate of 4.2%. Other errors include the "Fill a form to reply to a forum discussion" and "Send the forum discussion reply" interactions, with error rates of 2.6%.

The course creation time was recorded at 571.3 seconds, as shown in Figure 4.8.

Chapter 5

Discussion

5.1 Scalability

5.1.1 Cost

In order to provide a comprehensive understanding of the cost of deploying Moodle in different environments, we will analyse and discuss the results obtained from the cost estimates for various infrastructure setups and providers, as presented in Tables 4.2, 4.8, 4.9, 4.10, 4.21, 4.22, and 4.23.

Firstly, the monolithic LAMP stack, as a baseline, offers a straightforward and simple setup, with the lowest initial consulting costs. Its estimated monthly costs are approximately \$213 for AWS, \$220 for GCP, and \$680 for Azure, as shown in Table 4.2. While it has the advantage of being easy to set up and manage, the monolithic architecture lacks flexibility and scalability, which may be an issue for growing institutions or those with varying workloads.

As a more distributed and resilient architecture, the microservice LAMP stack provide better fault tolerance and can be scaled independently based on individual service needs. Microservices offers more flexibility and scalability, but its increased complexity leads to higher consulting costs. The estimated monthly costs for this setup are roughly \$219 for AWS, \$206 for GCP, and \$819 for Azure, as presented in Tables 4.8, 4.9, and 4.10. Interestingly, the monthly costs for AWS and GCP are very similar, or in some cases even lower, than the monthly cost of a Monolithic setup at the same providers. There are many possible explanations for this: microservice architecture could for example utilise resources in a better way than its monolithic counterpart, leading to cost savings. Additionally, inaccuracies in our estimates relating to instance types and required specifications might affect the results.

Figure 5.1 presents an estimated cost comparison of the various architectures across the different providers we have examined. As noted earlier, the monthly costs for both AWS and GCP appear to be significantly lower than those of Azure, regardless of the architecture in question.

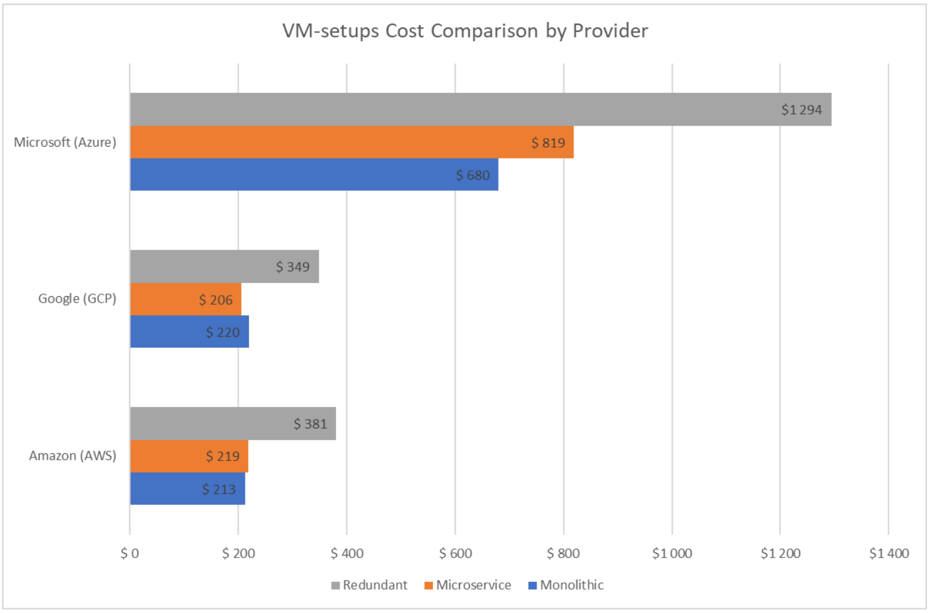


Figure 5.1: Figure of bar chart of VM-setups Estimated Cost Comparison by Provider

The alternative setups, including LNMP, LAMP with MariaDB, and LAPP with PostgreSQL, have similar costs to their LAMP counterparts, since the number of VMs and resources required are the same. The choice of web server or database technology would primarily depend on the organisation's preferences, technical expertise, and specific requirements.

Lastly, the redundant microservice LAMP stack is designed for enhanced fault tolerance and resiliency. Its estimated monthly costs are approximately $381 for AWS, $349 for GCP, and $1,294 for Azure, as illustrated in Tables 4.21, 4.22, and 4.23. However, its increased complexity results in higher initial consulting costs. This setup may be most suitable for organisations that require high availability and are willing to invest in the additional infrastructure and setup costs.

5.1.2 Performance

This section provides a detailed evaluation of the performance of Moodle deployment configurations under various stress loads. We will review and discuss the results obtained from the JMeter stress tests conducted on six architectural setups: monolithic LAMP, microservice LAMP, microservice LNMP, microservice LAMP (MariaDB), microservices LAPP, and redundant microservice LAMP. We aim to highlight key performance differences across the configurations and gain insights that could help with the selection of an optimal setup. The tests were conducted using the JMeter tool, with the load profile set to medium, simulating 100 consecutive users. Additionally, small tests were conducted on the monolithic LAMP and microservice LAMP architectures, giving us test results with varied infrastructure scaling.

The following graphs compares the stacks performances. For the sake of readability, we are excluding the results from the redundant LAMP setup in some of the graphs.

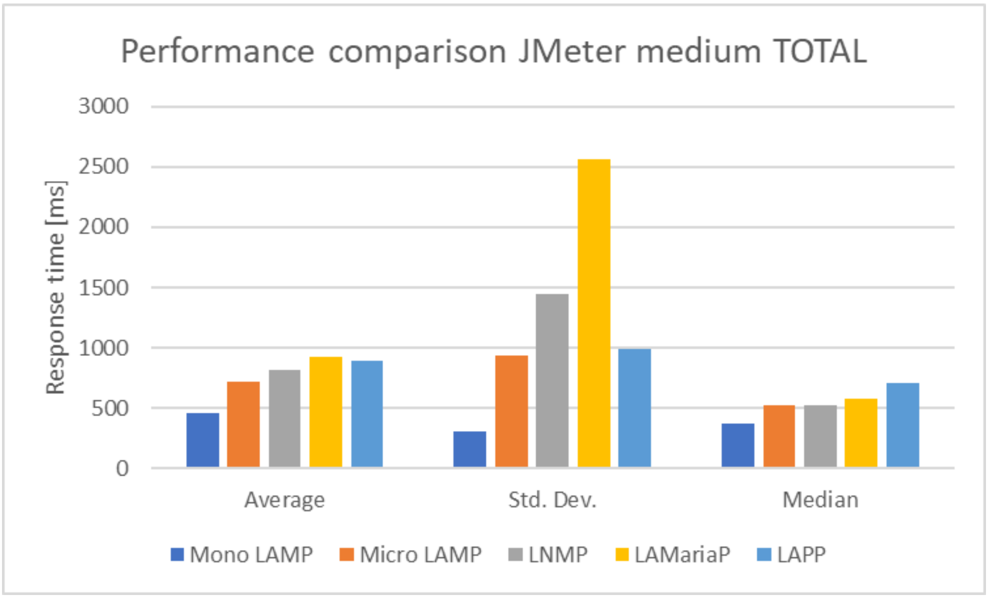


Figure 5.2: Graph of performance comparison on the medium size test

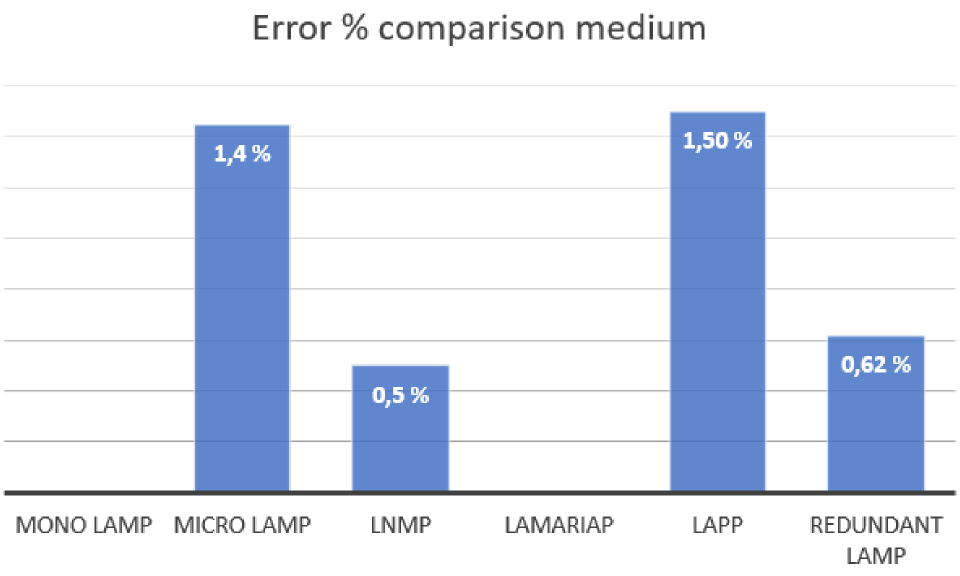


Figure 5.3: Graph of error percentage on the medium size test for all stacks

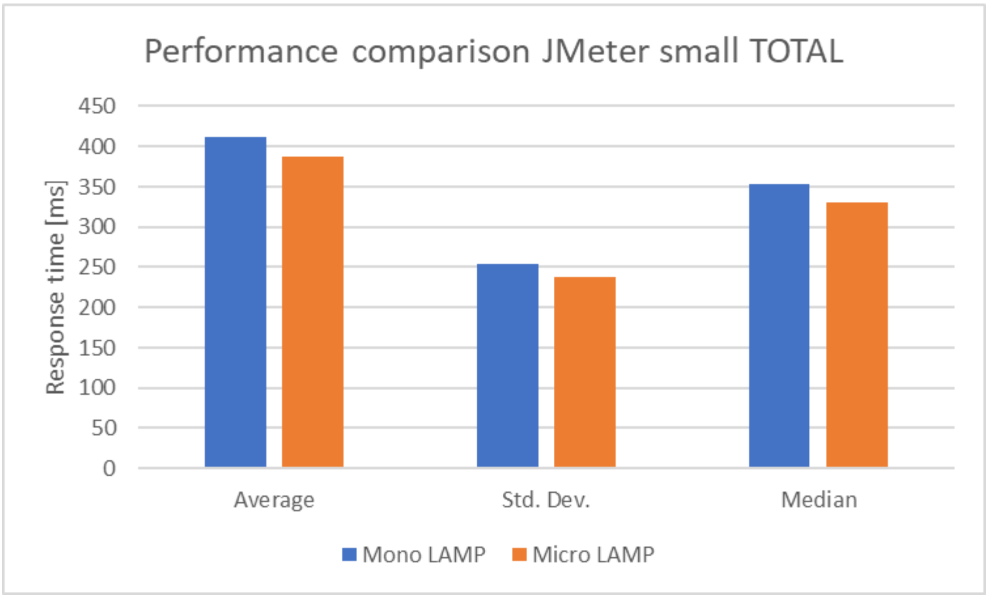


Figure 5.4: Graph of performance comparison on the small size test

The monolithic LAMP configuration performed the best out of all the stacks on the medium tests. When looking at the figure 5.2, we can see that the average, standard deviation, and median values of monolithic LAMP are lower than other configurations. This suggests that, purely performance wise, ignoring other benefits from a distributed microservice architecture, a monolithic LAMP configuration is the way to go. The low latency of having all the components of the configuration localised in the same server, in combination with the lower capacity of VMs in microservice architectures, most likely contributed to this performance disparity.

The LAMP microservice configuration performed well in the medium tests, although it is outperformed by the monolithic LAMP stack in regards to average, standard deviation, and median values. Most notably, the standard deviation values are much higher for the microservice stacks, suggesting that the response times are more varied and less consistent in comparison to the monolithic architecture.

The small tests revealed the opposite, with the microservice LAMP stack outperforming the monolithic, although only slightly. As seen in the small test comparison graph 5.4, the average, standard deviation, and median values of microservice LAMP are consistently lower than its counterpart. Despite the result, it is important to note that the test data could be affected by a range of factors, which we will discuss further in the reliability and validity section. Nevertheless, the results suggests that the capacity of microservice VMs are not a bottleneck in the small tests.

Regarding the presence of test errors, only the monolithic LAMP and MariaDB LAMP stacks avoided errors altogether, suggesting that these stacks could handle medium load without issues. For the remaining stacks, most errors were concentrated around the same interactions, namely the "View course" and "Logout" operations. We theorise that this is due to the "View course" interaction being too heavy in regards to system stress, causing errors that follows in to the next interaction on the table. Nevertheless, the presence of errors could negatively impact the overall system reliability and performance.

Lastly, the redundant LAMP microservice configuration demonstrated drastically slower performance compared to the other setups, as seen in the tables 4.24 and 4.25.The average response times across all operations were significantly higher, and a few interactions reported errors, although interestingly on interactions that have not reported any previous errors. These performance results reveal that even though redundancy can provide increased availability and resilience, it may also result in slower response times under medium load conditions, possibly due to the overhead of managing and synchronising the redundant components.

5.2 Availability

5.2.1 Cost

A highly available Moodle deployment requires a redundant infrastructure to minimise the impact of failures and reduce downtime. Implementing redundancy results in an increase in the number of VMs, leading to higher monthly costs, as shown in Tables 4.21, 4.22, and 4.23. Additional costs for monitoring, backup, and disaster recovery solutions could also be factored in, as these tools are crucial for maintaining high availability.

Downtime can have significant financial consequences, such as lost productivity, revenue, and potential damage to an organisation's reputation. The microservice architecture, due to its inherent fault tolerance and distributed nature, can help reduce the likelihood and duration of downtime. By isolating failures and allowing for independent scaling and maintenance of services, thereby reducing the amount of downtime in total, microservices could lead to lower overall costs in the long run.

Institutions with a large user base or those that rely heavily on Moodle may find that the investment in a highly available, redundant microservice LAMP stack is justified to minimise downtime and ensure consistent access to course materials. For smaller organisations or those using Moodle for training purposes, a simpler monolithic LAMP setup may suffice, with occasional downtime being an acceptable trade-off for lower infrastructure and maintenance costs.

5.2.2 Redundancy

As seen in our results including redundancy can become costly due to the increase in VMs. The same goes for high-availability. Each instance of Moodle has their own requirements, budget, system resources, user scale and position within the business layer. So what setups of Moodle should have redundancy and high-availability?

A university like NTNU which has tens of thousands of students who depend on the Moodle solution for some courses should have a scalable, secure, reliable and highly available setup. They could in practice schedule server updates during the nighttime, however this might impact international students or students working night shifts.

A small local school who uses Moodle for student or teacher training and courses might opt for an easy setup and may not have the same need for high availability. It would still be important that the solution is operational under expected load, but fewer demands might be imposed for up-time.

A company delivering Moodle as a Service will have to tailor their solution to their client, but their core business layers in the performance and ease of their solution compared to businesses doing it themselves.

A company planning to use Moodle as tool for training internally would again have different needs. Since Moodle is not an integral part in their core business operations their reputation or finances would not take an impact from the Moodle service having down-time. However, it might result in employees having their training delayed or inconvenienced, which might in turn further down the line impact them financially. Therefore, for a small to medium sized business implementing redundancy may be insightful or technically rewarding, but not necessary. It is unlikely that the investment cost will be leveraged by the increase in availability. However, building a solid foundation, with a microservice approach, would allow for the company to scale up their Moodle implementation if they ever need a larger scale or need to implement further redundancy on specific services.

5.3 Security evaluation

When looking at the security for Moodle, one of the first thing that needs to be ensured and kept up to date is the web server running the Moodle PHP application itself.

Another step that will generally apply to all implementations of Moodle, except for some isolated and internal implementations, would be to ensure the use of HTTPS with a proper certificate from either a service such as Let's Encrypt or a more commercial Certificate Authority (CA). This, along with keeping software updated, are both recommendations made by Moodle in their documentation[53]. HTTPS or SSL can help prevent attackers from intercepting and modifying this data, even if they have network access. For instance, confidential data could be intercepted by programs like Wireshark, where unencrypted content can be viewed in plain text.

Additional security considerations, which we implemented, include the use of a Bastion host with rules on the network and/or servers that only permit connections over certain ports from this host, such as over SSH. This can help mitigate many potential security issues as we can restrict outbound connections to ports

80 and 443. When using HTTPS correctly, there should be no need to allow connections over port 80, ensuring all connections are encrypted.

5.3.1 Monolithic

The monolithic setup poses several risks, including being a Single Point of Failure (SPOF) and being vulnerable to ransomware attacks. In addition, those who opt for a monolithic setup may not be as IT-literate and when following a guide forget to change default passwords or configurations, increasing their vulnerability to attacks.

To mitigate these risks, it is crucial to implement proper security measures such as regularly updating software and changing passwords. Additionally, having data backed up on a secondary machine is essential since backing up data on the same machine would still leave the data vulnerable to ransomware attacks, disasters, system failures or data corruption.

Updating a monolithic setup can be challenging since all components are tightly coupled, making it difficult to update individual components without experiencing downtime. This issue can lead to businesses being hesitant to update their monolithic applications, leaving them vulnerable to security risks and other issues.

5.3.2 Microservice

All the security measures mentioned above are essential for a microservice stack. Since a microservice architecture typically involves multiple small services working together to perform a larger function, it is crucial to ensure that each service is secure and protected against potential vulnerabilities.

Backing up data is particularly important in a microservice architecture, where data is spread across multiple services. In the event of a security breach or data loss, having backups can help ensure that critical data is not lost. A load balancer can also be especially important for a microservice architecture, where traffic may be distributed unevenly across services. By distributing traffic evenly, a load balancer can help ensure that all services are operating efficiently and effectively.

The principle of least privileges is also critical in a microservice architecture, as each service may be handling different parts of an organisation's data. By limiting access, the risk of data loss or unauthorised access is minimised. Similarly, using strong administrator passwords can help ensure that only authorised personnel can make changes to the system.

Finally, even if a microservice architecture is running purely on an intranet, using HTTPS or SSL is still essential, as discussed previously. In a microservice architecture, services are likely to communicate with each other over a network, and this communication may involve sensitive or confidential data. Overall, implementing these security measures is critical for a secure and robust microservice architecture.

5.3.3 Redundant

A redundant microservice setup offers both benefits and challenges in terms of security measures. One of the main advantages of redundant services is that they help maintain the availability of critical functions during service failures or security breaches. This prevents data loss or system downtime, both of which can have severe security consequences.

However, redundant services may also expand the attack surface of a system, potentially raising the risk of security breaches. For instance, if a service in a redundant setup becomes compromised, the attack might propagate more readily to other redundant services. Moreover, managing redundant services introduces complexity, necessitating supplementary security measures like load balancers or redundancy protocols.

To address these challenges, it is important to adopt security practices tailored for distributed systems. For instance, conducting regular data backups across all redundant services can help safeguard data even during an attack. Additionally, implementing load balancing is essential for ensuring that all redundant services operate optimally without being overwhelmed, which could expose them to attacks.

5.3.4 Comparison

In a monolithic architecture, security measures tend to be more straightforward, as all components of an application are combined into a single codebase. Backing up data and securing the single codebase can often be achieved through a single backup and security solution. Similarly, load balancing may not be as critical, as there are no separate services to balance traffic across.

In a microservice architecture, security measures must be more nuanced and complex. Backing up data may require multiple backup solutions to ensure that all data across all services is protected. Load balancing is critical to ensure that traffic is distributed evenly across all services. The principle of least privileges is also more challenging to implement, as there are multiple services, each with its own set of access requirements and privileges.

A redundant microservice setup can help ensure system availability in the event of a service failure or security breach. However, redundant services can also increase the attack surface of a system, potentially increasing the risk of security breaches. Managing redundant services can be more complex, requiring additional security measures such as load balancers or redundancy protocols.

In terms of results, a monolithic architecture has the advantage of being a simpler and more contained system, making it potentially easier to secure. However, it also means that a single vulnerability can compromise the entire system. A microservice architecture allows for greater flexibility and scalability, but requires more complex security measures to ensure the security of each individual service. A redundant microservice setup can help ensure system availability, but requires additional security measures to ensure that all redundant services remain secure.

In summary, each architecture has its own unique security challenges and requirements. It is important to consider the potential impact of each architecture on security measures and take appropriate steps to ensure that the system remains secure. This may require additional security measures specifically designed for distributed systems or redundancy protocols.

5.4 Monitoring

In this section, we discuss the monitoring, automation and documentation for the e-learning platform with a focus on maintenance, as highlighted by our results. We will focus on the ease of setup and use, as well as the potential for automation when managing the platform.

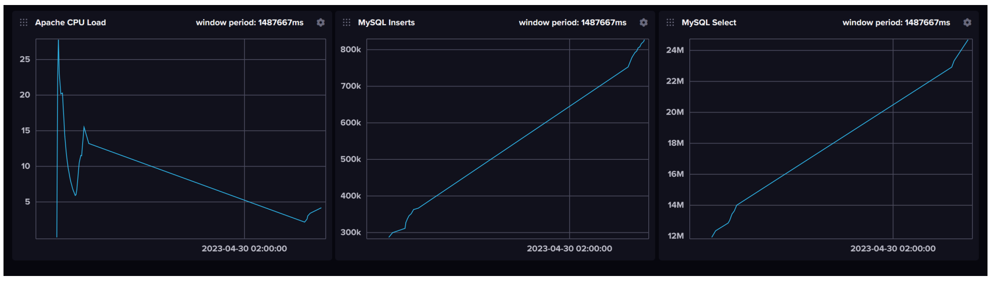


Figure 5.5: Screengrab of InfluxDB dashboard for Monolithic Medium test course creation and test

5.4.1 Monitoring and Documentation

Monitoring When performing our testing using JMeter test plans we were going to use the TI-stack to provide extra data for how these tests affected the CPU, Memory, Requests to the web server and Queries on the database. However, we didn't realise that the way we implemented the Telegraf agent with fetching its configuration by way of the InfluxDB server required the terminal window to stay open when executing and the configuration was not saved locally. We managed to gather some data during our initial baseline testing, which can be seen in figure 5.5, however this data has a time skip when the SSH timed out and stopped the Telegraf agent and we didn't start Telegraf correctly when running our tests on the different microservice implementations.

Documentation The documentation that is already available on Moodle, through their own documentation pages[1], covers for the most part only setting up Moodle in a monolithic manner, which is how we implemented our LAMP monolithic stack, as we mentioned in section 3.3.1. When it comes to available documentation for implementing Moodle with using microservices or in a high availability or

[1]See `https://docs.moodle.org/`

redundant manner they have some links to external sources where other people have done this but in a very singular manner. An example of this is the linked guide for setting up a highly available Moodle instance by several9s, which only talks about implementing it using a specific cluster management platform, ClusterControl[54]. This way of implementing Moodle clustered might be a good way of doing it but it requires the reader to understand much of how to change different aspects to work in their setup and it isn't a official Moodle guide so there is no immediate way to tell if it will work correctly or if there are special considerations that you have to take during the setup. In addition, there exists Docker images for Moodle as Bitnami LMS provided by VMware on docker hub which we have not explored, but for those who know Docker this might be a fine choice[55].

Ease of Setup and Use

Ease of Setup From our experience and results after setting up the different implementations we have looked there are some that were easier and some that were harder to implement. Specifically the LAMP monolithic stack, section 3.3.1, was much easier as it is covered directly in the Moodle documentation as mentioned previously when talking about documentation. On the opposite end was the setup of the redundant LAMP Microservice stack, as seen in section 3.3.6, where we followed to some extent the guide we mentioned in previously as it was linked from the Moodle documentation. Since we didn't run the cluster management platform that they used we had to make different choices and had to make our own choices on different aspects, in addition to this it was very hard to automate so this was done entirely manually.

User Experience and System Management The user experience and simplicity of managing the system varied somewhat between the different implementations, being easiest on our test using Ansible to setup the LAMP microservice stack. On a larger view when including considerations for the infrastructure and platform used for this it varies widely between providers, where we used NTNU's OpenStack solution SkyHiGh. As it appears to be a somewhat default OpenStack solution it was easy to find general OpenStack documentation on how to perform different actions such as setting up an internal network, provision hosts or add extra storage volumes. If instead we had used a different self-hosted cloud platform, say built entirely on VMware ESXi, the user experience and system management on the lowest level would be quite different and most likely more challenging. Another alternative, as we have mentioned before, would be to host it on a platform such as AWS that has an entirely different user experience again. Some of these experiences could be alleviated and be the same if we instead used Terraform to handle provisioning instead of "native" CLI belonging to the used platform.

Automation

Our investigation revealed that there are opportunities for automation in deployment, monitoring, backups, and scaling. The extent of automation depends on whether the infrastructure is owned by the admins, you go for Infrastructure as a Service (IaaS) or the cloud provider. Some technologies may trade simplicity for performance, but automation can help reduce the complexity of setup and management tasks while still reaping the performance benefits.

As we talked about in section 3.2.4 we would look at different choices for automating different parts of deployment and monitoring. We looked first at using HOT for provisioning and deployment with the use of bash scripts to automate the setup of the servers, which can be seen for the LAMP Monostack in appendix A.1. This provisioning could have been done instead with the use of Terraform as this is more flexible and is platform agnostic, that is it can be used on any cloud or data centre[56], whether Terraform or HOT is the better choice of provisioning IaC is beyond the scope of this paper.

The next step was to look at using either Puppet or Ansible as alternatives to using shell scripts with cloud-init for the configuration of the servers. The first thing that can be noted here is that Puppet is much better suited for long-term deployments and deployments that don't expect rapid re-provisioning since it is harder to grasp, can be daunting and more stable. However, as we were doing exactly this as part of the first two research questions, Ansible is much better as it works better with short-term or rapid re-provisioned systems and is easier to grasp and understand[57]. Secondly Puppet requires the use of a puppet agent service on the target hosts in addition to a puppet server service on the controller, whereas Ansible only requires Ansible being installed on the controller.

Employing tools like Puppet and Ansible can automate and streamline the majority of management tasks, such as software updates and Moodle updates. As configuration management and patching in an organisation is so important and can at times become hard, especially if working across multiple server, stacks or environments. Implementing a tool such as Puppet or Ansible allows for consistency in configurations, patch versioning and other maintenance tasks instead of doing it manually or using a collection of shell scripts[58]. However, given our main focus on accessibility and scalability the usage of these as management tools is something that was not focused on and they were primarily examined for deployment purposes.

As an alternative to managing the implementations directly on the server using Puppet or Ansible, we considered containerising Moodle with technologies like Kubernetes or Docker. This approach would potentially simplify infrastructure setup, offering greater flexibility in hosting options and possibly higher availability, especially when using a provider that supports clustered Kubernetes or Docker swarm. However, this exploration was primarily conducted through literature review and not applied in practice. We concluded that this approach warrants further investigation and presents an opportunity for future work.

Lastly we looked at using the TI-stack as a monitoring tool to keep track of CPU-usage, traffic and query execution while also looking at the possibility of setting up InfluxDB to handle automated alerts if the data passed set limits. We installed the Telegraf agent on each server, as can be seen in code 5.1, and then used the configuration downloaded from the InfluxDB instance when running the agent. An alternative way of doing this is to edit the configuration file for Telegraf on the server such that the required plugins and endpoint is already there and the agent can be run as a service.

Code listing 5.1: Bash shell script snippet from haproxy_reverse.sh

```bash
echo "Adding repositories, updating, installing and upgrading software..."
# influxdata-archive_compat.key GPG fingerprint:
#     9D53 9D90 D332 8DC7 D6C8 D3B9 D8FF 8E1F 7DF8 B07E
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

apt-get -y install haproxy telegraf net-tools
```

5.5 Reliability and validity of the results

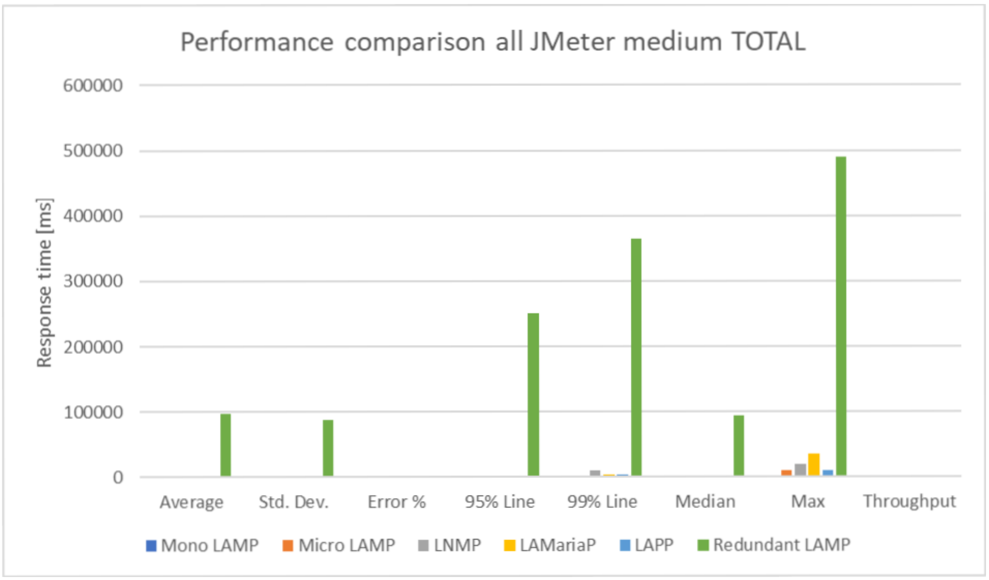


Figure 5.6: Graph of performance comparison on the medium size test for all stacks

When it comes to the reliability and validity of data, it is important to mention how the test results did not stay consistent throughout our testing of the various stacks. For example, we noticed differences in results from tests of the same stacks. This was especially noticeable for the LAPP configuration. By running the same test on the same setup at different time intervals, reveals that the performance differences could indicate that other users activity, caching and VM distribution across different compute nodes on SkyHiGh, can drastically affect the results.

The test results for the redundant stack are much worse than anticipated, as can be seen in figure 5.6 where the other stacks are near impossible to discern, especially when compared to figure 5.2. We could not find a specific underlying reason for the poor performance, so we can only speculate why this was the case. However, we theorise that this might be due to how the redundant infrastructure was set up and configured.

Chapter 6

Conclusion

6.1 Conclusion

6.1.1 What bottlenecks occur when scaling the e-learning platform?

As seen in the results and discussion chapters, scaling the e-learning platform could introduce bottlenecks affecting both cost and performance. The monolithic LAMP stack, while cost-effective and straightforward, lacks the scalability required for expanding organisations, leading to potential bottlenecks as system demands and cost increases.

On the other hand, microservice-based architectures offer superior scalability, but at the expense of higher initial consulting costs and increased complexity.In our case, the added complexity led to worse performance. Redundancy, while enhancing availability and resilience, also introduced severe performance bottlenecks, evidenced by the drastically slower response times in the redundant LAMP microservice configuration under medium load conditions.

In conclusion, performance bottlenecks emerge mainly due to the capacity limitations of the virtual machines in microservice architectures, as observed in the stress tests where the monolithic LAMP stack outperformed the microservice LAMP stack. Error occurrences in several interactions for most microservice stacks could also degrade system reliability and performance.

6.1.2 What considerations must be made to ensure redundancy for the e-learning platform?

To ensure redundancy for the e-learning platform, it's crucial to consider the platform's unique requirements, user scale, and its role in the organisation. Large-scale institutions may benefit from a highly available, redundant microservice LAMP stack to minimise downtime.

In order to ensure redundancy a duplication of working components or services is required. Expanding the infrastructure comes at a financial and technical cost.

During our implementation of a redundant microservice LAMP stack we did not achieve the results we had expected. We experienced increased loading times and reduced performance. The complexity of the resulting microservice stack was such that the considerations for how to connect the parts and ensure redundancy most likely led to this poorer performance.

On the other hand, smaller organisations or those using Moodle for training might find a simpler monolithic LAMP setup sufficient, where occasional downtime is an acceptable trade-off for lower costs. Businesses providing Moodle as a Service should tailor their solution to client needs, balancing performance, cost, and ease of use. For internal corporate use of Moodle, where the impact of downtime is less critical, a microservice approach might offer a scalable base for future growth and provide redundancy if needed.

6.1.3 How to secure the e-learning platform from unauthorised access and other vulnerabilities?

To ensure security in an e-learning platform, we advise adhering to the security recommendations outlined by Moodle Docs as a baseline. The attack surface and number of potential attack vectors will depend on various factors including the system's size, complexity, utilised technologies, network configuration, and access levels. However, one universal strategy is to keep software updated, follow strict security routines, and maintain appropriate access levels.

We recommend implementing the principle of least privileges. This strategy minimises potential damage, as each user has only the access they need to perform their duties. This approach can be effectively integrated within a microservices architecture, where the attack surface is already reduced by exposing only one service - such as a reverse proxy or load balancer - to external networks.

The use of secure network protocols, such as HTTPS, is critical for protecting data traffic from unauthorised access and interception. Avoiding default passwords and educating users about the risks of social engineering attacks further contributes to the overall security of the system.

However, securing the e-learning platform from other vulnerabilities will always be an evolving task due to the continual emergence of new threats. This challenge necessitates ongoing vigilance and regular system updates, as well as the use of firewalls and the expertise of security professionals. While it's impossible to guarantee 100% security, staying current and closely monitoring the system will help maintain a robust defence against potential vulnerabilities.

6.1.4 What should be documented and monitored to maintain and service the e-learning platform?

To maintain and service the e-learning platform, comprehensive documentation and diligent monitoring are crucial. Documentation should include a wide range of topics such as setup procedures for Moodle deployments, both monolithic and microservices-based, and the potential use of containerisation technologies like Kubernetes or Docker. It's also important to detail the use of configuration management tools like Ansible and Puppet, emphasising their role in managing short-term or rapidly re-provisioned systems.

On the monitoring side, it's vital to consistently track performance metrics, such as CPU usage, web server requests, and database queries. Tools like the TI-stack can be instrumental for this purpose. Additionally, setting up automated alerts for anomalies via InfluxDB is recommended to promptly address any issues. Regularly assessing user experience across different implementations ensures consistent performance and availability. It's also important to monitor aspects like other users' activity, caching, and VM distribution, as these can impact performance results.

Overall, diligent documentation and monitoring combined with efficient use of automation tools can significantly enhance the maintenance and servicing of the e-learning platform.

6.1.5 What are sensible choices of services and technologies for implementing a scalable and secure e-learning platform?

If you know your user size will stay consistent, want an easy setup that is well documented, do not have major security concerns, and want a reliable and well performing solution, then the monolithic LAMP stack would be our recommendation. This is mainly due to the fact that it performed the best out of all stacks during our testing, as seen in graph 5.2.

If you anticipate user growth necessitating scaling, prioritise security, are willing to accept less documentation and undertake more necessary configuration, the LAMP or LNMP microservice stacks may be for you. Although LAMP micro outperformed LNMP micro on both average and standard deviation values, as seen in graph 5.2, we believe some of this may be attributed to noise or other variables. Therefore, a re-run of the tests are likely to give new results. Nginx is a widely adapted web server and if you are comfortable with it and the options it provide, it may serve as a better web server than Apache.

If you need a highly available or redundant setup, we would advise going for a distributed LAMP or LNMP stack as mentioned above. However, you would need to research how to properly implement it. Our redundant setup did unfortunately not give us valid data to support the claim that a redundant setup performs well. Despite this, we believe that it is still possible to create a good, highly available and redundant clustered solution.

6.2 Future work

As we mentioned throughout the discussion we believe there are multiple areas that would both require and be interesting to look at further such as, but not limited to, the following; monitoring for data collection, redundancy, containerisation and LDAP.

Our research was limited to conducting small and medium JMeter tests on small and medium test courses. However, it would be interesting to explore the results from larger tests and courses. Furthermore, gathering data through test with JMeter and gathering real-time monitoring data from the TI-stack would give correlative data. Utilising TI plugins would give insight into all VMs CPU and RAM usage and additional data from web servers or databases. The following github page could act as a good starting point: `https://github.com/influxdata/community-templates/tree/master/apache_jmeter`

We did not get the results we were expecting for our redundant implementation and view these results as invalid. It would therefore be interesting to explore what results a properly set up redundant implementation would lead to.

Containerising Moodle using either Docker or Kubernetes could be an interesting avenue of research. As it would be interesting to see the ease of containerisation and explore the performance differences between virtual and physical hardware.

For those that need or are looking for a SSO option or use LDAP we have the start of a simple Ansible playbook for FreeIPA on the gitlab for the project, obs-bachelor, it might be beneficial to also look at the experiment/freeipa branch.

Bibliography

[1] Wikipedia. (2023), [Online]. Available: https://en.wikipedia.org/wiki/Certificate_authority (visited on 04/03/2023).

[2] C. Coronel, S. Morris and K. Crockett, *Database Principles: Fundamentals of Design, Implementation, and Management*, 3rd ed. Cengage, 2020, ISBN: 978-1-47376-806-2.

[3] Wikipedia. 'General data protection regulation.' (2023), [Online]. Available: https://en.wikipedia.org/wiki/General_Data_Protection_Regulation (visited on 10/04/2023).

[4] Wikipedia. 'Lightweight directory access protocol.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol (visited on 20/05/2023).

[5] Wikipedia. 'Transport layer security.' (2023), [Online]. Available: https://en.wikipedia.org/wiki/Transport_Layer_Security (visited on 04/03/2023).

[6] Moodle Community. 'Moodle docs - installation quick guide.' (2022), [Online]. Available: https://docs.moodle.org/401/en/Installation_quick_guide (visited on 23/01/2023).

[7] Moodle Community. 'Moodle docs - oracle.' (2017), [Online]. Available: https://docs.moodle.org/401/en/Oracle (visited on 23/01/2023).

[8] Moodle Community. 'Moodle docs - large installations.' (2022), [Online]. Available: https://docs.moodle.org/401/en/Large_installations (visited on 23/01/2023).

[9] A. Al-Ajlan and H. Zedan, 'Why moodle,' in *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*, 2008, pp. 58–64. DOI: 10.1109/FTDCS.2008.22.

[10] T. Martín-Blas and A. Serrano-Fernández, 'The role of new technologies in the learning process: Moodle as a teaching tool in physics,' *Computers and Education*, vol. 52, no. 1, pp. 35–44, 2009, ISSN: 0360-1315. DOI: https://doi.org/10.1016/j.compedu.2008.06.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036013150800095X.

[11] S. H. P. W. Gamage, J. R. Ayres and M. B. Behrend, 'A systematic review on trends in using moodle for teaching and learning,' *International Journal of STEM Education*, vol. 9, no. 1, p. 9, Jan. 2022, ISSN: 2196-7822. DOI: 10.1186/s40594-021-00323-x. [Online]. Available: https://doi.org/10.1186/s40594-021-00323-x.

[12] D. Miletic, *Moodle Security: Learn how to install and configure Moodle in the most secure way possible*. Packt Publishing, 2011, ISBN: 978-1-84951-264-0.

[13] S. Kumar and K. Dutta, 'Investigation on security in lms moodle,' *International Journal of Information Technology and Knowledge Management*, vol. 4, no. 1, pp. 233–238, 2011.

[14] J. C. G. Hernandez and M. A. L. Chavez, 'Moodle security vulnerabilities,' in *2008 5th International Conference on Electrical Engineering, Computing Science and Automatic Control*, 2008, pp. 352–357. DOI: 10.1109/ICEEE.2008.4723399.

[15] D. Amo, M. Alier, F. J. García-Peñalvo, D. Fonseca and M. J. Casany, 'Gdpr security and confidentiality compliance in lms' a problem analysis and engineering solution proposal,' in *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, ser. TEEM'19, León, Spain: Association for Computing Machinery, 2019, pp. 253–259, ISBN: 9781450371919. DOI: 10.1145/3362789.3362823. [Online]. Available: https://doi.org/10.1145/3362789.3362823.

[16] Microsoft. 'What is virtualization - definition: Microsoft azure.' (2023), [Online]. Available: https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-virtualization/ (visited on 22/02/2023).

[17] R. J. Chevance, *Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, and Storage Solutions*. Elsevier Digital Press, 2004, ISBN: 978-1-55558-333-0.

[18] K. Morris, *Infrastructure as Code: Managing Servers in the Cloud*, 1 (Early Release). O'Reilly Media, 2015, ISBN: 978-1-49192-435-8.

[19] J. Arundel, K.-J. C. Hsu, N. Khare, H.-C. C. Lee, H. Saito and T. Uphill, *DevOps: Puppet, Docker, and Kubernetes: get hands-on recipes to automate and manage Linux containers with the Docker 1.6 environment and jump-start your Puppet development*. Packt Publishing, 2017, ISBN: 978-1-78829-761-5.

[20] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen and B. Warren, 'Let's encrypt: An automated certificate authority to encrypt the entire web,' CCS '19, pp. 2473–2487, 2019. DOI: 10.1145/3319535.3363192. [Online]. Available: https://doi.org/10.1145/3319535.3363192.

[21] W3Techs. ‘Usage statistics of web servers.’ (6th Mar. 2023), [Online]. Available: `https://w3techs.com/technologies/overview/web_server` (visited on 06/03/2023).

[22] Netcraft. ‘February 2023 web server survey.’ (28th Feb. 2023), [Online]. Available: `https://news.netcraft.com/archives/2023/02/28/february-2023-web-server-survey.html` (visited on 06/03/2023).

[23] R. J. T. Morris and B. J. Truskowski, ‘The evolution of storage systems,’ English, *IBM Systems Journal*, vol. 42, no. 2, p. 205, 2003, Copyright - Copyright International Business Machines Corporation 2003; Last updated - 2022-10-20; CODEN - IBMSA7, ISSN: 0018-8670. [Online]. Available: `https://www.proquest.com/scholarly-journals/evolution-storage-systems/docview/222419839/se-2`.

[24] X. Lin, *Introductory Computer Forensics: A Hands-on Practical Approach*. Springer Cham, 2018, ISBN: 978-3-030-00580-1. DOI: `10.1007/978-3-030-00581-8`. [Online]. Available: `https://doi.org/10.1007/978-3-030-00581-8`.

[25] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas and L. Vivier, ‘The new ext4 filesystem: Current status and future plans,’ in *Proceedings of the Linux symposium*, Citeseer, vol. 2, 2007, pp. 21–33.

[26] R. Shirey, ‘Internet security glossary,’ IETF, RFC 4949, version 2, Aug. 2007. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc4949.txt`.

[27] S. Laan, *IT Infrastructure Architecture: Infrastructure Building Blocks and Concepts*, 3rd ed. Lulu Press Inc., 2017, ISBN: 978-1-32691-297-0.

[28] Microsoft. ‘Scaling up vs scaling out.’ (2023), [Online]. Available: `https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/scaling-out-vs-scaling-up/` (visited on 27/02/2023).

[29] anappi. ‘Haproxy canary deployment.’ (16th Jan. 2018), [Online]. Available: `https://db-blog.web.cern.ch/blog/antonio-nappi/2018-01-haproxy-canary-deployment` (visited on 06/03/2023).

[30] Conviva. (5th Aug. 2015), [Online]. Available: `http://www.conviva.com/canary-deployment-with-nginx` (visited on 06/03/2023).

[31] Arch Linux. (2022), [Online]. Available: `https://wiki.archlinux.org/title/TICK_stack` (visited on 04/03/2023).

[32] InfluxData. [Online]. Available: `https://www.influxdata.com/wp-content/uploads/Influx-1.0-Diagram_04.20.2020v2.png` (visited on 04/03/2023).

[33] InfluxData. ‘Telegraf 1.25 documentation.’ (2023), [Online]. Available: `https://docs.influxdata.com/telegraf/v1.25/` (visited on 04/03/2023).

[34] InfluxData. ‘Get started with influxdb oss 2.6.’ (2023), [Online]. Available: `https://docs.influxdata.com/influxdb/v2.6/` (visited on 04/03/2023).

[35] InfluxData. 'Chronograf 1.10 documentation.' (2023), [Online]. Available: https://docs.influxdata.com/chronograf/v1.10/ (visited on 04/03/2023).

[36] InfluxData. 'Kapacitor 1.6 documentation.' (2023), [Online]. Available: https://docs.influxdata.com/kapacitor/v1.6/ (visited on 04/03/2023).

[37] InfluxData. 'The tick stack.' (2023), [Online]. Available: https://www.influxdata.com/time-series-platform/ (visited on 04/03/2023).

[38] T. Edgar and D. Manz, *Research Methods for Cyber Security*, 1st ed. Syngress, 2017, ISBN: 978-0-12812-930-2.

[39] InfluxData. (2023), [Online]. Available: https://docs.influxdata.com/influxdb/v2.6/security/enable-tls/ (visited on 09/03/2023).

[40] InfluxData. (2023), [Online]. Available: https://docs.influxdata.com/kapacitor/v1.6/administration/security/ (visited on 09/03/2023).

[41] InfluxData. (2023), [Online]. Available: https://docs.influxdata.com/chronograf/v1.10/administration/managing-security/ (visited on 09/03/2023).

[42] A. K. Mudiyanselage and L. Pan, 'Security test moodle: A penetration testing case study,' *International Journal of Computers and Applications*, vol. 42, no. 4, pp. 372–382, 2020. DOI: 10.1080/1206212X.2017.1396413. eprint: https://doi.org/10.1080/1206212X.2017.1396413. [Online]. Available: https://doi.org/10.1080/1206212X.2017.1396413.

[43] DB-Engines. 'Db-engines ranking.' (2023), [Online]. Available: https://db-engines.com/en/ranking (visited on 12/05/2023).

[44] DB-Engines. 'Db-engines ranking - trend popularity.' (2023), [Online]. Available: https://db-engines.com/en/ranking_trend (visited on 12/05/2023).

[45] Netcraft. 'April 2023 web server survey.' (2023), [Online]. Available: https://news.netcraft.com/archives/category/web-server-survey/ (visited on 14/05/2023).

[46] Stack Overflow. 'Stack overflow 2022 developer survey.' (2022), [Online]. Available: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-databases (visited on 12/05/2023).

[47] Moodle. 'Load testing moodle with jmeter.' (2023), [Online]. Available: https://docs.moodle.org/dev/Load_testing_Moodle_with_JMeter (visited on 02/05/2023).

[48] Moodle. 'Test course generator.' (2023), [Online]. Available: https://docs.moodle.org/402/en/Test_course_generator (visited on 06/05/2023).

[49] Moodle. 'Jmeter test plan generator.' (2023), [Online]. Available: https://docs.moodle.org/402/en/JMeter_test_plan_generator (visited on 06/05/2023).

[50] Amazon Web Services. 'Aws pricing calculator.' (2023), [Online]. Available: https://calculator.aws/#/addService (visited on 01/05/2023).

[51] Google Cloud Platform. 'Google cloud pricing calculator.' (2023), [Online]. Available: `https://cloud.google.com/products/calculator` (visited on 01/05/2023).

[52] Microsoft Azure. 'Azure pricing calculator.' (2023), [Online]. Available: `https://azure.microsoft.com/en-us/pricing/calculator/` (visited on 01/05/2023).

[53] Moodle Docs. 'Security recommendations.' (2021), [Online]. Available: `https://docs.moodle.org/402/en/Security_recommendations` (visited on 14/05/2023).

[54] A. Sharif. 'Clustering moodle on multiple servers for high availability and scalability.' (2020), [Online]. Available: `https://severalnines.com/blog/clustering-moodle-multiple-servers-high-availability-and-scalability/` (visited on 13/03/2023).

[55] VMware. 'Bitnami docker image for moodle.' (2023), [Online]. Available: `https://hub.docker.com/r/bitnami/moodle` (visited on 14/05/2023).

[56] HashiCorp. 'Terraform by hashicorp.' (2023), [Online]. Available: `https://www.terraform.io/` (visited on 13/05/2023).

[57] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, 'Devops,' *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016. DOI: `10.1109/MS.2016.68`.

[58] S. Thakur, S. C. Gupta, N. Singh and S. Geddam, 'Mitigating and patching system vulnerabilities using ansible: A comparative study of various configuration management tools for iaas cloud,' in *Information Systems Design and Intelligent Applications*, S. C. Satapathy, J. K. Mandal, S. K. Udgata and V. Bhateja, Eds., New Delhi: Springer India, 2016, pp. 21–29, ISBN: 978-81-322-2755-7.

Appendix A

Code Appendix

The appended code files are strictly referenced within the thesis, for all our code see our gitlab `https://gitlab.stud.idi.ntnu.no/obs-bachelor/obs-bachelor`.

A.1 Monolithic Shell Setup Code

Code listing A.1: Shell script for LAMP Monolithic stack automated setup and configuration

```bash
#!/bin/bash

## Variables

ENGINE=apache2 # nginx or apache2
DOMAIN=${PUBLIC_IP} #Set to 'public' ip
DBTYPE=mysqli #'pgsql', 'mariadb', 'mysqli', 'auroramysql', 'sqlsrv' or
    'oci'
DBNAME=moodle
DBUSER=moodleuser
DATAROOT=/var/moodledata

DBHOST=127.0.0.1
DBPASS=MoodleP@ssword1

DBROOT=MoodleSuperSecure

ADMINPASS=P@ssword1
ADMINEMAIL='admin@mail.com'
FSITENAME='Moodle-site'
SSITENAME='moodle'

## Hosts file config

# Add manager to hosts file
echo "192.168.0.50 manager.lab manager" >> /etc/hosts
echo "192.168.0.60 influx.lab influx" >> /etc/hosts

# Set hostname
echo "$(hostname -I) $(hostname -s).lab $(hostname -s)" >> /etc/hosts
hostnamectl set-hostname $(hostname -s).lab

## Ubuntu Config ##

sed -i 's+//      "\${distro_id}:\${distro_codename}-updates";+
    "\${distro_id}:\${distro_codename}-updates";+'
    /etc/apt/apt.conf.d/50unattended-upgrades
systemctl restart unattended-upgrades

## Installations ##

echo "Adding repositories, updating, installing and upgrading software..."
# influxdata-archive_compat.key GPG fingerprint:
#     9D53 9D90 D332 8DC7 D6C8 D3B9 D8FF 8E1F 7DF8 B07E
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
add-apt-repository ppa:ondrej/php
add-apt-repository ppa:ondrej/apache2

apt-get update

apt-get -y install git curl telegraf graphviz ghostscript clamav
    php7.4-pspell php7.4-curl php7.4-gd php7.4-intl php7.4-xml
    php7.4-xmlrpc php7.4-ldap php7.4-zip php7.4-soap php7.4-mbstring
    apache2 libapache2-mod-php7.4 aspell php7.4 php7.4-mysqli mysql-server

#apt-get update  # might be unneeded

#unattended-upgrade -d

## PHP Config
echo "Changing PHP config"
sed -i "s:memory_limit = 128M:memory_limit = 256M:"
    /etc/php/7.4/apache2/php.ini
sed -i "s:;cgi.fix_pathinfo = 1:cgi.fix_pathinfo = 0:"
    /etc/php/7.4/apache2/php.ini
sed -i "s:upload_max_filesize = 2M:upload_max_filesize = 100M:"
    /etc/php/7.4/apache2/php.ini
sed -i "s:max_execution_time = 30:max_execution_time = 360:"
    /etc/php/7.4/apache2/php.ini
sed -i "s:;date.timezone =:date.timezone = Europe/Oslo:"
    /etc/php/7.4/apache2/php.ini

## MySQL setup and config
echo "Configuring MySQL"

mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED WITH
    mysql_native_password BY '${DBROOT}';"
mysql -u root -p${DBROOT} -e "ALTER USER 'root'@'localhost' IDENTIFIED
    WITH auth_socket;"

mysql -u root -p${DBROOT} -e "CREATE DATABASE moodle DEFAULT CHARACTER SET
    utf8mb4 COLLATE utf8mb4_unicode_ci;"
mysql -u root -p${DBROOT} -e "CREATE user '${DBUSER}'@'localhost'
    IDENTIFIED BY '${DBPASS}';"
mysql -u root -p${DBROOT} -e "GRANT
    SELECT,INSERT,UPDATE,DELETE,CREATE,CREATE TEMPORARY
    TABLES,DROP,INDEX,ALTER ON moodle.* TO '${DBUSER}'@'localhost';"

## Moodle download and move ##
echo "Starting Moodle download"

```

```
cd /opt

git clone git://git.moodle.org/moodle.git

cd moodle

git branch --track MOODLE_401_STABLE origin/MOODLE_401_STABLE

git checkout MOODLE_401_STABLE

cp -R /opt/moodle /var/www/html/

mkdir /var/moodledata

## Moodle config ##

echo "Changing Moodle config"
sed -i "s:\$CFG->dbtype    = 'pgsql';:\$CFG->dbtype    = '${DBTYPE}';:"
    /var/www/html/moodle/config-dist.php
sed -i "s:\$CFG->dbname    = 'moodle';:\$CFG->dbname    = '${DBNAME}';:"
    /var/www/html/moodle/config-dist.php
sed -i "s:\$CFG->dbuser    = 'username';:\$CFG->dbuser    = '${DBUSER}';:"
    /var/www/html/moodle/config-dist.php
sed -i "s:\$CFG->dbpass    = 'password';:\$CFG->dbpass    = '${DBPASS}';:"
    /var/www/html/moodle/config-dist.php
sed -i "s+\$CFG->wwwroot   = 'http://example.com/moodle';+\$CFG->wwwroot
    = 'http://${DOMAIN}';+" /var/www/html/moodle/config-dist.php
sed -i "s:\$CFG->dataroot  = '/home/example/moodledata';:\$CFG->dataroot
    = '${DATAROOT}';:" /var/www/html/moodle/config-dist.php
sed -i "s://   \$CFG->tool_generator_users_password =
    'examplepassword';:\$CFG->tool_generator_users_password = 'moodle';:"
    /var/www/html/moodle/config-dist.php #testing param

cp /var/www/html/moodle/config-dist.php /var/www/html/moodle/config.php

chown -R www-data:www-data /var/moodledata
chown -R www-data:www-data /var/www/html/moodle

## Web server config

echo "Changing Apache2 DocumentRoot"
sed -i "s:DocumentRoot /var/www/html:DocumentRoot /var/www/html/moodle:"
    /etc/apache2/sites-available/000-default.conf
sed -i "s:#Require ip 192.0.2.0/24:Require ip 192.168.0.0/24:"
    /etc/apache2/mods-enabled/status.conf
#Require ip 192.0.2.0/24

service apache2 restart

## Moodle install ##
```

```

echo "Moodle installation starting"
/usr/bin/php /var/www/html/moodle/admin/cli/install_database.php
    --agree-license --adminpass=${ADMINPASS} --adminemail=${ADMINEMAIL}
    --fullname=${FSITENAME} --shortname=${SSITENAME}

## Add cron ##

# Add a cron job for the www-data user to run Moodle's cron script
(crontab -u www-data -l ; echo "*/1 * * * * php -q -f
    /var/www/html/moodle/admin/cli/cron.php") | crontab -u www-data -

## Reboot ##

#echo "Time to reboot"
#reboot
```