

Joonas Hiltunen

COMPARISON OF CLOUD PLATFORMS FOR HOSTING MICROSERVICES

Determining the best cloud platform for
microservice hosting by costs and maintainability

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiner: David Hästbacka
Examiner: Kari Systä
April 2024

ABSTRACT

Joonas Hiltunen: Comparison of cloud platforms for hosting microservices
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
April 2024

As microservices, containerisation, and related tools have matured significantly in the past years, many companies have started to decouple their monolithic software products into smaller microservices. With this change, the companies have faced a new problem: how to host these new services efficiently and how to keep these services maintainable without spending vast amounts of money on DevOps.

The customer of this thesis was developing a customer information system, or CIS, that is used extensively within a niche industry and is looking for the best possible cloud platform for their needs. The software was quite large and complex, consisting of multiple separate modules. At the time of writing, the software was being converted from a modular monolith to a microservices solution. The customer desired to determine the cheapest and the most maintainable cloud platform for their needs.

The research was done using Design Science Research introduced by Peffers et al. The research was classified as a technical experiment that outputs an artefact, which in this case was the advisory of the best cloud platform for the software.

To determine the best-suited cloud platform for the future microservice-based solution, two of the largest cloud platforms, Amazon Web Services and Microsoft Azure, were compared. Multiple relevant platforms' services, such as Kubernetes service, virtual machines, and databases, were compared to determine the best options for the software. A simple definition for the maintainability of a cloud platform was formed, as no peer-reviewed standard definition for it could be found. The platforms were then compared based on multiple subjective and objective metrics described in the definition. The possibility of using proprietary services such as AWS Elastic Container Service was also explored.

The results showed that Amazon Web Services was the cheaper option. Simultaneously, Microsoft Azure was deemed better in terms of maintainability, with the most significant differences coming from higher uptime service level agreements of Azure. Proprietary services were considered and compared, but it was determined that they were not the right choice for complex software such as the one used in this research. In the end, a recommendation was given for Amazon Web Services if there are no other reasons to use Azure, such as previous know-how.

Keywords: Cloud platforms, cloud computing, microservices, hosting, scalability, maintainability

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Joonas Hiltunen: Pilvialustojen vertailu mikropalveluiden ylläpidon näkökulmasta
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Huhtikuu 2024

Mikropalveluiden, konttipalveluiden ja niihin liittyvien työkalujen viime vuosien merkittävän kehittymisen vuoksi monet yritykset ovat alkaneet hajottaa monoliittisiä ohjelmistotuotteitaan pienemmiksi mikropalveluiksi. Tämän muutoksen myötä yritykset ovat kohdanneet uuden ongelman: kuinka ylläpitää näitä uusia palveluita tehokkaasti ja miten nämä palvelut pidetään ylläpidettävänä ilman, että joudutaan käyttämään suuria summia rahaa DevOpsiin.

Tämän opinnäytetyön asiakas oli kehittämässä erikoisalalla laajasti käytettyä asiakastietojärjestelmää eli CIS-ohjelmistoa ja etsi parasta mahdollista pilvialustaa tarpeisiinsa. Ohjelmisto oli melko laaja sekä monimutkainen, ja se koostui useista erillisistä moduuleista. Kirjoitushetkellä ohjelmistoa oltiin muuntamassa modulaarisesta monoliitista mikropalveluratkaisuksi. Asiakas halusi selvittää halvimman, mutta myös ylläpidettävimmän pilvialustan tarpeisiinsa.

Tutkimus tehtiin Peffers et al.:n esittelemän Design Science Researchin avulla. Tutkimus luokiteltiin tekniseksi kokeeksi, joka tuottaa artefaktin eli tässä tapauksessa neuvon parhaasta pilvialustasta tarpeeseen.

Tulevalle mikropalvelupohjaiselle ratkaisulle parhaiten soveltuvan pilvialustan määrittämiseksi verrattiin kahta suurinta pilvialustaa, jotka olivat kirjoitushetkellä Amazon Web Services ja Microsoft Azure. Parhaiden vaihtoehtojen määrittämiseksi verrattiin useita alustojen palveluita, kuten Kubernetes-palvelua, virtuaalikoneita ja tietokantoja. Pilvialustan ylläpidettävyydestä muodostettiin yksinkertainen määritelmä, sillä sille ei löytynyt vertaisarvioitua standardimääritelmää. Tämän jälkeen alustoja verrattiin useiden määritelmässä kuvattujen subjektiivisten ja objektiivisten mittareiden perusteella. Myös mahdollisuutta käyttää alustojen omia palveluja, kuten AWS Elastic Container Serviceä tutkittiin.

Tulokset osoittivat, että Amazon Web Services oli halvempi vaihtoehto kahdesta vertaillusta palvelusta. Samaan aikaan Microsoft Azure voitiin kuitenkin pitää ylläpidettävyyden kannalta parempana, suurimmat erot tulivat esiin Azuren paremmista palvelutasosopimuksista. Alustojen omia palveluja pohdittiin ja vertailtiin, mutta lopulta todettiin, että ne eivät olleet oikea valinta tässä tutkimuksessa käytetyin kaltaisille monimutkaisille ohjelmistoille. Lopulta annettiin suositus Amazon Web Servicesille, mikäli Azuren käyttöön ei ole muita syitä, kuten aiempaa osaamista.

Avainsanat: Pilvialustat, pilvipalvelut, mikropalvelut, isännöinti, skaalautuvuus, huollettavuus

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

PREFACE

The work for this thesis was carried out for a customer project while I was working for Vincit Oyj. I want to thank Vincit and the client for allowing me to work on this thesis.

I especially want to thank my supervisor Jukka from the client, who acted as my instructor on the thesis. I am thankful for Jukka's insights and continuous guidance. He held weekly meetups for me and the other students doing their theses for the client, where we could exchange ideas and peer review each other's work.

I also want to thank Enni and my family for their continuous support, which meant a lot to me. I appreciated their encouragement and support through the ups and downs of the thesis writing process.

Tampere, 8th April 2024

Joonas Hiltunen

CONTENTS

1.INTRODUCTION.....	1
1.1 Research aim	1
1.2 Research Methodology	2
1.3 Structure of the research	4
2.CLOUD COMPUTING	6
2.1 Definition of cloud computing.....	6
2.1.1 Essential characteristics of the cloud model	6
2.1.2 Service models of cloud computing.....	7
2.1.3 Deployment models of cloud computing	8
2.2 Virtualisation.....	9
2.2.1 Virtual machines.....	9
2.2.2 Containerization	10
2.2.3 Container orchestration.....	11
2.3 Microservices	14
2.4 Serverless computing	16
2.5 Related research.....	17
3.MAINTAINABILITY OF CLOUD PLATFORMS	20
3.1 Overview	20
3.2 Definition in this thesis	20
4.CLOUD PLATFORMS	23
4.1 Amazon Web Services.....	23
4.2 Microsoft Azure	24
5.THE ANALYSED SOFTWARE PRODUCT	26
5.1 Overview of the analysed software.....	26
5.2 The software architecture of the analysed software.....	26
5.3 The current hosting solution of the analysed software.....	28
5.4 Steps required to host a new instance of the analysed software.....	30
5.5 Future development of the analysed software	30
6.DESIGN.....	33
6.1 Determining the hosting requirements of the software.....	33
6.2 Comparison of the cloud platforms based on the requirements	33
6.3 Designing example hosting solutions for the software	34
6.3.1 The designs on Amazon Web Services	35
6.3.2 The products and pricing of Amazon Web Services	38
6.3.3 The designs on Microsoft Azure	41
6.3.4 The products and pricing of Microsoft Azure.....	43
7.EVALUATION	48
7.1 Evaluating the platforms and designs based on costs	48

7.2	Evaluating the platforms and designs based on maintainability	51
7.3	Evaluating the possible use of vendor locking services and tools	54
8.	DISCUSSION	57
8.1	Limitations	57
8.2	Further development.....	57
9.	CONCLUSIONS.....	59
	REFERENCES	61

LIST OF FIGURES

<i>Figure 1.1 Design Science Research cycles adapted from Hevner (2007)</i>	2
<i>Figure 1.2 DSRM process model adapted from Peffers et al. (2007)</i>	4
<i>Figure 2.1 A comparison of server virtualisation and containerisation adapted from Erl, Mahmood and Puttini (2013.)</i>	11
<i>Figure 2.2 A simple overview of Kubernetes topology adapted from Newman (2021, p. 260)</i>	13
<i>Figure 2.3 Microservices allow for easier use of multiple technologies adapted from Newman (2021, p. 23)</i>	15
<i>Figure 2.4 An example of a FaaS containing one function for each endpoint as adapted from Newman (2021, p. 254)</i>	17
<i>Figure 5.1 Example of modular monolith architecture adapted from Newman (2021, p. 16)</i>	27
<i>Figure 5.2 A basic overview of the structure of the software</i>	28
<i>Figure 5.3 Simplified diagram of the current hosting solution of the analysed software</i>	29
<i>Figure 5.4 Example of the deployment of the concurrency test</i>	32
<i>Figure 6.1 Example of a deployment of a single microservice on AWS</i>	36
<i>Figure 6.2 Example of multiple microservices running under the same EKS Cluster with shared databases</i>	37
<i>Figure 6.3 Example of multiple fully independent microservices running under the same EKS Cluster</i>	38
<i>Table 1 Minimum and maximum price per month for each major service in AWS</i>	41
<i>Figure 6.4 Example of a deployment of a single microservice on Azure</i>	42
<i>Figure 6.5 Example of multiple microservices running under the same AKS Cluster with shared databases</i>	42
<i>Figure 6.6 Example of multiple fully independent microservices running under the same AKS Cluster</i>	43
<i>Table 2 Minimum and maximum price per month for each major service in Azure</i>	46
<i>Table 3 Comparison between services provided by AWS and Azure</i>	50
<i>Table 4 Uptime SLAs for different AWS and Azure services</i>	53
<i>Table 5 Comparing the platforms based on factors of maintainability</i>	54
<i>Table 6 Comparing vendor-specific services to Kubernetes</i>	56

LIST OF SYMBOLS AND ABBREVIATIONS

SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
FaaS	Function as a Service
BaaS	Backend as a Service
CIS	Customer Information System
VM	Virtual Machine
vCPU	Virtual Central Processing Unit
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
ELB	Elastic Load Balancer
ALB	Application Load Balancer
EKS	Elastic Kubernetes Service
AKS	Azure Kubernetes Service
VMSS	Virtual Machine Scale Set
SLA	Service Level Agreement

1. INTRODUCTION

With the substantial maturation of microservices, containerisation, and associated techniques in recent years, many companies have begun to break down their large software products into smaller microservices. With this shift, businesses must figure out how to host these new services effectively and manage them without shelling out a ton of cash for DevOps.

1.1 Research aim

This thesis aims to explore the most significant cloud platforms for hosting these kinds of microservices that are decoupled from monoliths and compare these platforms based on costs and maintainability. The aim is to design the best possible hosting solution for the client after the decoupling of their monolithic software has been completed while avoiding vendor lock-in. The main focus of this approach is on the costs of the finished system.

The second approach is determining the best platform in terms of maintainability. The aim is to use the same cloud platforms and designs used in the first approach, but the evaluation is done based on the maintainability of the finished system instead of costs.

The third approach is analysing if using platform-specific tools would allow for better solution maintainability even though it would mean a vendor lock-in. The aim is to determine if using platform-specific tools in the designs would be better.

Combining these three approaches makes it possible to select the best platform and design for hosting the software. Hence, the thesis is guided by the following research questions:

1. How do AWS and Azure compare in terms of costs and maintainability for hosting decoupled monolithic software?
2. Is it better for maintainability to use platform-specific services and tools even though it would mean vendor lock-in?

1.2 Research Methodology

The methodology of the research done in this thesis is design research science. Design research differs from empirical research as it doesn't just seek to describe, explain, and predict but instead seeks to develop new innovative artefacts which can help people overcome their problems. Design science, the selected research methodology for this thesis, is a special part of design research that is used mostly with information systems and IT. (Hevner and Chatterjee, 2010, pp. 1–8; Johannesson and Perjons, 2014, pp. 1–14)

A design research science project can be displayed as *Design Science Research Cycles*, as shown in Figure 1.1. The figure consists of three cycles: *Relevance Cycle*, *Rigor Cycle*, and *Design Cycle*. The *relevance cycle* is the first of the cycles and connects the design science activities and the research project's contextual environment. The *rigor cycle*, on the other hand, establishes a link between the design science activities and the knowledge, experience, and expertise underpinning the research project. An internal part of the design science research process, the *design cycle* alternates between the main tasks of building and evaluating the research's design artefacts and processes. (Hevner, 2007)

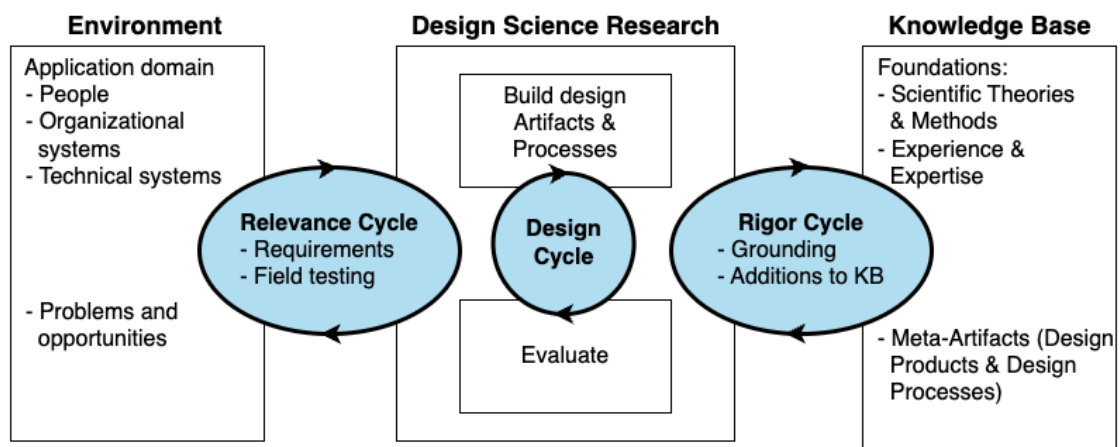


Figure 1.1 Design Science Research cycles adapted from Hevner (2007).

Chapters two, three, and four contain a comprehensive analysis of the fundamental concepts of cloud computing, the specific cloud platforms chosen, and their respective features. Based on the theoretical information in these chapters, the cloud platforms will be compared in chapter six, and designs will be created for all the cloud platforms. In Chapter seven, an evaluation of the designs, which are a key part of IT-based artefact design, is made regarding the research questions of the thesis (Hevner and Chatterjee,

2010, pp. 109–119). In conclusion, the recommended cloud platform will be selected, and it is the produced artefact of this thesis.

Peppers *et al.* (2012) study and list the different evaluation methods used in design science research papers and the artefact types produced by those papers. The produced artefacts were classified as an algorithm, construct, framework, instantiation, method, and model. The evaluation methods listed in the paper were logical argument, expert evaluation, technical experiment, subject-based experiment, action research, prototype, case study, and illustrative scenario.

According to the previously mentioned artefacts and evaluation methods, this project could be categorised as a technical experiment with the produced artefact type of instantiation. However, the evaluation method could also be described as a prototype. Peppers *et al.* (2012) define a technical experiment as an assessment of the performance of an algorithm implementation. This assessment can be done using real-world data, synthetic data, or no data at all. Instead of evaluating performance based on real-world circumstances, the focus is on evaluating technical performance. On the other hand, the prototype is described as an implementation of an artefact with the goal of proving its usefulness or suitability. Instantiation is described as the structure and organisation of a system's hardware, system software, or a component thereof, which could, in this context, be thought of as the structure of the recommended microservice solution. (Peppers *et al.*, 2012)

Peppers *et al.* (2007) introduced a process model for Design Science Research Methodology (DSRM), shown in Figure 1.2. The process model offers a methodical process for creating and assessing creative artefacts that address particular issues and challenges in real-world settings. The DSRM process model consists of six iterative steps, first, the problem and motivation are identified. Then the objectives and the predicted accomplishments of the better artefact of the new solution are defined. Third, the new artefact is designed and developed. After development, the use of the new artefact to solve the problem is demonstrated. The fifth step includes the evaluation of the effectiveness of the developed artefact. Depending on the results of the evaluation, the process returns to either objective definition or development or if the results are positive, the process can continue to the final step. The final step includes communicating the results via scholarly and professional publications. This process helps researchers identify relevant problems in the field and develop innovative artefacts to address those problems.

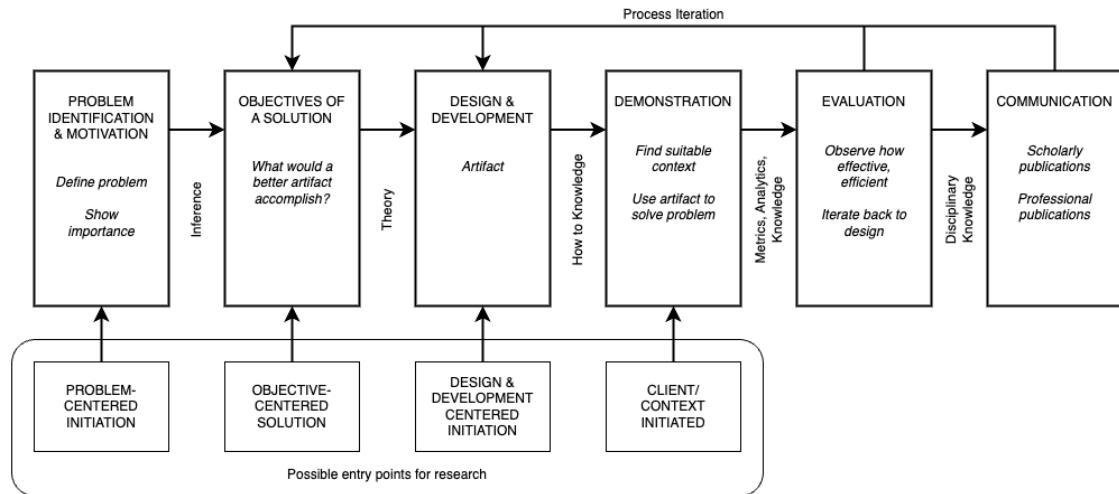


Figure 1.2 DSRM process model adapted from Peffers et al. (2007)

In this project, the problem identification and objective of a solution can be found at the beginning of this chapter. The actual design and development can be found in chapter six, where some possible hosting solutions, which are this project's artefacts, are designed. The artefacts are then evaluated in the following chapter seven.

1.3 Structure of the research

This thesis is composed of eight chapters, including the introduction chapter. In chapter two, a theoretical background about cloud computing and its features based on industry-standard publications with high numbers of citations is given. However, the theoretical background is based on grey literature in some parts, as scientific papers about some subjects can be sparse.

The definition of the maintainability of cloud platforms is discussed in Chapter Three. As there is no general definition for it, a definition used in this thesis is provided in the chapter, as well as the reasoning for the definition.

In the fourth chapter, the two cloud platforms to be used in this thesis, Amazon Web Services and Microsoft Azure, and their functionalities are researched based on the requirements of the software and the client. The researched functionalities include the cloud providers' support for container orchestration and managed databases.

The analysis of the needs of the software product that the solution is developed for in the thesis is done in chapter five. The software product being analysed is a monolithic software utilising Microsoft's ASP.NET framework and written on C#. The product is hosted on virtual machines in Microsoft Azure and on on-premises servers, utilising

PostgreSQL, and MongoDB databases. The software is currently in the process of being decoupled into multiple microservices.

In chapter six, the selected cloud providers are compared to each other based on various aspects, such as the services provided and the costs of those services. An example design of a possible hosting solution, including detailed costs, is also created for all the selected cloud platforms based on the results obtained from the analysis of the software product done in Chapter Four.

In Chapter Seven, the designs created in Chapter Five are evaluated based both on the costs of the designs and also on the maintainability of the complete system. The system's maintainability is determined by many factors, such as the ease of deploying new software product instances to the cloud.

Finally, possible ideas for further developments in the future are given in chapter eight. In chapter nine, conclusions are made based on the evaluation of the designs and a recommendation is made on what would be the best cloud provider for hosting the system based on the two main factors: costs and maintainability.

2. CLOUD COMPUTING

Cloud computing refers to the provision of computer system resources, such as processing power and data storage (cloud storage), without requiring direct user management. It provides immediate, on-demand, access to a shared pool of customizable computing resources as needed (Mell and Grance, 2011).

2.1 Definition of cloud computing

The National Institute of Standards and Technology (NIST) provides the most widely used definition of cloud computing. The definition lists five essential characteristics of a cloud model, multiple service models, and different deployment models.

2.1.1 Essential characteristics of the cloud model

The NIST defines five different essential characteristics of the cloud model. Those are on-demand self-service, broad network connectivity, resource pooling, rapid elasticity, and measured service (Mell and Grance, 2011).

On-demand self-service is the first of the five essential characteristics defined. It means that a consumer can independently and automatically allocate computer resources, such as server time and network storage, without the need for direct interaction with each service provider. (Mell and Grance, 2011)

Broad network connectivity is one of the fundamental aspects of cloud computing. It means that cloud services are accessible through a network and on a variety of devices, including portable devices, such as desktop computers, laptops, and cell phones. (Mell and Grance, 2011)

Resource pooling means that the cloud provider's computing resources are pooled to serve multiple different consumers. Physical and virtual resources are dynamically assigned and reassigned to consumers based on demand. The consumer generally doesn't have control or knowledge of the exact location of the resources being used, but, in most cases, they can specify a location at a larger scale, such as a data centre, region, or country. The resources that are pooled are, for example, processing, memory, storage, and network bandwidth. (Mell and Grance, 2011)

Rapid elasticity is the term used to describe the fourth essential characteristic of the cloud model. It means the capability to rapidly adjust to changes in demand by flexibly

allocating and releasing compute capabilities manually or automatically. (Mell and Grance, 2011)

Elasticity in cloud computing describes the system's ability to adopt workload changes by provisioning and de-provisioning resources automatically (Herbst, Kounev and Reussner, 2013). With some of the largest cloud providers, such as Azure, which had over four million physical servers in 2021, the capabilities available for customers to provision are essentially limitless (Donnelly, 2021).

Measured service is the last of the five essential characteristics. It means that the cloud systems control and optimise resource use automatically by using a metering capability, such as pay-per-use, at a level of abstraction suitable for the service. This allows payment to be based on actual consumption. Both consumers and service providers gain advantages from a transparent experience that is created by closely observing, managing, and documenting the utilization of resources. (Mell and Grance, 2011)

2.1.2 Service models of cloud computing

The NIST defines three different service models of cloud computing providers. The service models are Software as a Service, Platform as a Service, and Infrastructure as a Service. (Mell and Grance, 2011)

Software as a Service, or SaaS, refers to the provision of a cloud-based infrastructure by a provider, allowing consumers to utilize their software. The applications are usable via thin clients, such as web browsers or a program interface, and the users will never need installation on any device. With SaaS, the consumer doesn't have the capabilities to manage the underlying cloud infrastructure. (Mell and Grance, 2011; Kulkarni, Gambhir and Palwe, 2012; Chandrasekaran and Ananth, 2016)

Platform as a Service, or PaaS, enables users to deploy applications that are either custom-made or purchased, using programming languages, tools, and services provided by the cloud provider. As with SaaS, the customer does not manage or control the cloud infrastructure running the service but has the ability to modify, for example, configurations. (Mell and Grance, 2011; Butler, 2013; Chandrasekaran and Ananth, 2016) One popular example of PaaS is AWS Elastic Beanstalk, which makes it easy to deploy applications by automatically orchestrating various AWS services (Watts and Raza, 2019; Amazon Web Services, Inc., no date n).

Infrastructure as a Service, or IaaS, gives the consumer the most capabilities of the three service models. With IaaS, the consumer can provision processing, storage, networking, and other computing resources where they are then able to run the software, such as

operating systems or applications. Even though the consumer can provision the resources, they still do not manage or control the underlying cloud infrastructure. (Mell and Grance, 2011; Chandrasekaran and Ananth, 2016) Popular examples of IaaS services include Amazon EC2, Amazon S3, and Amazon VPC, which provide computing, storage, and virtual network for the consumer (Chandrasekaran and Ananth, 2016; Amazon Web Services, Inc., no date p).

The most popular cloud computing service model in 2022 was SaaS, with a market spending of 185 billion U.S. dollars. IaaS was the second most popular, with a market spending of 143 billion U.S. dollars, while PaaS was the smallest by a large margin with 84 billion U.S. dollars. Although SaaS is currently the most popular cloud computing model, IaaS has been gaining popularity faster since 2019, doubling its market spending since then, while SaaS market spending has risen 50%. (Open International, 2019; Lionel Sujay Vailshery, 2023)

2.1.3 Deployment models of cloud computing

Four different deployment models are defined in (Mell and Grance, 2011): private cloud, community cloud, public cloud, and hybrid cloud. The best deployment model for an organisation will depend on several factors, including the organisation's needs, the sensitivity of the data being stored, and the available budget.

A private cloud refers to a cloud infrastructure that has been made available for a single organisation's exclusive use. The infrastructure can be located either on-site or off-site and can be owned, managed, and operated by the organization itself, a third party, or a combination of both. (Mell and Grance, 2011; Khan, Freitag and Navarro, 2016)

In a community cloud, the cloud infrastructure is set aside for the sole use of a group of customers from organisations with shared concerns. The facility's location may be either on-site or off-site, and it can be owned, managed, and operated by one or multiple community organizations, a third party, or a combination thereof. (Mell and Grance, 2011; Khan, Freitag and Navarro, 2016)

The public cloud refers to cloud infrastructure that is accessible to the general public. It can be owned, managed, and operated by a business, academic, governmental organization, or a combination of these entities. Public cloud infrastructure is always located on the premises of the cloud provider. (Mell and Grance, 2011; Khan, Freitag and Navarro, 2016)

A hybrid cloud refers to a cloud infrastructure that consists of multiple distinct cloud infrastructures (private, community, or public). These infrastructures are connected using

standardized or proprietary technology, allowing for the transfer of data and applications between them while maintaining their unique entities. (Mell and Grance, 2011)

Each of these deployment models has advantages and disadvantages. A private cloud, for instance, might offer more security and control over sensitive data for an organisation, but it might also need more resources and maintenance than other deployment models. On the other hand, a public cloud might be more affordable and scalable but might not offer the same level of control over data security. (Microsoft, no date i)

Organisations must keep up with the most recent developments to select the best possible deployment model for their needs as cloud computing develops and new deployment models may appear.

2.2 Virtualisation

Virtualisation is one of the most critical technologies that make modern cloud computing possible, as it allows one physical resource to be converted into multiple virtual resources, which, in turn, can be used by numerous consumers. Some common types of virtualised resources are servers, also known as virtual machines, storage and networks, but most often, virtualisation is only associated with virtual machines. (Erl, Mahmood and Puttini, 2013; Sweeney, 2016)

Using a process called server consolidation, which means running multiple virtual machines on one physical server, hardware utilisation can be significantly increased, which allows for better resource optimisation. (Erl, Mahmood and Puttini, 2013)

Virtualisation can generally be run as either operating system-based or hardware-based. The main difference between the two ways is that the virtual machine management layer runs on top of the host operating system instead of hardware in operating system-based virtualisation. Hardware-based virtualisation is often more efficient due to having one less layer of abstraction, but on the other hand, operating system-based virtualisation provides significantly more flexibility. (Erl, Mahmood and Puttini, 2013)

2.2.1 Virtual machines

Virtual machines or virtual servers are virtual computer systems running on top of physical servers. They use their own operating system, which is independent of the operating system that they are running on top of. The guest operating system and the application software working on the virtual server are unaware of the virtualisation process, therefore the virtualised resources are deployed and used as though they are operating on a separate physical server. (Erl, Mahmood and Puttini, 2013)

The two primary types of virtualisation techniques used with virtual machines are full virtualisation and paravirtualisation. In full virtualisation, the hypervisor emulates the entire hardware environment, thus providing a complete and isolated virtual system for the guest operating system. Unlike full virtualisation, paravirtualisation requires modification of the guest operating system to enable communication with the hypervisor through an API. This approach reduces emulation overhead and results in better performance compared to full virtualisation. However, it necessitates changes in the guest operating system kernel, which may not be feasible for proprietary operating systems. (Sweeney, 2016)

2.2.2 Containerization

Containerization is one kind of operating system-based virtualisation used to run applications without deploying a virtual server for each solution, as the solutions are instead deployed within containers. Containers enable multiple cloud services that are isolated from each other to run on a single physical server while still accessing and sharing the same operating system kernel of the host they are running on. This means the containers don't contain their own operating system, making them use substantially fewer resources and start up faster than traditional virtual machines. (Erl, Mahmood and Puttini, 2013)

The application and all the required dependencies and libraries are packaged into container images. As these images contain everything the application requires, the application can be deployed by just using these without needing additional installations. The container images make the containers highly portable, allowing them to be deployed in any location that supports the container platform. As all of the libraries are only stored inside the containers, there will not be conflicts with, for example, different library versions between containers running on the same host. (Erl, Mahmood and Puttini, 2013; Docker Inc., no date b; IBM Cloud, no date a)

Virtualisation is the process of creating virtual versions of computer hardware platforms, storage devices, and network resources. It uses hypervisors to abstract physical servers into multiple virtual servers, each with its own guest operating system. Containers, on the other hand, are an abstraction at the application or service layer, bundling code and dependencies together. Unlike virtual servers, they don't require a guest operating system and use less storage space. Figure 2.1 depicts the difference between virtual machines and containers. (Erl, Mahmood and Puttini, 2013)

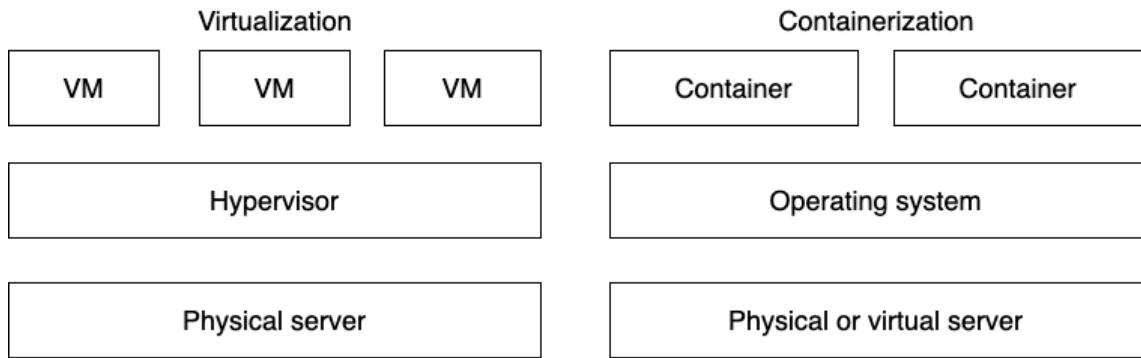


Figure 2.1 A comparison of server virtualisation and containerisation adapted from Erl, Mahmood and Puttini (2013.)

As mentioned earlier, portability is one of the container's most significant benefits. Other benefits include significantly reducing the CPU, memory, and storage usage footprint in comparison to virtual servers, which means that more efficient resource utilisation is achieved. Containers allow performance gains to be achieved by supporting multiple containers on the same infrastructure as a single virtual server. Containers also support a more agile process, as virtual servers cannot be created or deployed as quickly as containers. Tracking the version of a code and its dependencies is also made possible by containers. (Erl, Mahmood and Puttini, 2013; IBM Cloud, no date a)

Docker is currently the most popular container solution, and it was the most loved and wanted development tool in 2022 (C. Pahl *et al.*, 2019; Stack Overflow, 2022). Docker also has features that allow container management and container image building and management. Docker also provides Docker Hub, a container registry that enables the distribution of container images (Docker Inc., no date a).

As the complexity of container deployments has risen, the need for a better way to organise the containers has also risen. Container orchestration is the answer to that problem and will be studied next.

2.2.3 Container orchestration

After containers gained traction with the rise of the popularity of Docker, a new problem emerged: how to manage the containers across multiple machines. The solution to this problem was container orchestration. Container orchestration solutions automate many parts of container deployment, management, and monitoring, while having features that allow for defined communication and cooperation between multiple containers. (A. Khan, 2017; Newman, 2021, p. 259; IBM Cloud, no date b)

Multiple orchestration systems have been developed over time, but Kubernetes, which Google developed initially, has become the most popular. It has gained popularity as it

has a more comprehensive and sophisticated functionality in several areas compared to other orchestration solutions. (IBM Cloud, no date b) Kubernetes provides all of the common features of a container orchestration solution mentioned before. AWS and Azure both provide their own managed Kubernetes service (Amazon Web Services, Inc., no date j; Microsoft, no date g).

There are also many alternative container orchestration systems to Kubernetes; popular ones include Docker Swarm and Apache Mesos (IBM Cloud, no date b). Docker Swarm is a more lightweight software for orchestrating containers compared to Kubernetes. It is considerably easier to install and get started with. However, this has the effect of its more limited functionality, which makes it a good choice for small and simple applications, while Kubernetes is better for high-demand applications. (Rosen, 2022; Powell, 2023)

Regarding features, Apache Mesos with the Marathon framework is very similar to Kubernetes. The approach Mesos takes with container orchestration is quite different from how Kubernetes works, as Mesos is an operating system for data centres while Kubernetes focuses just on container orchestration. As the features are quite similar, Kubernetes is often a more popular choice because it is easier to set up in case only containerised workloads are used. (Chandrakant, 2022; Sharif, 2022)

Kubernetes works by running servers called *nodes* and control software called *control plane* inside a server cluster called *Kubernetes cluster*. The nodes can be either physical machines or virtual machines. Each node runs the essential components that it needs, such as the container runtime, kubelet, and kube-proxy. The container runtime is responsible for running the containers, and it can be, for example, containerd or Docker Engine. The kubelet is an agent that ensures the containers are running in a pod. Kube-proxy is a network proxy that runs on each node. (Newman, 2021, pp. 259–261; The Kubernetes Authors, 2023c) Figure 2.2 shows a simple overview of Kubernetes topology that contains a cluster, nodes, pods, and containers.

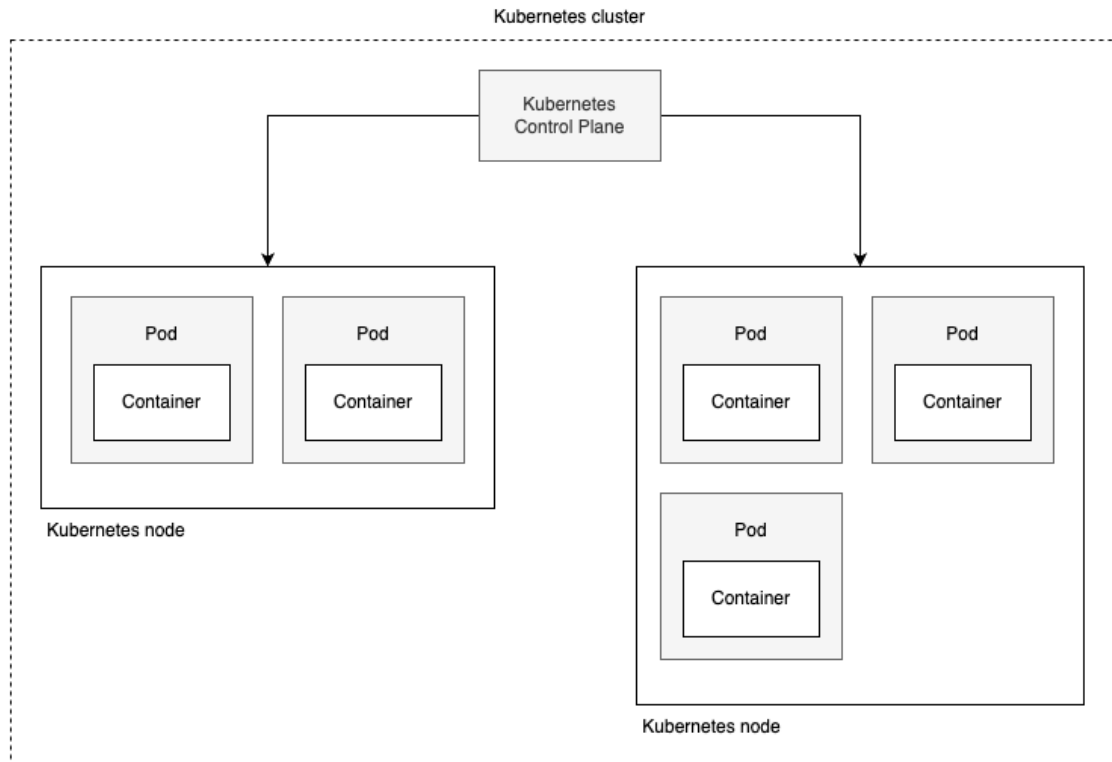


Figure 2.2 A simple overview of Kubernetes topology adapted from Newman (2021, p. 260)

A Kubernetes pod is the smallest possible deployable computing unit that can be created and managed in Kubernetes. Usually, a single occurrence of an application is enclosed within a pod. A pod is a group of one or more containers that are deployed together and share storage and network resources. Kubernetes assigns a pod to execute on a node according to its resource demands and scheduling policies. (Newman, 2021, pp. 259–261; The Kubernetes Authors, 2023d)

Kubernetes service provides a way to expose a network application running as one or more pods in a Kubernetes cluster. A pod can be considered ephemeral as it might shut down for many reasons, a pod should not be regarded as reliable and durable, but a Kubernetes service is designed to be long-lived. Thus, the service is there to route calls to and from existing pods. It can handle the pods being shut down or new ones launched. It should be noted that Kubernetes services are not deployed; instead, the pods are deployed and mapped to a service. (Newman, 2021, pp. 259–261; The Kubernetes Authors, 2023e)

Kubernetes deployment is a way to apply changes to pods. It makes doing things like rolling upgrades and rollbacks and scaling the number of nodes possible. The deployment controller does all of these changes at a controlled rate to prevent possible

issues from bringing the whole system down. (Newman, 2021, pp. 261–262; The Kubernetes Authors, 2023b)

As managing multiple services has become relatively easy with good container orchestration solutions, such as Kubernetes, it is often used to host small independent services called microservices. These microservices will be studied in the next section.

2.3 Microservices

Microservices are small independent services or applications that encapsulate functionality and make it accessible to other services over a network. Each microservice should be independently deployed, scaled, and tested, and it should only have a single responsibility. Generally, a more complex system is then constructed with multiple microservices that implement the entire system functionality. Microservice should hide as much information as possible inside and expose only the minimum amount of information required outside. This makes it possible to change the internal implementation of the microservice easily. (Thönes, 2015; Newman, 2021, pp. 3–5)

Independent deployability, which was mentioned earlier, is one of the critical concepts of microservices. It is necessary for the microservices to be loosely coupled so that the services can truly be independent. To be loosely coupled, microservices should control their own state. This means that there should not be any database sharing across the different services, but rather, if one service requires data from another, it should ask for it from the service. Another key concept of microservices is the idea of tightly modelling the services around business domains. This makes changes to business functionalities as efficient as possible, as there is no need to coordinate the work across multiple services and teams. It is important also to make sure that microservices don't become too large to handle or that there aren't too many of them to handle. However, there aren't any specifications of how large a microservice should be as long as the person or team maintaining it can fully comprehend it. (Newman, 2021, pp. 6–10; Harris, no date)

Microservices have multiple advantages, such as allowing the use of different technologies in different services, which means that the right tools can be selected for each job. A simple example of this is depicted in Figure 2.3. Microservices allow for easy scaling of only the required services that constrain performance, compared to monoliths, which require the entire system to be scaled. This helps with handling traffic spikes and controlling costs. Microservices also promote agility with small teams deploying frequently. (Newman, 2021, pp. 22–26; Harris, no date)

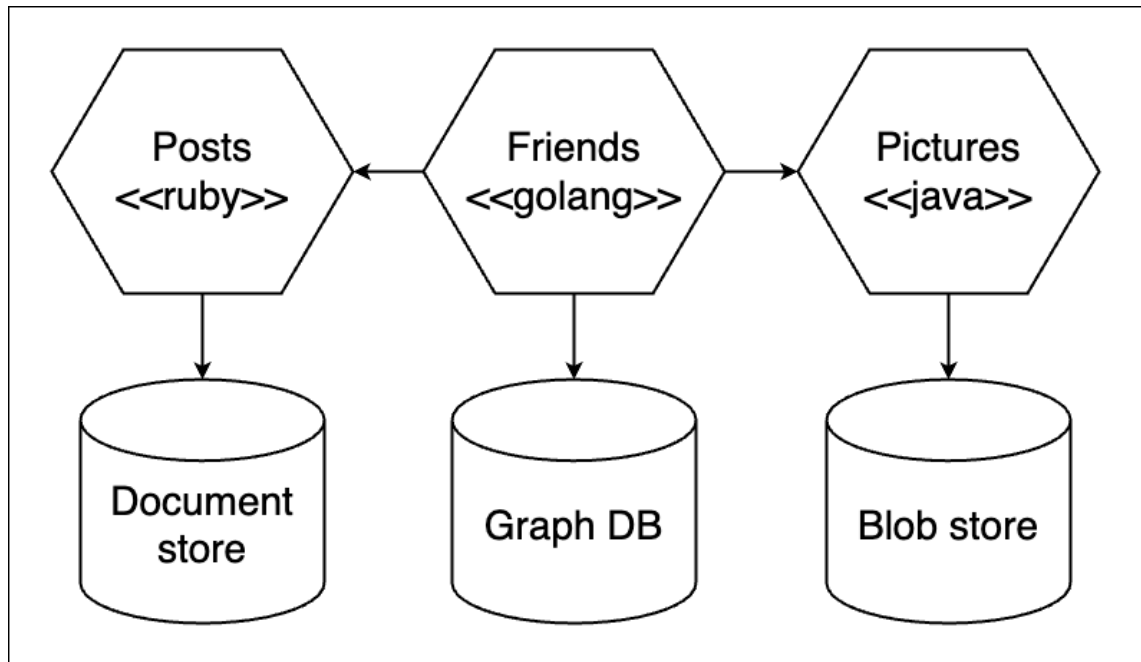


Figure 2.3 *Microservices allow for easier use of multiple technologies adapted from Newman (2021, p. 23)*

Microservices also have some disadvantages when compared to monoliths. For example, when there are multiple microservices, running them locally on one computer will become impossible, which will lower developer experience. The number of technologies can also become a large burden as numerous new ones have sprung up to enable microservice adoption, and selecting between these can be a hard task. Short-term cost increase is also one disadvantage of microservices, as there will be more processes to run, meaning more computers are needed. There are also other disadvantages related to reporting, monitoring, latencies, and data consistency, but those are not discussed here. (Newman, 2021, pp. 26–31; Harris, no date)

Microservices are frequently considered as an alternative to a monolithic architecture, which is an architecture that requires all system functionalities to be deployed together. Monoliths are generally divided into three categories: single-process, modular, and distributed monoliths. Single-process monolith is the simplest, as all the code is in a single module, whereas modular monolith contains multiple separate modules inside the same process. On the other hand, a distributed monolith contains multiple separate services that must be deployed together for some reason. Even though many people don't view monolith as a choice, there are numerous situations where it can be considered a better solution than microservices as it provides more straightforward deployment and simplifies code reuse. (Newman, 2021, pp. 14–26)

Microservices might not work in all situations, such as when working with entirely new products or startups, especially when the domain model has not yet stabilised. Another

situation where microservices are not optimal is when the software is meant to be deployed and managed by the customer, as this would cause unnecessary work for them. On the other hand, using microservices can seriously increase productivity when multiple developers need to be able to work on different services continuously when the service boundaries have already been clearly defined. Another good fit for microservices architecture is SaaS applications, as they benefit from the ability to do rolling updates on independent services. (Newman, 2021, pp. 31–33)

2.4 Serverless computing

In the last few years, serverless computing has become a popular choice for implementing different services, and all of the largest cloud providers are heavily invested in it. Even though serverless is nowadays almost synonymous with Function as a Service or FaaS, it also includes other kinds of services such as Backend as a Service or BaaS. (Roberts, 2018; Newman, 2021, pp. 249–256)

Function as a Service is a cloud computing service type that allows code execution in response to many different kinds of events without requiring a complex infrastructure usually associated with microservices. FaaS allows automatic scaling based on demand, which makes it excellent for handling traffic spikes. However, a cold start from zero running instances usually takes multiple seconds. It also costs money only when the functions are run, so in many cases, FaaS can be cheaper than running container-based microservices. (Roberts, 2018; Newman, 2021, pp. 249–256; IBM Cloud, no date c)

The most popular FaaS service in 2020 was AWS Lambda, with an 80% market share, and the second most popular, with a 10% market share, was Azure Functions (Eismann *et al.*, 2020; Amazon Web Services, Inc., no date m; Microsoft, no date e). The popularity of AWS Lambda can generally be explained by the fact that it was introduced two years before any other competing products were available, and AWS has the largest market share in general cloud computing (Eismann *et al.*, 2020). However, in 2021 and 2022, Azure and Google Cloud started to catch up to AWS (Datadog, Inc., 2021, 2022).

Backend as a Service is a type of cloud computing service where all backend-related aspects are outsourced to a BaaS provider so that the developers only have to maintain the frontend. This means that the BaaS provider provides, for example, database management, cloud storage, user authentication, notifications, and hosting. These services can be provided by either one BaaS provider or the application can utilise multiple different providers. (Roberts, 2018; Okta, 2023; Cloudflare, Inc., no date)

FaaS can be used to create cost-efficient microservices. There are many ways to implement these services, as there can be one function per microservice or one function per aggregate. Function per microservice would usually be the first step of moving to FaaS, and it only requires wrapping the current service into a provider-supported framework. A more efficient microservice could be achieved by separating every service endpoint into a separate function. In this way, there is no need to run the whole microservice for every request, which makes the requests more efficient. (Newman, 2021, pp. 249–256) An example of such FaaS is shown in Figure 2.4.

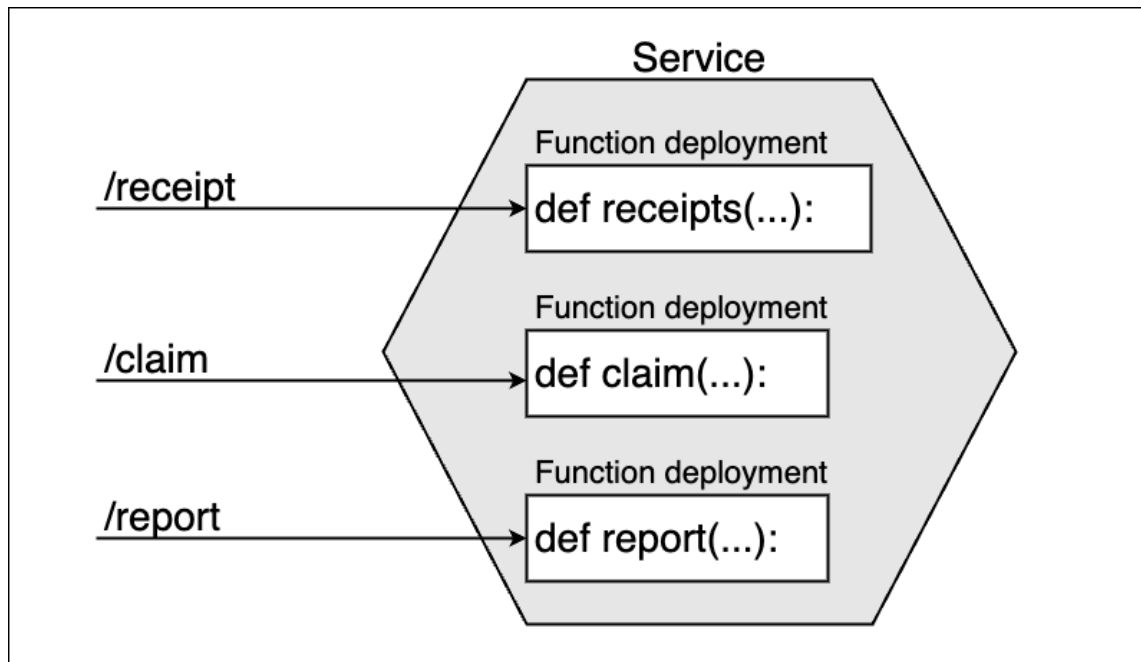


Figure 2.4 An example of a FaaS containing one function for each endpoint as adapted from Newman (2021, p. 254)

Serverless computing matches the characteristics of cloud computing, such as measured usage and rapid elasticity. With FaaS, you only pay for the computing power and execution time your code uses because the pricing is usage-based. Additionally, because the infrastructure automatically scales up and down in response to demand, serverless computing enables rapid elasticity and guarantees that you have enough processing power to handle traffic spikes.

2.5 Related research

There are limited academic papers written on subjects related to cloud provider comparison, and most of those concentrate only on the cloud providers' performance instead of the costs and maintainability. Most of the studies also seem to be mainly interested in the use of FaaS solutions. There are also some blog posts written about

the subject, which will be discussed here. The aim was to find as many recent papers as possible about the comparison as cloud technologies are constantly changing; however, all the relevant articles found were from 2017-2018.

Villamizar *et al.* (2017) discussed and compared the costs associated with running a web application in AWS using a monolith, microservices, and AWS Lambda. The study found that using a microservices architecture combined with AWS Lambda could reduce infrastructure costs by up to 77.08%, while moving from monolith to microservices only reduced costs by a little. The paper also noted that a monolithic approach may be more practical for software with hundreds or thousands of users. The paper is quite helpful for this thesis as it shows that we should not expect substantial cost savings by only moving to microservices.

Salah *et al.* (2017) compared the performance of a web application running in AWS EC2 virtual machines to containers running on AWS ECS. Surprisingly, the study found that the instances running on EC2 had a performance gain of 125% over the ones running on ECS. The study also found out that this is due to the fact that the AWS runs containers on top of EC2 virtual machines instead of running them on bare metal. The findings of this paper are helpful for this thesis as they show that AWS ECS should be avoided if possible.

Lloyd *et al.* (2018) studied the different factors that influence the performance of serverless microservices. The paper first identified the four states of serverless for microservice hosting: provider cold, virtual machine cold, container cold, and warm. The study then found that after 40 minutes of inactivity, the service performance degraded by about 15 times. The paper's findings are more useful for further development when moving to serverless microservices could be studied.

Maayan (2022) compared the costs of AWS and Azure on a general level. First, the article goes through the different products offered by the free tiers of the respective cloud providers. Next, the differences between pricing models, which differ a little, are discussed. Lastly, the compute and storage pricing are compared on a very general level, with the verdict being that for compute, AWS is cheaper, and for storage, Azure is. The article could be considered generally helpful guidance for this thesis, but as it only showed slight differences in pricing, the results themselves won't be used.

Taylor (2023) discussed the differences between AWS and Azure and compared the features of the clouds. The article explored various service areas provided by the cloud providers, such as infrastructure, storage, and pricing, to help understand the key differences between the platforms. The article noted that there is no overall best platform

as the choice between the two depends entirely on individual needs and requirements, as both platforms offer unique advantages. The article provides good grounds for this thesis as it recognises that the two largest cloud platforms are very similar feature-wise.

3. MAINTAINABILITY OF CLOUD PLATFORMS

3.1 Overview

As cloud computing has gained popularity in recent years by offering various advantages such as scalability, reliability, and flexibility, it has been noticed that maintaining resources running in the cloud requires careful attention to several aspects, such as performance and security. However, there hasn't really been any discussion about what makes a cloud platform more maintainable than another one.

Most of the large cloud platforms in use today are relatively similar in their offerings and the way they work, which makes comparing the maintainability of each one quite a difficult task. However, there are small differences between the platforms, and by comparing these differences, some differences in the platforms' maintainability can be noticed.

As said before, the maintainability of different cloud platforms is a relatively vague and subjective matter that hasn't really been researched before. As there aren't any definitions that could be considered standards in the subject, the definition used in this thesis will be given in the next subchapter of this chapter.

The maintainability of the different microservice deployment designs that are described later in Chapter 6 is measured based on multiple metrics, such as their cost, complexity, and size. According to Hasan *et al.* (2023), these metrics can be used to measure cloud architecture maintainability, along with others that would be unsuitable for this project.

3.2 Definition in this thesis

As mentioned before, there is not one clear definition for the maintainability of cloud platforms. As such, a definition used in this thesis will be discussed and defined in this chapter. The definition in this chapter can't, however, be considered a universal definition of the things affecting the maintainability of cloud platforms.

Maintainability can be deemed essential for the client in this case as it ensures cost reduction, agility, dependability, security, performance optimisation, and future-proofing. Cost savings, adaptability to changing business needs, high availability, a solid security posture, improved performance, and the ability to keep up with emerging technologies are essential for the client to succeed as a medium-sized software company.

Software maintainability generally refers to how easily a system can be updated, fixed, or modified to correct faults, improve performance, and meet changing needs or requirements ('IEEE Standard Glossary of Software Engineering Terminology', 1990). When software is maintainable, it is quick and easy to fix bugs, add new features, improve usability, increase performance and bring new developers on board the project (Crouch, no date). The quality of the system's design, the modularity of its components, the quality and thoroughness of its documentation, and the availability of effective testing and debugging tools are all elements that can influence software maintainability.

The concept of maintainability is important in all software-related work as software systems frequently change throughout their lifespans, and these changes must be made quickly and effectively to ensure that the system continues to function properly and satisfy user needs.

The maintainability of cloud platforms can consist of subjective and objective values in this context, so they need to be clearly separated. Subjective values that affect the perceived maintainability of cloud platforms could be, for example, prior experience working with the platform and perceived ease of use, which is also influenced by prior experience with the platform. The ease of use also affects the speed at which maintainers can make changes and fixes.

The quality of the system design would be one way to compare the maintainability of cloud platforms as it relates to the definition of software maintainability. However, in this thesis, it would mean comparing the design of the system running on the platform instead of the platform itself, so the main focus will be on comparing the platforms.

Some values affecting the maintainability of the platforms are also objective. The things affecting maintainability can be the scalability, reliability, and availability of the cloud platform, which also represent the usability of the platform. Scalability can be described as the cloud platform's capacity to manage growing workloads without compromising availability or performance. Reliability can be seen as the ability of a workload to carry out its intended function accurately and consistently at the anticipated time. On the other hand, availability describes the percentage of time a workload can perform its agreed function. (Amazon Web Services, Inc., no date e) Also, the amount of material, documentation, and discussions found online about the products offered by the cloud provider can affect the ability to maintain software on the cloud platform.

Lastly, there are values affecting maintainability that are, in some ways, both subjective and objective. These could be, for example, support for Infrastructure as Code -tools and the availability of different external design tools. Support for Infrastructure as Code

increases the maintainability of systems running on cloud platforms, but the support provided by different platforms might differ, and all the tools might not be supported at all by some platforms (Sharma, 2022). The external design tools can make a platform more maintainable as the modifications can first be designed in the design tool.

4. CLOUD PLATFORMS

Nowadays, a large number of cloud providers provide a great number of different services. Still, the largest three, Amazon Web Services (AWS), Microsoft Azure, and Google Cloud, dominate the market with a combined market share of 66% in the third quarter of 2022. In the third quarter of 2022, AWS was the largest of the three by a large margin, having a market share of 34%. Microsoft Azure followed AWS with a 21% market share, while Google Cloud had an 11% market share. (Synergy Research Group, 2022)

The two largest cloud providers, AWS and Azure, were selected to be the providers for this thesis based on multiple factors such as market share, popularity and wishes of the client. All of these cloud platforms provide all of the features required by the software.

4.1 Amazon Web Services

Amazon Web Services began offering cloud computing services to businesses in 2006, according to Mathew (2021). In the third quarter of 2022, it was the largest of all cloud providers, with a 34% market share, according to Synergy Research Group (2022). In 2021, AWS provided cloud computing services to hundreds of thousands of businesses in 190 countries. As of 2023, AWS provided over 200 fully featured services from data centres in multiple countries (Amazon Web Services, Inc., no date o).

The AWS services that are relevant to this project are described next. The first one is AWS Elastic Compute Cloud or EC2, which provides scalable on-demand computing resources to customers. It allows customers to create and manage virtual machines for various workloads. EC2 provides flexibility and cost efficiency that is not available with traditional on-premises servers. (Amazon Web Services, Inc., no date k)

AWS Fargate is a serverless computing engine that operates on a pay-as-you-go model. It eliminates the need for customers to manage servers, allowing them to focus on application development. It allows for easy deployment of containerised applications, with the customer only needing to define the required memory and compute resources to run the application. (Amazon Web Services, Inc., no date l)

AWS Elastic Container Service or ECS is a container orchestration service that makes running, scaling, and maintaining Docker containers simple. It is designed to be high-performance and fault tolerant, allowing the customers to easily manage containerised applications at a scale without managing the underlying infrastructure. ECS supports many different types of AWS computing platforms, such as EC2, AWS Fargate, or AWS

ECS Anywhere, allowing on-premises server use. (Amazon Web Services, Inc., no date f)

AWS Elastic Load Balancing or ELB is a load balancer service that provides three different kinds of load balancers: application, gateway, and network. It automatically distributes incoming traffic across multiple targets in one or more availability zones, making high availability and automatic scaling simple. The different kinds of load balancers work at different layers; for example, application load balancers work at OSI layer 7. (Amazon Web Services, Inc., no date i)

AWS Elastic Kubernetes Service, or EKS for short, is another fully managed service that simplifies container application deployment, management, and scaling using Kubernetes. EKS also fully integrates with multiple AWS computing platforms in a similar way to ECS. (Amazon Web Services, Inc., no date j)

Amazon RDS for Postgres is a fully managed relational database service with full PostgreSQL support, making it easy to set up, operate, and scale PostgreSQL databases in the cloud. It provides features for automatic backups, patching, and monitoring. RDS Read Replicas make it easy to elastically scale out beyond the capabilities and constraints of a single database instance. (Amazon Web Services, Inc., no date h)

Amazon DocumentDB is a fully managed NoSQL database service compatible with MongoDB and designed for high-performance, document-centric applications. It provides high availability, durability, and scalability, enabling developers to easily store, query, and index data in JSON format. DocumentDB best suits applications requiring flexible data models and rapid development cycles. (Amazon Web Services, Inc., no date b)

4.2 Microsoft Azure

Microsoft Azure was announced to the public in 2008 and has been operating ever since (Macmanus, 2008). In the third quarter of 2022, it was the second largest of all cloud providers, behind AWS, with a 21% market share (Synergy Research Group, 2022). As of 2023, the Azure cloud platform provided over 200 products and cloud services to customers worldwide from over 200 data centres (Zhang, 2022; Microsoft, no date l). According to Microsoft (no date h), 95% of Fortune 500 companies are using Azure.

Next, the Azure products relevant to this project are briefly described. The first one is the Azure Virtual Machines, which are scalable on-demand computing resources that enable

users to create and manage virtual machines that run in the Microsoft Azure cloud. They offer the same flexibility and cost efficiency as the AWS EC2. (Microsoft, no date k)

Azure Container Apps is a managed container service that simplifies deploying, scaling, and managing containerised applications in a similar way to AWS ECS but with the difference that Container Apps are always serverless compared to the multiple options on ECS. As with ECS, Container Apps abstract infrastructure management, allowing users to focus on application development. (Microsoft, no date b)

Azure Application Gateway is a load balancer for web traffic on OSI layer 7, which means it is similar to the Application Load Balancer of AWS ELB. As with AWS ELB, Application Gateway also makes distributing incoming traffic to multiple targets simple, thus simplifying high availability and automatic scaling. (Microsoft, no date a)

Azure Kubernetes Service, or AKS, is a fully managed container orchestration service that simplifies deploying, managing, and scaling containerised applications using Kubernetes. It offers seamless integration with other Azure services and provides a reliable, secure, high-performing environment for running container workloads across various sectors. Feature-wise, AKS competes directly with AWS EKS, with EKS having a few more features for some niche cases. (Amazon Web Services, Inc., no date j; Microsoft, no date g)

Azure Database for PostgreSQL is a managed relational database service that automates backups, patching, and monitoring. It enables users to easily set up, operate, and scale PostgreSQL databases in the Azure cloud. It competes directly with Amazon RDS for Postgres, with the same features. (Microsoft, no date d)

Azure Cosmos DB for MongoDB is a fully managed, MongoDB-compatible NoSQL database service designed for high-performance, globally distributed applications. It offers turnkey global distribution, elastic scaling, and low-latency access to data. Cosmos DB is suitable for applications that require flexible data models, rapid development cycles, and seamless integration with the MongoDB ecosystem. It directly competes with Amazon DocumentDB, with both having the same features. (Microsoft, no date c)

5. THE ANALYSED SOFTWARE PRODUCT

In this chapter, the basics of the analysed software is described, and its current hosting solution is reviewed so that it can be more easily compared in the next chapter. The steps required for hosting a new software instance are also reviewed.

5.1 Overview of the analysed software

The software analysed in this thesis is a customer information system, or CIS, used extensively within a niche industry. The software is quite large, consisting of multiple separate modules. Even though the software consists of multiple modules, it is currently being deployed as a monolith instead of microservice-based. This kind of software can be described as being a modular monolith (Newman, 2021, p. 16).

The customers, which in this case are companies in a niche industry, can keep track of their customers, customer contracts, and service usage, send invoices, keep track of their assets, and generate reports for their needs using the software.

The modules of the software can generally be described into three different groups. The first group contains modules such as reporting that communicate with external systems, and the second group contains modules that generally only communicate internally and keep most of the system's master data. The third and last group contains modules that are used for billing and for communicating data related to that externally.

5.2 The software architecture of the analysed software

The software is currently built using modular monolith -architecture. Modular monolith architecture is a variation of monolith architecture, where the single monolith process consists of multiple modules. The separate modules that the monolith consists of are, in this case, the modules mentioned previously, which are still currently quite tightly coupled. An example of the modular monolith -architecture is shown in Figure 5.1.

The software is almost completely written using Microsoft's C# language, which is used with .NET Framework version 4.8. The .NET Framework that is used is the latest and, at the same time, the last version available as the .NET Framework has been replaced by .NET, which software of this size can't easily be migrated into due to multiple API changes. One separate service is running on Java, but there are plans to possibly convert the service to use C# and the .NET ecosystem in the future.

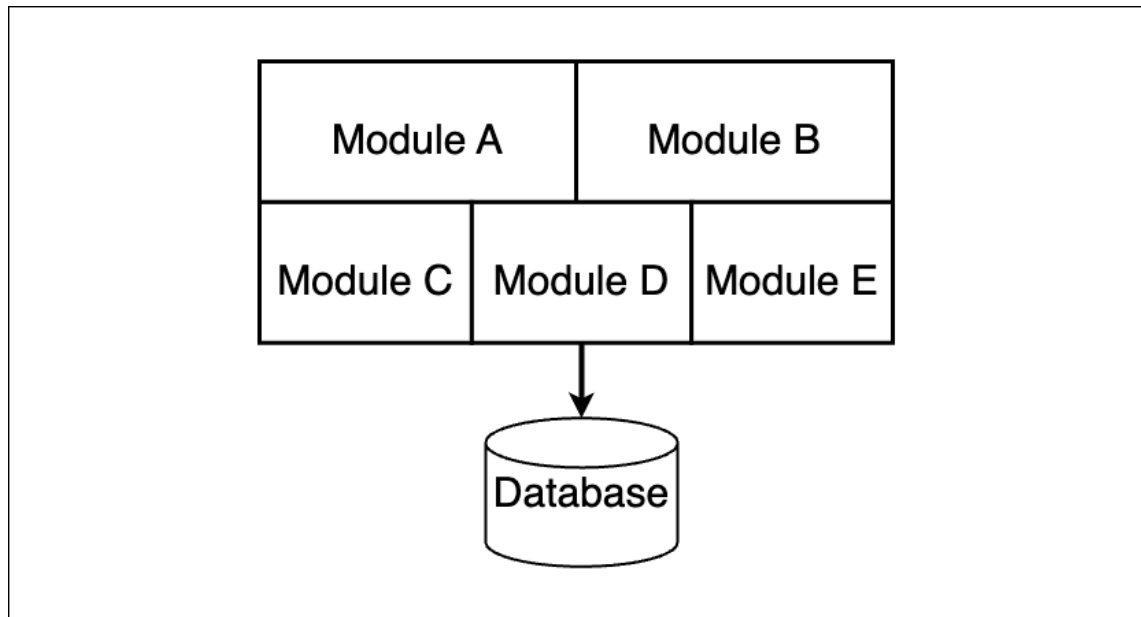


Figure 5.1 Example of modular monolith architecture adapted from Newman (2021, p. 16)

The software has multiple layers, including the UI, Façade, Base, and data. The software uses a façade pattern, which means that there is an object that serves as a front-facing interface that masks more complex underlying code. A basic level overview of the software's structure is shown in Figure 5.2, which was adapted and generalised from a proprietary diagram shown in the client's documentation.

The structure shown follows a simple pattern where all services have access to some common elements available to all services in the layer in addition to the service-specific ones. The topmost layer contains UI layer components acting as a presentation layer with access to common libraries available to all layers. The second layer is the façade-layer with façade interfaces and services with access to common infrastructure for the façade-layer. The third layer is the base layer, which contains the business logic for each of the separate services that the software entails. Finally, the last layer is the data layer or persistence layer, which handles connection and data movement between the software and the database. The data layer contains different projects for modelling and accessing the data.

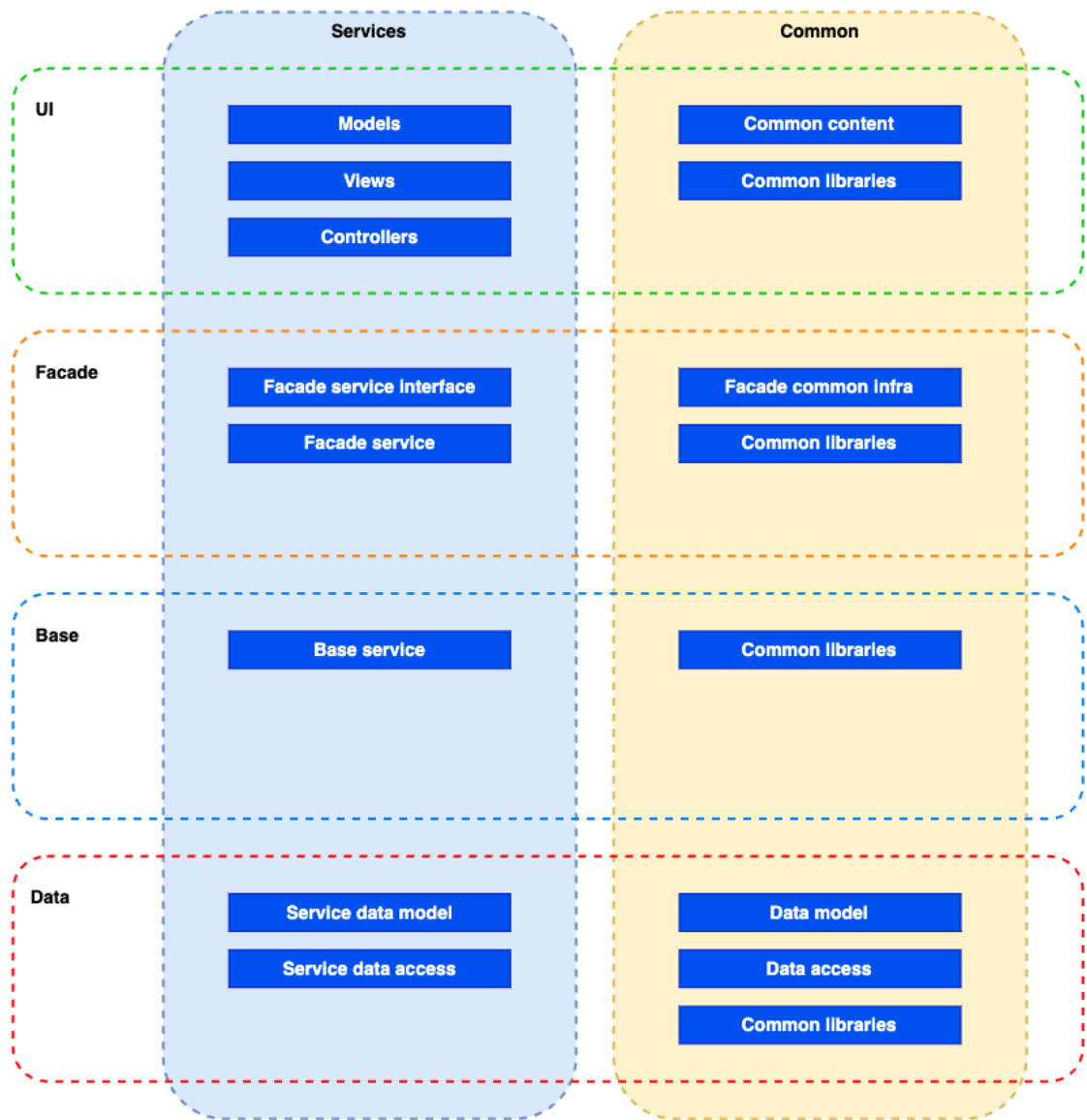


Figure 5.2 A basic overview of the structure of the software

Overall, the current modular monolith architecture and the façade pattern are both good designs on which to base the future decoupling for microservices. The modular monolith -architecture especially helps with this, as each module already has some border for responsibility.

5.3 The current hosting solution of the analysed software

The analysed software currently uses four different virtual machines that each host one part of the software solution. A simplified diagram of the current hosting solution is shown in Figure 5.3.

The main application server is deployed as a monolith to a single virtual machine in Microsoft's Azure. The software's application server has been determined to need

around 24 GB of RAM and at least four virtual CPUs to work properly and prevent unnecessary system slowness for the customer when running a single instance on the VM with the lightest real-world load. However, in some situations, multiple instances of the same software will be deployed on the same virtual machine. In these cases, the required RAM and vCPUs must be determined case by case.

The analysed software also requires a separate server, called an intra-server, to function correctly. The intra-server is responsible for providing the frontend and customer-facing API of the application. The intra-server is often run on a separate virtual machine, but it can also be run on the same one as the application server in some cases. Depending on the customer's configuration, the intra-server must have at least 20 GB of RAM available to function properly.

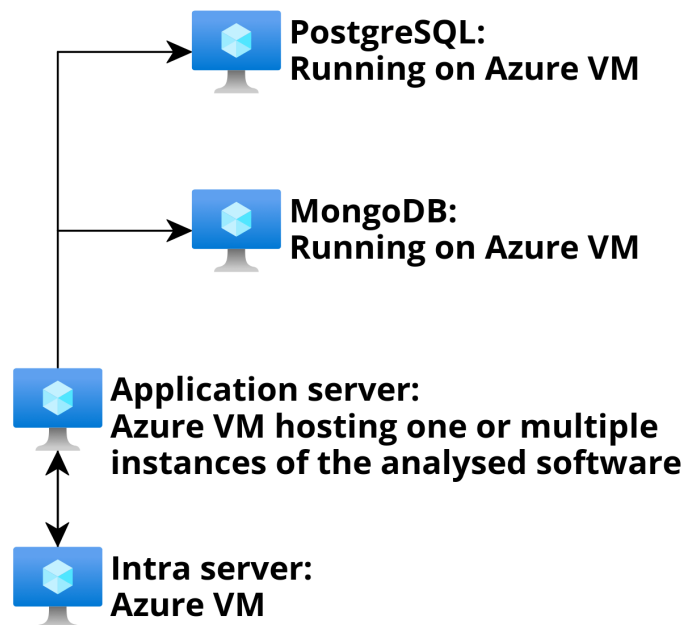


Figure 5.3 Simplified diagram of the current hosting solution of the analysed software

The application server uses PostgreSQL and is currently run on its own virtual machine. Postgres is the software's primary database, and most deployed databases contain hundreds of gigabytes of data. The virtual machine's requirements are determined separately for every customer, which means there isn't any standard size for these virtual machines.

The application server also uses MongoDB and runs separately on its own virtual machine. Some of the modules use MongoDB, but it is not the primary database of the software as it is mainly used for saving incoming and outgoing data in JSON format. As with the Postgres server, the virtual machine's requirements are determined separately for every customer's needs.

5.4 Steps required to host a new instance of the analysed software

The current process for hosting a new instance is mostly manual, although the actual deployment of the software is primarily automatic after most of the definitions have been created. The usually year-long process before the production environment starts with a discussion with the customer about their needs and the number of objects they have to store. The number of objects currently dictates the RAM the software needs, so the information must be known from the start.

After the initial requirements are known, the required VMs for the quality assurance environment are requested from the data centre operator and the operating systems are installed on those by the client. The software's first deployment can only be made after the operating systems are installed. Finally, when the software is deployed, data migration can be started, and the integrations can be done. When everything is working, the same process is applied to setting up the production environment.

While most of the process can't be automated as part of this project, others, like the virtual machine and environment setup excluding the data migration, could be. The automation for setting up the VM and database would, for example, be done as part of microservice deployment with Kubernetes or other container orchestration. This would be relatively easy to do as the basic environment without data is always the same.

5.5 Future development of the analysed software

As monolithic software systems have grown more complex and difficult to maintain, scale, and update, decoupling them has become increasingly lucrative for companies. Monolithic architectures are characterised by tight coupling, complexity, problems with scalability, and deployment challenges. These problems can be dealt with, and the software can be made more adaptable and flexible by decoupling the monolith into smaller, more modular components.

The process of decoupling the monolithic architecture of the analysed software into microservices has been going on for around a year. The software is quite large and previously written in a tightly coupled way, so the process has proven to be quite slow and laborious. The decoupling process is targeted to be ready at the end of 2023

After the decoupling has been completed, there will possibly be around nine different microservices that will be developed, deployed, and hosted separately from each other. The different microservices will, however, initially still be using a single PostgreSQL and MongoDB databases instead of each service having its own database. However, the

databases might be transferred from being hosted in a VM to a managed database service provided by AWS or Azure. The end goal for decoupling is a situation where each service has its own independent database and every other service it depends on.

As mentioned previously, the decoupled microservices will all initially still use one instance of both PostgreSQL and MongoDB databases. This kind of arrangement makes it possible for some new problems relating to concurrency to emerge that haven't been previously encountered. To address these possible issues, a proof-of-concept project has been started inside the company to determine if it is currently possible for the software to use the databases concurrently and what changes it could require from the system to support it.

For the proof-on-concept, the system design is mostly similar to the normal system, with the difference being that it uses Azure Virtual Machine Scale Set for the application server. A simplified example of this deployment is shown in Figure 5.4. This kind of setup makes it possible to test whether the current system can handle multiple instances of the application handling the data simultaneously. As the support for this is needed from the system before separately deployed microservices can be used, no microservices can actually be used before the outcomes of the project are known.

The Azure Virtual Machine Scale Set or VMSS makes it possible to create and manage a group of load-balanced VMs with the possibility of auto-scaling the number of VM instances based on demand or a defined schedule. The service can be used to easily provide high availability for software that needs to be deployed straight to a VM instead of Kubernetes or similar services. As mentioned earlier, in the proof-of-concept project, the VMSS runs multiple application server instances for testing. (Microsoft Learn contributors, 2023; Microsoft, no date j)

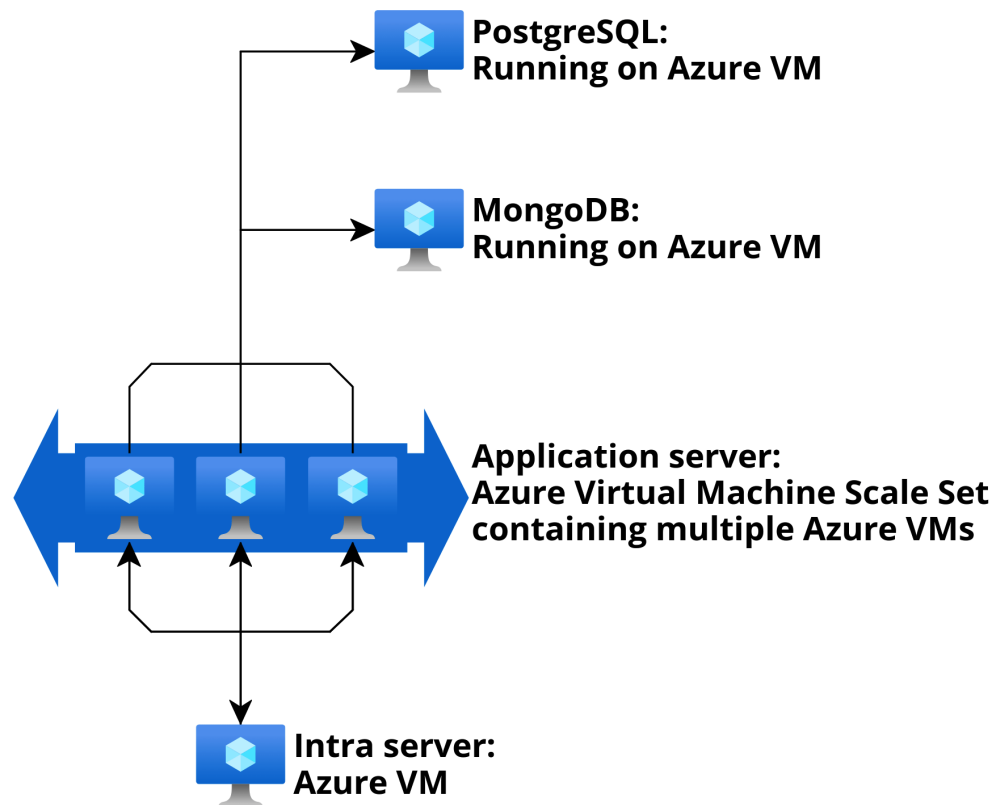


Figure 5.4 Example of the deployment of the concurrency test

The designs presented in the next chapter will be based on the planned state of the application after the decoupling project has been successfully completed, which means that the designs are based on a theoretical future version of the software. It also means that it doesn't consider any possible issues arising, for example, from concurrent use that is found as part of projects.

6. DESIGN

6.1 Determining the hosting requirements of the software

Each software has different requirements for it to work correctly, as does the software used in this project. One requirement is that the software currently only works on machines running Windows-based operating systems. In the future, after the work on decoupling the modules has been completed, this requirement will not be relevant as each module can easily be updated to use a newer version of .NET with support for Linux. In this project, the aim is not to consider the possibly different RAM and CPU needs of different modules as it would add unnecessary complexity in this development phase where there are no explicit requirements for each module.

For databases, the current version requires one PostgreSQL and one MongoDB server. However, it is expected that each microservice will have its own databases, so the services will be designed with that in mind, giving each microservice one PostgreSQL and one MongoDB server. When the decoupling has been finished, there will be more knowledge on which modules require which databases. To reduce cost, there is a possibility that all of the services will share one common PostgreSQL and Mongo server.

The initial design will include nine separate microservices, but as each of the microservices is identical, it will be easy to add more services as needed. Nine microservices were chosen as the software had nine high-level modules inside the three groups mentioned earlier. By making the microservices duplicatable, it will be much simpler to make changes and future additions that are prone to happen in an ongoing project such as this. For the initial design, the cloud providers' native PostgreSQL and MongoDB services will be used, but in the future, other solutions such as Mongo Atlas or self-hosting on a VM will be possible. The approximate number of requests coming to each microservice is unknown at the time of the project, which means that the load balancer pricing can't be calculated and needs to be done later in the project.

6.2 Comparison of the cloud platforms based on the requirements

Overall, both of the products provided by the cloud platforms are pretty similar in their offerings related to both Kubernetes and other services that the Kubernetes service will utilise. Both providers have data centres in Sweden, with Azure currently building a centre in Finland (Microsoft, 2022, no date f; Amazon Web Services, Inc., no date g).

The pricing of the managed Kubernetes clusters has some differences between the different providers. AWS provides only a single tier with a price of 0.10 USD per hour. On the other hand, Azure has three different tiers to choose from: free, standard, and premium. The standard is priced at the same rate of 0.10 USD per hour, while the premium is much higher, at 0.60 USD per hour.

The actual services of this project will run on either AWS EC2 or Azure VMs. The pricing for these depends on the resource requirements, but for this calculation, each node should have at least two vCPUs and 4 GB of RAM. With these requirements, a virtual machine from AWS costs 0.0432 USD per hour or around 31.536 USD per month, and from Azure, an identical machine costs 0.0368 USD per hour or around 26.864 USD per month.

As using serverless containers would be an option in this project, they could possibly be used. They are based on the seconds the application is running with the use of memory and vCPU billed separately. However, as the prices of these services are much higher than their VM alternatives and the benefits they provide are mitigated by the use of Kubernetes, they will not be used for this project.

It should be noted that all the services compared in this project are ones provided by the selected cloud providers. As such, services like Mongo Atlas, which utilise the service providers but are not directly offered by them, are not considered even though they might be used in the final solution.

On top of these prices, there will be other costs, such as data transfer in and out of the data centre, about which the client doesn't currently have data. However, these costs should usually be relatively small for services that don't deal with tasks like data transfers. It should also be noted that both providers offer different savings plans with a commitment to use the agreed number of monthly hours for a year or three years, depending on the contract.

6.3 Designing example hosting solutions for the software

The hosting solution designs in this thesis are done using DataDog CloudCraft, a proprietary SaaS providing tools for designing cloud environments for AWS and Azure. The reason for deciding to use this tool instead of free alternatives is that it allows for automatic cost calculations based on the designed environments. However, only a few Azure components were implemented entirely as of August 2023, with some of the components for AWS also missing, which made it necessary to manually calculate the costs for both.

Each of the designs will contain the services required to run all the microservices, and it will also include the cloud providers' analytics and logging services, as it was assumed that those would be necessary when deploying the application. However, some simplifications were made as there still was uncertainty about the final decoupled product and as the design had to be compatible with both of the cloud providers. Due to the uncertainty, each microservice will contain identical components and use the same computing platform instead of the final product, which will be finetuned for each service. It is also worth noting that the solutions will not contain any external components used for communications between different microservices, such as message queues, even though they may ultimately be present in the final solutions. This is also due to uncertainty about which services would need such components, as most of the communications between the services will at least initially be done using regular HTTP APIs.

6.3.1 The designs on Amazon Web Services

Designs for Amazon Web Services were done before they were done for Azure, so they acted as the base for the Azure designs. All the designs shown below will utilise AWS EKS clusters, with AWS EC2 providing the computing for the microservices. Figure 6.1 shows one example of a possible simple deployment of a single microservice, which could be duplicated to create all nine initial services. The deployment consists of the EKS cluster and databases. Inside the EKS cluster, there is one EKS workload with two AWS EC2 instances running the microservice for redundancy and high availability. The microservice has two databases for its own use: AWS RDS PostgreSQL and AWS DocumentDB. Access from the outside by the users and other microservices is designed to happen via a Kubernetes ingress that, in the case of the EKS cluster, is an AWS Application Load Balancer or ALB. If it would be later required, this load balancer could be switched to an AWS Network Load Balancer.

The benefit of selecting this kind of fully independent design would be the ability to easily control and monitor each microservice separately. This kind of design would also be quite simple to maintain as a separate entity, which means that this might be the best option in terms of maintainability. Also, as each microservice has separate databases just for its use, they can easily be scaled to match the actual need of the microservice they serve. The downsides of a deployment like this are mostly related to higher costs, which could be significantly higher than in some other designs. Also, even though the single microservice is easy to maintain as a separate entity, it can become very laborious to keep up with every separate microservice.

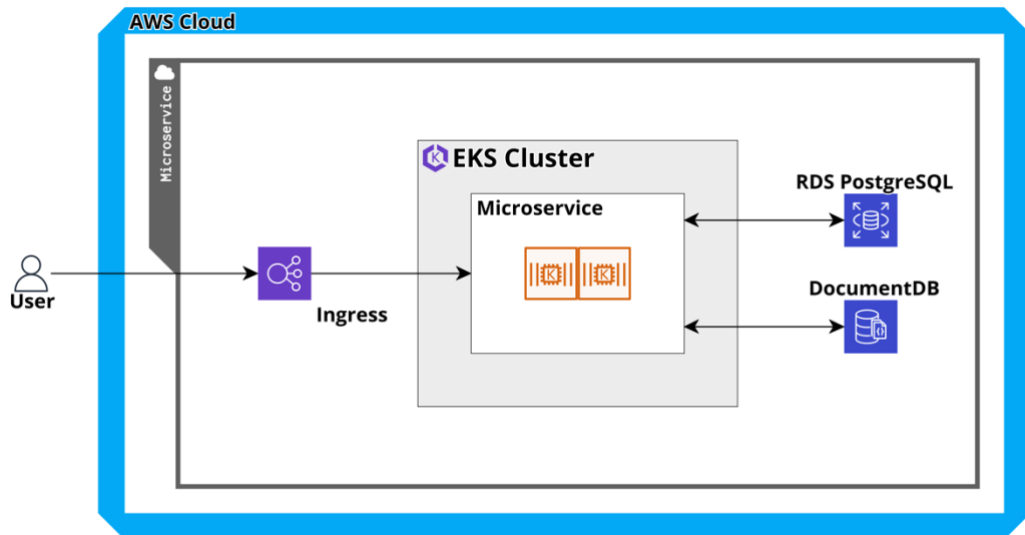


Figure 6.1 Example of a deployment of a single microservice on AWS

A more complex example where all nine microservices are running in the same AWS EKS cluster is shown in Figure 6.2. In this example, all the services share AWS RDS PostgreSQL and AWS DocumentDB databases. In this example, one load balancer acts as the Kubernetes ingress for the microservices. As each of the different microservices shares the databases, they need to have much higher performance compared to the designs where the databases are separate. In a design like this, the databases could become a bottleneck for the system. However, in a situation like that, the database service could be scaled onto a larger instance or in an extreme case, the services which use the databases the most could be given separate databases as the expectation was that the different services are not dependent on each other, meaning that each service still has its tables inside the database instead of sharing.

The main benefit of a system like the one in Figure 6.2 would be the cost savings from using the shared databases and a single EKS cluster. This is because using a single shared database would likely mean that the database would have a higher utilisation rate than separate ones, and the larger database instances usually have a better cost-to-resource ratio. The main downside of the design would also come from the fact that the databases are shared, which means that none of the microservices truly are independent of each other; if one database fails, all microservices would also fail. However, the chance of this happening is very slim, especially when using a high-availability database.

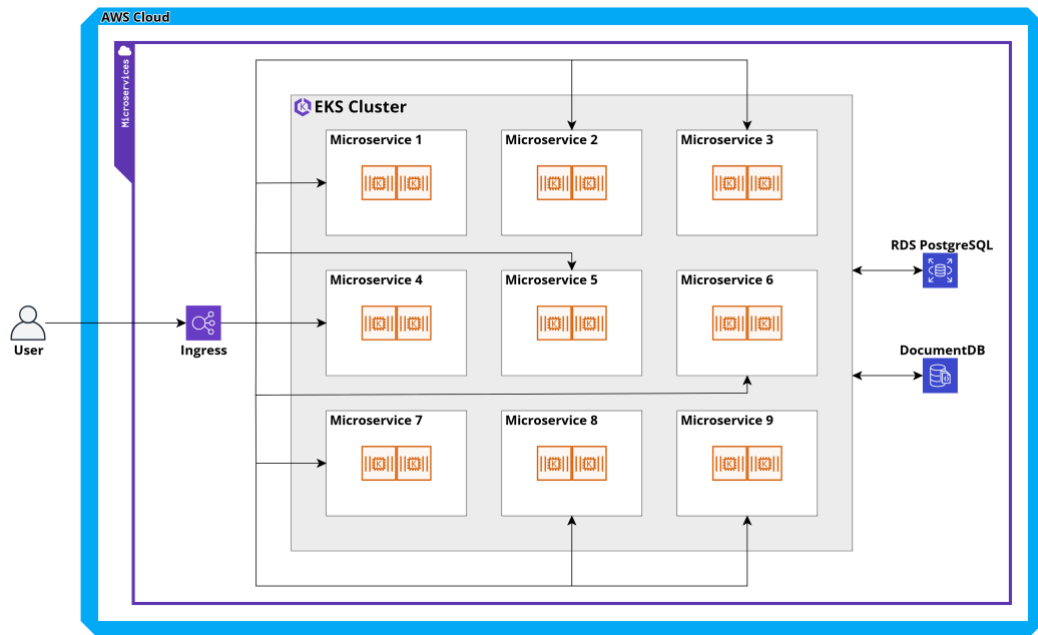


Figure 6.2 Example of multiple microservices running under the same EKS Cluster with shared databases

The design shown in Figure 6.3 is a hybrid model of the previous designs shown in Figure 6.1 and in Figure 6.2. In this design, all the microservices are in the same EKS cluster, similar to the previous design, but each has separate databases. The benefits of this design include separation between the different microservices but with additional costs which come from the multiple databases with possibly lower utilisation rates. This kind of design might also be more challenging to maintain because it has the most components in one system of all the described designs.

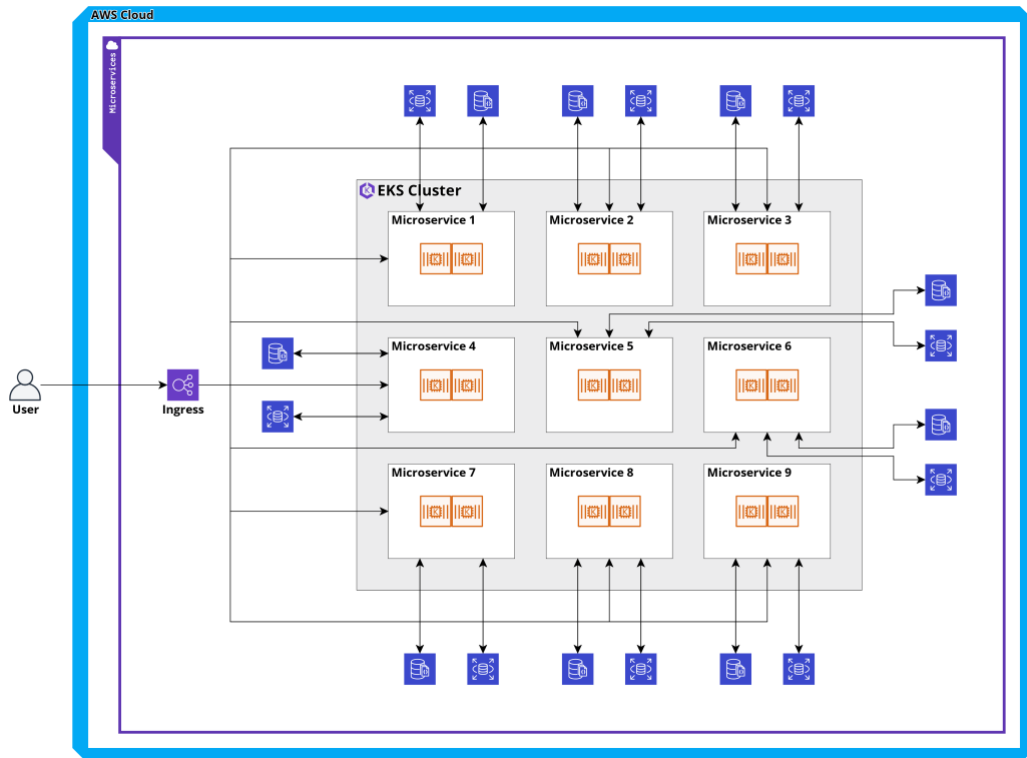


Figure 6.3 Example of multiple fully independent microservices running under the same EKS Cluster

The next chapter discusses the pricing for the components used in the designs shown in more detail. Preliminary costs for the whole design are also discussed, although it would only be possible to calculate an accurate price for such a design with the known details.

6.3.2 The products and pricing of Amazon Web Services

Calculating one accurate cost for all the services would realistically be impossible from the known data; therefore, it was decided that a better way would be to calculate different cost ranges for the services, such as computing, databases, and networking. These groups can then help estimate the overall costs of the designs when the requirements become more precise. When calculating monthly costs, it was assumed that the service would run 24/7 with the same load. All the prices discussed in this chapter were taken from the AWS Pricing Calculator (Amazon Web Services, Inc., no date d).

The base cost of an AWS Elastic Kubernetes Service cluster is 0.10 USD per hour, which rounds to around 73 USD per month, which only covers the cluster, and all used resources are billed separately. This means that, for example, the first option would cost at least 584 USD more per month for the EKS Cluster alone, which can sometimes be a significant percentage of the monthly costs. As running Kubernetes on providers like

AWS EKS and Azure AKS can quickly introduce cost blind spots, it is necessary to do these calculations again when all the numbers are known more reliably.

Starting with computing, as mentioned earlier, an instance with two vCPU and 4 GB of memory was planned as it has more than enough resources to run any of the microservices. In a case where each of the AWS EC2 instances would only run one pod, each pod would cost 31.536 USD per month, which in turn means 63.072 USD per month per microservice if the minimum of two instances are always running one microservice. The computing costs would account for 567.648 USD monthly with the planned nine microservices. However, if it were possible to use only 2 GB of memory for each VM, the price would come down to 15.768 USD per month, which would mean a monthly cost of 283.824 USD for nine microservices with two pods each.

The pricing discussed above would only be the case in the example shown in Figure 6.1 as when running on the same cluster, Kubernetes can officially support up to 110 pods per node or, in this case, per VM (The Kubernetes Authors, 2023a). For example, all of the microservices could be run on a VM with eight vCPU and 32 GB of memory, which costs 252.288 USD per month, and for redundancy reasons, at least two VMs should be used as nodes, which would double the price to 504.576 USD per month. If four vCPU and 16 GB of memory are enough, the monthly price could be lowered to 126.144 USD for one VM. Running two VMs instead of the 18 VMs mentioned previously would be more economical in most cases and could also increase maintainability tremendously.

The pricing of the AWS RDS for PostgreSQL depends on multiple different settings, such as instance type and the amount of stored data. Determining the best instance type would be hard with the current information, but for a single microservice, an instance with two vCPUs and 4 GB of memory should be enough. This kind of instance would cost 47.45 USD per month. For the shared database, an instance with eight vCPU and 32 GB of memory could be a good choice, with monthly costs of 381.06 USD. The cost for storage is 0.12 USD per month for each GB of data stored, with data backup costing only 0.09 USD. Additional costs are introduced by Amazon RDS Proxy, costing 12.41 USD per month per vCPU.

The Mongo-compatible DocumentDB also has multiple settings which affect the pricing. An instance similar to the lower-performance PostgreSQL database would cost 65.15 USD monthly. The most similar instance type to the larger PostgreSQL one, which DocumentDB supports, has four vCPUs and 32 GB of memory, costing 384.20 USD per month. The number of write and read operations is calculated and costs 0.22 USD per

million operations. In addition, the storage is priced at 0.119 USD per month for each GB of data stored, with data backup costing 0.023 USD.

For both types of databases, it is essential to consider high availability, which means doing a multi-AZ deployment in AWS. Multi-AZ means that the data in the primary database is continuously copied to a standby replica in another availability zone; in case the primary database fails, traffic is automatically routed to the standby replica, and no data is lost in the process. This kind of deployment will double the costs of the databases, but it can be considered an absolute necessity in a production environment.

The load balancer is an integral part of the designs, acting as an ingress between the cluster and the outer world. In AWS, the pricing for the Elastic Load Balancer consists of two different charges: a monthly charge of around 17.48 USD and a usage-based charge, determined by the amount of Load Balancer Capacity Units or LCUs consumed by the load balancer in an hour with an additional hourly cost of 0.0076 USD which translates to a monthly cost of 5.56 USD. LCUs are measured by the number of new connections per second, the number of active connections per minute, the number of bytes processed in gigabytes, and the number of rules processed. AWS monitors each of these metrics, and the one with the highest usage is the one determining the cost. One LCU consists of 25 new connections per second, 3000 active connections per minute, 1 GB of processed bytes for EC2 instance targets, and 1000 rule evaluations per second. The price of one LCU is 0.0076 USD per hour.

The last larger cost is logging, which is billed based on a multitude of parameters. These parameters are, for example, log-file collection, storage, and metric monitoring. Metrics can be considered the most expensive one, costing 0.30 USD per metric for one month, which can seem low, but the costs add up quite easily. According to an AWS CloudWatch example, one EKS cluster reports 24 metrics, every EC2 instance in the cluster reports eight metrics, a Kubernetes pod reports nine metrics and a service reports six metrics (Amazon Web Services, Inc., no date a). For the case where two large EC2 instances run the EKS nodes, this would mean 202 metrics plus the number of services in the final design. According to the aforementioned AWS CloudWatch example, Kubernetes will output, on average, 38 KB of logs per metric per hour to the CloudWatch logs. With the 202 metrics mentioned previously, this would mean 5.34 GB of monthly logs, priced at 0.54 USD per GB.

All of the services mentioned above are gathered in Table 1. Based on the previously described findings, the table displays the minimum and maximum calculated monthly pricing for each service. As most of the prices on AWS are billed hourly, the prices have

been converted to monthly prices by multiplying the hourly price by 730. It should be noted that the prices mentioned in this table are on-demand, and the prices can be drastically lower if signing up for 1-year- or 3-year contracts.

Table 1 Minimum and maximum price per month for each major service in AWS

Service	Minimum price per month (USD)	Maximum price per month (USD)
Kubernetes cluster	73.00 (1 cluster)	657.00 (9 clusters)
Computing	283.82 (18 VMs – 2 GB)	567.65 (18 VMs – 4 GB)
	252.29 (2 VMs – 16 GB)	504.58 (2 VMs – 32 GB)
Databases	861.40 (1 SQL)	1077.48 (9 SQL)
	463.26 (1 DocumentDB)	497.07 (9 DocumentDB)
Networking	17.48 + 5.56 (1 LCU)	
Logging/Metrics	63.51 (2 VMs)	107.73 (18 VMs)
Total	1736.50 / 1768.03	2956.18 / 3019.25

On top of these costs, there are some additional costs associated with these designs. For example, outward data transfer from AWS EC2 to the outside world is billed separately at a rate of 0.09 USD per gigabyte for the first ten terabytes. If additional IP addresses are needed for the EC2 instances, they are billed at 3.65 USD per month. Persistent storage for the EC2 instances will also be billed separately, if needed, for some microservices.

6.3.3 The designs on Microsoft Azure

The designs for Microsoft Azure were done after the designs for AWS, so the designs are mostly identical, with only the services changed to match the ones available on Azure instead of AWS. As with the AWS designs, all the Azure designs utilise Azure Kubernetes Service clusters with at least two Azure Virtual Machine -instances providing the computing for redundancy and high availability. Azure Database for PostgreSQL is used to provide the PostgreSQL database for the Azure designs, and Azure Cosmos DB is used to provide a Mongo-compatible NoSQL database. As with the AWS designs,

access from outside the microservice, both by users and other microservices, is designed to happen via a load balancer, in this case, an Azure Application Gateway. The three Azure designs shown next in Figure 6.4, Figure 6.5 and Figure 6.6 are similar to their AWS counterparts except for the changes in components described above.

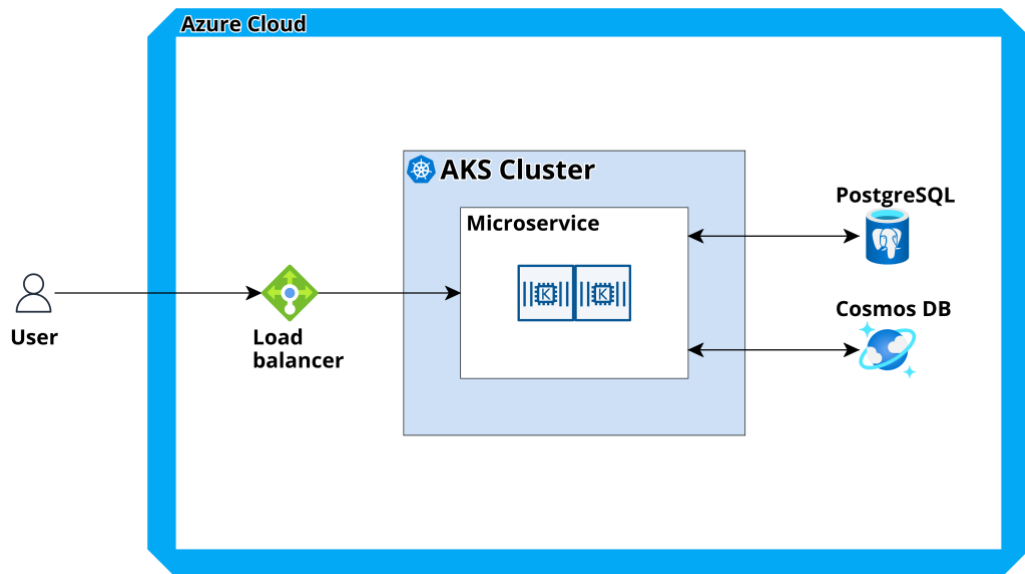


Figure 6.4 Example of a deployment of a single microservice on Azure

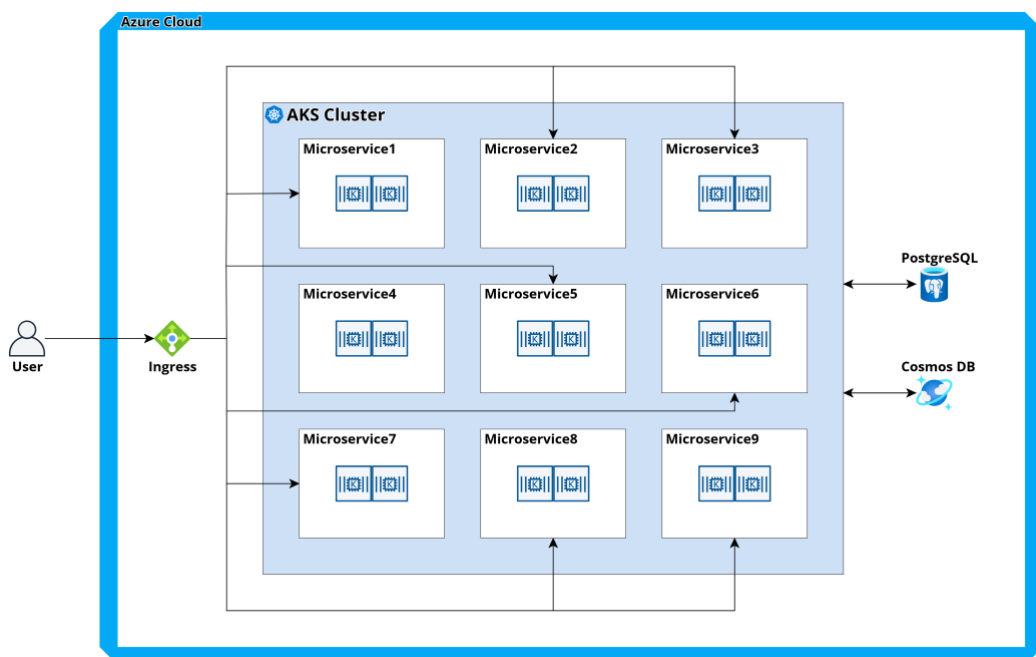


Figure 6.5 Example of multiple microservices running under the same AKS Cluster with shared databases

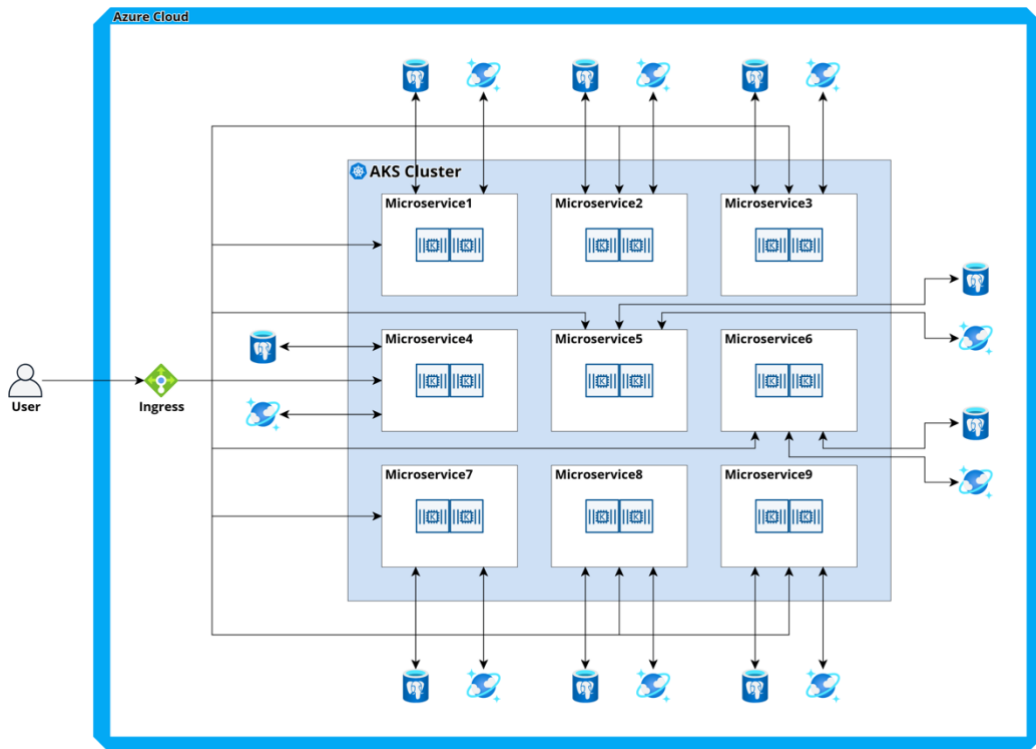


Figure 6.6 Example of multiple fully independent microservices running under the same AKS Cluster

As can be seen from the figures, both of the clouds offer basically the same basic products that the designs utilise. In the next chapter, the pricing of these components for Azure will be analysed so that the providers can be compared in Chapter 7. The comparison will help determine which provider offers a more cost-effective solution for these components.

6.3.4 The products and pricing of Microsoft Azure

The pricing of Microsoft Azure works similarly to Amazon Web Services and consists of multiple different components. As with the AWS, the major components that will be discussed are the AKS cluster, computing, databases, load balancing, and logging. The final costs will be presented as a table similar to the one in AWS pricing. As with the AWS pricing, all the prices for Azure were taken from the official Azure Pricing calculator (Microsoft, no date h). Even though the calculator offers the possibility to use euros, USD was used as the euro cost depends on the continually changing exchange rate.

Microsoft Azure offers the following three pricing tiers for their Azure Kubernetes Service: free, standard, and premium. The free tier is only meant for learning and experimenting, so it won't be considered in this case. The standard tier is advertised as suitable for all mission-critical workloads with a price of 0.10 USD per cluster per month, which means 73 USD per month. The standard tier is the one that matches the features provided by

AWS EKS. On top of these features, the premium tier includes Microsoft-backed long-term support for major and minor Kubernetes versions with a price of 0.60 USD per hour per cluster, which rounds up to 438 USD per month. This support can be a huge help if the development and DevOps teams can't, for some reason, keep up with the Kubernetes release cycle, where each minor version is supported for one year by the community. However, with this project, the standard tier would be enough. As with AWS, on top of the mentioned prices, all used resources are billed separately.

Virtual machines with the same kind of specifications as the ones in AWS were picked from Azure. The requirements were at least two vCPUs and 4 GB of memory for the virtual machine running only one pod. Eight vCPUs and 32 GB of memory were required for the two machines running all the microservices. The VMs for Azure were chosen from Azure's burstable selection as the VMs used on AWS designs were also burstable instances, meaning that they had the ability to burst to higher levels to support spikes. Pricing was also calculated for the smaller 16 GB instance to show the price difference. However, Azure does not seem to offer a VM with two vCPUs and 2 GB of memory in the same series of machines as the other examples, so that can't be compared.

The largest instance with eight vCPUs and 32 GB of memory was priced at 0.311 USD per hour, which means a monthly price of 227.03 USD. Meanwhile, the small instance with two vCPUs and 4 GB of memory is priced at 0.039 USD per hour or 28.40 USD per month. For comparison, the smaller instances with four vCPUs and 16 GB of memory would be 0.156 USD per hour or 113.88 USD per month. On top of these costs, possible disks are priced at 5.81 USD per month for a premium SSD with 32 GB of space.

For PostgreSQL, the most sensible option is Azure Database for PostgreSQL, which is available in two different deployment models: single server and flexible server. The single server is a simpler deployment to manage with a built-in high availability. Both of the models work differently from a usual database by separating computing and data. Because the single server provides much less control over the systems, a flexible server will be used in this project.

An instance with eight vCPUs and 32 GB for the flexible server costs 0.748 USD per hour, which means a monthly cost of 546.04 USD per month. Storage for the database is priced at 0.15 USD per month per GB, while backups are 0.124 USD per month. Notably, if a geo-redundant backup storage is wanted, the billed amount of backup data is double that of the used storage. The cheapest instance available has two vCPUs and 8 GB of memory with a monthly cost of 136.51 USD. Azure provides automatic high

availability with an automatic switch to standby in case of failure for the cost of the second instance.

Azure Cosmos DB, a scalable service with support for multiple different database APIs, such as MongoDB and Apache Cassandra, provides the MongoDB database for the project. Cosmos DB for MongoDB offers three kinds of database operations: standard, autoscale, provisioned throughput, and serverless. However, a fourth one, an instance, is available with Cosmos DB for MongoDB vCore. For all three basic database operations, the pricing is based on the number of request units or RUs; on the provisioned throughput operations, the price is calculated for RU per second, and for serverless request units, they are billed by a million requests. The request unit that is used in the pricing is a proprietary unit of measurement created by Azure, which means that it is quite hard to calculate a realistic cost even though Microsoft provides a simple request unit calculator. For the vCore version, the cost is determined by the chosen instance.

For the non-vCore version, the standard provisioned throughput is priced at 0.008 USD per 100 RU/s per hour, while the serverless operations are priced at 0.31 USD per million RUs. The vCore version has only a few instances to choose from. An instance with eight vCores and 32 GB of memory is priced at 0.979 USD per hour or 714.63 USD per month, while a slightly less performant instance with four vCores and 16 GB of memory is 0.489 USD per hour or 357.32 USD per month. The smallest possible instance is one with two vCores and 8 GB of memory, which is priced at 178.66 USD per month. In addition to the instance prices, 128 GB of storage is the minimum, with a monthly cost of 16.26 USD. On top of the costs incurred, storage is priced at 0.25 USD per month per GB; on the backup side, the first two are free, and after that, the backups are 0.14 USD per month per GB.

Taking high availability into consideration is important when discussing databases for production software. Azure offers easy-to-use high availability for both of the databases, with automatic fail-over in the case of the PostgreSQL database and completely automated high availability with Cosmos DB, which doesn't depend on any single VM instance.

Azure Application Gateway is a level seven load balancer that is used in this project as the Kubernetes ingress. The Application Gateway V2, the only version that can be used as the AKS ingress, has an hourly price of 0.26 USD, which translates to 189.8 USD per month. The pricing is similar to its AWS counterpart in that it has an hourly cost and a usage cost of 0.008 USD per capacity unit hour. A single capacity unit consists of 2500

persistent connections, a throughput of 2.22 Mbps, and one compute unit, which can handle 50 connections per second. As with AWS, Azure monitors each of these metrics, and a new capacity unit is used if any of these parameters exceeds the limit. In addition, outbound data is billed after the first 100 GB of data.

Logging and metrics are handled by Azure Monitor, which, like its AWS counterpart, is billed based on a multitude of parameters. A major difference between the providers is that all the metrics from the AKS cluster are collected for free at no additional cost. Log collection is priced at 0.645 USD per GB, and Azure estimates that the described designs create between 0.16 GB and 0.59 GB of logs, excluding stdout and stderr.

All of the services discussed above are gathered in Table 2. The table displays the minimum and maximum calculated monthly pricing for each service based on the previously described findings. As most of the prices on Azure are billed hourly, the prices have been converted to monthly prices by multiplying the hourly price by 730. As with AWS pricing, it should be noted that the prices mentioned in this table are on-demand, and the prices can be drastically lower if signing up for 1-year- or 3-year contracts.

Table 2 Minimum and maximum price per month for each major service in Azure

Service	Minimum price per month (USD)	Maximum price per month (USD)
Kubernetes cluster	73.00 (1 cluster)	657.00 (9 clusters)
Computing	511.15 (18 VMs – 4 GB)	511.15 (18 VMs – 4 GB)
	227.76 (2 VMs – 16 GB)	454.06 (2 VMs – 32 GB)
Databases	1092.08 (1 SQL)	2457.18 (9 SQL)
	1461.78 (1 Cosmos DB)	3508.47 (9 Cosmos DB)
Networking	189.80 + 5.84 (1 CU)	
Logging/Metrics	3.10 (2 VMs)	11.42 (18 VMs)
Total	3053.36 / 3336.75	7283.77 / 7340.86

These designs come with some additional expenses in addition to these costs. For example, after the first 100 gigabytes, the first ten terabytes of outgoing data transfer

from Azure to the outside world are charged at a separate rate of \$0.08 USD per gigabyte.

7. EVALUATION

The data gathered and displayed in the tables in Chapter 6 will be evaluated in this chapter, and the choices for the optimum services will be discussed along with the general cost level of the services for the designs. The designs' maintainability will also be evaluated and compared to the cost of the design.

7.1 Evaluating the platforms and designs based on costs

In this chapter, the focus is on deciding the best design and provider based on the costs of the services in the designs, while in the next chapters, the maintainability of the designs will be considered as well. The pricing for the services discussed in the chapter is visible in Table 1 and Table 2.

When it comes to Kubernetes clusters, it's clear that opting for a single cluster is the smarter choice, particularly when considering costs. This is because the clusters do not offer any discounts when multiple ones are acquired. Therefore, sticking with a single cluster is more cost-effective than investing in multiple ones. As can be seen from Table 3, the clusters are priced equivalently between both providers.

Considering the computing services, the solutions with two virtual machines come on top with lower costs across both cloud providers. Having only two virtual machines also lowers the work that must be done maintaining them, thus also saving money later on. With two virtual machines, all additional costs are also smaller, such as those caused by logs and VM storage space.

It can be seen that Azure seems to provide somewhat more affordable virtual machines between the providers. However, it should be noted that there isn't really any documentation available about how the virtual machines of Azure and AWS compare against each other in performance, as they are both just selling virtual CPU access instead of real CPU. This, on the other hand, means that even though Azure is cheaper when comparing the instance, AWS could very well have more performance per dollar. It should also be noted that both platforms provide much cheaper virtual machines with the same specifications but with ARM-based processors. However, getting the software discussed in this project to run on ARM natively would be nearly impossible.

The databases clearly create the largest difference in the pricing between the two cloud providers, and at the same time, they are the most expensive single service in the designs. Looking at the different designs, it becomes clear that, again, in terms of

database costs, the best design is the one with shared database instances. There doesn't seem to be any downside to this approach as a single AWS RDS or Azure Database for PostgreSQL instance can have multiple databases by default, which means that every microservice still gets a separate database for its own use (Microsoft Learn contributors, 2022; Amazon Web Services, Inc., no date c).

In the different designs, the total price for the databases depends a lot on the cloud provider chosen. This can be seen when comparing AWS, where the price for nine smaller databases only increased marginally, to Azure, where the price more than doubled. Overall, it seemed like Azure lacked in providing smaller instances for the databases and, for that matter, most of the services, which gave the impression that it is really an alternative to AWS only when the resource requirements for the project are quite large. Based on the discussion above, it is quite clear that when only taking the cloud providers' own products into account, AWS is a much cheaper option compared to Azure.

It should be noted that realistically, especially for the Mongo database, the best option would be to use an external database service like Mongo Atlas, which provides some cheaper options, at least when compared to the offering provided by Azure. Another realistic option would be to pay for a normal computing instance and set up the databases there, as that is something that the client in question has experience of doing. This could also provide quite considerable savings, especially for Azure, where the VMs generally seem to be cheaper than in AWS, as mentioned earlier.

When comparing the pricing of the OSI level seven, the application layer, and load balancers offered by the cloud providers, it becomes clear that Azure's offering is not again suitable for smaller projects as it is ten times more expensive than AWS' offering. This means that if the single microservice per Kubernetes cluster design were used, only the load balancers' fixed costs would amount to over 1700 USD per month for nine microservices. However, even when comparing the designs with a single load balancer, AWS is the clear winner in terms of costs.

The providers price the logging quite differently. Azure has taken an approach where the different metrics from AKS are free. While Azure has a higher price for log ingestion, the amount of logs outputted by this kind of program is not enough to make the cost very high. On the other hand, AWS charges a flat fare for each of the numerous metrics that the EKS and everything related to that output. While AWS has a lower log ingestion price, Azure is the better choice in this category when comparing the costs.

When comparing the pricing data collected in this thesis to the ones in Maayan (2022), it becomes clear that the prices change regularly between the platforms as they compete. In the Maayan (2022), it was found that AWS is generally cheaper for computing, while this study has had the opposite results as computing in Azure seems cheaper currently.

Based on costs, it seems that the designs shown in Figure 6.2 and Figure 6.5 would be the best choices because they have only one Kubernetes cluster with the possibility of using two virtual machines as Kubernetes nodes. In the designs, there is only one PostgreSQL and MongoDB database server or instance, which lowers the costs drastically compared to the solutions with nine separate database instances. Table 3 displays the optimum services, their price, and the total price for all services for both cloud providers based on these designs.

Table 3 Comparison between services provided by AWS and Azure

Service	Optimum services for AWS (USD / month)	Optimum services for Azure (USD / month)
Kubernetes cluster	73.00 (1 cluster)	73.00 (1 cluster)
Computing	504.58 (2 VMs – 32 GB)	454.06 (2 VMs – 32 GB)
Databases	861.40 (SQL)	1092.08 (SQL)
	463.26 (DocumentDB)	1461.78 (Cosmos DB)
Networking	17.48 + 5.56 (1 LCU)	189.80 + 5.84 (1 CU)
Logging/Metrics	63.51 (2 VMs)	3.10 (2 VMs)
Total	1988.79	3279.66

From the table above, it can be seen that overall, AWS seems to be much cheaper than Azure, but as was mentioned above, Azure's databases could be replaced by some external service such as Mongo Atlas, or they could be run on VM instance as self-hosted databases. However, based on these assumptions, it is clear that AWS would be a better choice based on costs.

7.2 Evaluating the platforms and designs based on maintainability

Evaluating the platforms and designs based on maintainability is not as clear as doing the evaluation based on costs, as some aspects of maintainability are subjective and thus vary from person to person. As mentioned in Chapter 3 some maintainability-related properties are more objective. These properties, such as scalability and reliability, won't differ much between the designs as they can all scale horizontally and vertically and are fault tolerant. The differences between the designs will become more about the cognitive workload when the services need to be handled.

The first design, which was displayed in Figure 6.1 and Figure 6.4, was easily the most expensive of the displayed designs to implement all nine services with, but in terms of maintainability, it is not quite as bad. This is due to the fact that it contains everything that the one microservice needs in its own cluster with no necessary dependencies on other services. This means that each deployment is a lot simpler than the designs with many more services. On the downside, this kind of design can become increasingly hard and complex to maintain if one person or team needs to singlehandedly manage multiple deployments.

The second and third designs share one cluster for all the microservices but differ in the way databases are used. In the second design shown in Figure 6.2 and Figure 6.2, the database instances are shared, which helps lower the monthly costs. In the third one, displayed in Figure 6.3 and Figure 6.6, each microservice also has its own database instances. Comparing these two in terms of maintainability, design number two clearly comes on top, mainly due to the fact it contains fewer components. Fewer components mean less complexity and size, which means a more maintainable architecture, according to Hasan *et al.* (2023).

Based on the points discussed above, it can be said that the best design is the one with shared databases and one Kubernetes cluster for situations that have a single person or team maintaining all the deployments. Next, the maintainability of the actual cloud platforms will be compared based on the assumption that the best design is used.

The differences become smaller when comparing the two cloud platforms based on the design chosen previously. Overall, it seems that subjective metrics are the ones that differentiate the two cloud platforms from each other. Things such as prior experience with a platform can thus become a large factor in choosing the cloud platform to use. Another factor can be perceived ease of use, which could, in this case, favour Azure as

it has fewer services overall to choose from, and its UI could be considered less clunky than that of AWS.

One critical factor when choosing a cloud platform based on its maintainability is the support for Infrastructure as Code or IaC tools that are familiar to the people maintaining the deployments on the platform. These tools are essential as they can make a massive difference in the efficiency of managing software on a platform. Terraform, one of the most popular and well-known IaC tools, has extensive support for both platforms. However, both platforms also have their own proprietary tools, which means that to change a platform, a new tool would have to be learned, which would decrease the platform's maintainability.

As AWS has held the title of the largest cloud platform for multiple years in a row, many tools that help with cloud software maintainability have been developed specifically for it. These tools, such as CloudCraft, which is used in this thesis to create the designs, increase the maintainability of a platform by making it quick and easy to try out different designs and settings. Even though support for Azure in these tools has been increasing in the last few years, it still doesn't seem to quite match the support that AWS gets.

Objective maintainability metrics like scalability, reliability, and availability form one part of the maintainability of a cloud platform and its services. In terms of the project discussed in this thesis, it seems like both cloud platforms offer quite the same value in these metrics. Both cloud providers seem to have more than enough capabilities to scale their services to a nearly unlimited scale. Regarding reliability and availability, both providers have financially backed Service Level Agreements, or SLAs, included in all of the products discussed in this thesis. The SLA uptime levels differ between each service for each cloud provider. Some of the uptime levels are shown below in Table 4.

Table 4 Uptime SLAs for different AWS and Azure services

Service	AWS SLA uptime (%)	Azure SLA uptime (%)
Kubernetes cluster	99.95 ¹	99.95 ²
Computing	99.5 / 99.99 ³	99.5 / 99.99 ²
PostgreSQL	99.95 ⁴	99.99 ²
MongoDB	99.9 ⁵	99.99 ²
Load balancer	99.9 / 99.99 ⁶	99.95 ²

Regarding the SLAs offered by the platforms, as seen from Table 4, Azure tends to provide slightly higher uptime percentages than AWS, but the differences are quite small. The largest difference is in the uptime SLA of the AWS DocumentDB and Azure Cosmos DB for Mongo, with a difference of 0.09%, which translates to roughly 39 minutes less downtime per month. Overall, the findings from the table indicate that Azure is more reliable and available, at least in terms of what they are willing to promise customers.

To conclude, Table 5 shows a comparison between the discussed platforms on the three different areas of cloud maintainability that were discussed in this chapter. The table shows that in scalability, there aren't any significant differences, but Azure has a significantly better uptime SLA.

¹ Amazon Web Services, Inc. (2022) Amazon EKS Service Level Agreement, Amazon Web Services. Available at: <https://aws.amazon.com/eks/sla/> (Accessed: 22 December 2023).

² Microsoft (2023) Licensing Documents, Microsoft Licensing. Available at: <https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services?lang=1&assetType=285> (Accessed: 22 December 2023).

³ Amazon Web Services, Inc. (2022) Amazon Compute Service Level Agreement, Amazon Web Services. Available at: <https://aws.amazon.com/compute/sla/> (Accessed: 22 December 2023).

⁴ Amazon Web Services, Inc. (2022) Amazon RDS SLA, Amazon Web Services. Available at: <https://aws.amazon.com/rds/sla/> (Accessed: 22 December 2023).

⁵ Amazon Web Services, Inc. (2022) Amazon DocumentDB SLA, Amazon Web Services. Available at: <https://aws.amazon.com/documentdb/sla/> (Accessed: 22 December 2023).

⁶ Amazon Web Services, Inc. (2022) Amazon Elastic Load Balancing Service Level Agreement, Amazon Web Services. Available at: <https://aws.amazon.com/elasticloadbalancing/sla/> (Accessed: 22 December 2023).

Table 5 Comparing the platforms based on factors of maintainability

	AWS	Azure
Scalability	No noteworthy difference	No noteworthy difference
Uptime SLA	Lower uptime SLA	Significantly better uptime SLA
External tools	Supported by all significant tools	Supported by most of the significant tools

It should be noted that on both platforms, the maintainability doesn't have large fluctuations, partly because Kubernetes is used in all of the design examples, and both platforms have full support for it. While using Kubernetes, there probably wouldn't be many differences in maintainability between any of the larger cloud platforms. However, both of the discussed platforms also have some proprietary services and tools that could be used to host the software discussed in this project, and these services will be discussed in the next chapter.

7.3 Evaluating the possible use of vendor locking services and tools

As mentioned in the previous chapter, the software described in this thesis could also be hosted using the proprietary services of cloud platforms instead of Kubernetes. Using proprietary services like these has quite a large downside of essentially causing vendor lock-in. The lock-in can be especially problematic when the prices of the proprietary service would increase significantly, and switching to another platform or service would be costly.

Instead of using Kubernetes, AWS offers its proprietary solution for hosting container apps with Elastic Container Service or ECS. It is similar to AWS EKS in that it allows running containerised applications on either AWS EC2 instances or as serverless in AWS Fargate. However, ECS lacks the possibility to orchestrate containers.

Compared to AWS EKS, the pricing for the ECS is exactly the same, with the only real distinction being that there are no additional costs on top of the computing costs for the EKS cluster. Even though the EKS cluster has a monthly cost, it is not very expensive when considering the computing and database costs.

In terms of maintainability, Kubernetes seems to be the better option for most cases, with some small and simple applications possibly being better off using ECS instead. While both of the services have the same basic functionalities, Kubernetes seems to offer more freedom in configuring the system. Compared to a Kubernetes solution, ECS seems to be easier to set up, at least for smaller applications. It might become complex for larger applications with limited support for advanced configurations.

Overall, ECS seems to offer many of the same features as EKS but packaged in a simpler way to allow for easier software setup, with some savings coming from the fact that the EKS cluster is not needed. While it offers an easier setup, it lacks the configuration possibilities of Kubernetes and ties the software using it to AWS with no direct possibility of changing the cloud platform.

Azure Container Apps is Azure's version of providing simplified hosting for containerised software. Technically, it acts as a wrapper on top of Kubernetes and utilises it underneath. Compared to the ECS, it only offers serverless plans, which are billed based on the used vCPUs and memory with different prices for active and idle usage. The Container Apps also support completely scaling to zero, which can save a lot of money if the used software receives significant portions of idle time, as no costs are incurred while scaled to zero.

Compared to Kubernetes in terms of maintainability, the Container Apps have some flaws similar to AWS ECS but provide an easy-to-use alternative that should reduce the time needed to maintain the software by some time. However, as with ECS, Container Apps provide more limited possibilities to configure the system in a certain way and using it effectively locks the software to use it.

The portability, the ability to transfer your solution straight from one cloud to another, of these solutions is quite similar as neither of them supports it. While the use of containers has helped a lot with portability, more complex solutions would still require manual configuration for each cloud separately if, for some reason, a switch of provider is desired, while Kubernetes should only need small changes to work on different clouds.

In Table 6, AWS ECS and Azure Container Apps are compared to Kubernetes services based on the solutions' portability, costs, configurability, and ease of setup that were discussed in this section. From the table, it can be seen that Kubernetes provides the most configuration options with full portability at the cost of ease of setup and a bit higher price.

Table 6 Comparing vendor-specific services to Kubernetes

	ECS	Container Apps	Kubernetes
Portability	No	No	Yes
Cost	Lower	Lowest	Higher
Configuration	Basic	Basic	Advanced
Ease of setup	Easy	Easy	Harder

When considering whether to choose Kubernetes over vendor-specific service, it is important to weigh the cost savings both to the lower configurability and to the non-portability. With a small and simple application, these services might be the best choice as there is little configuration to do, which in turn makes possible cloud platform changes simpler. However, with larger and more complex solutions, Kubernetes still seems to be the best choice, even though it costs a bit more.

Overall, there are some interesting alternative proprietary technologies that can be found from both of the platforms. In this chapter, AWS ECS and Azure Container Apps were discussed, both providing a much easier and cheaper option for Kubernetes. However, with both of the services, the user is locked into the vendor to use the service as the configurations are not transferrable to other services. The services also have a lot fewer configuration options when compared to Kubernetes, as they seem to be solely focused on providing an easy-to-use platform instead of focusing on the more complex cases, such as the software examined in this thesis. In the end,

8. DISCUSSION

8.1 Limitations

While the thesis outlines an evaluation of the two largest cloud service providers for microservice hosting in terms of costs and maintainability, it is necessary to recognise the apparent absence of thorough studies in this area, which leaves a quite large research gap in the knowledge. This thesis aims to fill part of the research gap, but it should explicitly be recognised that this thesis alone can't close such a large knowledge gap. The research gap had some limiting effects while writing the thesis as there were no studies on which to base this thesis.

The larger research gap was found to be about the maintainability of cloud platforms, which seemed to have not been studied before, outside of some blog posts that touch the subject very narrowly. This lack of studies meant that there was no clear definition of the maintainability of cloud platforms available. The thesis aims to create a simple definition of maintainability in Chapter 3, but the subject should be studied more broadly in the future, as discussed in Chapter 8.2.

8.2 Further development

As this thesis only touches the surface of many of the topics covered, it leaves many possibilities for future developments. The first possible idea for further development could be implementing the same kind of solutions as shown in this thesis and then comparing the actual products and costs against each other. This kind of study would, of course, allow for a more realistic comparison of the providers.

Another idea could be to try out the lambdas and functions features of the cloud providers and compare them. Comparing these serverless features and their pricing to, for example, the features described in this thesis could provide interesting points for deciding whether to prefer serverless over the more traditional VMs for computing.

It would also probably be quite essential to compare some of the other larger cloud providers to Amazon Web Services and Microsoft Azure as those could provide, for example, better pricing to compete with the largest providers. It would also be beneficial to compare these cloud providers to an on-premises solution as some companies have decided to return to on-premises at least partly after the public cloud providers have, in reality, cost significantly more than initially thought.

Lastly, a study could be done on the subject of cloud maintainability to define it more concretely as the subject was only briefly introduced in this thesis and studying it in more detail could fill a quite large research gap and provide the baseline definition for it. Overall, I think the study of cloud maintainability would generally be the most beneficial of these four ideas. Still, it would probably also be the most difficult one to make.

9. CONCLUSIONS

A growing number of businesses are beginning to divide their large software products into smaller microservices as a result of the advancements in microservices, containerisation, and other tools in recent years. With this kind of change spreading worldwide, the customer mentioned in this thesis had started their own process of decoupling their monolithic software into microservices.

The thesis started with a literary review focusing on cloud computing, emphasising technologies influencing microservices. Then, the maintainability of cloud platforms was discussed with the findings that the subject has not been studied previously. Due to the lack of a proper definition of maintainability, a simple definition of it was introduced in this thesis. In the final part of the theoretical part, the software product developed by the customer was introduced and analysed.

One of the goals of this thesis was to compare Amazon Web Services and Microsoft Azure, the two largest cloud platforms at the time of writing, for hosting microservices in terms of the costs and maintainability of the platforms. It was decided that only services provided directly by the platforms would be utilised in the thesis.

For the chosen two cloud platforms, three different example designs were drawn that presented different kinds of Kubernetes utilising microservice deployments that the customer's software could use. Costs for these designs were then calculated for both platforms, with some differences showing up between the platforms. The platforms were then compared in terms of subjective and objective maintainability metrics, such as uptime SLA and support for external tools.

The results showed that AWS was generally cheaper than Azure, with the largest differences coming from database services. However, it was noticed that Azure had cheaper virtual machines available than AWS ones with the same specifications. Azure was deemed better in terms of maintainability, with its promised availability being a bit higher than AWS's. In the end, when using only the services provided by the platforms, AWS would generally be the better choice for hosting microservices if there are no other reasons to use Azure, such as previous know-how.

The second goal of this thesis was to determine if it would be better in terms of costs and maintainability to use vendor locking services and tools of the cloud platforms. Azure Container Apps and AWS ECS were chosen to represent these proprietary tools as they closely match Kubernetes' features. The services were compared across portability,

cost, configuration options and ease of setup. The proprietary services were priced competitively to attract customers; using ECS doesn't cost anything besides the EC2 instance costs. Azure Container Apps was only available in serverless variants, so it was difficult to say whether anything was added to the price. The services provide much more maintainable options for Kubernetes but at the cost of losing many of the configuration options provided by Kubernetes and with no portability, resulting in a vendor lock-in. Overall, vendor-specific alternatives could be recommended for small and simple software but not for the kind discussed in this thesis, which benefits from the advanced configuration options provided by Kubernetes.

The study could be considered successful as both research questions were researched and answered. However, the maintainability side of the first research question did not receive as deep study as I would have liked since there were no good definitions available for the maintainability of cloud platforms. It should also be noted that some external services, like Mongo Atlas, should probably have been included in the study as there doesn't seem to be any reason not to use it over Azure Cosmos DB, for example.

The study can be considered reliable at the time of its writing, but the pricing of cloud services changes constantly, with both AWS and Azure launching new services quite often. These changes mean that the results of this study could, in the worst case, be outdated quite quickly. However, the services mentioned in this thesis can be considered mature, with very few changes affecting them directly. In any case, the reader should consider the current situation at the time of reading before planning any changes based on this study.

REFERENCES

A. Khan (2017) 'Key Characteristics of a Container Orchestration Platform to Enable a Modern Application', *IEEE Cloud Computing*, 4(5), pp. 42–48. Available at: <https://doi.org/10.1109/MCC.2017.4250933>.

Amazon Web Services, Inc. (no date a) *Amazon CloudWatch Pricing – Amazon Web Services (AWS)*, *Amazon Web Services*. Available at: <https://aws.amazon.com/cloudwatch/pricing/> (Accessed: 30 October 2023).

Amazon Web Services, Inc. (no date b) *Amazon DocumentDB, Amazon Web Services*. Available at: <https://aws.amazon.com/documentdb/> (Accessed: 31 March 2023).

Amazon Web Services, Inc. (no date c) *Amazon RDS FAQs | Cloud Relational Database | Amazon Web Services, Amazon Web Services*. Available at: <https://aws.amazon.com/rds/faqs/> (Accessed: 6 November 2023).

Amazon Web Services, Inc. (no date d) *AWS Pricing Calculator, Amazon Web Services*. Available at: <https://calculator.aws/#/> (Accessed: 4 November 2023).

Amazon Web Services, Inc. (no date e) *Concepts - AWS Well-Architected Framework*. Available at: <https://wa.aws.amazon.com/wat.concepts.wa-concepts.en.html> (Accessed: 5 May 2023).

Amazon Web Services, Inc. (no date f) *Fully Managed Container Solution – Amazon Elastic Container Service (Amazon ECS) - Amazon Web Services, Amazon Web Services*. Available at: <https://aws.amazon.com/ecs/> (Accessed: 31 March 2023).

Amazon Web Services, Inc. (no date g) *Global Infrastructure Regions & AZs, Amazon Web Services*. Available at: https://aws.amazon.com/about-aws/global-infrastructure/regions_az/ (Accessed: 13 September 2023).

Amazon Web Services, Inc. (no date h) *Hosted PostgreSQL - Amazon RDS for PostgreSQL - AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/rds/postgresql/> (Accessed: 31 March 2023).

Amazon Web Services, Inc. (no date i) *Load Balancer - Elastic Load Balancing (ELB) - AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/elasticloadbalancing/> (Accessed: 4 February 2024).

Amazon Web Services, Inc. (no date j) *Managed Kubernetes Service – Amazon EKS – Amazon Web Services, Amazon Web Services*. Available at: <https://aws.amazon.com/eks/> (Accessed: 16 March 2023).

Amazon Web Services, Inc. (no date k) *Secure and resizable cloud compute – Amazon EC2 – Amazon Web Services, Amazon Web Services*. Available at: <https://aws.amazon.com/ec2/> (Accessed: 31 March 2023).

Amazon Web Services, Inc. (no date l) *Serverless Compute Engine – AWS Fargate – AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/fargate/> (Accessed: 13 September 2023).

Amazon Web Services, Inc. (no date m) *Serverless Computing - AWS Lambda - Amazon Web Services, Amazon Web Services*. Available at: <https://aws.amazon.com/lambda/> (Accessed: 18 March 2023).

Amazon Web Services, Inc. (no date n) *Website & Web App Deployment - AWS Elastic Beanstalk - AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/elasticbeanstalk/> (Accessed: 3 March 2023).

Amazon Web Services, Inc. (no date o) *What is AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/what-is-aws/> (Accessed: 3 February 2023).

Amazon Web Services, Inc. (no date p) *What is IaaS? - Infrastructure as a Service Explained - AWS, Amazon Web Services*. Available at: <https://aws.amazon.com/what-is/iaas/> (Accessed: 3 March 2023).

Butler, B. (2013) 'PaaS Primer: What is platform as a service and why does it matter? | Network World', 11 February. Available at: <https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html> (Accessed: 3 March 2023).

C. Pahl *et al.* (2019) 'Cloud Container Technologies: A State-of-the-Art Review', *IEEE Transactions on Cloud Computing*, 7(3), pp. 677–692. Available at: <https://doi.org/10.1109/TCC.2017.2702586>.

Chandrakant, K. (2022) *Mesos vs. Kubernetes | Baeldung, Baeldung*. Available at: <https://www.baeldung.com/ops/mesos-kubernetes-comparison> (Accessed: 4 May 2023).

Chandrasekaran, K. and Ananth, A. (2016) 'Cloud Services and Service Providers', in S. Murugesan and I. Borislava (eds) *Encyclopedia of cloud computing*. Chichester, West Sussex, United Kingdom; Wiley (Wiley - IEEE), pp. 17–28. Available at: <https://doi.org/10.1002/9781118821930>.

Cloudflare, Inc. (no date) *What is BaaS? | Backend-as-a-Service vs. serverless | Cloudflare, Cloudflare*. Available at: <https://www.cloudflare.com/en-gb/learning/serverless/glossary/backend-as-a-service-baas/> (Accessed: 19 March 2023).

Crouch, S. (no date) *Developing maintainable software | Software Sustainability Institute*. Available at: <https://software.ac.uk/resources/guides/developing-maintainable-software> (Accessed: 5 May 2023).

Datadog, Inc. (2021) *The State of Serverless 2021 | Datadog, Datadog*. Available at: <https://www.datadoghq.com/state-of-serverless-2021/> (Accessed: 19 March 2023).

Datadog, Inc. (2022) *The State of Serverless | Datadog, Datadog*. Available at: <https://www.datadoghq.com/state-of-serverless/> (Accessed: 19 March 2023).

Docker Inc. (no date a) *Docker Hub Container Image Library | App Containerization*. Available at: <https://hub.docker.com/> (Accessed: 16 March 2023).

Docker Inc. (no date b) *What is a Container? | Docker*. Available at: <https://www.docker.com/resources/what-container/> (Accessed: 16 March 2023).

Donnelly, C. (2021) *Inside a Microsoft Azure datacentre: Cloud giant invites users on server farm virtual tour | Computer Weekly*. Available at: <https://www.computerweekly.com/news/252499582/Inside-a-Microsoft-Azure-datacentre-Cloud-giant-invites-users-on-server-farm-virtual-tour> (Accessed: 2 March 2023).

Eismann, S. *et al.* (2020) 'Serverless applications: Why, when, and how?', *IEEE Software*, 38(1), pp. 32–39.

Erl, T., Mahmood, Z. and Puttini, R. (2013) *Cloud computing: concepts, technology, & architecture*. 1st edition. Place of publication not identified: Prentice Hall (The Prentice Hall service technology series from Thomas Erl).

Harris, C. (no date) *Microservices vs. monolithic architecture* | Atlassian, Atlassian. Available at: <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith> (Accessed: 17 March 2023).

Hasan, M.H. *et al.* (2023) 'From Monolith to Microservice: Measuring Architecture Maintainability', *International Journal of Advanced Computer Science and Applications*, 14(5). Available at: <https://doi.org/10.14569/IJACSA.2023.0140591>.

Herbst, N.R., Kounev, S. and Reussner, R.H. (2013) 'Elasticity in Cloud Computing: What It Is, and What It Is Not.', in. *ICAC*, pp. 23–27.

Hevner, A. and Chatterjee, S. (2010) *Design science research in information systems*. Springer New York, NY (Integrated Series in Information Systems).

Hevner, A.R. (2007) 'A three cycle view of design science research', *Scandinavian journal of information systems*, 19(2), p. 4.

IBM Cloud (no date a) *What are containers?* | IBM, IBM. Available at: <https://www.ibm.com/topics/containers> (Accessed: 16 March 2023).

IBM Cloud (no date b) *What is container orchestration?* | IBM, IBM. Available at: <https://www.ibm.com/topics/container-orchestration> (Accessed: 16 March 2023).

IBM Cloud (no date c) *What is FaaS (Function-as-a-Service)?* | IBM, IBM. Available at: <https://www.ibm.com/sg-en/topics/faas> (Accessed: 18 March 2023).

'IEEE Standard Glossary of Software Engineering Terminology' (1990) *IEEE Std 610.12-1990*, pp. 1–84. Available at: <https://doi.org/10.1109/IEEESTD.1990.101064>.

Johannesson, P. and Perjons, E. (2014) *An introduction to design science*. Springer.

Khan, A.M., Freitag, F. and Navarro, L. (2016) 'Community Clouds', in S. Murugesan and I. Borislava (eds) *Encyclopedia of cloud computing*. Chichester, West Sussex, United Kingdom; Wiley (Wiley - IEEE), pp. 41–51. Available at: <https://doi.org/10.1002/9781118821930>.

Kulkarni, G., Gambhir, J. and Palwe, R. (2012) 'Cloud computing-software as service', *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, 1(1), pp. 11–16.

Lionel Sujay Vailshery (2023) *Global cloud computing revenue by segment 2022* | Statista, Statista. Available at: <https://www.statista.com/statistics/540499/worldwide-cloud-computing-revenue-by-segment/> (Accessed: 21 July 2023).

Lloyd, W. *et al.* (2018) 'Serverless Computing: An Investigation of Factors Influencing Microservice Performance', in *2018 IEEE International Conference on Cloud Engineering (IC2E). 2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 159–169. Available at: <https://doi.org/10.1109/IC2E.2018.00039>.

Maayan, G.D. (2022) *AWS vs. Azure Cost Comparison [2022]* - Codemotion Magazine, Codemotion. Available at: <https://www.codemotion.com/magazine/devops/cloud/aws-vs-azure-cost-comparison/> (Accessed: 23 February 2023).

Macmanus, R. (2008) 'Ray Ozzie Announces Windows Azure - "Windows in the Cloud" - NYTimes.com', 27 October. Available at: https://archive.nytimes.com/www.nytimes.com/external/readwriteweb/2008/10/27/27readwriteweb-windows_azure.html (Accessed: 23 February 2023).

Mathew, S. (2021) 'Overview of Amazon Web Services - AWS Whitepaper'. Available at: <https://d1.awsstatic.com/whitepapers/aws-overview.pdf> (Accessed: 26 January 2023).

Mell, P. and Grance, T. (2011) 'The NIST definition of cloud computing'.

Microsoft (2022) *Microsoft announces intent to build a new datacenter region in Finland, accelerating sustainable digital transformation and enabling large scale carbon-free district heating - Microsoft News Centre Europe, Microsoft News Centre Europe*. Available at: <https://news.microsoft.com/europe/2022/03/17/microsoft-announces-intent-to-build-a-new-datacenter-region-in-finland-accelerating-sustainable-digital-transformation-and-enabling-large-scale-carbon-free-district-heating/> (Accessed: 13 September 2023).

Microsoft (no date a) *Application Gateway - Load-Balancing Solution | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/application-gateway/> (Accessed: 4 February 2024).

Microsoft (no date b) *Azure Container Apps | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/container-apps> (Accessed: 31 March 2023).

Microsoft (no date c) *Azure Cosmos DB - NoSQL and Relational Database | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/cosmos-db/> (Accessed: 31 March 2023).

Microsoft (no date d) *Azure Database for PostgreSQL | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/postgresql/> (Accessed: 31 March 2023).

Microsoft (no date e) *Azure Functions – Serverless Functions in Computing | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/functions/> (Accessed: 19 March 2023).

Microsoft (no date f) *Choose the Right Azure Region for You | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/explore/global-infrastructure/geographies/#geographies> (Accessed: 13 September 2023).

Microsoft (no date g) *Managed Kubernetes Service (AKS) | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/kubernetes-service> (Accessed: 16 March 2023).

Microsoft (no date h) *Pricing Calculator | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/pricing/calculator/> (Accessed: 4 November 2023).

Microsoft (no date i) *Public Cloud vs Private Cloud vs Hybrid Cloud | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds/?cdn=disable#public-cloud> (Accessed: 18 June 2023).

Microsoft (no date j) *Virtual Machine Scale Sets | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/virtual-machine-scale-sets/> (Accessed: 12 May 2023).

Microsoft (no date k) *Virtual Machines (VMs) for Linux and Windows | Microsoft Azure, Azure*. Available at: <https://azure.microsoft.com/en-us/products/virtual-machines> (Accessed: 31 March 2023).

Microsoft (no date l) *What is Azure—Microsoft Cloud Services | Microsoft Azure, What is Azure?* Available at: <https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-azure/> (Accessed: 23 February 2023).

Microsoft Learn contributors (2022) *Servers - Azure Database for PostgreSQL - Flexible Server | Microsoft Learn, Microsoft Learn*. Available at: <https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/concepts-servers> (Accessed: 6 November 2023).

Microsoft Learn contributors (2023) *Azure Virtual Machine Scale Sets overview - Azure Virtual Machine Scale Sets | Microsoft Learn, Microsoft Learn*. Available at: <https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview> (Accessed: 12 May 2023).

Newman, S. (2021) *Building microservices: designing fine-grained systems*. Second edition. Sebastopol, CA: O'Reilly Media Inc.

Okta (2023) *What Is BaaS (Backend as a Service)? Definition and Usage* | Okta, Okta. Available at: <https://www.okta.com/identity-101/baas-backend-as-a-service/> (Accessed: 19 March 2023).

Open International (2019) *The three service models of Cloud Computing* | OPEN, Open. Available at: <https://www.openintl.com/the-three-service-models-of-cloud-computing/> (Accessed: 21 July 2023).

Peppers, K. et al. (2007) 'A Design Science Research Methodology for Information Systems Research', *Journal of management information systems*, 24(3), pp. 45–77. Available at: <https://doi.org/10.2753/MIS0742-1222240302>.

Peppers, K. et al. (2012) 'Design science research evaluation', in. *Design Science Research in Information Systems. Advances in Theory and Practice: 7th International Conference, DESRIST 2012, Las Vegas, NV, USA, May 14-15, 2012. Proceedings 7*, Springer, pp. 398–410.

Powell, R. (2023) *Docker Swarm vs Kubernetes* | CircleCI, CircleCI Blog. Available at: <https://circleci.com/blog/docker-swarm-vs-kubernetes/> (Accessed: 4 May 2023).

Roberts, M. (2018) *Serverless Architectures*, Martin Fowler. Available at: <https://martinfowler.com/articles/serverless.html> (Accessed: 17 March 2023).

Rosen, C. (2022) *Docker Swarm vs. Kubernetes: A Comparison* | IBM, IBM. Available at: <https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison> (Accessed: 4 May 2023).

Salah, T. et al. (2017) 'Performance comparison between container-based and VM-based services', in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN). 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 185–190. Available at: <https://doi.org/10.1109/ICIN.2017.7899408>.

Sharif, A. (2022) *Kubernetes vs. Mesos - CrowdStrike*, CrowdStrike. Available at: <https://www.crowdstrike.com/cybersecurity-101/observability/kubernetes-vs-mesos/> (Accessed: 4 May 2023).

Sharma, H. (2022) *Infrastructure As Code: A Comprehensive Guide* | ReviewNPrep. Available at: <https://reviewnprep.com/blog/infrastructure-as-code-a-comprehensive-guide/> (Accessed: 28 April 2023).

Stack Overflow (2022) *Stack Overflow Developer Survey 2022*, Stack Overflow. Available at: <https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-other-tools> (Accessed: 16 March 2023).

Sweeney, J. (2016) 'Virtualization: An Overview', in S. Murugesan and I. Borislava (eds) *Encyclopedia of cloud computing*. Chichester, West Sussex, United Kingdom; Wiley (Wiley - IEEE), pp. 89–101. Available at: <https://doi.org/10.1002/9781118821930>.

Synergy Research Group (2022) *Q3 Cloud Spending Up Over \$11 Billion from 2021 Despite Major Headwinds; Google Increases its Market Share*, Synergy Research Group. Available at: <https://www.srgresearch.com/articles/q3-cloud-spending-up-over-11-billion-from-2021-despite-major-headwinds-google-increases-its-market-share> (Accessed: 3 February 2023).

Taylor, D. (2023) *Azure vs AWS: Difference Between Them*, Guru99. Available at: <https://www.guru99.com/azure-vs-aws.html> (Accessed: 19 March 2023).

The Kubernetes Authors (2023a) *Considerations for large clusters* | Kubernetes. Available at: <https://kubernetes.io/docs/setup/best-practices/cluster-large/> (Accessed: 29 October 2023).

The Kubernetes Authors (2023b) *Deployments* | *Kubernetes*. Available at: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/> (Accessed: 16 March 2023).

The Kubernetes Authors (2023c) *Nodes* | *Kubernetes*. Available at: <https://kubernetes.io/docs/concepts/architecture/nodes/> (Accessed: 16 March 2023).

The Kubernetes Authors (2023d) *Pods* | *Kubernetes*. Available at: <https://kubernetes.io/docs/concepts/workloads/pods/> (Accessed: 16 March 2023).

The Kubernetes Authors (2023e) *Service* | *Kubernetes*. Available at: <https://kubernetes.io/docs/concepts/services-networking/service/> (Accessed: 16 March 2023).

Thönes, J. (2015) 'Microservices', *IEEE software*, 32(1), pp. 116–116.

Villamizar, M. *et al.* (2017) 'Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures', *Service Oriented Computing and Applications*, 11(2), pp. 233–247. Available at: <https://doi.org/10.1007/s11761-017-0208-y>.

Watts, S. and Raza, M. (2019) 'SaaS vs PaaS vs IaaS: What's The Difference & How To Choose – BMC Software | Blogs', *BMC Software*, 15 June. Available at: <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/> (Accessed: 3 March 2023).

Zhang, M. (2022) 'Microsoft Azure's Data Center Locations: Regions and Availability Zones', *Dgtl Infra*, 11 April. Available at: <https://dgtlinfra.com/microsoft-azure-data-center-locations/> (Accessed: 24 February 2023).