



**STRATEGIES AND CHALLENGES IN CLOUD-TO-CLOUD MIGRATION
USING INFRASTRUCTURE AS CODE**

Lappeenranta–Lahti University of Technology LUT

Master's Programme in Software Engineering and Digital Transformation

2024

Teemu Ketonen

Examiners: Professor Kari Smolander

Associate Professor Jussi Kasurinen

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Sciences

Software Engineering

Teemu Ketonen

Strategies and challenges in Cloud-to-Cloud migration using Infrastructure as Code

Master's thesis

2024

67 pages, 17 figures, and 4 tables

Examiners: Professor Kari Smolander and Associate Professor Jussi Kasurinen

Keywords: cloud computing, cloud migration, cloud-to-cloud migration, infrastructure-as-code

This thesis explores the increasingly relevant field of cloud-to-cloud migration, facilitated through Infrastructure as Code (IaC) technique. As the landscape of cloud computing continues to transform at a rapid pace, organizations seek to optimize operational costs, enhance system flexibility, and adjust to changing market demands. In this context, the capability to efficiently migrate between cloud providers becomes an important factor.

The research was conducted as a case study within a Finnish startup operating in self-service mobile payment systems. It provides a detailed analysis of migrating an application architecture from DigitalOcean to AWS using the AWS Cloud Development Kit (CDK), providing insights into the strategies, challenges, and outcomes of the process. The key findings suggest that while cloud migration involves considerable technical and operational challenges, the utilization of IaC can significantly mitigate these obstacles, enhancing the speed, reliability, and security of the migration process. This research contributes to the limited literature on cloud-to-cloud migration, highlighting the effective use of IaC to facilitate smoother transitions between cloud environments. The findings also provide practical insights for organizations planning similar migrations.

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

LUTin insinööritieteiden tiedekunta

Tietotekniikka

Teemu Ketonen

Strategiat ja haasteet pilvipalveluntarjoajan vaihtamisessa käyttäen infrastruktuuri koodina -menetelmää

Tietotekniikan diplomityö

2024

67 sivua, 17 kuvaa ja 4 taulukkoa

Tarkastajat: Professori Kari Smolander ja Apulaisprofessori Jussi Kasurinen

Avainsanat: pilvilaskenta, pilveen siirtyminen, pilvestä pilveen siirtyminen, pilvipalveluntarjoajan vaihtaminen, infrastruktuuri koodina

Tässä diplomityössä käsitellään sovellusarkkitehtuurin siirtämistä pilvipalveluntarjoajalta toiseen hyödyntäen infrastruktuuri koodina (IaC) -menetelmää. Organisaatiot pyrkivät optimoimaan kustannuksiaan, parantamaan järjestelmiään ja sopeutumaan muuttuviin markkinoihin pilvipalveluita hyödyntämällä. Palveluntarjoajien vaihtaminen on noussut keskeiseen asemaan nopean pilvipalveluiden kehityksen seurauksena.

Työ suoritettiin tapaustutkimuksena suomalaisessa startup-yrityksessä, joka tarjoaa mobiilipohjaisia itsepalveluratkaisuja. Työssä analysoidaan yksityiskohtaisesti sovellusarkkitehtuurin siirtoa DigitalOceanista AWS:ään käyttäen AWS Cloud Development Kit -työkalua. Työn tarkoitus oli tutkia siirtoprosessin strategian muodostamista, haasteita ja tuloksia yhdistäen IaC:n vaikutus koko prosessiin.

Työn keskeiset havainnot osoittavat, että vaikka pilvipalveluntarjoajan vaihtamiseen liittyy huomattavia teknisiä ja operatiivisia haasteita, IaC:n hyödyntäminen voi merkittävästi vähentää näitä esteitä, parantaen prosessin nopeutta, luotettavuutta ja turvallisuutta. Tämä työ edistää pilvipalveluiden välisen siirtymisen rajallista kirjallisuutta korostaen samalla IaC:n merkitystä siirtymien mahdollistajana pilviympäristöjen välillä. Työn tulokset tarjoavat myös käytännön näkemyksiä organisaatioille, jotka suunnittelevat vastaavia siirtymiä.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor, Kari, for his guidance and support throughout this journey. The LUT Entrepreneurship Society (LUTES) community has also been also very helpful and supportive, making sure I have enough nutrition and coffee to survive.

I am also grateful to my friends for being there by my side during this time. Aki deserves a special mention for his invaluable insights, peer support, and guidance whenever I needed it. Finally, special thanks go to Asta and Anna for their ongoing interest and inquiries about the progress of my thesis.

ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CAGR	Compound Annual Growth Rate
CDN	Content Delivery Network
CRM	Customer Relationship Management
FaaS	Function-as-a-Service
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
IaaS	Infrastructure-as-a-Service
IaC	Infrastructure as Code
NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
PaaS	Platform-as-a-Service
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SMEs	Small and medium-sized enterprises
TOSCA	Topology and Orchestration Specification for Cloud Applications

Table of Contents

Abstract	
Acknowledgements	
Abbreviations	
1	Introduction 8
2	Literature Review 11
2.1	Cloud Computing..... 11
2.2	Cloud Migration..... 14
2.2.1	Migration Challenges..... 14
2.2.2	Cloud Migration Strategies 17
2.2.3	Cloud Portability 19
2.2.4	Migration Frameworks and Models..... 22
2.3	Infrastructure as Code 23
2.3.1	Best Practices 26
2.3.2	Benefits and Challenges..... 28
3	Research Methodology 31
3.1	Research Method 31
3.2	Case Organization..... 32
3.3	Case Description and Motivation 33
4	Migration Planning..... 37
4.1	Migration Strategy Selection 37
4.2	Selection of AWS Services and Resources..... 38
4.3	IaC Utilization..... 42
4.4	Migration Plan 44
5	Results 47
5.1	Infrastructure Implementation 47
5.2	Overall Migration Process 51
6	Discussion..... 53
6.1	Discussion of Results..... 53

6.2	Lessons Learned	56
6.3	Limitations and Future Work.....	58
7	Conclusions	59
	References.....	61

1 Introduction

The cloud computing market is showing significant growth, projected to grow at a compound annual growth rate (CAGR) of 19.6% from 2021 to 2027. According to Gartner's 2024 forecast, the market valuation is expected to reach 1 trillion euros. (Gartner, 2024) This rapid expansion highlights the fundamental shift in how businesses are leveraging cloud computing to boost operational efficiency, scalability, and cost-effectiveness. However, as market demands and dynamics change, organizations are required to continuously refine their IT strategies. This often involves re-evaluating the used cloud service providers to ensure alignment with current requirements for cost, performance, and features.

Consequently, cloud-to-cloud migration, involving the transfer of applications, data, and services between cloud platforms, has become an important concept. This process presents numerous challenges, including the need to maintain system integrity and minimize operational disruptions. Additionally, it demands careful consideration of technology choices, strategic planning, and the implementation of best practices to guarantee smooth and secure transitions.

Previous studies have extensively explored individual aspects of cloud migration, such as security concerns, downtime minimization, and cost efficiency. However, there is a noticeable gap in comprehensive empirical research focused specifically on cloud-to-cloud migrations (Kolb, Lenhard and Wirtz, 2015). Studies have predominantly concentrated on migration from on-premises infrastructures to the cloud, rather than migrations between cloud platforms. This leaves a critical knowledge gap regarding the unique challenges and opportunities that arise during cloud-to-cloud transitions, such as interoperability issues, data loss prevention, and the maintenance of service availability during the migration process. Furthermore, as highlighted by Kolb et al. (2015) more case studies are required in the cloud-to-cloud migration area. Linthicum (2017) also supports this claim by mentioning that only a few cloud-to-cloud transitions have happened.

Additionally, Infrastructure as Code (IaC) has become an emerging approach that facilitates the management and deployment of computing infrastructure through codified files. This approach significantly reduces the likelihood of errors compared to traditional manual configuration methods. IaC is particularly advantageous in automating processes,

minimizing errors, and enhancing reproducibility, which are essential for efficiently managing complex or large-scale environments. Despite its potential benefits, the specific challenges and strategies involved in implementing cloud-to-cloud migrations utilizing IaC remain underexplored. This research aims to address this gap by investigating these challenges and evaluating the impact of IaC on the cloud-to-cloud migration process.

The goal of this research is to conduct a detailed analysis of cloud-to-cloud application migration, with a specific emphasis on the utilization of Infrastructure as Code to streamline this process. The study aims to contribute valuable insights to the academic field and provide practical insights for organizations planning cloud-to-cloud migration initiatives. The study sets out to identify the key challenges associated with cloud-to-cloud application migration, to evaluate the effectiveness of IaC in addressing these challenges. The research will use a case study methodology to answer the following research questions:

- What are the key challenges, strategies, and outcomes involved in migrating an application architecture between cloud providers?
- What are the benefits and challenges of using IaC in the migration process?

Certain limitations have been established to focus the scope and maintain clarity throughout the investigation of cloud-to-cloud migration using IaC. First, it is important to note that this study does not aim to compare or evaluate different cloud architectures or analyse the decision-making processes behind choosing to switch cloud providers. Such evaluations are beyond the scope of this thesis and involve a broader set of strategic considerations that are not addressed here. Additionally, the variability in cloud architectures and the specific configurations of services pose a significant limitation. The findings and strategies discussed may not be universally applicable to all cloud environments due to differences in infrastructure and service setups. Each cloud provider offers unique features and configurations, which could affect the transferability of the proposed solutions.

Furthermore, this research does not address the lock-in effect often associated with cloud providers. The lock-in effect can influence the ease or difficulty of migration processes, as proprietary technologies and platform-specific services can create dependencies that complicate migration efforts. This aspect is not considered within the scope of this thesis,

which could affect the applicability of the strategies in environments where lock-in is a significant factor.

In the next chapter, the theoretical foundation and related research on cloud computing, cloud migration, and Infrastructure as Code are presented. Chapter 3 introduces the research method and case project, setting the stage for an in-depth examination. Chapter 4 presents the different steps involved in the migration process, detailing each phase from planning to execution. Chapter 5 presents the results of the migration process, providing a detailed analysis of outcomes and achievements. Chapter 6 discusses the key results, connecting them to the existing body of research while highlighting their implications and lessons learned. Finally, the conclusions are presented in Chapter 7, summarizing the study and suggesting paths for future research.

2 Literature Review

This chapter establishes the theoretical framework to provide context for the remainder of the thesis. First, the importance of cloud migration, its challenges, and various strategies are presented. Additionally, relevant research in the context of cloud-to-cloud migration is presented to provide a thorough overview and to connect it to the research gap. Finally, IaC is introduced along with its advantages and challenges, highlighting its role in streamlining infrastructure migration and presenting established best practices for its effective utilization.

2.1 Cloud Computing

Cloud computing represents a significant transformation in how organizations manage and deploy their computing resources (Vaquero *et al.*, 2008; Zhang, Cheng and Boutaba, 2010). National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" (Mell and Grance, 2011). This means that instead of acquiring the hardware needed for applications and setting up on-premises infrastructure, organizations can lease these resources from cloud providers.

This flexibility not only supports fluctuating workloads but also contributes to significant cost savings, as it eliminates the need for big initial hardware investments and lowers the expenses related to over or underutilizing resources. Furthermore, the total operating expenses are lower because infrastructure provisioning and scalability require less management and configuration. (Shayan *et al.*, 2013) For example, cloud services often come with robust security measures, regular updates, and maintenance handled by the provider, reducing the maintenance needed from an organization (Zhang, Cheng and Boutaba, 2010). By utilizing cloud computing, businesses can innovate faster and adapt to changing market demands more efficiently. This transforms the operational models and facilitates new ways of working. (Haris and Khan, 2018) Cloud computing can be categorized into three service models: Infrastructure as a Service (IaaS), Platform as a

Service (PaaS), and Software as a Service (SaaS). (Mell and Grance, 2011) These models are displayed in Figure 1 with their distinct characteristics.

On premise	IaaS	PaaS	SaaS
Application	Application	Application	Application
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Operating system	Operating system	Operating system	Operating system
Virtualization	Virtualization	Virtualization	Virtualization
Networking	Networking	Networking	Networking
Storage	Storage	Storage	Storage
Servers	Servers	Servers	Servers

User managed
 Provider managed

Figure 1. Differences between cloud service models (Spjuth, Frid and Hellander, 2021)

IaaS offers the essential computing components, including virtual servers, storage, networking, and operating systems. While users lack authority over the underlying infrastructure, they can manage the operating system and its configuration. This allows for high levels of customization but also requires significant management expertise. (Mell and Grance, 2011) Some well-known examples of IaaS include Amazon Elastic Compute Cloud (EC2), DigitalOcean droplets, Microsoft Azure, and Google Compute Engine (GCE).

PaaS provides a managed platform for creating, deploying, and overseeing applications. This service model removes the complexity of managing underlying infrastructure, as users can access pre-configured environments and tools tailored for application development. (Mell and Grance, 2011) Furthermore, PaaS simplifies the development process by providing automated updates, provisioning, and scalability (Haris and Khan, 2018). This streamlining allows developers to concentrate on writing code rather than managing administrative duties

(Spjuth, Frid and Hellander, 2021). Popular PaaS offerings include AWS Lambda, Google App Engine, and Heroku.

SaaS offers fully functional software applications via the internet, often through monthly subscription models (Spjuth, Frid and Hellander, 2021). The underlying infrastructure cannot be managed except for application configuration settings (Mell and Grance, 2011). SaaS providers manage both the software application and the infrastructure, including maintenance tasks like updates and security fixes. Consumers access the application via the internet through a web browser on their device. (Haris and Khan, 2018) This model frees users from complex software and hardware management, making it ideal for businesses that want to streamline operations and focus on their core activities without the hassle of maintaining software and hardware. Some examples of SaaS are Gmail, Figma, and Dropbox.

Furthermore, beyond the service models, cloud computing deployment varies based on the organization's security and compliance requirements. These deployment models are public, private, hybrid and community clouds (Mell and Grance, 2011; Haris and Khan, 2018). Public clouds are managed by third-party cloud vendors who serve their infrastructure, like computing and storage, over the internet. AWS, Microsoft Azure, DigitalOcean, and Google Cloud are examples of popular public clouds. The main advantage of the public cloud is its scalability and elasticity, along with its per-usage pricing model, which makes it an attractive option for businesses that need to scale resources according to fluctuations in demand.

In contrast, private cloud is exclusively allocated to a single organization for their network usage and deployments. Private cloud infrastructure is typically hosted on-premises and offers greater control over security and compliance. (Srivastava and Khan, 2018) However, private cloud can be less cost-effective and more complex to manage than public cloud options depending on the organization's context and requirements (Haris and Khan, 2018).

Hybrid clouds merge public and private cloud environments, interconnected through technology that enables the sharing of data and applications between them (Mell and Grance, 2011). This gives businesses more flexibility, more deployment choices, and to migrate workloads based on specific needs. This model is particularly useful for balancing between flexibility and scalability of the public cloud with the security features of a private cloud. (Haris and Khan, 2018)

Lastly, community cloud provides infrastructure that is shared by a specific group of organizations. These groups have common interests, such as government agencies or research institutions that have similar needs in security and regulatory compliance. (Mell and Grance, 2011) This model offers a collaborative environment where infrastructure and resources are shared, combining aspects of privacy and control found in private clouds with the cost-effectiveness of public cloud (Goyal, 2014).

2.2 Cloud Migration

As presented earlier, the benefits of cloud computing drive organizations to move their legacy applications running on-premises infrastructure to the cloud. This process of moving applications or any parts thereof to the cloud is called cloud migration. (Jamshidi, Ahmad and Pahl, 2013) The key drivers for the migration include decreased starting and operating costs, efficient resource utilization, scalability, and easier maintainability (Jamshidi, Ahmad and Pahl, 2013; Rai, Sahoo and Mehfuz, 2015).

Cloud migration has been studied extensively in the literature from many perspectives. Jamshidi et al. (2013) conducted a systematic literature review on previous cloud migration research and found articles reporting their experiences, lessons learned, and best practices. Furthermore, some have developed frameworks and processes to facilitate the decision-making journey towards migration to the cloud. The existing research highlights cloud migration as a complex process that necessitates thorough planning and the evaluation of multiple factors to guarantee a secure and smooth transition to the cloud (Gholami *et al.*, 2016).

2.2.1 Migration Challenges

While moving to the cloud comes with many benefits there are challenges and concerns that need to be addressed before considering such a move. Migration tools are provided from big companies, like AWS and Azure, to ease and support the transition process of on-premises applications to the cloud (Balobaid and Debnath, 2018; Shastry *et al.*, 2022). Furthermore, frameworks and processes have been developed to simplify the decision-making journey towards migration to the cloud. Despite the support these tools, frameworks, and

organizations provide, there remain obstacles and critical factors to consider in the migration process. Many studies have explored the challenges in cloud migration, and in general cloud migration and cloud computing share similar characteristics in challenges (Gholami *et al.*, 2016, 2017; Dar, 2018; Maniah *et al.*, 2022; Staevsky and Gaftandzhieva, 2023; Wirasti *et al.*, 2023). These challenges are listed in Table 1.

Table 1. Cloud migration challenges

Challenge
Security and privacy
Regulation and compliance
Existing investments in IT
Cost management
Portability and interoperability
Service Level Agreements (SLAs)
Technical complexity
IT skill gap

Security and privacy are often considered the biggest challenge in cloud migration, particularly regarding the relocation of sensitive data from on-premises infrastructure to cloud environments. Concerns revolve around verifying the integrity, confidentiality, and availability of data during and after the migration process. (Dar, 2018; Shuaib *et al.*, 2019) This gives the organizations less control over the data and transfers the responsibility of its security to the cloud platform. Consequently, it becomes crucial for organizations to apply robust access control measures and encryption protocols to maintain data security and privacy. (Staevsky and Gaftandzhieva, 2023) Additionally, compliance with legal and regulatory requirements becomes a significant challenge when data is stored in the cloud. This includes concerns over data sovereignty and compliance with standards like General Data Protection Regulation (GDPR), which need to be carefully considered. (Wielki, 2015) Furthermore, the expanding threat landscape and the rising number of significant data breaches involving sensitive information increase security concerns (Dar, 2018).

The context within an organization plays a significant part in shaping the migration process and its considerations. Existing investments in on-premises infrastructure may make it more difficult to move to cloud. This is particularly applicable for large corporations while small and medium-sized enterprises (SMEs) may be more flexible in this aspect. (Rai, Mehfuz and Sahoo, 2014) Cost management also presents its own set of challenges. A report by McKinsey & Company indicates that 75% of cloud migration projects exceed their initial budget and 38% are behind schedule (Balakrishnan *et al.*, 2021). Additionally, when paying for resources per usage it may be difficult to estimate the total costs (Rai, Mehfuz and Sahoo, 2014). Therefore, organizations should conduct a thorough cost analysis before migration and continuously monitor cloud spending to avoid unpleasant surprises.

Shuaib *et al.* (2019) also identifies change management within the organization as another significant challenge. Cloud environments often require different skills compared to traditional IT infrastructure. Organizations might face challenges in retraining or hiring staff with the necessary cloud computing skills, leading to a gap that can slow down or complicate the migration process. (Shuaib *et al.*, 2019) Even though the skill gap can be addressed, technical complexity and adapting the legacy application for the cloud platform is also one of the core challenges (Staevisky and Gaftandzhieva, 2023). Further complicating things, cultural resistance and organizational readiness are often the primary causes for delays in the migration (Balakrishnan *et al.*, 2021).

Dependency on only one cloud provider can result in challenges related to vendor lock-in, making it difficult or expensive to change providers or deploy services across multiple clouds in the future (Kumar and Kumar Garg, 2012). Additionally, organizations must understand Service Level Agreements (SLAs) provided by cloud providers that guarantee certain levels of service. Organizations must understand these SLAs thoroughly to ensure they meet business requirements and to plan for contingencies in case the SLAs are not met. Key aspects in this case are availability, performance, cost, and security (Shuaib *et al.*, 2019). These factors highlight the need to carefully consider which cloud provider to choose.

Adapting applications for cloud infrastructure, which involves enabling scalability, security, and elasticity, presents significant technical and organizational challenges. Additionally, the technological readiness of the organization to execute the migration may require training and new skills of cloud domain to streamline the transition process. These considerations

highlight the complex nature of cloud migration. Consequently, choosing an appropriate migration strategy is important and requires careful planning.

2.2.2 Cloud Migration Strategies

Cloud migration strategies outline different methodologies that organizations can follow as they transition their operations to the cloud. These strategies provide a framework to guide businesses in selecting the most suitable approach for their particular requirements and constraints, considering factors such as cost, security, compliance, and technical compatibility (Andrikopoulos *et al.*, 2013). Understanding these different strategies, organizations are better equipped to navigate the challenges of adopting cloud technologies. This ensures a more smoother transition and optimizes the advantages of cloud computing. (Jamshidi, Ahmad and Pahl, 2013; Rai, Sahoo and Mehfuz, 2015)

Cloud migration strategies have undergone significant development within scholarly discussions. Andrikopoulos *et al.* (2013) identified four primary categories of possible migration paths: replace, partially migrate, migrate the whole application, and cloudify. This categorization has been adopted by many publications in the field (Jamshidi, Ahmad and Pahl, 2013; Gholami *et al.*, 2016). However, more recent studies, by for example Ahmad *et al.* (2018) and Shastry *et al.* (2022), have adopted modifications of Gartner's 5 Rs strategy. This strategy acknowledges the changing nature of cloud computing and the increasing complexity and number of cloud services, allowing organizations to leverage a broader range of options for their specific requirements.

Gartner's strategy has been since expanded and iterated to forms of 6Rs and 7Rs by cloud providers like AWS. The 6Rs and 7Rs include strategies such as 'Retain' for keeping certain applications or systems on-premises, and 'Retire' for decommissioning obsolete or redundant applications. (Shastry *et al.*, 2022) This expansion facilitates a more strategic alignment with business goals, operational efficiency, and cost-effectiveness, offering an extensive framework that addresses the full scope of cloud migration challenges and opportunities. The 7Rs migration strategy classification by AWS (2024c) is presented in Figure 2. A critical aspect that is missing from the 7Rs framework, compared to the classification by Andrikopoulos *et al.* (2013), is the lack of reference to migrating only some parts of the application to the cloud. Some solutions utilize a hybrid cloud approach, where they move

parts of their application to the cloud for example due to scalability. The rest of the application is either migrated in phases or some parts are kept on-premises for security and compliance requirements (Andrikopoulos *et al.*, 2013).

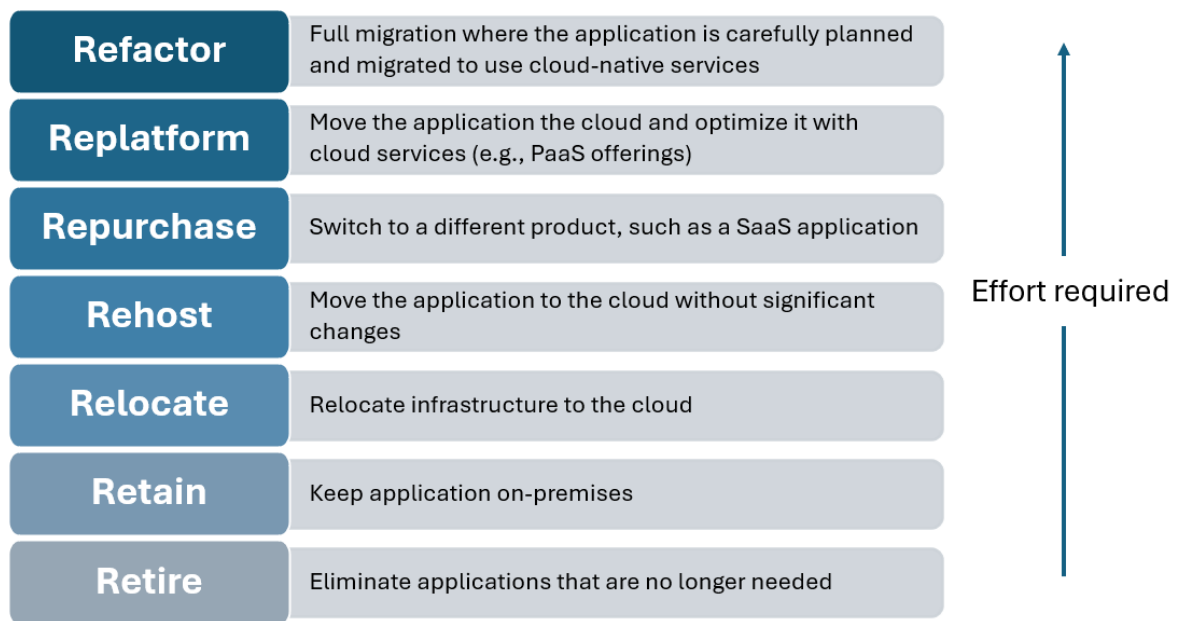


Figure 2. Classification of cloud migration strategies

The repurchase strategy means substituting an on-premises application with a comparable SaaS solution in the cloud, which may also include data migration. This could be for example transitioning an on-premises customer relationship management (CRM) solution to Salesforce, which is a leading SaaS CRM application (Salesforce, 2024). The relocate strategy leverages virtualization to transfer infrastructure to the cloud, streamlining the migration process. Among the various strategies, rehosting, replatforming, and refactoring are the most commonly used (Linthicum, 2017). The rehost strategy, also known as "lift and shift," means transferring virtual machines to the cloud without modifying the application, offering a cost-effective solution relative to other methods (Linthicum, 2017). Additionally, this type of migration incorporates the concept of live migration, allowing virtual machines to be moved to other platforms with minimal to no downtime (Agarwal and Raina, 2012; Strunk, 2012). This is critical in production applications that require uninterrupted operation to meet business demands.

Linthicum (2017) emphasizes that applications that are moved to the cloud should consider either leveraging cloud-native features through refactor or replatform strategies, or to be moved with little changes via the rehost strategy. The term cloud-native has been gaining traction within the technology industry, yet there remains a lack of consensus regarding its precise definition. Kratzke and Quint (2017) provide a clarifying perspective, describing cloud-native applications as those specifically designed and constructed for cloud environments from the ground up. According to their definition, such applications are inherently modular, with each component being independently scalable and distributable. This architectural approach highlights the essence of cloud-native design, emphasizing adaptability and scalability within cloud ecosystems.

This concept extends to migration patterns within the same cloud environment, where an application's architecture is transformed to better align with cloud principles (Linthicum, 2017). Andrikopoulos et al. (2013) support this view, outlining how previous architectures can be re-engineered to increase the utilization of the resources and services provided by the cloud platform. This may involve adopting microservices architecture, using message queues, or replacing IaaS components with PaaS functionality. For example, a Dockerized database instance could be migrated to a fully managed service like AWS RDS, which offers automated updates, backups, and monitoring (AWS, 2024b). This showcases a strategic approach to leveraging cloud capabilities for enhanced efficiency and scalability. Adopting a cloud-native strategy can be financially beneficial in the long term, as it allows for the optimization of resource utilization by ensuring payment only for what is used. However, the trade-off is that portability will be hard to other cloud providers (Miranda *et al.*, 2013). Furthermore, undertaking cloud-native refactoring may incur costs up to 30 times higher than those associated with simple rehosting (Linthicum, 2017).

2.2.3 Cloud Portability

The variety of services and ways they are delivered by cloud providers present substantial challenges in portability and interoperability, which need to be tackled during the development of applications. The complexity of migrating an application from one cloud platform to another is influenced by many aspects, including the specific cloud services utilized, the application's design, and the migration requirements, such as the extent of the

application components needing migration. Given this complexity, accurately estimating the re-engineering efforts required for each application component becomes a considerable challenge. (Miranda *et al.*, 2013) Many studies in the academia have explored the topics of cloud portability and interoperability, highlighting potential vendor lock-in as a significant concern. Linthicum (2017) for example emphasizes that while cloudification can offer numerous benefits there is a trade-off regarding portability, which means that the application is not easily portable to other cloud providers causing possibly a lock-in effect. This concern is also shared by Kolb and Wirtz (2014) who studied the portability of PaaS offerings between cloud providers.

Utilizing open-source and vendor-neutral technologies can enhance portability and reduce dependency on specific cloud providers. Efforts have been made to create standardized technologies to make cloud applications portable. Topology and Orchestration Specification for Cloud Applications (TOSCA) is a standard developed by the OASIS (Organization for the Advancement of Structured Information Standards) consortium. TOSCA aims to improve the ease of transfer and the management of cloud applications and services, facilitating their use across different cloud providers. It defines the processes for creating interoperable descriptions of both the application and infrastructure services in cloud environments, including their relationships, operational behaviours, and lifecycle processes. (Binz *et al.*, 2012) Similar initiatives to TOSCA exist, such as mOSAIC, which received funding from the European Commission to promote cloud application portability (Di Martino *et al.*, 2011).

TOSCA infrastructure refers to the modules and systems that are put in place to support the implementation of TOSCA specifications (Binz *et al.*, 2012). This infrastructure enables the modelling of cloud applications in a vendor-neutral format, which can then be deployed and managed across different cloud platforms without requiring changes (Artač *et al.*, 2017). Wurster *et al.* (2017) also highlight that TOSCA is able to help in porting on-premises applications to the cloud. There also exists TOSCA-based orchestrators like TORCH, which is able to read TOSCA notations and deploy these to different cloud platforms (Tomarchio *et al.*, 2021). Koziolk *et al.* (2023) however mention that TOSCA is missing applications in real-world context and the domain still needs more work and research. In addition, Kolb and Wirtz (2014) discuss that TOSCA is not currently suitable for PaaS migrations but can work for IaaS transitions.

While existing research predominantly addresses the migration of servers from on-premises to cloud environments, the critical aspect of migrating applications between different cloud service providers remains significantly underexplored (Kolb, Lenhard and Wirtz, 2015). Linthicum (2017) also supports this claim by mentioning that only a few cloud-to-cloud transitions have happened. Furthermore, Yussupov et al. (2019) emphasize the limited research available on cloud-to-cloud migration, particularly within the serverless computing context. Additionally, there is no joint agreement in the literature for the definition of cloud-to-cloud migration. Hussein et al. (2013) define the concept of cloud-to-cloud migration as moving infrastructure or applications from one cloud provider to another. Kolb et al. (2015) also appears to take the same stance in their research paper. In contrast, Narantuya et al. (2018) describe cloud-to-cloud migration as moving virtual machines from one cloud provider to another, which closely relates to the concept of live migration. This paper uses the definition outlined by Hussein et al. (2013).

Hussein et al. (2013) focused their study on the security aspect in cloud-to-cloud migration, highlighting it as a significant concern among organizations. A similar emphasis on security issues is noted in the context of transitioning on-premises systems to the cloud (Andrikopoulos *et al.*, 2013). This concern is especially relevant in the context of data migration across cloud providers, as investigated by Pant and Thakur (2013). In the field of data migration, it is important to develop a comprehensive plan, including data security, consistency, and potential downtime (Iqbal and Colomo-Palacios, 2019). Further exploring this theme, Ellison et al. (2018) dived into the specifics of database migration from one cloud environment to another, expanding the discussion on the complexities and challenges faced during cloud-to-cloud migration processes.

Kolb (2019) studied the portability of PaaS between cloud providers, focusing on the migration efforts needed to mitigate the lock-in effect. Through a detailed case study, the research presented the challenges of portability, highlighting that despite high-level component compatibility, low-level implementation discrepancies require code modifications and alternative deployment workflows. Similarly, Yussupov et al. (2019) studied the portability of Function-as-a-Service (FaaS) between different cloud platforms and found that the lock-in effect is in place even in small applications since the components were not compatible. Hartauer et al. (2022) discovered similar findings, however they found the use of a standardized framework and IaC implementation to help with the migration and

portability. In a more proactive approach, Miranda et al. (2013) put effort to make the transition between cloud platforms easier with techniques in how the application is designed and constructed to avoid lock-in effect in cloud platforms.

In an extensive case study, Kolb et al. (2015) assessed the effort required to migrate a real application across popular PaaS providers, such as AWS and Heroku, porting the application to seven different public cloud services. Their findings highlight the necessity for further research into unifying the cloud management interfaces to simplify application management and provisioning across cloud providers, reducing the burden on development efforts. They also emphasized the need for more cloud-to-cloud migration research, especially for real-world case studies, to deepen understanding and develop more effective strategies for tackling these challenges. This thesis aims to bridge these identified gaps by conducting an in-depth case study on cloud-to-cloud application migration and contributes to a clearer understanding of the practical challenges involved.

2.2.4 Migration Frameworks and Models

Due to the complexity of cloud migration and the various strategies, there have been many frameworks developed that aim to facilitate the decision-making process and highlight the concerns and tasks associated with the transition (Sabiri *et al.*, 2015; Hosseini Shirvani, Amin and Babaeikiadehi, 2022). Jamshidi et al. (2013) carried out a systematic literature review on cloud migration and proposed a consolidated process based on previous frameworks with distinct properties and concerns to develop a model for migrating on-premises solutions to cloud. The framework is based on 23 research articles related to cloud migration, including popular frameworks such as CloudStep, Cloud Adoption Toolkit, and CloudMIG. Compared to these frameworks, Jamshidi et al. (2013) paints a big picture of the process and the tasks involved.

The first phase in the model is migration planning. This phase includes steps such as feasibility analysis, cloud vendor selection, requirement analysis, decision on which cloud services should be used, and what to migrate. The final result of this phase is a concrete plan on how the migration will be carried out. In the next step, the legacy application is re-architected for the cloud platform. This involves code modifications to adjust the application to the cloud platform, infrastructure adaptation, and data migration. In the last stage of the

process, the migration to the cloud is executed, tested, and verified prior to being deployed in a production environment. After completing these steps, the final output is evaluated against the original migration plan. Furthermore, tasks such as governance and security analysis are considered throughout all three phases. This migration process is visualized in Figure 3.

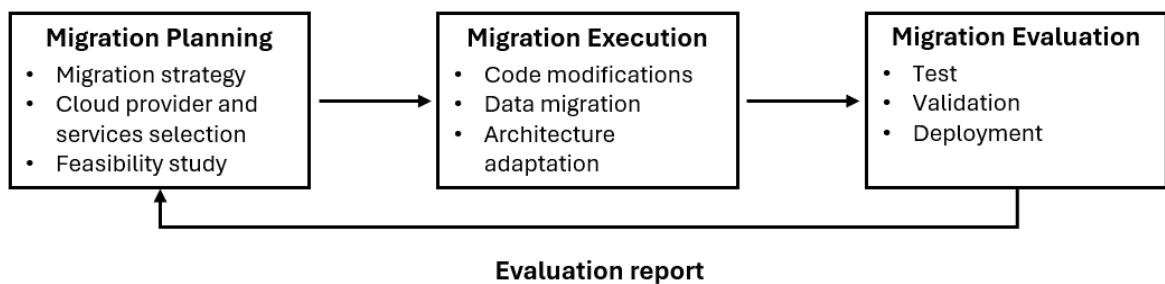


Figure 3. The Cloud-RMM migration framework (Jamshidi, Ahmad and Pahl, 2013)

The authors of this model directly report that the assumption for the migration process is that the legacy application is running on-premise infrastructure and moved to the cloud (Jamshidi, Ahmad and Pahl, 2013). However, while this framework does not exactly consider cloud-to-cloud migration it is arguable that it is still applicable. When exploring cloud migration strategies such as refactor or replatform, which involve transforming an existing application to be cloud-native or more suited for the cloud environment, the procedure includes phases and tasks that are similar to those when migrating from on-premises infrastructure to the cloud. In the planning stage, for instance, the existing architecture is analyzed and compared to the offerings available on the new cloud platform. The execution phase involves similarly data migration and adaptation to the new cloud services. And finally, in the evaluation phase, the application is deployed to the new cloud provider and tested.

2.3 Infrastructure as Code

The transition to cloud computing has significantly simplified the provisioning of computing and data resources, making what was once a complex and costly process accessible even to

small organizations. This transformation has streamlined the ability to scale and enhance operations efficiently. However, the manual provisioning often seen in many cloud environments introduces risks of inconsistency and human error, which Infrastructure as Code (IaC) aims to mitigate by automating these processes.

IaC is a core technique in DevOps and cloud computing, where infrastructure is managed and provisioned through code (Sokolowski, 2022). This replaces the error-prone manual configuration by system administrators with a structured and codified approach (Guerriero *et al.*, 2019). IaC enables DevOps teams to collaboratively automate the management, monitoring, and provisioning of resources in a cloud environment. This approach increases efficiency, reduces manual errors, and enables faster deployment cycle of applications. IaC treats infrastructure as if it were software. Thus organizations can ensure consistency in environments, streamline development and deployment processes, and achieve more scalable, manageable, and replicable infrastructure setups. (Rahman, Mahdavi-Hezaveh and Williams, 2019)

IaC can be implemented using various languages and tools, each with its own strengths and use cases. Popular choices include declarative languages, such as YAML, which is known for its simplicity and readability. Furthermore, programming languages like Python and TypeScript can also be used to define infrastructure for IaC, offering greater flexibility, modularity, and control. (Sokolowski, 2022) According to a study by Rahman *et al.* (2019), Chef and Puppet were found to be most frequently used IaC tools. Some other popular examples include Terraform, which supports IaC for various cloud providers and on-premises infrastructure. (HashiCorp, 2024) Figure 4 shows an example of a Terraform template to create a virtual machine running on Debian operating system inside Google Cloud. In addition, AWS provides their own tool called CloudFormation, which is written with YAML or JSON to create and manage resources on their platform. Furthermore, AWS offers the Cloud Development Kit (CDK), which builds on CloudFormation's capabilities, allowing infrastructure to be defined in programming languages like Python or TypeScript before being deployed to AWS's cloud environment. (AWS, 2024a)

```
resource "google_compute_instance" "vm_instance" {  
  name = "vm-instance"  
  machine_type = "f1-micro"  
  initial_node_count = 1  
  boot_disk {  
    initialize_params {  
      image = "debian-cloud/debian-9"  
    }  
  }  
  network_interface {  
    network = "default"  
    access_config {  
      // Ephemeral IP  
    }  
  }  
}
```

Figure 4. Terraform template to create a virtual machine in Google Cloud

Rahman et al. (2019) conducted a systematic literature review of the current research into IaC to identify the main themes and research gaps. Their analysis found that a considerable amount of research focused on the various frameworks and tools used in IaC, along with how these tools are applied in practice. The study also covered empirical analysis and testing related to IaC. Furthermore, Rahman et al. (2019) also pointed out several areas that require more research, highlighting a significant gap in research regarding industry best practices, security concerns, and the presence of anti-patterns in IaC practices. However, based on the extensiveness of anti-pattern literature (Rahman, 2018; Guerriero *et al.*, 2019; Dalla Palma *et al.*, 2020; Kumara *et al.*, 2021) this seems to have gained interest in the academia to support the successful adoption of IaC. Similarly to this study, Nedeltcheva et al. (2023) focused on reporting their challenges and experiences when applying IaC to a cloud-native application.

Infrastructure as Code has been fundamental in enabling the DevOps practice of continuous delivery and integration, supporting faster deployment and management of software and services (Kumara *et al.*, 2021). It has gained widespread adoption among large corporations like GitHub, Google, Netflix, Facebook, and Mozilla due to its advantages. Its popularity is

further evidenced by the increasing interest in IaC on search engines and within the academic community. (Rahman, Mahdavi-Hezaveh and Williams, 2019) However, adopting IaC requires careful management to avoid potential issues like defects and security flaws. This highlights the importance of adhering to the best practices in the development and maintenance of IaC scripts (Guerriero *et al.*, 2019).

2.3.1 Best Practices

Guerriero *et al.* (2019) surveyed 44 people in the industry about the challenges, best practices, and adoption of IaC. They found multiple instances of good habits, such as making the infrastructure break as fast as possible when deploying and making reusable modules and templates. Kumara *et al.* (2021) expanded on this research by conducting a systematic review of grey literature to draw and categorize the best practices and anti-patterns of IaC. They found 10 primary categories for best practices (see Table 2), which includes factors in both the implementation and design aspects of IaC. These best practices highlight the need for careful design in organizing scripts to modules and folder-level structures. Additionally, they outline the significance of following styling conventions and ensuring readability, which reinforces the need for a thoughtful and well-organized approach to IaC deployment and management.

Table 2. Best practices in IaC, adapted from (Kumara *et al.*, 2021)

Practice	Explanation
Focus on readability and understandability	Simple, unified formatting with meaningful, distinctive names
Avoid duplication	Modularize and reuse
Work with the IaC principles	Codify everything so that manual changes are not needed
Make incremental changes	Using version control
Prevent avoidable mistakes	Correct quoting styles and proper values
Plan for unavoidable mistakes	Writing tests and not ignoring errors
Use concise documentation	Use code as documentation

Well-structured repository	Use standardized, meaningful folder structures
Keep configuration data separate from code	Use configuration data sources and templates
Safeguard configuration data and code securely	Isolating secrets from code and using secure coding practices

Furthermore, Kumara et al. (2021) identified 4 high-level categories for bad practices (see Table 3). These issues were mostly concerned about breaking the core principles of IaC, such as using manual procedures alongside with IaC scripts and creating non-reproducible environments. Most importantly, they highlight the need to comply with the framework and utilize its functionalities when writing IaC. Another important factor is to use good naming conventions and make the scripts readable. These practices reflect principles that are also valued in traditional programming. Guerriero et al. (2019) also reported similar findings, with hardcoding found to be most commonly mentioned issue.

Table 3. Bad practices in IaC, adapted from (Kumara *et al.*, 2021)

Practice	Explanation
Violation of IaC principles	Working against the principles, e.g., not automating everything, non-reproducible environments
Bad readability and understandability	High complexity, poor modularity, not following styling convention
Improper project organization	Not taking advantage of version control or folder structure is not sound
Insecure configuration data and coding practices	Hardcoding or using insecure secret variables

The research by Guerriero et al. (2019) and Kumara et al. (2021) highlights the importance of adopting best practices, such as modularity and readability, and avoiding pitfalls like manual processes and hardcoding in IaC. These principles not only enhance infrastructure

management's efficiency and reliability but also reflect the broader practices of software development. This research offers a roadmap for optimizing IaC deployment, underscoring the importance of a well-organized strategy for future advancements in the field.

2.3.2 Benefits and Challenges

IaC offers a solution to several challenges compared to manual infrastructure management. One of these solutions is the avoidance of configuration drift. This is a common problem when environments or resources that are intended to be identical, deviate over time due to ad hoc manual changes or oversight. Configuration drift complicates system management and also reduces stability and deployment speed, especially in complex systems involving numerous microservices and modules. (Lwakatare *et al.*, 2019) With IaC organizations can ensure that their infrastructure is consistently deployed and maintained across all environments, mitigating risks associated with manual interventions (Novakova Nedeltcheva *et al.*, 2022). This consistency and repeatability can be beneficial in setting up different development, staging, and production environments with similar capabilities (Morris, 2016).

Moreover, IaC supports best practices such as defining everything as code, continuous testing and delivery, and building in small, manageable parts (Morris, 2016). These practices contribute to several key benefits including reusability, consistency, transparency, and cost optimization over time (Guerriero *et al.*, 2019). Codifying infrastructure provides teams with clearer visibility into system configurations and changes, facilitating easier debugging, knowledge sharing, and collaborative development. Furthermore, the code itself acts as documentation as pointed out by Werner *et al.* (2022). In a similar way, since infrastructure is committed to a code repository, it also provides versioning out of the box, and the ability to track the changes across different versions (Morris, 2016).

The modular approach of IaC simplifies understanding and updating of the system. Additionally, it enhances the potential for reuse in future projects with modules and templates, which reduces complexity and deployment times (Morris, 2016). Furthermore, IaC's built-in automation features enable dynamic scaling and effective resource management, which leads to more reliable and frequent updates, therefore improving product quality and delivering greater value to customers (Forrester, 2015).

While these benefits bring value for organizations there are also some challenges associated with IaC and its adoption. Some of these challenges vary based on the tools utilized, while others are more generally encountered. One significant concern is the resistance to change, especially within organizations accustomed to traditional practices (Forrester, 2015). Fear of disrupting existing, functioning systems and the trade-offs between speed and code quality present substantial obstacles. Prioritizing speed can lead to poor code while an excessive focus on quality may result in outdated systems slowed down by lengthy procedures. (Morris, 2016) The transition to IaC also requires a shift in work practices towards adopting DevOps. This change is not only technical but also cultural, which requires a re-evaluation of priorities and processes within organizations. Additionally, IaC prompts for new skills needed in the organization to adopt the technology. (Forrester, 2015; Guerriero *et al.*, 2019) According to Morris (2016) the training can be a time-consuming process due to the amount of available IaC tools and solutions, which can make it difficult to clearly understand the best practices and adoption techniques.

Furthermore, the practice of IaC itself presents its own set of challenges, including issues related to code readability, stability, security, organization, testability, and monitoring. Guerriero *et al.* (2019) interviewed people in the industry about the challenges associated with IaC. Their findings highlighted that IaC causes longer feedback loops due to having to wait for the infrastructure to be deployed, which complicates testing. Additionally, circular dependencies, which means that two or more resources depend on each other, were a challenge since they caused race conditions during infrastructure provisioning. These factors contributed to the poor testability and debugging of IaC. Furthermore, portability also becomes a problem in IaC since the infrastructure is defined for a specific cloud platform, making it troublesome to move to other vendors. (Guerriero *et al.*, 2019)

Security also emerges as a critical concern, with IaC scripts being vulnerable to defects that could lead to significant breaches, underscoring the need for secure coding practices from the start (Rahman, 2018). Some reported anti-patterns that reduce security are hardcoding secrets in the code and committing them into version control, which makes these secrets available for anyone with access to the repository and deployment pipeline. Furthermore, IaC increases the chance of forgetting about access control measures and firewall rules during the infrastructure setup. (Rahman, 2018)

Benefits	Challenges
<ul style="list-style-type: none">• Consistency• Prevents infrastructure drift• Faster deployments• Reduces human error• Cost optimization over time• Versioning• Code as documentation• Reduces complexity• Reuse and modularization	<ul style="list-style-type: none">• Longer feedback loop• Circular dependencies• Race conditions• Portability• Poor testability and debugging• Skill gap and change management• Security

Figure 5. Benefits and Challenges of IaC

To conclude, while IaC offers significant advantages for automating and managing infrastructure, it also presents a range of challenges from technical to organizational and security related issues. These insights are summarized in Figure 5. And although IaC has emerged as a critical practice for modern infrastructure management there is still little knowledge on how to maintain or create a strategy for its adoption (Guerriero *et al.*, 2019). Moreover, Guerriero *et al.* (2019) also highlight that there is limited academia surrounding IaC, highlighting the need for further research.

3 Research Methodology

This chapter outlines the research methodology, providing insight into the reasoning for its selection and how it aims to answer the research problem. It also introduces the case organization and its migration project, offering a detailed examination of the system's existing architecture and operational environment. Following this, the chapter focuses into identifying the motivations and challenges within the current environment, setting the background for further exploration of the migration process.

3.1 Research Method

The research adopts an exploratory case study methodology, focusing on a single case of cloud-to-cloud migration. This method is chosen due to its strength in providing in-depth and detailed insights into “contemporary and complex phenomena within their real-life contexts” as described by Yin (2009). The case study method allows for an exploration of the cloud migration process in its natural setting, enhancing the understanding of the dynamic processes involved. Additionally, Runeson and Höst (2009) emphasize that the case study research is particularly suitable for software engineering research, which requires an understanding of software development and operational processes in practice. This study was set to answer the following questions:

- What are the key challenges, strategies, and outcomes involved in migrating an application between cloud providers?
- What are the benefits and challenges of using IaC in the migration process?

This research will apply the Cloud-RMM migration framework presented in Chapter 2.2.4 to conduct the migration and report the findings. The data is collected through direct observations made by the researcher, who actively participated in the migration project. These findings will be integrated with the related research presented in Chapter 2, applying theoretical best practices for IaC, and addressing the identified challenges and considerations

of cloud migration. The migration planning process will be outlined in Chapter 4, followed by a detailed presentation of the migration results in Chapter 5.

The limitations of the case study approach will be acknowledged. Generalizability of findings may be limited due to the focus on a single case, impacting external validity (Runeson and Höst, 2009). However, the research will aim to contribute to the broader understanding of cloud migration between providers by providing a rich and detailed case study with in-depth insights. Furthermore, there is a high risk of observer bias as the findings are based on the reporting of a sole researcher.

3.2 Case Organization

The case organization is a self-service mobile payment platform start-up located in Finland. The company was founded in December 2023 and currently has 8 employees. Since its foundation, the company has released solutions for retail and restaurant industries. In the retail sector, it introduces a user-friendly Scan & Go model, enabling customers to effortlessly scan and purchase items using their own phones, thereby revolutionizing the shopping experience with its mobile application. Additionally, the company has released a similar service for student restaurants. In Finland, higher education students are required to show their student card to be eligible for a student discounted price. The case company has released an innovative solution within their mobile application, designed specifically for students. This feature enables users to seamlessly link their student status to the app. Once connected, students gain the ability to purchase meals at a discounted rate. At checkout, customers present a QR code from their receipt to a payment terminal at the cashier. This streamlined process not only facilitates savings for students but also enhances the overall dining experience with convenience and efficiency.

The development team for the company consists of three people. The author of this paper is working as the Lead Developer in the case company with 9 years of experience, and the other two are hired consultants outside of the company. The small team and relatively recent and compact project enable great flexibility and agility to even big transformations. To streamline their development process, the team employs the Kanban methodology, ensuring efficient workflow and continuous improvement. The team has no previous experience with AWS environment nor IaC technologies.

3.3 Case Description and Motivation

The current environment for the application is running on a single DigitalOcean droplet, which is a virtual machine operating on Ubuntu version 20.04. The lifecycle of the application is managed with Docker Compose, which is a tool that allows for easy definition and operation of multi-container applications, streamlining development and deployment processes. It enables efficient management of services, networks, and volumes through a single YAML file, allowing for the creation and initiation of services with one command. Docker Compose offers comprehensive lifecycle management through commands for starting, stopping, rebuilding services, monitoring their status, streaming logs, and executing one-off commands on services. (Docker, 2024) Within the application's architecture, each service is encapsulated within its own Docker container. The architecture consists of a backend, PostgreSQL database, Redis cache, a reverse proxy, web dashboard interface, a mobile application, and a database management service.

The backbone of the application is the backend, facilitating communication with user-facing applications like the mobile application and web dashboard service. The backend service consists of a payment service, recurring tasks, and offers application programming interfaces (APIs) for various functions, such as shopping cart management, order processing, store management, and product listings. The backend service is a containerized Node.js application built with TypeScript and Express.js.

In addition, the dashboard web application, developed with Flutter and hosted on an NGINX web server, is tailored to meet the operational requirements of restaurant staff and the organizational staff. This dashboard plays a key role in the management of store operations, product inventories, sales analytics, and financial accounting. The customer-oriented mobile app, also built with Flutter, interfaces with the backend via APIs to facilitate a variety of user actions within the application ecosystem.

The application's data persistence is provided by a PostgreSQL database, placed within a Docker container and utilizing default configurations. Access to this database is restricted to the internal network, which is safeguarded by firewall configuration. For database administration, the popular open-source platform pgAdmin is utilized. It offers a comprehensive user interface for efficient database management, SQL query execution,

database structure visualization, data viewing, and user and permission management. (pgAdmin, 2024)

Furthermore, a NGINX reverse proxy acts an intermediary service that forwards traffic to the intended services. This means that any other services than the reverse proxy cannot be accessed directly as they are blocked by firewall rules, thereby maintaining a secure operational environment. Instead, clients request a specific service from the reverse proxy, which redirects the request to the correct service. The reverse proxy also handles the domain routing for the configured services. These public-facing services with domains are the backend service, dashboard interface, and the pgAdmin management interface.

Lastly, a Redis service is incorporated for specific functionalities like session management, caching, and job queuing, leveraging its capabilities as a high-performance, open-source in-memory datastore (Eddelbuettel, 2022). Redis ensures quick, distributed access to data, with the resilience to maintain data even through backend updates. The comprehensive architecture and inter-service communication within the current deployment environment are illustrated in Figure 6.

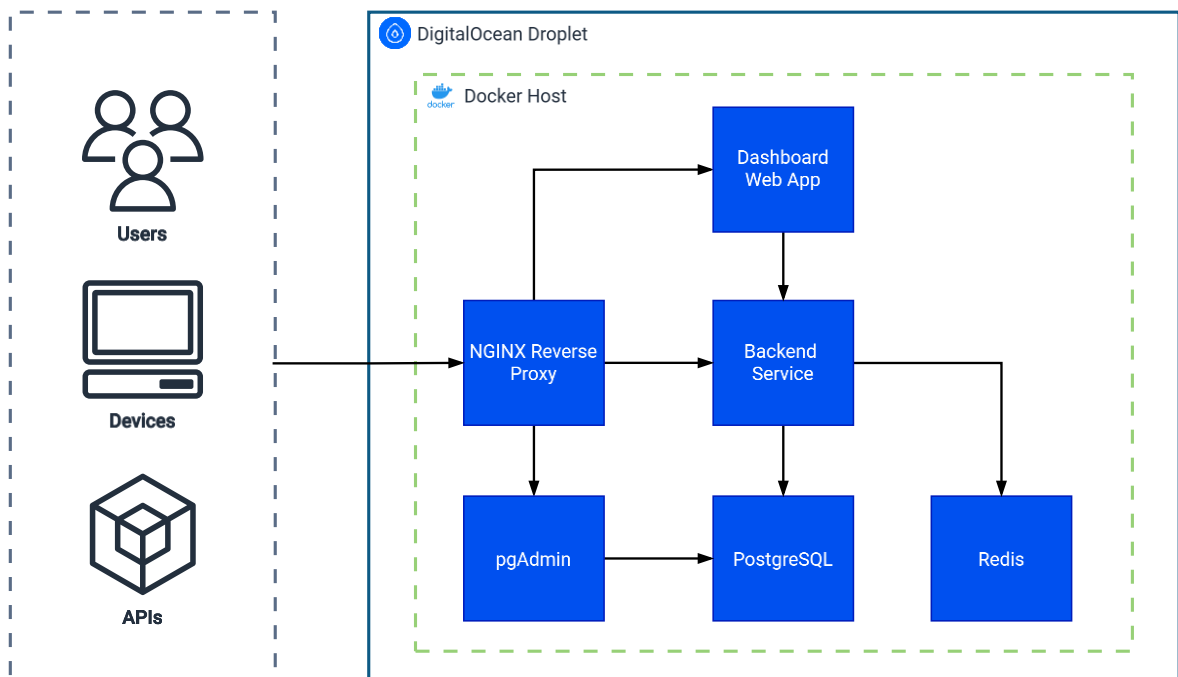


Figure 6. Current architecture in DigitalOcean

Overall, the existing architecture is not too complex, which makes it easy to manage and migrate to another cloud platform. The primary driver for changing the cloud provider to another is based on external business factors, and not things like pricing, performance, or even product offerings. The selected cloud provider for migration is AWS, which is one of the most popular cloud computing platforms. However, there are also a few pain points, which are identified with the current implementation that should be also resolved during the migration process. These challenges are shown in Table 1 for a comprehensive overview.

One of the primary concerns with the current configuration is the extensive manual oversight required for managing and updating the infrastructure. Such operations involve SSH access to the server, utilizing terminal commands for navigation, executing updates, and integrating the latest changes from version control systems like git. This approach carries a high risk of errors and inefficiencies, resulting from the cumbersome nature of manual command execution and the inherent risk of human error. A more streamlined approach would apply these updates automatically without manual intervention. Furthermore, the current update process introduces service interruptions during restarts, necessitating the scheduling of updates during periods of minimal usage to mitigate impact on service availability.

Moreover, the presence of a separate development environment on the same server, mirroring the production setup, introduces additional challenges. Manual management of these environments increases the risk of configuration drift, meaning disparities between the production and development settings. This risk is further emphasized by the method of updating configurations and secrets, which relies on direct manipulation of `.env` (dotenv) files through terminal-based text editors. This practice allows developers to separate configuration options from application code, enabling easier management of environment-specific settings (e.g., database credentials and API keys) without hardcoding them into the source code. However, such practices are labour-intensive and prone to inconsistencies.

Security configurations and the process of implementing updates also present significant challenges. Although considerable effort has been invested in establishing robust firewall rules to avoid unauthorized access, the evolving nature of security threats necessitates a more agile and responsive approach. Transitioning critical security updates to a third-party service and simplifying the management of security controls are suggested measures to improve the overall security posture of the infrastructure and the applications. Additionally, the current

implementation has failed to consider scalability and load balancing, which should be also addressed to handle increasing customer demand.

Table 4. Issues in the current environment

#	Issue
P1	Updates produce downtime while the services are restarting
P2	No automated provision of version updates
P3	Manual management and control over the infrastructure is prone to errors
P4	No load balancing or scalability available
P5	Configuration drift between production and development environments
P6	Secrets are managed in environment specific files, which are hard to manage and error-prone
P7	Security configuration and management are insufficient
P8	Monitoring is difficult
P9	Checking service logs is cumbersome since they must be accessed through SSH and terminal

When migrating the whole existing architecture to AWS the mentioned challenges should be also considered when planning the overall architecture. Also, due to this business requirement, the infrastructure cannot be moved as is, but requires careful consideration in the architecture to tackle these challenges. Additionally, as dictated by business requirements, the implementation of the infrastructure will leverage IaC to ensure efficiency and consistency. Furthermore, compliance with GDPR must be carefully integrated and maintained throughout the migration to ensure adherence to data protection regulations.

4 Migration Planning

This chapter develops a migration plan and strategy to move to the new cloud platform in the case organization. The migration plan addresses the presented problems and business constraints set in Chapter 3 and sets a strategy approach overview. An application architecture in the new cloud platform is presented. Furthermore, the DevOps pipelines to complement the IaC implementation are presented.

4.1 Migration Strategy Selection

For the migration to AWS, two primary strategies are evaluated: replatforming and refactoring. Replatform means selective improvement of the application through the integration of cloud-specific features without changing its fundamental architecture. This focuses on optimizing performance and maintenance by utilizing cloud capabilities, such as auto-scaling, managed storage and databases, and monitoring. (Linthicum, 2017) This approach aims to leverage the immediate benefits of the cloud's scalability and managed services without requiring a fundamental overhaul of the existing application infrastructure.

In contrast, the refactor strategy would involve making the application architecture cloud-native. Cloud-native means that the application is designed for the cloud from the start, leveraging its scalability and resilience (Kratzke and Quint, 2017). Although this approach offers potential long-term operational and performance benefits, it requires a significant commitment of resources, time, and expertise to re-engineer the application from the ground up (Linthicum, 2017).

However, the transition towards a cloud-native architecture, despite its potential benefits, is not currently favourable due to the extensive effort required. Consequently, the replatform strategy is chosen to proceed with, which is both cost-efficient and does not require major development work. Cloud features will be utilized as necessary to address the identified challenges, focusing on streamlining and simplifying management and maintenance tasks. While it is also technically feasible to migrate the infrastructure to AWS with minimal adjustments, this path is not preferred since it would not solve the previously discussed challenges relating to maintenance. AWS offers services, such as Amazon Elastic Compute

Cloud (EC2), which supports running the existing application on virtual machines using Docker images, aligning with the rehost migration strategy.

4.2 Selection of AWS Services and Resources

This section presents the chosen AWS services designed to replicate the functionality of the previous architecture while also addressing the previously encountered challenges. The service mapping is shown in Figure 7, where the existing architecture components are highlighted in blue. These are then connected to a service or services in AWS platform.

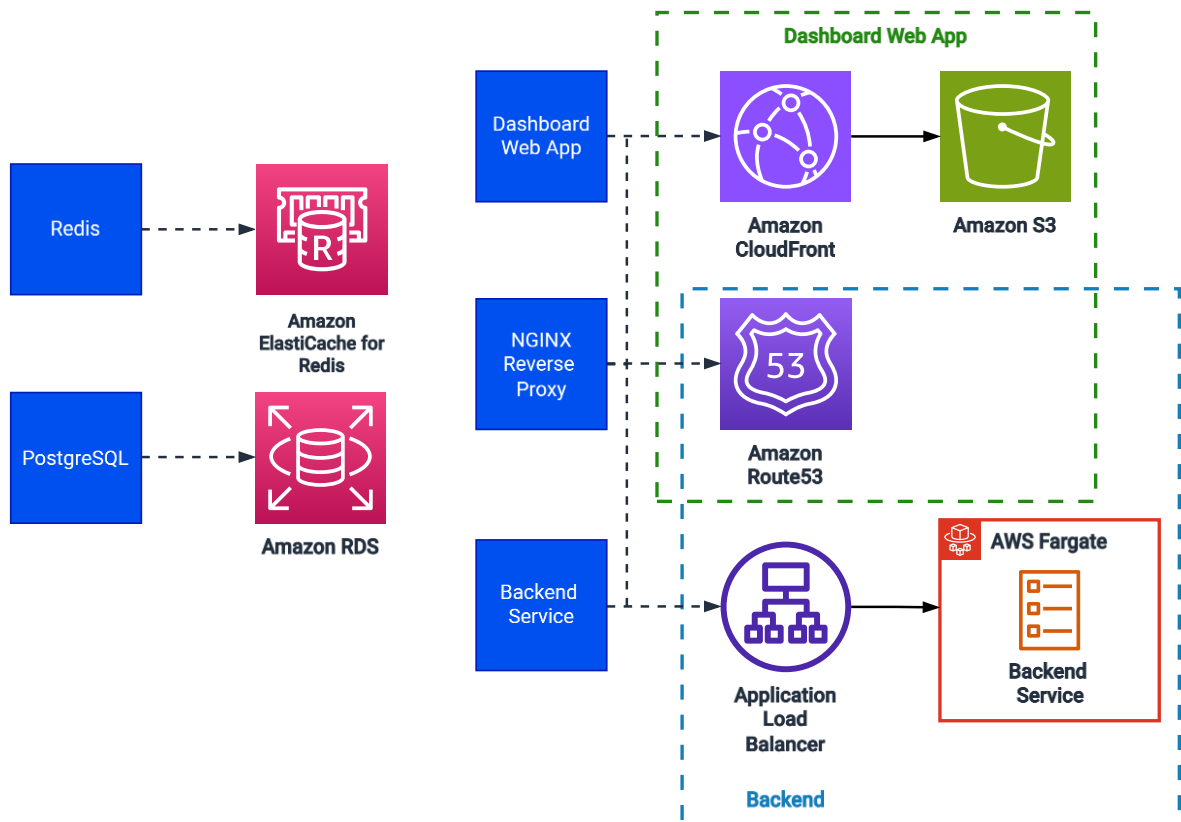


Figure 7. Mapping of cloud services to new platform

The current Redis instance is replaced with AWS's fully managed Amazon ElastiCache service for Redis. This managed caching service provides high availability, security, and compliance out of the box. Furthermore, the bare docker container running PostgreSQL is replaced with an RDS instance, which is AWS managed relational database service. This

offers simplicity and ease of management, such as automatic updates and patches, certificates, backups, data encryption, high availability, and security. These two services can be thus replaced with a drop-in replacement without changes needed in other parts of the application except changing the connection values to reflect the new services. However, when migrating the database, previous data needs to be moved to the new instance. The existing cache in Redis will not be migrated. Furthermore, the pgAdmin service will not be migrated to AWS due to security concerns and added configuration complexity. Instead, database management will be conducted from a local machine to AWS with carefully set firewall rules and access controls.

The previous NGINX reverse proxy is completely replaced with AWS's cloud offerings. The dashboard web application and backend service both require a domain name for easy access over the internet. For this purpose, Amazon Route53 will be used to provide a reliable way to provide domain name system (DNS) routing for these services. Route 53 will redirect traffic to the appropriate services, ensuring smooth integration with the broader suite of AWS services. This setup simplifies management and streamlines the process of securing SSL certificates, enhancing overall security and reliability.

The dashboard web application utilizes the AWS CloudFront content delivery network (CDN), which comprises a network of servers distributed across various locations. These servers cache content in proximity to end users, allowing content to be delivered from servers closest to users. This enhances the speed and performance of the website. Furthermore, the dashboard application content is placed on Amazon Simple Storage Service (S3), which is a scalable data storage service. The build artifact of the Flutter application is uploaded to S3 and served over the CloudFront service to end-users.

The backend service will be placed in Amazon Fargate, which is a serverless compute engine to run containerized applications. Fargate enables the deployment of containers without having to manage the underlying infrastructure (e.g., updates and configuration). This makes it easier and faster to develop and deploy applications. Furthermore, Fargate can be easily configured to scale without much configuration. It also operates on pay-per-use model, where charges are incurred only for the containers actively utilized. (AWS, 2024d) Fargate was selected over Amazon ECS for the easier and simplified management, offering solutions to both issues **P1** and **P4**.

Fargate directly integrates with AWS managed Application Load Balancer (ALB), which forwards the traffic between containers and handles load balancing. Together these services can be configured to automatically scale and manage revisions of new deployments without service downtime. A load balancer distributes incoming traffic across multiple targets, such as containers, ensuring high availability and scalability for applications. However, the backend service needs code modifications to support horizontal scaling since there can be multiple containers deployed simultaneously, necessitating that state management is distributed. Horizontal scaling means that multiple instances of the application are run concurrently to handle increased load. This requires distributing the workload across these instances efficiently. In the context of the backend service, horizontal scaling involves deploying multiple containers or instances of the service to serve a growing volume of requests. Additionally, Amazon Elastic Container Registry (ECR) will be used to upload the Docker container images and handle versioning.

The overall planned architecture in AWS is visualized Figure 8. The Fargate service is placed inside of a private subnet with egress, meaning that it can access the internet, which is required to call third-party APIs like contacting the payment service. This requires using a Network Address Translation (NAT) service, which allows the service to contact outside network while it is placed in a private network. Additionally, RDS and ElastiCache will be placed in private network, meaning that they cannot be accessed from outside of the virtual private network, which provides additional security.

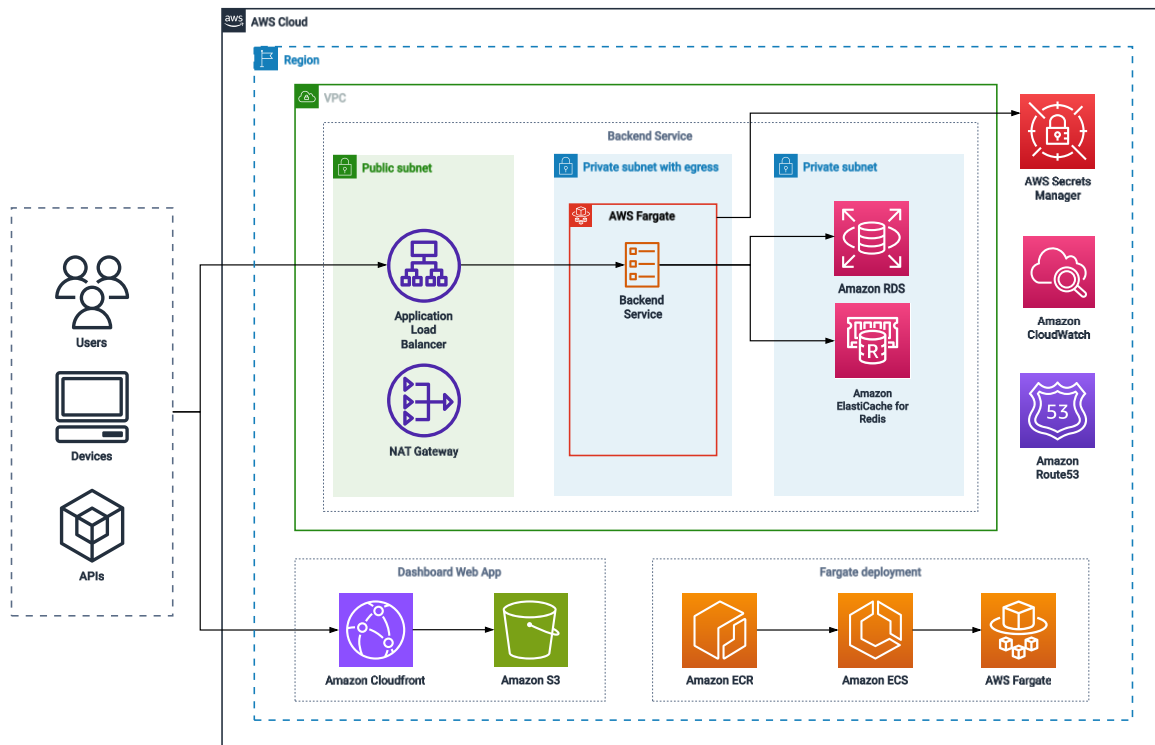


Figure 8. Planned architecture in AWS

AWS also provides a monitoring and observability service named Amazon CloudWatch. CloudWatch tracks the health and performance of cloud resources deployed on AWS. It collects metrics on how resources are performing (e.g., CPU and memory usage) and generated logs to facilitate troubleshooting and identifying errors. CloudWatch can be easily accessed in AWS dashboard where metrics can be visualized for usability. For example, performance monitoring is provided for RDS, ElastiCache, and Fargate services. Consequently, the adoption of CloudWatch addresses both issues **P8** and **P9**.

The management and protection of environment secrets and configurations will be centralized within AWS Secrets Manager, ensuring the secure handling of critical information. These secrets can be things such as database credentials, API keys, OAuth tokens. AWS Secrets Manager offers a robust solution for the centralized control and access of secrets, significantly mitigating the risk of inadvertent exposure in source code. This strategic approach directly addresses concern **P6** by safeguarding sensitive data through enhanced security practices and streamlined secret management processes.

Overall, the integration of AWS managed resources with built-in security features, private networking, data encryption, and Secrets Manager collectively increases security measures, directly addressing the mentioned **P7** issue. Furthermore, the infrastructure will be strategically deployed in the Stockholm region (codenamed eu-north-1), aligning with GDPR requirements. This location is also chosen for its geographical proximity to Finland, where the case organization operates, to optimize performance.

4.3 IaC Utilization

The described infrastructure will be fully constructed using IaC to automate both management and deployment of the infrastructure through code. Among the various IaC tools available, such as AWS CloudFormation, AWS Cloud Development Kit (CDK), Terraform, Ansible, Chef, and Pulumi, CDK was selected for the implementation. This decision was influenced by CDK's compatibility with TypeScript, allowing seamless integration with the existing codebase also written in TypeScript. Furthermore, since CDK is provided by AWS it provides seamless integrations with services in the ecosystem. This integration is facilitated by CDK's high-level constructs specifically designed for AWS services, including Fargate, Route53, Secrets Manager, and CloudFront. These constructs simplify the deployment and management of infrastructure, ensuring a more efficient and cohesive development process. Although opting for CDK introduces a potential dependency on AWS, posing a risk of vendor lock-in, this risk is considered manageable for the project. The IaC implementation will be developed with the best practices noted in Chapter 2.3.1.

Given the fundamental role of IaC in DevOps practices, the development of CI/CD pipelines is also planned. This strategy directly tackles the challenges identified as **P2** and **P3**, ensuring a streamlined process for continuous integration and delivery. Furthermore, in theory this should avoid infrastructure drift between development and production environments as identified by Morris (2016), addressing the concern raised in **P5**.

One pipeline will be developed per each application: backend and dashboard. Both applications are hosted on GitHub each in their own repository. The pipelines will be configured using GitHub Actions, which allows automating the build, test, and deployment pipeline directly within a GitHub repository. For the development environments the pipeline is triggered when new commits are pushed to the repository. However, for added stability,

the same setup for the production environment is only triggered when a new release is created.

When the pipeline is triggered, the process begins with the execution of continuous integration tasks. This includes building the application and then conducting unit testing to verify the functionality of the code. If this step is successful, the infrastructure will be deployed to AWS. CDK automatically checks if there are any changes in the code and proceeds accordingly with the infrastructure provision. In the backend deployment, this setup continues to handle database migrations, which are deployed to reflect changes in database schema. A docker image is then built and pushed to Amazon ECR for versioning. Following this, the new container image is deployed to ECS which starts a new container deployment. Upon the successful completion of this deployment, the previous deployment is terminated. This seamless transition ensures that there is no service interruption when the deployments are exchanged. This process is visualized in Figure 9.

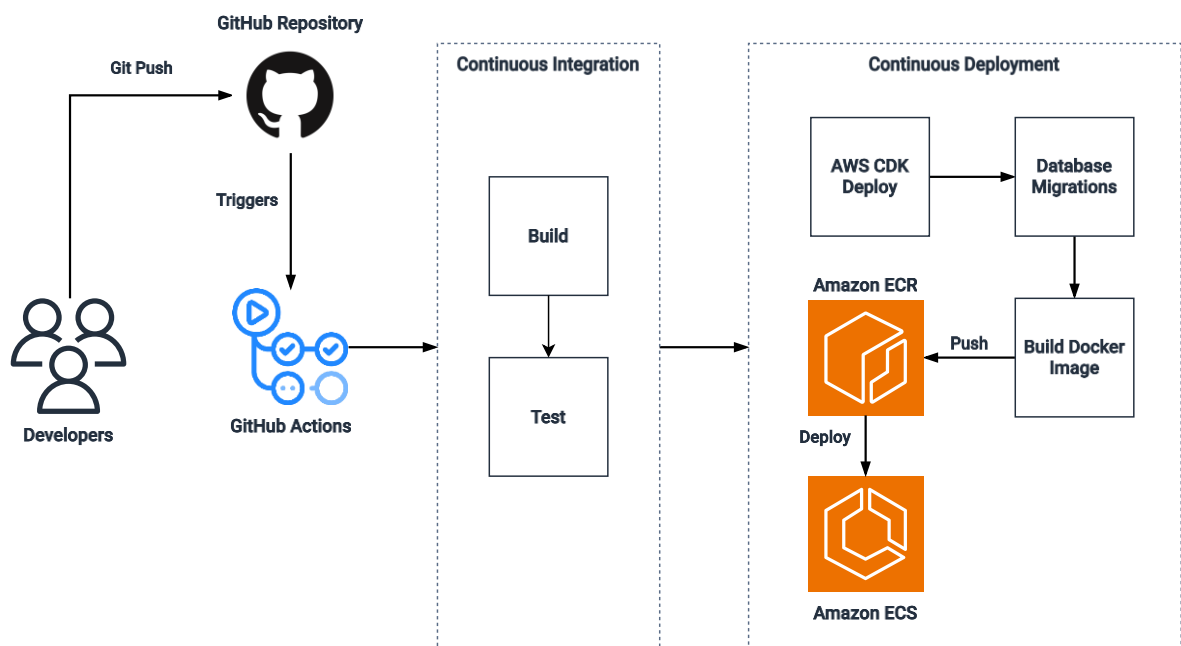


Figure 9. DevOps pipeline for the Backend service

For the dashboard pipeline, the continuous deployment looks a little different, which can be seen in Figure 10. After the infrastructure has been deployed, the new build artifact is uploaded to S3 storage. Subsequently, a request is made to Amazon CloudFront to invalidate

its cache. This step ensures that CloudFront recognizes the new contents and is able to serve these to end-users.

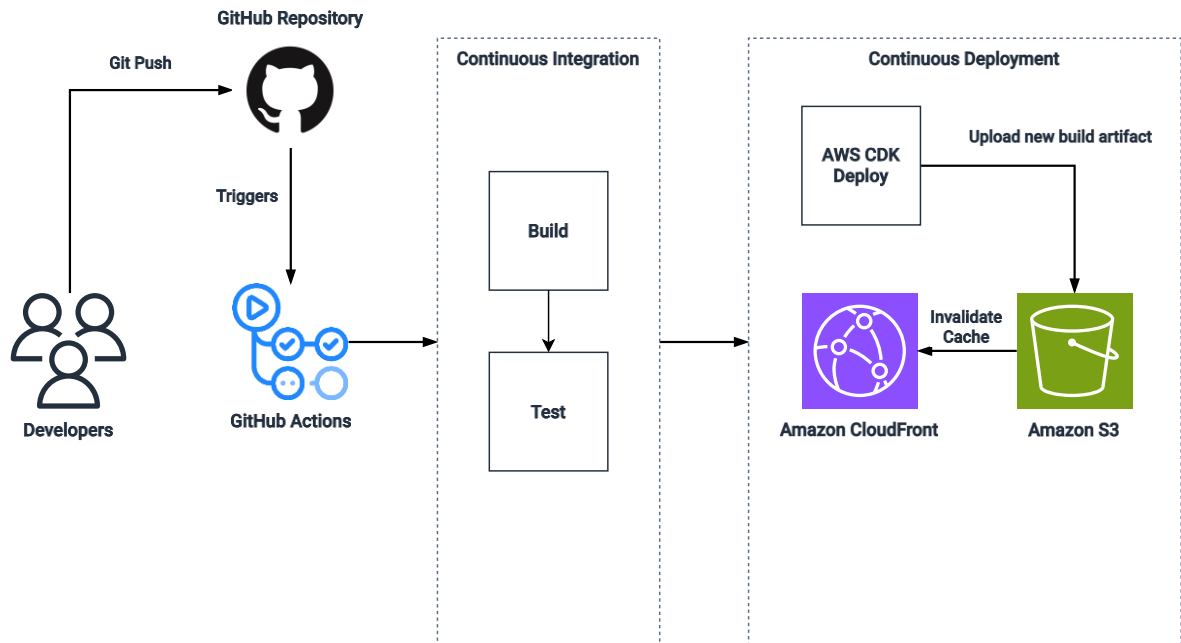


Figure 10. DevOps pipeline for the Dashboard web application

These pipelines lead to significant improvements in managing infrastructure changes and service updates, aligning with DevOps principles. They promote an automated and efficient process towards continuous deployment. As a result, both backend service and dashboard web application can receive timely updates and maintenance. Additionally, the IaC implementation presents an agile environment, which can quickly adapt to changes.

4.4 Migration Plan

The migration plan outlined in this section is carefully designed to ensure an efficient and minimally disruptive transition to the new cloud provider. It highlights the key phases in the transition process, including previous service termination and data migration. The migration is scheduled for after 4pm. This is when the app is least used because restaurants and stores,

its main users, are usually closed, which ensures minimal disruption and service availability. The complete migration process is visualized in Figure 11.

The initial phase of the migration involves the termination of the old service running on DigitalOcean. This decisive action is important to prevent any further data entry into the old system, effectively preventing the risk of data loss during the migration. Furthermore, stakeholder communication is also important at this stage to ensure that all customers are prepared for a temporary suspension of services to avoid operational disruption.



Figure 11. Migration process

Subsequent to the old service termination, a snapshot of the existing database is created. This method captures a full snapshot of the data at a specific moment, playing a key role in maintaining data integrity during the migration. Acting as a protective measure, this snapshot facilitates the recovery of data if there be any issues such as migration errors or data corruption. PostgreSQL provides natively a helpful ‘`pg_dump`’ command to create a comprehensive backup of the database.

The migration process advances to an essential phase of updating the DNS settings. This step plays a key role in redirecting network traffic from the old system towards the newly established infrastructure, thus effectively forwarding user access to the new service. Typically, DNS change (propagation) is expected to complete within a few hours, transitioning users smoothly to the new service environment without significant delay.

Following the DNS update, the deployment of the new infrastructure is performed by creating a new release in GitHub. This triggers the DevOps pipeline, which is in charge of deploying the infrastructure. After the infrastructure provision has completed, the restoration of the database snapshot onto the new infrastructure is performed. The data migration can be done with the ‘`pg_restore`’ command, which allows for schema and data restoration to the PostgreSQL database. This essential phase ensures that application data is preserved, ensuring uninterrupted service operation.

The final phase of the migration strategy involves a thorough testing and validation process for the newly implemented system. This in-depth evaluation aims to verify the operational integrity and performance efficiency of the updated infrastructure. Furthermore, it includes a detailed verification of the DNS change, confirming that end-users experience seamless and accurate access to services. This stage of extensive testing and validation is crucial, offering definitive confirmation that the migration has been executed successfully and that the new system is fully operational and meets the organization's requirements.

Additionally, this migration plan is designed with a rollback capability to the old system on DigitalOcean. This enables the quick reversion of DNS records to redirect network traffic towards the original service infrastructure, facilitating the re-establishment of the old system's operational status. This precautionary measure is important in mitigating risks linked to the migration process, offering an immediate solution in the event of unexpected complications. Consequently, it ensures the preservation of service integrity and continuity during the migration phase.

5 Results

This chapter presents the outcomes of the migration, assessing its success in relation to the initial objectives and planning. Additionally, it highlights the encountered challenges and examines the role of IaC throughout the migration process.

5.1 Infrastructure Implementation

The migration process spanned a duration of 15 days, during which the infrastructure code was organized across two distinct repositories, dedicated to the backend and the dashboard respectively. The migration required some code modifications in the backend service to enable horizontal scaling provided by Amazon Fargate. The remaining caching logic that was being managed locally was migrated to be handled by the Redis instance. Furthermore, WebSocket connections needed to be reconstructed to support a distributed workload among multiple container instances.

Following the required modifications the infrastructure implementation began with CDK. Within the backend, the IaC was divided into five stacks: Redis, backend service, database, bastion host, and network. In CDK, resources within stacks are provisioned as a single unit. This separation allowed for quicker feedback loops in case any of the stacks had problems, as one stack failure would not affect the others. Furthermore, the development and production environments were set up with slight variations, for example, by allocating fewer computing resources on the development environment to save on operating costs. This allowed the development environment to be configured separately as needed, and also prevented infrastructure drift between the two environments as the infrastructure definitions were shared.

In the dashboard only one stack was created for deploying the infrastructure consisting of CloudFront and S3. The IaC implementation was conducted following the best practices mentioned in the literature. For example, secrets retrieval was separated from the code as can be seen in Figure 12. Furthermore, configuration was separated from the code in a JSON file provided by CDK.

```
const firebaseSecret = secrets.Secret.fromSecretNameV2(  
  this,  
  "firebase-secret",  
  props.isProduction ? "prod/Firebase" : "dev/Firebase"  
);
```

Figure 12. Secrets retrieval from AWS Secrets Manager

Implementing IaC enabled the development team to efficiently construct their infrastructure within a new cloud environment, significantly accelerating development and progress. Defining the infrastructure through IaC was found to be more intuitive and easier to navigate than the traditional graphical user interface (GUI) in AWS, due to its declarative nature. Consequently, this approach facilitated a quicker learning curve. However, the GUI had to be used for specific operations, like the domain setup. Additionally, transitioning to the new cloud platform presented challenges and posed a significant learning curve. The AWS platform and the infrastructure components had to be studied to ensure optimal utilization and configuration.

Facilitating the progress, AWS had already created extensive documentation and examples on CDK usage. These provided almost out-of-the-box working solution for the CloudFront infrastructure to host a website, and many other examples, such as using Fargate with a load balancer and retrieving secrets from Secrets Manager. Figure 13 shows a code snapshot on how a Redis instance was created in the infrastructure. Security-wise, CDK also helpfully restricted firewalls rules to the least privilege by default. Therefore, only explicitly allowed connections could access these services, which helped enhance the overall security posture.

```
this.redis = new CfnCacheCluster(this, `${id}-redis-cluster`, {  
  engine: "redis",  
  cacheNodeType: "cache.t3.small",  
  numCacheNodes: 1,  
  engineVersion: "6.x",  
  autoMinorVersionUpgrade: true,  
  cacheSubnetGroupName: subnetGroup.ref,  
  vpcSecurityGroupIds: [securityGroup.securityGroupId],  
});
```

Figure 13. AWS CDK template for creating a Redis instance

IaC enabled fast deployments of new infrastructure configuration and resources together with the DevOps pipelines. These increased the reliability of deployments as they did not require manual effort, which is prone to errors. However, the feedback loop for infrastructure deployments was found to be quite long. It took on average 15 minutes to deploy the whole application infrastructure. And since problems could only occur in the last stack, this could cause an unnecessary long feedback duration to just deploy a small change. However, dividing the infrastructure resources in smaller, logical stacks reduced the time needed for provisioning, as this allowed CDK to only deploy stacks that had been changed. Furthermore, the stacks could be provisioned individually to improve the testing process. This functionality proved to be very beneficial for making minor adjustments and conducting subsequent tests.

Circular dependency errors occurred between resources during the infrastructure development. The error messages in CDK were helpful in pointing out to the right direction, and these compile-time issues were reported before the actual deployment, which provided a quick feedback loop. The dependency issue was finally resolved by reversing the connections between the dependent services. Figure 14 shows how the circular dependency problem occurred. Since the two resources reference each other through connections, it creates a race condition on how these resources are deployed, and thus preventing the provisioning. Connections are security groups to allow network connections in AWS that can be thought of as firewall rules. When reversing the connections to one side, the framework is automatically able to implicitly interpret the connection and the order of resource creation. This means that all of the connections have to be placed on one of the resources to avoid circular dependencies.

```

// Allow the Fargate service to connect to Redis
// (directionality reversed to avoid cyclic reference)
fargateApplication.service.connections.allowTo(
    props.redisConnection,
    ec2.Port.tcp(6379)
);

// Causes cyclical error
fargateApplication.service.connections.allowFrom(
    props.redisConnection,
    ec2.Port.tcp(6379)
);

```

Figure 14. Cyclic reference error

The AWS monitoring capabilities offered by CloudWatch presented an efficient and user-friendly platform for log observation and performance evaluation. This system facilitated the detection of performance bottlenecks through detailed graphical analyses. A particularly impactful application of this feature was its crucial role in resolving a performance bottleneck associated with job queuing. The use of CloudWatch enabled to identify the root cause of the issue and address a fix quickly. Figure 15 shows how the performance problem was identified in AWS CloudWatch, clearly demonstrating spikes in activity corresponding to the execution of the job.

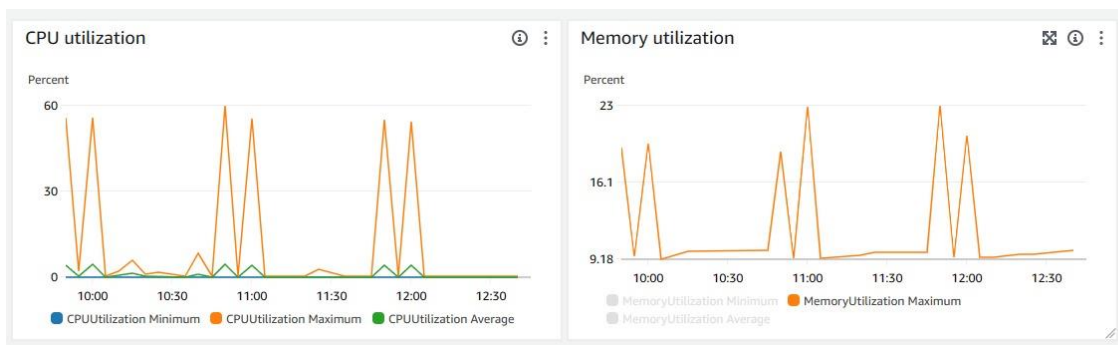


Figure 15. Performance issues detected with AWS CloudWatch monitoring

AWS health checks, designed to ensure the functionality of services, initially presented challenges. Health checks are used by Fargate to monitor the health of services and assess whether new instances should be provisioned to replace services that are down. The backend service creates a new log entry whenever the service is accessed. When AWS performed a health check, it would show up in AWS CloudWatch periodically and thus clutter the logs and complicate the monitoring process. This issue was resolved by the exclusion of the health check user agent from the logs, necessitating a code change in the backend service.

Additionally, another problem emerged related to health check during the backend service deployment process. The default configuration provided by CDK required only 2 unhealthy responses from the service within 20 second to detect that the deployment is not working. This caused an infinite container provision loop caused by Fargate's inherent nature of redeploying failed containers since the application took longer than required to start. By fine-tuning the parameters used to identify unhealthy deployments, this issue was successfully resolved.

Positioning the database within a private network introduced certain challenges. This required creating a separate virtual machine in the public network to act as an intermediary to access the database from outside. This setup is also referred to as a bastion host. The bastion host would be then allowed to access the database, which facilitates managing the database from local computers, like to perform the data migration. To maintain security measures, access to the bastion host was restricted to a carefully selected group of users. The introduction of the bastion host changed the overall architecture from what was initially planned. Apart from this adaptation, no changes were made.

5.2 Overall Migration Process

After validating and testing the configuration in the development environment, final testing was conducted with production settings. The migration was performed at late afternoon accordingly with the plan. No challenges emerged during the deployment to the production environment. Additionally, the DNS propagation had completed in under an hour. After verifying the operability of the production environment, the old environment was discarded in favour of the new one.

The issues encountered presented a challenge in learning the AWS ecosystem and the CDK tool, as they were new to utilize for the development team – presenting a skill gap. A critical aspect of this transition was ensuring robust security within the new environment, necessitating a thorough evaluation to align with best practices. The integration of this stack with continuous DevOps pipelines, coupled with the automation of most infrastructure elements, introduced additional technical complexity.

Benefits	Challenges
<ul style="list-style-type: none"> • Easier monitoring and observability • Improved security posture • Prevent infrastructure drift • Automated deployments • Reduced human error • Easier adaptation with IaC compared to GUI • Code as documentation 	<ul style="list-style-type: none"> • Circular dependencies • New cloud environment and IaC tool learning curve • IaC feedback loop • Technical complexity

Figure 16. Benefits and challenges of the migration

Furthermore, the process of migrating to AWS proved to be relatively straightforward. The majority of the services seamlessly transitioned to AWS-managed services as drop-in replacements. With little changes, the infrastructure was able to take a lot of advantage of the features available in cloud, such as security, scalability, and ease of management. However, tailoring the backend service to support horizontal scaling required additional effort, highlighting the complexity of ensuring scalability in cloud-based architectures.

The migration performed well against the initial plan. Only some changes were necessitated by placing the database in private network, introducing a bastion host as an intermediary access point. Furthermore, the problems found in the previous implementation were able to be properly addressed in the new architecture, facilitating new ways of working. The benefits and challenges of the migration are summarized in Figure 16.

6 Discussion

This chapter critically evaluates the findings from the migration process between different cloud providers while applying IaC to automate the infrastructure provision. It covers the challenges and outcomes of the migration process in the case study and connects the findings to the existing body of literature. Additionally, it assesses the impact of IaC utilization on the overall migration. It also considers implications of the findings, presents the limitations of the study, and proposes directions for future research.

6.1 Discussion of Results

The infrastructure migration to AWS was successfully completed according to the plan, underscoring the critical importance of careful and strategic planning. This process involved thoughtful decisions to tackle the technical problems present in the existing architecture, particularly through a detailed strategy for selecting and integrating AWS services as replacements for the old systems. However, the migration process was also characterized by a series of challenges that required strategic decisions for successful completion. One of the primary challenges encountered was the need to modify the backend service to enable horizontal scaling. These kind of portability issues are a key challenge in cloud migration as mentioned by Kumar & Kumar Garg (2012) and Linthicum (2017).

Strategically, the use of IaC emerged as a key factor for managing the migration complexity. However, the outcomes of the adoption of IaC were multifaceted. On one hand, the migration achieved its technical objectives, with the new cloud infrastructure supporting scalable, distributed applications and leveraging AWS managed services for enhanced security and management. On the other hand, the process presented the extensive learning curve associated with adopting a new cloud platform and tools, namely AWS and CDK.

The effective utilization of IaC with AWS CDK played a key role in automating and managing the infrastructure during the migration. IaC allowed to share infrastructure between development and production environments, thus preventing infrastructure drift, which supports the findings of Morris (2016). The same code was used to deploy the two environments, although with different configurations. The rapid provisioning of

infrastructure with DevOps pipelines allowed for continuous deployment, which led to less risk of human errors in deployment of new infrastructure and reduced the complexity and time taken to conduct updates. This also facilitated the testing process, as small changes could be made without significant effort.

Building infrastructure in a completely new cloud platform for the development team using IaC allowed for quick development and progression compared to accomplishing the same the traditional way. This infrastructure creation process was also facilitated by code examples, which were easier to follow through than tutorials explaining how to navigate the AWS dashboard. This resulted in a reduction in complexity, making it easier to develop and manage infrastructure using IaC, thereby supporting the claims by Morris (2016). Furthermore, in alignment with the findings presented by Werner et al. (2022), this study also found that the infrastructure code also functioned as documentation due to its declarative nature. IaC also allowed the validation of the infrastructure through the inspection of the code, which Morris (2016) also mentions.

However, the adoption of IaC was not without its challenges. The feedback loop for infrastructure deployments remained a bottleneck due to the time-intensive nature of deploying the entire application infrastructure. However, the division of the infrastructure into logical, smaller stacks enabled faster feedback loops and facilitated the migration and testing process. Furthermore, the occurrence of circular dependency errors highlighted the complexities of managing interdependent resources through IaC, requiring advanced troubleshooting and configuration strategies. These observations also align with the challenges presented by Guerriero et al. (2019).

The transition to a new cloud platform and toolset introduced significant challenges due to the development team's unfamiliarity with these technologies. To overcome this skill gap, efforts on learning and training were deemed important, as also noted by Guerriero et al. (2019) and Shuaib et al. (2019). Additionally, the complexity of the migration introduced various technical challenges, which was described as one of the core challenges in cloud migration by Staevsky and Gaftandzhieva (2023). However, comprehensive and strategic planning played a key role in pre-emptively addressing these issues, effectively mitigating the potential for unexpected problems.

It is also notable that the IaC implementation caused a direct portability issue, effectively leading to a vendor lock-in. The CDK tool has been developed by AWS to only function in their ecosystem. Consequently, if the architecture were to be deployed elsewhere, a similar effort would be needed to perform the migration. Guerriero et al. (2019) and Kumar & Kumar Garg (2012) have also raised similar concerns related to vendor lock-in. However, the overall portability effort required to move to the new platform was not deemed significant as most resources could be moved to AWS with a drop-in replacement.

Rahman (2018) raised concerns that the adoption of IaC increases the chance of forgetting about access control measures and firewall rules during the infrastructure setup. However, in this case, where CDK was being utilized, the results were contradictory. This is due to the inherent nature of CDK to apply the strictest rules by default, meaning that access needs to be explicitly stated. This security-focused design principle is considered throughout the framework. Consequently, security was not considered a significant challenge, but instead the focus was on verifying the security functionality through code inspection and by applying the best practices.

The findings suggest that while cloud migration involves considerable technical and operational challenges, strategic utilization of IaC can significantly mitigate these obstacles, enhancing the speed, reliability, and security of the migration process. However, the effectiveness of such strategies is dependent upon the development team's familiarity with the cloud platform and IaC tools, as well as their capability to adapt to new technologies. Furthermore, the use of IaC can initially pose more challenges but can be beneficial in the long run. These findings highlight the need for comprehensive training and resources with necessary knowledge to successfully conduct the migration.

This study contributes to the limited body of knowledge on cloud-to-cloud migrations. The migration project, through its challenges and successes, provides valuable insights into the dynamics of cloud-to-cloud migration and the instrumental role of IaC. These insights offer guidance for organizations navigate their cloud migration process while also illustrating the benefits of leveraging IaC for infrastructure management.

6.2 Lessons Learned

Many insights were learned from the migration process, which are helpful for practitioners when planning similar transitions (see Figure 17). First, the importance of planning cannot be understated. This sets the path for the rest of the migration process, ensuring its successful execution. Furthermore, thorough planning helps to discover possible problems and risks associated with the migration process. For example, the absence of fail-over considerations could have led to significant service disruptions. Additionally, leveraging the Cloud-RMM migration framework facilitated a comprehensive assessment of migration objectives and tasks, ensuring a structured and strategic approach to the transition process. This framework helps in identifying key areas for consideration and sets a clear roadmap for successful migration execution.

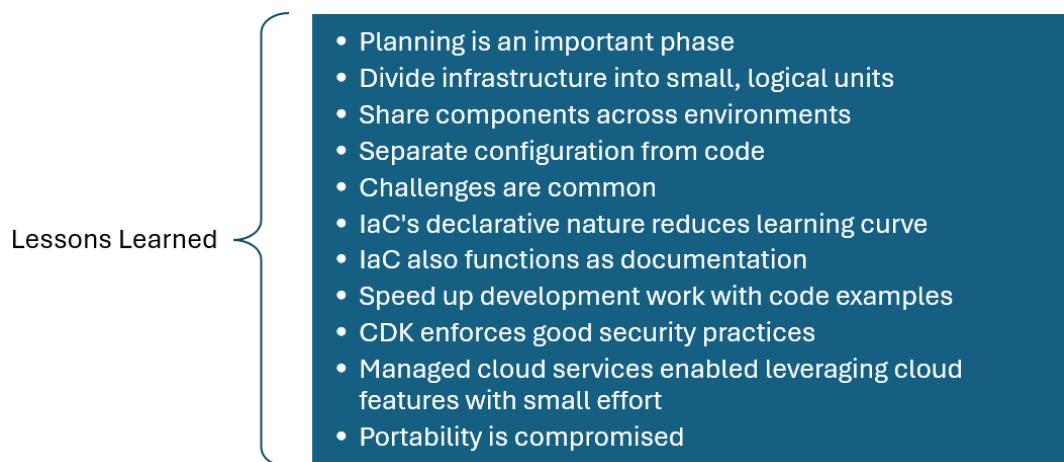


Figure 17. Lessons learned in the migration process

It is also important to note that not all challenges can be avoided; they are bound to happen. For example, circular dependency errors are a common occurrence when infrastructure is divided into smaller parts referencing each other. What is important though, is the knowledge that these errors can happen and how to solve them. In addition, best practices should be followed to minimize the number of challenges that happen during the development and maintenance phases. These will also reduce the total ownership cost in the long run. In this aspect, the division of infrastructure into small, logical stacks is important. As previously

mentioned, resources within stacks are deployed collectively as a single unit. This means that if a change is made within a stack, the whole stack needs to be redeployed. For example, services like a database are persistent and should not require changes often. This makes it a good choice to separate it in its own stack or to one with the same kind of requirements.

This approach also helps in making the infrastructure deployment break as fast as possible, thus reducing the provisioning time. Deploying the entire infrastructure as a single stack could lead to prolonged feedback loops, where errors might only occur after deploying the last resource, resulting in significant delays. However, dividing the infrastructure into stacks allows for independent deployment. Consequently, if errors occur, only the affected stacks require redeployment after implementing necessary fixes, optimizing efficiency.

Infrastructure components should be shared across different environments while retaining the ability to configure them independently. This means that, for example, both production and development environment use the same infrastructure, but with differing configurations. For instance, this allows for dedicating less expensive computing resources to the development environment, optimizing resource utilization. Additionally, the same principle should be also applied when dealing with sensitive information, such as secrets and API keys.

It was also noted that IaC has a lower learning curve than the traditional way of declaring infrastructure via a website interface. This is due to the declarative nature of IaC, which makes it easy to understand for developers by just glancing through the code. This means that the code should be self-documenting, and the infrastructure should be easily verifiable by reviewing the code. Furthermore, given this characteristic, it is highly recommended to follow documentation and utilize the existing code examples to speed up the development work.

Another key lesson related to placing the database inside a private network. During the planning phase, it was not noticed that this would also affect the management of the database, as it could not be accessed publicly. This required the introduction of a bastion host, that acted as an intermediary to access the database. Additionally, security considerations were important throughout the migration process. CDK simplified this task by enforcing the strictest access controls by default, facilitating the seamless application of essential security practices.

Lastly, switching to managed cloud services allowed to capitalize on cloud features with minimal effort. Some code modifications were required to adjust to the new cloud computing platform. This enhanced monitoring and observability allowed to pre-emptively fix potential problems. However, this aspect of building for a specific cloud provider compromises portability. Moreover, the utilization of IaC intensifies this issue, leading to a significant risk of vendor lock-in.

6.3 Limitations and Future Work

Several limitations have been identified within the study, primarily due to its design as a single case study, which significantly constrains its generalizability. One limitation is the focus on a single organization's migration to a specific cloud service provider. This scope may not capture the variety of challenges or benefits experienced by other organizations with different infrastructures or operational needs. Furthermore, due to the sensitive nature of the data involved in the migration, detailed technical and operational insights might have been omitted, limiting the depth of analysis regarding security practices and data handling.

Another significant limitation arises from potential researcher bias, as the individual conducting the migration was also responsible for reporting on it, which could affect the reliability and validity of the findings. Furthermore, this case study overlooks the evaluation of infrastructure aspects, such as cost and performance, which are factors that organizations often consider crucial when deciding to migrate to a different cloud provider.

Future studies could address these gaps by incorporating multiple case studies across various industries and cloud services, thereby offering a more comprehensive and generalizable understanding of cloud migration's impacts and challenges. Moreover, conducting longitudinal studies to assess the long-term effects of cloud migration on performance, cost, and security could provide deeper insights into the benefits observed. Additionally, a comparative analysis focusing on the role of IaC in influencing the financial and operational success of migration projects would offer valuable insights.

7 Conclusions

This thesis explored the strategies, challenges, and outcomes involved in migrating an application architecture between different cloud providers. Furthermore, the impact of leveraging IaC in the migration process was evaluated. The study was executed as a case study within a small Finnish startup operating in self-service mobile payment systems, offering a detailed examination of the impacts and experiences.

The migration strategy, which adopted a replatforming approach, facilitated the transition of the case organization's application architecture from DigitalOcean to AWS. This strategic change allowed the organization to utilize AWS's managed services, effectively addressing previously identified challenges such as scalability, security, and operational complexity. The adoption of managed services like Amazon RDS and Amazon ElastiCache streamlined operations while enhancing the security and reliability of the application architecture. Similarly, the utilization of Amazon Fargate for the backend service eliminated the need for server management, thus allowing the development team to focus on application improvement rather than infrastructure maintenance.

Furthermore, the adoption of IaC, specifically AWS CDK, was important in achieving an efficient, reliable, and scalable cloud infrastructure. By encapsulating the infrastructure definition within code, the case organization realized several key benefits. These include enhanced version control, repeatability of infrastructure deployment, and significant reductions in manual configuration errors. AWS CDK also enhanced the overall security, as the framework inherently enforces strictest access controls by default, thus requiring access to be explicitly stated. Additionally, IaC was found to have an easier learning curve than manual infrastructure definition in AWS dashboard. It also facilitated a seamless migration process by allowing for precise and predictable deployments, which could be automatically repeated across different environments, like testing and production. Furthermore, the use of IaC enabled continuous iteration and deployment, facilitating quick resolution of issues as they occurred.

The migration process also presented several challenges. Adjusting code to facilitate horizontal scaling in a serverless architecture and managing the complexities of data migration were significant technical obstacles. Additionally, the integration of new cloud

services required substantial training and a steep learning curve, especially with the adoption IaC. Specifically, IaC introduced additional technical challenges during the infrastructure implementation, such as circular dependencies, and a longer feedback loop for deployment. Additionally, the use of AWS CDK and the broader AWS ecosystem limited portability, leading to a potential vendor lock-in.

The findings suggest that while cloud migration involves considerable technical and operational challenges, strategic utilization of IaC can significantly mitigate these obstacles, enhancing the speed, reliability, and security of the migration process. Furthermore, this study contributes to the existing body of knowledge by providing a detailed analysis of cloud-to-cloud migration process and its challenges, which also serves as a practical guide for companies navigating the complexities of cloud infrastructure transitions.

One notable limitation of this research is its reliance on a single case study, which affects the generalizability of the results. Future research should expand on this work by exploring cloud-to-cloud migrations in different organizational contexts and with various cloud providers to build on the findings provided here. Further studies could also explore the long-term impacts of cloud migrations and IaC on organizational efficiency.

References

- Agarwal, A. and Raina, S. (2012) ‘Live Migration of Virtual Machines in Cloud’, 2(6).
- Ahmad, N., Naveed, Q.N. and Hoda, N. (2018) ‘Strategy and procedures for Migration to the Cloud Computing’, in *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*. *2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pp. 1–5. Available at: <https://doi.org/10.1109/ICETAS.2018.8629101>.
- Andrikopoulos, V., Binz, T., Leymann, F. and Strauch, S. (2013) ‘How to adapt applications for the Cloud environment: Challenges and solutions in migrating applications to the Cloud’, *Computing*, 95(6), pp. 493–535. Available at: <https://doi.org/10.1007/s00607-012-0248-2>.
- Artač, M., Borovssak, T., Di Nitto, E., Guerriero, M. and Tamburri, D. (2017) *DevOps: Introducing Infrastructure-as-Code*, p. 498. Available at: <https://doi.org/10.1109/ICSE-C.2017.162>.
- AWS (2024a) *AWS Cloud Development FAQs*, Amazon Web Services, Inc. Available at: <https://aws.amazon.com/cdk/faqs/> (Accessed: 17 March 2024).
- AWS (2024b) *Managed SQL Database - Amazon Relational Database Service (RDS) - AWS*, Amazon Web Services, Inc. Available at: <https://aws.amazon.com/rds/> (Accessed: 18 March 2024).
- AWS (2024c) *Prioritization and migration strategy - AWS Prescriptive Guidance*. Available at: <https://docs.aws.amazon.com/prescriptive-guidance/latest/application-portfolio-assessment-guide/prioritization-and-migration-strategy.html#migration-r-type> (Accessed: 6 March 2024).
- AWS (2024d) *Serverless Compute - AWS Fargate - AWS*, Amazon Web Services, Inc. Available at: <https://aws.amazon.com/fargate/> (Accessed: 2 April 2024).
- Balakrishnan, T., Gnanasambandam, C., Santos, L. and Srivathsan, B. (2021) ‘Cloud-migration opportunity: Business value grows, but missteps abound’.
- Balobaid, A. and Debnath, D. (2018) ‘Cloud Migration Tools: Overview and Comparison’, in A. Yang, S. Kantamneni, Y. Li, A. Dico, X. Chen, R. Subramanyan, and L.-J. Zhang (eds) *Services – SERVICES 2018*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 93–106. Available at: https://doi.org/10.1007/978-3-319-94472-2_7.
- Binz, T., Breiter, G., Leyman, F. and Spatzier, T. (2012) ‘Portable Cloud Services Using TOSCA’, *IEEE Internet Computing*, 16(3), pp. 80–85. Available at: <https://doi.org/10.1109/MIC.2012.43>.

- Dalla Palma, S., Di Nucci, D., Palomba, F. and Tamburri, D.A. (2020) ‘Toward a catalog of software quality metrics for infrastructure code’, *Journal of Systems and Software*, 170, p. 110726. Available at: <https://doi.org/10.1016/j.jss.2020.110726>.
- Dar, A. (2018) ‘Cloud Computing-Positive Impacts and Challenges in Business Perspective’, *Journal of Computer Science & Systems Biology*, 12. Available at: <https://doi.org/10.4172/jcsb.1000294>.
- Di Martino, B., Petcu, D., Cossu, R., Goncalves, P., Máhr, T. and Loichate, M. (2011) ‘Building a Mosaic of Clouds’, in M.R. Guarracino, F. Vivien, J.L. Träff, M. Cannatoro, M. Danelutto, A. Hast, F. Perla, A. Knüpfer, B. Di Martino, and M. Alexander (eds) *Euro-Par 2010 Parallel Processing Workshops*. Berlin, Heidelberg: Springer (Lecture Notes in Computer Science), pp. 571–578. Available at: https://doi.org/10.1007/978-3-642-21878-1_70.
- Docker (2024) *Docker Compose overview*, *Docker Documentation*. Available at: <https://docs.docker.com/compose/> (Accessed: 24 March 2024).
- Eddelbuettel, D. (2022) ‘A Brief Introduction to Redis’. arXiv. Available at: <http://arxiv.org/abs/2203.06559> (Accessed: 26 March 2024).
- Ellison, M., Calinescu, R. and Paige, R.F. (2018) ‘Evaluating cloud database migration options using workload models’, *Journal of Cloud Computing*, 7(1), p. 6. Available at: <https://doi.org/10.1186/s13677-018-0108-5>.
- Forrester (2015) ‘Infrastructure As Code: Fueling The Fire For Faster Application Delivery’.
- Gartner (2024) *Forecast: Public Cloud Services, Worldwide, 2021-2027, 2Q23 Update*, *Gartner*. Available at: <https://www.gartner.com/en/documents/4509999> (Accessed: 20 April 2024).
- Gholami, M.F., Daneshgar, F., Beydoun, G. and Rabhi, F. (2017) ‘Challenges in migrating legacy software systems to the cloud — an empirical study’, *Information Systems*, 67, pp. 100–113. Available at: <https://doi.org/10.1016/j.is.2017.03.008>.
- Gholami, M.F., Daneshgar, F., Low, G. and Beydoun, G. (2016) ‘Cloud migration process— A survey, evaluation framework, and open challenges’, *Journal of Systems and Software*, 120, pp. 31–69. Available at: <https://doi.org/10.1016/j.jss.2016.06.068>.
- Goyal, S. (2014) ‘Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review’, *International Journal of Computer Network and Information Security*, 6, pp. 20–29. Available at: <https://doi.org/10.5815/ijcnis.2014.03.03>.
- Guerriero, Mi., Garriga, M., Tamburri, D.A. and Palomba, F. (2019) ‘Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry’, in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA: IEEE, pp. 580–589. Available at: <https://doi.org/10.1109/ICSME.2019.00092>.

Haris, M. and Khan, R.Z. (2018) ‘A Systematic Review on Cloud Computing’, *International Journal of Computer Sciences and Engineering*, 6, pp. 632–639. Available at: <https://doi.org/10.26438/ijcse/v6i11.632639>.

Hartauer, R., Manner, J. and Wirtz, G. (2022) ‘Cloud Function Lifecycle Considerations for Portability in Function as a Service’, in *Proceedings of the 12th International Conference on Cloud Computing and Services Science. 12th International Conference on Cloud Computing and Services Science*, Online Streaming, --- Select a Country ---: SCITEPRESS - Science and Technology Publications, pp. 133–140. Available at: <https://doi.org/10.5220/0010999000003200>.

HashiCorp (2024) *Terraform by HashiCorp, Terraform by HashiCorp*. Available at: <https://www.terraform.io/> (Accessed: 17 March 2024).

Hosseini Shirvani, M., Amin, G.R. and Babaeikiadehi, S. (2022) ‘A decision framework for cloud migration: A hybrid approach’, *IET Software*, 16(6), pp. 603–629. Available at: <https://doi.org/10.1049/sfw2.12072>.

Hussein, N.I., Hashem, M. and Li, Z. (2013) ‘Security Migration Requirements: From Legacy System to Cloud and from Cloud to Cloud’, in *2nd International Symposium on Computer, Communication, Control and Automation*, Atlantis Press, pp. 248–252. Available at: <https://doi.org/10.2991/3ca-13.2013.62>.

Jamshidi, P., Ahmad, A. and Pahl, C. (2013) ‘Cloud Migration Research: A Systematic Review’, *IEEE Transactions on Cloud Computing*, 1(2), pp. 142–157. Available at: <https://doi.org/10.1109/TCC.2013.10>.

Kolb, S. (2019) *On the Portability of Applications in Platform as a Service*. University of Bamberg Press.

Kolb, S., Lenhard, J. and Wirtz, G. (2015) *Application Migration Effort in the Cloud - The Case of Cloud Platforms*. Available at: <https://doi.org/10.1109/CLOUD.2015.16>.

Kolb, S. and Wirtz, G. (2014) *Towards Application Portability in Platform as a Service, Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*. Available at: <https://doi.org/10.1109/SOSE.2014.26>.

Koziolk, H., Hark, R., Eskandani, N., Nguyen, P.S. and Rodriguez, P. (2023) ‘TOSCA for Microservice Deployment in Distributed Control Systems: Experiences and Lessons Learned’, in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C). 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, L’Aquila, Italy: IEEE, pp. 11–21. Available at: <https://doi.org/10.1109/ICSA-C57050.2023.00020>.

Kratzke, N. and Quint, P.-C. (2017) ‘Understanding Cloud-native Applications after 10 Years of Cloud Computing - A Systematic Mapping Study’, *Journal of Systems and Software*, 126, pp. 1–16. Available at: <https://doi.org/10.1016/j.jss.2017.01.001>.

Kumar, V. and Kumar Garg, K. (2012) ‘Migration of Services to the Cloud Environment: Challenges and Best Practices’, *International Journal of Computer Applications*, 55(1), pp. 1–6. Available at: <https://doi.org/10.5120/8716-7105>.

Kumara, I., Garriga, M., Romeu, A.U., Di Nucci, D., Palomba, F., Tamburri, D.A. and van den Heuvel, W.-J. (2021) 'The do's and don'ts of infrastructure code: A systematic gray literature review', *Information and Software Technology*, 137, p. 106593. Available at: <https://doi.org/10.1016/j.infsof.2021.106593>.

Linthicum, D.S. (2017) 'Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between', *IEEE Cloud Computing*, 4(5), pp. 12–14. Available at: <https://doi.org/10.1109/MCC.2017.4250932>.

Lwakatare, L.E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M. and Lassenius, C. (2019) 'DevOps in practice: A multiple case study of five companies', *Information and Software Technology*, 114, pp. 217–230. Available at: <https://doi.org/10.1016/j.infsof.2019.06.010>.

Maniah, Soewito, B., Lumban Gaol, F. and Abdurachman, E. (2022) 'A systematic literature Review: Risk analysis in cloud migration', *Journal of King Saud University - Computer and Information Sciences*, 34(6, Part B), pp. 3111–3120. Available at: <https://doi.org/10.1016/j.jksuci.2021.01.008>.

Mell, P. and Grance, T. (2011) 'The NIST Definition of Cloud Computing'.

Miranda, J., Guillen, J., Murillo, J.M. and Canal, C. (2013) 'Assisting Cloud Service Migration Using Software Adaptation Techniques', in *2013 IEEE Sixth International Conference on Cloud Computing. 2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*, Santa Clara, CA: IEEE, pp. 573–580. Available at: <https://doi.org/10.1109/CLOUD.2013.35>.

Morris, K. (2016) *Infrastructure As Code: Managing Servers in the Cloud*. 1st edition. Beijing: O'Reilly & Associates Inc.

Narantuya, J., Zang, H. and Lim, H. (2018) 'Service-Aware Cloud-to-Cloud Migration of Multiple Virtual Machines', *IEEE Access*, 6, pp. 76663–76672. Available at: <https://doi.org/10.1109/ACCESS.2018.2882651>.

Nedeltcheva, G.N., Xiang, B., Niculut, L. and Benedetto, D. (2023) 'Challenges Towards Modeling and Generating Infrastructure-as-Code', in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering. ICPE '23: ACM/SPEC International Conference on Performance Engineering*, Coimbra Portugal: ACM, pp. 189–193. Available at: <https://doi.org/10.1145/3578245.3584937>.

Novakova Nedeltcheva, G., De La Fuente Ruiz, A., Orue-Echevarria Arrieta, L., Bat, N. and Blasi, L. (2022) 'Towards Supporting the Generation of Infrastructure as Code Through Modelling Approaches - Systematic Literature Review', in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, pp. 210–217. Available at: <https://doi.org/10.1109/ICSA-C54293.2022.00048>.

Pant, P. and Thakur, S. (2013) 'Data Migration Across The Clouds', 3(2).

pgAdmin (2024) *pgAdmin - PostgreSQL Tools*. Available at: <https://www.pgadmin.org/> (Accessed: 26 March 2024).

- Rahman, A. (2018) ‘Anti-Patterns in Infrastructure as Code’, in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Vasteras: IEEE, pp. 434–435. Available at: <https://doi.org/10.1109/ICST.2018.00057>.
- Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019) ‘Where Are The Gaps? A Systematic Mapping Study of Infrastructure as Code Research’, *Information and Software Technology*, 108, pp. 65–77. Available at: <https://doi.org/10.1016/j.infsof.2018.12.004>.
- Rai, R., Mehfuz, S. and Sahoo, G. (2014) ‘Efficient Migration of Application to Clouds: Analysis and Comparison’, *GSTF Journal on Computing (JoC)*, 3(3), p. 23. Available at: <https://doi.org/10.7603/s40601-013-0023-z>.
- Rai, R., Sahoo, G. and Mehfuz, S. (2015) ‘Exploring the factors influencing the cloud computing adoption: a systematic study on cloud migration’, *SpringerPlus*, 4(1), p. 197. Available at: <https://doi.org/10.1186/s40064-015-0962-2>.
- Runeson, P. and Höst, M. (2009) ‘Guidelines for conducting and reporting case study research in software engineering’, *Empirical Software Engineering*, 14(2), pp. 131–164. Available at: <https://doi.org/10.1007/s10664-008-9102-8>.
- Sabiri, K., Benabbou, F., Moutachaouik, H. and Hain, M. (2015) ‘Towards a cloud migration framework’, in *2015 Third World Conference on Complex Systems (WCCS)*. *2015 Third World Conference on Complex Systems (WCCS)*, pp. 1–6. Available at: <https://doi.org/10.1109/ICoCS.2015.7483315>.
- Salesforce (2024) *Maailman johtava CRM-ohjelmisto*, *Salesforce*. Available at: <https://www.salesforce.com/fi/> (Accessed: 18 March 2024).
- Shastry, A.L., Nair, D.S., Prathima, B., Ramya, C.P. and Hallymysore, P. (2022) ‘Approaches for migrating non cloud-native applications to the cloud’, in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0632–0638. Available at: <https://doi.org/10.1109/CCWC54503.2022.9720856>.
- Shayan, J., Azarnik, A., Chuprat, S., Karamizadeh, S. and Alizadeh, M. (2013) ‘Identifying Benefits and Risks Associated with Utilizing Cloud Computing’, 3(3).
- Shuaib, M., Samad, A., Alam, S. and Siddiqui, S. (2019) ‘Why Adopting Cloud Is Still a Challenge?—A Review on Issues and Challenges for Cloud Migration in Organizations’, in, pp. 387–399. Available at: https://doi.org/10.1007/978-981-13-5934-7_35.
- Sokolowski, D. (2022) ‘Infrastructure as code for dynamic deployments’, in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery (ESEC/FSE 2022), pp. 1775–1779. Available at: <https://doi.org/10.1145/3540250.3558912>.

Spjuth, O., Frid, J. and Hellander, A. (2021) 'The machine learning life cycle and the cloud: implications for drug discovery', *Expert Opinion on Drug Discovery*, 16, pp. 1–9. Available at: <https://doi.org/10.1080/17460441.2021.1932812>.

Srivastava, P. and Khan, R. (2018) 'A Review Paper on Cloud Computing', *International Journal of Advanced Research in Computer Science and Software Engineering*, 8, p. 17. Available at: <https://doi.org/10.23956/ijarcsse.v8i6.711>.

Staevisky, N. and Gaftandzhieva, S. (2023) 'Cloud Migration: Identifying the Sources of Potential Technical Challenges and Issues', *International Journal of Advanced Computer Science and Applications (IJACSA)*, 14(12). Available at: <https://doi.org/10.14569/IJACSA.2023.0141204>.

Strunk, A. (2012) 'Costs of Virtual Machine Live Migration: A Survey', in *2012 IEEE Eighth World Congress on Services. 2012 IEEE World Congress on Services (SERVICES)*, Honolulu, HI, USA: IEEE, pp. 323–329. Available at: <https://doi.org/10.1109/SERVICES.2012.23>.

Tomarchio, O., Calcaterra, D., Di Modica, G. and Mazzaglia, P. (2021) 'TORCH: a TOSCA-Based Orchestrator of Multi-Cloud Containerised Applications', *Journal of Grid Computing*, 19(1), p. 5. Available at: <https://doi.org/10.1007/s10723-021-09549-z>.

Vaquero, L.M., Rodero-Merino, L., Caceres, J. and Lindner, M. (2008) 'A break in the clouds: towards a cloud definition', *ACM SIGCOMM Computer Communication Review*, 39(1), pp. 50–55. Available at: <https://doi.org/10.1145/1496091.1496100>.

Werner, C., Li, Z.S., Lowlind, D., Elazhary, O., Ernst, N. and Damian, D. (2022) 'Continuously Managing NFRs: Opportunities and Challenges in Practice', *IEEE Transactions on Software Engineering*, 48(7), pp. 2629–2642. Available at: <https://doi.org/10.1109/TSE.2021.3066330>.

Wielki, J. (2015) 'An analysis of the opportunities and challenges connected with utilization of the cloud computing model and the most important aspects of the migration strategy', in *2015 Federated Conference on Computer Science and Information Systems*, pp. 1569–1574. Available at: <https://doi.org/10.15439/2015F93>.

Wirasti, H.D., Seta, H., Witasryah, D., Prabu, H.K., Azzahro, A. and Hananto, B. (2023) 'Challenges on Cloud Computing Migration Strategy for Music Industry: A Systematic Literature Review', in *2023 International Conference on Informatics, Multimedia, Cyber and Informations System (ICIMCIS). 2023 International Conference on Informatics, Multimedia, Cyber and Information Systems (ICIMCIS)*, Jakarta Selatan, Indonesia: IEEE, pp. 699–704. Available at: <https://doi.org/10.1109/ICIMCIS60089.2023.10348989>.

Yin, R.K. (2009) *Case Study Research: Design and Methods*. SAGE.

Yussupov, V., Breitenbücher, U., Leymann, F. and Müller, C. (2019) 'Facing the Unplanned Migration of Serverless Applications: A Study on Portability Problems, Solutions, and Dead Ends', in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. New York, NY, USA: Association for Computing Machinery (UCC'19), pp. 273–283. Available at: <https://doi.org/10.1145/3344341.3368813>.

Zhang, Q., Cheng, L. and Boutaba, R. (2010) 'Cloud computing: state-of-the-art and research challenges', *Journal of Internet Services and Applications*, 1(1), pp. 7–18. Available at: <https://doi.org/10.1007/s13174-010-0007-6>.