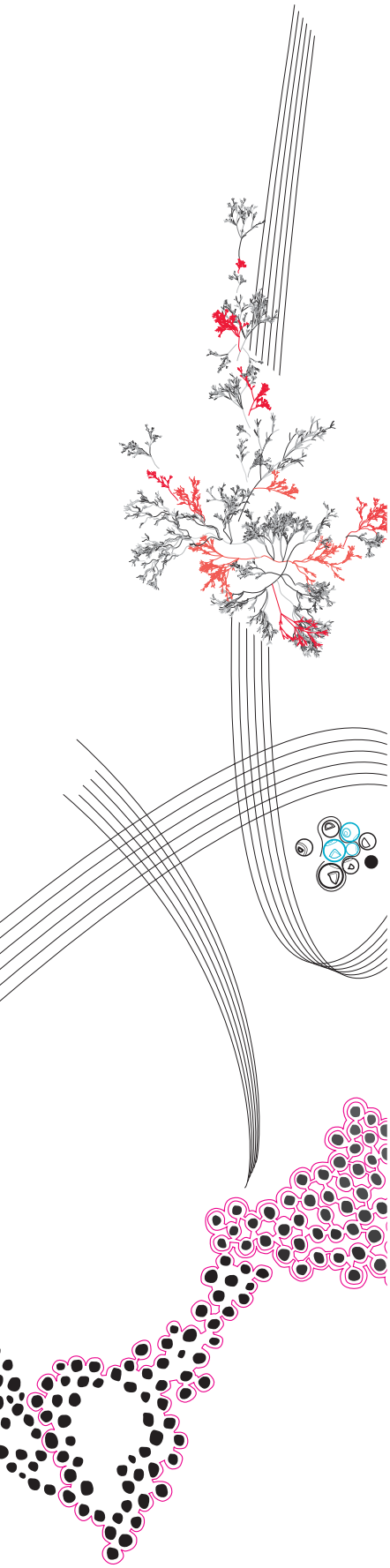

MSc Computer Science
Thesis

Private Legacy to Cloud: A Tailored Migration Method for Private Cloud Deployment of Legacy Software Projects

Max de Blok

Supervisor: Dr. Luís Ferreira Pires
Second Superviser: L.R. de Vries
External supervisor: Dr. F.J. Castor de Lima Filho
Company supervisor: Harm Jan Spier

October, 2024

Department of Computer Science
Faculty EEMCS, University of Twente
Thales

Abstract

This research aimed to create a migration method for private cloud migrations, which are defined in this research as migrations towards software and infrastructure that resemble public cloud infrastructure but are still hosted privately. This includes migration towards microservice architecture, containerisation and container orchestration. The migration solution is based on existing migration methods towards public cloud deployment and microservice architecture. The existing methods are adapted to mitigate the decreased benefits in terms of scaling, outsourcing and development speed and account for the inclusion of the infrastructure in the migration. The developed method accounted for these differences by adding new analysis steps, namely shared goal creation, identification of different sub-migrations and additional decision moments. The developed migration method was then validated using interviews with software development experts and a case study at a company preparing for this specific migration. The expert interviews resulted in positive responses and helpful additions to the framework. The case study showed that the migration analysis part of the method was successful. However, the rest of the method was not possible to directly test, as the analysis in the case study did not result in a clear migration path. Overall, the solution created is deemed an improvement over the public cloud migration methods when applied to private cloud migration.

Preface

My computer science journey started at the beginning of my master's. After my bachelor's in applied physics, I switched to follow my passion for developing cool software. Some courses and events during the Software Technologies master track inspired me more than others. One of these courses was the Software Oriented Architecture. It showed me a glimpse into the existing distributed software architectures and showed me Kubernetes. The course given by Dr. Luís Ferreira Pires led me to email him to guide my research regarding this topic. Thank you, Luís, for the inspiration and guidance during this research.

Another inspiring event during the masters was the guest lecture on software development practices at Thales, given by Harm Jan Spier. He presented me with a problem he foresaw with adopting containerisation into their project. The move towards containerisation was part of a larger migration towards what, in my mind, was cloud infrastructure. The transition towards cloud infrastructure and Microservice architecture is a very hot topic. The benefits these technologies bring are well understood and highly valued. However, as experienced during the creation of this work, the drawbacks of these technologies are not as widely understood and often underestimated. This makes analysing the transition towards these technologies difficult. This analysis becomes even more important in a restricted domain, as some of the benefits do not apply. I came into this journey convinced that the CMS should migrate towards microservices and that the problems identified could be solved. While developing and applying the migration method, I got to speak with many talented software developers. It was wonderful to talk to many engineers with the same passion. The analysis of these conversations and the insight and discussions with Harm Jan slowly changed my entirely positive view of the technologies to a more critical view. Thank you very much, Harm Jan, for the opportunity, the inspiring discussions, the valuable knowledge and the trust during this project.

I struggled during this research with the lack of objectivity. I learned during my bachelor's in physics to look for concrete data. During this research, however, I moved more and more towards human behaviour and processes. I knew the research direction was valuable to at least Thales, but the lack of proof was a mental struggle. I'm incredibly grateful for the help Ilia Awakimjan provided. With his help, I could interview experts from different companies, presenting and verifying my work. I want to thank Wouter Kersteman, Wesley Dekker, Quenten Schoemaker, and Menno de Jong for their time during these interviews. I also want to thank L.R. de Vries and Dr. F.J. Castor de Lima Filho for providing external validation to my work while being part of my research committee.

This work represents my journey during this research project. The report starts with understanding cloud-native development's incredible technologies and techniques. Afterwards, you are hopefully nearly as excited about these technologies as I was (and still am). It then shows you the need to consider the positives and negatives of engineering decisions and my complete analysis of the migration happening at Thales. Above all, the work shows you the biggest lesson I learned, that migrations are a complex software engineering task and, therefore, need to be treated using the same methods we use for everyday software engineering, like careful planning, analysis and iterative progression.

Have a fun journey through my thesis!

Contents

Glossary

Application	An entire software system excluding the infrastructure it runs on, can contain many services.
Cloud	On-demand available computational and storage resources.
Cloud Native Software	Software optimised to run on cloud infrastructure.
DevOps	Software development method that aims to combine the silos of development and operations.
High Assurance System	A system that tries to guarantee consistent functionality
Infrastructure	The software and hardware responsible for hosting the software of CMS.
Migration Method	A method to structure the analysis and plan creation concerning a migration.
Migration Plan	A concrete plan to tackle a specific migration.
NoOps	An IT environment automated and abstracted for the infrastructure to such a degree that no operations team is needed.
N+1 Architecture	An architecture where Components (N) have at least one independent backup component (+1) that serves as redundancy.
Private Cloud	The definition used in this work for a private distributed computational cluster.
Product	In this case, the entire system delivered to the customer.
Product Migration	The migration of the entire software product, this includes the infrastructure, organisation and applications.
Service	A single business case or functionality running as a separate process.
Silo	A department responsible for a specific part of the product, for example, the infrastructure.

Abbreviations

ACL	Anti Corruption Layer
API	Application Programmable Interface
CD	Continuous Delivery
CI	Continuous Integration
CN	Cloud Native
CMS	Combat Management System
DDD	Domain Driven Design
DDS	Data Distribution Service
FCS	Fire Control System
IP	Intellectual Property
JVM	Java Virtual Machine
MSA	MicroService Architecture
PaaS	Platform as a Service
SaaS	Software as a Service
SOA	Service Oriented Architecture

1 Introduction

Software migration is changing legacy software from an "as-is" system towards an envisioned "to-be" system that fulfils all current requirements with new characteristics. Migrations can be complex undertakings for large software projects, as characteristics of both the "as-is" and "to-be" systems need to be analysed and used to create a path from one to the other. Therefore, a method for performing this analysis is useful. This research aimed to create a migration method for private distributed computational clusters. For the rest of the work, this will be called private cloud [1].

The structure of this chapter is as follows: first, the motivation for why this research was conducted is presented. Then, the problem statement, objectives and approach are laid out in their respective sections. Lastly, the structure of the rest of the report is explained.

1.1 Motivation

From practical experience, it can be observed that all mission-critical software projects slowly turn into legacy software. The definition of legacy software varies; however, most definitions include that the software currently cannot easily be changed to meet new business requirements [32]. These legacy applications range from old applications written in COBOL and FORTRAN to more modern types of applications that have fallen behind in agility and flexibility. Companies highly value their software's agility and flexibility [33]. These characteristics of the software can change with migrations to new architectures, technologies or infrastructure. Migrations are often complex and time-consuming endeavours, implying that they are also potentially costly. The potentially complex nature of these migrations frequently requires significant investments. These investments only bring improvement to future development and do not always bring direct benefits to the customer. These complex migrations are also known to fail, making the potential cost even larger [54]. To minimize risk and persuade decision-makers about a migration, engineers need to plan and analyse it beforehand. As the analysis spans the "as-is" system, "to-be" system and the path between them, a migration method is needed to aid this extensive analysis. Every project has different requirements and specifications, meaning that methods need to be tailorable to the specifics of each project. This makes the generalization of migration methods a necessary task.

An example of a migration that many software teams are tackling is a transition to cloud infrastructures. The transition to public cloud brings many benefits, among outsourcing the platform, possible cost reduction and dynamic scaling [23]. Companies use specific software development methods to utilize the properties of the cloud to their maximum capacity. These properties together form the cloud-native approach, defined by the Cloud Native Computing Foundation as:" Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative Application Programmable Interfaces (API) exemplify this approach." [2].

Another prevalent migration often included in cloud-native migration, is towards Service Oriented Architectures (SOA) and, in recent days, mostly to MicroService Architectures (MSA). These software architectures directly oppose the architecture used in most legacy software systems, the so-called monolithic architecture. A monolithic software project structures its software as one extensive process, often separated into modules [33] which communicate using function calls. SOA focuses on separating individual services the software provides into their own (small) programs that are operated and developed independently. MSA takes this concept even further by splitting the dependencies and data ownership for each service, completely decoupling the life cycles of all services. These architectures can profit much more from the flexible scaling, and the efficiency of the cloud [39]. They give development teams complete ownership of services, including deployment and operation. Because SOA can utilize the cloud more efficiently, the SOA or MSA migration and Cloud migration are often performed together. Migrating to an SOA/MSA involves splitting the application into well-bounded services that usually need new supporting technologies to automate the now-decoupled building, deployment and monitoring processes.

Migrations are potentially complex and therefore prone to failure [54]. This means it is vital for the decision-makers of these projects to have resources available to assist in creating migration paths. Migration methods and plan creation for cloud and SOA migration are well documented in the literature. [35] [33]. These migrations are not possible for all software projects, primarily due to safety, security or domain restrictions [32]. These software projects are often deployed on privately hosted infrastructure and are not allowed or able to host their infrastructure somewhere else. The benefits of implementing cloud technologies and SOA are still attractive for software

[1] The term cloud does not directly fit here, as cloud refers to on-demand computation and private cloud often refers to a private cloud at a public cloud provider. However, the term is chosen as the private infrastructure is modelled after the public cloud.

projects subject to these restrictions. These applications could benefit from reworking their private infrastructure to resemble public clouds. Most technologies used in these public cloud clusters are open-source and can therefore be implemented in private clusters. Their applications can then be migrated as any application towards a public cloud deployment. Combining these two migrations, companies can migrate their product, often resulting in more future-proof and stable software deployments. This product migration spans the changes in the infrastructure and the software, which now have to be analysed together. The creation of the product migration method is the subject of this research.

1.2 Problem statement

From the above motivation, we concluded that legacy software's demand for agility and flexibility draws engineers of specific software applications towards cloud and SOA migrations. However, specific projects are limited by security, privacy or other reasons and cannot migrate to the public cloud. These projects currently lack methods for analysing and possibly performing a migration towards a private cloud. Migrating towards a private cloud means that the infrastructure is still in control of the company, meaning the cost, scaling and software development implications of the migration are possibly different.

This research addresses the problem of constructing an adaptable migration method for application and infrastructure migration towards private cloud deployment. The complexity of the private deployment, coupled with the characteristics of transitioning from monolithic structures to private clouds, requires a well-defined method. This method aims to allow these legacy software projects to effectively gain cloud benefits of scalability, agility and flexibility, as these benefits extend beyond migration towards public cloud. It also aims to guide engineers in the analysis of possible lost or reduced benefits, like dynamic scaling and cost reduction. The challenge lies in generalizing this method so it can be applied to a range of projects while still considering the security, privacy and other domain restrictions that may limit the migration of each specific project. Thus, the research focuses on formulating a migration method that strategically combines the advantages of cloud-native approaches with the constraints of private deployment, ensuring a future-proof and stable software environment with little overhead and smooth integration in the development process.

1.3 Research questions

The objective of the research is to improve the migration to private cloud deployments by designing a migration method that is based on cloud and SOA migration methods while complying with requirements and constraints set by the engineers of the analysed project so that engineers can increase the agility and flexibility of the system. A set of research questions supports this primary objective, which can be divided into three different stages of research: (1) Analysis of the context, (2) Creation of a method, and (3) Verification of the created method.

To create a solution to a problem, the context of that problem needs to be understood first. The context consists of understanding how to create a private cloud and what benefits a private cloud brings. To make an engineering decision, the possible downsides of cloud implementation must also be understood. Knowing this context allows the problem to be analysed in the next step. The research questions for analysis of the context are:

- **Q1:** What technologies and architectures contribute to a private cloud?
- **Q2:** Who are the stakeholders of these migrations, and what benefits do they expect?
- **Q3:** What migration methods currently exist for the transition towards cloud infrastructure?

After the context of the problem is established, the problem itself is analysed and solved. A solution is created using the context and the characteristics of the problem. To come to a possible solution, a few questions were answered:

- **Q4:** What are the characteristics of a migration method towards private cloud deployment?
- **Q5:** What are the differences between public and private cloud migrations?
- **Q6:** Based on the problem analysis, what would a possible migration method look like, and what steps should be included?

The last part of the research concerned verifying the created method and solution. The context and problem analysis resulted in factors separating public and private cloud migrations. The altercations made to existing methods to account for these factors are verified in the last part. The questions related to verification are:

- **Q7:** How do experts in the field of software architecture perceive the created method?
- **Q8:** Can the method be applied successfully in a project attempting a private cloud migration?

1.4 Approach

To create and validate a migration method for private cloud deployments, the Design Science Methodology for Information Systems and Software Engineering was used [59]. The approach to accomplish the main objective of the research followed the structure of the questions. These questions were also structured according to the Design Science Methodology. This research originated at a company attempting a private cloud migration. This company mainly served as a case study during verification, however, context and problem analysis also took place there.

The approach for this research consisted of several steps. These are listed below:

1. At the beginning of the research, the most crucial step was to understand the problem the company was experiencing, the software they used and the domain restrictions that apply to their project. To achieve this, engineers were interviewed, and accessible documentation was analysed.
2. To create a private cloud migration method, a thorough understanding of the underlying technologies and existing migration methods was imperative. To achieve this, a literature study was conducted on SOA, MSA, containerization, container orchestration, and cloud deployment.
3. To enable the comparison between private and public cloud, the context of the company needed to be understood. To build this understanding, engineers were interviewed to explain the context of the problem, the current architecture of the project, the current migration considerations and the expected migration problems.
4. The final part of the context analysis was to utilize the information on the context of the project and the information obtained in the literature study to analyse the factors separating public and private cloud.
5. The requirements for the new migration method were then deduced using the context and input from the project's experts.
6. Using information gathered in previous steps, a new migration method was created, fitting the requirements and adapting to the characteristics of migration to private cloud.
7. This migration method was then presented to external experts in software development, serving as a verification of the created solution. The external experts were deemed necessary as the experts of the subject company were directly involved in the creation of the method, therefore influencing their opinion.
8. As a second form of verification, the migration method was applied to the migration of the subject company. The analysis resulting from the application was then presented to the project's engineers. This served as a last verification of the results.
9. Lastly, the interviews and case study results were evaluated, and the effectiveness of the designed method was assessed.

The steps listed above show that this research project did not complete a full design and engineering cycle. This was caused by the domain restrictions imposed on the case study company, which is of a military nature, meaning that large parts of their products are classified. As access to the source code was restricted and this research aimed to keep its content undisclosed, it was impossible to see or edit the project's source code. This means that direct implementation of the method was not possible. Therefore, the research had to end with the verification of the design, while the actual implementation of the migration was left to the project engineers.

1.5 Structure

The structure of the report follows the three categories of the research questions. Chapter 2 lays out the theoretical foundation to understand the problem analysis, which includes the technologies and architectural patterns used. The last part of this chapter summarizes the current migration strategies identified in the literature concerning MSA and Cloud migrations, including migration path construction, migration plan construction and best practices from scientific literature and the industry. Chapter 3 presents the factors separating public and private cloud migration and the new solution addressing these factors. This solution is then applied to a case study in Chapter 4. After this case study, the method is evaluated using expert opinions and the results from the case study; these results are then discussed in Chapter 5. Chapter 6 concludes the thesis and summarizes our findings.

2 Background

To create a migration method for private cloud migrations, understanding the underlying technologies and development strategies of cloud deployments is critical. These consist of virtualization and orchestration. In modern cloud deployments, the virtualization of choice is containerization. These containers are then orchestrated using container orchestrators. Most cloud deployments also utilize some form of SOA; in most cases, the form of SOA is MSA. The software development in these projects is often structured after the DevOps development practice. After a sufficient understanding of the technologies, architectures and development processes is reached, the benefits of combining them for cloud optimization are explained. Lastly, the theory on how to migrate towards these technologies and concepts is discussed.

2.1 Service Oriented Architecture

Both Service Oriented Architecture (SOA) and MicroService Architecture (MSA) are styles of software architecture. Software architecture can encapsulate many things, everything from technologies and designs incorporated into the project, to how the organisation of the project is tackled. There is no exact definition for software architecture, but it contains all the decisions and information needed for the architects to create a successful software project [15]. Many different architectural styles exist, all with their associated benefits and drawbacks. This section will explain the advantages and disadvantages of SOA. MSA is an evolution of SOA that is primarily used in cloud deployments; its differences with traditional SOA are highlighted and explained.

The benefits and drawbacks must be clear to understand when to use SOA. As the introduction mentions, SOA tries to split an application into services that serve a specific goal. These services are deployed on hardware as their own isolated, virtualised processes. SOA is entirely different from a monolithic style of architecture. In a monolithic application, the isolation of parts of the application happens on a code level. With SOA, the different services are isolated on a process level. These processes communicate via network calls. The fundamental principle of SOA is that services serve customers. These customers can be other services or application users [42]. These customers can be oblivious to the implementation of these services, as well as their location and other specifics. This is achieved through services being explicit, implementation-independent interfaces, which in turn allows teams to independently change the implementation of the services without affecting other services. These definitions hold both for SOA and MSA. However, MSA takes this isolation one step further and isolates the entire services, including data ownership [51] [38] [44].

Most companies migrating towards cloud deployments utilise MSA, in part because of the extra isolation provided by the separation of the application in services [2] [39]. For this reason, the advantages and challenges of MSA are laid out in the following subsection. After the advantages are explained, the differences between SOA and MSA are listed, as not all companies can transform their software into MSA during migration or use SOA as an in-between step towards MSA. Therefore, it is helpful to have an understanding of both.

2.1.1 Advantages of MSA

Potential main advantages of MSA are:

1. **Scaling**: As services are deployed as separate processes, scaling the application is much more flexible than a monolithic application. Scaling for software can be modelled using the ScaleCube seen in Figure 1 [51]. Where monolithic applications mostly scale by X-axis scaling, duplicating the whole application, or Z-axis scaling, by copying the entire application and splitting, for example, the user base, MSA applications can also utilise Y-scaling. They decompose the hole application into separate parts and, if needed, scale those over the X-axis. This enables them to scale much more efficiently.

2. **Continuous development and deployment**: Fully separating an application in services that run as separate processes and have their own data stores makes them fully isolated. This means teams working on a service are not dependent on the developers of other parts of the application. This holds as long as the API of the services remain stable. Teams can then change software, code and dependencies without interfering with other teams. They can also test their changes simply by deploying the service to a test environment and running tests against the API [51] [44].

3. **Component isolation**: By separating the functionality as services instead of classes or modules, as done in monoliths, it becomes impossible for developers to bypass the service boundary. As now, the data between

components has to go through network components. This means the separations are automatically enforced and will not fade over time in development [51].

4. **Fault isolation**: In software, faults are unavoidable. If a monolith fails, the application fails in its entirety. For MSA, this is not the case. If a service fails, the failure is often isolated to a single service. This means only a part of the application might not work, or maybe only one of the instances of that service is down, meaning it still functions only with a little less capacity. Most of the time, it is also much easier to start a new instance for one service than to restart an entire monolith [51].

5. **Easier experimentation and technological freedom**: As mentioned in item 2 above, as the services can be developed in isolation, it is much easier to experiment with and test new technologies. As dependencies are used in many locations inside the monoliths, it is often difficult to swap them for newer technologies. As services are not directly dependent on each other, they are also free to select different technologies like programming languages, allowing developers to choose the best options that suit their respective engineering problems [51] [44].

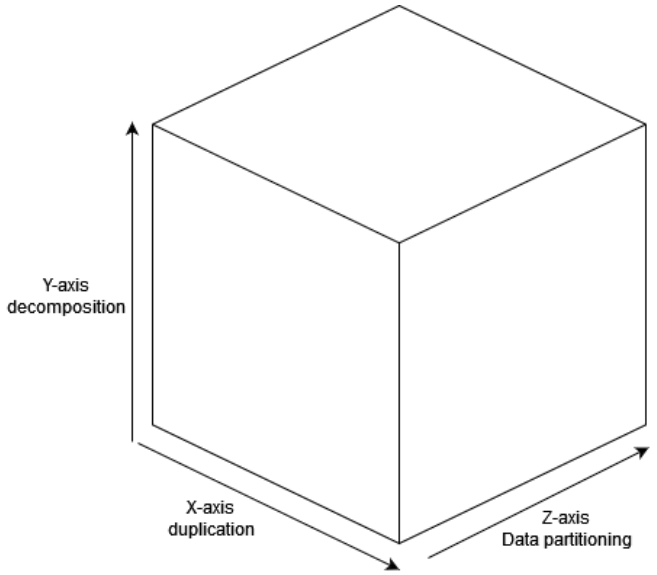

Figure 1: The Scale Cube

2.1.2 Disadvantages of MSA

Potential main disadvantages of MSA are:

1. **Additional complexity**: Microservices add complexity to software systems due to increased interprocess communication, handling system-wide failures, and data being spread across services [44]. The architecture's possibly numerous independent services demand careful management of moving parts in production to maintain system stability and debugging efficiency. Automation becomes essential for tasks like deployment, scaling, and monitoring, alleviating manual burdens. However, these complexities can lead to more demanding and complex system maintenance, requiring specialised expertise for efficient management. Overcoming communication overheads, load balancing, and service discovery hurdles is essential to ensure seamless service interactions.

2. **Additional latency:** MSA can lead to increased latency due to inter-service communication overhead [44]. The services must make network calls to exchange data, which requires extra processing time.

3. **Hard to find appropriate splits of services**: Splitting services effectively in a microservices architecture is challenging; if done incorrectly, it can lead to the emergence of a "distributed monolith" [51]. In this scenario, the system's services become tightly coupled, losing the benefits of independence and scalability offered by microservices. This coupling can result in harder maintainability and development and an increased risk of

cascading failures across the system. Correctly identifying service boundaries, ensuring loose coupling, and considering domain-driven design principles are crucial to preventing the pitfalls of a distributed monolith. This is quite difficult as it is more seen as an "art" than a science because there are no practical algorithms to solve this problem.

4. **New features across service boundaries increase in complexity**: Deploying features that cross service boundaries in a microservices architecture is inherently more challenging than deploying features confined within a single service or monolith [51]. When a feature spans multiple services, changes made to one service may impact others, needing careful coordination between the teams responsible for those services and limiting individual service teams' autonomy. Proper testing and integration become crucial to ensure that the distributed changes work harmoniously together, making continuous integration and automated testing vital components of successful cross-service feature deployments.

5. **Hard to determine when to adopt the pattern**: MSA solves problems that come with large or scaling software applications [51]. It enhances scaling, isolation and enables multiple teams to develop more efficiently. However, the negatives of added complexity are present even in small-scale projects. Therefore there usually is not a clear moment when to adopt MSA into a project.

6. **Integration testing harder**: Integration testing in a MSA becomes more challenging compared to traditional monolithic systems [38]. As services operate independently, testing interactions between services is essential to ensure seamless communication and data consistency, especially with new features that cross service boundaries. However, the increased number of services and their distributed nature require complex test scenarios encompassing various service combinations.

2.1.3 SOA compared to MSA

MSA focuses heavily on isolation of services. This means services can be deployed, developed and scaled independently [42]. These benefits are deemed important in cloud environments. Conversely, SOA has centralized data management, smart communication, and more extensive services. This means the communication often occurs via busses, and the amount of services is generally lower [51]. The impact of these differences is that SOA usually still relies on singular points of failure. The database and message bus are required for the entire collection of services. Also, there are fewer but larger services, so the dynamic scaling possibility is lower. On the other hand, the complexity and latency are often lower as the network consists of fewer services.

2.1.4 MSA in practice

As can be concluded from the previous sections, MSA increases the productivity of small teams while increasing the complexity of the overall project. Its strengths lie in teams' ability to develop rapidly and produce continuous software upgrades. To facilitate this, organisational changes and tooling are needed [25]. Switching to a DevOps style will ensure continuous delivery is possible. To effectively achieve this, automation of deployment is critical, mostly done using a combination of two tools. First, containerisation is used to make sure the deployment process is generalised for all hardware. Second, an orchestrator tool manages the deployment and monitors the real-time state, ensuring it matches the described state documented in the project repository.

2.2 Virtualization and Containerization

Virtualisations can be used to effectively divide computers' computational resources, allowing the hardware to be divided over different programs or operating systems [21]. Virtualisation is running a software system in a layer abstracted from the layers below. This can be the hardware layer, the operating system, or any other layer [53]. A suitable implementation of virtualisation needs the following key features [21]:

1. **Isolation:** As computational power is divided between different processes, some being possibly malicious, isolation is key. It is never acceptable for one process to negatively affect other running processes.

2. **Variety:** A virtualisation has to work properly on different hardware setups and operating systems, as virtualisation is mainly used to generalise the deployment of applications to different sets of hardware.

3. **Overhead:** Overhead is the duplication created when running multiple virtualisations on one machine. Because of the generalisation, some overhead is inevitable. However, it is vital to minimise this overhead as much as possible, as they increase storage and impact startup times.

In the past, virtualisation was achieved using virtual machines. This means creating an abstraction layer on top of the operating system to run virtual operating systems. However, this requires an entire operating system for each process, resulting in a significant overhead [53]. Many optimisations have been made for virtual machines, for example, Xen [21]. Still, the overhead remained larger than desired.

2.2.1 Containers

The most prevalent virtualisation option in cloud architecture is containerisation. Containers are an abstraction on the application level [53]. The container architecture can be seen in Figure 2.

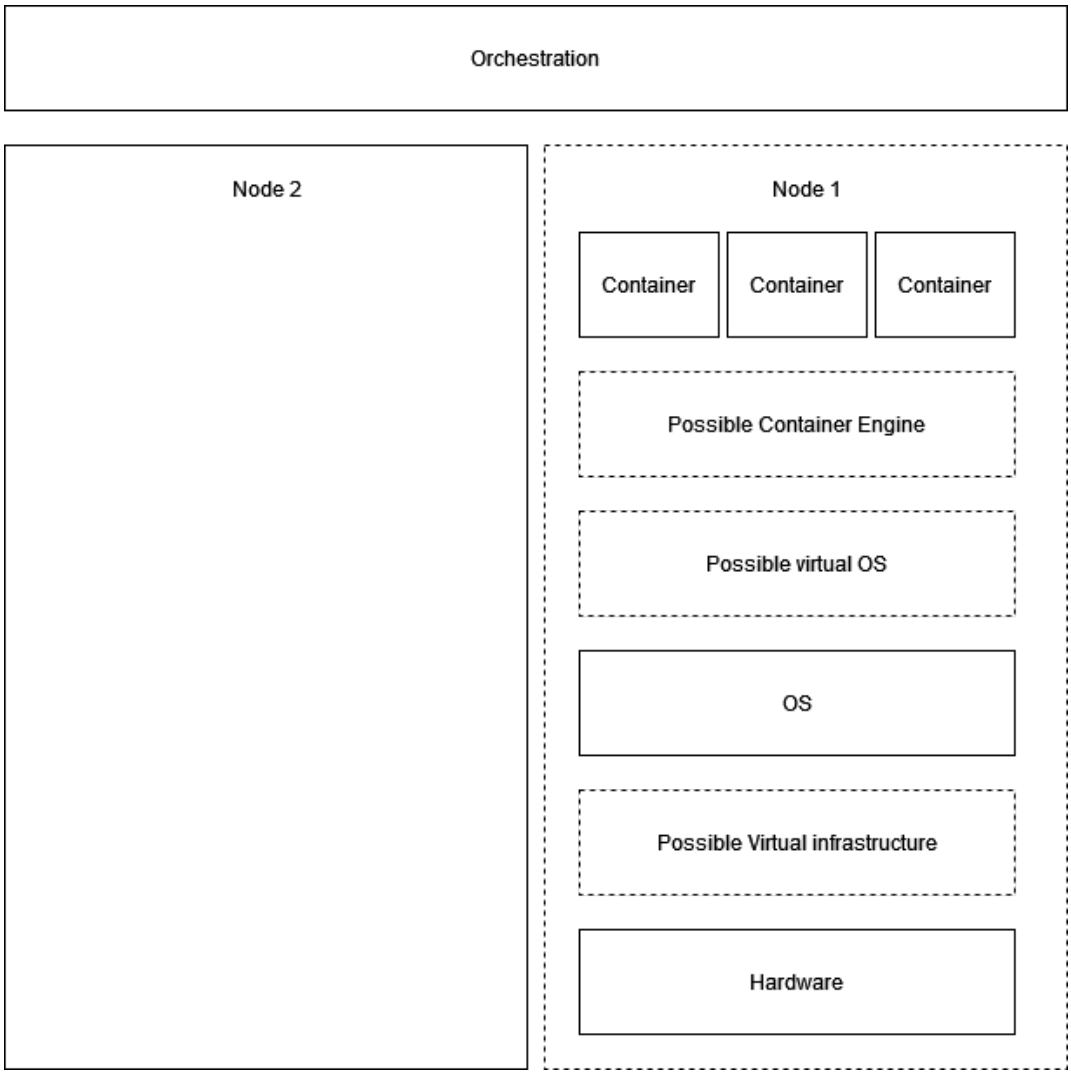

Figure 2: Architecture model for containers, including orchestration

The idea behind containers dates back to the early nineties [12]. The concept of a container is to run isolated processes using the kernel of the underlying operating system. It relies on Linux namespaces and Cgroups [22]. The container operates within its namespace, making it impossible for the process to see processes outside its namespace [34]. Cgroups are then used to restrict the processor and memory access to certain limits. This has the advantage of not allowing a malicious process to hog all hardware and allowing orchestrators to keep track of what nodes have space available for new processes. Most container solutions use a container engine to manage their containers. This architecture can be seen in Figure 2, and is the architecture used by Docker [16], the most used container solution [53]. Docker uses union mounts to stack read-only file systems, which allows container images to be extended upon and to share underlying file systems [49]. This feature is powered by overlay file systems built into the Linux kernel [34]. In these file systems, only the top layer is mutable. Mutations are done using a copy-on-write, leaving the

shared layers untouched. This allows containers to share file systems without affecting each other, saving download time and memory usage.

Containers must fulfil the virtualisation requirements and a few extra requirements [34]. In addition to isolation, variety, and reduced overhead, they need a versioning system and fast startup to meet orchestrating needs. The versioning system is so the orchestrator can keep track of what version is running, and the speed needs to be fast enough so in case of failure the new container is up in a matter of seconds.

2.2.2 Benefits and drawback of containers

The popularity of use and development using container technology instead of virtual machines is mainly caused by the many benefits it brings to the MSA architecture and cloud industry [53] [49] [24]. Containers can run on all most hardware with a small footprint, as the OS is not duplicated. These are the main reasons the industry values them highly [49]. Containers do come with security downsides. As they share the OS kernel and interface via network, they are vulnerable to data loss and malicious influence via the kernel.

2.2.3 Options

As of this current moment, Docker is by far the most dominant option used for containerising, as it is currently listed as being used in more than 80% of cases [5]. Docker offers various features, including many extendable images and a cloud container registry [6]. There are, however, alternatives like LXC [8] or Containerd [3]. Containerd is used a lot in combination with Kubernetes, as it is also a project of the Cloud Native Computing Foundation. Another option when an open source and rootless container option is needed is Podman [10]. It provides a deamonless runtime that can run containers rootless, this can be very useful for secure environments. It comes with its own container build tool called Buldah but can also interface with others.

To make sure containers can be used between different container runtimes, the Open Container Initiative (OCI) was created. This initiative specifies the runtime, distribution, and image specifications for containers that comply with it. This means containers build with Docker can be run using Podman and the other way around.

2.3 Container Orchestration

When using microservices running in containers, many small processes must be deployed and managed across multiple machines. As large applications now often include thousands of containers, and the interactions between these services scale exponentially, these tasks must be automated [37]. For this reason, container orchestrators have been developed. In the past, many big companies developed their own container orchestration solutions, like Borg, developed by Google [58]. Nowadays, many open-source solutions exist, some of which were built upon the lessons learned from previous in-house orchestrators. One example is Kubernetes [7], an open-source solution developed by Google.

2.3.1 Functionality

Container orchestrators can automate the deployment, maintenance, configuration and monitoring of microservice applications. Therefore, fully understanding and setting up an orchestrator can be challenging. First, it is vital to understand the reasons of using an orchestrator. It is then possible to explain the features of orchestrators. After the features are described, a general overview of the architecture of an orchestrator is given.

Google learned what benefits an orchestrator brings by using and building Borg [58], which was built to run their services and jobs on their private cloud. They identified three different benefits while operating Borg.

1. The cluster hides detail. It automatically handles failures and resource management, so engineers can spend time on other issues.
2. The cluster operates with very high reliability.
3. The cluster spreads the load effectively across many machines.

The orchestrator becomes an abstraction layer over all different machines in the cluster [17]. It handles resource management, scheduling and service management. This allows engineers to define a desired state with the requirements they need for the cluster, and the cluster tries to match this state in the real world continuously. This allows the cluster's desired state to be incorporated into version control systems.

To achieve the benefits mentioned, an orchestrator needs to be able to address four concerns [34]:

1. **Dynamic scheduling:** The cluster needs to span multiple machines, called 'nodes', and support many different services. The cluster must be able to determine where to schedule these services. The scheduler needs to account for the hardware needs of the application. When a node fails, all services running on it must be automatically rescheduled to another node.

2. **Distributed state:** The entire cluster needs to know what services are running where. Services must be able to communicate at all times, so every node needs to be able to direct network requests to the correct node. Also, the cluster control plane needs to monitor the current state to see if the state of the cluster matches the state described. Lastly, in case of a network failure between parts of the cluster, the cluster can restore the disconnected parts using state information.

3. **Hardware Isolation:** The cluster is tasked with isolating the containers with the differences in these environments so that they run identically on all nodes, unless they have specific hardware requirements, like for example a specific sensor.

4. **Multitenancy:** As the cluster can be used by many different users, the cluster needs to provide isolation, security and reliability to all tenants. This includes, but is not limited to, securing network, data and secrets and restricting resources of users.

To achieve this, orchestrators are generally structured in similar ways [52]. Developers add the instructions for the orchestrator into version control. The management plane of the orchestrator reads these instructions and schedules or alters the applications running on the compute cluster. As any of the nodes connected to the cluster can fail at any time, the management plane constantly monitors what is running where [34]. As every process running in the cluster, including all parts of the management plane, are objects controlled by the orchestrator, it can automatically recover from most failures. This general architecture can be seen in figure 3. To achieve highly scalable applications, the orchestrators also have advanced networking options [52], automatic resource allocation and load balancing.

As the orchestrator has full authority over the compute cluster, its security is vital. To secure an orchestrator and its cluster, the following is required [37]:

- The images pulled and executed by the cluster must be trusted; otherwise, malicious code can be easily executed.
- Fine-grained access control must also be in place, including identity validation.
- The nodes should be directly patchable in case of a problem.
- The attack surface must be reduced as much as possible.

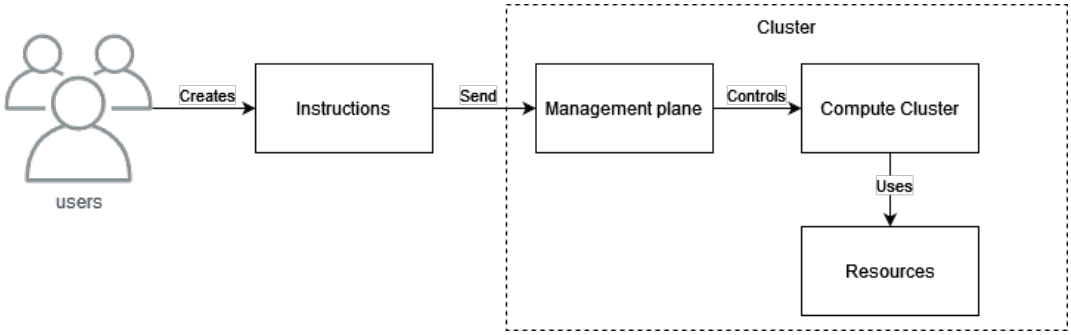

Figure 3: Basic architecture of container orchestrator

2.3.2 Organizational changes

Container orchestration obviously benefit from MSA and containerisation within the application. This means DevOps is also recommended when using a container orchestrator [18]. Specifically, the separation in teams responsible for their respective services and configuration as code are important. The cluster needs to be secure, so keeping all configuration secure and limiting access using access control is imperative [37].

Container orchestrators allow changes to be rolled out fast, if needed to only parts of the user base and allow them to quickly be rolled back if needed. Meaning that companies able to test in production can utilise the orchestrator to cut back on testing infrastructure. For companies where this is not possible, the orchestrators allow testing the new application in a sectioned off part of the cluster using namespaces. Overall, this shows that organisations can increase increment cycles and decrease testing infrastructure.

2.3.3 Kubernetes overview

Kubernetes is the market-leading option in corporate solutions and research [45]. It is an open-source solution focusing on being extendable and portable [7]. The combination of being extendable and the primarily used orchestrator resulted in a large ecosystem of extensions for Kubernetes. Many different versions of Kubernetes exist, incorporating different implementations for networking, storage or other parts of Kubernetes. Essential to the design of Kubernetes is that every component is described as an API object. This API remains stable over every implementation and version of Kubernetes [34] [50]. This means that configurations written for Kubernetes will remain valid for the foreseen future.

Kubernetes uses pods as the scheduling unit [34] [7] [50]. These pods contain one or more containers. Inside a pod, containers are connected directly using a local network. Pods are immutable once deployed, meaning if a container fails or needs to be updated, the pod is destroyed and replaced. Pods can be directly scheduled using the Kubernetes API, either using the command line tools (kubectl) or by providing a YAML file with all the pod requirements. This includes but is not limited to what containers to run, their specifications, hardware- and network requirements. To schedule and maintain these pods, Kubernetes has a lot of different components that can be scheduled, all scheduled as API objects.

2.4 DevOps

DevOps found its origin in 2008 to resolve the conflict between development and operations teams in cases where quick response time was necessary [27]. DevOps aims to unify the two silos of development and operations within organisations using automated development, deployment and infrastructure monitoring [28]. The resulting combined teams work on continuous operational feature deliveries. DevOps requires a culture shift within the organisation, which can be costly to implement. Nowadays, DevOps is widely adopted in software development, and there are many reasons for its adoption. Research shows that DevOps can lead to faster time to market, better software quality and improved productivity [26].

To achieve DevOps, automation is mandatory [28]. Build tools help with fast iteration, Continuous Integration (CI) integrates the changes from developers into the codebase with possible verification of the changes, and Continuous Delivery (CD) automatically delivers the code to the production environment. The CI/CD toolchains bring the vital characteristic that infrastructure can be treated as code. These automation options also heavily benefit from containerisation and container orchestration, as these technologies help generalise and automate the deployment procedures and accept configuration as code. These benefits are further enhanced by utilising MSA in combination with DevOps [19]. After adopting MSA, small DevOps teams can focus on their services and own the entire life cycle of these services, reducing their dependency on other teams.

Companies can adopt DevOps to different degrees. For some products, especially older ones, the cost of fully implementing DevOps is too high or implementation is impossible [41]. DevOps offers most for products that need frequent updates to stay competitive. Companies can adopt bridges between silos or semi-connected teams instead of entire DevOps teams. These options do seem to provide fewer benefits [41]. Companies with other environmental constraints, like the automotive industry, can still apply the principles of DevOps [28]. This shows that even when traditional DevOps is not possible, adopting parts of DevOps can still bring benefits. However, in these domains, especially when the company is not developing one singular product, introducing DevOps becomes impossible. This is discussed in Chapter 3.

2.5 Cloud Native

Cloud Native (CN) is a form of software development that directly designs software for utilising the cloud and maximise the provided benefits. This is typically done using MSA, containerisation and container orchestration [2]. The term is widely used to describe applications optimised for running on the cloud. Most CN application share common characteristics, these include [31]:

1. Operate at a global scale.

2. Scale well, often with thousands of concurrent users.
3. Built with the assumption that underlying infrastructure changes and fails.
4. Built for continuous integration and deployment.
5. Security is vital.

The cloud is a network of computational devices that offer services on demand. It functions as an abstraction layer over the individual machines, removing the need to interface directly with them [23]. Clouds can be fully public, offering Platform as a Service (PaaS) or Software as a Service (SaaS), private and hybrid, or a combination of the two. When these services are obtained from a third-party cloud provider, the first benefits experienced from the migration are outsourcing of maintenance of hardware in the case of PaaS up to possible outsourcing of parts of the software in the case of SaaS. Utilising public clouds, hardware usage can be scaled on demand. This is useful for applications that might grow rapidly or where traffic load varies over time. Utilising MSA can maximise this benefit. Scaling only the parts of the application that receive high traffic enables cost reduction and more efficient scaling [51].

CN applications do not have to be hosted on public cloud. Applications can run on private cloud setup, that still offer computation on demand, like Borg [58]. Applications that are optimised utilising the same principles as mentioned above can still be seen as CN applications. In essence, it is an approach of software development, focussing mostly on resilient and scaling software.

2.6 Migration

Migration is going from a current system, towards a new envisioned system. Migrations can span many changes, stakeholders and requirements. They need to be carefully planned and tracked. This is especially true when migrating towards MSA and CN. In these migrations the application needs to be rearchitected, new technologies added and possibly the organisation restructured. To reach a successful outcome, migration planning and strategy are vital [36].

Below, the theory on how to create these migration plans is presented. The information is attained from both the scientific literature and the software industry. It first explains what makes MSA and cloud-native migrations complex. Then it gives the theory of what a migration is and the specific phases involved. The information on the phases includes how to analyse the migration and create a migration plan.

2.6.1 Cloud and MSA Migration characteristics

The transition to CN is a multidimensional problem that increases exponentially with the complexity of the software [20] [37]. Additionally, as these companies often need to do quick iterations of their software to stay competitive [48], the migration towards CN also often has an additional focus to increase the agility and flexibility of the software [57].

Within a typical cloud migration the architecture of the project needs to be changed to MSA, containerisation and container orchestration need to be introduced and the project's organisation needs to be optimised for cloud development. As becomes evident from the theory above, each of these subjects are quite complex. Meaning, that during a cloud migration, planning on how to tackle these implementations together needs extensive knowledge and careful planning. This is why multiple migration methods for CN exist.

2.6.2 Migration methods

A lot of research is done towards migration plan creation [54] [47] [30]. Migrations are often split into three distinctive phases: (1) pre-migration, (2) in-migration and (3) post-migration [33] [35]. Each of these phases is elaborated on, as they remain similar across all migrations.

2.6.3 Pre-Migration

This phase starts at the first consideration of the migration and runs up until the actual work of the migration starts. The phase spans all the analysis needed to create a successful migration plan and the actual creation of the plan [33] [35]. The migration analysis in this work is based on the method presented in [36], depicted in Figure 4, and extended using other migration research. This method was chosen as the base, as it offers a clear structure for

the analysis phase of the migration. As the goal of the pre-migration phase is to create a concrete migration plan, the analysis must give all information needed to achieve this goal.

The analysis starts by analysing the organisation. The organisation dictates the available resources, acceptable risk, domain and overall goal of the migration [54] [33]. The resources analysed dictate what the migration is limited by. The migration can either be limited in time or cost. The limiting resource needs to be carefully tracked during the migration, as otherwise the migration is likely to fail because of running over budget [54]. The goals of the migration need to be clear, as this allows the planning to be created with the goal in mind and allows the migration to be evaluated after its completion. Especially, noting the quality-factors that are aimed to be improved, allows for accurately gauging the success of the migration. The quality-factors are also a useful metrics for other companies considering the same migration [30]. Lastly, the organisational analysis can result in constraints, set by for example risk or the domain of the company. These constraints need to be resolved before the migration analysis can continue [36].

After this organisational analysis, the application and cloud options are analysed together. For the cloud, a cloud provider needs to be selected that fits all the requirements of the project. The application needs to be analysed to determine how it needs to be changed to optimally utilise the cloud. Both these analyses can result in constraints. These again need to be addressed before the migration planning can continue. If the constraints cannot be resolved, the migration should be aborted.

Lastly, a migration plan is created. This is a complex task as it needs to be specifically fitted to the characteristics of the project, the goal that is aimed to be achieved and the complexity of the attempted migration. Additionally, the starting point for every individual migration is different. There are many different migration strategies possible [40], with some more complex than others. To create a project specific plan, the analysis done above is used. The plan is bounded by the limited resource and a migration strategy fitting the risk analysis is chosen. Then an actual concrete planning is created. The research has certain tools available to aid the creation of these plans. Situational Method Engineering [43] is one of these examples that can be adapted to this specific migration [20]. Overall, it is advised to create a plan with iterative steps. Intermediate analysis of the impact of each step can be used to evaluate and adjust the migration plan accordingly [1] [54]. This feedback cycle reduces the risk of the migration.

2.6.4 In-Migration

During the migration phase, the migration is executed according to the plan and steps it defines. During this phase, software is redesigned, developed and deployed [33]. The migration strategy dictates the size of the migration steps. This could be the entire application at once or small isolated services. It is important to keep the rest of the application consistent between steps, as otherwise it is not possible to measure the direct impact of the step.

The actions can also be defined by patterns [20] [36]. This allows method engineers to construct a migration process from an extendable method database. Engineers can then follow these methods and refactor the application in place.

During this phase, it is crucial to keep track of the cost expended for the migration and the progress [54]. This is crucial when evaluating the migration plan and whether it needs changing during the migration. For this reason, it is vital to do the cost and risk analysis beforehand. Without this analysis, there is no way to measure the progression of the migration.

If the data gathered by tracking the migration shows the migration does not have the desired effect, the migration should be aborted, as failed migrations are costly [54]. How to abort a migration depends on the strategy chosen to tackle the migration. However, it is always advised to prepare a rollback strategy if the migration is effecting the production environment [48]. This rollback strategy dictates how engineers revert changes made to rebuild the production environment to a state not influenced by changes made by the migration.

2.6.5 Post-Migration

There is little research focusing on this phase of migration. Certain engineers do not even define a specific endpoint to their migrations and consider it an ever-lasting process [30]. A recent study [33] has shown that in the post-migration stage of MSA migrations, the maintenance concerns of companies do not resolve as their scalability and flexibility concerns do. Also, the security gains no specific advantage. The performance concern only grows after migration. Using these metrics, companies can decide for themselves if, even without the improvement in the maintenance quality-concern, a migration towards MSA is still beneficial. This directly shows the value of these quality-concerns. It is a way to verify if the theoretical benefits provided by a migration are actually attained in the industry during migration.

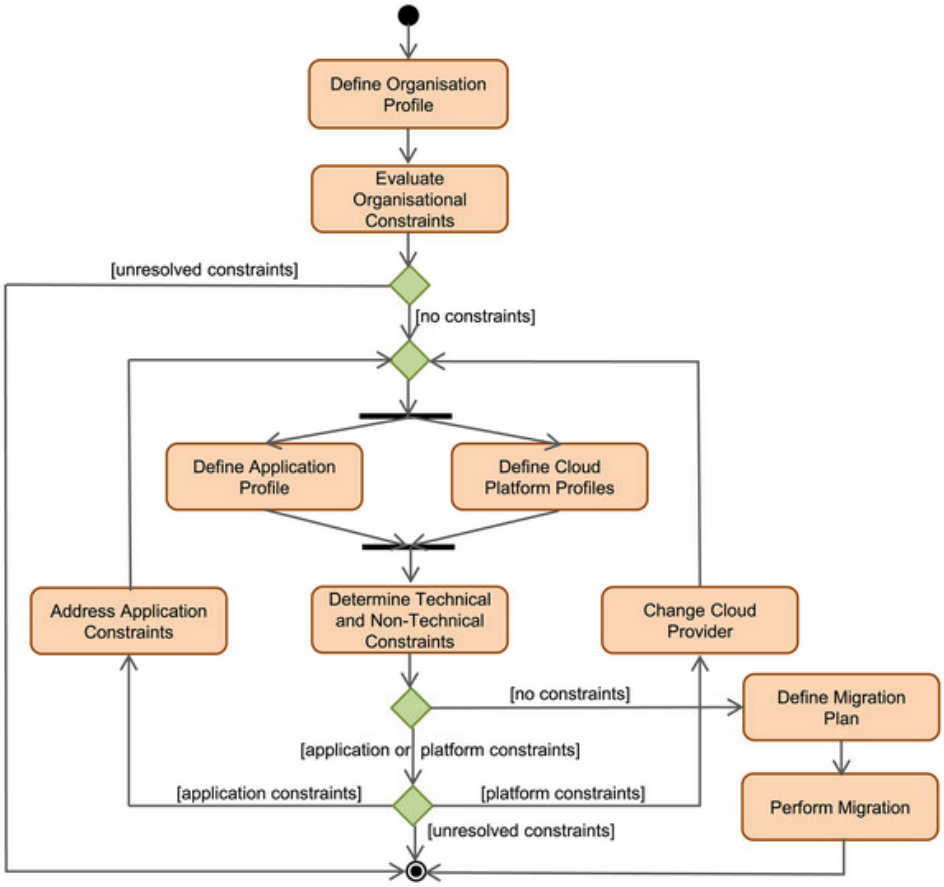

Figure 4: Situational migration description and process model from [36]

3 Private Cloud Migration Method

This chapter presents the migration method we developed, the observations that prompted its design, and how these observations were gathered. The observations were made at a naval software project, and served together with the theory presented in Chapter 2 as the basis for the migration method. The chapter first outlines the observation collection process. It then explains the different factors between a private cloud migration and a traditional cloud-native (CN) migration, and the implications of these factors on the migration analysis. Lastly, the created migration method is explained.

3.1 Research activities

During this research, observations about private cloud migrations were gathered and verified using two strategies. Firstly, the research was conducted at a company undergoing a private cloud migration, which served as the case study. Secondly, external engineers were interviewed.

3.1.1 Case study

The case study took place at a military company. Due to the classified nature of this case study, nearly all information was acquired in conversations. These conversations were not allowed to be recorded. The research took place at the offices of the company. The conversations were informal, allowing engineers to discuss their goals and vision for the migration. Some engineers, especially the lead architects of the project, were asked more frequently to verify the gathered information and help with reasoning about the implications. Lastly, the observations, reasoning about the observations and resulting advice were presented to the engineers of the company, serving as a last validation of the work.

3.1.2 Interviews

Experts of the CN software environment were also interviewed for external validation. These interviews were unstructured, allowing the experts to highlight parts of the migration theory they deemed important. The questions and transcripts from the interviews can be seen in Appendix A. These interviews served to validate the observations, check if generalisations for the observations were missed that did not apply to the case study company, and to check if they deemed the choices made in the created method, presented below, valid.

3.1.3 Method development

As public and private cloud migrations closely resemble one-another, a suitable migration method from public cloud migration was chosen. The selected method can be seen in Figure 4 and originates from [36]. This method is selected as it has a clear structure to the analysis phase, which is the main focus of this newly developed method. Private cloud migration do have certain factors which need more attention during the analysis phase. These factors are discussed in Section 3.2. The method is adapted to these factors to create a private cloud migration method.

3.2 Private Migration Factors

To adapt the selected migration method, the factors separating CN migration and private cloud migration need to be understood. These factors, observed at the case study company, are listed below. The implication of these factors is then reasoned about. These implications form the base of the adaptations to the original method.

3.2.1 Limited outsourcing

The first and most apparent factor is the limitation in the outsourcing of the infrastructure. Many works on CN migration start their reasoning from the benefits obtained from not having an infrastructure team and the cost benefits this can bring [36]. Additionally, if the application is migrated towards MSA the dynamic scaling properties of the cloud can result in even more cost benefits. This dynamic scaling and global properties of the cloud can allow applications to scale fast and cost-effectively around the world.

Some companies are however limited by security, privacy laws or by the obligation to deliver their servers as part of the product, which typically happens because of security concerns. These obligations limit their ability to outsource their infrastructure and therefore limit the cost and scaling benefits. This means the hardware stays the responsibility of the company but also remains in full control of the company. It also means other properties of

the cloud, like multitenancy and arbitrary hardware, can be eliminated. Some companies can still achieve benefits with dynamic scaling applications on private clusters, but this needs a sufficiently diverse computational load, like in the case of Google [58].

3.2.2 Product delivery

The traditional CN migration looks at cases where the company operates using continuous software delivery [36]. These companies continuously serve their users while improving their product. This setup allows companies to quickly evolve their software and stay competitive. However, not all software companies operate this way. Certain companies, for example in the military domain, deliver products that include the infrastructure the software runs on. These products often no longer evolve when shipped to the customer. In some cases, as with this case study, the entire operation phase of the software is not in the control of the company. Additionally, these products can often be configured to the needs of the customer, meaning that there is not one continuous product, but configured deliveries. These changes combined mean that the development and operation cycle is incomplete. Such a broken DevOps cycle is illustrated in Figure 5. This figure shows the cycle splitting at the delivery stage, however in practice this can be in other places as well. For example, the product could be configured to the needs of the customer or already be packaged as different products. This means that some concepts of DevOps can still be achieved, however, the continuous cycle is never possible.

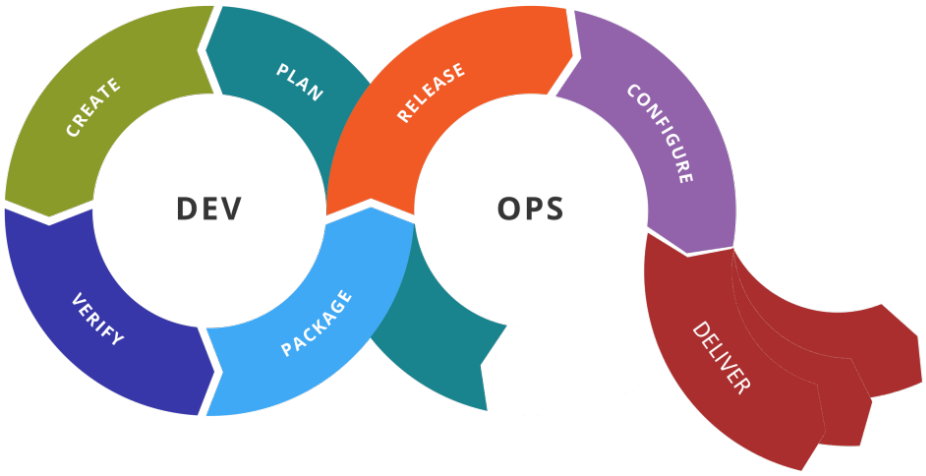

Figure 5: Illustration of broken DevOps cycle

3.2.3 Different focus

Most traditional CN companies value the flexibility and agility of their software highly. These are also the leading concerns when migrating towards MSA [30]. For certain companies, flexibility and agility are not the top concerns for their software project. This holds especially for companies developing high assurance systems, like military applications. These companies often have their main focus on the integrity of the software, requiring extensive testing setups. CN companies often shift towards reducing their testing setups and move towards testing in production [48]. For companies with strict testing requirements, this is not or only partially possible.

3.2.4 Implications

To account for the factors listed above, their impact on the migration needs to be understood. Below, the implications of the three factors are explained:

- **No outsourcing:** This factor gives insight into whether the infrastructure can be outsourced, partially outsourced or not outsourced at all. If the project needs to be hosted privately, it should be seen if a private computation on-demand setup, like Borg, is beneficial, as it has a profound impact on the benefits of dynamic scaling. If no outsourcing is possible the infrastructure needs to migrate in-house, resulting in a migration

happening alongside the application migration, with its own stakeholders, constraints and goals. These vertical organisational structures, referred to as silos, migrate side by side, so that the overall product migration now consists of multiple sub migrations.

- **Product delivery:** This factor affects the possibility of transitioning to DevOps. Depending on how the product is delivered and who is responsible for the operation phase of the software, the cycle could be impossible to fully implement. This means that certain features relating to orchestrators are impossible to utilize. This can also have a negative influence on the effectiveness of MSA, as MSA highly benefits from DevOps [19]. The difference implies more analysis is needed on the extent DevOps can be applied, how the company still benefits from DevOps concepts and the effect on the migration towards MSA.
- **Different focus:** A difference in the main quality concern, again, works against some of the features of the cloud. If the priority of the developers is integrity, a lot of orchestrator features will never be used. It also means that the possible, project-wide complexity MSA brings is a harder trade-off to make.

Overall, these three factors increase the complexity of the analysis phase of the migration. New stakeholders are introduced and the overall benefits, especially relating to cost and MSA, are no longer always present. To account for the implication of these factors, the selected migration method has been adapted. The resulting method is presented in Section 3.3.

3.2.5 Other private cloud driver

The implications of the factors explained above arise from the assumption that a product needs a private cloud deployment due to environmental restrictions. This shifts the reasoning towards high assurance systems, leading for example to the different focus factor. However, more and more companies are opting for private cloud for financial reasons [13] [14]. Running their more standard software applications on public cloud infrastructure is no longer cost-effective.

For these applications, the factor impact is relatively different. If they choose the private cloud deployment options, they are possibly still able to achieve the continuous deployment benefits brought by cloud. They are still able to utilise the method created in this work, however their migration resembles a CN migration even more. Therefore, some of the steps added in the new migration method might redundant, as the migration of high assurance systems remains the main focus of this work.

3.3 Migration method

The method we propose is an adaptation of the method from [36] seen in Figure 4. The method was adapted based on the implication of the factors differentiating private cloud migration and public cloud migration. The new migration method increases the attention towards the migration analysis phase, as the attainable benefits of the migration change, the implications of the domain analysis increase and the amount of stakeholders also increase. The new migration method, is shown in Figures 6 and 7, and is explained below. The explanation focuses mostly on the adaptations made in this work. The steps that remained unchanged are not explained in detail here, further elaboration can be found in Section 2.6.3.

3.3.1 Migration analysis

The method, and therefore the migration analysis, starts at the starting point of Figure 6. Here, the organisation profile and organisational constraints are analysed. If this results in a private or partially private cloud migration, the method moves on to the private cloud migration diagram, seen in Figure 7.

The analysis of the private cloud migration starts by assessing the factors introduced by the move to private cloud, discussed in Section 3.2. This analysis aims to find out to what extent these factors apply. Once there is a clear picture of the factors and their effect on the migration, a goal for the product migration can be created. The product is defined as the infrastructure and applications deployed on the infrastructure. Each application and the infrastructure itself has a corresponding stakeholder group. A vertical organisational structure here defined as a silo. These silos are present within the company even before the migration, and need to all agree on the overall goal of the migration. Therefore, the analysis of private implications and creation of product goals should be performed by a set of engineers from all silos. This promotes consensus about the entire product migration.

After consensus is reached for the overall goals of the product migration, the analysis splits up towards the respective silos. Each individual silo now needs to analyse how to migrate their part of the product and how much

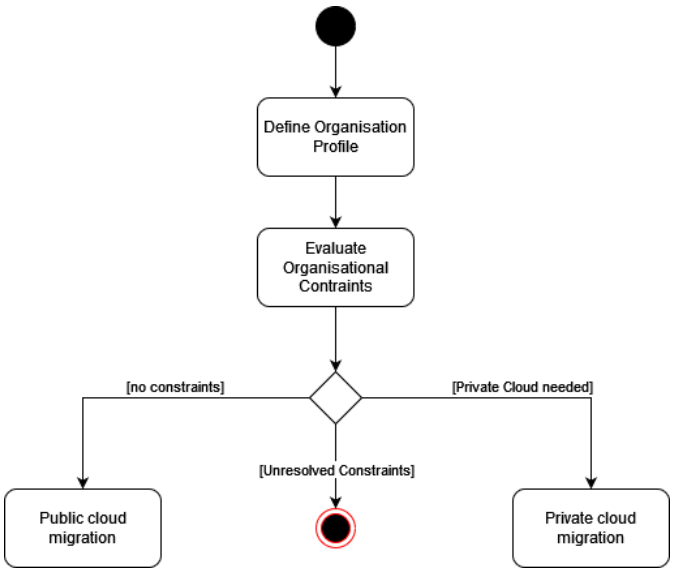

Figure 6: Decision diagram for choosing what migration model to choose

work this will entail. They also need to find what constraints block the migration for them. Lastly, each silo needs to look at what the other silos are planning to do and what their subgoals are. This last step promotes consensus between silos further, as the perspectives of the other silos are understood. The addition of these steps again promotes communication between the silos. This improved communication is promoted to make more use of the increased control over the infrastructure. In private cloud, the infrastructure can take any form, as it is still in full control of the company. If the infrastructure teams and applications teams work together, they can possibly help solve constraints of one another.

After the constraints are resolved, the migration is evaluated again. The product migration goals are clear, and each silo has a clear analysis of their respective part of the migration. As the migration analysis is split over different silos, a final evaluation moment is added to the method. In this evaluation moment, engineers of each silo again analyse if the product migration is worth the investment. Together, they either decide to proceed with the migration or cancel it.

3.3.2 Observations

With the case study, we could observe the factors and challenges faced in private cloud migration for high assurance systems. The project analysed as the case study is a software product deployed to naval ships. There is strictly no outsourcing possible, as the infrastructure is part of the product. After the product is deployed to the ship it can no longer be altered and monitoring for development purposes is also strictly off limits. Each ship also gets a specifically tailored solution, which is adapted to the hardware and the specific needs of the ship. Lastly, as the software operates in the military domain and is responsible for operating the weapons aboard the ship, the focus of the project is fully on the integrity of the software.

The observations at the company also contributed to the adaptations made to the migration method. At the start of the research, the infrastructure team had already started the migration. As the application teams had to comply with the changes in the infrastructure, this forced them to start their own migration. One of the application teams opted to start anew, fully embracing the new infrastructure. The other team was unsure how to tackle the migration and what their migration goals were. Each team wanted to improve the product, however, each had a different view of what this entailed. It was observed, by us in performing the analysis together with the engineers, that acknowledging the existence of different silos with their own respective goals helped in the migration analysis. This led to the adaptations made to the method.

3.4 Migration strategies

Just like in the migration to public cloud, a migration to private cloud entails changing the architecture of the software. A private migration, however, also entails a migration of the infrastructure. As multiple parts of the product needs to be reworked, a migration strategy needs to be chosen. This migration strategy dictates what the

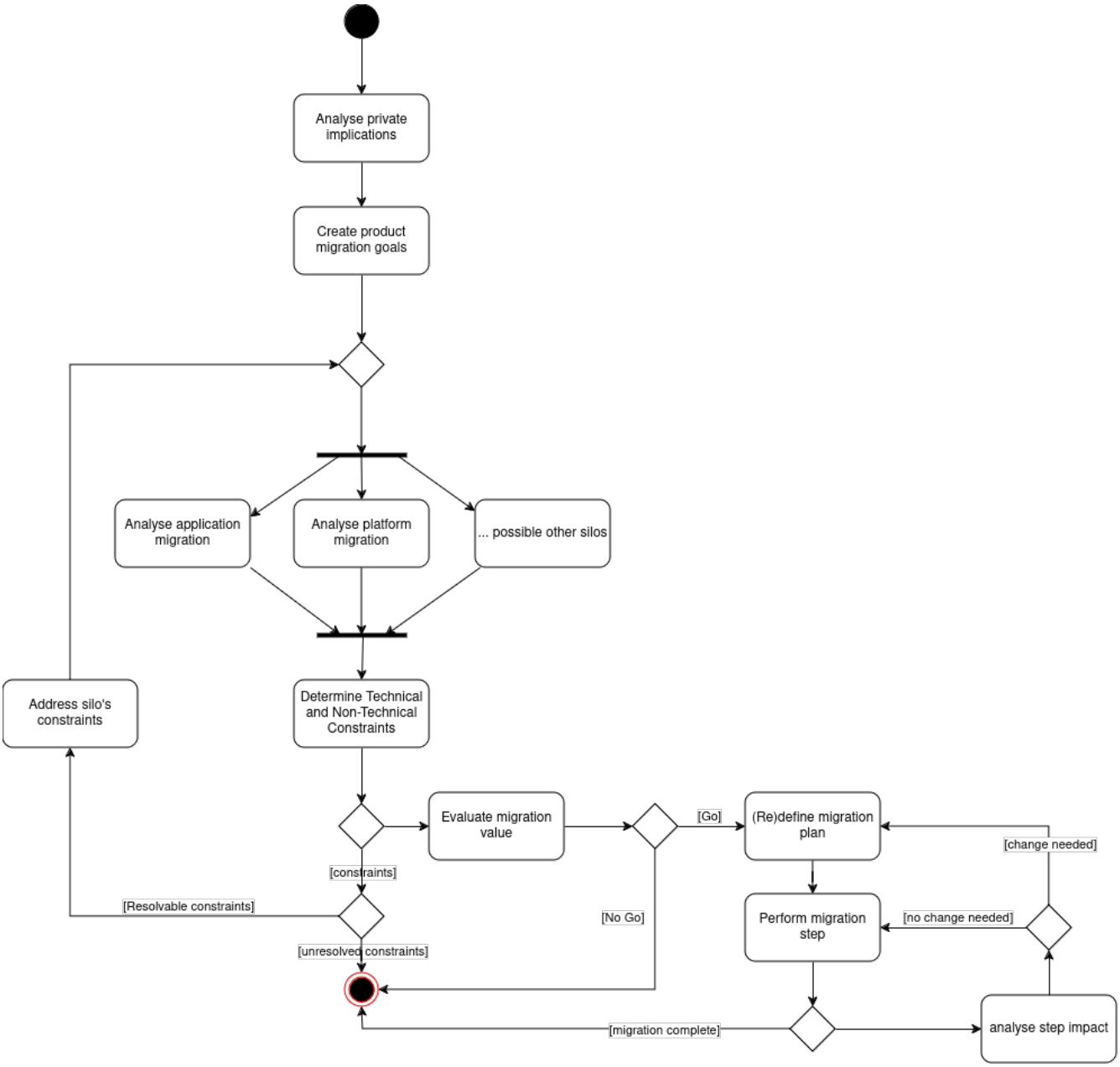

Figure 7: Situational migration description and process model for private cloud migration

migration plan looks like. We found four migration strategies in the literature and grouped them in the migration matrix seen in Figure 8. Moving along the horizontal axis increases reward as time and cost potential are high, but the risk is also high. Moving on the vertical axis increases technological freedom, but also the complexity of the migration. The four strategies were also presented to the experts, who acknowledged them and deemed the migration matrix a helpful visualisation.

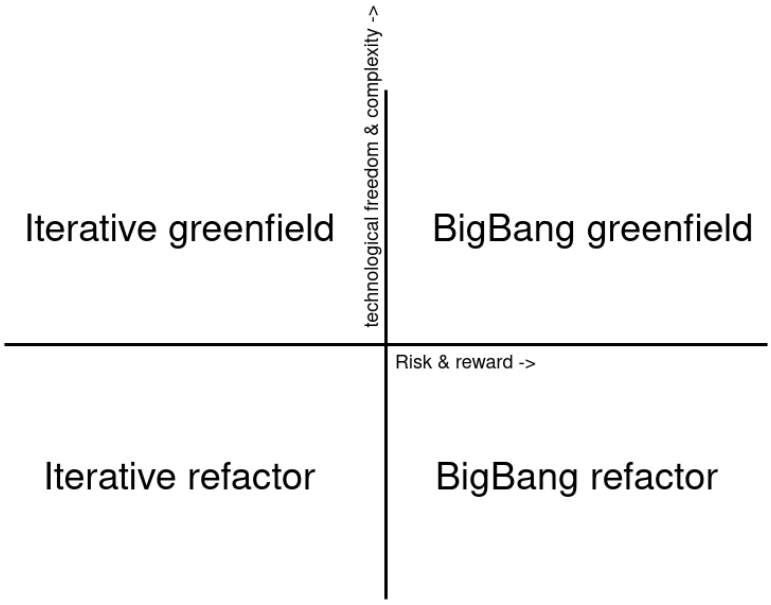

Figure 8: Decision matrix for migration strategies

3.4.1 BigBang greenfield

This strategy is familiar, and consists of simply starting anew. It gives optimal technological freedom, but is quite a risky endeavour. It is difficult to achieve all the same requirements with new technologies or architectures, and difficult to make a concrete plan. To create a plan for this, one should tackle the problem as any new starting software project, however, using the requirements from the existing to-be-migrated system. Because of the risk, this strategy is not advisable for large software systems.

3.4.2 Bigbang refactor

This strategy is very similar to the previous one, however instead of creating a new system, the entire project is refactored at once. This decreases the complexity, as the parts not changed stay intact, however, it decreases the technological freedom. As all the changes happen in one go, the strategy is risky, due to the lack of feedback cycles. It does however save time and can be useful for smaller projects or migrations. To create a plan for this strategy, one should first analyse what needs to be changed in the migration, decide how to tackle these changes and then perform them.

3.4.3 Iterative refactor

This strategy is widely used for CN migration. A plan consisting of small changes is created, and each change is then performed consecutively. As most orchestrators have the capabilities for rolling back changes and routing calls, the changes can be tested inside the cluster and rolled back if they are not satisfactory. The feedback cycles increase the time the migration takes, however it also decreases the risk if combined with monitoring and rollback strategies. To create a migration plan for this type of migration, a list of all proposed changes is created. This can, for example, be done using DDD's event storming [29], isolating services which are migrated one by one, or using method engineering with a method repository created for CN migration [36]. As the changes happen within or next to the original application, the migration is not free of the original technological choices.

3.4.4 Iterative greenfield

The iterative greenfield resembles the previous strategy, however, requires the inclusion of an Anti Corruption Layer (ACL), which is a concept from DDD [29]. This ACL converts calls from the old system to the new system, allowing engineers to develop a new system without the burden of the old system. The migration is then performed in the same manner as in the case of iterative refactor (Section 3.4.3), by doing small independent changes. An overview of this strategy can be seen in Figure 9. The ACL is however a complex piece of technology that sits at the heart of the application. If engineers deem technological freedom necessary and require a low-risk migration strategy, then this strategy should be chosen. First, the characteristics of the new system should be designed. This gives an idea of how the ACL should transfer the calls. Then the migration plan can be created further using the same tools as mentioned in Section 3.4.3, making sure to update the ACL in between each step.

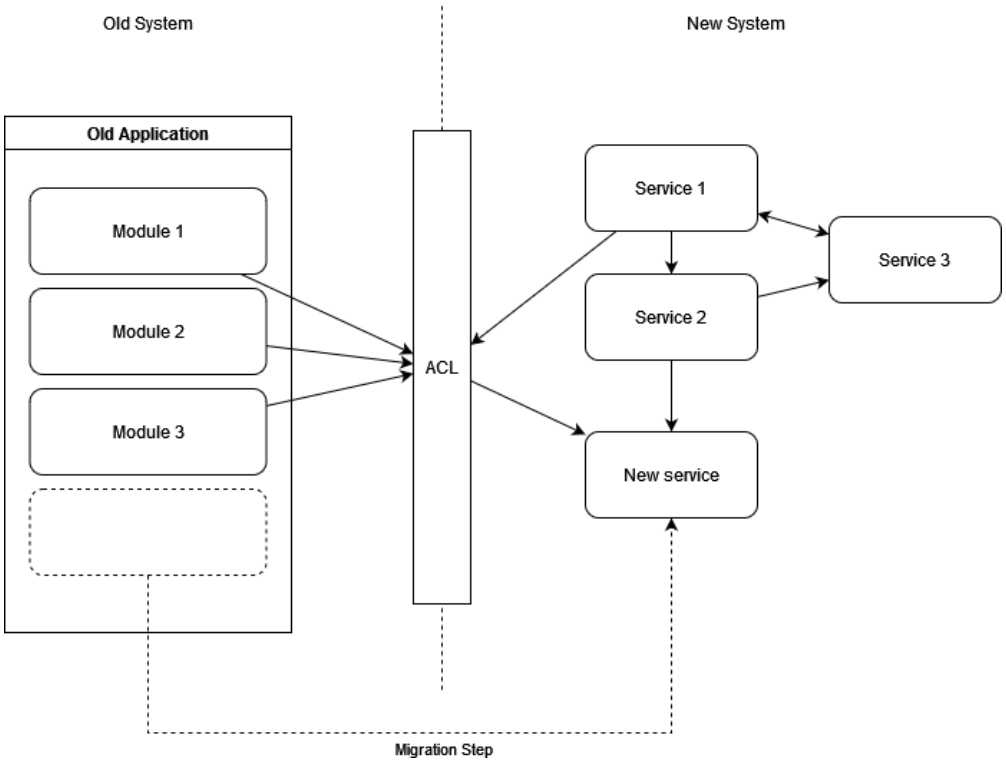

Figure 9: Typical application architecture during migration with ACL

3.5 Applying the migration method

The method gives a structured overview of how to perform the analysis and focuses extra attention on the factors that set apart private cloud migration from moving to public cloud. From the analysis, it becomes apparent that certain aspects of the migration can be limited by the restrictions that force a move to private cloud. Especially, MSA and DevOps can be limited in their implementation. The migration method diverts more attention into the value they bring for the private cloud migration.

The migration method also puts a lot of emphasis on the cooperation between silos. As the infrastructure of the product stays the responsibility of the company and needs to evolve as well, knowledge about it needs to stay in the company. The people responsible for different areas of the product need to determine together what the goals are for the migration.

The increase in the complexity of the analysis also directly shows the need for evaluating the value the migration brings to the overall product. As there are more silos, and possibly fewer benefits to be obtained in the migration. Once the decision is made to proceed with the migration and the goals are clear, a migration plan should be created. The strategies listed can give shape to this migration plan. For large private cloud migrations, an iterative strategy is advised, as it reduces the risk of the migration. The nature of this strategy allows engineers to reflect on the changes made, analysing if the benefits that are strived for are actually attained. The migration should then be performed according to the migration plan. After the migration, the whole process should be evaluated, as it

happens with any other migration. Especially, the concerns that were set out to be addressed should be evaluated to assess if the intended improvements were achieved [33].

4 Case Study

This chapter presents the application of the migration method, discussed in Chapter 3, in the case study. It starts by explaining how the research started within the project, depicting the problems needing a solution. Here the project context is also explained, as this is necessary for the migration analysis. Then the analysis itself is performed according to the migration method. To keep the chapter as short as possible, only the organisational and project information is presented that is critical to the analysis. Lastly, the analysis is applied to the original problems and advice for the future of the project is given.

4.1 Project introduction

The application in this case study is Tacticos, which is a Combat Management System (CMS) developed by Thales [11]. The software gathers, processes and displays data from all connected sensors aboard naval ships ranging from patrol boats to destroyers. It then collects operator input and takes actions accordingly through interfaces with machines or weapons aboard the ship. The software can also interface with other software products, both other applications developed by Thales and externally created software applications. The software is deployed on over 200 ships and used by 25 navies worldwide. More than 200 engineers are working on the project concurrently. The number of engineers is set to expand rapidly in the coming years to fulfil the requirements for the large order of six F126 Frigates by the German Navy. This delivery includes the software running on the ships, the infrastructure and many training facilities, meaning the product includes the infrastructure the application runs on. The customer can also configure the application depending on, for example, what sensors or weapons the software interfaces with. As the project is located in the military domain, the project also includes extensive testing setups designed according to the V-model [56].

The infrastructure teams of the product started a migration to include containerization and container orchestration into their infrastructure. They had clear reasons for this migration and a plan how to achieve their migration goals. They started the migration independently of the application teams and indirectly forced the application teams to migrate towards deployment on this new infrastructure. A different software project at Thales, Fire Control Systems (FCS), started this migration by following the BigBang Greenfield strategy. The CMS team foresaw some problems with adapting their software. These problems served as the basis for this research.

4.1.1 Foreseen problems

This case study started with the analysis of the problems foreseen with adapting the CMS application to run on the new infrastructure. The move was scheduled to happen relatively quickly, in a time span of around two years. The lead architects foresaw certain problems with this adaptation and needed solutions for the problems at hand. These problems are discussed in the next sections.

Failover

As the current infrastructure already spans a cluster of machines aboard the ship, it already has orchestration and failure-handling solutions. The current orchestration is an in-house developed application that can orchestrate the cluster using DDS [4] and OSGi [9]. The way it does this can be seen in Figure 10. The orchestrator, displayed in the bottom right corner, schedules a tracking process. This process then sends a message via DDS to OSGi in a Java Virtual Machine (JVM) that is supposed to schedule the to-be-scheduled application. OSGi then enables the specified Java bundles to run the specified part of the application. These steps happen in sub-second times. When needed, there are even solutions to decrease the failover time even further.

On the other hand, Kubernetes works with redundancy. This means multiple instances of an application part are run simultaneously, and the load is balanced between them. This load should never max out. Therefore, if one fails, the others can bear the load until a new instance is started. This means the startup time only has to be faster than the average failure time. Kubernetes often has an instance start time in the order of seconds or more.

If the application is ported to the new architecture, including Kubernetes, either redundancy or fast failover needs to be introduced. As the migration is limited in time, a complete re-architecture of the software is not possible, nor is the removal of the current technologies. This means that either Kubernetes needs to be changed for fast failover without redundancy, going completely again the design philosophy of Kubernetes, or redundancy needs to be introduced into the current application. The latter option would require load balancing through DDS or introducing another middleware that natively supports load balancing.

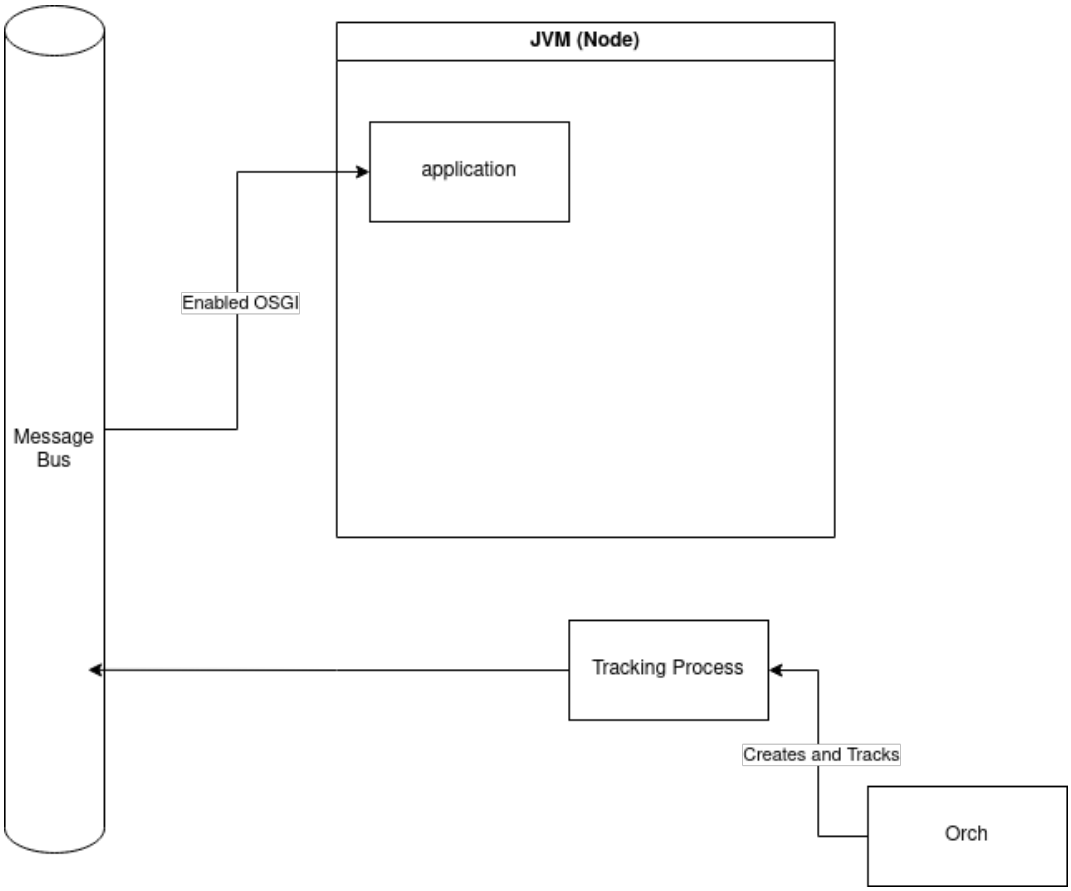

Figure 10: Functionality of the current orchestration solution in Tacticos

Patching

Currently, developers patch the software while it is being tested. The OSGi containers all have a shared platform library, referred to as the platform. Engineers can patch this platform to test the effect of their changes. Both the patching and the platform as a whole introduce problems. If the platform is added to the containers, the container needs to be rebuilt for every patch done by the engineers. Additionally, as the platform is added to most containers, every time it changes all these containers have to be rebuilt. This drastically increases built time and version storage. If we mount the platform into the container, the previous problems are resolved. However, now not all code is contained in the container and the running software can be changed without creating a new container, which goes directly against the best practices of container development [34].

Container introduction

The last foreseen problem is related to the previous one. The testing infrastructure is modelled after the V-model [56], seen in Figure 11. It is currently unclear at what level of the V-model the containers should be introduced. If the V-model is explored bottom up, with the knowledge that building some containers takes more than an hour, a reasoning for the introduction of containers can be created. If the containers are introduced at the unit testing level, engineers need to wait more than an hour for feedback on their changes. Feedback from unit tests needs to be as fast as possible, so an hour is unacceptable. Moving up one level, the subsystem is tested. Here the patching problem is at play, with the addition that the feedback still needs to be fast. Furthermore, at this level and the previous one, it is not known how many subsystems are running in one container. The deployment is dependent on the infrastructure requirements set by a project, as the project can require a single machine or a hundred. There is no way to test all different configuration options at this level, so a general container would have to be constructed. This is against the container best practices. On the other hand, if the introduction of containers is done at the next level of the V-model, new changes are introduced at the product integration level. Testing the containers at this later stage can increase the time it takes to fix problems found, as feedback cycles become longer. MRS testing takes longer than the previous target and unit testing, so this means feature development might slow down.

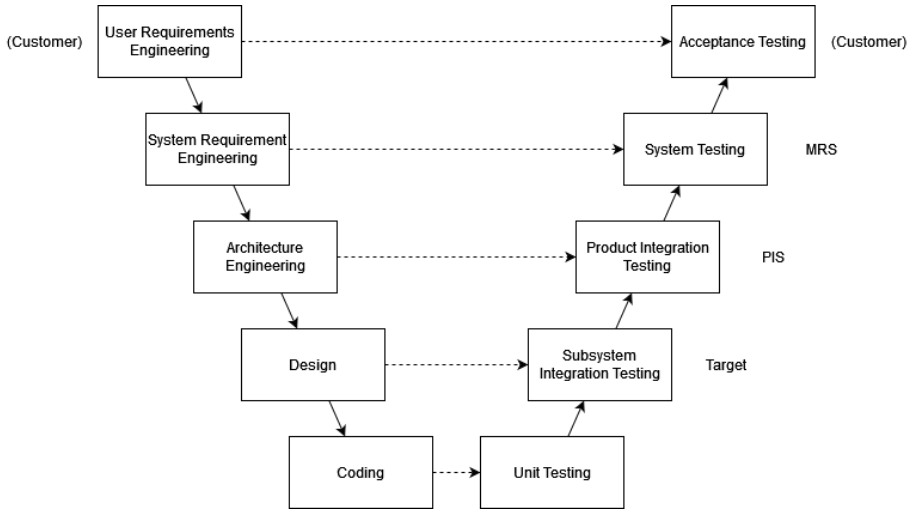

Figure 11: Traditional V Model with Thales system indicated

4.1.2 Strategy

The analysis shows that there are no perfect solutions. Completely reworking the software to the new infrastructure is not directly possible, and choosing between other options with drawbacks needs a vision of the future. This is why the migration analysis was conducted. Analysing the migration should result in a clearer picture of what the end goal is, and what should be done in this first migration, and therefore gives argumentation to what solution is most suitable for these problems.

4.2 Migration analysis

Here, the migration method as presented in Chapter 3 is applied to this case study. There are, however, some restrictions that apply to the analysis. Because of the military domain, much of the project is classified and could therefore not be accessed for this work. This means much of the documentation and all the source code was not available for us. It also means that conversations about classified parts of the projects were not possible.

For this reason, the migration analysis is fully based on observations from conversations and might not be completely objective. It does not function as an example of a complete migration analysis, as much of the cost and code analysis were not possible. Instead, it serves as a verification of the applicability of the described method. Only the migration analysis part of the method is possible to verify in this case study. As the classified nature, size of the project and time constraints limit us from creating a migration plan and tracking its progress and effectiveness.

4.2.1 Context and Scope

This step consists of performing an organisational analysis and identifying the goals of the migration. The analysis highlights the most important information from the context. The migration was started by the infrastructure team. The engineers of Combat Management Systems (CMS) now have to migrate their software towards the new infrastructure. This migration is the focus of this case study. They foresee problems with the migration and have no clear vision of what they ultimately want to achieve. The migration analysed here is limited in time, as it is part of the next big delivery of the product, set to happen in the next two years. During this time, the software is also heavily extended with new features to satisfy the needs of the customer.

The next thing according to the method is to analyse to what degree the factors apply.

1. Outsourcing the infrastructure is strictly not possible, as it is part of the product, installed on the ship. Dynamic scaling on the ship is not an option either, as the power, cooling and space requirements of the infrastructure are fixed.

2. Continuous delivery of software releases is also not possible. The application's features and deployment configurations are project dependant, and the operation phase of the software is completely out of the control of the company.

3. As the domain of the project is military, the focus is fully on the integrity of the product. There are strict testing requirements, which have to be followed.

As the migration originated from the infrastructure team, the overall product goals were never directly analysed. The new infrastructure solves certain problems that the infrastructure team has, however if the specific solution they chose to solve these problems is beneficial to all involved silos is not clear. Therefore, there are product migration goals, however a consensus between silos about these goals has never been reached. To analyse this migration further, the individual migration of the silos should now be analysed.

4.2.2 Infrastructure

The infrastructure team is the driver of the changes. They have already analysed their proposed migration, defined objectives and have already started the migration. The goals they set out to achieve with this migration were identified to be:

1. **Move to open-source:** this goal aims to phase out in-house developed solutions, like the current orchestrator seen in Figure 10. In doing this, the infrastructure team aims to decrease the personnel onboarding time, as engineers can be hired who are already trained in open-source solutions. Additionally, this will allow them to implement existing solutions created by the open-source software community, saving time and cost.

2. **Add abstraction layer over hardware:** Kubernetes functions as an abstraction layer over the nodes in the cluster. This means that once a node is connected to the network, Kubernetes handles the node. Consequently, engineers barely have to interface with individual nodes any more.

3. **Create more flexible infrastructure:** by implementing containerisation into the project, the infrastructure is able to run every application that can run inside a container. Also, as Kubernetes runs on most hardware, the cluster is also more flexible on what hardware it can run.

4. **Reduce service time:** the introduction of the abstraction layer makes updating the entire system simpler. As new versions can be rolled out using Kubernetes, the time in port can be decreased. This is highly desirable in the naval systems, as the ships are required to spend more and more time on missions.

They have decided to attain these goals by working towards a new infrastructure, including containerisation and Kubernetes. They have already started the migration and are well underway in developing this new infrastructure.

4.2.3 CMS analysis

For CMS the migration has not yet started and this is the first analysis of the migration that took place. The goals for their migration were gathered from conversations with the engineers. These goals are the following:

1. **Decoupling:** the current application suffers from a lot of coupling. Decoupling the application is set to improve the maintenance of the software and increase the development speed of new features.
2. **Technological freedom:** Adding containerisation to the project decouples the operating system from the software. Parts that are not ready to update can be kept on the current operating system and the rest can be updated. As of this moment, this update has to happen for all parts at once. The team is also more flexible to add external software to the project that can already run in containers, as Kubernetes can now handle the integration.
3. **Security:** With the added isolation introduced by containerisation, the security of the application is expected to be improved.

Because the migration is happening alongside the development of new changes for the next release of the product, the risk of the migration must be low, and no major architectural changes are possible. This resulted in requirements for the migration. These requirements were obtained from the project's lead architect.

1. Idle time of the developers must not increase significantly with the introduction of Kubernetes and containers.
2. Developer satisfaction must not decrease with the introduction of Kubernetes and containers.
3. New development procedures must be based on the best practices found within the software community.
4. The downtime after failure must not increase.
5. All current technologies have to be functional in the new infrastructure environment.

After the goals and requirements, a set of constraints was also identified. These constraints needed to be solved either for the migration to be possible in general, or to make certain migration strategies possible. The constraints can be found in Appendix B. Some of these constraints, especially concerning the running of older technologies in the new cluster, can be solved together with the infrastructure team. One of the constraints requires a solution in which the current and new orchestration solutions are connected, forming a hybrid solution. This constraint gives benefits for the application developers, giving them the opportunity to iteratively migrate the application to the new infrastructure. The hybrid setup does however go against the goal of an open-source infrastructure set by the infrastructure team. This is why analysis of the overall product goal is needed, as some of these decisions transcend the needs of one silo.

The overall analysis shows us that because of the risk requirements, the overall impact of the migration must be kept low. This is in line with what the external experts emphasised during the interviews. Trying to change a lot during the migration increases the risk of the migration and increases the complexity of tracking the impact of specific changes. Therefore, it is important to limit the scope of the migration as much as possible.

4.3 Reasoning

As the constraints have been resolved, the migration is now evaluated. This step results in a Go or No-Go decision. We see two ways this evaluation can be conducted: (1) reasoning about how the goals can be attained within the restrictions set by the requirements or (2) solutions to the problems presented in Section 4.1.1 can be sought, again within the restrictions set by the requirements and reasoning about the future of the project. These two evaluations of the migration are presented below.

4.3.1 Goal evaluation

Three goals have been isolated for the application and the effect of these goals is the following:

1. **Decoupling:** Decoupling could be linked to a move towards MSA, as the service split requires the decoupling of the application. Containerisation aids in the isolation of these decoupled services. On the other hand, the decoupling of the application is not directly linked to the move to the new architecture. It is a characteristic of the software that can be achieved independently of the migration. As the theory of MSA tells us, decoupling is a difficult undertaking [51]. As the migration is bounded in time and requires low risk, decoupling is best done separately or after this migration, as doing this requires major architectural changes to the software.
2. **Technological freedom:** This freedom is directly achieved by the new infrastructure, which means that as long as the application is containerised and moved to the new infrastructure, this goal can be reached.
3. **Security:** The extra isolation layer is achieved by utilising the new infrastructure. However, containers do come with their own new set of security concerns, which are explained in more detail in Appendix B.6. As with the previous goal, as long as the application is moved to the new infrastructure, this goal can be achieved.

The goals above do not directly dictate how the application should change. Only, the decoupling goal requires big changes to the application, the other two goals are achieved by the properties of the infrastructure. The decoupling goal can be achieved in many ways. This ranges from directly decoupling the application, with current technologies, to utilising the new infrastructure to facilitate a move towards MSA.

4.3.2 Problem evaluation

In this section, the analysis is related to the original problems of failover and container introduction. The analysis of the patching problem is discussed with the analysis of the container introduction problem, as they are intertwined.

Failover

The failover problem has two solutions, as discussed in Section 4.1.1. The analysis tells us that for the infrastructure the preferred solution is very clear. Solving the problem with redundancy keeps the infrastructure in line with the philosophy of Kubernetes. It is what the software was designed to do, what the Kubernetes engineers they hire are familiar with, and does not require them to construct many custom extensions on top of Kubernetes.

For CMS the problem has a preferred solution, however, this goes completely against the best solution for the infrastructure team. Reworking the CMS application to fully utilise redundancy is a complex task, as it requires re-architecting the application. Additionally, the CMS engineers will not have time during this migration to rework the application to utilise the standard Kubernetes technologies. This means Kubernetes is required to work through DDS, which is not originally designed to do load balancing. The architecture the CMS engineers would then be working towards, also benefits from the scaling and rollout features of Kubernetes, which they never plan to use. Therefore, it is most likely not a good option to work towards. The other option of constructing a hybrid solution where the current orchestration principles are still utilised is optimal. This allows CMS engineers to isolate services that benefit from more scaling, and keep the rest as it is. Keeping both systems in the infrastructure opposes most of the goals of infrastructure, as the infrastructure becomes more convoluted supporting both the old and the new technologies.

Container introduction

This part discusses the solution for both the patching and the container introduction problem together. The solution is either to introduce the containers in the Target testing level of the V-model, seen in Figure 11 or the MRS one layer above. Introducing the containers at the target level introduces the patching problem. The solution to the patching problem is to either mount the platform in the containers or include the platform in the image. Neither of these solutions is optimal, as mounting goes against industry standards and raises development times. Both of these solutions oppose the requirements of migration. Next to the patching problem, the form of deployment is ambiguous at this level, meaning testing all configurations is impossible.

Introducing the containers at the higher MRS level solves these problems. At this level, a specific product is tested, meaning that the deployment is known. This is also in line with the minimal impact migration. Introducing the containers here minimizes the impact they have on the developer experience. However, this means that containers are introduced quite late in the testing infrastructure, so if they fail a test, the feedback cycle is longer and more costly.

4.3.3 CMS reasoning

The conclusion from the above reasoning is that the move to the new infrastructure does work towards the Technological freedom and security goal of the CMS engineers. However, the problems they foresaw are problems introduced by the new infrastructure and do not have optimal solutions for CMS. There is the option to fully re-architect the application towards the new infrastructure, which is bound to be a large and costly undertaking. If a sufficient level of MSA is achieved, the failover problem is resolved and the containers could be introduced in lower levels of the testing infrastructure, as building time should decrease.

Splitting into service optimises the application for dynamic scaling, fast-moving teams responsible for their own services and a continuous DevOps environment [51]. These are all traits that are not beneficial to the CMS application. The downsides of MSA, increased latency and increased inter-service complexity, do definitely apply to CMS. This means that the move towards MSA might not be the right choice for CMS.

4.4 Resulting advice

For CMS, the goals remain unclear, even after the migration analysis. The foreseen problems have solutions, but none are completely ideal. Completely reworking the application towards MSA does not seem beneficial for the amount of work this requires. This leaves two options for the migration. Either the move to the new infrastructure is a No-Go, or a hybrid solution needs to be found. This hybrid solution will also impact the goals of the infrastructure team.

All the solutions require cooperation between the two silos. The reasoning results in the need for the shared goal creation. This consensus is the only way to decide on the future of this migration. Currently, CMS is split between either a hybrid cloud setup, or moving the application to the new infrastructure without rearchitecting it. The first goes against the goals of the infrastructure and is only acceptable if the application is migrated to MSA in the long term, and the second leaves CMS with little benefits from the new introduced technologies. Consensus between silos about the future of the product is the only way to choose if one of these options should be chosen.

Taking the reasoning even further, the engineers should consider if the migration is overall beneficial for the entire product. The infrastructure goals are achieved, but maybe they can be achieved in a way that does not add Kubernetes to the project. Kubernetes is a complex piece of software for which most features will not be used. In addition, when new in-house software is built on top of Kubernetes to enable the existing technologies to still function in the cluster, the open-source benefits of Kubernetes are lost. Even though the migration is already underway, it is still valuable to consider this before even more resources are put into the migration.

4.4.1 What to do next

The advice of what to do next applies to the CMS application team, as the analysis of their migration was the main focus of the research. In our vision, the migration analysis leads to a few possible options. These options are presented below in the order they should be tried.

1. CMS engineers should first discuss together with the infrastructure team what the specific problems are that the infrastructure team is trying to solve. Together they should explore different solutions to these problems. Possibly there is an option to address these problems, without pushing deployed software towards cloud native development. However, as the infrastructure team already put in significant effort into the migration and the other application team already migrated as well, this exploration might not succeed.

2. If no new solution is found for the new infrastructure, the infrastructure team and the CMS team should together explore what form of a hybrid infrastructure is acceptable to both. An infrastructure containing both the old and new orchestration methods allows CMS to migrate slowly, reducing the risk of the migration. It also decreases the amount of modifications the infrastructure team needs to make on top of Kubernetes to support the old technologies. It does however result in an infrastructure which still contains the old orchestrator, which is against the open-source goal of the infrastructure team.

3. After the two options above have been tried, the last strategy is to migrate the application to the new infrastructure, while changing as little as possible. This results in little benefit to the application, but can serve as a baseline from where they can again explore future migration options, like decoupling and a possible introduction of services.

Overall, the goal should be to get the infrastructure team informed of the migration analysis of CMS and create a vision for the migration together with them. For the future of the overall product, it is key to analyse migrations that span the entire product with all silos.

5 Discussion

This chapter first summarises the findings of the research in relation to the method. Then, the results are interpreted, and the implications related to the research objectives are discussed. Finally, the limitations of the work are addressed, and recommendations for further research are given.

5.1 Findings

This research was structured around the Design Science Methodology of Information Systems and Software Engineering [59]. The case study done as part of this research took place at Thales, a company working in high-assurance systems. The case study focused on a naval software system. Due to the military nature of this project, only the design cycle of this method was possible. The design cycle aimed to analyse the problem and design a suitable solution for this specific problem. The findings are, therefore, related to the particular solution created during this research. Because the validation of this created solution also depends on personal experience and expert opinions, the line separating concrete findings and interpretations is thin.

The result from the context analysis, the theory and the case study indicated that a migration method was needed, and one from the analysed literature was chosen, namely from [36]. This method was adapted to address the implication of the factors analysed between CN migration and private cloud migration. These adaptions were a more careful analysis regarding the benefits of the migration, the addition of multiple silos and additional decision moments. A set of four migration strategies was also presented on how to create a migration plan for the more complex migration.

The application of the newly created method highlighted the analysed factors between CN migration and private cloud migration. The lost benefits and the inclusion of multiple silos that conducted different migrations simultaneously increased the complexity of the analysis of the migration. The silos all had their own separate view of the migration. These different goals made understanding the overall migration hard. The external experts interviewed also addressed this same problem when presented with the created migration method. Therefore, a shared goal should be defined before starting the individual migration analysis, as is standard practice within the software industry [48]. In addition, a Go/No-Go decision moment was added as a conclusion of the analysis phase of the method. This inclusion resulted from the observation that the benefits and drawbacks of private cloud migration are highly dependent on the domain restrictions of the project, and therefore the migration might is not always worth the investment.

The resulting method can be seen in Figures 6 and 7. To evaluate the method, we applied it at Thales and presented it to external experts of cloud software engineering. The results of applying the method in the case study were in line with what the method was designed to do. As the migration was already underway for some of the silos, the analysis focussed on the proposed migration, but also the characteristics of the migration that was already happening. Following the method during the observational analysis was sufficient to reason about the goals of the overall migration and what is further needed. The outcome of this reasoning was that an overall goal for the product migration is needed, and that the silos need to find a solution that is acceptable to all involved silos, as the current proposed migration is not directly beneficial to the application. Additionally, they need to decide together if the migration is worth the investment for all silos involved. These finding confirm our decision of adding the addition of silos, a Go/No-Go decision moment and more analysis towards achieved benefits, to the migration method.

5.2 Interpretation and Implication

The findings stated above lead us to the implication that the design of the method was successful. The evidence is, however, mostly based on conversations because many of the underlying project details were not accessible to us. Therefore, the conclusions are limited. The resulting method fits the expectations; the design is based on the information learned in the context analysis of the project and constructed from observations at Thales.

However, the results for the case study were unexpected. The expectation was to improve the migration and to help them create a migration plan. The migration analysis would definitively give a clear solution for the patching and container introduction problems, that this research originally set out to solve. The result, however, was that no migration plan seemed to fit the migration analysis, resulting in advice to reassess the migration with all silos together and to reduce the impact of the migration as much as possible. This result aligns with what was found in the literature to be important for migration analysis, as seen in [54]. It is essential to identify beforehand whether the migration is possible and beneficial. Otherwise, significant investments are made with negative returns. The result from the method is heavily influenced by Thales. There is no comparison with other companies, so it is

impossible to say whether the method is biased towards this outcome. However, as most of the companies building high assurance systems heavily focus on the integrity of their software, this outcome might be representative.

Most of the migration theory we investigated, emphasises the creation of a migration plan as the challenging part of the migration method, as is the case with [20] [46] [36]. This is likely the case as they handle more standard applications, which the cloud infrastructures are designed to handle, like web shops, web applications and other applications with dynamic scaling needs. For these applications, the benefits mentioned in the literature for cloud and MSA apply. As this is not the case for high-assurance systems, like military applications, the emphasis of the migration method automatically shifts more towards the analysis part of the migration methods. The migration plan creation, however, should not be any more simple than with the more traditional CN companies. The plan now possibly includes many different stakeholder groups, like, for example, an infrastructure team. The creation of a migration plan could however not be tested, as the actual migration and details needed to create this plan were out of scope for this research.

The observation from applying the method is that it directly promotes the DevOps way of working. The alterations to the method were made because of observations made during the case study, which are most likely related to the new technologies being designed for DevOps. They shift some of the configuration forward during development. This realisation allows companies to consider organisational changes that would aid the migration beforehand. As DevOps empowers MSA and the introduced technologies [19], it can be beneficial to consider the organisational migration towards DevOps when planning for the migration. The migration towards DevOps is however not always possible for companies. The need to analyse what parts of DevOps are possible, mostly related to the benefits brought by MSA and container orchestrators, is an important insight from this work.

Overall, the created solution seems to fit the problem attempted to be solved. It addresses the identified differences with public cloud migration, and these differences lead to the eventual outcome of the case study. This outcome would have never been possible with the proposed public cloud migration methods found in other works. The result aligns with the experts' suggestions and seems to be an acceptable conclusion to the application of the method. This leads us to believe that the method we developed is also helpful to other companies that are considering private cloud migration.

5.3 Limitations

The first and most obvious limitation of this work is related to the small sample size. As the research is based on one case study, the generalizability is limited. The case study served both as inspiration and verification, meaning that the new focus points added to the migration analysis were added in part due to theory, but also due to observations at Thales. The case study therefore solely verified the observations made at Thales and the implications they had there. It verified the reasoning created from the theory, however, it does not allow us to draw the conclusion that the reasoning holds for every company wanting to migrate towards private cloud. Additionally, as all factors identified between private and public cloud migration were present at Thales, it is impossible to determine what effects they have on the migration if they only partially apply. There might also be more factors at play that did not apply to Thales. By presenting the migration method to external cloud experts, the method was again verified. They pointed out how the factors can only partially be present in companies and what they thought this would mean for the migration. However, as these were unstructured interviews with limited concrete data, this does not allow us to generalise the claims to say that the method fits every company perfectly. The interviews did support the method, as all experts deemed it a useful tool for analysing a private cloud migration with the characteristics described in this work.

Another limitation resulting from the case study is that it was impossible to follow the entire migration during this work. The migration is both too complex to analyse in detail during this work, and a lot of information is classified. Additionally, the migration is scheduled for a time period longer than the time set for this research. Therefore, the decisions made, and the resulting migration can not be analysed for this work. This limits the verification and applicability of the method. The method can never be fully applied, as creating a migration plan is not possible in this work, meaning that the migration strategies are never verified. It also does not allow us to directly see the impact of the resulting advice. Presenting the results at the company showed that engineers were interested in the advice and agreed with the analysis. However, what the impact is on the project is not known.

The work heavily relies on other works in the migration domain. As this work, many of these works suffer from a lack of concrete data, like [20]. Migrations take a long time to complete and are hard to evaluate afterwards. This means the evaluation is often based on experience, leading to less objective research, making it harder to compare methods. This lack of research towards the post-migration stage has also been identified in other works [33] [35]. This is a limitation to the theoretical background of the work. The best way found to add concrete data is to utilize data from surveys, like [30]. As far as we know, these do not exist for the migration of high-assurance systems.

A limitation resulting from the approach to solving the problem is that the Thales's organisational structure has mostly been ignored in the analysis. This was done because the research focused on the application's migration. However, as becomes apparent in the work, the organisation also has to migrate during this migration, as the entire development process of the product might change. This migration of the organisation and the analysis it brings is mainly left out of the theory, meaning that the analysis of the product migration is incomplete. As the organisational structure of companies building high-assurance systems is often structured different from fast moving CN companies, research needs to be done to analyse how they could change best when migrating to private cloud. In addition, leaving out this organisational analysis means the knowledge and decision-making structures needed for the migration are not analysed either. How this needs to change during a private cloud migration is not known.

5.4 Recommendations

The limitations do not invalidate the conclusion that the created method is an improvement over existing migration methods with respect to private cloud migration. This research has direct practical implications for Thales and serves as a tool for other companies. For companies building high-assurance systems, this work can give perspective into tackling cloud migrations, what is essential to analyse and how to structure the analysis and migration method creation. It provides an insight into the specific topics that make the private cloud migration a unique migration compared to regular cloud migration, giving companies a head start with the realisation that the product migration consists of multiple individual migrations. This realisation can allow the engineers of the company to create a shared goal, work more closely together, and evaluate the migration for every silo involved before starting. In turn, this can help speed up communication and avoid decisions that are not beneficial for all silos and therefore save time.

The other crucial practical implication of the work is showcasing that private cloud migration is not as straightforward, beneficial as public cloud migration. Domain restrictions, especially in the environment of Thales, limit the benefits provided by the migration and increase the adverse side effects. It is imperative for a company considering the migration to assess what they aim to achieve with the migration, compare that with what others achieved and evaluate whether the migration would be beneficial. The case study offers a perspective into this dilemma and gives an example of reasoning for this subject.

One of the recommended research directions is to research more different cases of private migration. Additional companies serve as verification of the factors analysed between private and public cloud migration, seeing if the impact is what is described here and if there are more factors involved. These additional case studies at high-assurance projects possibly also allows analysing the migration in its entirety.

Another valuable research direction would be to conduct a survey, as done with MSA migrations [30]. This would allow the migration's benefits and drawbacks for restricted domains to be assessed. This survey could also shed a light on how beneficial the adoption of MSA is in restricted, not-scalable deployments. The resulting information from such surveys would enable companies to make more well-informed decisions about migration as they have information on what benefits they can expect. The restrictions in these domains are broad, and the projects often have restricted access, so such a study could be hard to set up.

A research direction, often mentioned in other migration works, is to analyse the post migration stage. A study could define metrics on how to measure the effectiveness of a migration. This would provide researchers with a more objective way to analyse the migration results, therefore simplifying the research towards software migrations in general. Specific metrics result in more objective comparisons that could be of aid next to personal experiences. The definition of a post migration analysis also allows companies to structure and plan their migrations towards the more understood post-migration stage.

6 Conclusion

This work set out to improve the migration of privately deployed software to cloud technologies and architectures by designing a migration method that is based on cloud and MSA migration methods while complying with requirements and constraints set by the engineers of the project so that engineers can increase the agility and flexibility of the system. The work tackled the development of the solution using a Design Science Methodology. The context analysis provided the work with a theoretical foundation on the involved technologies, architectures and existing migration strategies. It also provided context for why to adopt these patterns and technologies and their drawbacks. This theoretical foundation laid the basis for the solution, which adapted an existing migration method with the identified factors of private cloud migration. This method was then successfully applied in a case study and presented to experts.

Overall, the method created proved a useful tool in analysing a private cloud migration during the case study. The case study mostly focussed on one of the applications the product consisted of and analysed its migration. The analysis provided enough arguments to successfully reason about the migration plans of the case study, regardless of the classified nature of the project. The presentation of the insights at the case study company was received positively. However, the infrastructure migration and organisational structure of the company were not analysed thoroughly. This work leaves these areas of the method open for further research. External cloud software experts were also presented the method. The experts deemed the migration method a good solution to tackle the described migration, and deemed the four identified migration strategies helpful. These strategies and migration plan creation was, however, not possible to validate in the case study, leaving it to be tested in future work.

By applying the method in the case study, we ended up questioning the overall product migration. The analysis showed that the overall goal of migration was undefined and that to create a successful migration for both infrastructure and the applications running on the infrastructure the silos needed to cooperate. This result is directly in line with the altercations made to the original CN migration as part of the solution presented in this work. It also showed that the benefits of MSA are different for high-assurance systems. The scaling and flexibility benefits might not outweigh the increase in complexity for these projects. This results in an even more complex reasoning for the application migration. As more companies considering private cloud migrations create high-assurance systems, this outcome is likely not unique.

To our knowledge, this work is the first to look at cloud migration for companies that cannot migrate towards the public cloud. It shows that the migration analysis is more complex for these companies, as the benefits normally expected cannot always be obtained. This was clearly seen in the case study. Future research into this direction can improve the migration method we defined by adding organisational analysis and testing it further. The organisational analysis can point out in what form these companies can adopt DevOps successfully, how to manage the extra knowledge needed for private cloud infrastructure. . Future works can also apply the method to more companies considering the migration and gather data from companies not able to move to public cloud that have already gone through the private cloud migration.

Overall, the work leads us to the conclusion that the creation of the desired migration method was a success. It led to an improved understanding of the migration happening at the case study company and was well received during the presentations. The reasoning provided in this work would have never been created without the migration analysis, leading to an expensive, possibly non-beneficial migration. This shows that no matter how beneficial a considered migration seems beforehand, evaluating it thoroughly and in a structured manner is crucial before starting.

References

[1] Agile manifesto. URL: https://agilemanifesto.org/.

[2] Cloud native computing foundation. URL: https://www.cncf.io/about/who-we-are/.

[3] containerd – containerd overview. URL: https://containerd.io/docs/.

[4] Dds. URL: https://www.adlinktech.com/en/vortex-opensplice-data-distribution-service.

[5] Docker - Market Share, Competitor Insights in Containerization. URL: https://6sense.com/tech/containerization/docker-market-share.

[6] Docker Hub Container Image Library | App Containerization. URL: https://hub.docker.com/.

[7] Kubernetes Documentation. URL: https://kubernetes.io/docs/home/.

[8] Linux Containers. URL: https://linuxcontainers.org/.

[9] Osgi. URL: https://www.osgi.org/resources/what-is-osgi/.

[10] Podmain. URL: https://podman.io/.

[11] TACTICOS - Combat Management System. URL: https://www.thalesgroup.com/en/markets/defence-and-security/naval-forces/above-water-warfare/tacticos-combat-management-system.

[12] The Use of Name Spaces in Plan 9. URL: http://doc.cat-v.org/plan_9/4th_edition/papers/names.

[13] We have left the cloud. https://world.hey.com/dhh/we-have-left-the-cloud-251760fb.

[14] Why companies are leaving the cloud. https://www.infoworld.com/article/2336102/why-companies-are-leaving-the-cloud.html.

[15] Introduction to Software Architecture. In Zheng Qin, Xiang Zheng, and Jiankuan Xing, editors, *Software Architecture*, Advanced Topics in Science and Technology in China, pages 1–33. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-74343-9_1.

[16] Docker: Accelerated, Containerized Application Development, May 2022. URL: https://www.docker.com/.

[17] Isam al Jawarneh, Paolo Bellavista, Filippo Bosi, Luca Foschini, Giuseppe Martuscelli, and Amedeo Palopoli. Container Orchestration Engines: A Thorough Functional and Performance Comparison. pages 1–6, May 2019. doi:10.1109/ICC.2019.8762053.

[18] Rainer Alt, Gunnar Auth, and Christoph Kögler. DevOps for Continuous Innovation. In Rainer Alt, Gunnar Auth, and Christoph Kögler, editors, *Continuous Innovation with DevOps: IT Management in the Age of Digitalization and Software-defined Business*, SpringerBriefs in Information Systems, pages 17–36. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-72705-5_3.

[19] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, 2016. doi:10.1109/MS.2016.64.

[20] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. Microservices migration patterns. *Software: Practice and Experience*, 48(11):2019–2042, 2018. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2608, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2608, doi:10.1002/spe.2608.

[21] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, October 2003. URL: https://dl.acm.org/doi/10.1145/1165389.945462, doi:10.1145/1165389.945462.

[22] Eric W Biederman. Multiple Instances of the Global Linux Namespaces.

[23] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011.

[24] Emiliano Casalicchio and Stefano Iannucci. The state-of-the-art in container technologies: Application, orchestration and security. *Concurrency and Computation: Practice and Experience*, 32(17), September 2020. URL: https://onlinelibrary.wiley.com/doi/10.1002/cpe.5668, doi:10.1002/cpe.5668.

[25] Ricardo Colomo-Palacios, Eduardo Fernandes, Pedro Soto-Acosta, and Xabier Larrucea. A case analysis of enabling continuous software deployment through knowledge management. *International Journal of Information Management*, 40:186–189, June 2018. URL: https://www.sciencedirect.com/science/article/pii/S0268401217308782, doi:10.1016/j.ijinfomgt.2017.11.005.

[26] Jessica DÃaz, Daniel LÃ³pez-FernÃ¡ndez, Jorge PÃ©rez, and Ãngel GonzÃ¡lez-Prieto. Why are many businesses instilling a devops culture into their organization? *Empirical Software Engineering*, 26(2):25, Mar 2021. doi:10.1007/s10664-020-09919-3.

[27] Patrick Debois. Agile infrastructure and operations: How infra-gile are you? In *Agile 2008 Conference*, pages 202–207, 2008. doi:10.1109/Agile.2008.42.

[28] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016. doi:10.1109/MS.2016.68.

[29] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[30] Paolo Francesco, Patricia Lago, and Ivano Malavolta. Migrating towards microservice architectures: An industrial survey. pages 29–2909, 04 2018. doi:10.1109/ICSA.2018.00012.

[31] Dennis Gannon, Roger Barga, and Neel Sundaresan. Cloud-native applications. *IEEE Cloud Computing*, 4(5):16–21, 2017. doi:10.1109/MCC.2017.4250939.

[32] Mahdi Fahmideh Gholami, Farhad Daneshgar, Ghassan Beydoun, and Fethi Rabhi. Challenges in migrating legacy software systems to the cloud — an empirical study. *Information Systems*, 67:100–113, 2017. URL: https://www.sciencedirect.com/science/article/pii/S0306437917301564, doi:10.1016/j.is.2017.03.008.

[33] Muhammad Hafiz Hasan, Mohd Hafeez Osman, Novia Indriaty Admodisastro, and Muhamad Sufri Muhammad. Legacy systems to cloud migration: A review from the architectural perspective. *Journal of Systems and Software*, 202:111702, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0164121223000973, doi:10.1016/j.jss.2023.111702.

[34] Alan Hohn. *The book of Kubernetes: a complete guide to container orchestration*. No Starch Press, San Francisco, 2022.

[35] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, 1:142 – 157, 02 2014. doi:10.1109/TCC.2013.10.

[36] Pooyan Jamshidi, Claus Pahl, and Nabor C. Mendonça. Pattern-based multi-cloud architecture migration. *Software: Practice and Experience*, 47(9):1159–1184, 2017. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2442, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2442, doi:10.1002/spe.2442.

[37] Asif Khan. Key Characteristics of a Container Orchestration Platform to Enable a Modern Application. *IEEE Cloud Computing*, 4(5):42–48, September 2017. Conference Name: IEEE Cloud Computing. doi:10.1109/MCC.2017.4250933.

[38] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert. Microservices. *IEEE Software*, 35(3):96–100, May 2018. Conference Name: IEEE Software. doi:10.1109/MS.2018.2141030.

[39] Robin Lichtenthaeler, Mike Prechtl, Christoph Schwille, Tobias Schwartz, Pascal Cezanne, and Guido Wirtz. Requirements for a model-driven cloud-native migration of monolithic web-based applications. *SICS Software-Intensive Cyber-Physical Systems*, 35, 08 2020. doi:10.1007/s00450-019-00414-9.

[40] David S. Linthicum. Cloud-native applications and cloud migration: The good, the bad, and the points between. *IEEE Cloud Computing*, 4(5):12–14, 2017. doi:10.1109/MCC.2017.4250932.

[41] Daniel López-Fernández, Jessica Díaz, Javier García, Jorge Pérez, and Ángel González-Prieto. Devops team structures: Characterization and implications. *IEEE Transactions on Software Engineering*, 48(10):3716–3736, 2022. doi:10.1109/TSE.2021.3102982.

[42] Thomas Bradford Jorge Diaz Fabio Hasegawa Corneliu Holban Sandra Jolla Rajesh Nagpal Sridharan Rajagopalan Murthy Rallapalli Balaji Ramarathnam Zeljko Soric Shankara Sudarsanam Vatatmaja Venkateshmurthy Mike Ransom, Prakash Bhargave and Carl Vollrath. *The Solution Designer's Guide to IBM On Demand Business Solutions*. Redbooks, 2005.

[43] I Mirbel and J Ralyté. Situational method engineering:: combining assembly-based and roadmap-driven approaches. *REQUIREMENTS ENGINEERING*, 11(1):58–78, MAR 2006. doi:10.1007/s00766-005-0019-0.

[44] Dmitry Namiot and Manfred sneps sneppe. On Micro-services Architecture. *Interenational Journal of Open Information Technologies*, 2:24–27, September 2014.

[45] Nikolas Naydenov and Stela Ruseva. Cloud Container Orchestration Architectures, Models and Methods: a Systematic Mapping Study. In *2023 22nd International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–8, East Sarajevo, Bosnia and Herzegovina, March 2023. IEEE. URL: https://ieeexplore.ieee.org/document/10094059/, doi:10.1109/INFOTEH57020.2023.10094059.

[46] Espen Tønnessen Nordli, Sindre Grønstøl Haugeland, Phu H. Nguyen, Hui Song, and Franck Chauvel. Migrating monoliths to cloud-native microservices for customizable saas. *Information and Software Technology*, 160:107230, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0950584923000848, doi:10.1016/j.infsof.2023.107230.

[47] Isaac Odun-Ayo, Rowland Goddy-Worlu, Lydia Ajayi, Boma Edosomwan, and Fiona Okezie. A systematic mapping study of cloud-native application design and engineering. *Journal of Physics: Conference Series*, 1378(3):032092, dec 2019. URL: https://dx.doi.org/10.1088/1742-6596/1378/3/032092, doi:10.1088/1742-6596/1378/3/032092.

[48] G. Orosz. *The Software Engineer's Guidebook*. Pragmatic Engineer B.V, 2023. URL: https://books.google.nl/books?id=BjRc0AEACAAJ.

[49] Claus Pahl, Antonio Brogi, Jacopo Soldani, and Pooyan Jamshidi. Cloud Container Technologies: A State-of-the-Art Review. *IEEE Transactions on Cloud Computing*, PP:1–1, May 2017. doi:10.1109/TCC.2017.2702586.

[50] Nigel Poulton. *The Kubernetes Book: 2023 Edition*.

[51] Chris Richardson. *Microservices Patterns*. Manning Publications, Place of publication not identified, 2019. URL: https://brad.idm.oclc.org/login?url=http://library.books24x7.com/library.asp?bookid=147125.

[52] Maria A. Rodriguez and Rajkumar Buyya. Container-based cluster orchestration systems: A taxonomy and future directions. *Software: Practice and Experience*, 49(5):698–719, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2660. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2660, doi:10.1002/spe.2660.

[53] Vitor Silva, Marite Kirikova, and Gundars Alksnis. Containers for Virtualization: An Overview. *Applied Computer Systems*, 23:21–27, May 2018. doi:10.2478/acss-2018-0003.

[54] Harry M. Sneed and Chris Verhoef. Cost-driven software migration: An experience report. *Journal of Software: Evolution and Process*, 32(7):e2236, 2020. e2236 smr.2236. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2236, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2236, doi:10.1002/smr.2236.

[55] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. Container security: Issues, challenges, and the road ahead. *IEEE Access*, 7:52976–52996, 2019. doi:10.1109/ACCESS.2019.2911732.

[56] United States and United States Office of Naval Research. *Symposium on Advanced Programming Methods for Digital Computers : Washington, D.C., June 28, 29, 1956*. Office of Naval Research, Dept. of the Navy [Washington, D.C.], [Washington, D.C.], 1956.

[57] Devi Priya V S, Sibi Chakkaravarthy Sethuraman, and Muhammad Khurram Khan. Container security: Precaution levels, mitigation strategies, and research perspectives. *Computers Security*, 135:103490, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0167404823004005, doi:10.1016/j.cose.2023.103490.

[58] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[59] Roelf J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014. 10.1007/978-3-662-43839-8. doi:10.1007/978-3-662-43839-8.

A Interviews

To verify the work done for this research software experts from different companies were interviewed. This was done to see what they thought of the created migration method, its application in the case study and their view on applying the method in different situations. The companies these engineers were from each had a different background. Their backgrounds were security, medical and web-based. The reason for this spread was to achieve different perspectives on the created method to generalize it as much as possible.

The goal of the interview is to verify the theory for migration theory creation, fill in possible gaps, evaluate the generalization and get expert opinion on the solution resulting from the theory. To achieve this goal, first, the initial problem was explained. Then the structure of figure 7 was explained and walked through. Then, once feedback was gathered on this process, the analysis resulting from these steps was explained. Arriving at the defining migration plan step, the theory of the migration matrix was highlighted. Then the decision and migration plan were explained.

A.1 Questions asked

Introduction

1. What is your name?
2. What is your function?
3. How do you rate your knowledge on a scale of 1-10 on cloud-native development?

Theory presentation

Start by explaining the problem faced by Thales and handing over the printed figures from the theory. Then take them through the theory while checking their understanding. Also, point out where this theory differs from the theory for cloud-native migrations. Now, follow up on these questions.

1. Do you think the alterations for private migration can be useful and why?
2. Are there things you would add to the analysis before planning the actual migration?

Resulting migration analysis

Walk through the diagram again, detailing what the results are of each part of the analysis.

1. Are the overall parts of the analysis clear and do you have a clear picture of the goal of the migration?
2. Is there anything missing from the analysis that would be necessary for a valid migration?

Migration plan creation theory and found solution

Discuss the theory and then ask questions.

1. Do you think the migration plan fits the analysis?
2. Where would you add the abstraction layers to the testing setup?
3. Is there anything you would change or add to the migration plan?

Closing

1. Where do you think the strengths of the overall theory and solution resulted in a lie?
2. In what area do you think it can be improved and how?
3. Is there anything else you would like to add?

A.2 Interview Wesley Dekker

Introduction

1. Wesley Dekker
2. DevOps Engineer
3. 8

Theory presentation

1. The split between platform and application is very useful. These two will operate separately making the focus on communication between the two very important.
2. Possibly the addition of the current state of the CI/CD and overall the current state of the technologies involved as this can drastically change the overall migration.

Resulting migration analysis

1. Yes
2. The analysis seems complete.

Migration plan creation theory and found solution

1. Yes the migration plan seems to fit the analysis
2. In practice, it's not the containers themselves giving issues. It's configuration, pods and allocations that are problematic. It is very important to first decide who will be responsible for these configurations. That is what should decide where the testing should happen.
3. -

Closing

1. In these situations, it is very important to keep looking at the bigger picture. That is where these methods can give much-needed structure to the analysis.
2. -
3. -

A.3 Interview Quenten Schoemaker

Introduction

1. Quenten Schoemaker
2. DevOps Engineer
3. 8

Theory presentation

1. The split between the platform and the application is a useful concept. Both have different concerns that need to be analysed. Also, the possibility of solving constraints either by platform alterations or application alteration is a valid addition.
2. It is important to add the analysis if the migration is really the best option to begin with. Also, it is important to identify how private the cluster is, meaning if the cluster is completely disconnected from the outside world or if only parts of the cluster have to be privately hosted. In other words, it is vital to determine the contours of your cluster before migration.

Resulting migration analysis

1. The overall analysis is sound, however it is hard to get to the underlying details, meaning the analysis stays abstract.
2. These are mentioned in the above part about additional parts that should be added. Also, it would be good to add analysis towards in-between options, like, in this case, virtualization instead of containerization.

Migration plan creation theory and found solution

1. It is important to not do the migration in one big bang. This holds for both the technology implementation and the introduction into the testing infrastructure.
2. -
3. Add explicit migration step options. The migration is otherwise going to fail.

Closing

1. Overall, it's good to have this structure.
2. For every migration step done add the rollback strategy and the analysis strategy. To perform the feedback cycles these need to be clear beforehand. Otherwise, the migration strategy is valid, and he would use it to create a migration plan for a similar migration problem.
3. -

A.4 Interview Ilia Awakimjan

Introduction

1. Ilia Awakimjan
2. Staff engineer platform
3. 10

Theory presentation

1. Yes, the analysis that platform and application have different goals and definitions of improvements is very valid. Also, the possibility of solving each other constraints is a valid observation.
2. The important part of the migration is before the separation into platform and application. The two silos need to beforehand discuss the goal of the migration and together come to a definition of success that grants them both benefits and can be measured. This is something that is vital to do together because the migration path splits afterwards. Also, the two silos do not have to be the only stakeholders in the process. There can be many more silos added to the diagram for specific cases. However, the structure stays similar.

Resulting migration analysis

1. The overall analysis is clear and the goal is clear as well. The reason why this goal is aimed to be obtained is missing. Containerization is not going to fix the problems for the application laid out and, in his opinion, not even the goals of the platform.
2. As mentioned before, the shared goal creation, together with the platform, resulted in this decision. In his decision the resulting migration should have never been attempted as it is not going to fix the issues presented and is only going to increase the complexity of the project.

Migration plan creation theory and found solution

1. The iterative migration fits the migration analysis because the benefits for application are scarce and it is therefore vital to measure the impact and to mitigate the risk of migration. However, it is key to experiment to see if iterative migration is even possible.

2. As late as possible. As DevOps is not possible it is key to keep the configuration consistent over the application. This decreases the complexity and improves the testability. It might not result in as fast development as with DevOps, however this is never the goal of development to begin with. Therefore it is advisable to introduce the costly abstraction layer as late as possible.

3. In general, the idea of obtaining benefits for the application in this migration should be dropped, in his opinion. As DevOps, dynamic scaling and outsourcing hardware are not possible the benefits of this migration are minimal, especially for the application. If the decision is set in stone by the platform team, the application should be converted to it with minimal impact on the application itself and keep to the current structure, as MSA is going to increase complexity too much.

Closing

1. The split of the silos with their respective goals is a useful tool for businesses to realise that the split will exist and the analysis of possible migration strategies is useful as well. However, for pure platform migrations, there can also be sole configuration migrations.

2. The generalization can be improved, but this is hard to do without losing the value of the theory. There also needs to be something added to a shared goal creation. This is vital to hard tasks as this allows tracking of the progress of the migration and makes sure all silos work towards the same goal.

3. His advice to Thales would be to completely reconsider the migration also on the platform side. These tools are designed for DevOps and many of their functionality is unusable in this use-case. They, on the other side, add a lot of complexity to a project. There most likely are better options to solve the problems faced in the project. Secondly avoid the distributed monolith at all costs. This is, in his experience, the worst case scenario, as it has the negatives of both sides.

A.5 Interview Menno de Jong

Introduction

1. Menno de Jong

2. Senior software engineer

3. 7.5

Theory presentation

1. Yes, it is a useful addition, as without both silos, the migration is impossible. These are the most standard silos and both need to be used to achieve the optimal flexibility

2. It is important to add a risk and cost analysis. This determines what you have available and the strategy for your migration. There should also be a decision moment after the analysis to decide if the migration is worth the cost and if it will actually be conducted or not.

Resulting migration analysis

1. Clear

2. Cost and time analysis are vital for the creation of a good migration plan.

Migration plan creation theory and found solution

1. Yes, these are the questions that need answers to create a valid migration plan. The problem statement is complete in his eyes, the answers will lead to valid migration plans. Using stakeholders from different teams and products within the company answers to these questions should lead to a valid migration plan.
2. Either directly at the team level or delayed to the end. The ownership of teams of the containers gives many benefits; however, it will require big organizational restructuring. It also depends on how many new interfaces the containers provide. If all communication stays on DDS, the containers can be added at the end of the testing phase. However, if the containers bring added interfaces, the testing should be done at a product level.
3. -

Closing

1. This analysis method drives both silos to work together and stimulates the flexibility of both. This helps to obtain optimal value for both.
2. 80/20 rule. It is important to also know that some parts of an application are not worth migrating as that would cost too much. It is ok to leave them in the old state or discontinue them.
3. -

A.6 Evaluation

The interviews were conducted with experts in the field of software development, with no association with Thales. Overall the experts responded positively to the created method. They agreed with the changes made to the original method and deemed it a useful tool for migration analysis. They agreed with the addition of silos, the four identified migration methods. They also agreed with the addition of iterative steps in the migration plan but emphasized that this is not always possible.

They also mentioned a set of improvements that were then discussed with the other experts. These were also applied in the case study. These were:

1. Addition of possible other silos.
2. Shared goal creation.
3. Go/No-Go moments for the migration.

The experts were less positive about the migration plan created thus far. The conclusion to the case study was not yet reached, so possibilities were presented. They pointed out the flaws of the options and argued against the migration, which led to less feedback on the creation of the migration plans.

B Constraint solutions

In this appendix the small studies towards solutions for the constraints are discussed. Each constraint is laid out using the same structure. First, the problem is discussed. Then a description of the constraint is given in more detail and the reason why it is important is given. Lastly, the solution approach and the actual solution are discussed. Some constraints were directly solved using experimentation, while others have multiple solutions with corresponding trade-offs. Where needed, the required research is shown and discussed.

B.1 DDS

Constraint: DDS must still function in the new cluster.
Description: DDS currently functions as the communication layer between instances running on different nodes of the cluster. The layer currently handles all communication happening in the system. There are initiatives to change this for parts of the communication, but it will most definitely be part of the system at the end of the migration. DDS functions using multicast, so the new infrastructure must support this.
Solution: This constraint can be solved by altering the infrastructure and can, therefore, be solved with the infrastructure team. They brought up the constraint, and they added a second network layer to the cluster to enable DDS and the multicast feature. The constraints are treated as solved, as the infrastructure is off-limits for this work, and the solution to the constraint cannot be directly tested.

B.2 IP protection

Constraint: Thales's intellectual property (IP) must be protected when stored on disk and viewed from within the operating system.
Description: The hardware on the ship might be accessible to an individual with ill intentions. Therefore, Thales needs to protect its IP and that of the companies whose software is integrated into the application. There are three ways the IP can be accessed. Firstly, through direct access to the hard drives. To mitigate this risk, the entire filesystem on the ship's hard drives is encrypted. The OS can decrypt the storage and use the files. However, people can still access machines that can decrypt the drives. Therefore, the applications stored on the disk also need to be encrypted. This is done by creating encrypted software and installing it on the drives. The first step of loading the application into memory is to decrypt it using a custom application. This mitigates the risk of stealing applications using the host machine. The last possibility of stealing IP is when it is already loaded in memory. Maintenance should not have access to the memory, and it is hard to mitigate using encryption, so not directly protecting this is acceptable. The problem is in protecting the applications on the drive, as they will be stored in the form of containers in the new architecture.
Solution approach: To solve this problem, options were researched and the solution that fit the project best was chosen.
Solution: The problem originates because applications are now stored in the form of containers and no longer as encrypted bundles. The lifespan of the container is important to understand in this story. The containers will be stored in a container repository, most likely a nexus repository. Once the cluster starts a part of the application, the container is pulled from the repository. The container is then started, meaning the container is placed on a host machine, and the application is started.

Three options were identified: (1) The container repository could be encrypted. (2) The containers could be encrypted during the build and decrypted when pulled from the registry. (3) The application inside the container could be encrypted.

Out of these solutions, 3 is the one that resembles the current implementation the most. However, this means that each container has to include an application that decrypts the application beforehand. This, however, also means that if the images are stolen, the decrypt mechanism is included in the image. It also means that decryption keys must travel into the container at runtime, meaning more communication with secret information. Lastly, it contains more Thales-specific software. Decreasing the Thales-specific knowledge was one of the drivers of this migration for the platform team. Therefore, this option was not deemed optimal.

Option 1 and 2 are similar in functionality. However, option 2 gives more flexibility and is supported out of the box, using the tools Thales is considering using. As can be seen in the documentation of the Podman push and the Podman pull commands, they have options to encrypt and decrypt the image. Only encrypting specific images also allows for flexibility in choosing when image encryption is needed. This option enables the cluster to decrypt the images directly when the image is pulled, using a secret stored in the secret storage option of the

cluster. This option was selected because of the flexibility and the fact that it is natively supported by the tools used.

B.3 OSGi

Constraint: OSGi has to be able to function inside a containerized application.
Description: As mentioned under current orchestration, Thales currently uses OSGi as a means of in-process orchestration. If one of the nodes in the cluster fails, other Java processes are tasked with enabling already loaded parts of the system. This also allows the jars that make up parts of the application to be swapped without reinstalling the system, allowing easy patching. As eliminating OSGi from the project during the 2-year migration time will not be possible, the constraint is that OSGi should be able to run inside a container.
Solution approach: The hypothesis is that containerized OSGi is supported. This hypothesis is directly tested using code, and best practices are researched.
Solution: Running OSGi runtimes inside a container has been observed to work. The research also showed that the Dockerfiles and even the containers can be automatically constructed using a plugin for the Maven build tool. This specific plugin builds Dockerfiles for Apache Karaf with a small runtime and configurable versions. This shows that the constraint is resolved, and the container generation can even be automated.

B.4 Orchestrating the monolith

Constraint: The orchestration of the current application must still be possible using the new technologies to enable iterative migration.
Description: To allow iterative migration, an environment should be set up that allows the new and the old systems to coexist. However, as the platform team does aim to migrate to Kubernetes as soon as possible, a solution needs to be created where the OSGi orchestration can be controlled using Kubernetes.
Solution approach: This is again a constraint that can be solved with the help of the infrastructure team. The constraint was discussed with the team, and an option of controlling the tracking processes (seen in figure 10) of the old orchestrator was discussed. As again, the source code of the project and, therefore, the orchestrator is off-limits for this work, this solution can again not be tested as part of this work.

B.5 Patching

Constraint: Patching new changes into the testing systems on the subsystem integration and system integration level must be possible, without completely reinstalling the software, within minutes.
Description: Currently, the software uses OSGi, which allows jars to be swapped into processes. This allows test systems to integrate developers' changes without having to re-install the software. These systems test the changes of multiple teams many times a day. It is vital for the development cycle that test results are returned to the developers within minutes. This means the system has to be able to incorporate many changes at a fast pace.

This problem is not commonly found in MSA architecture, as all services have their own codebase, making incorporating changes into a system fast. This is because only the container changed needs to be rebuilt and tested. Thales will not adopt this architecture in the coming years. Its platform contains code used by all services, e.g., a shared data model. This results in that for every change to the platform, all containers need to be rebuilt.
Solution approach: The solution approach was to collect possible solutions and evaluate them using the requirements set for the migration.
Solution: When testing in containers on the target systems, the testing infrastructure for individual teams has two main options. The first option is to rebuild containers whenever something needs to be changed, the other is to patch the software inside the running containers.

Both options have their respective problems. The first option takes quite a long time for each change. Every time something changes inside the platform, all containers must be rebuilt. Optimization can be made here, like copying the platform into the containers as the last step, allowing for the utilization of the caching functionalities of the built tools. The second option is to mount the platform into the container, allowing for easy patching without container rebuilding. This, however, goes against industry standards and increases the complexity of the testing setup [34].

As the infrastructure configuration is not an application developer's responsibility, the containers should not have to be tested during the testing of their changes. This means the containers can be left out of the testing until the system integration stage. This solution negates the problem entirely and still allows application developers to

test their changes to a satisfactory level. If the changes break the container infrastructure, the problem is system-wide and should be brought to the infrastructure team.

B.6 Security tooling

Constraint: The security of the containerized environment must not be lower than the current system.
Description: Security is one of the main concerns of Thales. Currently, there are certain tools used in the developer environment to scan executables to find vulnerabilities. With the introduction of containers, some of these security scanning features might have to be changed. Also, the focus of the security might have to change as with containers, they might be more prevalent in other locations.
Solution approach: Use scientific research for security focus points in containerized software environments, this means highlighting specific areas of interest for containers and how to secure them. Then analyse the current tooling used and suggest changes.
Solution: Thales already has high-security requirements for their current distributed application. A lot of the security requirements are, therefore, already in place. Container security is not very heavily studied in scientific literature [55]. However, the study [57] lists areas of interest for container security. The security hinges on securing the development process and configuration of the containers. Security concerns like untrusted hosts are not a direct problem for Thales.

Thales already heavily secures its build pipeline, implements vulnerability scanning, and limits access to packages. This should all also be done for containers. The configuration of the containers should follow industry standards, and a standard should be applied to all containers where possible. These configurations should be tested within the system testing.