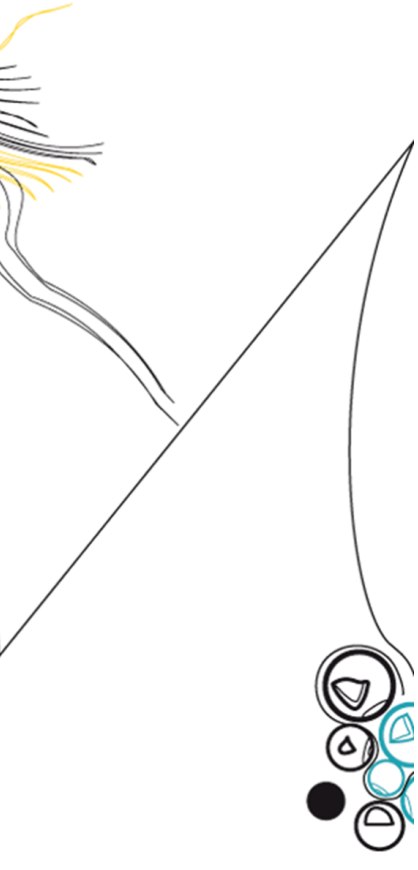
UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

Designing a container management solution to improve flexibility and portability, and reducing cost for iPaaS solutions.

Martijn H. Woudstra
Master Thesis
m.h.woudstra@alumnus.utwente.nl



First Supervisor:
Dr.ir. J.M Moonen (UT)
Second Supervisor:
Dr.ir. J.L. Rebelo Moreira (UT)
Company Supervisor:
Dhr. S. Kaya (eMagiz)

Telecommunication Engineering Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Executive Summary

Enterprises rely more and more on multiple systems in order to execute daily tasks. In order to connect these different systems, middleware can be deployed to overcome syntactic an semantic interoperability. Creating and maintaining these integrations can be a lot of work and costly. iPaaS platforms jump into this gap by providing a platform to easily create and maintain these integrations. Developers design the required integration on the platform, test the integration and deploy it. This deployment happens in the cloud or on-premise to facilitate the integration. Depending on how much the integration is expected to use, a cloud resource is appointed to the integration.

However, several problems occur from this deployment. Firstly, most platforms use on cloud provider and create solutions based on that cloud provider, increasing vendor lock-in. Switching from one cloud provider to another is difficult, since all cloud specific processes need to be changed. This calls for a solution that works cloud agnostic. Secondly, traditional cloud resources are bought in terms of a specific size of Virtual Machine (VM). This is done based on the estimation of how much the integration would require. In order to ensure the integration always works, the machine size is overestimated. Due to this, during non-peak moments, a lot of resources are bought, but never used, resulting in waste of cloud resources.

This research aimed to find a generic solution for iPaaS providers to increase flexibility and portability, and to reduce cost. For this, requirements were discussed in terms of what the new architecture should be able to do. Based on the requirements, several different frameworks and tools were discussed, and assessed against the requirements. Kubernetes came out as best fitting solution, matching 11 out of 13 requirements. The other two requirements were depending on the cloud architecture of the iPaaS provider.

Next, using the reference architecture of iPaaS providers and the result of the frameworks and tools comparison, a gap analysis was performed, and a target architecture was designed. This architecture fulfilled 24 out of 25 requirements. Validation also showed portability and flexibility increased. To apply the theory in practice, a case study was performed. From this, a decrease of 28% was identified. Although overhead cost increased by 10%, dynamic cost decreased by 36.5%. Since dynamic cost scale with the amount of customers, this outweighs the increase of overhead cost.

This research produced several artefacts: A suggested reference architecture for iPaaS providers' cloud architecture, a reference architecture applying Kubernetes to the iPaaS providers' cloud architecture, and a cost analysis of the reference architecture applied in a case study.

Contents

List of Figures

List of Tables

Acronyms

AD	Active Directory
AKS	Azure Kubernetes Service
API	Application Programming Interface
AWS	Amazon Web Services
CA	Certificate Authority
CAGR	Compound annual growth rate
CAPEX	Captital Expenditure
CDM	Canonical Data Model
CF	Cloud Foundry
CLI	Command-line Interface
CNI	Container Network Interface
CO	Container Orchestration
CPU	Central Processing Unit
CRD	Custom Resource Definition
CRI-O	Container Runtime Interface for the Open Container Initiative
CRON	Command Run On
CTO	Chief Technical Officer
DaaS	Desktop-as-a-service
DC/OS	Distributed Cloud Operating System
DNS	Domain Name System
DSRM	Design Science Research Methodology
EC2	(Amazon) Elastic Compute Cloud
EKS	(Amazon) Elastic Cloud Kubernetes
ESB	Enterprise Service Bus
ES	Event Streaming
ESP	Encapsulating Security Protocol
FQDN	Fully Qualified Domain Name
GC	Garbage Collection
GiB	Gibibyte
GKE	Google Kubernetes Engine
GPS	Global Positioning System
HA	Highly Available
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure-as-a-service
IP	Internet Protocol
iPaaS	Integration Platform-as-a-Service
kubectl	Kube Control
LB	Load Balancer
LXC	Linux containers
MAC	Media Access Control
MoSCoW	Must, Should, Could, Would like

NIST	National Institute of Standards and Technology
OPEX	Operational Expenditure
OS	Operating System
PaaS	Platform-as-a-service
PV	PerstistentVolume
PVC	PerstistentVolumeClaim
QOS	Quality-of-Service
RAM	Random Access Memory
RBAC	Role-based access control
RKT	Rocket
SaaS	Software-as-a-service
SLA	Service Level Agreement
SLO	Service Level Objective
SSL	Secure Sockets Layer
TAP	Test-, Acceptance-, Production-
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TXT	Text records
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VC	Virtual Container
vCPU	Virtual CPU
VM	Virtual Machine
VPC	Virtual Private Cloud
XaaS	Anything-as-a-Service
YAML	YAML Ain't Markup Language

Chapter 1

Introduction

In this chapter, the motivation and problem that this research focuses on will be elaborated on. From that, research questions will be deducted. Next, the approach to answer the research questions will be elaborated on. The structure of the report will be presented to the reader, followed by the scope of the report, and an introduction to eMagiz, a company that helped verify this report.

1.1 Motivation

The market of microservices is growing rapidly. The 'Microservices Market' is expected to grow from $832 million in 2020 to $2701 million by 2026[1], meaning a Compound annual growth rate (CAGR) (compound annual growth rate) of 21.7%. For Integration Platform-as-a-Service (iPaaS) solutions, the growth is projected to be even larger with a CAGR of 38% between 2021 and 2026[2]. These two markets growing side by side is no surprise, taking that iPaaS solutions are ideal for integrating microservices. iPaaS solutions are made to enable connecting and deploying microservices easily.

iPaaS solutions have major responsibilities when it comes to the availability of the system it supports. Disruption of a system could potentially disrupt message delivery of customers' messages, which could lead to wrong actions due to missing data, or disruption of the information chain. At the same time, iPaaS solutions are an ideal place to integrate new technologies; the amount of impact changing a single iPaaS solution has is generally greater than when one company itself implements these new technologies and patterns. This is because these platforms generally serve many different enterprises.

1.2 Problem Statement

Deploying cloud services or Anything-as-a-Service (XaaS) solutions on bare-metal has been widely described as costly[3]. Virtualization has been adopted by the industry as an alternative because of its measurable and intuitive benefits including reduction of Captital Expenditure (CAPEX) cost improved staff efficiency, and quicker deployment options[3]. One of the main issues in virtualization is resource optimization. Containerization is a going trend adopted by many enterprises to deal with this issue. Containers are "*packaged, self-contained, ready-to-deploy parts of applications and, if necessary, middleware and business logic*"[4]. Containers omit the need for an Operating System (OS), in contradiction to Virtual Machine (VM)s which need an OS, and additional overhead. Furthermore, they instantiate faster[5] and offer

near bare-metal performance [5]. However, Khan described that security is a drawback of containerization due to less isolation because of the shared kernel access [6].

Additionally, Sharma showed that interference (simultaneous accessing of resources) when many containers are deployed on one host might lead to reduced performance [5]. Furthermore, management becomes an issue when handling many different containers. One can imagine having hundreds of containers running for different tenants can be impractical if, for example, a customer request comes in and requires adjustment of their containers. This calls for a need to deploy and manage these containers flexibly.

iPaaS providers choose to use containers because containers are environment independent, meaning that the providers do not have to set up all environments with specific software like databases or network adapters. Instead, these are all packed within the container, meaning that (almost) any environment is capable of hosting these containers. For iPaaS solutions, customers can design their integration with the platform. Afterward, their integration can be deployed as a packaged container, usually on the virtual infrastructure of the iPaaS provider.

The downside of running integration in containers is that resources of the host need to be shared, meaning that if one container starts consuming many resources, other containers experience delay [5]. Additionally, containers will share the underlying kernel, which causes security issues [5] as mentioned earlier.

Customers should be able to spin up and tear down their integration containers, while the iPaaS provider should be able to manage and service all containers. This, however, causes many of the already mentioned problems to arise. There is a need for additional tooling to solve these problems. Many different solutions exist to achieve the desired situation. However, to the best of our knowledge, no research has been conducted on what requirements apply to iPaaS solutions, which solution would best solve these problems, and how to implement this solution.

1.2.1 Example scenario for a customer

Assume a hypothetical company *IOps*, which shares many characteristics with real-world logistic companies. IOps is a logistic company that operates internationally in Europe, focuses on transporting electronic devices all over Europe. IOps has trucks to deliver goods from A to B, but they also outsource contracts to other transport companies. IOps has many different systems contributing to their main business goal. For example, IOps has a bookkeeping system to record the incoming and outgoing funds. Additionally, they have an order system to track the current, past, and future contracts with all required information like pickup address, destination address, pickup time, weight, etc. Customers can also send a request to this system when they need something to be shipped. Furthermore, IOps has systems in place to track truck drivers and their loads. These are connected to the Global Positioning System (GPS) locations of the truck. For customers, a portal exists to track the goods they ordered. This system is connected to the order system and the tracking system. All these systems are bought as is and are not developed by IOps.

To reduce the workload of transferring data from one system to the other system, these need to be connected. However, all systems have a different definition of how these messages look (syntactic meaning), what they mean (semantic meaning). IOps could develop middleware software to connect one system to the next, but since all systems are connected, the number of adapters will quickly explode (exponentially, see [7] and Figure 1.1b), and becomes unmanageable.

IOps wants to have a **central place to manage these integrations** and to **reduce the number of integration**, to reduce maintenance cost for these integrations, and to have a more clear overview of what and between which systems integrations exist. Furthermore, they

(a) Integrations required without CDM (Nodes = 6, Edges = 30)

(b) Integration required with CDM (Nodes = 6, Edges = 12)

Figure 1.1: Integrations required with and without CDM

want to be able to **expose their order system to receive messages from customers**. Lastly, they experience more traffic on their tracking system during the day and thus want to dynamically **give the system more bandwidth and resources during this time.**

The business goals of IOps can be achieved using iPaaS solutions. Firstly, the iPaaS solution can be used to have a central place where all integrations can be overviewed. Secondly, to reduce the number of integrations required, IOps can define a Canonical Data Model (CDM) in the iPaaS solution. A CDM is a central model where all incoming messages map onto and outgoing messages map from. This way, there is a central data model where all integrations map onto, reducing the needed integrations to the number of systems, making the equation linear (See Figure 1.1b). Thirdly, to receive messages from customers, IOps can define an Application Programming Interface (API)-Endpoint in the iPaaS solution, connect it to the CDM and use the existing integration to send it to their order system.

The last business goal of IOps is more difficult. Often, one has to decide how many resources a system would get, thus either overestimating for the slower periods or underestimating for the faster periods. To solve this, the system should be either manually or automatically scaled up and down.

Not defined as a business goal, but a requirement of the system is as **little downtime as possible, and no loss of data**. Therefore, the iPaaS solution should also have redundancy available.

In the next subsection, the customer requests are translated into business goals for iPaaS providers.

1.2.2 Motivation for iPaaS providers to handle the problem

The business goals mentioned above are problems iPaaS providers try to solve. One problem described earlier, the CDM, is not in the scope of this research. Two other problems mentioned, **giving the system more bandwidth and resources**, and **the need for redundancy**, are covered by this research. There are a few options how to add this redundancy and temporary scale-up. However, these options rise challenges that need to be overcome. To be able to add redundancy, one can startup a replication of the application. This requires the storage volumes to be able to handle multiple connections in a stateful manner. The feature of having more computing power based on needs is called *auto-scaling*. Two methods of auto-scaling is *vertical auto-scaling* and *horizontal auto-scaling*. Horizontal auto-scaling means increasing the number of workers to handle the request. Vertical auto-scaling is the addition of resources to the existing worker (usually VMs) [8]. Vertical scaling is useful up to a certain point since scaling cannot go beyond the resources the physical system has [8]. Horizontal scaling has more options (since adding a new machine is not limited by the current stack), but also brings

additional challenges. For example, when do you need to add a new VM, and how can you decide when to remove unused VMs. Or even replacing VMs with VMs with more resources. On a high level, the following reasons exist to solve the problems at hand:

- Firstly, a group of containers (cluster) should be able to manually or automatically scale up or down depending on the used or requested resources, and the resources available. This raises challenges for administrators of such clusters. Manually checking the resources is labor-intensive, and burst changes could be hard to identify and react to. This problem is one of the problems that need to be solved. Having autoscaling also opens up the opportunity to adopt 'pay-per-use' structures. (*Flexibility)*
- Furthermore, reducing costs is always on the mind of enterprises. By scaling up and down containers, the Operational Expenditure (OPEX) cost can be reduced since resources will be freed when not used. *(Cost)*
- Lastly, customers of iPaaS solutions might have requested to have their integrations deployed on certain cloud host platforms. This motivates to adopt container management to facilitate deployments on multiple cloud platforms. *(Portability)*

Other than these three reasons, additional reasons can be addressed, which do not have a direct cause that leads to the research questions but are strongly influenced by the result of the treatment. Therefore, these influenced issues will also be discussed.

- Firstly, Opara-Martins has argued that the adoption of cloud has stimulated vendor lock-in [9]. This offers a risk but also a limitation in portability since only the offered services of one vendor can be used. Therefore, this problem needs to be addressed as well. *(Prevent vendor lock-in)*
- Secondly, the literature suggests that the maturity level of an enterprise can be improved by virtualizing resources, location, and ownership [10]. Since the problem at hand already is about shared resources of containers, this opportunity can be looked at as well. This increases the level of Maturity. *(Maturity)*
- Thirdly, to be able to adopt future possibilities in technological advancements, like serverless computing, some prior dependencies need to be solved, of which one is the container management strategy. Due to this, there are open opportunities for future strategies which can be identified and explored. *(Looking towards the future)*

We see the motivation from iPaaS providers to solve the two problems mentioned has three main reasons:

- **Flexibility**: iPaaS Providers want to be able to automatically scale up and down containers to deliver performance according to the customer's needs.
- **Cost**: Reducing the resource utilization when little workload is offered to reduce OPEX cost.
- **Portability**: Ability to deploy on multiple cloud hosting platforms.

Additionally, by solving the problems described, three other topics could be improved, since there is a direct or indirect influence between these topics and the problems above:

- **Prevent vendor lock-in**: To reduce the risk of vendor lock-in due to the adoption of cloud containers, [9] vendors are looking for possibilities to adopt hybrid solutions.
- **Maturity**: Containerization and virtualization of resources, location, and ownership increase maturity level [11].
- **Looking towards the future**: Managed containers open up future possibilities, like serverless computing.

These goals described above are model in Figure 1.2 with their relations.

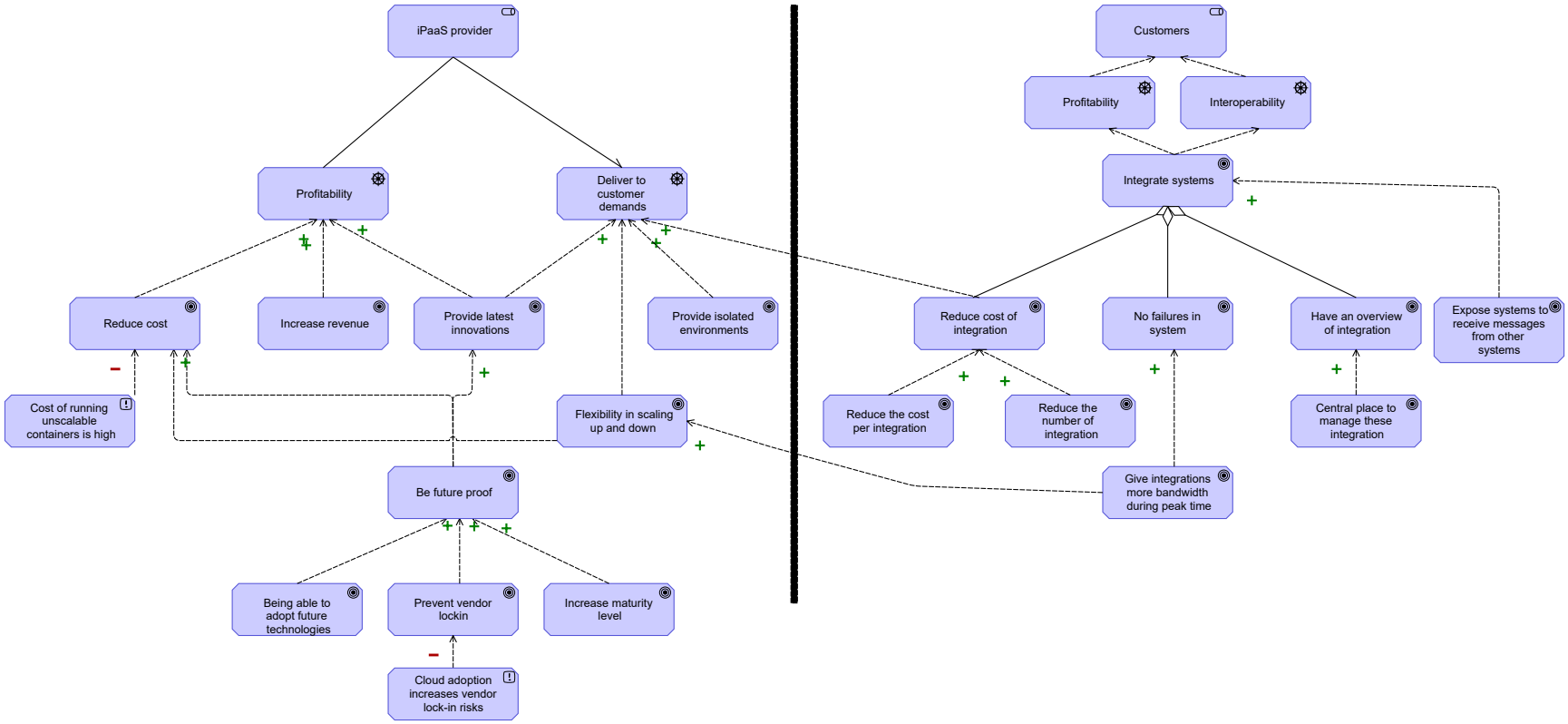


Figure 1.2: Motivation view related to the problem statement

1.3 Research Questions

Based on the Problem Statement, research questions have been formulated. These questions are in line with the Design Science Research Methodology (DSRM) [12], as further described in Section 1.4.

The main research question is formulated as:

MQ How should an iPaaS implement container orchestration in order to improve flexibility and portability, and reducing cost?

In order to answer the main question, several subquestions are defined:

SQ1 What is container orchestration and iPaaS, and how can they work together to improve flexibility portability, and reducing cost?
Rationale: *To investigate the relevance of container orchetration for an iPaaS, these two topics will first be evaluated. In addition, this question will address whether these two topics can be used in conjunction to achieve the intended benefits.*

SQ2 Which specific requirements should be covered to improve the flexibility and portability and reduce the cost of an iPaaS solution to ensure feasibility and similar functionality?
Rationale: *After the literature discussion, requirements for the solutions should be discussed to take into account during the design process.*

SQ3 What container orchestration solutions exist for iPaaS cloud architectures and what is the tradeoff between these solutions? How do these solutions meet the requirements and what would be the disadvantages of these solutions?
Rationale: *Available techniques are reviewed and discussed, and how they would be able to fullfill the requirements set in the previous subquestion*

SQ4 Can a reference architecture be derived as solution design, and can this architecture be introduced into the as-a-service deployment landscape of iPaaS solutions?
Rationale: *Taking into account the requirements and available techniques, a solution design must be made. This should be done to a generic iPaaS case, to see if an reference architecture can be derived.*

SQ5 Does the reference architecture solve the problem identified, taking into account the requirements of portability, flexibility and cost reduction? of iPaaS solutions?
Rationale: *The design should be validated in terms of validity of the model, whether it improves on the desired aspects, and whether the solution holds in a real-world case.*

SQ6 To which XaaS solutions would this reference architecture be applicable, and what advantages would this bring?
Rationale: *The solution is made for iPaaS solution. However, it might solve more problems existing, and thus the solution is assessed whether it can be generalized or used in more domains.*

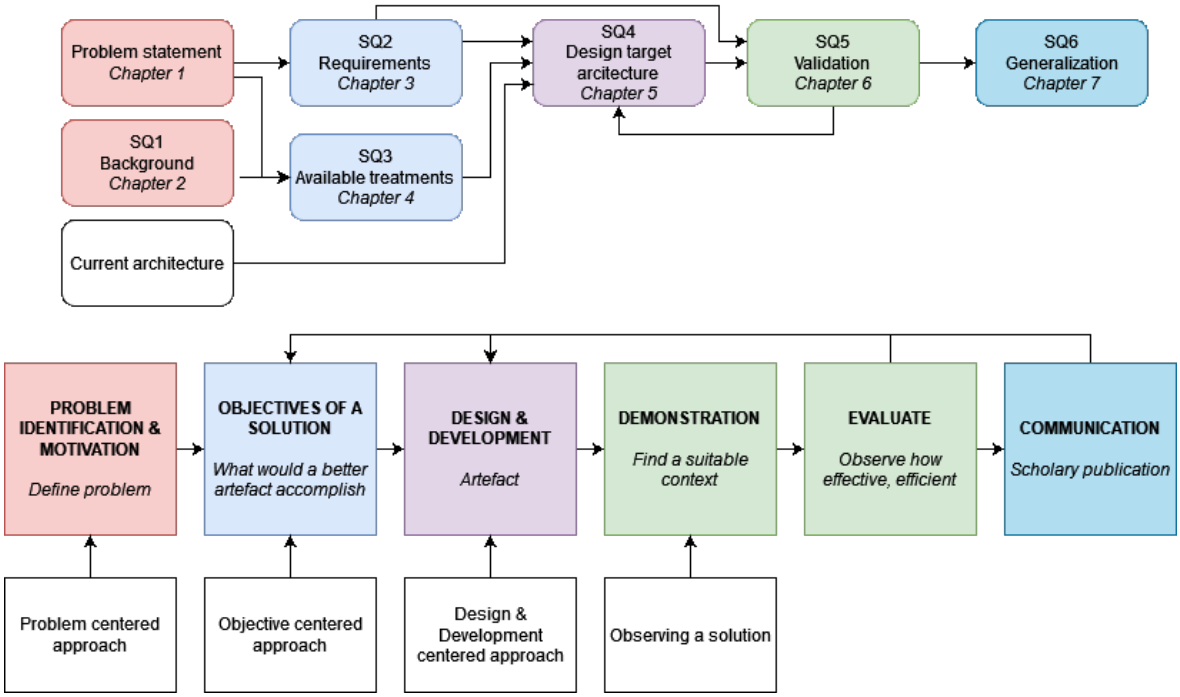


Figure 1.3: Combining Chapters of the report with Peffers' DSRM model[12]

1.4 Approach

During this research, Peffers' DSRM[12] will be used. Peffers' methodology tries to provide handles on how to perform such a design research. It guides the researcher from the problem statement and requirements to an artifact. Peffers' methodology allows researchers to start at any part of the process given the available knowledge.

Peffers' DSRM describes 6 activities (see figure 1.3):

1. Problem identification and motivation
2. Define objectives of a solution
3. Design & develop
4. Demonstrate
5. Evaluate
6. Communicate

Given the research problem, four different entry points for research exist: *Problem centered approach* starting from activity 1, an *objective centered approach*, starting from activity 2, a *development centered approach*, starting from activity 3, and lastly, when a *solution already exists or is observed*, starting from activity 4.

The research questions in 1.3 are already formulated according to Peffers DSRM.

SQ1 asks for a literature review to the definition of iPaaS and container orchestration, and the dynamic between them. This step will also give more insight in the problem at hand. **SQ2** asks to define requirements, as per step two in the DSRM, 'Define objectives of a solution'. This step can be performed by conducting interviews with experts from the field to give input for these requirements.

Finding requirements is a process which can be -performed in several ways. From comparison of these techniques in the report of Rehman[13], the type which should be used is heavily dependent on the situation of the requirement engineer. Each technique has pros and cons,

but Rehman suggest at least to use a combination of techniques. Abbasi[14] adds various properties to the different techniques. This table is shown in Table 1.1.

Since experts are available to answer questions, this opportunity is used to determine requirements for the system. From the experts, Qualitative and Quantitative data is needed to define the right requirements. For this, the technique of performing an interview was chosen. The interviewee would first answer some basic questions about the system (like purpose and stakeholders). Afterwards, requirements in Must, Should, Could, Would like (MoSCoW) format would be established. This would give us requirements for the system which would be prioritized according to the MoSCoW importance ladder

Techniques of Requirement Elicitation		Direct /Indirect	Qualitative/Quantitative data	Communication	Understanding the domain
Classic/Traditional Techniques	Interviews	Direct	Qualitative Data and Quantitative Data	Single-directional with the exception of interviews	Yes
	Surveys	Indirect	Qualitative Data and Quantitative Data	Single-directional with the exception of interviews	Yes
	Questionnaires	Indirect	Quantitative Data	Single-directional with the exception of interviews	Yes
Cognitive /Analytical Techniques	Card sorting	Indirect	Quantitative Data	Single- and Twodirectional	Yes
	Laddering	Indirect	Qualitative Data and Quantitative Data	Single- and Twodirectional	Yes
	Repertory Grids	Indirect	Qualitative Data and Quantitative Data	Single- and Twodirectional	Yes
Modern and Group Elicitation Techniques	Brain Storming	Direct	Qualitative Data	Twodirectional	Yes
	JAD	Direct	Qualitative Data	Twodirectional	No
	Prototyping	Direct	Qualitative Data	Twodirectional	No
Social Analysis	Ethnography	Direct	Qualitative Data	Single- and Twodirectional	Yes
	Direct Observation	Direct	Qualitative Data	Single- and Twodirectional	Yes
	Passive Observation	Indirect	Qualitative Data	Single- and Twodirectional	Yes

Table 1.1: Evaluation of Requirement Elicitation Technique according to Abbasi[14]

SQ3 asks to review available techniques, and discuss their advantages and disadvantages. From the process in SQ1, a number of techniques will arise, which can be discussed and compared. Lastly, their documentation can be consulted to discuss (without a proper prototype) how the solution would meet the requirements or not.

SQ4 asks for a solution design. From SQ2 and SQ3 a solution direction is already presented, by a 'best solution' to solve the problem, and the requirements that the solution needs to fullfill. However, an additional component is needed, being a reference architecture of the current situation. This will be discussed based on literature and interviews. Combining these three components gives enough information to perform a gap analysis, and design a target architecture.

Next, **SQ5** asks whether the solution has fulfilled the requirements and the objective of this research. This needs to happen in six steps. Firstly, the requirements of SQ2 are validated with a prototype. Secondly, the models from SQ4 are validated with the help of experts for validity. Next, it needs to be proven the flexibility and portability has increased. This will be done by comparing situations in the old scenario with situations in the new situation. Lastly, two cost components need to be verified. Firstly, the overhead cost need to be discussed to determine how it compares to a current situation. Then, the dynamic component needs to be verified. Combining the overhead and dynamic compontent gives us proof if the solution reduces cost. Lastly, in Subsection 7.3.1 the validity of the research itself is validated. For this, we use the Design-Science Research Guidelines of Hevner[15]. Hevner defines seven guidelines for a valid design research as can be found in Table 1.2:

Lastly, in **SQ6** the solution is generalized. Different XaaS solutions might benefit from the architecture, and this is discussed.

Guidelines	Description
Guideline 1: Design as artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation
Guideline 2: Problem relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems
Guideline 3: Design evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods
Guideline 4: Research contributions	Effective design-science research must provide a clear an verifiable contribution in the areas of the design artifact, design foundations and/or design methodologies.
Guideline 5: Research rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact
Guideline 6: Design as a search process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1.2: Design Science Research Guidelines according to Hevner[15]

1.5 Report structure

This report is structured as follows:

- Chapter 1 introduces the problem and defines the questions which need to be answered to solve the problem. The research design is also described.
- Chapter 2 will provide the information required for the rest of the report in terms of virtualization, as-a-service deployments (SaaS, PaaS, iPaaS), and container orchestration solutions.
- Chapter 3 will define requirements of the solution.
- Chapter 4 discusses available techniques including their strong and weak sides, and how they would fit in the solution design or not.
- Chapter 5 will discuss how the solution can be designed.
- Chapter 6 will discuss the validity of the design by reflecting to the functional- and non-functional requirements. This includes a cost analysis.
- Chapter 7 will draw a conclusion. Future work and a generalization will also be discussed.

1.6 Assumptions

Assumptions used during the research will be discussed in this section.

Firstly, we need to assume the size of the business to make accurate estimations of cost and available resources like available working hours. For this research, we assume the business which wants to adopt the solution to be small-sized, meaning less than 50 employees and less than €10 million turnover. Furthermore, this document is written based on documentation

and versions currently available. Future version might work differently. Lastly, this report uses euro's instead of USD. We take the exchange rate at the moment of writing of 1 Euro = 1.10 USD. All figures presented will be in euro's.

1.7 About eMagiz

To validate the solution proposed, an iPaaS provider was contacted. eMagiz is based out of the Netherlands and has between 10 and 50 employees. The company has an online portal where customers can design integrations in five different steps: Capture, Design, Create, Deploy and Manage. First, the company is introduced, and the general workings of the platform, after which the solution is validated.

eMagiz[16] was founded in 2011 and shares the same building and owner as CAPE Groep[17]. eMagiz operates from Enschede, near the University of Twente, and develops a similarly named platform that is a low-code, enterprise integration platform as a service (iPaaS). eMagiz aired its online, cloud-based platform, around 2013, but it has been building its integrations long before that. They continued building integrations, which lead to an Enterprise Service Bus (ESB) solution. Later, in 2020, it launched two other integration patterns; API Gateway and Event Streaming. It provides an online environment to set up data integrations, which can be self-managed and are cloud-hosted.

In the next sections, relevant background information to understand the assignments during the internship is explained.

1.7.1 Platform overview

In this section, the design process for customers in eMagiz is explained. The readers should note that not everything is elaborated on here because it is not needed for the rest of the report. Only important principles, which are needed to understand the design choices, are described.

Capture Phase

Customers of eMagiz purchase a full-stack web-based integration platform. In this platform, a web interface is offered to design integrations. Firstly, one starts in the capture phase where the customer designs the systems involved and the messages that flow to the different applications. Here, the user also provides information on whether these messages should be handled by an ESB, the API Gateway, or by Event Streaming (ES). A small example is given in Figure 1.4. For typical customers, the number of systems and messages will be many times bigger. Notable is that all different message types are displayed without integration type (Gateway, ESB or Streaming) here. This will be defined in the next section, the Design Phase.

Design Phase

In the design phase, the focus is on designing how the messages are formatted and how the messages are transformed. How this is done, depends on the receiving and sending systems, and is for the customers to design. Using the portal (web interface), customers can define messages (as can be seen in Figure 1.5) for the receiving and sending systems, and define a mapping between them (see Figure 1.6).

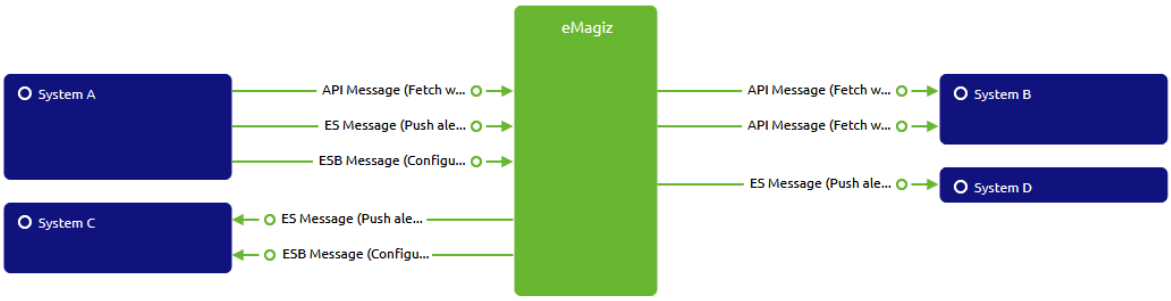


Figure 1.4: Example output of capture phase

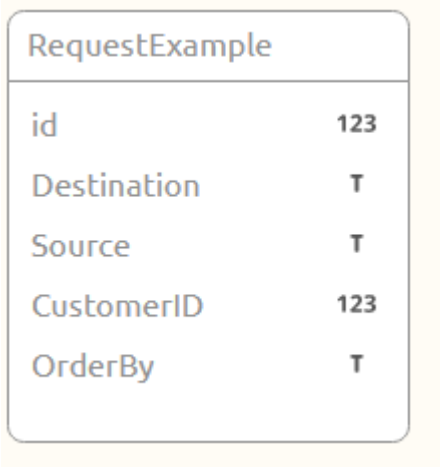


Figure 1.5: Example of a message definition

Create Phase

In the Create Phase, the actual flow (instructions on how to process a message in detail, including validation and transformations) is created. From the design phase, a basic flow is generated, which validates the messages and transforms them (as defined in the design phase). However, much more customization might be desirable. For example, one side could be sending XML messages, while the other end receives JSON. Or, one needs to fetch data from another source to successfully transform the data. eMagiz has made many of these wanted customizations available. An example flow is illustrated in Figure 1.7. When satisfied with the integration, one can define Unit Tests to validate the result of the integration.

Figure 1.6: Example mapping from CDM to message definition

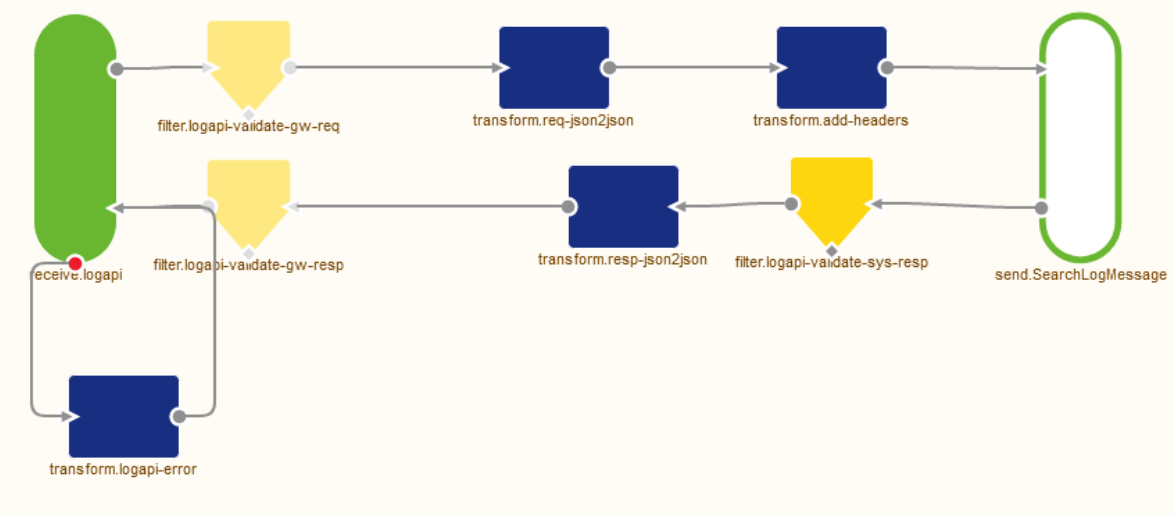


Figure 1.7: Example of basic API-Gateway flow generated by eMagiz

Deploy Phase

In the Deploy Phase, a release is created. This release is a collection of releases of integrations (such as the one in Figure 1.7). When the release is ready, it can go through several Test-, Acceptance-, Production- (TAP)environments. When Production is reached, the integration is deployed in the cloud. This process is fully automated. After a short while, the integration is deployed and ready to receive messages.

Manage Phase

In the Manage Phase, the customer can review the performance of his integrations. Log- and error messages are visible, as well as the current status of the flows and exceptions happening in the flows. Customers can also set alerts if wanted.

Chapter 2

Background

In this chapter the literature review will be summarized containing the most relevant information required for the research. Firstly, some basic concepts required which need to be at hand for the rest of the report are explained, like virtualization and containerization, Software-as-a-service (SaaS) and iPaaS. Later, we will dive into more detailed concepts like Kubernetes.

2.1 Virtualization

Virtualizing is the process of adopting cloud providers to host virtual machines and using those machines to run certain software. Using virtualized servers over physical hardware has many benefits, including the low CAPEX cost (initial investment cost) due to the pay-as-you-go models most providers use, the ability to quickly scale up or down, use different operating systems, and to break down large applications into smaller applications. This last benefit prevents monolithic software, which can become large, complex, and full of unsolvable technical debt[18]. Breaking these applications into smaller microservices prevents this. These smaller microservices can then be deployed on a virtual machine. Typically, these machines can adjust better to the resources needed by the microservices.

To go into more detail about virtualization, we first need to know what containerization is.

Containerization is usually confused with virtualization, but there are some fundamental changes. As virtualization talks about the infrastructure to deploy software on, containerization focuses more on the software to deploy. Since these microservices are usually deployed in virtualized environments, the context of the server can vary from OS to available resources. A containerized application or service is configured in such a way that all required libraries, binaries, and configuration files are present (or can be made present) in the container. Therefore, deploying containers becomes independent of the environment it is running in. This allows users to deploy their containers on all kinds of machines.

A container can exist in two states, running (as a container) and stopped (as an image or configuration file). When a container is stopped, it is a flat-file better known as the Container Image. The container image is like a blueprint how the running container should look like. To start a container, a runtime engine is needed that translates the flat Container Image into a running service. Many examples exist, like Docker, Container Runtime Interface for the Open Container Initiative (CRI-O), Linux containers (LXC) and Rocket (RKT). These engines prepare mount points for file storage, pull required images, and handles (user) requests, either through the command line or automated.

Now that we have a definition for containerization, we can come back to the more high-level concept of virtualization. Different types of virtualization exist, including: VM , Virtual

Container (VC), containers within VM, lightweight VMs and Unikernals. VMs are virtual machines with a hypervisor managing all the virtual machines running under it, including storage, network, and computing resources. These VMs are completely isolated from each other and have a high level of security. On the other side, the cost for management of VMs is high and takes long to instantiate. VC are smaller, cost fewer resources and time to instantiate, but are less secure and are less scalable. This difference is due to VCs not having dedicated bare metal resources, but sharing them with other containers[5]. Usually, however, this problem of shared resources is easier to solve than the startup time of VMs and the required bare metal resources.

Containers within VMs combine the positive aspects of both. Containers running inside VMs have high isolation due to the nature of the VMs (only containers of one tenant can run in one VM), while still having fast instantiate timers due to the nature of containers[5].

Lightweight VMs are similar to containers within VMs, but have an customized kernel, specifically designed for fast bootup. These kernels usually don't have legacy support and bootloaders. Specializing even more and removing all unncessesary resources is referred to as Unikernels. These kernels are specifically compiled for one specific task. These Unikernels are very fast, but lack customization.

All five options are interesting for different applications. For example, VMs are interesting for running servers due to their high level of isolation and flexibility. VCs are usefull when instantiation time and resource utilization is more important, but security isn't the greatest interest. Containers in VMs are interesting for hosting multi-tenant environments with many different containers. Lightweight VMs and unikernels are interesting for applications with very little variation in the tasks they perform. This is summarized in Table 2.1.

	Advantage	Disadvantage
VMs	Isolation, Security at system level	Increased cost and time for instantiation, migration
VCs	Lightweight, smaller footprint, less cost and time to instantiate. Isolation at the application level	Less secure, low networking bandwidth and scalability, performance interference
Containers inside VMs	Increased Isolation, security at both system and application level. Minimal Migration time and latency	Increased time to instantiate and boot
Lightweight VMs	Highly kernel dependent, less boot time	Single purpose applications alone served better
Unikernels	High Isolation, security, smallest footprint, portability and , interoperability, less power consumption	Single user, Single application bound

Table 2.1: Advantages of different virtualization strategies[19]

Virtualization solves most problems that arise with SaaS projects, like scalability, data integration, unified data access, and hosting flexibility[20]. Using virtualization, instances of the software solution can be easily deployed, scaled up or down, and maintained.

In the next sections, different as-a-service concepts will be explained. SaaS is the most strict service type allowing no customization, Platform-as-a-service (PaaS) allows to deployment of custom software, and Infrastructure-as-a-service (IaaS) allows for custom OS and middleware (see Figure 2.1). iPaaS is a combination of IaaS and PaaS solutions, offering an interface for defining integrations over the IaaS layer or over the PaaS layer, depending on the solution.

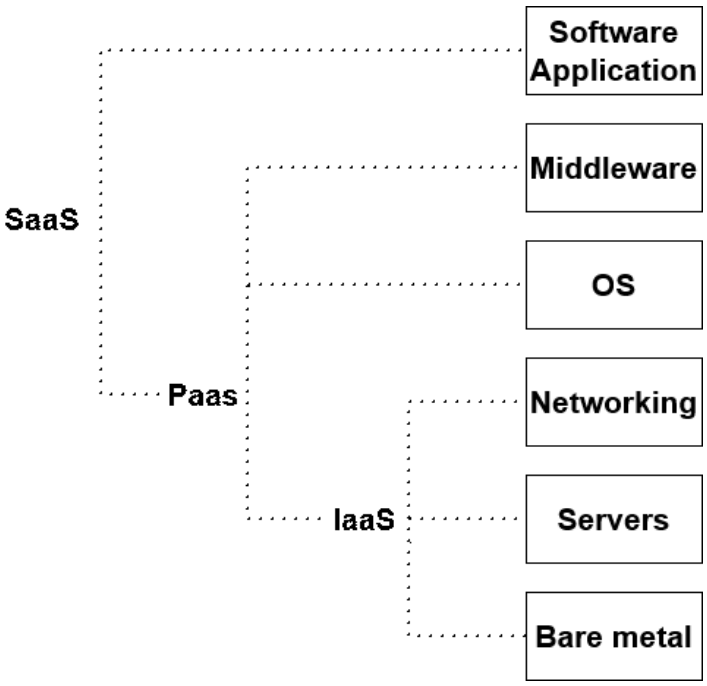


Figure 2.1: SaaS vs PaaS vs IaaS

2.2 SaaS

Since the rise of SaaS platforms around the dot com explosion in 2001, SaaS solutions have become an unmissable piece of technology in many different enterprises. Gibson states that *"Software-as-a-Service gives subscribed or pay-per-use users access to software or services that reside in the cloud and not on the user's device"* [21]. Thus, SaaS solutions are applications that you don't have to install. Rather, they are accessible through the internet. National Institute of Standards and Technology (NIST) [22], the National Institute of Standards and Technology of the United States, defined SaaS as *"The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings."* [23]. The most prominent benefit of adopting SaaS solution is the decrease of cost since it is low on initial cost, software, hardware, and staff [21]. The target audience for SaaS solutions is end-users. These users can access the application through a web interface. Together with the birth of SaaS solutions, many solutions relying on virtualization have also been introduced. This includes PaaS, IaaS and Desktop-as-a-service (DaaS) [24]. Another solution is the iPaaS.

2.3 PaaS

In contradiction to SaaS solutions, PaaS solutions don't offer any specific software application. Instead, the platform can be used to deploy applications on. For example, developers can use PaaS solutions to deploy their custom software on to test or release it. PaaS solutions, therefore, offer the hardware, some software, and some middleware for developers to use. SaaS solutions can be deployed and run on these PaaS solutions. NIST [22] defines a PaaS

solution as: *"The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment"* [25]. When abstracting even further, and omitting even more pre-installed components, IaaS solution offers infrastructure without for example databases and an OS. The next section will elaborate more on IaaS solutions.

2.4 IaaS

In contradiction to PaaS, IaaS does not offer much out of the box solutions. Typically, the physical layer (bare metal), the networking, and the firewall are present, but customers can install their own OS and tools on the infrastructure. Typical applications of IaaS solutions are the deployment of entire clusters of containers.

NIST [22] defines IaaS as: *"The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer can deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls)[26]."*

Similarly to PaaS solutions, iPaaS solution offer the hardware and software needed. iPaaS, however, focus more on defining integrations than general SaaS solutions, and have more specific software installed than PaaS solutions. The next section will elaborate more on iPaaS solutions.

2.5 iPaaS

According to Serrano (2014), an iPaaS *"is a suite of cloud services that enable users to create, manage, and govern integration flows connecting a wide range of applications or data sources without installing or managing any hardware or middleware."* [27]. Adopting an iPaaS solution is therefore interesting when many different services must be integrated, to manage all integrations centrally and not have to worry about hardware or middleware. NIST does not provide a definition for iPaaS.

Potocnik describes several characteristics and requirements of iPaaS solutions [28], as can be found in Table 2.2.

Taking into account the responsibilities, there are four main components an iPaaS should contain according to Ring and Ebert [29] [30], as can be found in Table 2.3.

2.6 Container Management

Container Management is *"a set of practices that govern and maintain containerization software. Container management tools automate the creation, deployment, destruction, and scaling of application or systems containers."* [31].

Container Management tools have a few parts in common, for example, orchestrator, resource monitor, scheduler, discovery service, state storage, and overlay network. [32]

Very often, the orchestrator already contains other parts of container management, like the scheduler and the state storage. Therefore, we will describe all aspects under the container orchestrator.

Characteristic	Description
Data integrity and security	Data should be complete and consistent, and be transported securely.
Data transformation and migration	iPaaS should allow for transformations between different data types and formats.
Connectivity	iPaaS solutions should allow connectivity using standardized connection patterns.
Governance and management	iPaaS solutions must have mechanisms in place to govern data and integrations.
Orchestration	Automatic configuration and management of applications. For iPaaS this is important for deploying solutions for applications.
Monitoring	iPaaS solutions should offer insight into the performance of the integrations.

Table 2.2: Characteristics of iPaaS solutions according to Potocnik[28]

Responsibility	Description
Integration processes	Specify the logic of how and when data are exchanged between applications.
Data mappings	Data mappings between the attributes of source and destination data objects.
Pre-built adapters	Pre-built adapters to connect to different types of applications.
Supporting elements	Functionalities to support the development of components 1–3.

Table 2.3: Responsibilities of iPaaS solutions according to Potocnik[28]

2.7 Container Orchestrator

Redhat describes container orchestration as: "Container orchestration automates the deployment, management, scaling, and networking of containers".[33] This means that container orchestration should take care of scaling up and down in terms of resources and replications, handle internal networking, deployment strategies. Khan describes seven capabilities of container orchestration platforms[6] (see Table 2.4).

Khan also gives examples of such platforms[6]

- Kubernetes
- Amazon's Elastic Container Service
- Mesosphere
- Docker Swarm
- Microsoft Azure (providing Kubernetes, Swarm and Mesosphere)
- Rancher OS
- Nomad

Rodriguez[34] has defined a reference architecture for container orchestration (see Figure 2.2). In this figure, *Jobs* are applications submitted by users. These jobs consist of *tasks* which are usually independent and homogeneous. *Resource requirements* are requirements in terms of Central Processing Unit (CPU) and memory they require to execute the job. *Quality-of-Service (QOS) requirement* defined fault-tolerances, time constraints and priorities

Cluster state management and scheduling	- Flexible scheduling of tasks across the cluster and supporting maintenance activities and provides the control mechanisms for algorithms. - Reliable state management and repartitioning of data or resources across the cluster - Informing dependent systems of changes so they can react appropriately to cluster changes - Throttling system task and changes
Providing high availability and fault tolerance	- Elimination of single points of failure by adding redundancy to the system - Reliable crossover from system A to system B when neccesary - Failure detection
Ensure security	-
Simplifying networking	-
Enabling service discovery	-
Making continuous deployment possible	-
Providing monitoring and governance	- For infrastructure like VMs - For containers

Table 2.4: Capabilities of container orchestration platforms[6]

for example.

The *Cluster Manager Master* consists of eight components. The *Resource Monitor* tracks the resource usage of each worker and component. Additionally, the *Accounting* component keeps track of metrics relevant to the owner of the cluster, like the number of jobs and energy consumption. The *Admission control* is responsible for measuring two metrics, i) whether the resource quota requested by jobs is being fulfilled, and ii) whether there are enough resources available in the cluster. The *Task Scheduler* maps tasks onto cluster resources. It does this by taking into account available resources and requirements of the task, and its priority. The *Task Relocator* can also be seen as a scheduler, which reschedules if anything happens, like a dying container, lack of resources, or excessive use of resources. The *Task Launcher* is responsible for starting a task on a specific worker based on the input of the scheduler or relocator. Lastly, the *Resource Provisioner* adds new workers either automatically or manually by an administrator.

For the *Compute Cluster*, each worker has a *worker agent* responsible for all worker processes like collecting metrics and starting containers. The *Container* is the host of the actual service and can be of various types.

2.8 Examples of iPaaS solutions

Many iPaaS solutions exist, all focussing on different aspects of the solution or different industries. Mulesoft is one of the leaders according to Gartner[35]. It offers extensive data integration tooling and API management. Their built-in CI/CD solution and integration testing make testing and deploying integrations low effort. Additionally, customers can choose to deploy locally or in the cloud offered by Mulesoft. Another iPaaS solution is Boomi[36], which focuses more on low-code development and uses AI to help the developer with Smart Data Mapping. Workato[37] is an iPaaS platform focusing on the amount of

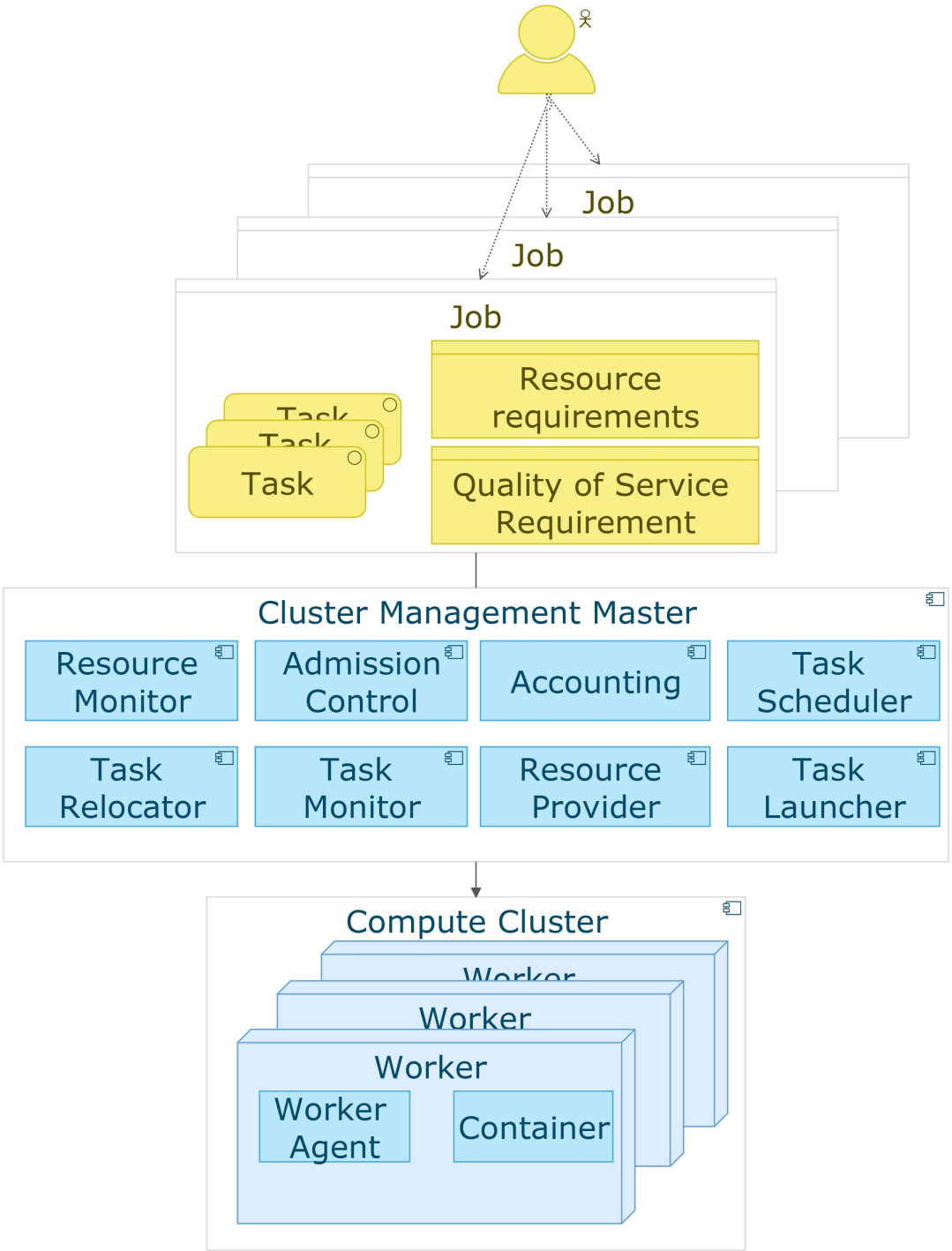


Figure 2.2: Reference architecture Container Orchestration according to Rodrigues[34]

prebuilt adapters. Workato has over a thousand prebuilt adapters, which customers can click together in their low-/no-code platform. Workato focuses on improving business processes by allowing integration with many accounting- and sales tools. Pricing plans are focused on large enterprises. Jitterbit[38] focuses more on the middle large customers. It also offers a low-/no-code platform. Jitterbit offers "a mix of cloud-based application, data, and process-integration capabilities offered at reasonable subscription charges"[39]. Other platforms include IBM[40], Snaplogic[41]

Chapter 3

Requirements

In this chapter, the requirements of the system will be defined. First, we will analyze the current situation by consulting literature and experts in the field.

Three interviewees agreed to help define requirements. Firstly, a team lead working for an iPaaS provider. Secondly, a Chief Technical Officer (CTO) at an iPaaS Provider, and lastly a Software Delivery Manager at an iPaaS Provider.

Firstly, the requirements are summarized, after which the requirements will be elaborated in more detail. The reader is made aware that requirements will be quantified and be made measurable in Section 3.4.

3.1 Functional requirements

Functional requirements are defined for the target solution to determine whether the solution solves the problems identified. The requirements are separated into four categories; Isolation- and Security requirements, Pricing requirements, Organizational requirements, and Responsibility requirements.

3.1.1 Isolation and Security Requirements

iPaaS solutions usually facilitate multiple customers. These customers buy an iPaaS solution with the understanding their solution will simply work. If they mess up, their integration can be affected, but for others to mess up should not interfere with the workings of their solutions. This raises the requirement that *Customers **must** not be able to decrease the performance of other customers.* With the assumption or agreement of no disruption, most iPaaS solutions also want to isolate and obfuscate references to other customers, even if they are not accessible. From here, the requirement *Customers **must** not be able to identify other customers on the solution* is formulated.

As for security, there are also some requirements. Secure Sockets Layer (SSL) was initially made for clients to secure against man-in-the-middle attacks [42]. This is done by verifying the certificates of the server by the client. for iPaaS solution, it is as important to trust the client as the client trusting the server, and thus instead of SSL, Two-Way SSL is recommended, and thus *Industry-standard security **must** be allowed, like two-way SSL and currently acceptable encryption solutions.*

On the other hand, iPaaS solutions are often used to connect legacy systems [27]. Since iPaaS solutions connect such legacy systems, iPaaS solutions should be able to handle older policies, to not lose support for these systems. How far a solution should go with this requirement is dependent on future and current customers and thus since the solution is typically middleware, also *older standards **should** be allowed up to a certain degree.*

Standard in the industry is to have three lanes for a solution, Testing, Acceptance, and Production. Production should be as stable as possible with no to nearly no downtime. Acceptance is an exact replica of production to test a future deployment on production but does not handle the workload of production. Testlane is a lane that is, as the name suggests, for testing purposes. This *can*, depending on the policy of the iPaaS Solution, include load testing, where the goal is to understand how much load a system can take. To not stress the Production environment, this usually happens on Testing or Acceptance lanes. Therefore, it would be ideal to *have Testing and Acceptance on different clusters. If not possible, these load tests must not interfere with the working of the cluster.*

Lastly, most likely the solution will contain some clustering strategy. This is a common discussion in all fields like the energy supply field[43], the Electrical Engineering field[44] but also the in the field being discussed now. Clustering strategy contains tradeoffs when choosing more workers per cluster or more clusters for the number of nodes[45]. This itself is worth researching further. However, the interviewees mentioned that the number of individual clusters should be low to reduce cost and have a multi-tenancy solution, and thus *in terms of nodes per cluster ratio, the number of clusters* ***should*** *be low.* (Multi-tenant).

3.1.2 Pricing and Organizational Requirements

Setting a budget for a proposed solution helps determine what components can be in and which are not possible due to budget pressure. However, interviewees have mentioned that the initial setup is expected to cost quite a substantial amount due to the complexity of the process. For this, the interviewees suggested working with an estimated cost for a solution instead of working with a budget. *For the OPEX, it should be close or lower to the current OPEX cost.* For the entire solution, *an estimation of the solution cost* ***must*** *be made.*

Interviewees mentioned that for their businesses, there was no need for a 'pay as you go' model, where billing is done based on the exact (virtual) hardware usage. Instead, they work with packets, where a customer buys the solution, and with that buys a certain amount of hardware. Therefore, *Metrics like Memory (Random Access Memory (RAM)) and CPU usage* ***can*** *be available.*

As mentioned in Section 1.2, the adoption of cloud solutions including container orchestration can introduce vendor lockin. Most providers (like Azure, Amazon Web Services (AWS), and Google) provide solutions that introduce (partially) vendor lock-in. Interviewees mentioned that the current dependency on one cloud provider is not desired and thus vendor lock-in ***should*** *decrease vendor dependency/lock-in*

Lastly, *the solution* ***must*** *be able to scale, but this does not have to happen automatically.*

3.1.3 Responsibility requirements

To determine which party is responsible, requirements are defined. Firstly, as mentioned in Pricing and Organization Requirements, customers work with packets. This means customers can use the totality of the packet and is responsible for scaling up and down within the packet. These packets have been determined together with consultants and thus should offer sufficient capacity. However, *if an increase of packets is required, this should be done by the iPaaS provider.*

iPaaS solutions provide some type of interface to handle deployments of solutions. Wherever these solutions will be hosted (public/private/hybrid cloud or on-premise), the portal should be leading. This means that if a customer has access to their solution in the portal, they should also have access to the web environment. (This does not mean they can change the environment, but users which are set to have access in the portal should be able to access parts of the solution in the cloud as well).

3.1.4 Summary of functional requirements

Isolation and Security

:
- Customers **must not** be able to identify other customers and access their data on the solution (IS1).
- Customers **must not** be able to decrease the performance of other customers (IS2).
- Lanes (Testing & Acceptance, Production) **should** run on different clusters. At least Testing & Acceptance performance **must** not interfere with Production performance (IS3).
- Industry-standard security **must** be allowed, like two-way SSL and currently acceptable encryption solutions (IS4).
- Older standards **should** be allowed up to a certain degree (IS5).
- In terms of nodes per cluster ratio, the number of clusters **should** be low. (Multi-tenant) (IS6).

Pricing

:
- An estimation of the solution cost **must** be made (P1).
- Operational costs should be similar or lower than the current OPEX costs (P2).
- The billing strategy is based on a packet, and not on CPU or Memory usage, so these metrics **can** be available (P3).

Organizational

:
- Deployment of the solutions **should** decrease vendor dependency (O1).
- Horizontal scaling **must** be possible, but does not have to happen automatically (O2).

Responsibility

:
- Increasing packet sizes and thus maximum available resources are up for the iPaaS provider (R1).
- Initial setup can be complicated and **can** require specialized personnel, but day-to-day activities **must** be executable for non-experts of the software (R2).
- The iPaaS solution is the source of truth, and thus all settings **must** be configured in the portal and the cluster **must** act upon pushed from the portal (R3).
- The iPaaS solution is responsible for user credentials, but customers **must** be able to access their deployments with these credentials where ever the solution is hosted (R4).

3.2 Non-functional requirements

Other than functional requirements, several non-functional requirements were applicable. These have mostly to do with disaster recovery, maintenance, cost, and performances .

3.2.1 Disaster recovery

Disaster recovery is important to define requirements about since, in Subsection 1.2.1, *little downtime as possible, and no loss of data* is described as one of the goals of both the customer and the iPaaS provider. These requirements define how much risk is mitigated or accepted. For this aspect, we look at the load balancers and the workers.

Load Balancer (LB)s are required to enable access to the solution. LBs are a gatekeeper to the entire solution. Therefore, these load balancers should be Highly Available (HA). When

one LB fails, another should take over. We assume here the major cloud providers have reasonable Service Level Agreement (SLA), and thus define the time *when a load balancer fails, it **should** take no more than the time equal to cloud provider standards.* For workers, the same applies. When one worker fails, another should take over. However, since this is not the gatekeeper of the solution, but rather one of the many workers, the severity of the failing node is less worrying. Since health probes need to expire, containers need to be torn down and spun up again, *the maximum time for workers to be replaced **should** be 180 seconds.* Furthermore, underlying cloud environmental components could fail. This could cause containers to crash and thus lose data. Guaranteeing zero loss of data is not reasonable, and thus we use the same availability as cloud providers use. Therefore, *a worker **must not** lose more than 0.05% of its messages.* Lastly, a for the master plane of the cluster. Again, this value is set to the cloud standards. Therefore, *the availability of the master plain **must** be equal to or more than 99.95%.*

3.2.2 Cost

Concerning changes in cost, certain requirements are determined. This is to ensure the solution will not be significantly more expensive than desired. Three states are defined: idle, where the cluster is not ready to accept messages, but costs are made due to buying in certain aspects of the solution, unused, where there is no workload but the cluster can start working, and running.

- At an idle state, the cluster may at most cost 400 Euro (C1)
- At an unused state, the cluster may at most cost 600 Euro (C2)
- The increase of cost per customer (including cluster overhead) may at most be 5 % of the old situation (C3)

3.2.3 Maintenance

Except for the cost of the solution, maintenance costs should also be taken into account. This can, for example, be updating the cluster about security or performance or checking alerts from the solution.

We define two different types of maintenance, day-to-day, and expert maintenance. Day-to-day maintenance is maintenance performed by any developer, who does not need to have specific knowledge. This includes checking cluster health, handling alerts, and monetization. Expert maintenance is maintenance performed by someone with specific skills, like updating the framework version or managing worker-/node groups. As defined in Section 1.6, we assume a small company with less than 50 employees. Therefore, available time for additional tasks is low. We assume that *Day-to-day cluster maintenance **should** take at most 8 hours per month* and *Expert cluster maintenance **should** take at most 8 hours per month*

3.2.4 Performance

Lastly, non-functional requirements for performance are defined. This is with regards to processing speed. This is defined based on the time for the cluster to adapt to change requests. *When a chance in replication is requested, this change **should** take at most 180 seconds.* This number is based on requirement DR2.

3.2.5 Summary of non-functional requirements

Disaster recovery

- When a load balancer fails, it should take at most the amount of seconds in line with cloud providers' SLAs before load balancing is continued (DR1).
- When a worker fails, it may take at most 180 seconds before containers are available again (DR2)
- When a worker fails, it may lose at most 0.05% of messages(DR3)
- The availability of the cluster should be at least 99.95% (DR4)

Cost

- At an idle state, the cluster may at most cost 400 Euro (C1)
- At an unused state, the cluster may at most cost 600 Euro (C2)
- The increase of cost per customer (including cluster overhead) may at most be 5 % of the old situation (C3)

Maintenance

- Day-to-day cluster maintenance should take at most 8 hours per month (M1)
- Expert cluster maintenance should take at most 8 hours per month (M2)

Performance

- When a change in replication is requested, it should take at most 180 seconds (Pe1)

3.3 Dependencies between requirements

Between the requirements there exist dependencies. For example, the requirement to not be able to identify other customers connected to the cloud solution (IS1) is in contradiction (or at least conflicts with) the requirement to have a low number of cluster to nodes ratio (IS6). This is because having a cluster per customer would solve IS1, but cannot be executed because of IS6. All dependencies are given in Table 3.1 and 3.2, and Figure 3.1. The table and figure show that some conflicting requirements are in place. In the design phase, these requirements must be weighed when choosing an available option. There are also dependencies between the non-functional requirements within the disaster recovery section.

Additionally, we see that most of the requirements influence the cost, as expected. However, since any function might increase cost, these dependencies have been omitted in the figures. Apart from this dependency, there are no dependencies between functional and non-functional requirements.

Relation \ Requirement	**IS1**	**IS2**	**IS3**	**IS4**	**IS5**	**IS6**	**P1**	**P2**
Conflicting	IS6	IS6	IS6, R2	IS5	IS4	IS1, IS2, IS3		
Strengthening		IS3, R1					P2	P3
	P3	**O1**	**O2**	**R1**	**R2**	**R3**	**R4**	
Conflicting			R2	R2	IS2, O2, R1			
Strengthening				IS2		R4	R3	

Table 3.1: Dependencies between requirements

Relation \ Requirement	DR1	DR2	DR3	DR4	C1
Conflicting					
Strengthening	DR4	DR3	DR2	DR1	
	C2	**C3**	**M1**	**M2**	**Pe1**
Conflicting					
Strengthening					

Table 3.2: Dependencies between requirements

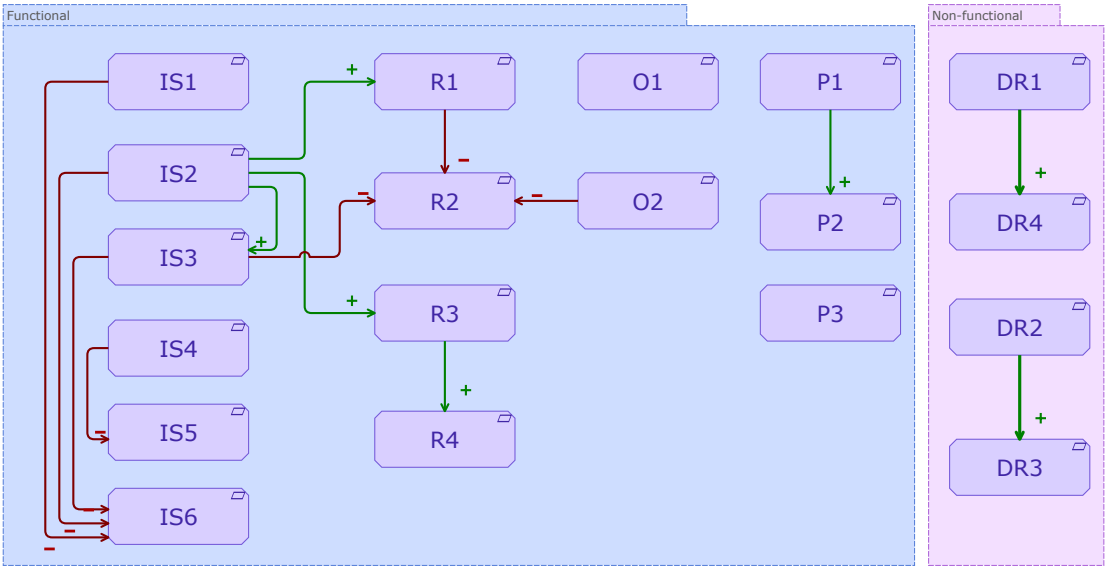


Figure 3.1: Dependencies between requirements.

3.4 Measurability of the requirements

To determine whether the solution has fulfilled the requirements specified, a method of assessing the requirements must be defined. Most will be trivial, but some require a specification of how the values should be measured. Requirements that require no measure definition (like IS1, which is fulfilled if no method of finding other customers can be found) are not described in this section.

The term performance is used in **IS2**, **IS3** and **Pe1**. Performance is defined as the average execution time of a request. This includes overhead. For this comparison, an equal amount of Virtual CPU (vCPU) and RAM should be allocatable for the request. This means that the server should not be pressured and thus be able to deliver less performance. Furthermore, the limitations set on the containers performing the request should be equal. Thus, both situations should be equally capped or have unlimited (at least excess) resources.

IS5 mentions a *a certain degree* of older standards should be allowed. This requirement is impossible to fully define. However, what is meant here, is that there should be options to allow older standards, which are not recommended anymore. For example, CloudFlare published 11.36% of the websites are using Transport Layer Security (TLS)v1, while TLSv1.3 is out already[46]. Knowing iPaaS solutions act as middleware for legacy systems, standards like TLSv1 should be supported. Which standards should be allowed is impossible to define

due to the many standards present in terms of encryption, security tokens, and more.

IS6 defines the *number of nodes per cluster* should be low. This means that a multi-tenant solution should exist where multiple customers should be able to work on the same cluster. However, no value was defined for this requirement. For this requirement, the business perspective to add a new cluster should be the reason to start a new cluster, not the limitations of the cluster.

P2 requires comparison of current and future OPEX cost. For this, we use the hypothetical case of 10 customers in the old situation with an average usage comparable to the existing situation. The total OPEX cost is compared to the OPEX cost in the new situation. The reason to use 10 customers is to share the possible bigger overhead cost across more customers and have a more realistic estimation. Due to the nature of container orchestration (sharing resources and thus reducing OPEX cost), we can assume more customers mean that the OPEX cost of the new solution will only decrease more.

O1 mentions a *decrease in vendor dependency*. This can be achieved in two ways. Either the solution can be run across multiple hosts (e.g. a master plain with AWS, and workers with Azure), or the option to port the full solution on multiple cloud providers. E.g. running the solution on AWS and Azure requires minimal changes. For the non-functional requirements, **DR2** mentions an amount of time *before containers are available again*. This can be tested by tearing down a container manually and recording the time it takes for the cluster to start a new container. This includes the time to detect a container stopped working, using a reasonable liveliness check according to the specification of the solution.

Furthermore, **DR3** mentions a loss of messages. We define these as messages which are impossible to reproduce. For example, some messages might be stored on queues and since no response was given by the worker, a retry is executed. These are not lost messages. However, when messages cannot be reproduced, these are lost packages.

3.5 Relation to the business motivation

In section 1.2, the problem statement and the motivation view have been described. To show that the requirements directly contribute to the goals identified, this section will describe the relations.

In Figure 1.2, a motivational view was already created. In this chapter, requirements were defined. These can be combined to show the relevance of the requirements. Figure 3.2 shows the relations between the requirements and the motivational elements. Some will be elaborated on below. However, elaborating all of them would be most trivial or would add little value to the reader.

P2 defines that current OPEX cost should be lower or similar to the current OPEX cost. This can be directly related to the goal of the iPaaS provider to reduce cost. This then contributes to the profitability driver of the iPaaS provider.

O1 defines the requirement that vendor dependency should decrease. Decreasing vendor dependency also reduces the vendor lock-in goal. The requirements define that the solution should be able to run on multiple cloud providers, and thus the solution can be ported to different vendors.

One of the main cost drivers of businesses without container orchestration is the wasted resources. Usually, VMs are bought but not fully utilized. On the other hand, when more computing resources are needed, this requires manual intervention. **O2** therefore defines this manual intervention should not be necessary, and therefore contributes to the goal of flexible scaling, but also contributes to the goal of being able to give more resources in times where a peak in processing power is required.

In terms of isolation, **IS1** prevent customers to detect each other, and **IS2** prevent customers from influencing each other. These two requirements contribute to the goal to provide isolated environments.

Lastly, **R3** and **R4** define the portal should be the source of truth both for credentials of the customer as for the infrastructure, independent where the solution is deployed. These two requirements thus relate directly to the goal of a central place to manage the integrations.

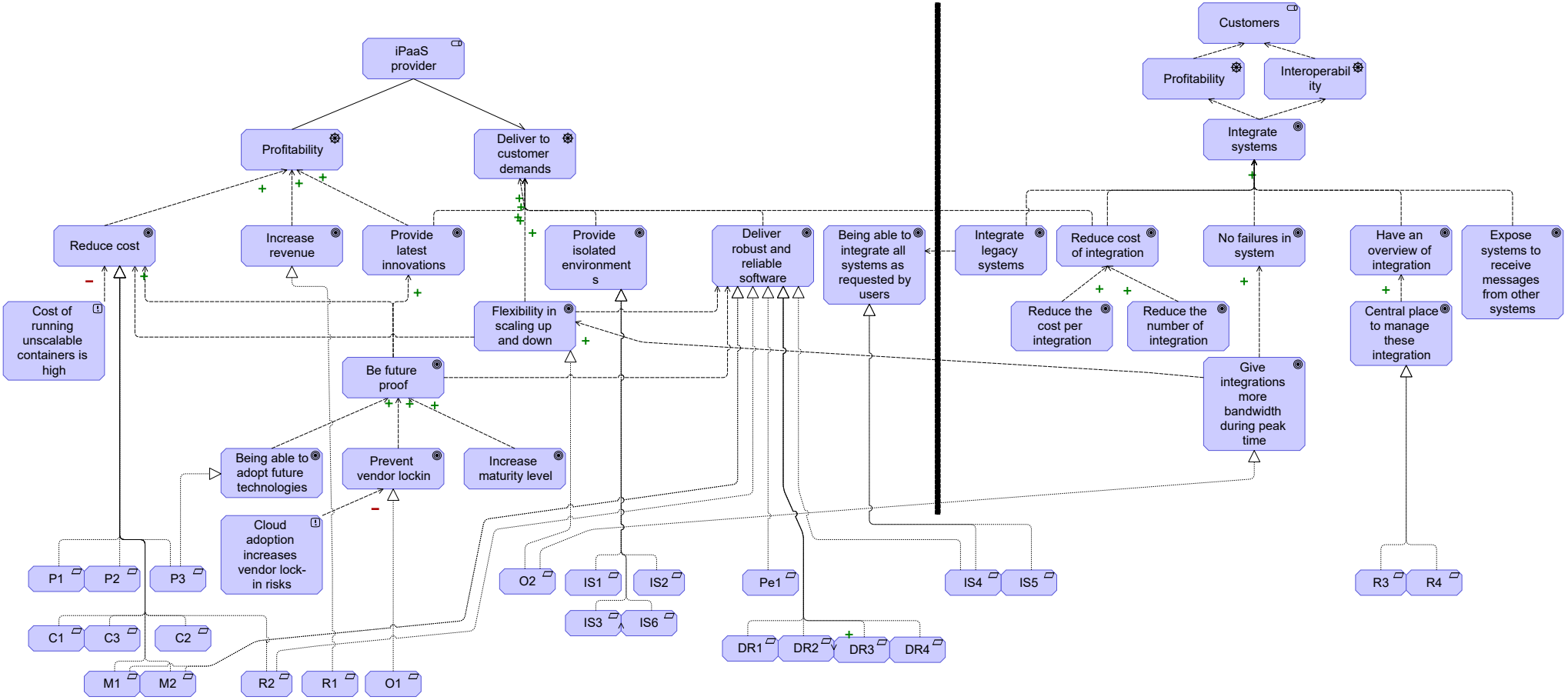


Figure 3.2: Requirements and motivation view combined

Chapter 4

Container Orchestration Solutions

In this chapter, possible container orchestration solutions are discussed, what their benefits and disadvantages are, and how they relate to the requirements. This chapter will regard Kubernetes, Docker Swarm, Spring Boot Skipper, and Apache Mesos (and related).

4.1 Kubernetes

In this section, we describe how Kubernetes works and what requirements it has.

Kubernetes, or K8s (omitting 8 letters between the K and s), is an open-source system for automating deployment, scaling, and management of containerized applications[47]. Kubernetes consists of different components with their own task. To discuss the components, we need to discuss some concepts and naming conventions.

4.1.1 Kubernetes Components

When deploying Kubernetes, a (Kubernetes) **cluster** is created. This cluster is the entirety of all components working together. Within the cluster, at least one worker should be present. This is usually one server or VM. In Kubernetes, one such worker is called a **node**. Within a node, multiple **pod**s can be run. Pods are representations of (multiple) running containers. Next to worker nodes, one or more master nodes, also known as control-plane, is present. The master plane manages the worker nodes; it starts new workers if needed, and can be spread over multiple nodes for high availability. Within the master plane, multiple components are present:

- **API Server**: The API Server is the cluster gateway. This is the entry point for communicating with the cluster, like deploying new services. The API Server also acts as a gatekeeper by verifying authentication.
- **Scheduler**: After validating the request by the API Server, the scheduler schedules a process on a specific worker node based on certain factors like resources available. Note that the scheduler only decides *which worker* the process will be placed. The actual spinning up is done by a different process.
- **Controller manager**: Detects cluster state changes, like crashes or failures. It will try to reschedule a new pod if needed by making a request to the scheduler.
- **etcd**: Also referred to as the cluster brain. It is a key-value store of the state of the cluster, holding information like cluster health and available resources. Note that no application data is stored here.

Usually, when the cluster is deployed on the cloud, master nodes are managed by the cloud provider. This means they aren't viewed as separate nodes but as part of the cluster

settings. Additionally, to master nodes, worker nodes exist. Within the worker nodes, multiple components are present:

- **Kubelet**: Runs the actual pods inside a node.
- **Kube proxy**: Network proxy which to handle requests from the API Server.
- **Container runtime**: Software responsible for running the container. These runtimes should be CRI (Container Runtime Interface) compliant. Examples are CRI-O[48], RKTlet[49] and frakti[50].

As for communication, **Kube Control (kubectl)** is a command-line tool to interact with the cluster. Using kubectl you can communicate with the API Server. Other than kubectl, graphical interfaces and direct API connections are possible. However, kubectl is the most powerful.

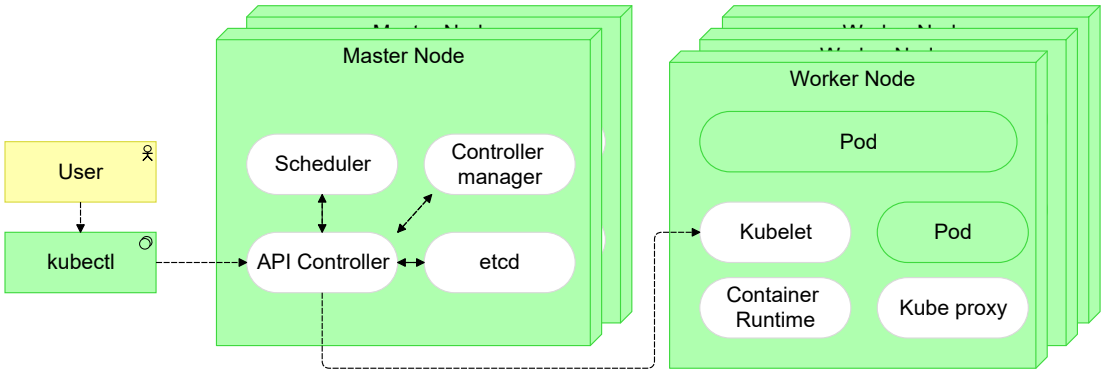


Figure 4.1: Kubernetes componenets

Everything mentioned so far is required to set up a cluster. The next objects explained are optional to configure the cluster to ones liking. The objects are split between Workload Objects, Service Objects, Config- and Storage Objects, and Cluster Objects

The following Objects are a subset of available Workload Objects:

- **Pod**: As mentioned before, pods are the equivalent of a (set of) running container(s).
- **Deployments**: Even though pods are the actual running containers in Kubernetes terms, users will not directly interact with pods. Deployments are the blueprint of how a pod should look like and is the way of creating pods for users. Deployments require container images to set up the container of the pods.
- **ReplicaSet**: To manage the replicas required from the deployment, there is another layer between a deployment and a pod, called a ReplicaSet. A ReplicaSet starts the required number of pods defined in the deployment. Users will usually not touch replica sets.
- **StatefulSet**: Persistent storage is handled by *volumes*, but having replication has consequences for stateful storage. For that, StatefulSets are created, which are API objects used to manage stateful storage. StatefulSets are also meant for stateful applications and are similar to deployments.
- **DaemonSets**: DaemonSets are sets that deploy pods to every node. Examples of uses for daemon sets are network services or log services that should exist on every node. The network Domain Name System (DNS) is such a service that lives in DaemonSets.
- **Jobs** and **CronJobs**: Jobs are short-living pieces of executable code running once or scheduled. One example could be to run a single unit test. CronJobs run repeatedly on a predefined interval.

- **Taint**: Taints are labels to determine what objects can be scheduled on what infrastructure. For example, a NodeGroup can contain a taint to only allow pods with the same toleration to be scheduled on the node.
- **Toleration**: Tolerations are labels to determine how to handle taints. Tolerations could contain information to allow certain services to be scheduled on tainted nodes.

The following Objects are a subset of available Service Objects:

- **Service**: Every time a pod starts, it gets assigned a new internal IP address. This will start causing problems when a pod restarts, for example when a connection is hardcoded. Therefore, you can attach a service to a pod. A service has a permanent IP address and lives separate from the pod. This means if the pod dies and restarts, the service will connect to the pod again and stay the same. There are four types of services.
  - *ClusterIP*: Exposes the service on the internal network on a specific cluster IP. Default for services.
  - *NodePort*: Exposes the service externally on `http(s)://<NodeIP>:<NodePort>`. Users can choose on what Node the request lands.
  - *LoadBalancer*: Exposes the service externally on a load balancer. The load balancer redirects requests to nodes.
  - *ExternalName*: Exposes the service externally on a `CNAME` record. Proxies need to be created manually.
- **Ingress**: Instead of Internet Protocol (IP) addresses, usually Uniform Resource Locator (URL)s are in a human-readable format. To translate the URL to IPs, an Ingress can be configured. Ingress are definitions (configs) of how to forward the human-readable URL to IP addresses or service. Additionally, an **Ingress controller** will then handle the forwarding to the service from the readable URL to an IP or service by collecting all Ingresses and deciding how to forward these messages. Ingress Controllers can be set up as third-party plugins.

The following Objects are a subset of available Config and Storage- Objects:

- **ConfigMap**: A ConfigMap is a set of key-value pairs of configuration settings. Note that this is stored in plain and thus should not contain sensitive information.
- **Secret**: Essentially the same as ConfigMap, but made to store secret information.
- **Volume**: Kubernetes does not handle resource allocation for storage. Instead, volumes should be added to Kubernetes for pods to store data. Volumes are non-persistant.
- **PersistantVolume**: Unlike Volumes, PerstistentVolume (PV) survive reboot by living separated from pods. A PV is Volume that is added to the cluster by administrators for pods to use either statically, or dynamic by the cloud provider based on StorageClasses. These PVs can be claimed and released.
- **PersistantVolumeClaim**: PerstistentVolumeClaim (PVC)s are the method for pods to request a piece of the PV. A PVC can requests different types of storage and thus define request properties, like an amount, access type, and StorageClass.
- **StorageClass**: Storage classes are methods to dynamically provision PVs. Depending on the setup, cloud providers usually offer different types of storage classes.

The following Objects are a subset of available Cluster Objects:

- **Namespace**: Kubernetes allows separation of resources (deployments, services) based on namespaces. This allows for multi-team or multi-tenant setups. Note that not all components are namespaced. For example, Nodes and PersistantVolumes are not namespaced.

- **(Cluster) role**: Roles are additive permissions for certain resources (get, watch, list, create, delete, patch, update). Cluster roles are the same, except they are not namespaced.
- **(Cluster) role binding**: Role binding binds certain roles to users, groups, or service accounts. Identically to Cluster Role, Cluster Role Binding is not namespaced.
- **Node**: As mentioned before, a node is a machine (bare metal or VM) that is either a worker or master node.
- **Service account**: While users have (something similar to) user accounts, services have service accounts to grant permissions to certain resources.

4.1.2 Kubernetes Requirements

Installing Kubernetes on a production environment involves several steps and requirements. To set up a cluster, one must install some kind of deployment tools `kubeadm` or `kops`. Which one is best depends on the type of deployment that is desired. Kops for example also creates the underlying VMs and servers on AWS, while Kubeadm assumes the infrastructure is in place.

Kubernetes has the following requirements for its nodes:

- A compatible Linux host.
- 2 GB or more RAM per host.
- 2 CPU cores or more per host.
- Network connectivity between all nodes.
- Unique hostname, Media Access Control (MAC) address, and product Universally Unique Identifier (UUID) for every node.
- Certain ports must be open.
- Swap must be disabled (To be changed in future versions >1.22).

For a minimal setup, one master node is required.

The required ports (Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)) can be found in table 4.1.

Master/Worker	TCP/UDP/Both	Port number	Direction
Master	TCP	Inbound	2379-2380
Master	TCP	Inbound	6443
Master	TCP	Inbound	10250
Master	TCP	Inbound	10257
Master	TCP	Inbound	10259
Worker	TCP	Inbound	10250
Worker	TCP	Inbound	30000-32767

Table 4.1: Required ports by Kubernetes

Additionally, Kubernetes has some boundaries of what a cluster can handle as listed below. More clusters can be created to overcome this problem:

- No more than 110 pods per node
- No more than 5000 nodes
- No more than 150000 total pods
- No more than 300000 total containers

Additionally, one of the requirements of Kubernetes is to have a network plugin installed to handle more complex networking operations like IP Filtering, traffic monitoring, etc. Kubernetes allows third-party plugins to be installed. These plugins should adhere to the Container Network Interface (CNI). These plugins might require additional port specifications. A (partial) list of third-party plugins is provided here[51].

Lastly, the cgroup driver[52] is used to determine available resources. Kubernetes (kubelet specifically) uses systemd by default. Other tools (like Docker) should maintain the same cgroup driver to prevent overestimating resources. Switching cgroup drivers late in the process has shown to be problematic[53].

So far, production environments have been discussed. In addition, several tools exist for development deployments. Minikube[54] spawns a single container in docker and deploys a single node on the Minikube cluster. Minkube is especially useful for development purposes since it is lightweight and comes with the ability to set up an entire cluster with a single command. Kind[55] is similar to Minikube but allows for multiple nodes in form of docker containers.

4.1.3 Kubernetes Cloud Providers

The above configurations focus on on-premise solutions. However, many solutions exist from cloud providers. Examples are:

- (Amazon) Elastic Cloud Kubernetes (EKS) from Amazon[56]
- Google Kubernetes Engine (GKE) from Google[57]
- Azure Kubernetes Service (AKS) from Azure, Microsoft[58]
- OpenShift from RedHat[59]

These Kubernetes-as-a-service providers allow for easier management of the cluster instead of having to manage the cluster yourself. These providers usually have an interface or command-line tool to automate the creation of clusters. For example, EKS can be configured using the AWS portal. Additionally, `eksctl` from Weaveworks[60] can be used to configure AWS EKS taking some configuration tasks away from the developer.

Due to the scope of the research, we cannot dive into the performance factors of all platforms available. However, Pereira has described the performance of three platforms, being EKS, GKE and AKS[61] [56] [57] [58]. Therefore, we consider these three platforms as possible options for hosting K8s clusters.

Performance

Pereira describes that for network-intensive applications, GKE has the best performance. For CPU-intensive containerized applications, AWS has the best performance. For a general-purpose solution, no comparison exists, due to the many different options possible. These options are heavily dependent on the goals of the solution. The results are based on comparable Intel Xeon E5-267x CPUs and 8GiB of RAM.

Cost

As for the cost of the clusters, the three platforms offer similar prices. EKS offers a price of $0.10 for each cluster per hour. Additionally, (Amazon) Elastic Compute Cloud (EC2) instances can be added to the cluster as workers. A full list of instances is available here[62]. Which type fits best is dependent on the type of workload. Note that additional sources, such as storage, load balancing, etc. might be required.

Req	Yes/No	Explaination
IS1	Yes	By using RBAC to disable list operations for sensitive resources
IS2	Yes	Using resource limits for pods or quotas for namespaces
IS3	Yes	Setting up a different cluster is possible, but by separating lanes with namespaces, quotas should be sufficient
IS4	Yes	Nginx Ingress can handle Client Certificate Authentication . Nginx is widely used for handling Hypertext Transfer Protocol (HTTP) requests in the industry already.
IS5	Yes	Nginx Ingress has (disabled) functionalities to support older standards
IS6	Yes	Kubernetes can scale up to 5000 nodes per cluster
P1	?	*Not possible to determine at this point*
P2	?	*Not possible to determine at this point*
P3	Yes	Many solutions exist like Prometheus
O1	Perhaps	This depends on the implementation of the cluster, but is possible
O2	Yes	Horizontal scaling is possible
R1	Yes	Using quotas for namespaces an administrator can set boundries. Pods scale outmatically within this boundry
R2	Perhaps	Depending on the set-up most can be automated
R3	Yes	Using the YAML config files an administrator can manage the entire cluster
R4	Yes	Using YAML files, one can create a user on the cluster

Table 4.2: Fulfilment of requirements of Kubernetes

For GKE, the same base price applies of $0.10 for each cluster per hour. Additionally, the price for workers needs to be added on top of this and additional resources. Information can be found here [63].

Other than GKE and EKS, AKS does not ask for a cluster management fee, and only asks for additional costs for the VMs and additional resources. Prices can be found in the Azure pricing calculator [64].

Limitations

Many limitations exist when adopting one of these providers. For example, EKS natively adopts the Amazon Virtual Private Cloud (VPC) Container Network Interface (CNI). This CNI cannot be used outside of the Amazon VPC. Amazon claims to have close relations with four partner products, Calico, Cilium, Weave Net, and Antrea. Amazon suggests taking support when these other CNIs are used.

GKE on the other hand does not use CNI plugins and uses default kubenet. Azure offers the Azure CNI and kubenet.

4.1.4 Satisfaction of the requirement

The requirements defined in Chapter 3 are reflected upon to see if Kubernetes could fulfill them. The results are in Table 4.2. In this table, we see almost all requirements can be met, with the exception of *O1* and *R2*. These depend on the implementation of the solution, meaning that there exist options to fulfill the requirement, but might not be possible due to the implementation. Additionally, *P1* and *P2* are not discussed, since these require a solution before a cost estimate can be given.

4.2 Docker Swarm

Docker Swarm is, similar to Kubernetes, a container orchestration software. It is managed by Docker. It can automate deployment, manage to scale, and apply self-healing. In terms of architecture, it is similar to Kubernetes. It uses master and worker nodes to define the hierarchy between nodes and the separation of responsibility.

4.2.1 Requirements

Installing Docker Swarm introduces some requirements to the setup. The following requirements need to be met. The required ports required can be found in Table 4.3:

- A compatible Windows, Linux or Mac host with Docker installed (Docker Desktop or the packages docker-ce, docker-ce-cli and containerd.io)
- Network connectivity between all nodes
- Certain ports must be open
- Absolut minimum of 512MB RAM, 2GB recommended
- Optional: If desired to create an overlay network with encryption, IP Protocol 50 (Encapsulating Security Protocol (ESP)) should be enabled.

Master/Worker	TCP/UDP/Both	Port number	Direction
Master & Worker	TCP	Both	2377
Master & Worker	Both	Both	7946
Master & Worker	UDP	Both	4789

Table 4.3: Required ports by Docker Swarm

4.2.2 Docker Swarm Cloud Providers

Unlike Kubernetes, Cloud providers do not advertise with a Docker Swarm service. Since the setup of Docker is very autonomous and has all required components out of the box, deploying it on bare VMs is not unusual. Docker has compatibility with all major cloud vendors in their Command-line Interface (CLI) (command-line interface), which creates containers on the cloud platform.

4.2.3 Cost

Docker Swarm comes with docker and is free of charge. Three main charges apply additionally, which are the cost of the servers to run the Docker (master/worker) nodes, the optional use of Docker Trusted Registry for storage of images for on-premise solutions, and Universal Control Pane for managing nodes in on-premise situations. The nodes are usually ran on EC2 like instances, of which the prices can be found here [64] [63] [65].

4.2.4 Satisfaction of the requirement

The requirements defined in Chapter 3 are reflected upon to see if Docker Swarm could fulfill them. The results are in Table 4.4. In this table, we see that *IS1* and *IS2* cannot be fullfilled. This since Docker Swarm does not support multi-tenancy. For this reason, we see *R1* and *R4* are also not supported. As for *O2*, this is not supported, since auto scaling is not possible.

4.3 Spring Cloud Skipper

Spring Cloud Skipper (Skipper) is an initiative of Spring (by VMWare). It originated from Spring Cloud Data Flow but was migrated to a separate tool. The goal of Skipper is to provide an abstraction layer between Spring Boot applications and Cloud Providers. Skipper can communicate with a local environment (for development or on-premise solutions), with Cloud Foundry (CF) (a PaaS solution for entire application stacks), or with Kubernetes.

Req	Yes/No	Explaination
IS1	No	Docker Swarm does not provide the tools to setup multi tenancy
IS2	No	Since no multi-tenancy exist, there is no viable way of isolating customers
IS3	Yes	Setting up a different cluster is possible, but by separating lanes with dedicated nodes is possible. Questionalble is if this is desired
IS4	Yes	Security (like Two-Way SSL) is enabled and enforced by default
IS5	Sortof	Older standards can be manually installed into containers, but this is not desired
IS6	Yes	Docker Swarm does not limit in terms of nodes per cluster. Performance can decrease.
P1	?	*Not possible to determine at this point*
P2	?	*Not possible to determine at this point*
P3	Yes	Many solutions exist like Prometheus
O1	Yes	Any cloud platform can run Docker
O2	No	Autoscaling does not exist in Docker. Additional efford is required.
R1	No	Since no multi-tenancy exist, all customers can use all resources
R2	Yes	Docker Swarm is really intuitive and low-effort
R3	Yes	Docker YAML Ain't Markup Language (YAML) files can change the cluster (Except R4)
R4	No	Users do not exist in Docker

Table 4.4: Fulfillment of requirements of Docker Swarm

Since its initial release in 2018, its adoption hasn't really grown and does not involve a large community backing its progress. For example, only 57 questions are tagged with `Skipper` and 27 with `spring-cloud-skipper` on StackOverflow.

This also shows in the low amount of documentation and example present in sources. Mainly the websites of spring.io[66] contain limited documentation. For example, the process of deploying applications using Skipper is not documented in how it mutates the cloud environment.

4.3.1 Spring Cloud Skipper Requirements

Skipper itself is a Spring Boot application. It requires Java and some tool to store the state. The list of available databases is provided here[67]. Next to that, a working environment is needed, like Cloud Foundry or Kubernetes.

- Java $\geq$ 8 installed
- Connection to a running environment (like CF or Kubernetes)
- Some database to store the state

4.3.2 Cost

The cost of Skipper itself is free since the entire solution is open source and requires no licensing, except the hosting cost of Skipper. The cost of hosting Skipper is difficult to estimate since Skipper does not mention system requirements. However, since Skipper only deploys Spring applications to other environments, we can assume the service is minimal. Initial testing shows usage of less than 500MB memory. However, the environment connected to Skipper is not free. For these costs, we refer to the sections already described (Section 4.1.3).

4.3.3 Activity of Skipper

Little documentation is found of Skipper, and thus the activity of Skipper is researched. Scholar yielded two relevant results [68] [69] since 2020 (one year after initial release) with the key "Spring Cloud Skipper". IEEE returned no results, similar to Scopus. More unscientific sources returned very little results as well. Youtube returned one video from 2019 about Skipper, and searches for conferences, lectures, or presentations about Skipper yielded no result.

4.3.4 Conclusion

Spring Cloud Skipper does not have the active community or documentation the author is looking for. Even though not explicitly mentioned in the requirements, the solution is meant to be durable for many years to come. Skipper had some activity in 2019, but lost all its momentum and does therefore not satisfy the requirement to be a valid solution for the problem. Although it is part of a bigger application stack called Spring Cloud Data Flow [70], the problem Data Flow solves is too broad and does not focus on the scope of this research.

4.4 Apache Mesos

The paper of Truyen [71] mentions Apache Mesos and two additions to the core product, Marathon and Aurora. Apache Mesos abstracts the individual resources of nodes to a single pool of resources. Marathon was designed to provide endpoints for starting, stopping and scaling jobs using Mesos. Aurora was made to allow for running long-living jobs and Command Run On (CRON) jobs. Both are interesting when looking at the capabilities they offer as described by Truyen. However, Aurora has been discontinued, and Marathon has been practically dead ever since the open-source Mesos has been integrated into Distributed Cloud Operating System (DC/OS). The open-source variant is not maintained anymore.

Unfortunately, DC/OS has since then also announced to drop support and will no longer be maintaining the solution.

4.5 Comparison

From the literature, few Container Orchestration (CO) solutions were discussed. Furthermore, many of these smaller solutions seem to have disappeared under the pressure of Docker and Kubernetes. Kubernetes and Docker both fulfill some of the requirements that were set, but not all under all conditions. The fulfillment of the requirements has been discussed for Kubernetes and Docker Swarm in subsections 4.1.2 and 4.2.4 respectively. In table 4.5 these are shown side to side. In this comparison, we see that Kubernetes fulfills the requirements best, while Docker Swarm is unable to fulfill at least five of the requirements. From this, we decide to use Kubernetes as technology to facilitate the solution design.

Req	Kubernetes	Docker Swarm
IS1	√	×
IS2	√	×
IS3	√	√
IS4	√	√
IS5	√	≈
IS6	√	√
P1		
P2		
P3	√	√
O1	≈	√
O2	√	×
R1	√	×
R2	≈	√
R3	√	√
R4	√	×
Total	11 √, 0 ×, 2 ≈	7 √, 5 ×, 1 ≈

Table 4.5: Fulfillment of requirements of different treatments

Chapter 5

Solution Design

In this chapter, a model of the baseline architecture and processes will be described. The models are discussed in Section 5.1. Next, in Section 5.2, all previously gained information (like requirements, available techniques, and baseline architecture and processes) is combined to derive a reference architecture to solve the problem at hand. Then, a gap analysis is performed to determine what steps need to be taken to move from the baseline architecture to the target architecture in Section 5.3. Additionally, instructions will be given to achieve the architecture in the three major cloud hosts, Google Cloud, Microsoft Azure, and AWS. These setups will also be discussed in Section 5.5. This chapter uses the stardards as described by the ArchiMate® 3.1 Specification [72] as framework for how to design the models.

5.1 Baseline Architecture

In this section, the baseline reference architecture for an iPaaS will be derived. Only parts relevant to the research question will be discussed.

For this section, it is assumed that all cloud hosts offer the same functionality, which is later verified in Chapter 6. Additionally, we assume all iPaaS providers have their solutions currently hosted at either Azure, AWS, or Google Cloud.

Firstly, the business processes for customers are discussed. Next, the cloud architecture is discussed. This means, when an integration is designed, it should be receiving, transforming, and redirecting messages (or relevant actions). These integrations can both be run on-premise or in the cloud. For now, we omit the possibility of on-premise solutions, since these can be very different depending on the requirements, while most cloud providers offer similar services, making the architecture more generic. Lastly, the deployment process is described. An overview of the entire baseline architecture will also be given.

5.1.1 Customer's business processes

For customers of the iPaaS solution need for a new integration might arise. This can have various reasons, explained in subsection 1.2.2. When this happens, a developer is required to design the integration. For this, the developer defines the system requirement of the systems which need to be integrated. Based on these requirements, the requirements for the integration are defined. The developer designs the integration in the iPaaS platform conform the requirements. To ensure the integration works as required, tests (unit-, regression-, load tests, and more) can be defined. When these tests succeed, the integration can be deployed. the iPaaS platform 'activates' the integration (see subsection 5.1.3). From here, the integration is active and needs to be monitored to ensure the performance is as expected. The developer could define alerts to act on unexpected events, like increased message load or degraded

integration performance. With traditional deployments as described in subsection 1.2.2, acting on an increased message load is difficult, since scaling is not automated or not possible at all. An overview of the business process is available in Figure 5.1

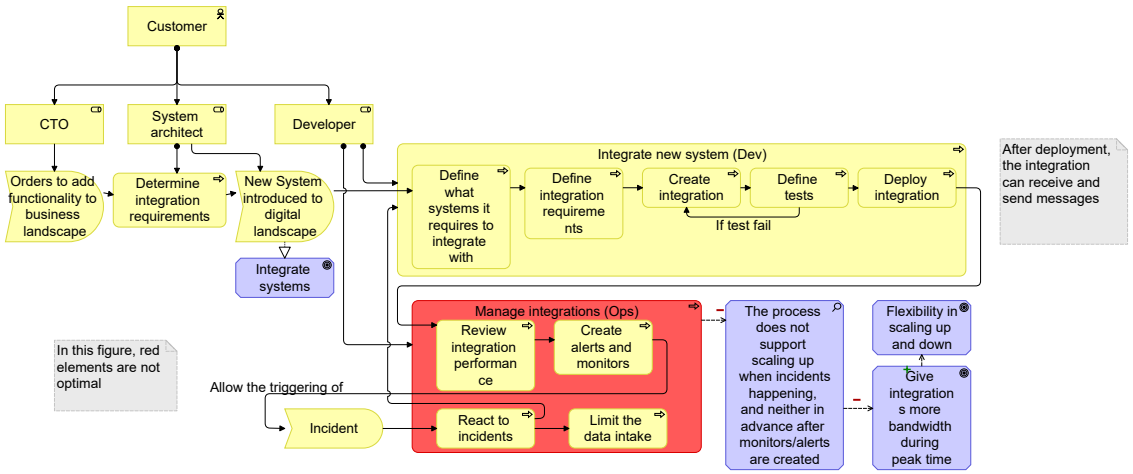


Figure 5.1: Business process of integrating a system of an iPaaS customer

5.1.2 Baseline Cloud Architecture

When a deployment is activated, it is deployed on the cloud. The cloud should have a certain architecture to serve the integration. For customers, this is a black-box principle; they do not need to know how the cloud is designed. However, for this project, it is pertinent. Starting from the entry of the cloud, there are different options for how such cloud architecture can look, depending on the requirements. On the edge of the cloud, some endpoint should be exposed to receive data on. This can be either a Fully Qualified Domain Name (FQDN) or an IP address. Either a single endpoint or multiple endpoints (with or without DNS) can be available. For example, a solution could have one endpoint `receive.iops.com`, and reroute traffic based on the content of the message. On the other hand, a solution could have many endpoints, e.g. `customer1.iops.com` with DNS routing. Both options are viable and depend on requirements like isolation and security.

These entry points are most likely backed by a load-balancer. This load-balancer makes sure the entry point redirects traffic to one or, more desirable, multiple processors. This ensures all traffic is not loaded onto one single processor, but split into multiple processors, to allow for more processing power. Once again, depending on the requirements, these load balancers could be chained. This is useful when the solution is hosted in multiple cloud zones (or regions). The first LB would redirect the message to the right zone, while the second LB would redirect the message to the right processing instance.

As for security, it is possible the LB also needs to perform some actions here. When the message has hit the LB, the message is in a 'secure' environment, meaning the message cannot be altered, and the routing can be fully managed by the iPaaS provider, some security measures can be lifted here. One such example is SSL Termination. When a user visits a website, it's usually served over Hypertext Transfer Protocol Secure (HTTPS), meaning the data is encrypted over SSL. Since the message is received, the authentication part of the message can be dropped. This is called SSL Termination. However, depending on the application behind the LB and security policies, it could be the SSL connection needs to be maintained, and thus that the LB needs to keep the SSL connection. This is called SSL

Pass-through, see Figure 5.2. SSL offloading introduces risks, (e.g. man-in-the-middle-attack), and should be used with care. One can re-encrypt the message again (called SSL bridging), but this introduces overhead due to encrypting and decrypting the message.

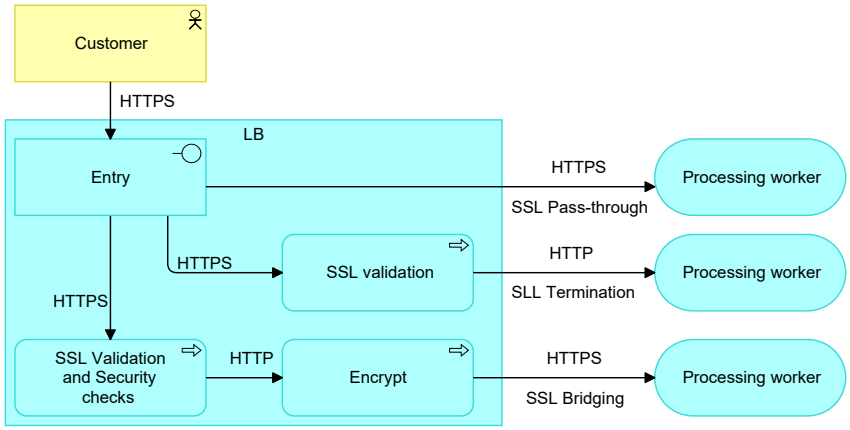


Figure 5.2: SSL Termination, Pass-through or Bridging in LBs, simplified

From here, the message can reach the right destination. To ensure isolation, customers are often placed within their VPC. This ensures the isolation can be configured.

For iPaaS providers, providing logs and metrics to be available to customers allows customers to evaluate their integrations and change them if needed. These logs and metrics are collected from containers or runtimes and saved to a central storage site. Examples of stacks which allow *metrics* to be stored and evaluated are Datadog[73], Prometheus[74] (for scraping) plus Grafana[75] (for displaying), or ElasticSearch (for storing) and Kibana (for displaying) from the Elastic stack[76]. Many more examples of solution stacks exist apart from these three. For logs, Fluentbit[77], LogPoint[78] and Apache's logging libraries[79] are common choices. These metrics and logs are stored on some persistent storage, like Amazon S3, or Azure NetApp Files. A general view can be found in image 5.3

5.1.3 Deployment process

In the last two subsections, the generic cloud setup and the business process is defined regarding deploying and running the integration. Figure 5.1 describes a process `Deploy Integration`. For customers, this could simply be a button, but in the software layer, processes are running to deploy the configured integration to the cloud. When the customer hits the deploy button, the configuration as defined in the platform is saved and packed with the required dependencies

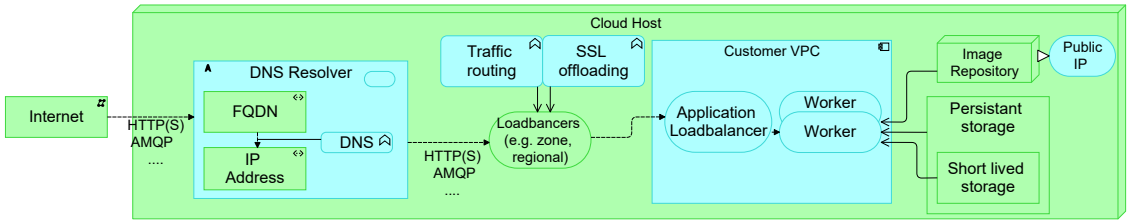


Figure 5.3: Message Flow within the cloud host

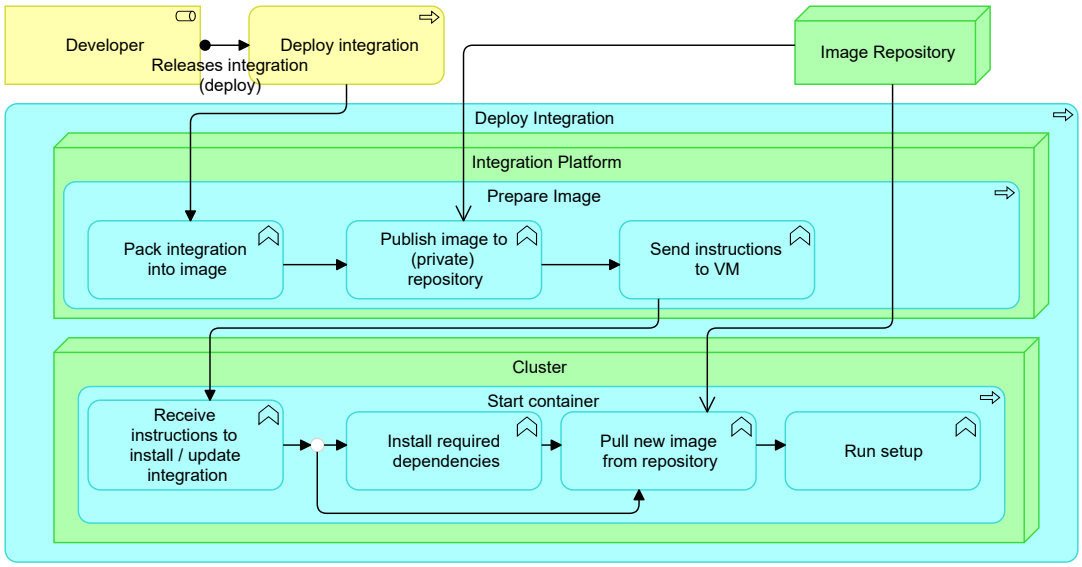


Figure 5.4: Overview of the deploy process without autoscaling

(if any) into an image. Additionally, a configuration or start script is generated. This script defines what the VM needs to do to run the image. The image is pushed to a (private) repository, so it can be pulled from a central place to multiple destinations. The configuration is uploaded to the VM. This configuration can have many different forms, depending on the container runtime on the VM. This configuration defines where to download the image and the steps required to run the image. With this configuration, the VM can start running as expected. During this process, messages (like the command to upload the configuration), should be running through the infrastructure according to Figure 5.3. The deployment process is available in Figure 5.4

5.1.4 Overview Deployment Process

Combining the views from previous subsections, an overview can be made of the entire process of deploying an application by the customer. An overview can be found in Figure 5.5. This model is an aggregation of the previous models, where the top part (business layer) describes the business process, the middle part describes the deployment process, and the bottom part describes the cloud architecture.

5.1.5 Summary

To solve the problems at hand, a new architecture needs to be designed. This architecture needs to fulfill the requirements in chapter 3, but also not change the current behavior. The exception to this is the addition of the auto-scale functionality. To summarize:

- A DNS resolver should be available, which can redirect to a specific VPC or equivalent of it.
- LBs should be in place both inside the customer VPC, and in front of customer VPCs.
- The LB in front of customer VPCs should be able to reroute traffic and to handle SSL offloading.

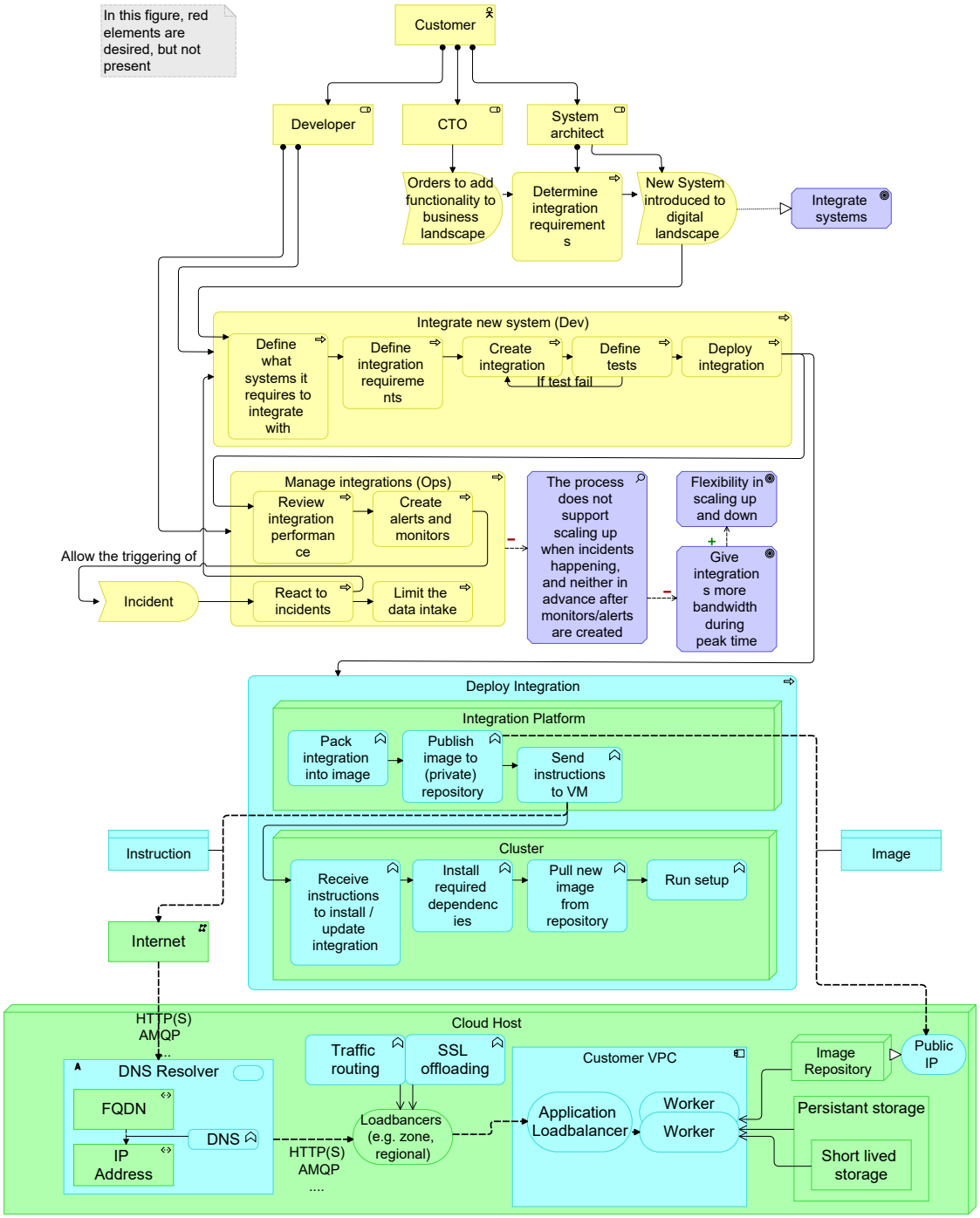


Figure 5.5: Overview of the deploy process with autoscaling

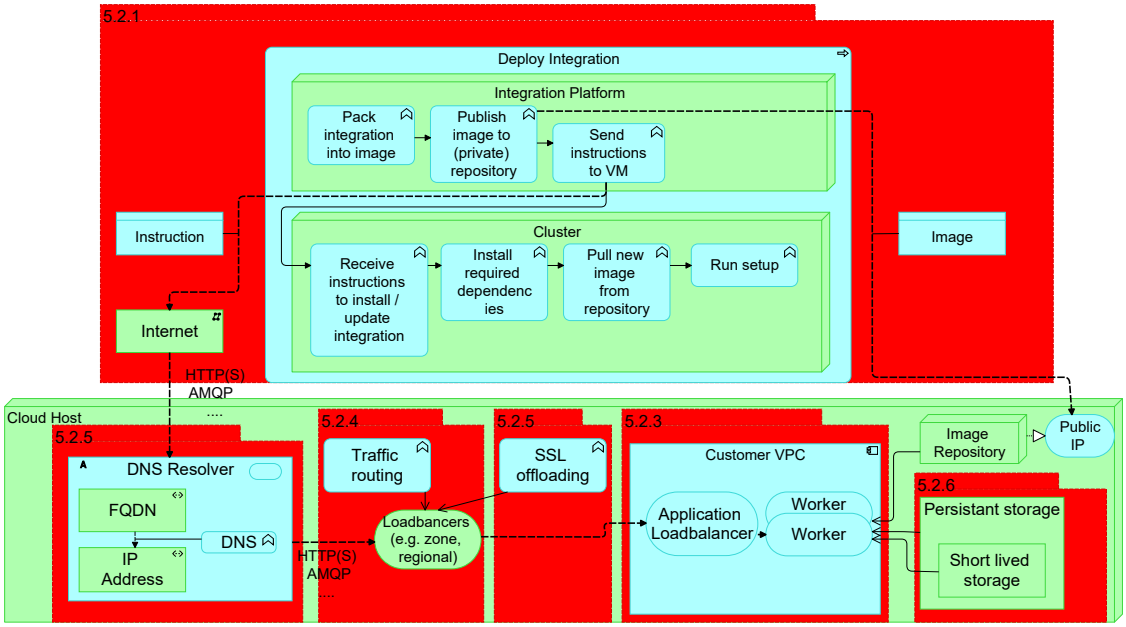


Figure 5.6: Mapping of the baseline architecture to the chapters of Section 5.2

5.2 Target Architecture

In the previous sections, the baseline architecture and problems are described. This section will describe how the target architecture should be to comply with the requirements defined in Chapter 3. The acquired knowledge from Chapter 4 is used for this. This section is divided into several subsections, each tackling a different part of the solution, to segregate the solution into smaller parts. These parts include:

- **Deployment Process**: The desired way to communicate with the cluster.
- **Master Plane**: The configuration point of the Kubernetes cluster, with cluster-wide configuration
- **Worker Plane**: The setup and configuration of workload processors
- **Exposing Interface**: The routing from the public IP to the right pods
- **DNS and TLS**: Securing and automating access for HTTPS.
- **Metrics, Logs and Alerts**: Observability of the cluster

In order to show how aspects of the target architecture are mapped onto the baseline architecture, a overview is given in Figure 5.6.

5.2.1 Deployment Process

This subsection will describe how the cluster can be managed programmatically. For this, the deployment process needs to be adapted. This section will discuss how Figure 5.5 can be changed to be able to communicate with the Kubernetes Cluster API to make changes to the cluster.

Firstly, it is required to describe how Kubernetes determines what actions to perform. Kubernetes keeps track of the current state of the cluster. When a developer sends a new update to the cluster (e.g. scale up a deployment), Kubernetes will compare the current state and the desired state of the cluster. If these values do not match, Kubernetes will determine the steps required to meet the desired state, and execute those. This means, that if

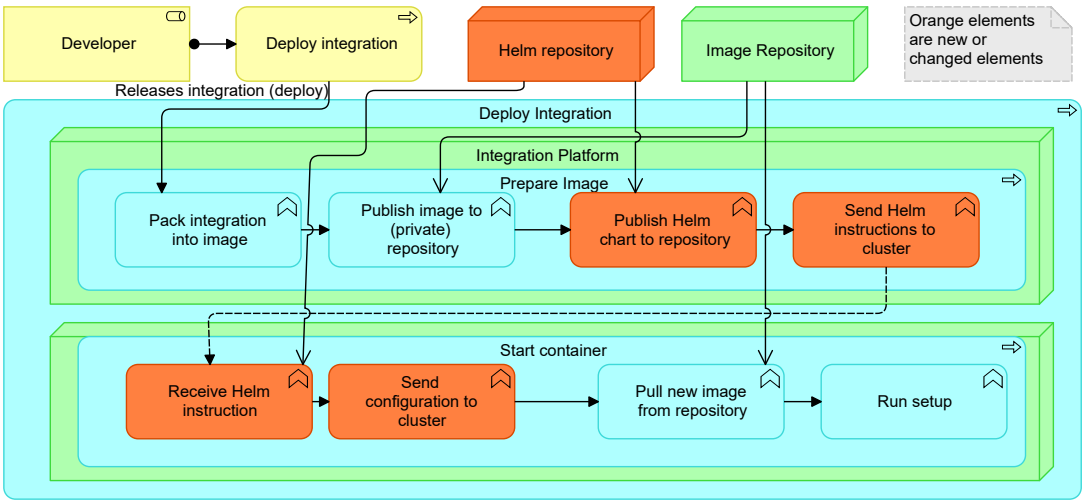


Figure 5.7: New deployment process using Helm

a configuration is applied twice, Kubernetes will not perform any action, since the desired state is equal to the current state.

When an integration is defined in the iPaaS portal, and the integration is released, similar to the current situation, the integration is packed into an image and sent to the (private) repository. However, instead of sending instructions to the VM, a helm[80] instruction is sent to the cluster. Helm can be seen as a package manager for Kubernetes, similar to apt or yum on Linux distributions. Using helm charts, developers can define templates for a deployment. This template will be filled by a file containing variables. Kubernetes can then download the image and its required dependencies, and configure its architecture as defined in the chart. Other than Helm, there are no other viable package managers for Kubernetes. The adjusted deployment process can be found in figure 5.7. This step ensured that communication with the API server was possible and the use of helm charts was available. This contributes to requirement M1 to ensure the deployment process takes (nearly) no time.

5.2.2 Master Plane

In this subsection, the master plane is discussed. In the baseline architecture, no such component is present. The master plane is needed for the management of the cluster and the enable the ability of container orchestration. In the cloud, the master plane mainly consists of settings (like zonal or regional clusters, security principles, and scaling profiles).

Within all three cloud vendors, the master plane of the cluster is managed by the cloud host. Most cloud hosts have two at least two flavors of clusters. For GKE, this is a normal cluster or an autopilot cluster. In the normal cluster, the nodes need to be maintained, while the autopilot configures nodes for you. This autopilot cluster's pricing model is based on the pods and their usage, while a normal cluster's pricing model is based on the number of clusters. Azure offers two flavors as well, the normal cluster and Azure Arc. Azure Arc lets you connect other services to the Azure stack. However, this requires more maintenance, and Microsoft does not offer guarantees about this. Taking portability into mind, the generic cluster is the best option.

The master plane is the controlling instance in the cluster. Here, some settings are required to enable or disable. Most cloud hosts offer similar settings, but some options might be

different. For example, GKE has an option to enable or disable *network policies*, while other hosts enable this by default.

Many settings are up to iPaaS providers to decide, depending on their specific business goals and SLAs. However, some settings should be considered for all iPaaS providers.

- **IP Range Autorization**: The API server of the cluster should be heavily protected. Anyone having access to here could be a serious breach. Therefore, allowing access to the API server from only a specific (range of) IP(s) should be considered.
- **Maintenance window**: Enabling automatic upgrades improves security. However, this also disrupts services. When the master plane upgrades, no scaling can occur. To prevent this from happening during primary times, a maintenance window should be set.
- **Autoscaling profile**: When pods are running on nodes, sometimes moving pods from one node to another might allow for deletion of one node. However, this can cause disruption or the dropping of connections. Therefore, a business choice should be made whether this is desired or no disruption is desired.
- **Application layer secret encryption**: On all hosts, storage layer encryption is enabled. Another layer of security, application layer secret encryption, can increase the security of secrets. However, this requires maintenance of the encryption key, increasing the maintenance effort required.

Based on the general need, it is recommended to allow only IP ranges used by the cluster administrators to be allowed or to set up a jumper pod. A jumper pod is a pod that developers can access, allowing administrators to enable only traffic from within the cluster to be allowed. For the Maintenance window, it is recommended to set to a time where the least scaling occurs, and no additional workloads need to be scheduled. The autoscaling profile is up for the business to decide, but given the objectives of this research, an economic setting will be used. As for the application layer secret encryption, it is advised to enable this when enough maintenance effort can be assigned to it.

This step ensured that a cluster exists (and possibly more can be created for other TAP-environments). This ensured the completion of requirements IS3 and enabled options to solve other requirements.

5.2.3 Worker Plane

This subsection discusses how worker planes are set up in the target architecture. This can be compared to the Customer VPC (multiple of them) of Figure 5.3.

The worker plane consists of node pools and nodes. Node pools contain a collection of nodes with certain specifications, like size and availability zone. Node groups can be configured to scale up depending on the pressure on the node pool, and automatically start or tear down nodes. When nodes are available, Kubernetes will automatically schedule pods and provide required resources (like PVs) to run workloads.

An overview is included in Figure 5.8. This image will be explained more in the next Subsection.

This step enabled setting up automatic scaling NodeGroups, contributing to O2. This also contributes to M1, since no manual scaling is needed after this step. Lastly, the ability to enable automatic healing was achieved. Automatic healing contributes to DR1 and DR2.

5.2.4 Exposing Interface

In this subsection, principles from the baseline architecture are translated to Kubernetes principles regarding the internet-facing load balancer. This can be mapped to the DNS

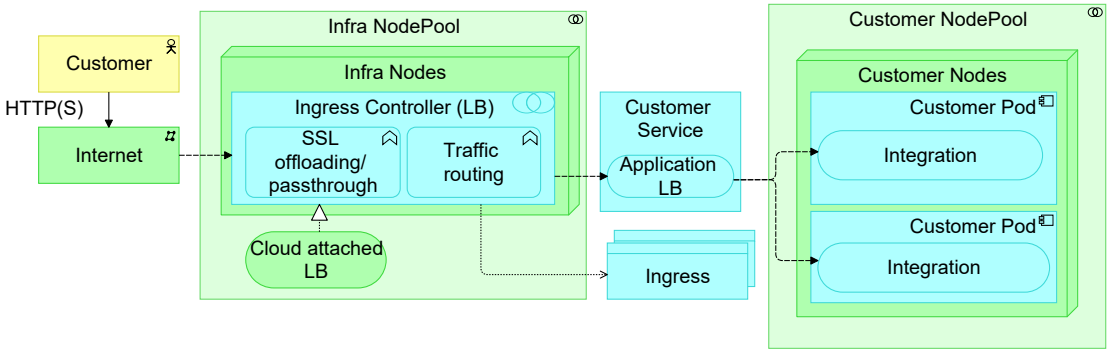


Figure 5.8: Ingress controller as deployed in Kubernetes

Resolver and the first LoadBalancer in Figure 5.3.

Firstly, the internal DNS resolver principle in the baseline architecture can be exchanged for an *Ingress Controller* in Kubernetes. *Ingress Controllers* forward traffic according to the rules defined in the *Ingresses*. Kubernetes does not use a default *Ingress Controller*, but allows for importing one of many available controllers, as listed here[81]. *Ingresses* allow setting forwarding rules based on host, path, or port to specific services. Depending on the chosen plugin, it could handle actions like SSL Offloading and Traffic routing. If we attach a *Service* to the *Ingress Controller*, we can define it as a LoadBalancing service. When this is defined, the cloud host will attach a Front Facing LB. This allows external access which can be HA (if the amount of *Ingress Controller* pods in the replication is greater than one), and with the forwarding rules as specified by the *Ingresses*. An overview can be found in Figure 5.8

This step contributes to IS1, because it ensured routing is enabled and authentication on the routes can be applied. Furthermore, this step allowed to setup two-way SSL as defined in IS4, and older standards can be allowed as per IS5.

5.2.5 DNS and TLS

In the previous Section, the forwarding strategy for internal traffic was discussed. In this Subsection, the method of generation and validating the TLS certificates is discussed, equal to the Traffic Routing and SSL Offloading functionality of Figure 5.3

For traffic to reach the cluster, an external DNS needs to be configured. This allows nameservers to forward traffic from anywhere to reach the cluster. One option is to manually set the DNS records according to needs. However, this requires additional effort, and is more prone to errors, when IP addresses change. Instead, this section introduces a plugin called external-dns[82]. External-DNS updates DNS records exposed by the cloud provider based on the rules defined in the *Ingresses*. External-DNS is available for all major cloud vendors. External-DNS will create Text records (TXT)-records to keep track of the DNS management. Furthermore, it creates A-records pointing to the IngressController. This path is not secured yet and will give warnings when accessed. For this, Cert-Manager can help. Cert-Manager can, based on Ingress rules, solve certificate challenges (e.g. from LetsEncrypt) and install acquired certificates automatically. This ensures websites are encrypted and secure automatically based on ingress rules. One thing to note is that there are rate limits in place, which should be taken into account. An overview can be found in figure 5.9

This step contributes to M1 and M2, since deploying a workload now automatically updates the DNS records and generates TLS certificates, lowering the amount of manual labour needed. This step also increased security by automatically providing encryption over the traffic.

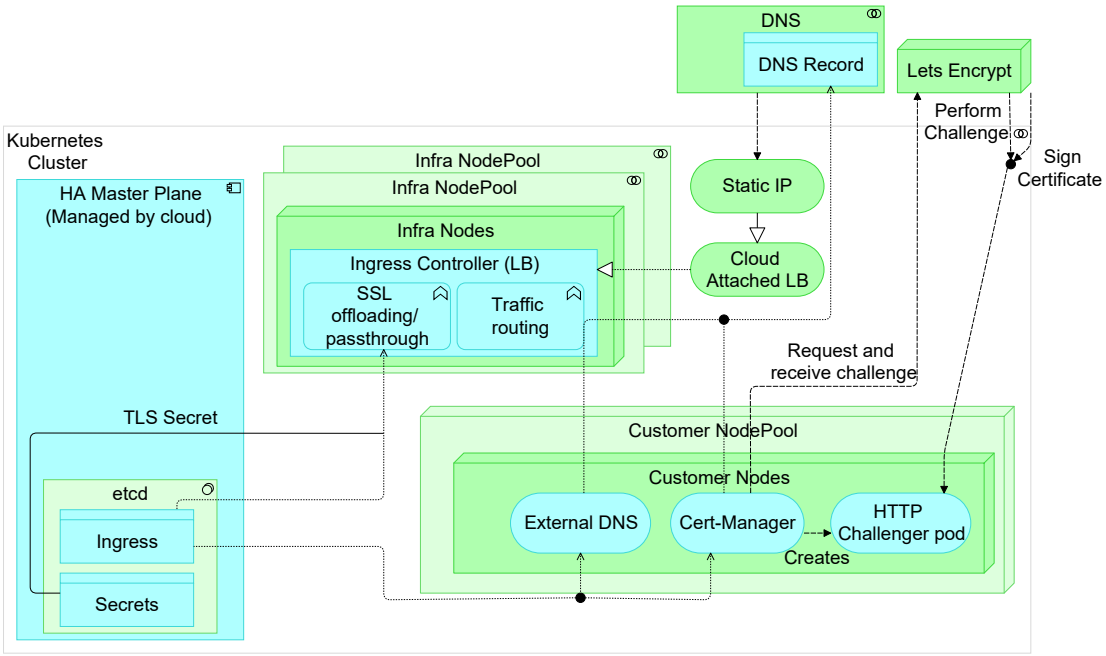


Figure 5.9: DNS auto update and TLS auto challenge

5.2.6 Metrics, Logs and Alerts

Observability is important when hosting cloud solutions to be able to react to events quickly or to get insight into the performance of the solutions. Important aspects to achieve observability are Logs, Metrics and Alerting, Tracing, and Visualization [83]. As mentioned in 5.1.2, quite a lot of stacks exist for this. However, some are more preferred than others. Since Kubernetes exposes metrics over an HTTP endpoint, a pulling (or scraping) solution to get these metrics would fit best with Kubernetes architecture. When we consider the budget as defined in requirements C1, C2 and C3, we prefer to use free, open-source solutions. Prometheus [74] is perfect for this. It scrapes metrics from HTTP endpoints defined in its configuration and stores it as a time series. However, Prometheus stores all data scraped, causing the storage to quickly fill. Therefore, another storage should be present to store the aggregated and more finely chosen metrics for long-term storage. Doing this also enables to set the retention period of Prometheus to the desired value, so the storage used by Prometheus does not grow substantially. Since Prometheus metrics are time series, the underlying storage should be able to handle time series. However, which solution is used does not make a big difference. Therefore, storage solutions should be similar to the solution currently adopted. However, one requirement could be that there should be an exporter from Prometheus to the storage solution (see [84]). As pods can expose metrics over HTTP, node- and cluster metrics should also be available to ensure full observability over the cluster. For nodes, the Prometheus Node exporter [85] could be used, to expose metrics about individual nodes and scrape them with Prometheus. As for cluster metrics, Kube State Metrics [86] exposes these metrics. For logs, visualization and traceability, one should again consider where the data is written to. This is the main influence for what solution should be chosen and depends on the current application stack. An overview can be found in figure 5.10

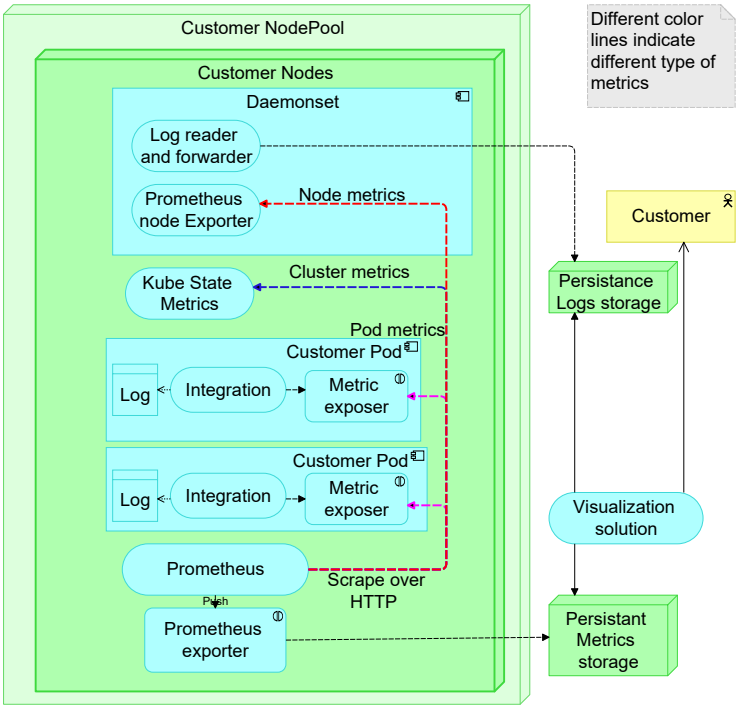


Figure 5.10: Overview of architecture to collect and display logs and metrics

5.2.7 Overview Target Architecture

Combining all models from the previous subsection, a general architecture can be derived. This can be found in figure 5.11. This figure is a combination of all models mentioned in previous subsections. Since all aspects have been discussed before, this image is meant to provide an overview of the solution architecture.

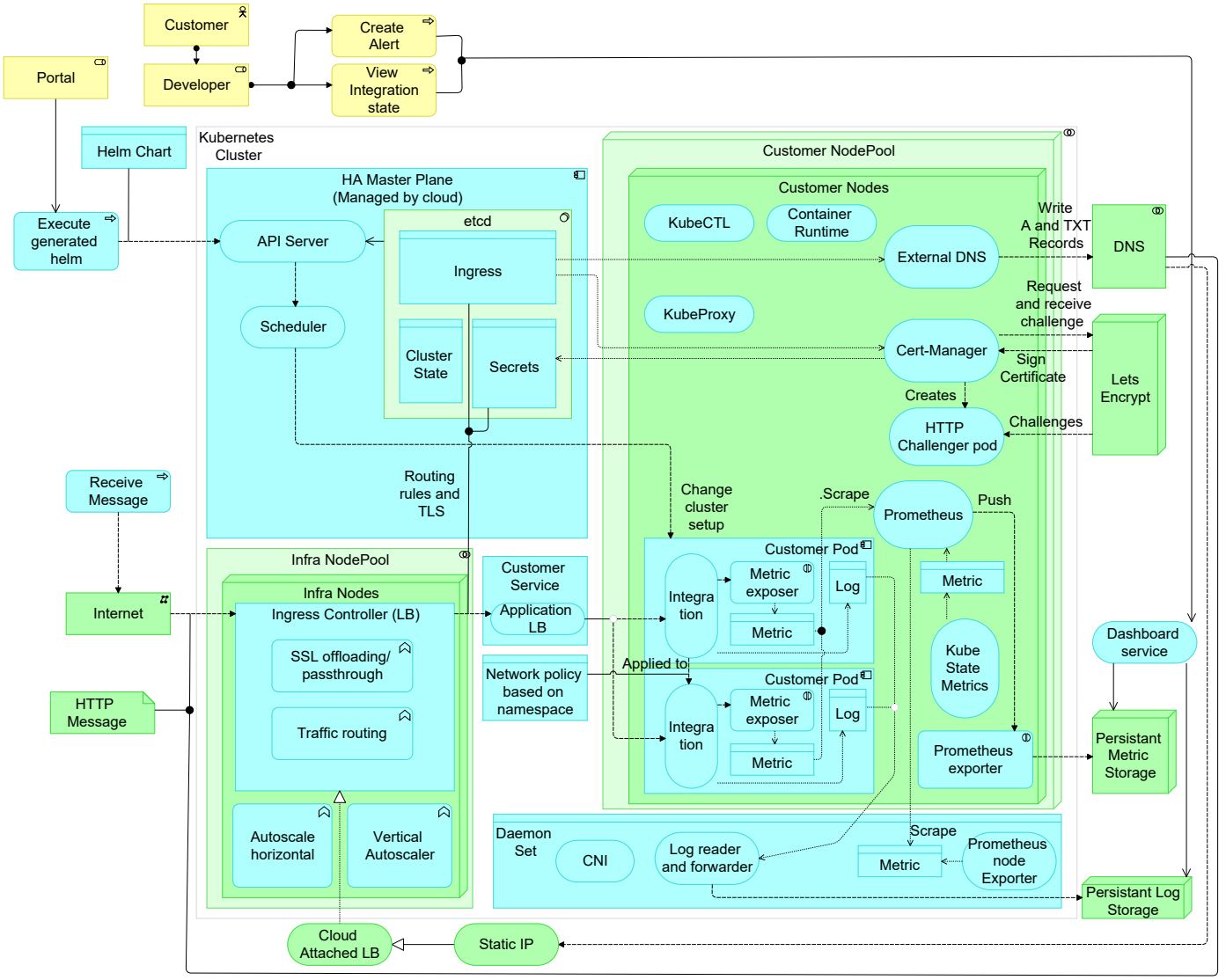


Figure 5.11: Overview of the target architecture

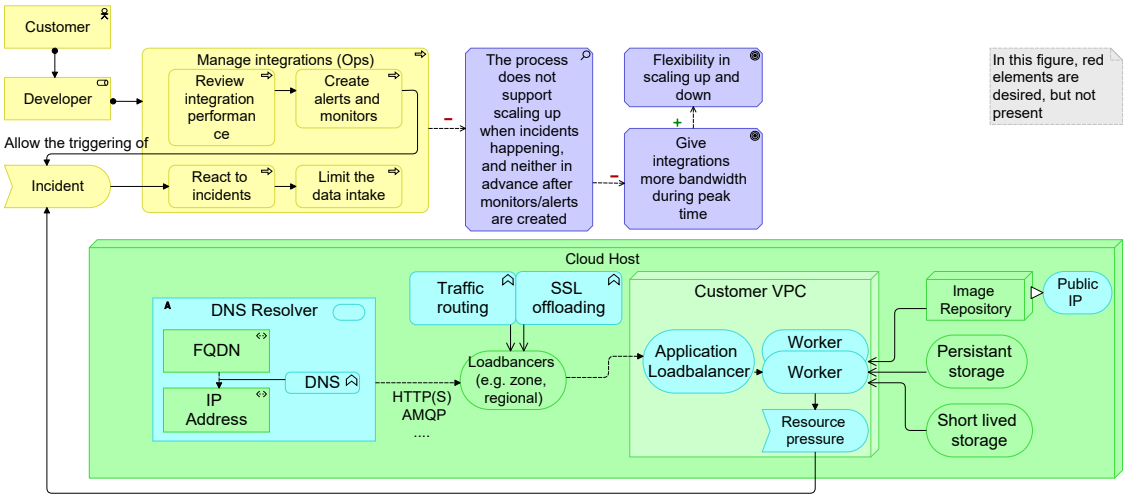


Figure 5.12: Situation without autoscaling

5.3 Gap Analysis

In this section, the baseline architecture and target architecture are analyzed to determine what steps are required to achieve the architecture that satisfies the objective of the research. An overview is given in figure 5.14.

5.3.1 Incident due to high load

Depending on the business processes, there could be a spike in required resources. This can happen due to scheduled processes, like data replication, nightly processing (e.g. route scheduling), or other events. During this time, it is possible the existing resources are not sufficient, and resource pressure occurs. This can cause all kinds of problems, like loss of data due to cleanup policies, timeouts on connections causing loss of data, and many more of these types of events. This could be problematic for a customer and thus must be avoided. From comparing Figures 5.12 and Figures 5.13, it is clear that due to the lack of the option to scale up processors, only limiting the data intake is possible. Instead, it would be better to have some auto-scaling features as illustrated in Figure 5.13. This feature can start new containers when pressure is high or tear down containers when pressure is low.

5.3.2 Requires steps to perform

In the previous Subsection, the need for processor scaling was described. This is the main gap that needs to be closed. To do so, intermediate steps are required.

- Firstly, there is a need to be able to communicate with the cluster. In the baseline architecture, the ability exists to create images and upload them to VMs, and to control the VMs. However, new integrations need to be developed. Communication with the Cluster API needs to be created, to control the cluster. Additionally, helm operations need to be configured. Helm is used to upload workloads to the cluster. This step is

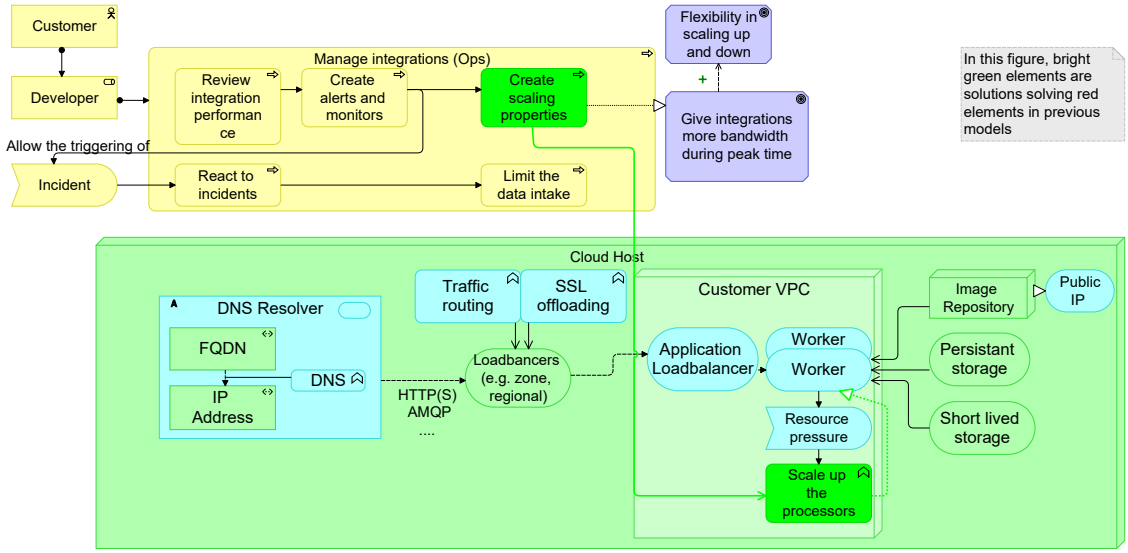


Figure 5.13: Situation with autoscaling

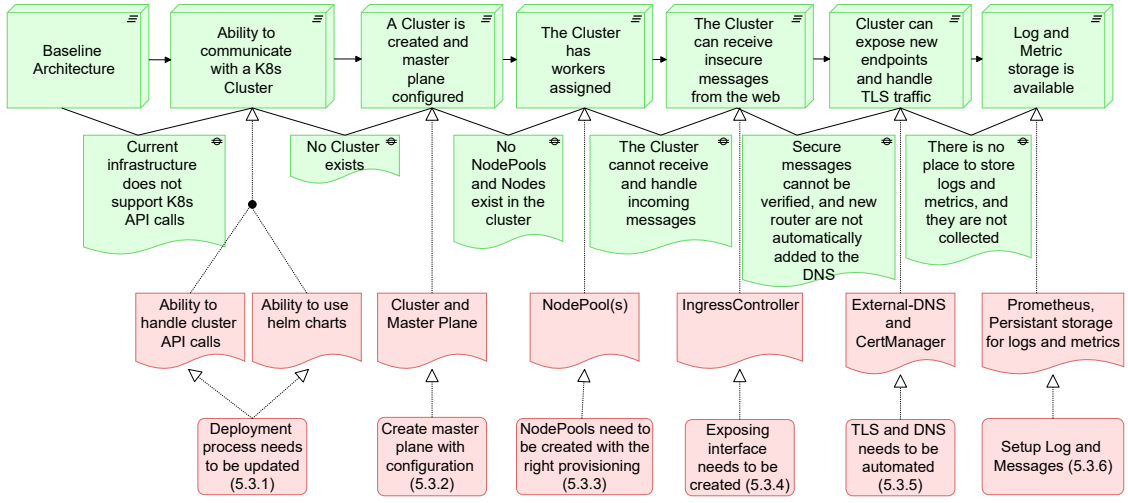


Figure 5.14: Overview of the gap analysis

important for the iPaaS provider to manage the workloads. This was elaborated in Subsection 5.2.1. This contributes directly to requirement M1.

- Secondly, a cluster needs to be created and configured. This ensures a cluster exists to perform the next deployment steps. This was elaborated in Subsection 5.2.2. This contributes directly to requirement IS3.
- Thirdly, worker NodePools need to be configured. NodePools spin up Nodes, on which customer integrations run. This was elaborated in Subsection 5.2.3. This contributes directly to requirements O2, DR1, DR2, and M1.
- Fourthly, a mechanism to perform routing needs to be installed. This mechanism will also allow receiving messages. This was elaborated in Subsection 5.2.4. This contributes directly to requirements IS1, IS4, and IS5.
- Fifthly, the routing endpoints need to be published to the DNS server to allow routing based on URLs. This also allows to automatically generated certificates and apply them to endpoints. This ensures the connection is secured. This was elaborated in Subsection 5.2.5. This contributes directly to requirements O1, M1, and M2.
- Lastly, infrastructure for generating, scraping, and storing logs, metrics and alerts need to be configured. After this step, the cluster is operational, and workloads can be deployed to the cluster. This was elaborated in Subsection 5.2.6. This contributes to no requirement directly but ensures the target architecture is capable of the same functionality as the baseline architecture.

5.4 Installation instructions

To replicate the prototype setup as defined in the previous section, the steps as defined in the gap analysis are required. This section will describe how the total solution can be configured by giving intallation instructions for the target architecture. A basic setup is given, and customizations are up to the reader to implement. To change the configurations, the values file of the helm chart can be used. The specifics can be found in the repository of each component. For this example, fluent-bit will be used as a log processor, ElasticSearch as log storage, Kibana as log dashboard, Grafana as metric dashboard, Prometheus as a metric scraper, and Nginx-ingress as IngressController.

1. Deploy a cluster with the cloud provider where you want the cluster to be run. Specific documentation for GKE[87], AKS[88] and EKS[89] are available.
2. Access the cluster via the CLI of the cloud host
3. Add the required helm repositories:

```
helm repo add nginx-stable https://helm.nginx.com/stable &&
helm repo add prometheus-community
↪  https://prometheus-community.github.io/helm-charts &&
helm repo add jetstack https://charts.jetstack.io &&
helm repo add fluent https://fluent.github.io/helm-charts &&
helm repo add external-dns https://kubernetes-sigs.github.io/external-dns/ &&
helm repo add elastic https://helm.elastic.co &&
helm repo update
```

4. Deploy the Nginx Ingress Controller via helm by executing:

```
helm upgrade -f nginx-values.yaml --install ingress-nginx ingress-nginx \
--repo https://kubernetes.github.io/ingress-nginx \
--namespace ingress-nginx --create-namespace
```

   Note here that the values.yaml is from the official repository. Only values about metrics scraping are edited.

5. Install prometheus. This also installs node-exporter and kube-state-metrics.

   ```
   helm upgrade -f prometheus-values.yaml --install prometheus
   ↪  prometheus-community/prometheus --namespace metrics --create-namespace
   ```

   Note here that the values.yaml is from the official repository. Values that were updated were to disable alertmanager and pushgateway, and changes Ingress settings.
6. Install Fluent-Bit

   ```
   helm upgrade -f fluentbit-values.yaml --install fluent-bit fluent/fluent-bit
   ↪  --namespace fluentbit --create-namespace
   ```

   Note here that the values.yaml is from the official repository. The config section needs heavy tweaking based on the requirements, including the elastic endpoint of step 12.
7. Install Cert-manager

   ```
   helm upgrade --install cert-manager jetstack/cert-manager --namespace
   ↪  cert-manager --create-namespace --set installCRDs=true
   ```

   Note here that the values.yaml is from the official repository. Custom Resource Definition (CRD)s automatic installation was enabled.
8. Install a cluster issuer. The following yaml file was applied. For this situation, nginx was used to perform a HTTP challenge from LetsEncrypt. More options exist. Note that this uses the Acceptance environment, and should be changed to a production environment.

   ```
   cat <<EOF | kubectl apply -f -
   apiVersion: cert-manager.io/v1
   kind: ClusterIssuer
   metadata:
     name: letsencrypt-acceptance
   spec:
     acme:
       email: EMAIL_OMITTED
       server: https://acme-staging-v02.api.letsencrypt.org/directory
       privateKeySecretRef:
         name: cluster-private-key
       solvers:
       - http01:
           ingress:
             class: nginx
   EOF
   ```

9. Perform the installation steps according to your cloud host provider from

   ```
   https://github.com/kubernetes-sigs/external-dns#deploying-to-a-cluster
   ```

10. Install external-dns

    ```
    helm upgrade -f externaldns-values.yaml --install external-dns
    ↪  external-dns/external-dns --namespace external-dns --create-namespace
    ```

    Note here that the values.yaml is from the official repository. TXT related settings, domain filters, providers and source were updated.
11. Install the ECK operator for Elastic and Kibana:

    ```
    helm upgrade --install elastic-operator elastic/eck-operator -n elastic-system
    ↪  --create-namespace
    ```

12. Deploy instances of ElasticSearch and Kibana

    ```
    cat <<EOF | kubectl apply -f -
    apiVersion: elasticsearch.k8s.elastic.co/v1
    kind: Elasticsearch
    ```

```
metadata:
  name: prototype
  namespace: elastic-system
spec:
  version: 8.0.1
  nodeSets:
  - name: prototype
    count: 1
    config:
      node.store.allow_mmap: false
EOF &&
cat <<EOF | kubectl apply -f -
apiVersion: kibana.k8s.elastic.co/v1
kind: Kibana
metadata:
  name: prototype
  namespace: elastic-system
spec:
  version: 8.0.1
  count: 1
  elasticsearchRef:
    name: prototype
EOF
```

Depending on your setup, it might be required to expose ElasticSearch or Kibana through an Ingress rule.

13. Install the prometheus-elastic-exporter

```
helm upgrade -f elastic-exporter-values.yaml --install
↪  prometheus-elastic-exporter
↪  prometheus-community/prometheus-elasticsearch-exporter -n monitoring
↪  --create-namespace
```

In this values file, properties like the elastic endpoint and configuration should be set.

14. Deploy integration containers by providing helm charts in a repository, and applying them to the cluster, in a similar way to the steps above. Due to the wide variety of setups of the integration containers, we cannot define exactly how these containers should be started. However, a possible example is given in Appendix B. This deployment is based on a stateless container that can be downloaded from an image repository.
15. Deploy Network Policies to all namespaces according to the required security. An example can be found in AppendixA.

5.5 AWS, Azure and Google Cloud

In the previous section, a boilerplate was designed to deploy a Kubernetes cluster according to the requirements defined. In this section, these will be applied to the different cloud providers.

5.5.1 Google Cloud

Using the Web UI of Google Cloud, a cluster was created. This took into account the settings as discussed in Subsection 5.2.2. To reduce cost, the infra node group was omitted and all pods were hosted in a single node group. After running all commands defined in Subsection 5.4, the solution was successfully deployed. After deploying a test image on the cluster and setting the routing in an *Ingress*, the DNS was updated, and a certificate was issued. The test deployment was available after a short while and handled messages as expected. To test

isolation, a second pod was deployed in a different namespace. The network policy as defined in Appendix A was applied to later validate the solution.

One thing to note on Google Cloud is that there is no fast volume type available to perform write-many operations. This was not required by the solution but is something to bring to the attention of the reader. Google Cloud has released File Store[90], but this is based on NAS storage, which might be too slow for certain goals and is relatively expensive.

5.5.2 Microsoft Azure

The same solution was deployed to Microsoft Azure. Given the instruction in Subsection 5.4, the solution was deployed. However, following the instruction of step 9 was not successful. This is due to the method of obtaining an account to validate the solution. This account was provided by the educational institute. This means the account was in the Active Directory (AD) of the institute. One step in the instruction was to create a service account within the AD. This operation was not allowed by the institute. This means that automatically creating DNS entries were not verified. The integration between external-dns and AzureDNS is in Beta phase, which means "*Community supported, well tested, but maintainers have no access to resources to execute integration tests on the real platform and/or are not using it in production*[82]". Therefore, it is safe to assume this would work with the right permissions.

5.5.3 AWS

For AWS, an account was provided by the educational institute. Giving the right permissions to create the desired resources appeared to be quite difficult, and required multiple meetings between the account manager and the researcher. The required settings can be found here[91]. Additionally, AWS strict rules on how the infrastructure of the cloud should be set up for nodes to join the cluster. An example of a valid VPC can be found here[92]. Furthermore, plugins that are by default enabled on Azure and Google Cloud, need to be added manually on AWS, being the CoreDNS (for service discovery), VPC CNI (for pod networking), and kube-proxy (for service networking). Lastly, to manipulate Route53 entries, permissions are required. This can be found in the documentation of external-dns[82] and cert-manager[93]. After this, the cluster could be set up as defined in Subsection 5.4 successfully. In short, AWS requires more setup before cluster creation, and is more strict in its permissions, but can work as the other two clouds.

5.6 Conclusion

In this Chapter, research question **SQ3**, "*Can a reference architecture be derived as solution design, and can this architecture be introduced into the as-a-service deployment landscape of iPaaS solutions?*" was answered. Since the previous two research questions were answered, requirements and possible treatments were available in this chapter. Additionally, this chapter derived a model of the baseline architecture of iPaaS solutions. Using the two findings and the reference architecture, a new architecture was designed. This architecture was made to fulfill the requirements and to optimize the value attributes of portability and reduction of cost. In short, Figure 5.11 provides a reference architecture with the answer to the research question, where it is integrated in the as-a-service deployment landscape of an reference iPaaS architecture.

Chapter 6

Validation

In this chapter, the proposed solution is validated. In the research question, the goal of the research was defined. It should increase portability and flexibility while decreasing cost. Additionally, no functionality should be lost. To validate the solution, the reference architecture is validated with experts to validate the validity of the model. Next, the functional and non-functional requirements are validated by assessing whether the requirements are met based on test implementation, findings of the experts and design of the solution. Then, a case study is discussed, where the target architecture is applied. Validated is whether the old functionality still works. Furthermore, the dynamic and static cost component of the old situation and new situation is discussed. Lastly, the increase of both the portability and flexibility is discussed.

6.1 Validation of Reference Architecture

To validate the correctness of the reference architecture, two experts were consulted to give feedback on the previous versions of the models. They were asked to give feedback about the model itself, and the use whether it is in line with the ArchiMate® 3.1 Specification[72]. They were asked to validate and comment on Figures 5.1 up to and including 5.14. Their comments included: Spelling errors or require rephrasing on Figures 5.1, 5.5, 5.6, 5.7, 5.12, 5.13, and 5.14. The experts indicated missing relations or wrongly used relations in Figures 5.1, 5.4, 5.5, 5.6, 5.7, 5.10, 5.11, and 5.14. Advice on restructuring was given on 5.14. A change of object type (e.g. from technology- to application layer) was suggested for Figures 5.3 and 5.14. These suggestions have been reviewed, and almost all were adopted in the current version of the figures.

About the experts:

Experts consulted in this section were two students who successfully completed the Enterprise Architecture course at the University of Twente. This course aims to:

- To become familiar with the most important Enterprise Architecture (EA) frameworks (e.g. Zachman, ArchiMate, etc.), methodologies (e.g. TOGAF), specification (ArchiMate and other modeling languages), and analysis (qualitative and quantitative) approaches and with their applications in different areas of research and practice (e.g., smart logistics and smart industry, enterprise security and risk management, etc.).
- To be able to formulate a business problem, analyze that problem (using any methods or theories that may have been provided during the BSc or MSc program), translate that problem into an EA change process, and propose a migration strategy from a baseline

EA to a target EA that solves the original problem, and is based on a clear motivation for the design decisions taken.
- To apply hybrid EA-data analytics techniques as a means to support decision problems.

Both successfully finished the course with an excellent grade of a 8.5. Additionally, both have participated in the course Architecture of Information Systems, where the knowledge was again tested, and both passed this as well.

6.2 Functional Requirements

In this section, the functional requirements from Section 3.1 are validated. This is done by building a prototype as defined in section 5.2 and 5.4. Then, depending on the requirement, either the documentation was consulted whether the requirement is fulfilled, a test was set up to validate the requirement or a combination of both. The result can be found in Table 6.1.

Req	Yes/No	Explaination
IS1	Yes	Combining network policies and Role-based access control (RBAC) rules, this is achieved
IS2	Yes	Nodes are automatically scaled up if needed. Using ResourceQuotas, maximum resources can be limited
IS3	Yes	Two clusters were created.
IS4	Yes	Two way SSL is supported, not automatic generation of certs
IS5	Yes	Nginx IngressController allows setting allowed standards [94].
IS6	Yes	This is achieved.
P1	Yes	An estimation was made for static and dynamic components, see Subsection 6.4.2
P2	Depends	This is heavily dependent on iPaaS provider's setup, see Subsection 6.4.2
P3	Yes	These are exported to any sources from Prometheus
O1	Yes	The solution was successfully deployed on AWS, Google Cloud and Azure
O2	Yes	Horizontal Scaling was setup.
R1	Yes	Packet sizes can be increased by using ResourceQuotas
R2	Yes	Day to day activities can be automated using Helm Charts integrated by the portal.
R3	Yes	Using Helm charts, value files can be defined from the portal.
R4	Yes	The iPaaS solution can create service accounts for each customer. Credentials for this don't exist

Table 6.1: Fulfillment of functional requirements of the solutions

To validate **IS1**, a setup was used, where four pods were created. Firstly, an IngressController. Secondly, a tester pod, having the image `k8s.gcr.io/e2e-test-images/jessie-dnsutils:1.3` was used, in namespace `not-prototype`. Next, two simple HTTP pods were deployed in namespace `prototype`. One of these pods was exposed by the IngressController. Except for that setting, both pods were identical in terms of security policy (see Appendix A). Using curl from the pod in the `prototype` namespace only the exposed pod in the `prototype` namespace was reachable. The other pod in the `not-prototype` namespace did not reply to curl commands. This validates the isolation across namespaces. A representation can be found in figure 6.1

In terms of identification of other customers, the `cert-manager` did expose the namespace where the certificate is placed in the TXT record. Therefore, namespaces should be hashed or at least not be relatable to the customer. All containers are run by service accounts that do not have role bindings to list sensitive information, like namespaces, and were set up to have restrictive RBAC-rules.

As for **IS2**, since customers can be limited using ResourceQuotas, pods of customers cannot grow indefinitely. Also, setting limits on pods limits the growth of pods. This means

that using a combination of Limits and ResourceQuotas, pods in a namespace can be limited in how they grow horizontal and vertical. Assuming the node size is sufficient, this will cause no problems on the other customer's pods. This was validated by spinning up more pods than allowed (or requesting more resources than allowed). These pods were not scheduled. On the other hand, infrastructural components (IngressController for example), can scale automatically without limitations, causing them to scale up to what is needed to handle all traffic.

IS3 was executed successfully.

IS4 was proven to work by generating client-side certificates manually. Although `cert-manager` can generate server certificates automatically, generating client certificates (and distributing them) is more difficult. Due to the scope of the requirement, it was sufficient to show client certificates worked manually. This was verified by setting up an *Ingress* rule with client certificate validation on. When the certificate was not sent, it resulted in a 403 (Unauthorized) error. When the certificate was sent, the result was returned successfully.

For **IS5**, there are many options to allow for older and less secure standards, as can be found in [94].

IS6 was executed successfully.

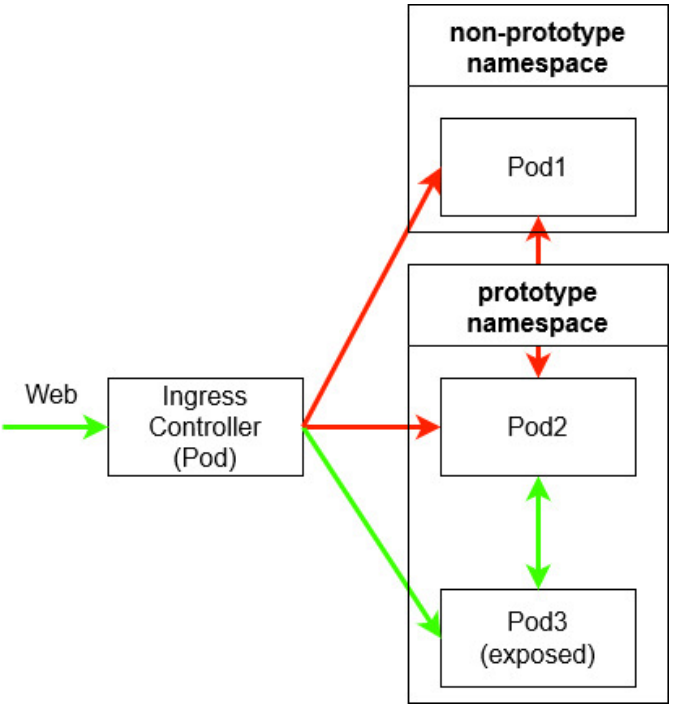


Figure 6.1: Validation setup of IS1. Red lines indicate unsuccessful communication, green lines indicate successful communication

P1 dictates that an estimate of the cost must be made. By running the prototype on all clouds, an estimation was made. However, it is not possible to estimate for a specific case. Some iPaaS providers might have more customers, increasing dynamic costs. This is further elaborated in Subsection 6.4.2.

The same applies for **P2**. Comparing OPEX cost can only be done in specific cases. More details can be found in Subsection 6.4.2.

P3 can be performed with Prometheus metrics. This, however, requires percize configurations, which we cannot proof detailled. The possibility is present. Additional tools include KubeCost [95].

O1 dictates that vendor dependency should decrease. It was proved this solution can run on all three major cloud providers. This shows the solution can be moved from one host to another, and thus the dependency decreases. This, however, is still up to the iPaaS providers to implement.

In order to fulfill **O2**, a HorizontalPodAutoscaler was setup. Detailed instructions are available here [96]. Using Prometheus metrics, custom scaling metrics could also be set up.

As for Responsibility, **R1** can be fulfilled by updating a namespace's ResourceQuota from the iPaaS solution portal.

R2 defines that daily tasks should be low effort. Since Kubernetes is autoscaling and auto-healing, little effort has to be done to keep the cluster running. Updates of Kubernetes

master plain and nodes take little time (or are fully automated). Initial setup or development is expensive since the communication between portal and Cluster API should be set up and be programmed based on the iPaaS provider's current workings.

In terms of **R3**, the portal can send desired cluster states to the Cluster API. This ensures the desired state by the portal is achieved.

R4 is difficult to discuss. In Kubernetes, workloads are run by ServiceAccounts. 'Users' are not a Kubernetes concept. Instead, anyone having a valid certificate signed by Kubernetes Certificate Authority (CA) is considered authenticated. Other authentication options include OpenID Connect Tokens, Webhook Tokens, and Bearer Tokens (see [97] for details).

6.3 Non-Functional Requirements

In this section, the non-functional requirements are validated. This is done by building a prototype as defined in section 5.2 and 5.4. Then, depending on the requirement, either the documentation was consulted whether the requirement is fulfilled, a test was setup to validate the requirement or a combination of both. The result can be found in Table 6.2.

Req	Yes/No	Explaination
DR1	Yes	Load balancers are hosted by the cloud provider. Containers can have as many replicas as desired.
DR2	Yes	By default, probing takes 40 seconds, which gives enough time for containers to startup.
DR3	No*	By default, workers could lose more messages.
DR4	Yes	Cluster availability is defined in the SLA of the cloud provider to be sufficient.
C1	Yes	An idle cluster costs between €65,- and €145,- per month. See Subsection 6.4.2
C2	Yes	A running cluster costs between €200,- and €260,- per month using one node (2 vCPU and 8GB RAM). See Subsection 6.4.2
C3	Depends	This is heavily dependent on current setup, see Subsection 6.4.2
M1	Yes	Dashboards can be defined, to watch cluster health, but all processes require no intervention
M2	Yes	Updating always takes time, but using helm charts the infrastructure can be maintained easily. Customer images are up for customers to maintain.
Pe1	Yes	On average, node creation takes 90 seconds. Startup of the container is dependent on it's function and size.

Table 6.2: Fulfillment of non-functional requirements of the solutions
**See explanation in the text*

DR1 describes that load balancers should not take more than the SLAs guaranteed time to start. Since IngressController containers can be spun up multiple times, it is only the time it takes for the cloud host which dictates how long this process takes. We can assume the cloud host does not take longer than their SLA.

As for the workers described by **DR2**, this is a bit different. Two possible methods to determine whether a pod is healthy exist. First, the container can crash properly (by exiting with an error code), causing Kubernetes to restart it or spin up a new pod. Secondly, if the container is unable to crash itself, a liveliness probe can be used. When the container is healthy, it should respond to some request defined in the container spec. When this probe request is not fulfilled properly, kubelet will assume it crashed, and restart it properly. A default value for this liveliness probe to test liveliness is 10 seconds, with the failure threshold being set at 3, meaning it can take at most 40 seconds before the container is deemed unhealthy, and restarted with default values. Custom configuration can make this even faster. We assume 140

seconds is enough time for containers to startup, although due to containers being different depending on their function, some containers can take longer to startup. In most cases, this would be enough time, or it would be up to the developers of the integration to split it, and thus we accept this requirement as completed.

As for **DR3**, it is marked as No in Table 6.2. By default, messages are not stored anywhere, and thus messages will be lost. Kubernetes does not implement a mechanism to solve this problem. However, as will be explained in 6.4, software solutions exist that can act as persistent queues. These persistent queue solutions are, for example, ActiveMQ Artemis[98], RabbitMQ[99] or Apache Kafka[100]. Kubernetes can support such systems but is up to the provider to implement

When considering the availability of the cluster (**DR4**), the availability of the master plane is meant. This is fully hosted by the cloud provider, and thus is generally is available according to the SLA. However, some exceptions rise here. When updating cluster settings or upgrading the control plane, the master plane needs to 'rebuild' and does not take new requests. For zonal clusters, this means no requests can be taken. Regional clusters, however, have more than one control plane and keep working as intended. NodePools are updated one at a time. Surge can be enabled to make Kubernetes add nodes temporarily while NodePools are upgrading. Note that no cluster configuration changes can be made when NodePools are upgrading. Also note that upgrading nodes cordons the nodes, meaning the pods need to be rescheduled. This should however not disrupt stateless containers in most cases. All combined, the cluster's master plane can be available in HA in regional clusters. Cluster configuration can be disrupted, but this does not cause downtime of the cluster access.

C1 and **C2** are elaborated in more detail in Subsection 6.4.2. This showed both requirements are met. For **C3**, this is again heavily dependent on the current situation. More details can be found in Subsection 6.4.2.

M1 and **M2** dictate that maintenance time should not exceed 8 hours for everyday maintenance and 8 hours for expert maintenance. For daily maintenance, dashboards can be configured to watch cluster health. This gives an overview of the current state in an instant. This ensures only maintenance is needed when the dashboard shows errors. In any other case, no intervention is needed. Although it is not provable in this research, the author did not have to maintain anything during the prototype, and thus the requirements are deemed fulfilled. In terms of expert maintenance, most updates are done automatically after setting the right configurations. These operations need to be watched, but no intervention should be required. Therefore, this requirement is also deemed fulfilled.

As for **Pe1**, provisioning nodes takes time. The time this takes depends on the cloud host and the type of node to be provisioned. Although the node might appear as 'not ready', pods will already be installed on the node. Therefore, it is hard to determine whether the requirement has been met. However, tests show that this takes about 90 seconds, and never longer than 180 seconds (n=9).

6.4 Case Study

To validate the solution, a case study was done. This was performed at eMagiz, which was introduced in Section 1.7. The workings of the portal have already been discussed in Subsection 1.7.1. The cloud architecture has not yet been discussed. Although customers deploy their integration with a press of a button, there is an environment deployed in the cloud to ensure all integrations can be run. Due to confidentiality and security, the exact setup cannot be described but is similar to the general overview in Figure 5.3.

6.4.1 Applying the solution design to eMagiz

To validate the reference architecture, the baseline cloud architecture of eMagiz was compared to the architecture in Subsection 5.1.4. The reference architecture was applied to the current cloud deployment of eMagiz and was verified with experts.

First, the baseline cloud architecture of eMagiz was discussed with three experts at the company. The architecture was modeled and can be found in figure 6.2, and was compared to Figure 5.3. A few things stand out being different from the model in Subsection 5.1.4. Firstly, two types of customer container deployments exist; single lane and double lane. Single lane has no application LoadBalancer within the customer VPC and one processor, while double lane does have an application LoadBalancer and two processors. Furthermore, an Artemis instance is deployed to queue AMQP messages. This ensures no messages are lost. HTTP messages are directed straight to the processors.

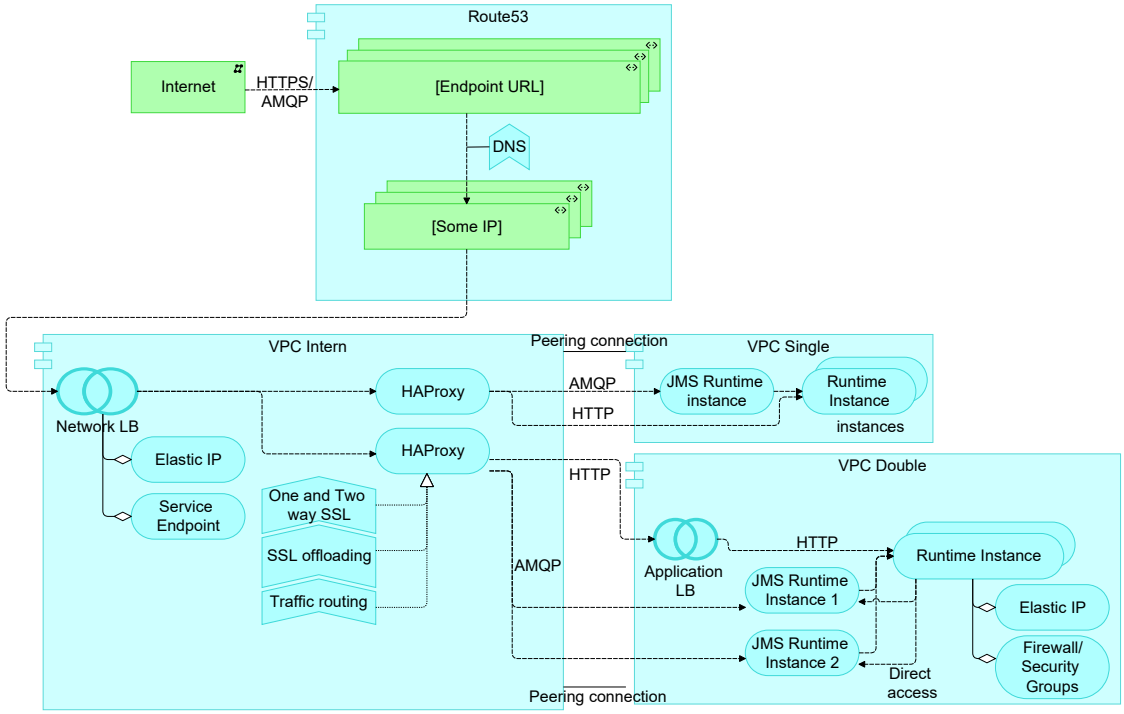


Figure 6.2: Cloud view of eMagiz.

Comparing Figure 5.4 with the situation at eMagiz, little differences can be identified. An image is created and pushed to an image repository, after which an instruction is sent to the VM to pull the image and set up the container.

Looking at it from a business perspective, customers can choose to either have one or two processors and no dynamic scaling between these options. This is in line with the identified problem; if the containers are not under workload, bought computing power is not used, while there is no way to scale up when a single lane solution is having trouble processing the data.

In general, the solution of eMagiz can easily be mapped on the models of Section 5.1. Two main differences from the general solution have to be taken into account though. Firstly, the image generated by the eMagiz platform could be different than the general workloads used to define the installation instructions defined in Section 5.4. Secondly, there is an additional component, being the Artemis instance(s), in front of the workers.

After comparing the architecture of eMagiz with the architecture defined in Section 5.1,

the generated image of eMagiz was investigated. This was done to ensure the image had no other dependencies which weren't already described in this report. One such exception exists. To pull the image from the repository, an ImagePullSecret was needed. This secret allows for authentication with the repository when pulling the image. How to set up such secrets is defined in the Kubernetes documentation[101]. This, however, stresses the importance of securing cluster access, since secrets can be compromised when cluster access is compromised.

Since the image does not have a helm chart yet, a custom YAML definition was created for the image. This custom definition consists of:

- **Namespace**: This to ensure the container is separated from other aspects of the solution
- **ResourceQuota**: Limiting the used resources within the namespace to validate its working and limit costs.
- **ServiceAccount**: This account runs the container.
- **Role**: A role with permissions according to what is needed for the container. In this case, `GET` operations on `secrets` and `GET` operations on `services`
- **RoleBinding**: To bind the role to the service account
- **Service**: To expose the container to the cluster on an internal IP and service name. Ports opened are according to the eMagiz requirements.
- **Deployment**: To run the container. The `ImagePullSecret` is used here.
- **Ingress**: To expose the container externally on the web.

Additionally, the IngressController was deployed (using steps 3 and 4 in 5.4) to ensure the Ingress routing was working. After that, the customized YAML definition deployed successfully on the cluster, and the container worked as intended; it was available to the web on the defined URL. The container was set up to use authentication, which was tested and worked as intended.

After testing the integration container, the other aspect not in the target architecture, the Artemis deployment, was tested. This was deployed similar to the integration container. Some additional ports were required to make Artemis and the integration container able to communicate, but no other problems arose.

This (partial) solution was tested to determine whether identified problems were solved. Firstly, the integration container was scaled up to three instances. By accessing the solution in the browser it was observed that messages were distributed to different containers each time in round robin fashion. This is expected and desired behaviour. Messages are evenly distributed between processors and thus share the workload. However, due to how eMagiz deploys the Artemis server, replicating Artemis was not possible. Artemis can either dynamically or statically find its peers, and eMagiz was set to do this statically. This finding has been documented. After this finding, a different option to validate the solution was identified. ArtemisCloud.io[98] is a solution to deploy Artemis on Kubernetes. This solution has been preconfigured to discover its peers dynamically. This proves that there is an option to deploy Artemis in a scalable manner. However, eMagiz was not configurable to use the ArtemisCloud instance instead of the eMagiz Artemis instance, and thus the complete workings could not be verified. (The author believes that changes in the eMagiz Artemis would allow the solution to work). Due to this limitation, the Artemis server was not tested to be scalable, since this would break the solution.

After validating the two parts that are different from the identified architecture, the full installation instructions were followed to deploy the entire solution. Several configuration files (`values.yaml`) needed to be changed to find and work with the Artemis server and the integration container.

6.4.2 Cost Analysis of the Case Study

During the case study, the cost of the solution was studied. To do so, some assumptions were already defined in section 1.6, including the exchange rate of €1.00 equals $1.10. All figures here are in euro's.

For this section, a cluster was set up using two nodes of each 2 vCPUs, and 8GiB Memory was used, for a total of 4 vCPU and 16 Gibibyte (GiB) Memory. Each node uses 32 GiB as the boot volume. Only on-demand prices were considered. Spot instances are only available when the cloud host has excess resources, but this solution cannot rely on this. Contracts could reduce the cost of VMs, but require a contract period, which is dependent on the iPaaS provider again. Therefore, flat, on demand prices were used. Lastly, the overhead of the cluster did not take the full 4 vCPU and 16 GiB ram it was able to take. Instead, on average, 2 vCPU (out of 4) and 1.2GiB Memory (out of 16GiB) was used for the overhead. This meant that, on average, 2 vCPU and 14.8 GiB Memory was available for workloads. This means the actual cost of the overhead is lower. However, since these VMs need to be reserved anyway (even when not completely used), we maintain this price as overhead.

The cost was divided into two parts. The static cost (or overhead cost) which is not connected to the amount of data that needs to be processed. This consists of the cost for the Kubernetes Engine (running the cluster), (part of) the Cloud Logging, Networking (service discovery), and DNS. For the dynamic part of the cost calculation, the Compute engine cost and the Cloud Logging cost were taken into account.

In the validation, three cloud providers are concidered, Google Cloud (GKE), AWS (EKS), and Azure (AKS).

Google Cloud

Running on Google Cloud in February 2022, we see the following cost for the static components, as can be found in Table 6.3:

Category	Component	Cost per month
Kubernetes Engine	Regional Kubernetes Clusters	€72,96
Cloud Logging	Log Volume	*€2,83
CloudDNS	ManagedZone	€0,20
Networking	Service Discovery	€0,10
Total		€76,09

Table 6.3: Cost of static components in Google Cloud
**Not all logs were static cost, see below*

For Cloud Logging, the total cost was €3,00. The number of logs contributing to static cost was around 10M log messages. The total volume of log messages was about 10.6M messages. Therefore, the estimated cost of the static component for log messages, was calculated as

$$\frac{10M}{10.6M} * €3.00 = €2.83$$

.

Running in February 2022, we see the following cost for the dynamic components, as can be found in Table 6.4

For Cloud Logging, the total cost was €3,00. The number of logs contributing to dynamic cost was around 0.6M log messages. The total volume of log messages was about 10.6M

Category	Component	Cost per month
Compute Engine	E2 Instance Core running in EMEA	€80,65
Compute Engine	E2 Instance RAM running in EMEA	€34,34
Compute Engine	Loadbalancing Minimum price	€16,03
Compute Engine	Static IP Charge	€6,52
Compute Engine	Storage PD Capacity (64 GiB)	€2,51
Compute Engine	Traffic (3.6 GiB)	€0,30
Cloud Logging	Log Volume	*€0,17
Total		€143,32

Table 6.4: Cost of dynamic components in Google Cloud
**Not all logs were dynamic cost, see below*

messages. Therefore, the estimated cost of the static component for log messages, was calculated as

$$\frac{0.6M}{10.6M} * €3,00 = €0,17$$

.

Combining the static and dynamic components of the overhead cost, gives us a total as described in Table 6.5.

	Cost per month
Static	€76,09
Dynamic	€143,32
Total (x1 scale)	€214,78
Total (x10 scale)	€1504,66

Table 6.5: Overhead cost using Google Cloud

AWS

As mentioned in Subsection 5.5.3, AWS has more requirements on how the VPC should be set up. This introduces higher overhead costs. Additionally, AWS does not split its usage as much as Google does. For AWS, to save credit, only three days in March were used for testing. However, this gives us enough indication to extrapolate this over the entire month.

Running in march and extrapolating cost, we see the following cost for the static components, as can be found in Table 6.6:

Category	Component	Cost per month
Kubernetes Engine	Regional Kubernetes Clusters	€72,96
VPC	NatGateway	€69,09
Route53	HostedZone	€3,60
Total		€142,55

Table 6.6: Cost of static components in AWS

Running in march and extrapolating cost, we see the following cost for the dynamic components, as can be found in Table 6.7:

Category	Component	Cost per month
Computing Engine	EC2 Instances (t4g.large)	€101,94
Computing Engine	Storage (64GiB)	€6,93
Total		€108,87

Table 6.7: Cost of dynamic components in AWS

Combining the static and dynamic components of the overhead cost, gives us a total as described in Table 6.8.

	Cost per month
Static	€142,55
Dynamic	€108,87
Total (x1 scale)	€256,05
Total (x10 scale)	€1235,88

Table 6.8: Overhead cost using AWS

Azure

Azure does not count static cost for clusters. However, for production environments, an SLA needs to be bought. Otherwise, an Service Level Objective (SLO) is applied to the product. SLO offer no guarantees about the availability, which is undesired for production environments. Running in march and extrapolating cost, we see the following cost for the static components, as can be found in Table 6.9:

Category	Component	Cost per month
Kubernetes Engine	Regional Kubernetes Clusters	€0,00
SLA	Uptime SLA*	€65,68
AzureDNS	HostedZone	€0,50
Total		€66,18

Table 6.9: Cost of static components in Azure
**SLA* *is needed to guarantee 99.5% uptime. Without, you get a free SLO of 99.5%*

Running in march and extrapolating cost, we see the following cost for the dynamic components, as can be found in Table 6.10:

Category	Component	Cost per month
Computing	Nodes	€130,31
Computing	Storage (64GiB)	€4,32
Storage	ReadWrite Operations	€0,00
Total		€134,63

Table 6.10: Cost of dynamic components in Azure

Combining the static and dynamic components of the overhead cost, gives us a total as described in Table 6.11.

	Cost per month
Static	€66,18
Dynamic	€134,63
Total (x1 scale)	€200,81
Total (x10 scale)	€1412,48

Table 6.11: Overhead cost using Azure

Comparison

In this subsection, the total cost of the three cloud hosts is summarized. Also, to indicate the cost when the solution grows, a 10 times larger NodePool is described, where the total computational powers are 20 vCPU and 80GiB Memory (x10 scale). The result can be found in Table 6.12.

	AWS	Google Cloud	Azure
Static	€142,55	€76,09	€66,18
Dynamic	€108,87	€143,32	€134,63
Total (x1 scale)	€256,05	€214,78	€200,81
Total (x10 scale)	€1235,88	€1504,66	€1412,48

Table 6.12: Cost comparisons overhead cost in AWS, Google Cloud and Azure. Costs are per month

This comparison was done based on relatively small machines (2 vCPU and 8GiB Memory). Which machines should be used is an entire research itself. Larger machines mean the overhead space is minimal but is expensive when only partly used. In this comparison, we see that AWS has high overhead cost, but low dynamic cost. This would mean AWS gets cheaper the more you use it. Azure has no cost for the cluster itself, meaning that if the SLA is not the highest priority (e.g. for a development cluster), this cluster could save quite some money (static cost is nearly nothing for Azure). Google is not as widely adopted as AWS, but cheaper, and has more functionalities than both AWS and Azure, and handles network traffic better [61].

Comparing overhead cost to the current situation

In this subsection, the estimated overhead cost is compared to the actual overhead cost. To do this, a few things need to be elaborated. Cost breakdown differ greatly between the old situation and the new situation. Almost all aspects will differ.

As such, we try to group current cost which could potentially be mapped onto the new situation cost, but this is far from accurate. Furthermore, due to confidentiality, we cannot post exact cost breakdown. Current static cost is around €60,-. On top of this, dynamic cost worth €330,- are made.

This means the static cost are a total overhead of around €390,-. This includes all containers required to run the basic setup. In order to compare the cost of previous Subsections to the costs of the current situations, the dynamic machine costs were updated to match the machines used in the current setup. Since Kubernetes uses shared resources, this can be

placed on different machines. The cost of the same resources compared to the current set up are displayed.

In the new situation, this would come down to the values found in Table 6.13.

Host	Static cost	Similar dynamic cost	Total Cost
Current	€60,00	€330,00	€390,00
AWS	€142,55	€310,90	€453,45
Google	€76,09	€338.73	€414.82
Azure	€66,18	€424,96	€419,14

Table 6.13: Cost comparison overhead cost in AWS, Google Cloud and Azure compared to the current situation. Costs are per month
Note: Dynamic cost do not equal dynamic cost in previous sections, since dynamic cost have been upscaled to match dynamic resources of the current situation

From this comparison, it is clear that the new situation has higher overhead cost. This was expected, as the overhead cost was expected to increase due to the required infrastructure and Kubernetes master plane cost. It is expected that the dynamic cost will make up for this.

Comparing dynamic cost to the current situation

In this subsection, we compare cost of the current solution and the target solution in terms of dynamic cost. For this, we use the prices of AWS to elaborate the process, but in the end a short comparison is also made for Azure and Google Cloud.

For the dynamic cost comparison, a few VMs were considered. Their cost were assessed and it was assessed how they could be fit on a Kubernetes stack. For this, a few different customers and integrations were considered. First, the VM sets with their remarkable features and average usages are described, followed by the machine properties. For an overview, see Table 6.14. After all VMs are described, all VM-sets are placed on a fictional Kubernetes stack to estimate the dynamic costs. These costs will be elaborated in text.

The first VM-set showed a clear daily patterns. Each day between 18:00 and 0:30 a spike in CPU usage was visible. On other times, the VM used about 5 to 7% CPU on average. On spiking moments, this increased up to little over 70%. However, no spike in Memory used could be detected during these hours. Rather, the used memory did not spike at any moment. Gradually it went up from 60 to 61%, but Garbage Collection (GC) brought it back to 60%. During GCs, a spike was visible in CPU usage. It was noted that 40% of the disk was used. A visualization of the CPU usage can be found in Figure 6.3.

The next VM showed other signs. During weekend days, the CPU usage was lower than during weekdays. During weekdays, CPU usage was between 8 and 18%. On weekend days this is on average 4%. Three peaks were observed, two of which due to GC, and one peak (55%) seemed to be due to a high burst of data, but no pattern was identified. RAM usage was between 48% and 53%. Disk usage was 34%. A visualization of the CPU usage can be found in Figure 6.4.

The third VM showed a CPU usage of between 8 and 9%. The were no spikes visible in the data. Again, no RAM was affected during this time. This usage was between 70 and 71%. Disk usage was 45%. A visualization of the CPU usage can be found in Figure 6.5.

Across all runtimes, RAM did not GC a lot. This means that most (or all) data is stored in the RAM. Additionally, the overview averages all cores. This means that low CPU does not mean all cores are idle, some cores might be working fully, while others are idle. E.g., when core one is processing a lot of data, while core two is idle, it would still show up as 50% CPU usage.

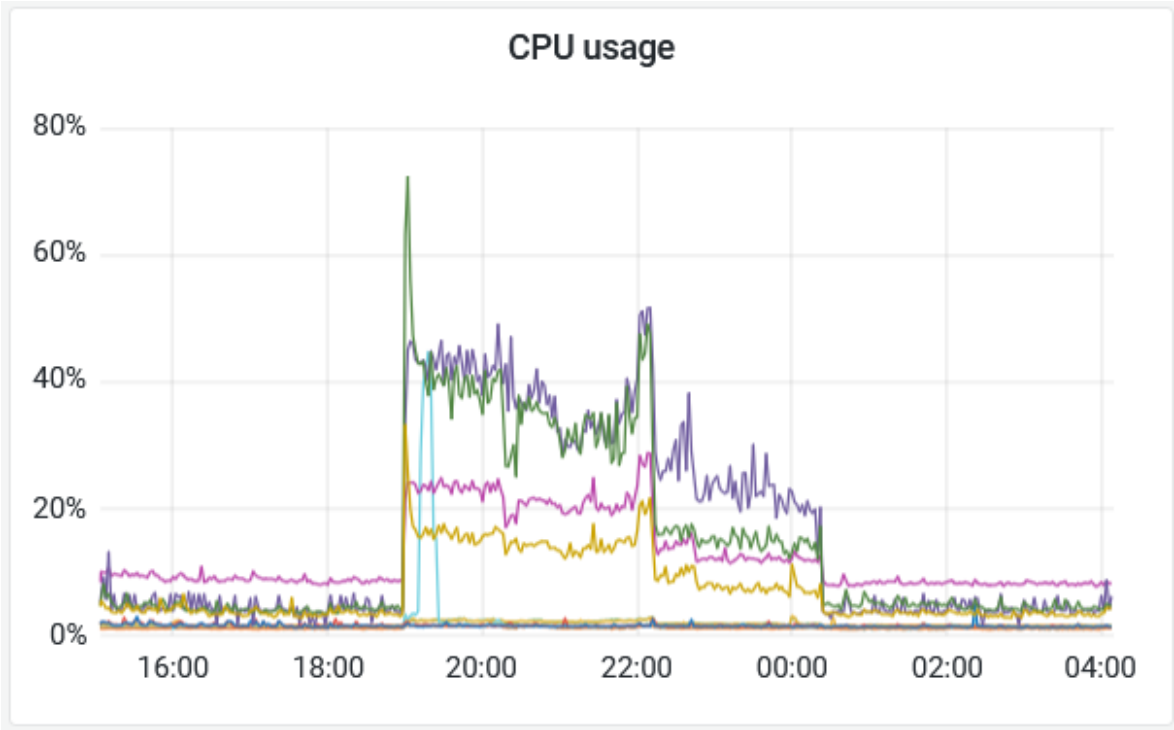


Figure 6.3: Statistics of CPU usage of VM-set 1. *All lines are the CPU usages of the different VMs in the set.*

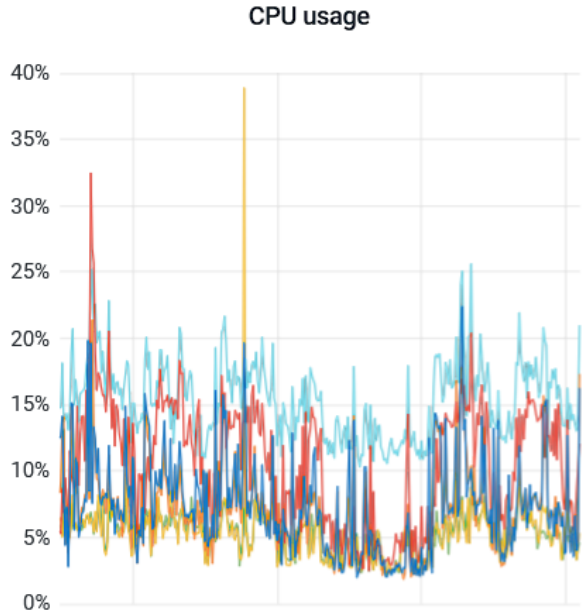


Figure 6.4: Statistics of CPU usage of VM-set 2. *All lines are the CPU usages of the different VMs in the set.*

VM-set 1 consists 32 CPU cores and 128 GiB RAM. Total cost for this VM-set (on demand) is about €1040,-. In order to determine the cost in the new situation, the total usage of all machines combined need to be determined, as Kubernetes can schedule it one one node instead of different VMs. For this VM-set, a total of 100GiB RAM (99,6) and 1 CPU core (0,94) is required during non-peak times, and 100GiB RAM and 5 CPU cores (4,34) are

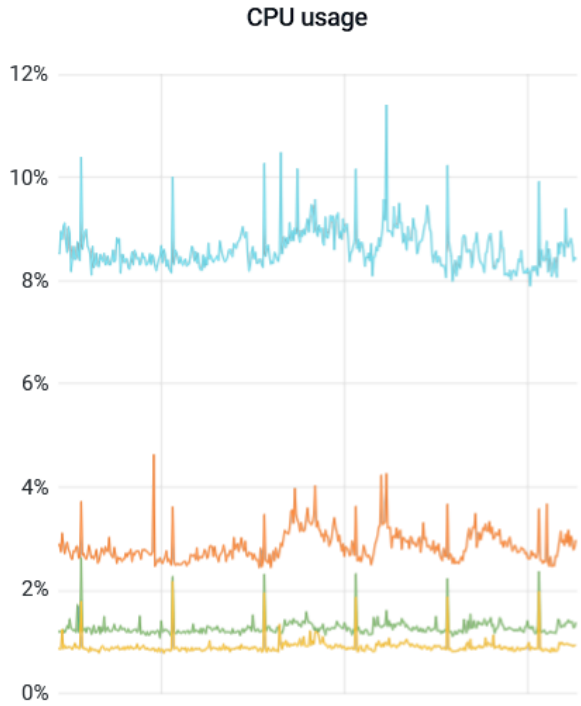


Figure 6.5: Statistics of CPU usage of VM-set 3. *All lines are the CPU usages of the different VMs in the set.*

VM-set	Spike?	Pattern	Base CPU	Peak CPU	Peak period	RAM%	Disk
1	Yes	Daily	5-7%	73%	3 Hours	60-61	40
2	Yes	Weekends	4%	8-18%	Days	48-53	34
3	No	-	1-9%	-	-	70-71	45

Table 6.14: Spike patterns in VM sets

needed during peak times. For this, Kubernetes would need two nodes of 8 CPU cores and 64 GiB RAM for a total of 16 CPU cores and 128 GiB RAM. The total cost for this would be €660,- (2 times r6g.2xlarge) which is a decrease of 36% in cost. This nodegroup would have more than sufficient overhead to host the solution. In this comparison, two large machines were used to make sure that replication can happen over atleast two nodes, but not limiting the nodes to smaller machines to minimize of the overhead per node.

VM-set 2 consists 16 CPU cores and 64 GiB RAM. The cost of this would be around €510,-. For this VM-set, a total of 47GiB RAM (46.8) and 2 CPU core (1.28) is required during non-peak times, and 47 RAM and 3 CPU cores (2.5) are needed during peak times. Since we already have VM-set 1 on the Kubernetes stack, and can use the leftover space, only a r6g.xlarge machine of 4 cores and 32 GiB ram would be needed, bringing the total to 20 CPU cores, and 160 GiB RAM for a total use of 7 cores during peaks, and 3 cores during offtimes, and 147GiB of RAM, giving us 13GiB ram and many cores for overhead.

The total cost for the old situation would be around €1550,- for dynamic cost, while using the Kubernetes shared resources would be €994,- for the dynamic cost. This is a decrease of 35%.

$$\frac{€994 - €1550}{€1550} * 100 = -35\%$$

Finally, VM-set 3 consists 16 CPU cores and 64 GiB RAM. The cost of this would be

€530,-. For this VM-set, a total of 45GiB RAM (44.48) and 1 CPU core (0.52) is required. This VM-set does not have peak times so this is also the required resources during peak times. To add this solution to the Kubernetes nodes, we need a total of 4x r6g.2xlarge machines are needed to put all three VM-sets on the stack. The cost of this would be €1319,80 for 32 Cores and 256GiB RAM. Adding this solution to the stack, the reduced dynamic cost is 36,5%

$$\frac{€1319 - €2077}{€2077} * 100 = -36.5\%$$

	Old Situation	New Situation
Max CPU (Cores)	7,36	7,36
Max RAM (GiB)	191	191
Allocated CPU (Cores)	64	32
Allocated RAM (GiB)	256	256
Total cost per month	€2077,-	€1319,-

From this, three things become evident. Firstly, the sharing of resources can save substantial amounts of money, since the unused VM space can now be used by other customers. This means the unused space in a VM is drastically decreased. Secondly, the RAM usage is the limiting factor here in reducing cost even more. This should be noted as a recommendation. Lastly, we notice that the RAM is needed both in peak and non-peak times, such that we cannot use scaling to handle peaktimes and thus reduce the number and size of nodes.

For Azure, the machine types and cost are different. Here, it would be best to get E8-4as machines, with 4 CPU cores and 64GiB RAM. Four of these would satisfy the needs for a total of €1446,79. On Google Cloud, five n1-highmem-8 machines could be used. Sizing is a bit different for Google Cloud machines. The total resources for this solution would be 40 CPU cores and 260 GiB of RAM for a total of €1401,59. This includes a Sustained Use Discount applied since specific machines were used. An overview can be found in Table 6.15

	Current Situation	AWS	Google	Azure
Max CPU (Cores)	7,36	7,36	7,36	7,36
Max RAM (GiB)	191	191	191	191
Allocated CPU (Cores)	64	32	40	16
Allocated RAM (GiB)	256	256	260	256
Total cost per month	€2077,-	€1319,-	€1401,-	€1446,-

Table 6.15: Comparison current- and new cost for dynamic components on AWS, Google Cloud and Azure

6.4.3 Conclusion Case Study

In the previous Subsections, the overhead and dynamic cost were calculated. Using these two cost breakdowns, a total estimate can be made. This can be found in Table 6.16. A graphical representation is given in Figure 6.6. From this, we can see a decrease in cost of 28%. On all clouds, the overhead cost of Kubernetes increases compared to the current solution. This is different for the dynamic cost, which are lower than the current situation. Knowing th dynamic cost will change when more customers are on the Kubernetes stack, the potential to reduce cost increases, since the overhead cost will not increase, which is the biggest factor in the new situation. We note here, that the memory used is the bottleneck of reducing cost even more. We must also note that the VMs chosen for the NodeGroup were based on basic knowledge. Potentially, better chosen machines could decrease cost more.

	Current Situation	AWS	Google	Azure
Overhead cost per month	€390,00	€453,45	€414,82	€419,14
Dynamic cost per month	€2077,00	€1319,00	€1401,00	€1446,00
Total cost per month	€2467,00	€1772,00	€1815,00	€1865,00

Table 6.16: Cost comparison eMagiz case study in the current- and new situation for overhead and dynamic components combined

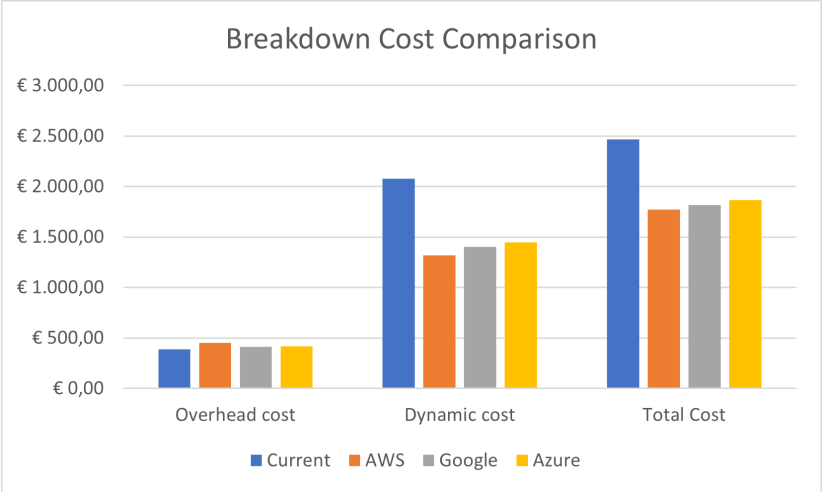


Figure 6.6: Cost comparison current- and new situation. Costs are per month

The decreased cost can be calculated as:

$$\frac{€1772 - €2467}{€2467} * 100 = -28.1\%$$

6.5 Validation Portability

In this section, the possible increase in portability is discussed. As per Techopedia, portability is "*a measure of how easily an application can be transferred from one computer environment to another. A computer software application is considered portable to a new environment if the effort required to adapt it to the new environment is within reasonable limits*"[102]. Thus, this section needs to prove the solution can be moved easier than before.

Current implementations are heavily dependent on the cloud environment. These environments can be deployed as code using various tools. AWS uses CloudFormation[103], Google uses Google Deloyment Manager[104], and Azure uses ARM Templates[105]. This means, that every mutation of the cloud requires full implemention of these manager. A central place, like a portal, would thus have to integrate with all these tools.

Instead, Kubernetes, once initialized, uses YAML files (or helm charts when available) and API calls, independent of the cloud provider. Moving an integration on AWS requires a CloudFormation mutation, and on Azure a ARM Template. Both, when running Kubernetes, can accept YAML files.

For example, the template in Appendix C is an example how to deploy an SQL instance to AWS according to `https://computingforgeeks.com/setup-aws-rds-mysql-db-instance-with-cloudformation/`. For Azure, the template can be found in Appendix D or `https://azure.microsoft.com/en-us/resources/templates/managed-mysql-with-vnet/`. For Google, the template can be found in Appendix E, which uses

`https://stackoverflow.com/questions/51152743/how-to-create-mysql-database-with-user-and-password-in-google-cloud-platform-usi`. All three examples use different syntaxes to deploy an mysql server. In Kubernetes, either using Helm or YAML files this is the same for all three cloud providers. The YAML file can be found in Appendix F. The installations using Helm are (using the Bitnami repository):

```
helm repo add bitnami https://charts.bitnami.com/bitnami &&
helm install mysql bitnami/mysql -n mysql --create-namespace
```

Additionally to the move internally within the cloud, moving from AWS to Azure (for example) and the other way around, both require the same YAML, only sent to a different endpoint.

6.6 Validation Flexibility

In this section, the possible increase in flexibility is discussed. Flexibility is a widely used term in IT, but no formal definition exist for this specific domain. Most definitions focus on the flexibility of software or SaaS solutions, not as much on IaaS or iPaaS domains. Golden discussed that the definition of flexibility consist of four dimensions: temporal, range, intention and focus[106]. Temporal is concerned with *the time which it takes an organisation to react to change*. This means if a company can respond fast to changes, it has an higher temporal flexibility than when responses are slow. Also, the fit of the solution is important. If the solution fits the problem better, it is considered more flexible. Range is defined as *the potential responses that the organisation possesses, first, to changes which were foreseen and, second, to those not foreseen.* This is measured by the amount of possible responses to foreseen (versatility) and unforeseen (robustness) events. Intention is either offensive or defensive. Offensive flexibility uses changes to gain business advantages, while defensive react to changes to minimize impart. Last, focus can be internal or external. Internal focus concerns organizational structure and employee flexibility. External focus concerns suppliers and alliances. Golden mentions that intention and focus are based on the domain flexibility is applied to[106].

In this validation step, the author sees no reason to take into account intention and focus. These two aspects are, as far as the author is concerned, not applicable to software solutions. Rather, these would apply to the organization. In this validation phase, this is not of interest.

Therefore, the two axis and four measurements, efficiency and responsiveness (temporal), and versatility and robustness (range), need to be assessed to determine how flexibility has changed.

In terms of robustness, Kubernetes has built in mechanisms for disaster recovery. Although it does not natively support node auto-repair, most cloud providers do offer this functionality. Next to this, Kubernetes will restart failed containers based on the auto-start policy, meaning that failed containers will start again if configured so. Furthermore, replications allow to catch single point of failures. Traditionally, deploying two instances is not a problem, but detencting if one is down, and then starting another instance, might require additional tooling. Kubernetes offers this natively. This increases both the versatility and the robustness of the solution.

Furthermore, since Kubernetes uses Ingresses to define routing rules, canary release become more easy. This allows to be more flexible in how deployments should be rolled out. Although canary releases are possible without Kubernetes

These assumptions were tested with three employees at the company. The results can be found in Table 6.17:

Aspect	Result	Remark
Efficiency	4.0	Clients always start with smaller-medium sized cloud resources. At some point there is a need for more, but it's very unpredictable when. This is easier with the proposed solution
Responsiveness	4.3	Scaling of containers and auto provisioning increases responsiveness greatly.
Versatility	3.3	No comments
Robustness	4.0	Because the Kubernetes control plane covers a larger part of the total solution compared to the baseline architecture, the chances that something unforeseen needs changes outside of this control plane decrease significantly
Average	**3.9**	

Table 6.17: Flexibility assessment
In this table, 1 means a heavy decrease, while 5 means a heavy increase of flexibility. All weights are equal.

Subject	How
Requirements	The author has validated the requirements using the prototype
Models	Two experts have validated the models. Feedback has been encorporated in current models
Portability	Portability has been reasoned why it should have improved, using real scenario's
Flexibility	Flexbility has been reasoned, and three employees have been asked about their opinion
Real Scenario	The reference architecture was applied to a case study in order to validate the prototype
Dynamic Cost	Prototype cost have been compared to the case studies' current dynamic cost
Static Cost	Prototype cost have been compared to the case studies' current static cost

Table 6.18: Validation strategies in this research

From this, it is clear that on three of the four aspects, according to the employees, flexibility increases significantly. For one of the aspects, it increases slightly. Combined with the theoretical grounds for improved flexibility gives enough support to claim flexibility increases.

6.7 Conclusion

In this chapter, the solution was validated. Multiple strategies have been used, summarized in table 6.18. In this chapter, it was validated most requirements were met, except for the need to persist messages on failures. The requirement has been discussed, and most enterprise solutions will have this solved within their product. The models in this report were validated, and changed based on expert feedback. Portability and flexbility were assessed, and determined to have increased. Cost comparison of the current and target architecture showed a substantial decrease in cost.

Chapter 7

Conclusions

In this chapter, the research will be concluded. First, a conclusion is drawn, answering the research questions of Section 1.3. Next, the solution is considered for other domains than this research focussed on. This could open up potential new research and could provide a broader domain to which the research extends. Then, the research is discussed. During the discussion, the validity of the approach is validated, alternative interpretations are presented, and limitations are described. Also, the contribution to science and industry are discussed.

7.1 Revisiting the Research Questions

In this section, an answer will be given to the research questions defined in chapter 1.3. This research focussed on giving direction to iPaaS providers to improve their portability, flexibility and decrease cost by introducing Kubernetes into the cloud architecture. This was needed because traditional VMs can cause a lot of waste of money by not utilizing the available resources. Furthermore, adopting clouds without a framework that is accepted by all major clouds can introduce vendor lock-in, and decrease portability. Kubernetes furthermore claims to improve flexibility. The research questions will be broken down to give precise answers to the questions.

The main research question of this research is defined as:

MQ How should an iPaaS implement container orchestration in order to improve flexibility and portability, and reducing cost?

This research provides a reference architecture of the assumed cloud architecture and a possible target architecture. Kubernetes has been proven to increase portability and flexibility, and reduce cost for the assumed cloud architecture, using the reference architecture in Figure 5.11. In order to elaborate on the reasoning behind this answer, one by one the Subquestions will be discussed based on the findings. The first question defined was:

SQ1 What is container orchestration and iPaaS, and how can they work together to improve flexibility portability, and reducing cost?

The answer to this question can be found in Chapter 2. In this chapter, different types of XaaS solutions are discussed, including iPaaS. iPaaS is a type of XaaS focussed on designing integrations using a portal, and deploying the integration in the cloud. This cloud deployment can be reached by systems to connect to other systems in such a way, that semantic and syntactic differences are solved. Container orchestration is a type of container management focusing on seven main capabilities, including: Cluster state management, providing high availability and fault tolerance, ensuring security, simplifying networking, enabling service

discovery, making continuous deployment possible, and providing monitoring and governance. Both can work together by applying container orchestration to the cloud-deployed containers handling the integrations. This introduces benefits like failure detection, state management, flexible scheduling across the cluster, and the possibility to add redundancy. Most container orchestrations work with frameworks that allow being run on multiple clouds (improving portability), have mechanics for failure detection or solving (flexibility), and make better use of the scalability of containers (reducing cost).

Next, SQ2 was defined to determine important requirements specifically for iPaaS solutions:

SQ2 Which specific requirements should be covered to improve the flexibility and portability and reduce the cost of an iPaaS solution to ensure feasibility and similar functionality?

This question was answered in Chapter 3. Four categories were defined for functional requirements, including isolation and security, pricing, organization, and responsibility. For the non-functional requirements, an additional four categories were defined. These are disaster recovery, cost, maintenance, and performance. A total of 25 requirements were defined, based on interviews held with different people from iPaaS providers.

Next, the available solutions are discussed in the next question:

SQ3 What container orchestration solutions exist for iPaaS cloud architectures and what is the tradeoff between these solutions? How do these solutions meet the requirements and what would be the disadvantages of these solutions?

Chapter 4 was devoted to answering this question. Using the techniques found in the literature of the previous question, and by researching these specific techniques, four solutions were discussed: Kubernetes, Docker Warm, Spring Cloud Skipper, and Apache Mesos. Apache Mesos was discontinued between the last reference in the literature and now, and thus was no viable option. For Spring Cloud Skipper, the community was too small. Only a few sources discussed this solution causing their activity to be very low. It was also not widely adopted. On the other hand, both Kubernetes and Docker Swarm are actively followed by the community and have a high activity. These two solutions were discussed in more detail including a comparison of costs. Furthermore, the fulfillment of the requirements was discussed. In this comparison, Kubernetes fulfilled more requirements than Docker Swarm. For Kubernetes, in that stage of the research, it was not clear if Kubernetes would decrease vendor dependency, and whether it would require expert personnel. Other requirements were fulfilled. Choosing Kubernetes over Docker ensured more requirements were fulfilled. However, it was also discussed Docker Swarm was easy to set up and manage. This trade-off had to be accepted to fulfill critical requirements like the ability to handle multi-tenancy.

Using the results found in Chapter 3 and 4, the question was discussed:

SQ4 Can a reference architecture be derived as solution design, and can this architecture be introduced into the as-a-service deployment landscape of iPaaS solutions?

For this, Section 5.1 discussed the current architecture of iPaaS providers. Next, the gap was discussed in Section 5.3. This was combined with the requirements and the information about Kubernetes to define a reference architecture, considering the deployment process and the cloud architecture. A prototype was created, and an installation manual was given to replicate the setup. On all three major cloud providers, the prototype worked as intended, given the differences explained in Section 5.5. The result was a reference architecture as can be found in Figure 5.11.

With the prototype and reference architecture ready, the next question was tackled:

SQ5 Does the reference architecture solve the problem identified, taking into account the requirements of portability, flexibility and cost reduction? of iPaaS solutions?

This question contains multiple validation steps. First, the reference architecture needed to be validated for validity. This was checked by experts and accepted as described in Section sec:validationmodels. Next, the fulfillment of the requirements needed to be validated. This was done in Section 6.2 and 6.3. One requirement was not fulfilled, being "*When a worker fails, it may lose at most 0.05% of messages*". Kubernetes does not provide mechanisms for this. However, in the enterprise setting, most providers will already have such measures available. It would be up to the iPaaS providers to ensure this does not happen. Furthermore, two requirements could be fulfilled but were too much dependent on the setup of the cloud architecture. These were P2 and C3. P2 could not be verified, because there is no data available on the average OPEX cost of iPaaS providers. However, in Subsection 6.4.2, it was discussed for the case study. The same applies to C3 since no average is available for overhead per customer. Next, the dynamic and static components for cost were analyzed. This was done by comparing the cost of the case study with the cost of the old situation. The total of the overhead and dynamic costs showed a decrease of 28,1% for the entire solution. Although the overhead cost increased by 10%, the dynamic cost made up for this as it decreased by 36,5%. The total decrease in cost will most likely increase more with more customers migrating to the Kubernetes stack.

Next, SQ6 was answered:

SQ6 To which XaaS solutions would this reference architecture be applicable, and what advantages would this bring?

This was discussed in Subsection 7.2.1. The comparison was made with multi-tenant SaaS and PaaS solutions. However, no evidence has been given. This is up for future work to discuss this more in-depth. For this, additional literature needs to be generated to do this more in-depth.

Reflecting on the Main Question of this research, we have all the required information to answer this:

MQ How should an iPaaS implement container orchestration in order to improve flexibility and portability, and reducing cost?

This research provides a reference architecture of the assumed cloud architecture and a possible target architecture. Kubernetes has been proven to increase portability and flexibility, and reduce cost for the assumed cloud architecture, using the reference architecture in Figure 5.11. Industry can use this reference architecture and cost analysis to determine if the proposed architecture also applied to their business.

7.2 Contributions

In this section, the contributions are summarized. First a generalization is discussed. This was already mentioned in Section 7.1. This is needed because contributions in the research might also be applicable to other domains, which opens up contributions to multiple areas of research, and opens up options for future work. Then, the contribution to both practice and research are discussed. Lastly, recommendations to practice are given.

7.2.1 Generalization

In this section, the solution is generalized. First, it is discussed whether the solution could be generalized for all iPaaS solutions. Secondly, it is discussed if the solution can be applied to other domains than the domain discussed in this research, iPaaS. This research was designed to fit the needs of iPaaS, but some parts could perhaps be reusable for other domains. It was already discussed that integrations of customers are like a black-box, the infrastructure should be compatible with many different implementations of these integrations. If we use this principle, we can abstract what application is run behind the infrastructure. The infrastructure should work even if we change whatever is run within the containers (or pods).

Firstly, we generalize towards all iPaaS solutions. As discussed, a reference architecture was designed to attempt to make the solution as generic as possible for all iPaaS solutions. However, it is hard to validate this, as only one case study was considered. On the other hand, the architecture derived is generic enough that most iPaaS solutions would be able to use this as a starting point, adding what they have additionally to the reference model themselves. This would mean the additional aspects need to be added to the analysis, including its cost in the cost comparison. This is then customer-specific and thus did not belong in the reference model. Therefore, the author argues this iPaaS model would be generic for all iPaaS.

Secondly, towards more X-as-a-service principles is generalized. In SaaS solutions, companies provide a software solution, usually to multiple customers (multi-tenant), either by deploying one solution, or multiple solutions, one for each customer. In the latter case, it would be very similar to the current proposed architecture, where namespaces separate customers, but one central entry point could be used. The separation happens in the cloud, on the infrastructure before the actual application. This would mean the current solution would fit such situations, where the deployed solution is able to scale efficiently (e.g. consists of microservices). For applications where separation happens on the application layer, this solution would also be applicable, but less interesting. This is because the use of Ingresses is not required, and no separate endpoints are needed, and thus the power of external-dns and cert-manager is not fully used. However, some overlap can exist and thus would be interesting for future research. More research has already been performed in the SaaS domain, for example by Verreydt[107]. This still opens up the opportunity to compare results for both domains, check if there are similar results, and thus the possibility to make a general solution for multiple domains.

In general, anything built on top of the IaaS layer would maybe be interesting to apply this solution on. This also includes PaaS solutions for the same reasons as for SaaS solutions. The abstraction of middleware and OS can be inside containers, making it nearly no different from SaaS solutions. Kubernetes would deliver the IaaS layer, on which solutions could be built. Especially situations with multi-tenant requirements might benefit from this solution.

7.2.2 Contribution to practice

The goal of this research was to give a scientific basis to the industry for transition to frameworks used for container orchestration. Currently, no literature exists giving the industry basis for such transition. This research adds to the limited literature by providing a scientific basis for choosing what frameworks to adopt and potential solutions how to integrate this. The industry can use this paper to compare their cloud setup and cost and determine how much their architecture compares to the reference architecture provided. If that seems to be closely related, they can use the results of this research to determine the potential benefit of implementing such a framework. Thus this research contributes to practice by providing:

- A reference architecture for the domain of iPaaS, allowing enterprises to compare

themselves to the reference architecture of this report.
- A manual on how to deploy the reference architecture on the three biggest cloud providers.
- Empirical results comparing the reference architecture without container orchestration to an architecture with container orchestration.
- Reasoning why some plugins were used to give a base to the industry what plugins would work or not.

As discussed in Subsection 7.2.1, some results could also apply to SaaS solutions, but the validity of this generalization still needs to be validated.

7.2.3 Contribution to research

Contribution to research has overlap with contribution to practice. Firstly, the reference architecture derived in this research could be a base for research to discuss the architecture of the iPaaS domain. Furthermore, a potential generalization was identified between the cloud architecture of iPaaS solutions and the (multi-tenant) SaaS cloud architecture.

Thus this research contributes to research by providing:

- A reference architecture for the domain of iPaaS, which can be used in future work to compare iPaaS architectures.
- A potential generalization between SaaS and iPaaS solutions.
- A reference architecture for adopting Kubernetes in iPaaS solutions.
- Empirical evidence that sharing of resources can save substantial amount of money.
- An observation that high RAM usage limits the reduction of cost when replication is needed.

7.2.4 Recommendations to Practice

This research was validated by applying the hypothesis to the real world scenario of eMagiz. From this, several recommendations were found which could both contribute to the reduction of cost of eMagiz, and potential enablers for future research or development. These were all discussed in Section 6.4 already, but will be summarized here.

- The scaling was made difficult by the amount of RAM assigned to each VM and container. Scaling up these containers would require substantial RAM. This causes inefficient scaling. Therefore, scalable containers should be identified, and its RAM usage should be as low as possible. This enables more efficient scaling of high volume containers.
- eMagiz Should setup systems to either detect or ask customers for peak times. This allows to scale in advance. Scaling based on resource pressure could delay messages, while it was identified that these peak times were on set intervals. This allows scaling in advance and no delay in the delivery of messages.
- eMagiz Should pack containers with high velocity together or in separate containers. This allows for scaling only on containers which require more resources, without scaling low velocity containers.

7.3 Limitations and Future Work

In this section, the results of the research are discussed. This includes the discussion of the validity of the research, interpretation of the result, possible alternative strategies that were considered, and limitations of the work.

7.3.1 Validity of the research

In this subsection, the validity of the research is discussed. The guidelines of Hevner[15] were discussed in Section 1.4.

To determine to what extend the research was valid, the guidelines are assessed in Table 7.1.

Guidelines	Assessment
Guideline 1	The research produces a reference architecture and installation instructions
Guideline 2	The solution is technology based, and virtualization and containerization are relevant topics
Guideline 3	The solution was validated based on requirement fulfilment, model validation, cost comparison, performance and flexbility assessment. These validations were based on multiple methods, including a case study, architecture analysis, functional testing, informed argumentation and scenarios.
Guideline 4	After extensive literature research, no previous research was found combining Container Orchestration and the domain of iPaaS providers. This research combined industry applied knowledge with scientific research.
Guideline 5	This research did not provide many empirical results, except for the cost analysis. However, Hevner mentions the *how well* is important, rather than *how* and *why*. The author believes this is achieved by the multiple validation methods used This assesses all aspects of the solution rather than only looking if it works.
Guideline 6	This research was not performed iteratively. This causes the solution to be non-optimal. For an optimal solution, the result of the validation step should be input for another design cycle
Guideline 7	The research was presented to stakeholders from a company (eMagiz) in this domain, aswell as to researches at the university of faculties of both Behavioural, Management ad Social Sciences, and Electrical Engineering, Mathematics and Computer Science.

Table 7.1: Design Science Research Guidelines according to Hevner[15] assessed

From this table, we determine that six guidelines have been followed. One guideline, number 6, was not fulfilled, since the research was not done iteratively. This means the optimal solution has not been reached. It can be argued that the first prototype and the case study were steps in an iterative process, since the prototype did not fully work in the case study, and changes had to be made. Additionally, components of the research (including requirement engineering) were done in multiple sessions. Due to time constraints of the research, this could not be done iteratively. Nonetheless, this guideline was not fulfilled as described. This notion has been accepted as a flaw of the research. Additionally, guideline 5 was deemed followed by the author. This, however, could be subject to discussion, as not many empirical results were presented.

7.3.2 Alternative interpretation of results

In this subsection, some alternative views on the research will be discussed. This is to give the author's view on other interpretations of the work done by this research.

Firstly, one can argue the research validated the working of Kubernetes instead of providing proof of the solutions working for the specific domain chosen for the research. However, during the case study, a solution currently used by iPaaS solutions was obtained, and based on this, the prototype was validated. This indeed also validates the workings of Kubernetes, moreover, it validates the correct workings of solutions currently used by iPaaS solutions. This is a more

specific validation than just validating that Kubernetes works.

Secondly, the cost comparison is based on just one company's infrastructure, and thus one can say it is not representative for the entire domain. Although there is some truth in this, it must be noted that a generic architecture has been derived for iPaaS solutions first. This was later validated to be in line with the model made, and thus the company would be representative for the model designed earlier. Additionally, even if this were not the average case, this would most likely be on the least beneficial side for the research. The author believes that the cost of the case study was low, and similar solutions would either be equal or more expensive, making the solution even more cost-efficient than the current case study showed. The motivation behind this is that the VM-sets considered in the case study were relatively large, with busy machines. Other VM-sets were smaller, but also showed more unused VM space. This could lead to even better use of resources when these VM-sets are considered.

7.3.3 Limitations of the research

In this Subsection, various limitations of the research are discussed.

Firstly, we notice that Subsection 7.3.1 concludes not all guidelines of a Design-Science Research have been followed. As mentioned, the research was not performed iteratively. This means the solution found is possibly not optimal. Instead, the results found in the validation should have been input for another round of development. This was not done due to the time available for this research. Creating the requirements, assessing possible techniques, and creating the prototype were all time-intensive.

Secondly, although cost improvement has been shown for the case study, there is no generalization available to claim cost reduction. For this, either literature needs to become available, or more case studies have to be performed. For now, this research was validated using the infrastructure of the case study. Additional case studies could overcome this limitation and add to the claim of this research. This is more extensively discussed in Subsection 7.3.2.

Thirdly, the case study did not allow for empirical evidence of the benefit of auto-scaling of pods. Since RAM was always the limiting factor, and not the peak load on the integrations, this could not be validated from the case study. One can imagine performing emprical tests where GC is forced would discuss whether the RAM used could be lowered. It could also show performance issues when more GC occurs. For this, one can limit the amount of RAM available and thus force GC, and see what happens with the performance.

Fourthly, No evidence was found to support the claim that the treatment might work for SaaS solutions, rather than only for iPaaS solutions. More work is required to validate this claim. Due to time constraints this claim was not investigated further.

Lastly, it was observed that one requirement *"When a worker fails, it may lose at most 0.05% of messages"* was not fulfilled. This was discussed in Section 6.3. This is a limitation of the treatment designed in this research, and raises potential for an improved architecture. Later developments in the field might add functionalities to solve this problem. A few projects were refenced in Section 6.3 which might be able to solve the problem with additional tooling.

7.3.4 Future work

The author has identified a few possibilities for future research, allowing to get more insight into some aspects of the industry. From this, research could deliver insights for industry in terms of models or frameworks to aid in decision making. Many of the points summarized here have already been discussed in detail.

- A base model for cloud architecture of iPaaS was derived. This should be validated or extended. Doing so will give future work better estimation models, as this research ran

into this boundary.

- This research was validated based on one case study, both due to confidentiality and time available. Performing similar experiences at iPaaS providers could confirm or reject this generalization, giving more insight into the industry.
- As discussed in Subsection 7.2.1, SaaS application where separation of customers happens in the application layer was thought to be less relevant to apply this research on. However, this is an assumption, and the contradiction might be true. This opens for future research to apply the research on this specific domain, and to make changes if necessary. The results of that research might also shed new light on this research.

Bibliography

[1] ResearchAndMarkets, *Cloud Microservices Market - Growth, Trends, COVID-19 Impact, and Forecasts (2021 - 2026)*, 2021. [Online]. Available: https://www.researchandmarkets.com/reports/4787543/cloud-microservices-market-growth-trends.

[2] ExpertMarketResearch, *Integration Platform as a Service (iPaaS) Market*, 2021. [Online]. Available: https://www.expertmarketresearch.com/reports/integration-platform-as-a-service-ipaas-market.

[3] ENTERPRISE MANAGEMENT ASSOCIATES, "Reducing Operational Expense (OpEx) with Virtualization and Virtual Systems Management," Tech. Rep., Nov. 2009.

[4] C. Pahl, "Containerization and the PaaS Cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015. DOI: 10.1109/MCC.2015.51.

[5] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and Virtual Machines at Scale," in *Proceedings of the 17th International Middleware Conference*, New York, NY, USA: ACM, Nov. 2016, pp. 1–13, ISBN: 9781450343008. DOI: 10.1145/2988336.2988337. [Online]. Available: https://dl.acm.org/doi/10.1145/2988336.2988337.

[6] A. Khan, "Key Characteristics of a Container Orchestration Platform to Enable a Modern Application," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 42–48, Sep. 2017. DOI: 10.1109/MCC.2017.4250933.

[7] Stackoverflow, *What is the maximum number of edges in a directed graph with n nodes? [closed]*, 2011. [Online]. Available: https://stackoverflow.com/questions/5058406/what-is-the-maximum-number-of-edges-in-a-directed-graph-with-n-nodes.

[8] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A Virtual Machine Re-packing Approach to the Horizontal vs. Vertical Elasticity Trade-off for Cloud Autoscaling," 2013.

[9] J. Opara-Martins, R. Sahandi, and F. Tian, "Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–18, Dec. 2016, ISSN: 2192113X. DOI: 10.1186/S13677-016-0054-Z/FIGURES/11. [Online]. Available: https://link.springer.com/articles/10.1186/s13677-016-0054-z%20https://link.springer.com/article/10.1186/s13677-016-0054-z.

[10] R. Nishant, S. C. Srivastava, and B. Bahli, "Does virtualization capability maturity influence information systems development performance? Theorizing the non-linear payoffs," *Proceedings of the Annual Hawaii International Conference on System Sciences*, vol. 2020-January, pp. 5503–5512, 2020, ISSN: 15301605. DOI: 10.24251/HICSS.2020.677.

[11] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A Comparative Study of Containers and Virtual Machines in Big Data Environment," in *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2018-July, IEEE Computer Society, Sep. 2018, pp. 178–185, ISBN: 9781538672358. DOI: 10.1109/CLOUD.2018.00030.

[12] K. Peffers, T. Tuunanen, C. Gengler, *et al.*, "The design science research process: A model for producing and presenting information systems research," *Proceedings of First International Conference on Design Science Research in Information Systems and Technology DESRIST*, Oct. 2006.

[13] T. U. Rehman, M. N. A. Khan, and N. Riaz, "Analysis of Requirement Engineering Processes, Tools/Techniques and Methodologies," *International Journal of Information Technology and Computer Science*, vol. 5, no. 3, pp. 40–48, 2013, ISSN: 20749007. DOI: 10.5815/ijitcs.2013.03.05. [Online]. Available: http://www.mecs-press.org/.

[14] M. Ahmed Abbasi, J. Jabeen, Y. Hafeez, D.-e.-B. Batool, and N. Fareen, "Assessment of Requirement Elicitation Tools and Techniques by Various Parameters," *Software Engineering*, vol. 3, no. 2, pp. 7–11, 2015, ISSN: 2376-8037. DOI: 10.11648/j.se.20150302.11. [Online]. Available: http://www.sciencepublishinggroup.com/j/se.

[15] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, ISSN: 02767783. [Online]. Available: http://www.jstor.org/stable/25148625.

[16] eMagiz, *Emagiz*. [Online]. Available: https://www.emagiz.com/.

[17] Cape Groep, *Cape groep*. [Online]. Available: https://www.capegroep.nl/en/homepage/.

[18] H. Sneed and R. Seidl, *Softwareevolution: Erhaltung und Fortschreibung bestehender Softwaresystem*, 1st ed. dpunkt.verlag, 2013, p. 302, ISBN: 978-3-86490-041-9.

[19] A. Gowri, "Impact of Virtualization Technologies in the Development and Management of Cloud Applications," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 7, no. 2, pp. 104–110, Jun. 2019, ISSN: 2147-6799. DOI: 10.18201/ijisae.2019252789. [Online]. Available: https://www.ijisae.org/IJISAE/article/view/891.

[20] S. Liu, Y. Zhang, X. Meng, S. Liu, Y. Zhang, and X. Meng, "Towards High Maturity in SaaS Applications Based on Virtualization: Methods and Case Study," *International Journal of Information Systems in the Service Sector (IJISSS)*, vol. 3, no. 4, pp. 39–53, 2011, ISSN: 1935-5688. [Online]. Available: https://econpapers.repec.org/RePEc:igg:jisss0:v:3:y:2011:i:4:p:39-53.

[21] J. Gibson, R. Rondeau, D. Eveleigh, and Q. Tan, "Benefits and challenges of three cloud computing service models," in *Proceedings of the 2012 4th International Conference on Computational Aspects of Social Networks, CASoN 2012*, 2012, pp. 198–205, ISBN: 9781467347921. DOI: 10.1109/CASoN.2012.6412402.

[22] NIST, *Software as a Service (SaaS)*. [Online]. Available: https://csrc.nist.gov/glossary/term/software_as_a_service.

[23] NIST-CSRC, *Computer Security Resource Center*. [Online]. Available: https://csrc.nist.gov/.

[24] S. A. Rajathi and J. Devagnanam, "Exploring and Understanding The Cloud Environment with Resource Allocation Techniques," *Journal of Science and Technology*, vol. 6, p. 7, 2021. DOI: 10.46243/jst.2021.v6.i3.pp07-12. [Online]. Available: www.jst.org.indoi:https://doi.org/10.46243/jst.2021.v6.i3.pp07-12.

[25] NIST, *Platform as a Service (PaaS)*. [Online]. Available: https://csrc.nist.gov/glossary/term/platform_as_a_service.

[26] NIST, *Infrastructure as a Service (IaaS)*. [Online]. Available: https://csrc.nist.gov/glossary/term/infrastructure_as_a_service.

[27] N. Serrano, J. Hernantes, and G. Gallardo, "Service-Oriented Architecture and Legacy Systems," *IEEE Software*, vol. 31, no. 5, pp. 15–19, Sep. 2014, ISSN: 0740-7459. DOI: 10.1109/MS.2014.125. [Online]. Available: https://ieeexplore.ieee.org/document/6898686/.

[28] M. Potocnik and M. B. Juric, "Integration of SaaS using IPaaS," in *Pr oceedings of the 1st Inter national Confer ence on CLoud Assisted Ser viceS*, vol. 1, 2012, pp. 35–41.

[29] K. Ring, "EAI: Making the right Connections," *Ovum Reports, Boston*, 2000.

[30] N. Ebert, K. Weber, and S. Koruna, "Integration Platform as a Service," *Business and Information Systems Engineering*, vol. 59, no. 5, pp. 375–379, Oct. 2017, ISSN: 18670202. DOI: 10.1007/s12599-017-0486-0. [Online]. Available: https://zapier.com.

[31] E. Mell and B. Pariseau, *What Is Container Management and Why Is It Important?* Apr. 2021. [Online]. Available: https://searchitoperations.techtarget.com/definition/container-management-software.

[32] M. Imdoukh, I. Ahmad, and M. Alfailakawi, "Optimizing scheduling decisions of container management tool using many-objective genetic algorithm," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, e5536, Mar. 2020, ISSN: 1532-0634. DOI: 10.1002/CPE.5536. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.5536.

[33] Redhat, *What is container orchestration?* Dec. 2019. [Online]. Available: https://www.redhat.com/en/topics/containers/what-is-container-orchestration.

[34] M. A. Rodriguez and R. Buyya, "Container-based cluster orchestration systems: A taxonomy and future directions," *Software: Practice and Experience*, vol. 49, no. 5, pp. 698–719, May 2019, ISSN: 1097-024X. DOI: 10.1002/SPE.2660. [Online]. Available: https://onlinelibrary-wiley-com.ezproxy2.utwente.nl/doi/full/10.1002/spe.2660.

[35] Gartner, *Gartner Magic Quadrant for Enterprise Integration Platform as a Service*, Sep. 2020. [Online]. Available: https://www.gartner.com/en/documents/3990698/magic-quadrant-for-enterprise-integration-platform-as-a-.

[36] Boomi, *Boomi*. [Online]. Available: https://boomi.com/.

[37] Workato, *Workato*. [Online]. Available: https://www.workato.com/integrations/netsuite.

[38] Jitterbit, *Jitterbit*. [Online]. Available: https://www.jitterbit.com.

[39] Ovum, *Ovum Decision Matrix: Selecting a Cloud Platform for Hybrid Integration Vendor, 2019–20*. [Online]. Available: https://omdia.tech.informa.com/OM010250/Ovum-Decision-Matrix-Selecting-a-Cloud-Platform-for-a-Hybrid-Integration-Vendor-201920.

[40] IBM, *IBM App Connect*. [Online]. Available: https://www.ibm.com/nl-en/cloud/app-connect.

[41] Snaplogic, *Snaplogic*. [Online]. Available: https://www.snaplogic.com.

[42] S. Gupta and S. Dhir, "An Enhanced Approach to Use SSL for End To End Security," *(IJCSIT) International Journal of Computer Science and Information Technologies*, vol. 5, no. 2, pp. 1053–1057, 2014, ISSN: 0975-9646. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.658.5398&rep=rep1&type=pdf.

[43] D. Xie, Q. Zhou, X. You, B. Li, and X. Yuan, "A novel energy-efficient cluster formation strategy: From the perspective of cluster members," *IEEE Communications Letters*, vol. 17, no. 11, pp. 2044–2047, Nov. 2013, ISSN: 10897798. DOI: 10.1109/LCOMM.2013.100813.131109.

[44] Z. Jin, D. Y. Kim, J. Cho, and B. Lee, "An Analysis on Optimal Cluster Ratio in Cluster-Based Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 15, no. 11, pp. 6413–6423, Nov. 2015, ISSN: 1530437X. DOI: 10.1109/JSEN.2015.2459374.

[45] D. Weibel, *Architecting Kubernetes clusters — choosing a worker node size*, Jul. 2019. [Online]. Available: https://learnk8s.io/kubernetes-node-size.

[46] CloudFlare, *Why tls 1.3 isn't in browsers yet*. [Online]. Available: https://blog.cloudflare.com/why-tls-1-3-isnt-in-browsers-yet/.

[47] Kubernetes, *Kubernetes*. [Online]. Available: https://kubernetes.io/.

[48] cri-o, *Cri-o, lightweight container runtime for kubernetes*. [Online]. Available: https://cri-o.io/.

[49] Kubernetes, *Rktlet*. [Online]. Available: https://github.com/kubernetes-retired/rktlet.

[50] Kubernetes, *Frakti*. [Online]. Available: https://github.com/kubernetes-retired/frakti.

[51] Kubernetes, *Cluster Networking*. [Online]. Available: https://kubernetes.io/docs/concepts/cluster-administration/networking/.

[52] kubernetes, *Configuring a cgroup driver*, 2021. [Online]. Available: https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/configure-cgroup-driver/.

[53] *Container runtimes*. [Online]. Available: https://kubernetes.io/docs/setup/production-environment/container-runtimes/#cgroup-drivers.

[54] The Kubernetes Authors, *Minikube*. [Online]. Available: https://minikube.sigs.k8s.io/docs/start/.

[55] The Kubernetes Authors, *Kind*. [Online]. Available: https://kind.sigs.k8s.io/.

[56] Amazon, *Amazon Elastic Kubernetes Service (EKS)*. [Online]. Available: https://aws.amazon.com/eks/.

[57] Google, *Google Kubernetes Engine*. [Online]. Available: https://cloud.google.com/kubernetes-engine.

[58] Microsoft, *Azure Kubernetes Service (AKS)*. [Online]. Available: https://docs.microsoft.com/en-us/azure/aks/.

[59] RedHat, *Red Hat OpenShift*. [Online]. Available: https://www.redhat.com/en/technologies/cloud-computing/openshift.

[60] WeaveWorks, *eksctl*. [Online]. Available: https://eksctl.io/.

[61] A. Pereira Ferreira and R. Sinnott, "A performance evaluation of containers running on managed kubernetes services," *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2019-December, pp. 199–208, Dec. 2019. DOI: 10.1109/CLOUDCOM.2019.00038.

[62] Amazon, *Amazon EC2 Instance types*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/.

[63] Google, *Google GKE Pricing*. [Online]. Available: https://cloud.google.com/kubernetes-engine/pricing.

[64] Microsoft, *Azure Pricing Calculator*. [Online]. Available: https://azure.microsoft.com/en-us/pricing/calculator/.

[65] AWS, *AWS Pricing Calculator*. [Online]. Available: https://calculator.aws/.

[66] Spring, *Spring*. [Online]. Available: https://spring.io/.

[67] Spring, *Spring cloud skipper reference guide*. [Online]. Available: https://docs.spring.io/spring-cloud-skipper/docs/2.8.1/reference/htmlsingle//#getting-started-system-requirements.

[68] F. Gutierrez, "Spring Cloud Data Flow Internals," *Spring Cloud Data Flow*, pp. 265–310, 2021. DOI: 10.1007/978-1-4842-1239-4{_}9. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-1239-4_9.

[69] F. Gutierrez, "Spring Cloud Data Flow: Introduction and Installation," *Spring Cloud Data Flow*, pp. 209–262, 2021. DOI: 10.1007/978-1-4842-1239-4{_}8. [Online]. Available: https://link.springer.com/chapter/10.1007/978-1-4842-1239-4_8.

[70] Spring, *Spring cloud data flow*. [Online]. Available: https://spring.io/projects/spring-cloud-dataflow.

[71] E. Truyen, D. Van Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks," *Applied Sciences 2019, Vol. 9, Page 931*, vol. 9, no. 5, p. 931, Mar. 2019, ISSN: 20763417. DOI: 10.3390/APP9050931. [Online]. Available: https://www.mdpi.com/2076-3417/9/5/931/htm%20https://www.mdpi.com/2076-3417/9/5/931.

[72] T. O. Group, *Archimate® 3.1 specification*. [Online]. Available: https://pubs.opengroup.org/architecture/archimate3-doc/.

[73] Datadog, *Cloud monitoring as a service | datadog*. [Online]. Available: https://www.datadoghq.com/.

[74] Prometheus, *Prometheus - monitoring system and time series database*. [Online]. Available: https://prometheus.io/.

[75] Grafana, *Grafana: The open observability platform | grafana labs*. [Online]. Available: https://grafana.com/.

[76] Elastic, *Free and open search: The creators of elasticsearch, elk and kibana | elastic*. [Online]. Available: https://www.elastic.co/.

[77] FluentBit, *Fluent bit*. [Online]. Available: https://fluentbit.io/.

[78] LogPoint, *Award winning siem software - simple, flexible and scalable - logpoint*. [Online]. Available: https://www.logpoint.com/en/?utm_source=capterra&utm_medium=cpc&utm_campaign=logmanagement.

[79] Apache, *Apache log4net – apache log4net: Home - apache log4net*. [Online]. Available: https://logging.apache.org/log4net/.

[80] H. Community, *Helm, the package manager for kubernetes*. [Online]. Available: https://helm.sh/.

[81] Kubernetes, *Kubernetes Ingress Controller*. [Online]. Available: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/.

[82] kubernetes-sigs, *External-dns*. [Online]. Available: https://github.com/kubernetes-sigs/external-dns.

[83] R. Picoreti, A. P. D. Carmo, F. M. D. Queiroz, A. S. Garcia, R. F. Vassallo, and D. Simeonidou, "Multilevel observability in cloud orchestration," pp. 770–775, Oct. 2018. DOI: 10.1109/DASC/PICOM/DATACOM/CYBERSCITEC.2018.00134.

[84] Prometheus, *Exporters and integrations | prometheus*. [Online]. Available: https://prometheus.io/docs/instrumenting/exporters/.

[85] Prometheus, *Prometheus node exporter: Exporter for machine metrics*. [Online]. Available: https://github.com/prometheus/node_exporter.

[86] Kubernetes, *Kubernetes kube state metrics: Add-on agent to generate and expose cluster-level metrics*. [Online]. Available: https://github.com/kubernetes/kube-state-metrics.

[87] Google, *Creating a zonal cluster | kubernetes engine documentation | google cloud*. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/how-to/creating-a-zonal-cluster.

[88] Microsoft, *Quickstart: Deploy an aks cluster by using azure cli - azure kubernetes service | microsoft docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough.

[89] AWS, *Creating an amazon eks cluster - amazon eks*. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html.

[90] G. Cloud, *Filestore*. [Online]. Available: https://cloud.google.com/filestore/.

[91] AWS, *Identity and access management for amazon eks*. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/security-iam.html.

[92] AWS, *Creating a vpc for your amazon eks cluster*. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/creating-a-vpc.html.

[93] T. cert-manager Authors, *Cert-manager io*. [Online]. Available: https://cert-manager.io/.

[94] Nginx, *Configmap - nginx ingress controller*. [Online]. Available: https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/#ssl-ciphers.

[95] Kubecost. [Online]. Available: https://guide.kubecost.com/hc/en-us/articles/4407601807383-Kubernetes-Cost-Allocation.

[96] *Horizontalpodautoscaler walkthrough*. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/.

[97] Kubernetes, *Authenticating*. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/authentication/.

[98] A. C. IO, *Artemis on kubernetes*. [Online]. Available: https://artemiscloud.io/.

[99] RabbitMQ, *Rabbitmq*. [Online]. Available: https://www.rabbitmq.com/.

[100] Apache, *Apache kafka*. [Online]. Available: https://kafka.apache.org/.

[101] Kubernetes, *Pull an image from a private registry*. [Online]. Available: https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/.

[102] Techopedia, *What does portability mean?* [Online]. Available: https://www.techopedia.com/definition/8921/portability.

[103] AWS, *Aws cloudformation*. [Online]. Available: https://aws.amazon.com/cloudformation/.

[104] Google, *Google cloud deployment manager documentation*. [Online]. Available: https://cloud.google.com/deployment-manager/docs.

[105] Microsoft, *What is azure resource manager*. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview#template-deployment.

[106] W. Golden and P. Powell, "Towards a definition of flexibility: In search of the holy grail?" *Omega*, vol. 28, no. 4, pp. 373–384, 2000, ISSN: 0305-0483. DOI: https://doi.org/10.1016/S0305-0483(99)00057-2. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305048399000572.

[107] S. Verreydt, E. H. Beni, E. Truyen, B. Lagaisse, and W. Joosen, "Leveraging kubernetes for adaptive and cost-efficient resource management," in *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*, ser. WOC '19, Davis, CA, USA: Association for Computing Machinery, 2019, pp. 37–42, ISBN: 9781450370332. DOI: 10.1145/3366615.3368357. [Online]. Available: https://doi.org/10.1145/3366615.3368357.

Appendices

Appendix A

Network Policy

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
  namespace: customer-0001
spec:
  policyTypes:
  - Ingress
  - Egress
  podSelector: {}
  ingress: []
  egress: []
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-exposed
  namespace: customer-0001
spec:
  podSelector:
    matchLabels:
      role: external
  policyTypes:
    - Ingress
    - Egress
  ingress:
    # External: In from all over port 443,80
    - from:
        - ipBlock:
            cidr: 0.0.0.0/0
      ports:
        - port: 443
        - port: 80

    # Cluster: In from all with label role=ingress over port 80,443
    - from:
        - namespaceSelector: {}
          podSelector:
            matchLabels:
              role: ingress
      ports:
        - port: 80
```

```
        - port: 443

    # Namespace: In from all
    - from:
        - podSelector: {}

  egress:
    # External: Out to all over port 80,443
    - to:
        - ipBlock:
            cidr: 0.0.0.0/0
      ports:
        - port: 443
        - port: 80

    # Cluster: Out to none

    # Namespace: Out to all
    - to:
        - podSelector: {}

    # DNS: Out to DNS
    - to:
      - namespaceSelector: {}
        podSelector:
          matchLabels:
            k8s-app: kube-dns
      ports:
        - port: 53
          protocol: UDP
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-all
  namespace: customer-0001
spec:
  policyTypes:
  - Ingress
  - Egress
  podSelector: {}
  ingress: []
  egress: []
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-but-namespace
  namespace: not-prototype
spec:
  podSelector:
    matchLabels:
      role: internal
  policyTypes:
    - Ingress
    - Egress
```

```
ingress:
  # External: In from none

  # Cluster: In from none

  # Namespace: In from all
  - from:
      - podSelector: {}

egress:
  # External: Out to all over port 80,443
  - to:
      - ipBlock:
          cidr: 0.0.0.0/0
    ports:
      - port: 443
      - port: 80

  # Cluster: Out to none

  # Namespace: Out to all
  - to:
      - podSelector: {}

  # DNS: Out to DNS
  - to:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
      - port: 53
        protocol: UDP
```

Appendix B

Example Integration Deployment

```
# https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
apiVersion: v1
kind: Namespace
metadata:
  name: customer-0001
---
# See https://kubernetes.io/docs/concepts/policy/resource-quotas/
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: customer-0001
spec:
  hard:
    cpu: "<CPU>"
    memory: <MEMORY>Mi
    pods: "<PODS>"
    replicationcontrollers: "<REPLICATION_CONTROLELRS>"
    resourcequotas: "<RESOURCE_QUOTAS>"
    services: "<SERVICES>"
---
# See https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-customer-0001
  namespace: customer-0001
automountServiceAccountToken: false
---
# See https://kubernetes.io/docs/reference/access-authn-authz/rbac/
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: role-podrunner
  namespace: customer-0001
rules:
  - apiGroups:
      - ''
    resources:
      - secrets
    verbs:
      - get
```

```
  - apiGroups:
      - ''
    resources:
      - services
    verbs:
      - get
---
# See https://kubernetes.io/docs/reference/access-authn-authz/rbac/
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: role-binding-podrunner-sa
  namespace: customer-0001
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: role-podrunner
subjects:
  - kind: ServiceAccount
    name: sa-customer-0001
    namespace: customer-0001
---
# See https://kubernetes.io/docs/concepts/services-networking/service/
apiVersion: v1
kind: Service
metadata:
  name: service-alpha
  namespace: customer-0001
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
    - name: https
      port: 443
      protocol: TCP
      targetPort: 443
  selector:
    app.kubernetes.io/name: pod-alpha
---
# https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    role: integration
  name: deployment-alpha
  namespace: customer-0001
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: pod-alpha
  revisionHistoryLimit: 10
  minReadySeconds: 60
```

```
  template:
    metadata:
      labels:
        app.kubernetes.io/name: pod-alpha
    spec:
      containers:
        - name: pod-alpha
          image: <IMAGE>
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
            - name: https
              containerPort: 443
              protocol: TCP
          resources:
            requests:
              cpu: 90m
              memory: 90Mi
      imagePullSecrets:
          - name: <IMAGE_PULL_SECRET>
      nodeSelector:
        kubernetes.io/os: linux
      serviceAccountName: sa-customer-0001
      terminationGracePeriodSeconds: 30
---
# See https://kubernetes.io/docs/concepts/services-networking/ingress/
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-alpha
  namespace: customer-0001
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-acceptance
    nginx.ingress.kubernetes.io/rewrite-target: /\$1
\# Uncomment for mTLS
\#    nginx.ingress.kubernetes.io/auth-tls-verify-client: "on"
\#    nginx.ingress.kubernetes.io/auth-tls-secret: "<TLS_SECRET>"
\#    nginx.ingress.kubernetes.io/auth-tls-verify-depth: "1"
\#    nginx.ingress.kubernetes.io/auth-tls-pass-certificate-to-upstream: "false"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
      - customer-0001.iops.com
    secretName: customer0001iops
  rules:
  - host: customer-0001.iops.com
    http:
      paths:
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: service-alpha
```

```
port:
  number: 443
```

Appendix C

AWS Cloud Formation SQL Deployment

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Create a DB subnet group and MYSQL Database"

Parameters:
  VPC:
    Type: String
    Description: The VPC to create the cluster
    Default: vpc-ID

  PrivateSubnet01:
    Type: String
    Description: The subnet for the DB cluster
    Default: subnet-ID

  PrivateSubnet02:
    Type: String
    Description: The subnet for the DB cluster
    Default: subnet-ID

  MasterUsername:
    Type: String
    Description: The username for our database.

  MasterUserPassword:
    Type: String
    Description: The password for the database.
    "NoEcho": true

  ParameterGroup:
    Type: String
    Description: The name of the database parameter group created.

Resources:
    EC2SecurityGroup:
        Type: "AWS::EC2::SecurityGroup"
```

```
    Properties:
        GroupDescription: "Database instances security group"
        VpcId: !Ref VPC
        SecurityGroupIngress:
          -
            CidrIp: "*.*.*.*/32"
            FromPort: 3306
            IpProtocol: "tcp"
            ToPort: 3306
        SecurityGroupEgress:
          -
            CidrIp: "0.0.0.0/0"
            IpProtocol: "-1"

RDSDBSubnetGroup:
    Type: "AWS::RDS::DBSubnetGroup"
    Properties:
        DBSubnetGroupDescription: "Subnet Group for mySQL database"
        DBSubnetGroupName: !Sub "\${AWS::Region}-aws-dxl-database-subnet-group"
        SubnetIds:
          - !Ref PrivateSubnet01
          - !Ref PrivateSubnet02
        Tags:
          - Key: Name
            Value: eu-central-1-test-db-cluster
          - Key: createdBy
            Value: Maureen Barasa
          - Key: Project
            Value: test-blog
          - Key: Environment
            Value: test

RDSDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
        DBInstanceIdentifier: aws-dxl-database-1
        AllocatedStorage: 100
        DBInstanceClass: db.m5.large
        Engine: "MYSQL"
        MasterUsername: !Ref MasterUsername
        MasterUserPassword: !Ref MasterUserPassword
        BackupRetentionPeriod: 7
        MultiAZ: true
        EngineVersion: 8.0.20
        AutoMinorVersionUpgrade: true
        Iops: 1000
        PubliclyAccessible: false
        StorageType: io1
        Port: 3306
        StorageEncrypted: true
```

```
            CopyTagsToSnapshot: true
            MonitoringInterval: 60
            EnableIAMDatabaseAuthentication: false
            EnablePerformanceInsights: true
            PerformanceInsightsRetentionPeriod: 7
            DeletionProtection: true
            DBSubnetGroupName: !Ref RDSDBSubnetGroup
            VPCSecurityGroups:
              - !Ref EC2SecurityGroup
            MaxAllocatedStorage: 1000
            DBParameterGroupName: !Ref ParameterGroup
            MonitoringRoleArn: !Sub "arn:aws:iam::\${AWS::AccountId}:role
                                    /rds-monitoring-role"
            Tags:
              - Key: Name
                Value: aws-dxl-database-1
              - Key: createdBy
                Value: Maureen Barasa
              - Key: Project
                Value: test-blog
              - Key: Environment
                Value: test

Outputs:
   Cluster:
    Description: The DB Cluster Name
    Value: !Ref RDSDBInstance

   SubnetGroup:
    Description: The db subnet group name
    Value: !Ref RDSDBSubnetGroup
```

Appendix D

Azure Arm template SQL Deployment


```
{
  "\$schema": "https://schema.management.azure.com/schemas/2019-04-01
  /deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.4.1008.15138",
      "templateHash": "6360281189485765882"
    }
  },
  "parameters": {
    "serverName": {
      "type": "string",
      "metadata": {
        "description": "Server Name for Azure database for MySQL"
      }
    },
    "administratorLogin": {
      "type": "string",
      "minLength": 1,
      "metadata": {
        "description": "Database administrator login name"
      }
    },
    "administratorLoginPassword": {
      "type": "secureString",
      "minLength": 8,
      "metadata": {
        "description": "Database administrator password"
      }
    },
    "skuCapacity": {
      "type": "int",
      "defaultValue": 2,
```

```
      "metadata": {
        "description": "Azure database for MySQL compute capacity"
      }
    },
    "skuName": {
      "type": "string",
      "defaultValue": "GP_Gen5_2",
      "metadata": {
        "description": "Azure database for MySQL sku name "
      }
    },
    "SkuSizeMB": {
      "type": "int",
      "defaultValue": 5120,
      "metadata": {
        "description": "Azure database for MySQL Sku Size "
      }
    },
    "SkuTier": {
      "type": "string",
      "defaultValue": "GeneralPurpose",
      "allowedValues": [
        "Basic",
        "GeneralPurpose",
        "MemoryOptimized"
      ],
      "metadata": {
        "description": "Azure database for MySQL pricing tier"
      }
    },
    "skuFamily": {
      "type": "string",
      "defaultValue": "Gen5",
      "metadata": {
        "description": "Azure database for MySQL sku family"
      }
    },
    "mysqlVersion": {
      "type": "string",
      "defaultValue": "8.0",
      "allowedValues": [
        "5.6",
        "5.7",
        "8.0"
      ],
      "metadata": {
        "description": "MySQL version"
      }
    },
    "location": {
```

```
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    },
    "backupRetentionDays": {
      "type": "int",
      "defaultValue": 7,
      "metadata": {
        "description": "MySQL Server backup retention days"
      }
    },
    "geoRedundantBackup": {
      "type": "string",
      "defaultValue": "Disabled",
      "metadata": {
        "description": "Geo-Redundant Backup setting"
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "defaultValue": "azure_mysql_vnet",
      "metadata": {
        "description": "Virtual Network Name"
      }
    },
    "subnetName": {
      "type": "string",
      "defaultValue": "azure_mysql_subnet",
      "metadata": {
        "description": "Subnet Name"
      }
    },
    "virtualNetworkRuleName": {
      "type": "string",
      "defaultValue": "AllowSubnet",
      "metadata": {
        "description": "Virtual Network RuleName"
      }
    },
    "vnetAddressPrefix": {
      "type": "string",
      "defaultValue": "10.0.0.0/16",
      "metadata": {
        "description": "Virtual Network Address Prefix"
      }
    },
    "subnetPrefix": {
      "type": "string",
```

```
      "defaultValue": "10.0.0.0/16",
      "metadata": {
        "description": "Subnet Address Prefix"
      }
    }
  },
  "functions": [],
  "variables": {
    "firewallrules": [
      {
        "Name": "rule1",
        "StartIpAddress": "0.0.0.0",
        "EndIpAddress": "255.255.255.255"
      },
      {
        "Name": "rule2",
        "StartIpAddress": "0.0.0.0",
        "EndIpAddress": "255.255.255.255"
      }
    ]
  },
  "resources": [
    {
      "type": "Microsoft.DBforMySQL/servers/virtualNetworkRules",
      "apiVersion": "2017-12-01",
      "name": "[format('{0}/{1}', parameters('serverName'),
      parameters('virtualNetworkRuleName'))]",
      "properties": {
        "virtualNetworkSubnetId": "[resourceId('Microsoft.Network/
        virtualNetworks/subnets', parameters('virtualNetworkName'),
        parameters('subnetName'))]",
        "ignoreMissingVnetServiceEndpoint": true
      },
      "dependsOn": [
        "[resourceId('Microsoft.DBforMySQL/servers', parameters('serverName'))]",
        "[resourceId('Microsoft.Network/virtualNetworks/subnets',
        parameters('virtualNetworkName'), parameters('subnetName'))]"
      ]
    },
    {
      "type": "Microsoft.Network/virtualNetworks",
      "apiVersion": "2020-06-01",
      "name": "[parameters('virtualNetworkName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[parameters('vnetAddressPrefix')]"
          ]
        }
```

```
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks/subnets",
      "apiVersion": "2020-06-01",
      "name": "[format('{0}/{1}', parameters('virtualNetworkName'),
       parameters('subnetName'))]",
      "properties": {
        "addressPrefix": "[parameters('subnetPrefix')]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/virtualNetworks',
         parameters('virtualNetworkName'))]"
      ]
    },
    {
      "type": "Microsoft.DBforMySQL/servers",
      "apiVersion": "2017-12-01",
      "name": "[parameters('serverName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('skuName')]",
        "tier": "[parameters('SkuTier')]",
        "capacity": "[parameters('skuCapacity')]",
        "size": "[format('{0}', parameters('SkuSizeMB'))]",
        "family": "[parameters('skuFamily')]"
      },
      "properties": {
        "createMode": "Default",
        "version": "[parameters('mysqlVersion')]",
        "administratorLogin": "[parameters('administratorLogin')]",
        "administratorLoginPassword": "[parameters('administratorLoginPassword')]",
        "storageProfile": {
          "storageMB": "[parameters('SkuSizeMB')]",
          "backupRetentionDays": "[parameters('backupRetentionDays')]",
          "geoRedundantBackup": "[parameters('geoRedundantBackup')]"
        }
      }
    },
    {
      "copy": {
        "name": "firewallRules",
        "count": "[length(variables('firewallrules'))]",
        "mode": "serial",
        "batchSize": 1
      },
      "type": "Microsoft.DBforMySQL/servers/firewallRules",
      "apiVersion": "2017-12-01",
      "name": "[format('{0}/{1}', parameters('serverName'),
      variables('firewallrules')[copyIndex()].Name)]",
```

```
      "properties": {
        "startIpAddress": "[variables('firewallrules')[copyIndex()].StartIpAddress]",
        "endIpAddress": "[variables('firewallrules')[copyIndex()].EndIpAddress]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.DBforMySQL/servers', parameters('serverName'))]"
      ]
    }
  ]
}
```

Appendix E

Google deployment mananger SQL Deployment

```
{% set deployment_name = env['deployment']  %}
{% set instance_name = deployment_name + '-instance'  %}
{% set database_name = deployment_name + '-db'  %}

resources:
- name: {{ instance_name }}
  type: gcp-types/sqladmin-v1beta4:instances
  properties:
    region: {{ properties['region'] }}
    settings:
      tier: {{ properties['tier'] }}
      backupConfiguration:
        binaryLogEnabled: true
        enabled: true

- name: {{ database_name }}
  type: gcp-types/sqladmin-v1beta4:databases
  properties:
    name: {{ database_name }}
    instance: $(ref.{{ instance_name }}.name)
    charset: utf8

- name: insert-user-root
  action: gcp-types/sqladmin-v1beta4:sql.users.insert
  metadata:
    runtimePolicy:
    - CREATE
    dependsOn:
    - {{ database_name }}
  properties:
    project: {{ env['project'] }}
    instance: $(ref.{{ env['deployment'] }}-instance.name)
    name: testuser
    host: "%"
    password: testpass
```

Appendix F

Kubernetes YAML SQL Deployment

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    app: mysql
  clusterIP: None
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
          # Use secret in real usage
        - name: MYSQL_ROOT_PASSWORD
          value: password
        ports:
        - containerPort: 3306
```

```
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-pv-claim
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```