

Author  
**Tanja Fraundorfer, BSc**

Submission  
**Institute of Networks and  
Security**

Thesis Supervisor  
**Assoc. Prof. Mag. Dipl.-Ing.  
Dr. Michael Sonntag**

03 2021

# **CLOUD INTERCHANGEABILITY**



Master's Thesis  
to confer the academic degree of  
Diplom-Ingenieurin  
in the Master's Program  
Computer Science

## **Abstract**

Cloud computing becomes more and more relevant in everyday business of different kinds of companies. Therefore, migration of system and services to the cloud is also an important topic. But not just the migration to the cloud, the transfer of systems between clouds is also interesting. A reason for changing the cloud provider could be that contracts or offerings of the cloud providers changed. Thus, this thesis is about the step of migrating systems to, from and between clouds. For this, three cloud platforms are examined: GCP (Google Cloud Platform), AWS (Amazon Web Services) and Azure (Microsoft). Different systems will be imported to those platforms, exported from them, and moved between them. There are two options for moving systems between cloud platforms: direct and indirect. Indirect means exporting the system from the one cloud platform, saving it on the local system and then importing it to the other cloud platform. The test systems consist of a database, a virtual machine, a docker container and a storage system. As it turned out, some systems are easier to move to a certain platform than to others, e.g. for virtual machines all cloud providers set the strictest requirements. GCP offers the most tools and various options how to import systems from a local system and from other cloud platforms. Of course there are some differences in the offered tools and services, nevertheless the principles are more or less the same in all cloud platforms.

## Kurzfassung

Cloud Computing wird immer relevanter für verschiedenste Firmen und deren Alltag, deswegen ist die Migration von Systemen in die Cloud ein wichtiges Thema. Aber nicht nur die Migration in die Cloud, auch die Migration zwischen zwei Cloud-Plattformen ist ein interessantes Thema. Ein Grund um den Cloud-Anbieter zu wechseln sind z.B. Änderung der Angebote oder Vertragsbedingungen. Deswegen beschäftigt sich diese Masterarbeit mit der Migration von Systemen zu, von und zwischen Cloud-Plattformen. Es werden dafür drei Cloud-Plattformen analysiert: GCP (Google Cloud Plattform), AWS (Amazon Web Services) und Azure (Microsoft). In diese Cloud-Plattformen werden Systeme importiert, von ihnen exportiert und zwischen ihnen migriert. Es gibt zwei Möglichkeiten Systeme zwischen den Cloud-Plattformen zu migrieren: direkt und indirekt. Bei der indirekten Variante wird das Testsystem zuerst von der einen Cloud-Plattform exportiert, lokal gespeichert und dann in die andere Cloud-Plattform importiert. Das Testsystem besteht aus einer Datenbank, einer virtuellen Maschine, einem Docker Container und einem Speichersystem. Es stellte sich heraus, dass manche Teile des Testsystems leichter in manche Cloud-Plattformen importiert werden können als andere. Alle Cloud-Anbieter stellen die strengsten Anforderungen für den Import von virtuellen Maschinen. GCP stellt die meisten Tools zur Verfügung, um Systeme zu importieren. Auch wenn einige Unterschiede in bereitgestellten Tools und Diensten bestehen, sind sich die Cloud-Plattformen im Großen und Ganzen sehr ähnlich.

# Cloud interchangeability

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Scope . . . . .	3
<b>2</b>	<b>Theoretical Background</b>	<b>4</b>
2.1	Definition of Cloud Computing . . . . .	4
2.2	Characteristics of cloud computing . . . . .	5
2.3	Service Models . . . . .	7
2.3.1	Infrastructure as a Service (IaaS) . . . . .	8
2.3.2	Platform as a Service (PaaS) . . . . .	9
2.3.3	Software as a Service (SaaS) . . . . .	9
2.3.4	Other service models . . . . .	10
2.4	Deployment Models . . . . .	10
2.4.1	Private cloud . . . . .	10
2.4.2	Public cloud . . . . .	11
2.4.3	Community cloud . . . . .	11
2.4.4	Hybrid cloud . . . . .	12
2.5	Cloud Migration . . . . .	12
2.5.1	Migration process . . . . .	13
2.5.2	Migration strategies . . . . .	14
2.5.3	Mobility of services in the cloud . . . . .	16
2.5.4	Automation of the migration process . . . . .	16
2.6	The cloud platforms investigated . . . . .	18
2.6.1	Google Cloud Platform (GCP) . . . . .	18
2.6.2	Amazon Web Services (AWS) . . . . .	19
2.6.3	Microsoft Azure . . . . .	20
<b>3</b>	<b>Approach</b>	<b>22</b>
3.1	Tests to perform . . . . .	22
3.1.1	Migrating a local system to a cloud platform . . . . .	22
3.1.2	Migrating a system from a cloud platform to a local system . . . . .	23
3.1.3	Migration between cloud platforms . . . . .	23
3.2	Test Systems . . . . .	24
3.2.1	Databases . . . . .	24
3.2.2	Virtual Machines . . . . .	25
3.2.3	Docker . . . . .	26
3.2.4	Storage . . . . .	26
3.3	Assessment criteria . . . . .	26
3.4	Migration approach . . . . .	27

<b>4</b>	<b>Migrating a local system to a cloud platform</b>	<b>28</b>
4.1	Databases . . . . .	28
4.1.1	To GCP . . . . .	28
4.1.2	To AWS . . . . .	30
4.1.3	To Azure . . . . .	32
4.2	Virtual Machines . . . . .	33
4.2.1	To GCP . . . . .	33
4.2.2	To AWS . . . . .	34
4.2.3	To Azure . . . . .	36
4.3	Docker . . . . .	37
4.3.1	To GCP . . . . .	37
4.3.2	To AWS . . . . .	38
4.3.3	To Azure . . . . .	39
4.4	Storage . . . . .	40
4.4.1	To GCP . . . . .	40
4.4.2	To AWS . . . . .	40
4.4.3	To Azure . . . . .	41
<b>5</b>	<b>Migrating a system from a cloud platform to a local system</b>	<b>42</b>
5.1	Databases . . . . .	42
5.1.1	From GCP . . . . .	42
5.1.2	From AWS . . . . .	44
5.1.3	From Azure . . . . .	44
5.2	Virtual Machines . . . . .	45
5.2.1	From GCP . . . . .	45
5.2.2	From AWS . . . . .	46
5.2.3	From Azure . . . . .	47
5.3	Docker . . . . .	47
5.3.1	From GCP . . . . .	48
5.3.2	From AWS . . . . .	48
5.3.3	From Azure . . . . .	49
5.4	Storage . . . . .	49
5.4.1	From GCP . . . . .	49
5.4.2	From AWS . . . . .	49
5.4.3	From Azure . . . . .	49
<b>6</b>	<b>Migration between cloud platforms</b>	<b>50</b>
6.1	Direct – from cloud platform to cloud platform . . . . .	50
6.1.1	Databases . . . . .	50
6.1.2	Virtual Machines . . . . .	51
6.1.3	Docker . . . . .	52
6.1.4	Storage . . . . .	54
6.2	Indirect – with the help of a local system . . . . .	55
6.2.1	Databases . . . . .	55
6.2.2	Virtual Machines . . . . .	56
6.2.3	Docker . . . . .	57
6.2.4	Storage . . . . .	57
6.3	Comparison . . . . .	57

<b>7 Summary and Conclusion</b>	<b>59</b>
7.1 Results . . . . .	59
7.2 Implications and future directions . . . . .	61
<b>References</b>	<b>62</b>
<b>List of Figures</b>	<b>65</b>
<b>List of Tables</b>	<b>66</b>

# 1 Introduction

Clouds - people who are not familiar with computer science usually think about the sky. Maybe a blue one with some nice white clouds in it or a cloudy and cold day when everyone prefers to sit inside instead of going out. Contrary to this, computer science people think about a technology which offers a wide range of opportunities, services and business models. The latter is what this thesis is about. The technology which changed a lot in computer science, like clouds outside can change the weather quickly. People or companies can now easily rent infrastructure or applications and don't need to buy the hardware themselves. Still instead of hardware, people or companies now need the Internet to connect to the cloud and this is one of the few prerequisites for using cloud computing. There is not much more technical equipment needed. The internet is also the area where the cloud symbol has already been known for a long time. It is used to show a network where not all the connections are known exactly. In [38] it is said that engineers started to use this symbol for cloud computing for the same reasons as it was used in the internet - the customers don't know what is hidden behind the cloud. They just get the service, which can be applications, infrastructure or computing power. Although it is not known where exactly the physical hardware is located. It can be based in any computer center the cloud provider owns.

Not having dedicated hardware has another advantage. Customers can share the infrastructure and thanks to virtualization, they do not know that they are working on the same physical hardware. This, as everything, has advantages and disadvantages. The infrastructure can usually be used more efficiently and therefore it is cheaper to maintain. Yet also the customer has benefits: within certain limits the needed computing power is granted to them as they need it. This is especially useful if they have peak times where certain resources or computing power is used for a short time and not a constant utilization. So it is a win-win situation for provider and customer. Moreover the infrastructure is administered centrally by the provider and so energy could be saved with the right management. Considering that unused parts of the infrastructure don't need to be cooled or supplied with electricity, this also helps the environment.

## 1.1 Motivation

Cloud computing is getting more and more important in recent years, and it does not just open up new business opportunities but also ways to save money and use resources more efficiently. A reason why this is the case, is that clouds are not limited to one business sector. All sectors can use cloud computing and take advantage of it. This gives the cloud a very broad set of possible areas where it can be used. Also it can still be further developed and adapted to the needs of the users. That's why it is critical to pay attention to cloud systems, how they can be used and most important how a system can actually be set up in the cloud.

Cloud computing offers a lot of advantages in comparison to traditional computing systems. It is more flexible when it comes to meeting the actual needs of the customer with regards to the available resources and scalability. Because it is based on a pay-per-use model the customer just pays for the services actually used. These are only some reasons for transferring systems to the cloud. Of course then the question arises how easy this transfer or migration is and what happens if it turns out that it wasn't the best solution

for the company. Is it possible and with how much effort to revert to a local system? Maybe also another cloud provider would have been a better choice. On the grounds that the offered services are more suitable to the customers situation, there are better SLA (Service level agreements) granted or simply because the other provider is cheaper. Normally people start comparing providers and their offerings when they first want to move something to the cloud but also when the current contract should be renewed. All those reasons given for using cloud computing or changing providers, the central questions are the following: How much work is it? Is there something that can't be implemented in the cloud environment? Which resources and experts are needed for the migration? And of course, how much will it cost?

While providers generally try to gain customers, they do not want to lose them. So they would not be too interested in cooperating with others to make the movement between the clouds easy. Importing and setting up new systems in their cloud should be their daily business but exporting data often means losing business. The compatibility problem arises from the history. When cloud computing started, everyone implemented their own system and started offering services. No one would use an standardized approach, because there was none and so the different systems were created. Also because customers have different needs and different providers have different target groups it is just logical that the functionality in particular clouds varies. When the functionality is not the same, certainly the underlying technology needs to adapt. Moreover some providers maybe want to offer the best technology, while others aim for customers which desire simple and cheap systems.

An interesting project is GAIA-X which is carried out by the European Union. On the one hand, they aim to create a competitive offer to the American and Asian cloud platforms and on the other hand, they want to have a data service, which is protected by European data protection laws. Also, it is meant to be one big cloud ecosystem and therefore connect various cloud platforms of different providers. In this way a common standard could be established between European cloud providers.

While there are different service models in cloud computing, a big part of it is SaaS (Software as a Service), where the customer just uses the application which is offered by the cloud provider. This doesn't only save hardware costs for the hardware on which the application needs to run but also for a central data server and the development costs for the software, if it would be created by the company. In cloud computing the company doesn't buy a program and then use it locally but rather kind of rents the program on a pay-per-use base. In [37] salesforce.com is given as an example. When just the application is used then the export of the data could be problematic, also finding or developing another application which works with exactly the same data could be quite time and resource intensive.

When migrating a system, no matter for what reasons, it is always a good time to think about what one expects from the systems that he or she is working with. Which features and functionalities are needed and should be kept, which should be changed and, are there any new ones needed? Then one also has to think about the current source and how the data can be extracted from it. Often there are other software versions needed considering that cloud providers most of the time just support certain versions of a program. Then one has to consider that other applications may need to be compatible with the newer software too. Depending on the kind of the upgrade or change in software the employees need to be trained to work with the newer version or the other software. Moreover if not



everything can be moved to the cloud at the same time, a plan needs to be made about how the new and the old system can work together. Maybe it is even necessary to run both systems for some time in parallel and then consistency between the systems has to be ensured. An alternative would be, that the desired option is to have some sort of mix, where not just for the migration but for the general use some data and applications are hosted in the cloud and the more delicate data is stored on the local infrastructure.

Also, one has to have a good look at the used systems and maybe even improvements to the work-flows can be made and thus work could be made more smooth and efficient. At first this maybe seems like an exhausting and really big task, because one has to think a lot about the migration process as well as who is the most suitable cloud provider. Still, when all the work is done, when the new system is up and running and costs can be saved it was hopefully totally worth the effort and one can enjoy the benefits of cloud computing.

## 1.2 Scope

In this thesis the three biggest cloud providers and their products are examined. Those are Microsoft Azure, Google Cloud Platform and Amazon Web Services. The main question is the migration between the clouds and a local system but also migrations between the cloud providers. So for each cloud there are the following scenarios: migrating a local system to the cloud, exporting a system from the cloud and setting it up locally again, and migrating a system between the cloud platforms. The last scenario can happen directly or indirectly depending on the options that cloud providers offer. Indirect means with an extra step over a local setup, which may include making some changes to the system to be able to import it again to another cloud. For these experiments a test system is created. This test system consist of a database, virtual machines, docker containers and a structure to test the Blob storage possibilities in the clouds. Specific applications, like mail or messaging services, are not tested here.

## 2 Theoretical Background

Clouds are constantly changing and evolving, which is why the definition and theoretical background also changes. This is not surprising because cloud systems are in use and therefore try to suit the needs of the users as well as the different providers continue developing this technology to keep ahead in business. This does not always happen with a common standard, because every provider tries to have the best and most innovative product. Nevertheless there are some definitions, models and characteristics which were set in the past few years and which are going to be discussed in the first part of this section.

The second part of this chapter is dedicated to the cloud providers and their platforms in general. The three biggest ones on the market are Google with Google cloud platform (GCP), Microsoft with Azure and Amazon with Amazon web services (AWS). Those are the platforms on which the experiments will take place later. This section introduces them and gives a first impression of their common aspects but also their differences. Next a short discussion of the migration process and if or how it can be automated follows. In this section some tools are introduced which are already used by the different cloud providers.

### 2.1 Definition of Cloud Computing

A very simplified but appropriate definition of cloud computing was given in [32]: “Cloud computing in simple terms means storing and accessing data and programs over the Internet instead of our computer’s hard drive.” Of course cloud computing is not restricted to storing data and using programs but can also include the usage of infrastructure. In [33] the cloud is defined as “a service or group of services.” It is also said that the cloud definitions changed over time and are still going to change, because services are always adapted to the needs of the customer.

In [38] cloud computing is defined as a form of internet computing where the customer can rent certain services from the cloud provider. Servers, storage, platforms and applications are a few examples for these services. More than that, pay-per-use billing is part of the definition here.

According to NIST [30] “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [...] that can be rapidly provisioned and released with minimal management effort or service provider interaction.” This means cloud computing needs to be easily scalable, also the scaling needs to happen fast and as far as possible it should be automated. Thus no or hardly any interaction with the administrator of the provider is necessary. Besides the cloud should be reachable at all times. Moreover NIST [30] defines five essential characteristics for clouds, three service models and four deployment models. These characteristics and models are going to be discussed in the chapters 2.2, 2.3 and 2.4.

As can be seen these definitions are not completely the same but still build upon a common ground. The clearest and thought-out definition comes from NIST and this is also probably the one that one should go with when working in the cloud environment.

## 2.2 Characteristics of cloud computing

NIST [30] defines five essential characteristics for cloud computing. These five characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. They describe how a cloud is supposed to behave and can be seen in figure 1. Without these characteristics a solution is not a real cloud solution. But because local solutions often don't have those characteristics, the migrated systems have to be adapted to run in the cloud. To enable resource pooling, for example, the systems have to be virtualized.

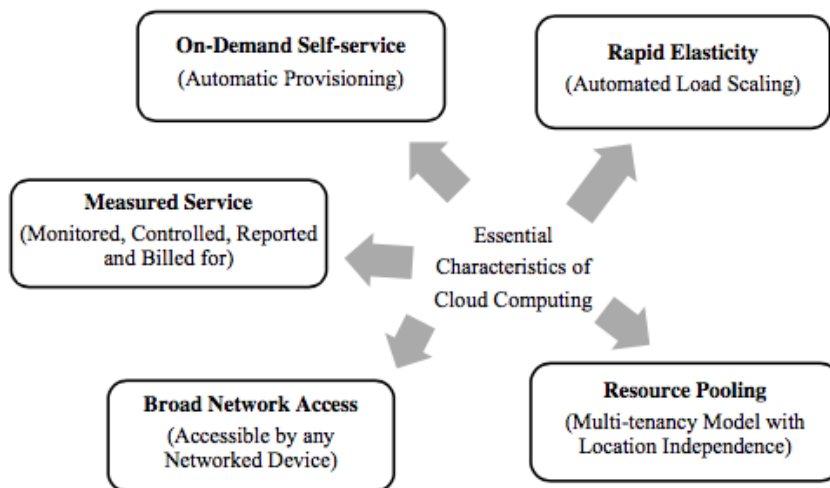


Figure 1: The five essential characteristics of cloud computing [21].

### On-Demand Self-service

One key point in cloud environments is the on-demand self-service. Here an automated process is introduced, so that the customer can gain access to the application without human interaction. This means no administrator or other person has to give their approval or has to perform any system changes. Of course not all rights can be granted in that way, because one has to consider compliance issues. This is the case when separation of duty is needed and therefore a user needs a certain approval to perform a task. This is often required if a law like the Sarbanes- Oxley Act (SOX) applies to the business field. Nevertheless there is less administration needed than without self-service [33]. Moreover the pay-per use model guarantees that the customer just pays for the resources, which have been used. When using on-demand self-service, the user just needs access to the online interface. The idea of this setup is shown in figure 2. One can see that the customer just needs a device, like a PC, mobile device or laptop, to connect with the user interface. Then the customer can connect to the provider's resources over the internet.

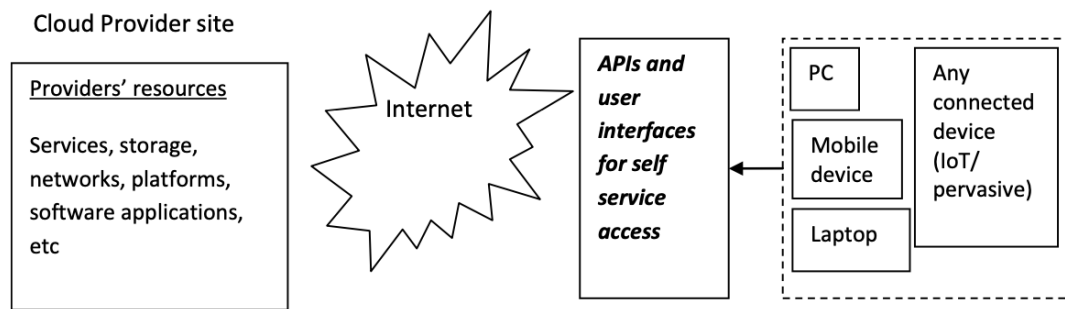


Figure 2: “Self-service based access” [37].

### Rapid Elasticity

In common systems, if more capacity is needed, usually new hardware needs to be added. This can take some time and is expensive. In the cloud environment adding capacity happens fast and mostly automated. This is possible because the infrastructure is normally already there and just has to be assigned. In [37] is pointed out that the main goal of this characteristic is that the time gap between the changed demand and the resource availability is as small as possible. This change can be a needed increase or decrease of resources. The scalability is not just important to handle bursts (short times when the customer needs more capacity) but also helps the provider saving costs, because not needed resources don't need to be cooled or consume power. The elasticity is often implemented with triggers, which are set off if a certain threshold is met [33].

### Broad Network Access

According to [33] there are three main points for this characteristic. The first point is that the user should not be expected to have an extraordinary internet connection. Basic internet access needs to be enough for using the cloud services. Therefore they should need just a reasonable amount of bandwidth. The second point is that no client or just a lightweight client should be required. This improves the usability, due to the user not having to download anything first. Last but not least, the third point is that the cloud should be accessible from as many different devices as possible. Nowadays people work with all kinds of devices which meet their needs best and so it is just logical that the cloud should be usable with all of them.

### Measured service

Measuring the service is not just needed for handling scalability but also for correct billing. This is necessary for all per-per-use or pay-as-you-go contracts and should happen automatically. There are different metrics which can be used. In [33] various examples are mentioned such as used time, bandwidth, storage and data. Of course the measuring and billing should be transparent for the customer and there should just be the resources and times charges which were actually consumed [37]. This has the advantage that if a service isn't used for a certain time nothing is charged and therefore this is an opportunity for the customer to save money. But the services not only have to be measured because of the billing but also for optimizing the usage and predicting, which resources may be needed in the future [21].

## Resource Pooling

Resource pooling and multi-tenancy are mechanisms to save costs. Usually they are realized with the help of virtualization and so the physical resources can be used in the best possible way [33]. The assignment of the resources usually happens automatically and depending on the current needs of the customer. These needs often change with time. Moreover the resource assignment is location independent [21]. This means that the user has no knowledge about where the resources, which are currently used, are located. This is not always good, because for certain businesses it is needed, due to legal reasons, to have their data stored within certain countries or continents, e.g. within the European Union. The basic scheme of resource pooling can be seen in figure 3. There, three tenants share the resource pool of the distributed infrastructure but they are separated due to virtualization.

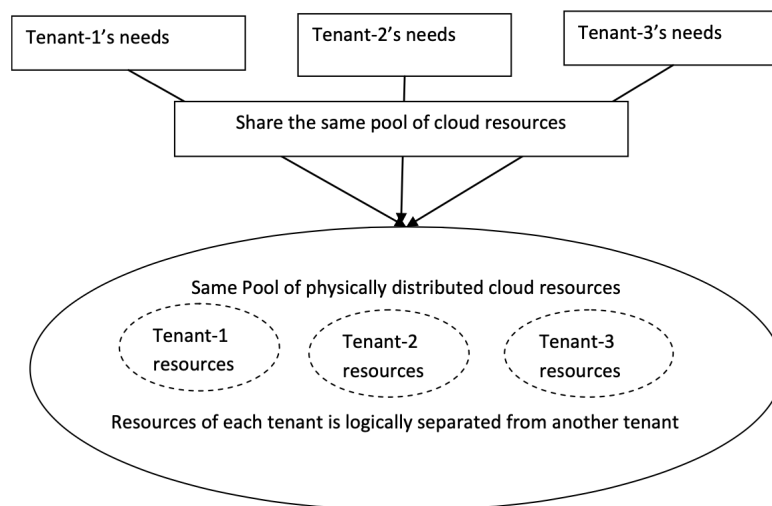


Figure 3: “Resource Pooling and multi-tenancy” [37].

## 2.3 Service Models

The three most common and also by NIST [30] defined service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). As the name says the three models differ in the offered service but usually they are all offered as a pay-per-use model. While some customers maybe just need to run software others want to manipulate the data on a deeper level. In SaaS the customer just uses the provided software or applications but nothing else. IaaS on the contrary provides lower level access and the customer can manipulate resources on the operating system level [22]. In this chapter the service models are described, the relevance for the migration process can be found in chapter 2.5.2. Figure 4 shows an overview of which levels can be manipulated or need to be managed by whom (customer or service provider) in different service models.

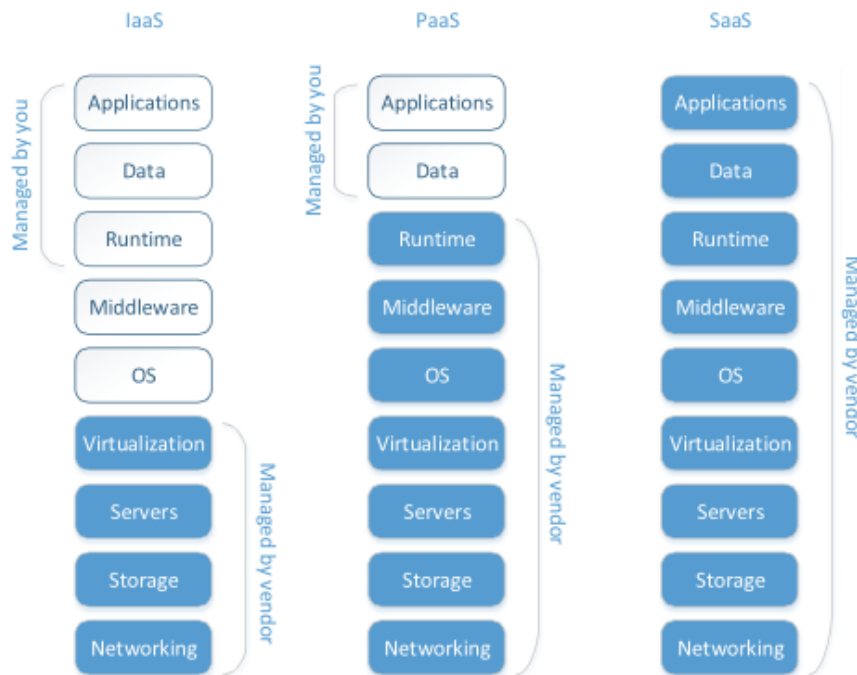


Figure 4: Responsibilities in the three service models [22].

### 2.3.1 Infrastructure as a Service (IaaS)

The NIST [30] definition of IaaS should give a first idea of what IaaS is: “The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to 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, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls)”.

IaaS is the service model where the customer has the most control over the system. The provider offers physical or virtual machines to the customer, where everything that is needed can be installed. Physical infrastructure is the pricier option but the customer has direct access and a dedicated infrastructure. Virtual machines can be cheaper, because the provider is able to make use of multi-tenancy [35]. Many times there can also be additional services booked, such as load balancers, VLANs and firewalls [22]. Moreover the infrastructure can be scaled depending on the needs of the customer. Hardware costs are saved as well as costs for employees to maintain the hardware [9].

In [9] the shared virtualization vulnerability is mentioned. This vulnerability can occur when some resources on a server, such as memory and CPU, are shared between the different virtual machines. Then it can happen that a malicious virtual machine can somehow gain access to other virtual machines over the shared resources. In this way an attacker could get different data from the system. Another variant would be that an attacker directly attacks the virtualization and gains access to the physical hardware. This is why the provider should care about security a lot, because the customer doesn’t control these parts of the rented infrastructure.

### 2.3.2 Platform as a Service (PaaS)

Again first the definition from NIST [30]: “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”.

In real life this means nothing else than renting an application development environment on a pay-per-use basis. This development environment usually consists of an web server, programming-language execution environment, a database and of course an operating system. These resources don't have a fixed capacity but scale automatically depending on the needs of the customer [22]. Kubernetes fulfil these requirements, for this reason many cloud providers offer Kubernetes-based services. This can be convenient, the customer just needs to take care about everything above the operating system level, like needed applications and their maintenance. So the customer can fully concentrate on the main thing - implementing and testing the desired applications or services for which the cloud is rented [35]. Applications can be installed and customized as the customer needs them but the development platform itself may just offer different configuration options and it has to be chosen from a limited set of offered platforms [35].

As in every cloud service model the providers have direct access to the data, since it is stored on their infrastructure. Because of this and because the customer doesn't control the layers underneath the operating system the provider is responsible for the security of the infrastructure. The customer just has to ensure the security of the applications [9]. Another point that the customers should consider when developing something in the cloud is, that they have to accept the update cycle of the provider regarding the underlying software and that this can have impacts on the development [9].

### 2.3.3 Software as a Service (SaaS)

“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 [...] 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”. This is the definition from NIST [30] for Software as a Service.

SaaS aims for customers who just need some applications and don't want to install or maintain them by themselves. The users have access to the application software and the database but no underlying layers [22]. So the provider can create different views or partitions but let the customers work on the same applications [35]. The biggest advantage for customers are the saved costs for licensing, hardware and maintenance [9]. On the other hand the automated updates and upgrades can cause trouble, because the users may need to be trained on the altered software. Also customization is only possible within certain limits [35].

Security wise there are quite a few points to pay attention to. Not only is the data saved in the data center of the provider but somehow it needs to be transferred there. So these steps have to be secured as well as the data privacy has to be ensured. No unauthorized user should have access to the data. This implies a strong authentication

and authorization method. Moreover the web application itself has to be secured, so that no intruders can enter the network through it [9].

### 2.3.4 Other service models

Besides the three above mentioned and most common service models, there are some others which will be mentioned shortly in this section. But of course there is a big number of different services so this should just give an idea about some other options.

In [32] RaaS (Recovery as a Service) is mentioned. This service helps companies to better manage their backups. Usually it also contains archiving and disaster recovery. In this way data loss is prevented and due to the providers know-how the recovery probably happens faster.

Another option is called DbaaS (Database as a Service). This model offers not just storage but also the database platform. In PaaS the customer would also need to pay for the development tools, which may not be needed" [35]. So this is a cheaper option, if just the database is needed. One more service is mentioned in [35] and this is DaaS (Desktop as a Service). As the name says this service provides a desktop for simple tasks. The customer can choose between dedicated and pooled desktops.

## 2.4 Deployment Models

There are four deployment Models regarding to NIST: public, private, community and hybrid clouds [30]. They are made to meet different organizational needs. With regards to where the infrastructure is located, who has the power over it and who cares for updates and maintenance for example [34]. Now this section is going to give a short overview of these models and their advantages and disadvantages. Regarding the migration, the main point is if the infrastructure is on-premises or not and if the hardware has to be bought by the organization itself (e.g. private cloud).

### 2.4.1 Private cloud

NIST [30] defines private clouds as follows: "The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises".

The main point here is that the cloud is just used by one organization. This means that the whole cloud infrastructure is managed and controlled by one organisation. This includes the infrastructure and maintenance of the client systems. Therefore the organisation needs to invest money in the infrastructure which is needed and may be needed in the future. These costs can be quite high, also costs for experts need to be calculated [34]. On the other hand the organisation has dedicated resources, thus no sharing of resources with other organisations [21]. Also there are more options for customizing the cloud environment, such as using different software versions and controlling the update cycle [34].

When it comes to security a private cloud is a good choice. Since the cloud is controlled by the organisation using it, no other party has access to the data and even the access within the organisation can be controlled and restricted easily [34].

If the hardware is bought by the organization itself, the planning of the infrastructure



and setting it up could also be considered as part of the migration process. Migrating to private clouds can happen over the local LAN, if the cloud infrastructure is on-premises.

### 2.4.2 Public cloud

Public clouds are defined by NIST [30] in the following way: “The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider”.

This means that the organisation using it totally relies on a third party - the service provider. This has two big advantages: cost savings for infrastructure and experts, and the adaptation of computing powers to the currently needed resources. Moreover public clouds are easy accessible from all places with internet access and have a good availability [34]. Of course the availability depends on the set service-level-agreements. Another main characteristic of public clouds is that customers just pay for the resources used in the billing time, which is usually a month [21].

One disadvantage is that the data is stored on a third party’s infrastructure, which means they would also have access to the data. Also the data ownership is not clearly defined. General security is an issue of the service provider and the customer can’t control it [34]. Compared to private clouds, public clouds cannot be customized as easily and since the cloud provider is responsible for the maintenance the customer has to put up with the update cycles and times [34].

But then again in this deployment model the customer just has to care about the devices and connections which are needed to access the cloud. The service provider has to ensure that the service is implemented correctly and can be used by the customer [34].

The migration to public clouds always has to happen over the internet, since they are usually far away from the companies premises. Nevertheless the good thing is that no hardware has to be bought and therefore one can focus on the migration of the software. Also public cloud providers often offer tools or experts, which help with the migration. Those are described in chapter 2.5.4.

### 2.4.3 Community cloud

Community clouds are shared clouds between a certain group or organisations. NIST defines them as follows [30]: “The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises”.

According to [34] this is the most unpopular and unknown model. While shared infrastructure, experts and costs are an advantage of this model, a problem can be the exact separation of who owns and who is responsible for which part of the system. Also data security can be an issue, if all organisations who own the community cloud, have access to all data [34].

For the migration, the same things apply as for public clouds.

### 2.4.4 Hybrid cloud

The fourth model is the hybrid cloud and is defined in the following way by NIST [30]: “The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)”.

This describes nothing else than two different cloud deployment models which work together. Here the advantages of private and public clouds can be used. Nevertheless it is complex to implement and therefore an expensive option. Moreover data needs to be shifted between the clouds, so not only does the bandwidth has to be big enough but also the connection has to be secured properly [34].

On the plus side having two (or more) clouds connected can create redundancy and so ensure higher availability. Hybrid clouds can help to use the cloud infrastructure in an optimal way [21]. Sensitive data, for example, can be stored in the private cloud and the public cloud can be used for not so critical services to save costs. This concept can be seen in figure 5.

Since the hybrid cloud is a composition of different deployment models, the migration depends on the used deployment model. Moreover the interface between those different clouds has to be set up during the migration process.

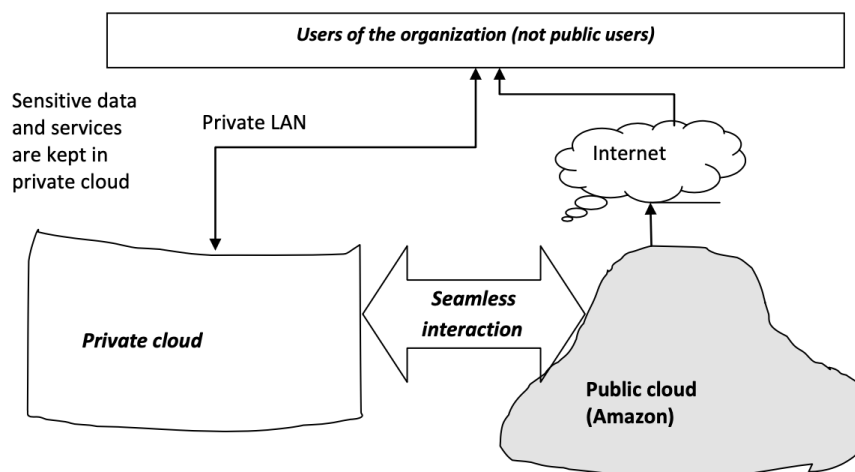


Figure 5: Scheme of a hybrid cloud [37].

## 2.5 Cloud Migration

Some ideas and motivations, to move systems to the cloud were already given in chapter 1. Basically they can be sorted into the following categories: business, technical, security and privacy motivations [36]. So there are lot's of different reasons for deciding to move a system to the cloud but once the decision for migrating to a cloud has been made, what has to be considered? What is special about moving a system to a cloud and not to a new local infrastructure?

First one has to think about which parts of the system should be migrated. Should everything be moved to the cloud or just parts of the current system? Figure 6 gives

an overview of the possibilities: either everything stays in a local system or the whole application stack gets moved to the cloud. Also parts of the application stack can be replaced with cloud software or a partial migration is done. For a partial migration only some system components are moved to the cloud. In [23] “migrating only the auditing functionality of a healthcare system to the cloud” is given as an example for a partial migration. But of course there are also other combinations feasible. Depending on the system which should be migrated, big changes can be necessary. Cloud platforms often have their own solutions, so the existing applications have to be changed accordingly to be used in the cloud. Also cloud platforms have other characteristics (see chapter 2.2) than local systems, so it is no surprise that the requirements for systems are different. Maybe also data security or other legal questions have to be considered. Moreover the question can arise, if and how the cloud systems work together with the remaining legacy system. Furthermore one has to know, which migration strategy suits which system and of course, how the system can be migrated. Automation of the migration maybe saves some time and working hours but also has higher costs than not using special migration tools. Another decision that has to be made is, if a company is paid to do the migration or if the “self-service” of the cloud platforms is used. Last but not least a suiting cloud provider has to be chosen.

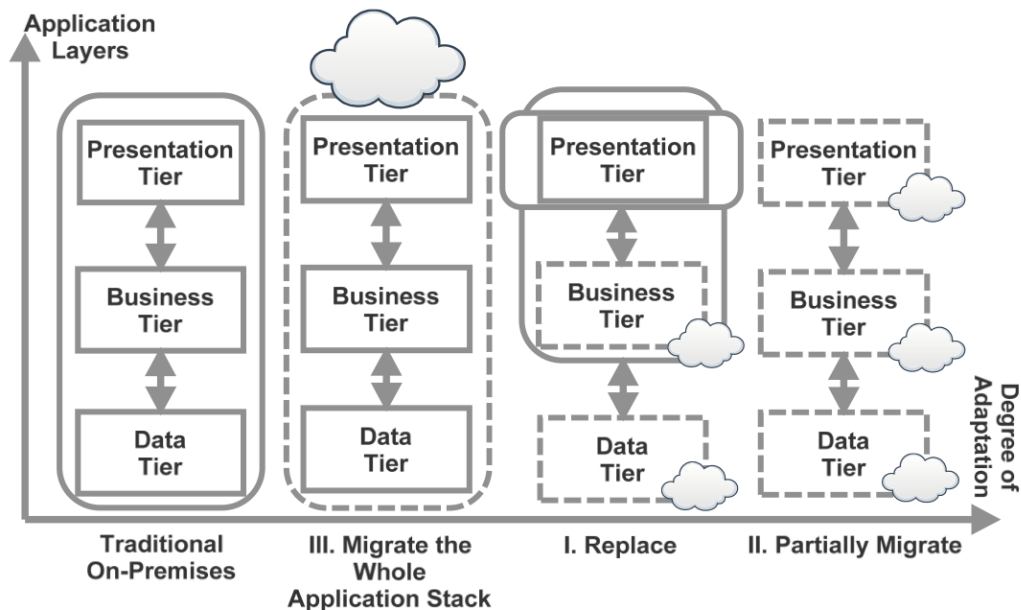


Figure 6: Parts of a system, which can be migrated to a cloud platform [23].

### 2.5.1 Migration process

As was described in chapter 2.3, there are different service models in the cloud. So the migration process depends on what should be migrated and the resulting service model to which it should be migrated. Nevertheless, there are the following five general steps defined in [1]. Although this is written by Stephen Orban, who is Head of Enterprise Strategy at AWS, these steps also apply to migrations to other cloud platforms and provide a good guideline.

- Migration preparation and business planning  
Here the goals for the migration should be specified, a time-line should be set and the costs evaluated. Based on that the decision can be made, if the migration to the cloud should happen or not.
- Portfolio discovery and planning  
Now a migration plan needs to be made. Here it has to be considered how complex an application is, what the business impact is and if one already has experience with the migration. For a first try, small and uncritical applications would be a good starting point.
- Designing applications  
Next it is time to make a plan, how to migrate single applications. For each application different changes have to be considered and therefore a suitable migration strategy has to be found (see chapter 2.5.2).
- Migrating and validating applications  
The actual migration happens. Also it has to be checked, that everything worked as expected and can be used in the cloud.
- Modern operating model  
The application is operating in the cloud and old systems can be shut down.

### 2.5.2 Migration strategies

As can be seen in figure 7, five different migration strategies were defined in [36]. Retain and retire is what happens if a system is not moved to the cloud. Retain means simply not moving but keeping the old system on the local infrastructure. Retire means, that the old system is not needed anymore and will not be moved to the cloud nor kept on the local infrastructure. The actual migration strategies are replace or purchase, rehost, replatform, refactor and reengineer. Each strategy has different use cases and depends strongly on the kind of application or system, which should be migrated.

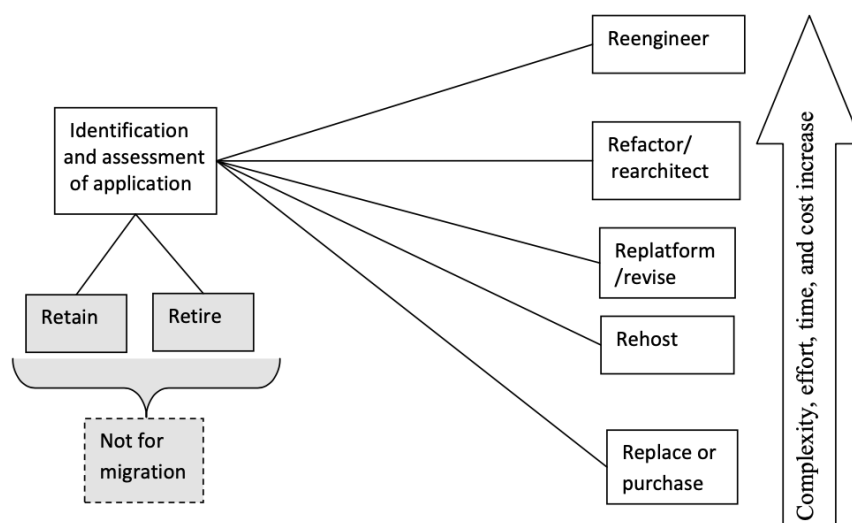


Figure 7: Migration strategies [36].

Replace or purchase systems is a good option, if the old system is retired. In this scenario a new application is either bought or an existing application of the cloud provider is used. And while this is probably the fastest way compared to the other options, the solution can just be customized to a certain point. So maybe some business processes need to be adapted [36].

When an application is rehosted, it is only moved from the local system to the cloud. No changes are made to the platform or code of the application. Reshosting systems is usually faster and less cost intensive than replatforming or reengineering them [36]. This is what is examined in chapter 4. There are, if possible, no changes made to the test system. But it should be usable in the cloud after the migration without the need of any local system.

In the replatform or revise approach the platform is only changed slightly. The function is not changed but slight alterations to the platform can improve the performance in the cloud a lot, according to [36].

As the name says, refactor or rearchitect a system means changing it on the code or architectural level. Here bigger changes are done to the system but the functionality still stays the same and the system is optimized for cloud usage [36].

Finally, when reengineering a system to migrate it to the cloud, the system gets totally changed and optimized for cloud usage. This is definitely the most complex migration strategy and usually needs more budget and time than the other strategies. Nevertheless at the end one has a system which can use the cloud resources in the best way [36].

However, all of these migration strategies strongly depend on which parts of a system should be migrated to a cloud environment and to which service model it should be migrated. Every service model has its own requirements and limitations, which are described briefly in the following paragraphs.

**Migration to IaaS:** Using IaaS as service model is recommended when a whole virtual machine should be moved and if there is no time or desire for reengineering the application. Then rehosting is the migration strategy of choice. Nevertheless if special hardware is needed or the data needs to be stored in a certain location, e.g. for legal reasons, then the choice of the cloud provider is important. Because not every cloud provider can administer special hardware [39].

**Migration to PaaS:** Usually when systems are moved to a PaaS infrastructure, refactoring is the used migration strategy. In PaaS not only the hardware restrictions have to be considered but the system has to be adapted to the used platform. In this way a cloud optimized application can be build. Nevertheless restrictions according the used programming language, database and middleware have to be considered [39].

**Migration to SaaS:** Replacing a software is the easiest option one has, if an application should be moved to a SaaS platform. Also revising or reengineering the application is possible, but for this a long migration period has to be accepted and a lot of work has to be done [39].

### 2.5.3 Mobility of services in the cloud

A special case of cloud migration is the migration between two cloud platforms. While for the migration from a local system to a cloud platform the system usually needs to be prepared or adapted to the needs of a cloud environment [36], the migration between clouds is a lot about compatibility of the different services and platforms. This mobility between clouds is also called portability and in [31] it “is expected to ensure that an application, service or data works in the same manner regardless of the consumed Cloud services”. Next to other economical, technical and legal reasons, portability should be provided to prevent a vendor lock-in. Data, services or applications can be transferred between cloud platforms. Of course, for some systems portability is easier to implement than for others. So it is not likely to have the option to move a value added service offered by a cloud provider to another cloud platform. Nevertheless, portability can be accomplished by using open libraries and services and employing standards for example. This are definitely tasks for the cloud providers and can’t be done by the customers. Unfortunately standardization often contradicts with the marketing strategy of cloud providers, which try to offer unique products and solutions [31].

### 2.5.4 Automation of the migration process

Moving systems from one infrastructure to another can be very complicated, no matter if it is between clouds or between a local infrastructure and a cloud environment. Therefore automating this process would help a lot. Unfortunately this is not always possible, because every company and organization has its own individual and complex infrastructure, which is adjusted to their needs. Nevertheless, cloud providers sometimes offer tools, which can handle the migration of small and uncomplicated systems. If this isn’t possible and one doesn’t have the know-how to migrate one can still pay for a team of experts to do the migration. Another attempt for data migration is used by AWS with their product AWS Snowball. If huge amounts of data need to be transferred, sometimes uploading it directly to the cloud is not the best option. That’s why AWS offers AWS Snowball. On this device up to 50TB of data can be stored, which then it is sent to an AWS location where the data gets migrated to the cloud [6].

**Tools offered by cloud providers:** All cloud provider offer certain tools for importing systems, but not all have the same functionality. The tools for migrating virtual machines from one cloud platform to another cloud platform are described in section 6.1.2. Once these tools are set up, high numbers of virtual machines can be migrated at once. They all work on the same principle, in the documentation of Azure [17], it is called Agent based migration. And the tool for Azure is called Azure Migrate: Server Migration. For the migration a server has to be set up in the source cloud. This is a so called replication appliance server, which coordinates the migration of the virtual machines to the destination cloud (Azure in this case). The replication appliance server itself does not get migrated. Moreover an agent has to be installed on the virtual machines, which should be migrated. The architecture can be seen in figure 8.

The virtual machine migration tools from GCP and AWS have a similar architecture. In GCP it is called Cloud Migrate for Compute Engine and in AWS SMS (server migration service) connector. Interesting here is, that only virtual machines from Azure can be

migrated to AWS, while GCP offers import strategies for AWS and Azure and Azure offers to import virtual machines from GCP and AWS.

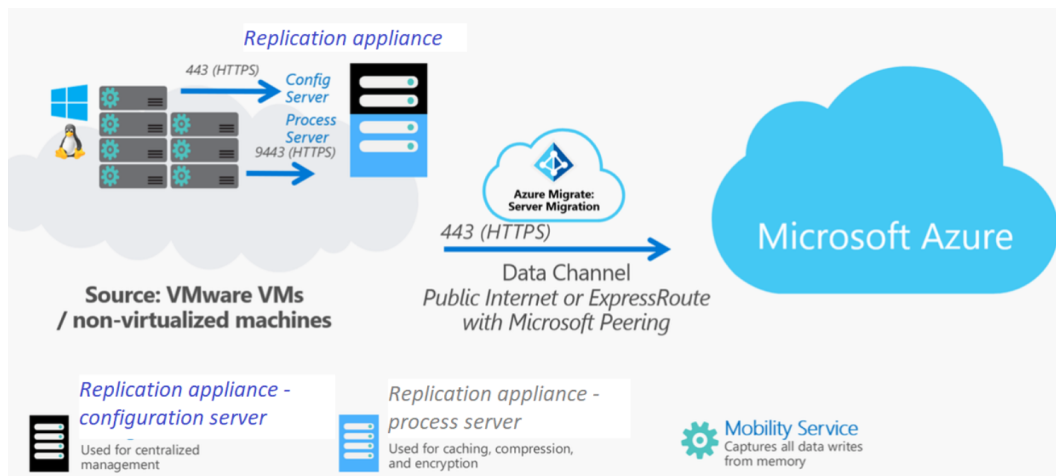


Figure 8: Azure migration architecture [17].

Not only virtual machines can be migrated automatically. Also for the storage import GCP offers a tool to directly import data from AWS and Azure. How it works is explained in section 6.1.4. The Transfer Service can also be used for directly migrating data from a local system to GCP.

With Azure Migrate not just virtual machines can be migrated. According to [16], also a Web app migration assistant and Azure Data Box exist. With the Web app migration assistant .NET and PHP web applications can be assessed and migrated to Azure. With the Azure Data Box offline data can be migrated. Moreover there are tools to assess the systems, which should be migrated and shows potential problems.

**Tools offered by 3rd party vendors:** Not only with offering cloud services money can be earned but also with the migration. So it is no surprise, that there exists a number of third party tools, which help with the migration process to a cloud platform. AWS lists some tools from competency partners [3], which offer tools not just to assess the migration but also for the migration itself. Among other things, these tools help with migrating servers and data. Datadog, for example, helps comparing and in a second step combining systems which run on-premises and in the cloud. Deloitte offers a ATAmotion Migration Module, which assists with the migration of big workloads for Windows and Linux systems [3].

**Partner assisted:** Another option is to get help with the migration form another company. This partner assisted migration is definitely not a bad idea, if someone has a big system to migrate but no experience in doing so. Contrary to using tools, which make the migration easier, here the migration is fully guided or done by a third party. AWS for example, not only lists tools but also competency partners [3]. Also Azure has a list of trusted partners, which offer a managed migration.

## 2.6 The cloud platforms investigated

Organisations usually have a big selection of cloud platforms to choose from. There are not just GCP, Azure and AWS but also vCloud Air from VMware, Oracle Cloud from the Oracle Corporation, OTC (Open Telecom Cloud) from T-Systems International, IBM cloud Computing from IBM and Cloud Foundry from the Cloud Foundry Foundation (non-profit organization), to name a few. While Google, Amazon and Microsoft are maybe the most popular providers, the others also have interesting offers and connect their cloud solutions with their other products. One example is Oracle where the user can choose whether a virtual box should be created locally within the VirtualBox application or in the cloud. In figure 9 the actual market shares of GCP, AWS and Azure are seen.

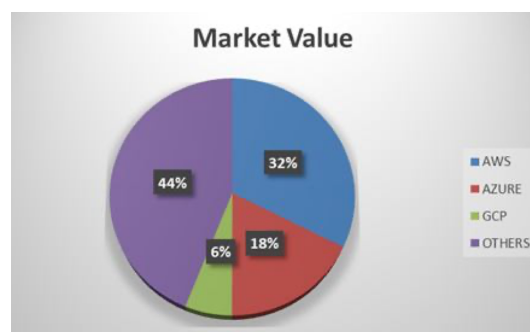


Figure 9: Market shares of GCP, AWS and Azure in 2020 [10].

Now the question is, which points should be considered before choosing one of these cloud providers. In [24] some aspects are mentioned such as features, security and familiarity with the brand. As will be seen in this section, the offered services and features are not the same for all providers and so one has to choose the most suitable one. Moreover different organizations have different needs, not just when it comes to the services but also securitywise. Companies dealing with financial or health information maybe have stricter security guidelines than others. Brand familiarity is important, because at the end of the day, users are still human beings and it is known that people usually stick to what they already know. Moreover the prices are of importance when choosing a cloud platform but they are not always easy to compare, specially because the offered services are not exactly the same.

The focus of this section will be on Google Cloud Platform, Microsoft Azure and Amazon Web Services, which are used for the experiments in this thesis. Obviously there are some common grounds but also differences in billing, availability and offered services. Every cloud provider has strengths and key competences on which great emphasis is laid and which keep them ahead of the competitors. Also the cloud provider offer different free services for familiarising with their products. These offers were used for the experiments.

### 2.6.1 Google Cloud Platform (GCP)

Google Cloud Platform is a product of Google, LLC (former Google Inc.). Google, LLC was founded in 1998 by Larry Page and Sergey Brin and is mainly known for the search engine. Other products, which were developed by Google are Google Drive, Google Maps, Chrome and Gmail for example. These products are not just really popular but they are



used by billions of people each day [27].

Google Cloud platform is the public part of the Google Cloud, this means it works on the same infrastructure as other Google services [24]. The logo of the cloud can be seen in figure 10, it is a cloud in the colors of Google. GCP offers lots of different services and products. The services are listed in [26] and have a broad range, including computing and hosting services, Big Data services, machine learning services (with an AI Platform), networking services, database services and storage services. The computing and hosting services cover PaaS, which is called App Engine in this case. It can be used to develop apps in various programming languages. Also serverless computing, container and virtual machines are part of the computing and hosting services. The main parts of the networking services are routers, firewalls, load balancers and Cloud DNS. Moreover Google has a very good offer of AI services, which is where they develop a lot and try to keep ahead of the other providers.



Figure 10: Logo of GCP [29].

Users have different options how to work with GCP. There is the Google Cloud Console – the graphical user interface – on the one hand and the command-line interface and the client libraries on the other hand. The command-line is called `gcloud` and can be used via a browser based shell (Cloud Shell), if one doesn't want to install the Cloud SDK. So users can choose the method they prefer working with.

As everywhere in business, tracking costs is also important in GCP. To do so GCP offers a pricing calculator and every account has a billing page. This page shows the current costs and gives an overview of how the costs evolve. The billing is based on projects, which need to be created before one can work in the cloud. Each project is associated with one billing account [29]. Google uses a pay-per-use model, so users just pay for the resources virtually used. Moreover some free services are offered for beginners or people who just want to have a look and try some things. Those are a 90-day free trial with a free budget of 300\$, with this all services can be tested within this period and the set money limit. Afterwards the normal billing rules apply. Another option are the free tiers, those are services which are offered for free in general within a certain limit [28].

### 2.6.2 Amazon Web Services (AWS)

Jeff Bezos founded Amazon.com, Inc in 1994 in Seattle and in 2006 Amazon Web Services, Inc was founded as a subsidiary of Amazon [7]. So Amazon Web Services just offers cloud services, nevertheless the logo of AWS (see figure 11) is similar to the one of Amazon.

Lots of services are offered by AWS in all kind of areas. Examples for such services are development tools, machine learning services, container, media services, blockchain applications and storage services. The database services naturally include SQL options but also a graph database and a time-stream database, where billions of events per day can be stored in an uncomplicated way. In the network and security category various services are offered, such as Amazon VPC (virtual private cloud), identity management for applications, secrets and firewall managers. Also a so-called Amazon Detective, which can be used to find potential security issues in the system is offered [4].

To easily find and buy new software and services AWS introduced its marketplace. The offered products are usually not software and services which are developed by AWS but from third parties. So independent software vendors can sell their products there, if they passed the curation process of AWS. A big advantage is that the billing happens via the AWS account, so the customer doesn't have to create new accounts for buying those products. And as everything in cloud computing also the prices and license options are very flexible here. Customers and sellers can agree on what seems right to them [8].



Figure 11: Logo of AWS [2].

To keep an eye on the costs AWS offers a cost explorer with budget, cost and usage reports as well as a dashboard where the most important information about current costs is given. Contrary to GCP and Azure there is no free test budget in AWS. Nevertheless there are some services which are offered for free. They are divided in three groups: the ones that are always for free, the ones which are just for free in the first twelve months of AWS usage and then there are some free trials which change over time and which are just available for short time spans [2].

### 2.6.3 Microsoft Azure

In 1975 Microsoft was founded in Albuquerque by Bill Gates and Paul Allen. Its most famous product is the operating system Windows. While Windows is already used since 1985 it took many more years until cloud computing became relevant. Finally Azure was launched in October 2008 and in 2018 the services could already be used in 140 countries [11]. Obviously the logo of Azure is held in corporate identity with other Microsoft products, as can be seen in figure 12.

Today Azure offers more than 200 applications and services “to solve today’s challenges and create the future”, as it says on the homepage [14]. Of course database services are offered, mainly for SQL but also some non SQL options. Moreover AI and machine learning services, container, virtual machines, development tools, storage services, blockchain applications and many more. The networking and security services include gateways, DDoS protection, Azure DNS, firewalls, load balancers, Azure defender (to safeguard hybrid clouds), virtual networks, a security center and many more [12].

A highlight is the Windows Virtual Desktop, there users can easily connect to a Windows desktop with any device. No extra license is needed, if the customer has already a Microsoft 365 or Windows per user license. So Windows 10 can be remotely used, even with multi-session access to reduce license costs [15].



Figure 12: Logo of Azure [13].

Microsoft not just offers a tool to calculate the costs for running applications but also a tool to calculate the savings when migrating a system to Azure. Azure Cost Management and Billing gives an overview about current costs, helps analysing the spendings and manage budgets. In Austria and Germany the free trial offer of Azure is similar to the one of GCP. Azure gives a free budget of 170€ in the first month and certain services are free to use within the first 12 months. After this month or if the credit is used earlier, the user needs to update the account and a pay-per-use billing system will be used further [13].

## 3 Approach

Every experiment has to be defined before it is performed. So in this section the methods for the performed experiments are described. The test system and test scenarios are introduced, as well as an explanation given, why they were chosen and what criteria need to be fulfilled, so that the experiments can be counted as successful. This is the foundation for the evaluation of the experiment, whether something can be improved, if parts of the experiment failed, or if everything worked out and all goals have been accomplished.

Three things need to be defined, before the experiments can be started. The first point are the used cloud platforms, the second point is to define the test scenarios and the third is to define the test system. The reason for using GCP, Azure and AWS is simple: they are the biggest and most popular cloud platforms. As for the test scenarios in this thesis: various tests and experiments are performed, which are connected to migrating systems to clouds, between clouds and from a cloud to a local system. Also the preparation of the systems is examined, if and what changes need to be done when they are moved. Therefore it is essential to get to know the system requirements of the different cloud platforms. Moreover it is the foundation for choosing which systems can and should be run in a cloud environment and which are better kept on a local infrastructure. Also the test system is an important part of this constellation. To have meaningful results the test system needs to be up-to-date and relevant for businesses. So it was tried to find technologies which are widely used and which are at least theoretically compatible with all used cloud providers.

### 3.1 Tests to perform

There are three types of test scenarios. The first one is probably the most relevant one for companies, because it is about migrating a local system to a cloud system. Every company, which uses on premises solutions and thinks about moving to the cloud, has to think about this scenario. The second one is the reverse process: moving a system from a cloud environment back to a local system. This is relevant if a company is not satisfied with the the cloud solutions or the privacy and data guidelines changed and no public cloud can be used anymore. The third scenario is about moving systems between different cloud providers. This can be necessary if another provider offers better prices or services. All of the mentioned scenarios have to be performed for all parts of the test system.

#### 3.1.1 Migrating a local system to a cloud platform

First, everything needs to be set up locally for this scenario. How this local setup looks like is described in section 3.2. After this first step, the actual experiments can start. Next the system requirements need to be checked and the local system has to be prepared if necessary. GCP for example offers a pre-check tool for virtual machines, to check if everything is ready for moving the virtual machine to the cloud. Then, the system has to be exported in a certain format, which can be used for the import to the cloud. This is the main step: Moving the system to the cloud. This happens for all three cloud platforms, so the local system is moved to GCP, Azure and AWS. The point is to find out, if the import of different systems is possible to all three cloud platforms and what effort has to be made to transfer the test system. The result of this experiment should be a running system in the cloud. This means the system should work as before. So user, settings and

data should be available. To test this, the system is started in the cloud. For virtual machines and docker container this means that they should be running as in the local environment. For databases certain SELECT and SHOW statements are tried and they should give the same results as in the local environment.

### 3.1.2 Migrating a system from a cloud platform to a local system

Here the starting point is the cloud platform. So before the experiment can start, the system has to be set up in the cloud from scratch. This time the data, user and other relevant settings need to be exported from the cloud platform and set up locally. The necessary environment is already installed on the local platform, but everything else needs to be transferred. From the cloud platform the files and data are exported and then moved to the local system, hopefully with no editing in-between. This local system then needs to be started. The experiment is successful, if everything is up and running locally and all data and users are still present.

### 3.1.3 Migration between cloud platforms

When migrating the test system between cloud platforms there are two options how this can happen. One is the direct version: some cloud providers offer to directly import something from another cloud platform. If this isn't possible the second option is needed – the indirect one. Indirect means, first exporting a system from the cloud environment, then saving it locally, if necessary making some changes, and finally importing it to the second cloud platform. For this option no direct interface between the cloud platforms is needed.

**Direct – from cloud to cloud.** Not every system can be moved directly between clouds. On the one hand it depends on the kind of system that should be transferred and on the other hand on the used cloud platforms. So this experiment is limited to the offered options of the cloud providers. And therefore only the following scenarios are tested:

- Virtual machine: from GCP and AWS to Azure, AWS and Azure to GCP, Azure to AWS (a direct transfer from GCP to AWS is not supported)
- Database: for all combinations of cloud platforms
- Docker container: from AWS to GCP and Azure and from GCP to AWS and Azure (a direct transfer from Azure to another cloud platform is not supported)
- Storage: from AWS to GCP and Azure, from GCP to AWS and from Azure to GCP (a direct transfer from Azure to AWS and from AWS to GCP is not supported)

As usual, when an experiment starts with an export, first the system needs to be set up in the cloud environment. This already happened in the previous test scenario, which is described in 3.1.2. Then a connection between the clouds has to be established, so that the system can be moved and set up in the new cloud environment. Also here the requirement is, that the system is running on the other cloud platform and all data and users are migrated.

**Indirect – with the help of a local system.** This is the backup option, if a direct move from one cloud platform to another is not possible. Basically the steps from 3.1.2 and 3.1.1 are combined here. In exactly that order: first exporting the system to a local environment and then importing it to the other cloud platform. For this, first the system has to be set up in the according cloud platform or taken from previous experiments. Locally everything should be prepared for storing the exported data, which maybe needs to be modified somehow, depending on the used cloud platforms. Eventually the system can be moved to the other cloud platform. This will take place for all combinations of test systems and cloud platforms.

## 3.2 Test Systems

To get a fair comparison of the cloud providers, a test system is set up. This system is used – as far as possible – for all experiments in this thesis. It was tried to find popular products and options which are often used so that they are compatible with every tested cloud environment. Nowadays companies have lots of different systems and some of them are covered with this test system. First there is a database with triggers, views, stored functions and procedures and users. Moreover virtualization is a big topic, so another part of the test system are virtual machines and docker containers. Last but not least, companies usually have big amounts of data, so also a storage system is part of the test system. Storage system means, in this case, a set of different objects.

### 3.2.1 Databases

Databases are a very important part of every IT infrastructure. For the experiments here a MySQL database (version 5.7.31) was chosen and together with MySQL Workbench (version 6.3.8) set up locally on Ubuntu 18.04 LTS (running in VirtualBox). As database Sakila is used. This sample database is published as open source under the terms of the BSD License. It models a system to rent DVDs in different stores. A scheme of the database can be found in figure 13.

Sakila not just contains tables and test data but also triggers, views, stored functions and procedures. The tables contain information about the film, film category, in which language the film is available, the actors but also about the inventory and staff of the store, the customers, their payments and rentals as well as their addresses.

Triggers automate processes in the database. These processes get started, if a certain event happens. Inserting, deleting or updating something can be such an event to set off a trigger. In Sakila there are six triggers. Three of them set the current dates, when either a customer is created, a rental or a payment is done. The other three are about the `film` and `film_text` tables. Those two tables share some columns and so there are three triggers which copy the data into the other table, e.g. if a film is inserted into `film` then the trigger inserts this also in `film_text`. The same happens with update and delete.

Moreover there are three stored procedures. One checks if a film is currently available (in stock) in a certain store. Another one checks if any copies are not in the store at a given time and the third lists the top customers of the previous month.

Also three stored functions are available in Sakila, those give the currently available budget of a customer account, show if a certain film is currently rented by a customer, and weather a specific item is currently available in a store.

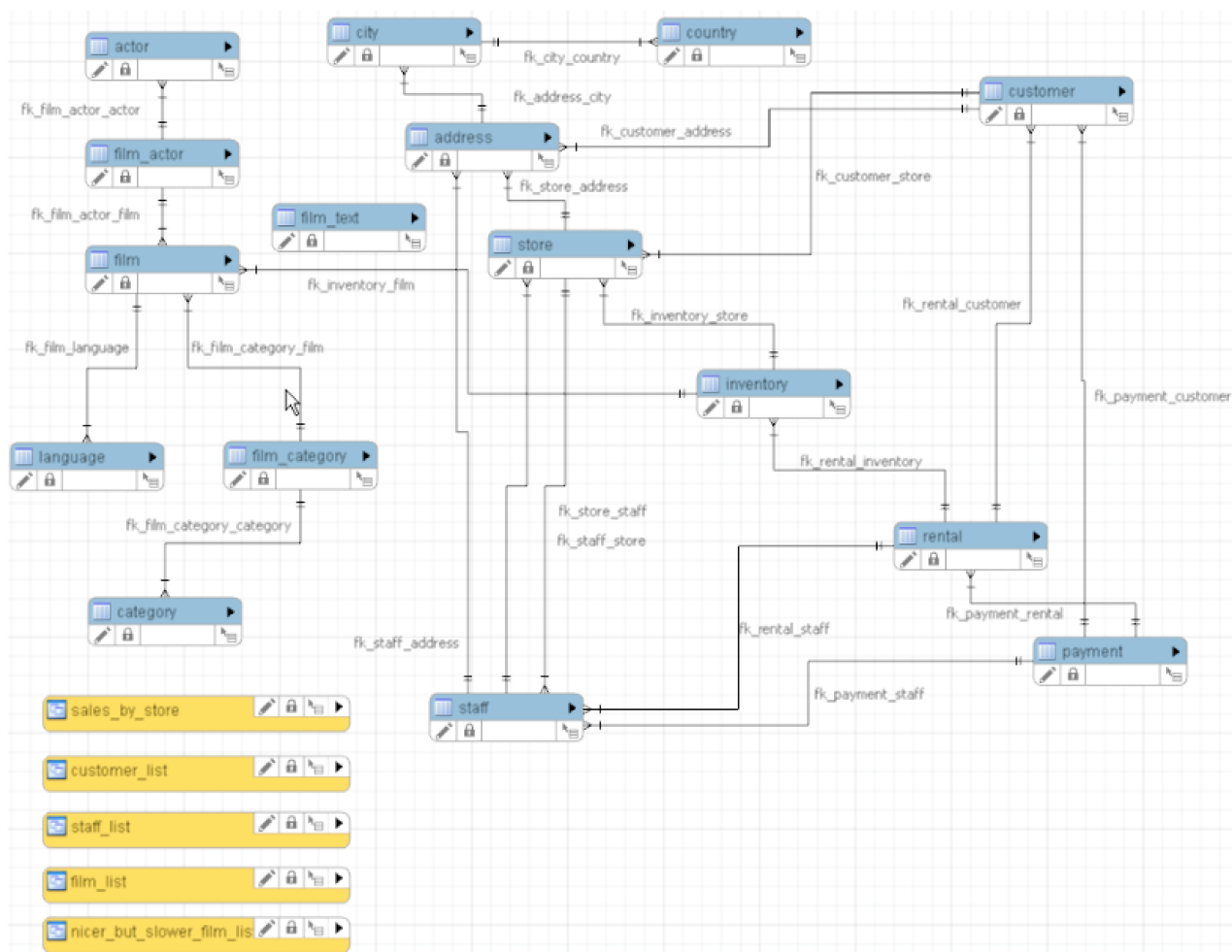


Figure 13: Scheme of Sakila (sample database) [19].

Also seven views are available. Those provide special views on the data, for example the `nicer_but_slower_film_list`, which gives the same information as the `film` table but formats the data in a visually better way. Nevertheless this takes more time than creating the ordinary view. Moreover there are views for the staff and film list, the made sales by each store and the made sales depending on the film category.

Also some users are created with different access rights. Of course there is the root user with the according rights. A user `tanja` also with root-rights simulates the database administrator. The users `test1` and `test2` with less rights simulate users of the database. Those just work with certain, limited data access and don't have any rights to create new users.

### 3.2.2 Virtual Machines

Another important part are virtual machines. To test this two virtual machines were set up. Considering all the options which are supported by different cloud providers, the size of the virtual machines and needed licenses, the final choice was Linux Ubuntu 16.04 LTS (64 bit) and Linux Ubuntu 18.04 LTS. The VirtualBox guest additions are installed as well as a GUI and some files are copied into the virtual machine to see, if after the

migration the virtual machine is still fully usable and has all the data on it. Locally it is running in VirtualBox (version 6.1.14) [20].

### 3.2.3 Docker

The next part of the test system is a docker container. A simple docker container, which contains Ubuntu is used. It was taken from the official Ubuntu repository. To check if data and installed programs are also migrated and work, some files were saved there and the web server nginx was set up in the container. Locally docker is running on Ubuntu 18.04 LTS (in a virtual machine).

### 3.2.4 Storage

Finally also (file) storage should be checked, so a simple structure with sub-folders, files, images and videos is set up. This whole setup has a size of 1.83GB and includes 1 891 different objects. Access rights are considered in the experiments, but other metadata is not. Three users have access to the data, one is the owner, one has read and write access and the third can just read.

## 3.3 Assessment criteria

It is important to define the conditions, which have to be fulfilled so that the experiments count as successful. In this case it has to be decided when a migration is successful. No matter, if the migration was to the cloud, to a local system or between two clouds, the criteria and expectations are the same. The point is, that regardless to where the system is migrated it has to work as before and offer all contained functions. It should not be necessary that after the migration the user needs to copy any data or reinstall programs. So the following checklist gives an overview about the assessment criteria:

- the system has to be up and running
- all features which were used before need to be available in the new system
- as far as possible all settings should be as they were in the old system
- no data should be lost
- everything (data, features, settings) can be transferred within the main migration process, the users do not have to add anything manually after the migration process

To some parts of the test system, all of the above mentioned points apply. To others, like the storage part, just one or two points apply. For the storage, for example, the fourth criteria is the relevant one. There all sub folders and data should be moved from one system to the other without data loss. For this it would be not realistic to think that access rights can be transferred easily without manual work. Nevertheless most cloud platforms offer ACLs (access control lists), so that different user have different accesses. For databases on the contrary all points apply. The most interesting point is probably the second one. It means that not only the data and table structure needs to be migrated but also triggers, views, stored functions and procedures, as well as users and their rights. If this migration is possible in one migration step or if manual adjustments are necessary,



the experiments in later sections will show. Whether all the data has been migrated can be checked in MySQL-Workbench or with the following statements. They show the tables, the count of the datasets in one table (this has to be executed for all tables), triggers, stored functions and procedures, views and users.

```
SHOW TABLES;  
SELECT count(*) FROM <table>;  
SHOW TRIGGERS;  
SELECT * FROM information_schema.routines;  
SELECT table_schema, table_name FROM information_schema.views;  
SELECT * FROM mysql.user;
```

For virtual machines the points above mean, that all the installed programs, the saved data and system settings are transferred. Optimally also the GUI should be transferred. A working virtual machine should be up and running at the end, maybe even multiple machines which were created from the same image.

Docker container have similar criteria as virtual machines. The installed programs and features should still be there and be able to run.

### 3.4 Migration approach

Usually there is more than one way to import a system to the cloud or export it from there. So, after looking at the different ways of migrating systems, the one which requires the least working hours and has little cost is chosen. To have an eye on the costs is important, because for this thesis free tiers and free trial credits were used. These are provided by the different cloud providers for a certain period of time. Also the system which is migrated should not be changed if possible. If a certain migration attempt is not working, another one will be tried, if one is available. Moreover it is tried to only use third party tools for the migration if there is no other way. Otherwise the import, export or transfer function of a cloud platform will be used.

## 4 Migrating a local system to a cloud platform

Before one can work with the tools offered by cloud providers, the local systems have to be migrated to the cloud. So this chapter is about migrating the different test systems to the three cloud platforms GCP, AWS and Azure. First the migration of the database, then the virtual machine, docker and finally the storage will be examined.

### 4.1 Databases

To import the database to the different cloud platforms, it first has to be exported from the local system. Fortunately MySQL-Workbench offers an option for that and so the export is quite uncomplicated. The database, which should be exported can be chosen, as well as single tables and views can be excluded. Moreover the structure and data can be exported at once. It can be chosen to also dump stored procedures, functions, events and triggers. The export screen can be seen in 14. Users need to be exported later separately, this will be described for each cloud platform in the according section.

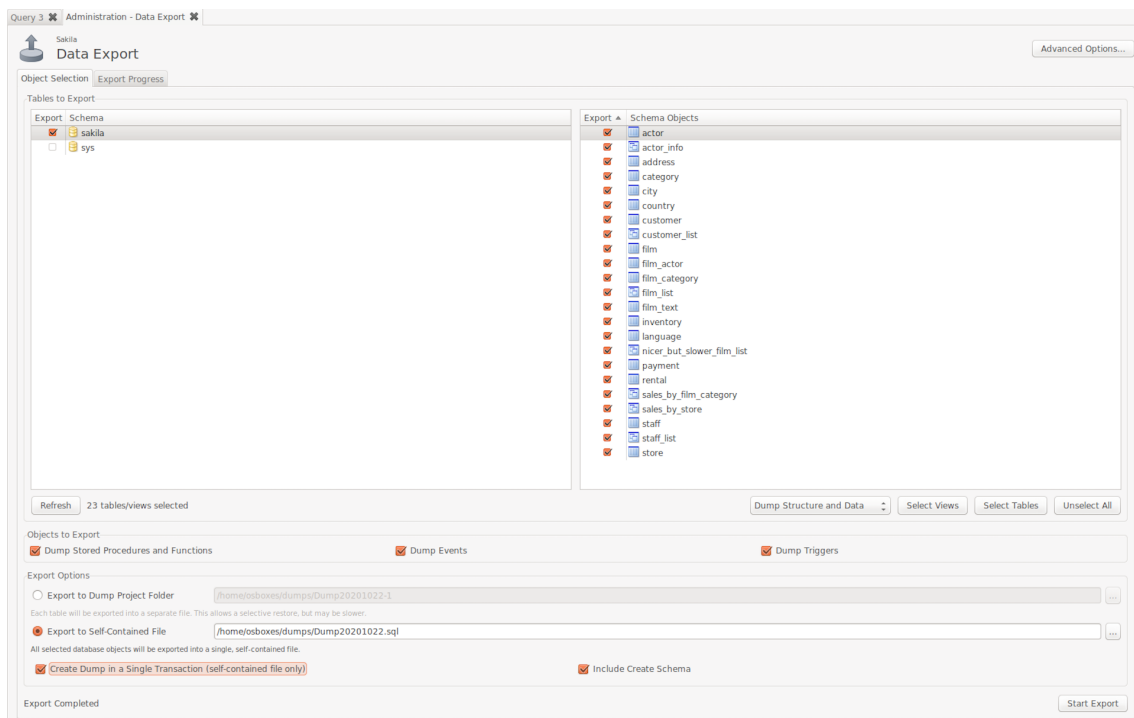


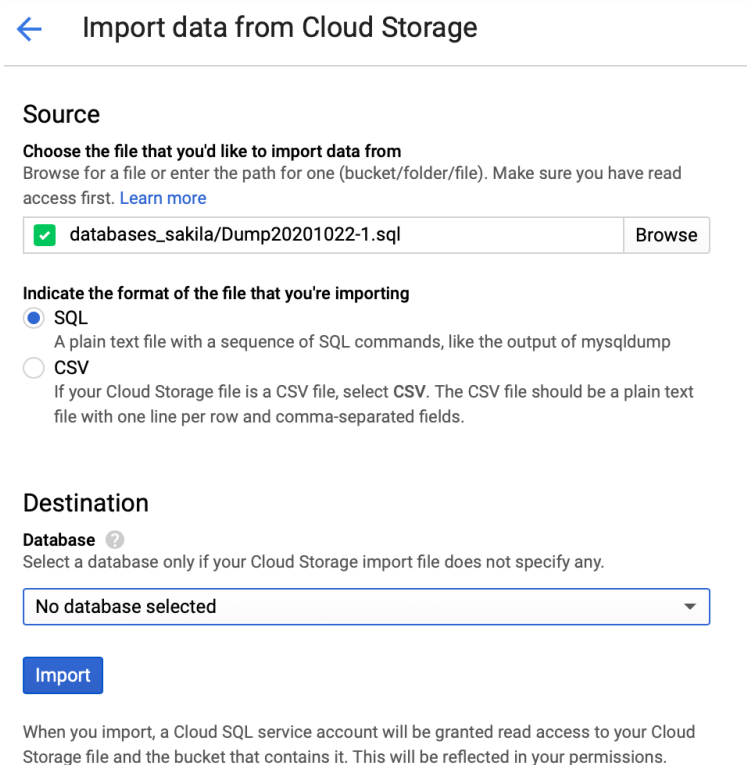
Figure 14: Mask for exporting a database with MySQL-Workbench.

#### 4.1.1 To GCP

To transfer the database into GCP, the created dump needs to be uploaded to the cloud storage, from there it can be imported to GCP. Before this can be done, an instance has to be created. It needs a unique instance ID. The region where it will be saved can be chosen, also a password for the root user and the version of the database can be assigned. Next the `log_bin_trust_function_creators` flag has to be set to one. This for some reason can't be done with gcloud (cloud shell) but has to happen in the GCP-Console. There, first one has to click on the "Edit" -button, scroll down to "Flags" and chose the

`log_bin_trust_function_creators` flag. If this flag isn't set the import of the triggers will not work.

The actual import happens with the menu in the GCP-Console. This can be seen in figure 15. There are not many options: a source can be defined, which must be in a bucket of the cloud storage, the imported file format must be given, here it is SQL and finally a database can be chosen to which the data should be imported. In this case nothing was chosen, because a new database is created. In this way, structure, data, views, triggers, stored procedures and functions are imported. Everything is there as it should be, just users can't be imported in that way.



← Import data from Cloud Storage

**Source**

Choose the file that you'd like to import data from  
Browse for a file or enter the path for one (bucket/folder/file). Make sure you have read access first. [Learn more](#)

databases\_sakila/Dump20201022-1.sql

Indicate the format of the file that you're importing

SQL  
A plain text file with a sequence of SQL commands, like the output of mysqldump

CSV  
If your Cloud Storage file is a CSV file, select CSV. The CSV file should be a plain text file with one line per row and comma-separated fields.

**Destination**

**Database** ?  
Select a database only if your Cloud Storage import file does not specify any.

No database selected

When you import, a Cloud SQL service account will be granted read access to your Cloud Storage file and the bucket that contains it. This will be reflected in your permissions.

Figure 15: Database import screen of GCP.

To import users some more steps are necessary. First the data needs to be extracted from the local system. This can be done with `“SELECT * FROM mysql.user”`. This query gives the usernames and their grants. For creating other users directly in the database, a user has to have root privileges (in GCP they are called super privileges), which is not granted to any user in GCP. Nevertheless there are two options how to create users: in the Cloud-Console or with the following `gcloud` command: `“gcloud sql users create [user_name] -host=[HOST] -instance=[INSTANCE_NAME] -password=[PASSWORD]”`. Then every created user has the role `cloudsqlsuperuser` with the following grants: `CREATEROLE`, `CREATEDB` and `LOGIN`. So every created user has the same permissions, just the name, password and instance can be chosen. These default settings then can be customized – to change the permissions, the `mysql` commands `REVOKE` and `GRANT` can be used. However there are limited options, because not even to the root user `SUPER` privileges are granted.

One important point is how to connect to the cloud database. There are some options, for example the cloud shell. This may be the easiest way, because it is just a shell which

appears in the web browser. Other options are computing engines or docker images. Computing engine means nothing else than a virtual machine in which a MySQL client can be installed. Then this can be connected via a private IP-address. For this option the instance has to be configured to allow private IP-addresses, which can be done in the section “connections”. Don’t forget that the SQL-instance and computing instance have to be in the same region. Of course also the locally installed MySQL-Workbench can be used to connect by using the IP of the database. The SQL-statement to connect is the following: `mysql -h <IP> -P 3306 -u root -p`.

#### 4.1.2 To AWS

In the mysqldump, which will be imported to AWS, triggers, stored functions and procedures were excluded, because there is no way to import them. It is a big disadvantage of AWS that these parts of the database have to be transferred manually.

But let’s start at the beginning, because before a dumpfile can be imported, a database instance has to be created. In AWS this service is called RDS (Relational database service). Because the free tiers are used for this experiment, the “easy create” option has to be used. The advanced option is only available for paid services. Nevertheless some settings can be changed while and after the instance is created. A region has to be chosen, as well as a name for the instance and a master username and password.

As in GCP the flag `log_bin_trust_function_creators` has to be set to one. This flag is needed, so that views can be imported and triggers, stored functions and procedures can be created later. If it is not set to one the views will not be imported and triggers, stored functions and procedures can not even be created manually. The flag can be set in the AWS-Console but a new parameter group has to be created, because the given group can’t be modified. A parameter group in AWS is nothing else than a set of configurations, which is required if multiple instances are used. This are again some clicks and some searching for the right options, but once this and a restart of the database instance are done, the actual import can be started.

In the documentation of AWS a statement can be found to directly load the sqldump to the database. This statement looks as follows and needs to contain the information about the source database, a user with rights to all tables and the information about the destination database, which is in AWS. Stored functions, procedures and triggers have to be excluded otherwise the import will fail.

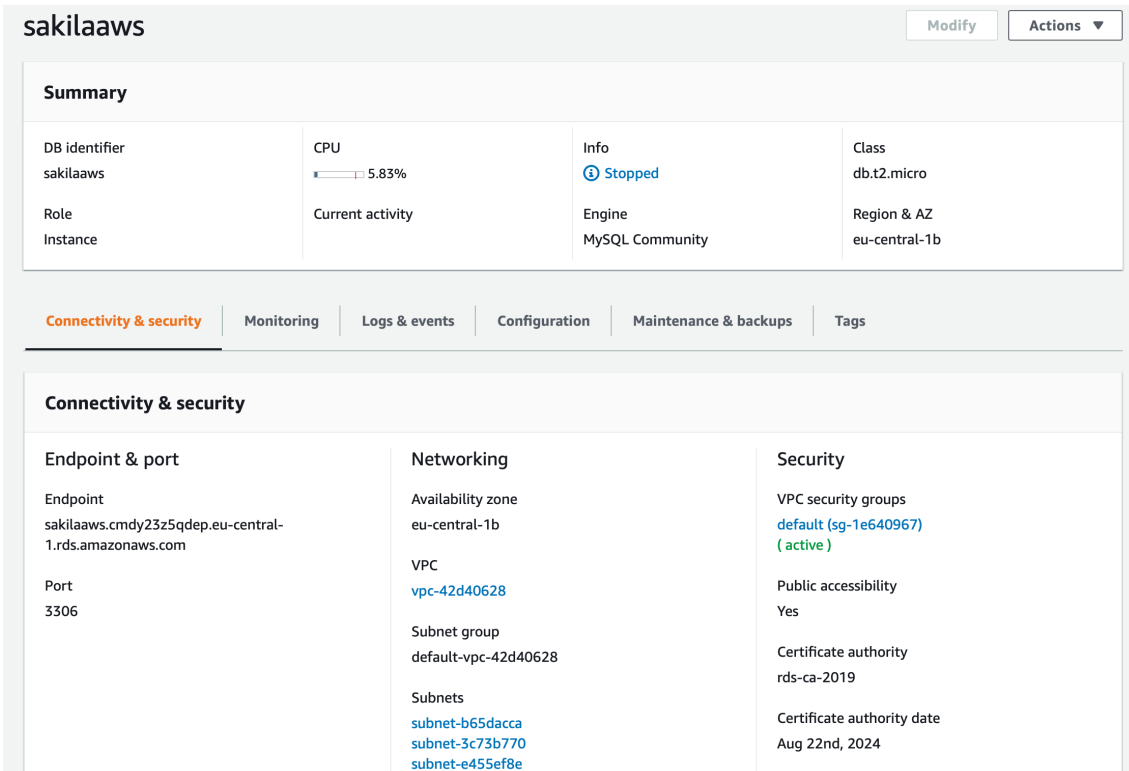
```
mysqldump -u <user>
  --databases sakilaLocal
  --single-transaction
  --compress
  --order-by-primary
--routines=0 --triggers=0 --events=0
  -p | mysql -u admin
  --port=3306
  --host=sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com
  -pPasswordAWS
```

After this is done the schema of Sakila, data and views are in the RDS-database. However, triggers, stored functions and procedures now need to be added with the CREATE

statement. These statements can be copied from a mysqldump, which includes triggers, stored functions and procedures. Another option would be to do another mysql dump, which includes only triggers, stored functions and procedures, remove the definers with a “sed” (as it is done in 6.1.1) and directly pipe it into the new database. The point is, that the triggers, stored functions and procedures have to be created in a second step and not when the database is created.

Also the users have to be added similar as in GCP: getting the information about the users from the local database and then adding them manually with the CREATE statement for user.

To connect to the database either the public access has to be enabled or the inbound connections in the security group have to be changed. For the inbound connections the IP-address of the local computer can be added, of course also a range of IP addresses can be added. Informations needed to connect are the endpoint and port. To find them one has too look in the detail information of the instance. A screenshot of these informations can be seen in figure 16. The name is sakilaaws, the endpoint is `sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com` and the port is the default port (3306). Knowing this, a connect from MySQL-Workbench or a SQL-client can be established with the following statement, where admin is the masteruser which was created before: “mysql -h `sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com` -P 3306 -u admin -p”



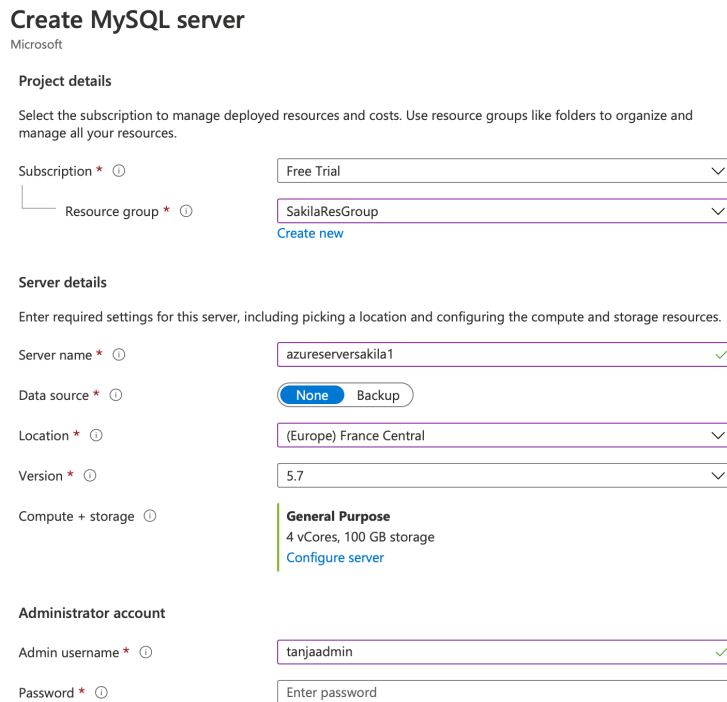
The screenshot shows the AWS RDS console for instance 'sakilaaws'. The 'Connectivity & security' tab is selected, displaying the following information:

Endpoint & port	Networking	Security
<b>Endpoint</b> sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com	<b>Availability zone</b> eu-central-1b	<b>VPC security groups</b> default (sg-1e640967) (active)
<b>Port</b> 3306	<b>VPC</b> vpc-42d40628	<b>Public accessibility</b> Yes
	<b>Subnet group</b> default-vpc-42d40628	<b>Certificate authority</b> rds-ca-2019
	<b>Subnets</b> subnet-b65dacca subnet-3c73b770 subnet-e455ef8e	<b>Certificate authority date</b> Aug 22nd, 2024

Figure 16: Connection information of the created AWS RDS instance.

### 4.1.3 To Azure

To migrate a database to Azure, first a database instance has to be created. This happens by clicking on “Azure DB for MySQL servers”. Then one can choose between a single and a flexible server. Here the cheaper option, a single server, was chosen. In figure 17 possible settings can be seen. The subscription is for the billing, the resource group is for organizing different resources: if they are in one resource group they have the same permissions and life cycle. Next the servername, location and version can be chosen and if it is create from a backup. Here no data source is chosen. Last an administrator account is set. As additional settings data encryption and tags can be set, none of them was used here. Finally it can be created.



**Create MySQL server**  
Microsoft

**Project details**  
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ  ✓  
Resource group \* ⓘ  ✓  
[Create new](#)

**Server details**  
Enter required settings for this server, including picking a location and configuring the compute and storage resources.

Server name \* ⓘ  ✓  
Data source \* ⓘ  None  Backup  
Location \* ⓘ  ✓  
Version \* ⓘ  ✓  
Compute + storage ⓘ **General Purpose**  
4 vCores, 100 GB storage  
[Configure server](#)

**Administrator account**

Admin username \* ⓘ  ✓  
Password \* ⓘ

Figure 17: Mask for creating the database instance.

After the creation, the firewall rules need to be modified so that a connection can be established. The easiest way is to choose “add current client IP address” under “Connection security”, of course also IP ranges can be added. Also the flag “`log_bin_trust_function_creators`” has to be set to one in the server parameters. Now a connection can be established, e.g. via MySQL-Workbench. The necessary connection information can be found in the server details as shown in figure 18. Alternatively there is also a section “Connection strings”, which shows the connection strings for different applications. Also a connection via the Azure-CLI can be established by using the following command, where the host and a user have to be given:

```
mysql --host=azureserversakila.mysql.database.azure.com
--user=tanjaadmin@azureserversakila.mysql.database.azure.com -p
```

From this point the import is straightforward, it can be done with the import option of MySQL-Workbench, which was connected to the created instance earlier. And because the

[▶ Start](#)
[✎ Reset password](#)
[↺ Restore](#)
[🗑 Delete](#)
[↺ Restart](#)
[❤ Feedback](#)

[^ Essentials](#)

Resource group ( <a href="#">change</a> )	: <a href="#">SakilaResGroup</a>	Server name	: azureserversakila.mysql.database.azure.com
Status	: Stopped	Server admin login name	: tanjaadmin@azureserversakila
Location	: France Central	MySQL version	: 5.7
Subscription ( <a href="#">change</a> )	: <a href="#">Free Trial</a>	Performance configuration	: General Purpose, 4 vCore(s), 100 GB
Subscription ID	: ed130248-1416-4653-95d1-e85c5f0f73f3	SSL enforce status	: <a href="#">ENABLED</a>
Tags ( <a href="#">change</a> )	: <a href="#">Click here to add tags</a>		

Figure 18: Server details of “azureserversakila.”

`log_bin_trust_function_creators` flag already is enabled, also triggers, stored functions and procedures can be imported.

Users can’t be created automatically in Azure. Either one uses MySQL-Workbench to create them or the `CREATE` statement. For getting the users and their permissions from the local database the `SELECT * FROM mysql.user` statement can be used.

## 4.2 Virtual Machines

Before a virtual machine can be uploaded to the cloud, it must be exported from the local VirtualBox. Here virtual appliances (ova-files) are used to transfer the virtual machine to GCP and AWS. The Open Virtualization Format 1.0 is used and within a short time the ova-file is created by VirtualBox. Azure is a special case, it only accepts VHD as import format. Also the virtual machines have to be prepared accordingly. This process is discussed in section 4.2.3.

### 4.2.1 To GCP

Google offers a precheck-tool to determine whether the virtual machine, which should be migrated to GCP, actually can be imported. This is quite comfortable, since it is much quicker to download and run the tool than trying an import, which might fail. The output of the precheck-tool can be seen in figure 19. There are some points (SHA2 Driver Singing Check and Powershell Check) which are skipped, because they just need to be checked on Windows systems. Also the operating system version is checked and although it says in the documentation that Ubuntu 18.04 LTS can be imported to GCP, the precheck-tool throws the following error: “FATAL: version:18.04 not supported”. Nevertheless it was tried to import Ubuntu 18.04 LTS but it didn’t work and the following error message was shown: “Could not fetch resource: - Quota ‘SSD\_TOTAL\_GB’ exceeded”. However, checking the according quotas shows, that the peak-usage was actually less than 50% of the limit. That’s why Ubuntu 16.04 is used for this experiment. Next the disk is checked, it needs to have grub as boot loader. The SSH Check verifies, if SSH is running on port 22. If not this can cause trouble because the gcloud CLI connects to this port.

After the precheck-tool ran successfully, the ova-file can be created and uploaded to the cloud storage of GCP. Then the Cloud-SDK has to be installed on the local computer and now there is just one command left to import the virtual machine:

```
gcloud compute instances import ubuntu16gcp
--os=ubuntu-1604
```

```

ubuntu@ubuntu1604: ~/Desktop
ubuntu@ubuntu1604:~$ cd Desktop
ubuntu@ubuntu1604:~/Desktop$ sudo ./import_precheck
#####
# SHA2 Driver Signing Check -- SKIPPED #
#####
* INFO: Only applicable on Windows 2008 systems.
#####
# OS Version Check -- PASSED #
#####
* INFO: OSInfo: &{Hostname:ubuntu1604 LongName:Ubuntu 16.04.7 LTS ShortName:ubuntu Version:16.04 KernelVersion:#118-16.04.1-Ubuntu SMP Sat Sep 5 23:35:06 UTC 2020 KernelRelease:4.15.0-117-generic Architecture:x86_64}
#####
# Powershell Check -- SKIPPED #
#####
* INFO: Not applicable on non-Windows systems.
#####
# Disks Check -- PASSED #
#####
* INFO: `lsblk -l` results:
NAME           MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sr0              11:0    1 1024M  0 rom
sda              8:0     0  100G  0 disk
|-sda2           8:2     0    1K  0 part
|-sda5           8:5     0  99.3G  0 part
|_-ubuntu--vg-swap_1 253:1   0   976M  0 lvm  [SWAP]
|_-ubuntu--vg-root_ 253:0   0  98.3G  0 lvm  /
|-sda1           8:1     0   731M  0 part /boot

* WARN: translation only supports single block device. Will attempt to determine if translation is possible...
* INFO: root filesystem on device "sda" partition "ubuntu--vg-root"
* INFO: root filesystem device is MBR.
* INFO: GRUB found in root filesystem device MBR
#####
# SSH Check -- PASSED #
#####
* INFO: SSH (OpenSSH) detected on port 22.
ubuntu@ubuntu1604:~/Desktop$

```

Figure 19: Output of the precheck-tool.

```
--source-uri=gs://vmupload/Ubuntu_16.04_local.ova
```

This statement creates an instances which is called `ubuntu16gcp`. The flag `--os` defines the operating system version and `--source` gives the path to the cloud storage, where the appliance is stored. This statement has to be executed for every instance which should be imported. It takes about 25 minutes to create the instance and import the virtual machine. After that, a connection can be established with the cloud shell.

A big disadvantage is, that the GUI cannot be displayed in the cloud. If one needs a GUI there is a workaround necessary. GCP recommends Chrome Remote Desktop for Linux, but there are also some alternatives. For Chrome Remote Desktop a script to automate the installation is offered by GCP, which can be found in the documentation. Another option is to use VNC and connect that with the computing engine. VNC is a graphical desktop sharing system and can be configured so it provides a GUI for the GCP compute engine. For that the VNC server has to be installed on the computing engine and VNC viewer on the local computer.

#### 4.2.2 To AWS

Different than GCP, AWS does not offer a precheck-tool. So the operating system version and other prerequisites have to be checked manually. After this and after installing the AWS-CLI, the ova-file can be uploaded to a S3 bucket. This can be done with the following statement: `aws s3 cp ubuntu16local.ova s3://uploadvm1`. The upload takes some time and only worked from Ubuntu; when trying to start the upload on MacOS an error, connected to the multipart-upload was thrown. It also didn't work without the multipart-upload, so Ubuntu 18.04 LTS was used to perform the upload. Also the drag-and-drop option of the AWS-Console didn't work, although it should for objects of



this size.

Next a so called IAM (identity and access management) role has to be created. This is necessary to have the rights for the import and can be done with the AWS-CLI. For that two json-files are needed: trust-policy.json and role-policy.json. Their contents can be seen in figure 20 and 21. A template for the files can be found in the AWS documentation and just the customized information needs to be filled in. They define the used services and resource locations. To create the role the following statements are used:

```
aws iam create-role --role-name vmimport
--assume-role-policy-document "file://trust-policy.json"
```

```
aws iam put-role-policy --role-name vmimport --policy-name vmimport
--policy-document "file://role-policy.json"
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": { "Service": "vmie.amazonaws.com" },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:Externalid": "vmimport"
        }
      }
    }
  ]
}
```

Figure 20: Contents of trust-policy.json.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::uploadvm1",
        "arn:aws:s3:::uploadvm1/*"
      ]
    }
  ]
}
```

Figure 21: Contents of role-policy.json.

For the actual import another json file is needed: containers.json (see figure 22). And with the statement “aws ec2 import-image -description "Local Ubuntu 16.04 LTS" -disk-containers "file://containers.json"”, the image is imported. Unfortunately this does not show the progress of the image import. To see the progress another statement

has to be used: “`aws ec2 describe-import-image-tasks --import-task-ids import-ami-1234567890abcdef0`”. This shows the current state, if one wants to know the state a few minutes later the statement has to be executed again.

```
[
  {
    "Description": "Ubuntu 16.04 LTS",
    "Format": "ova",
    "UserBucket": {
      "S3Bucket": "uploadvm1",
      "S3Key": "Ubuntu_16.04_local.ova"
    }
  }
]
```

Figure 22: Contents of containers.json.

Finally the instance can be launched. This can be done in the AWS-Console under “Create Instance”. There the uploaded image can be selected, some settings can be edited but for this experiment the default settings are just fine. Then the instance can be launched. The VirtualBox guest additions didn’t start but all data is still there. One can connect over SSH with the following statement: `ssh -i "ubuntuaws2.pem" ubuntu@ec2-18-193-222-141.eu-central-1.compute.amazonaws.com`.

If a GUI is needed, AWS recommends to install MATE Desktop environment. For that the MATE desktop environment and Tiger VNC have to be installed on the virtual machine. Next a VNC client has to be installed on the local computer, then the two can be connected.

The whole import process worked without problems for Ubuntu 16.04 but the import for Ubuntu 18.04 quit with the following error message: “ClientError: Unsupported kernel version”. Locally Ubuntu 18.04 was running without problems in VirtualBox. It is disappointing that AWS just lists the operating system versions, which can be imported (there Ubuntu 18.04 is included) but no kernel versions and then throws such an error.

### 4.2.3 To Azure

While all cloud providers have strict requirements and restrictions for the virtual machines, which can be imported to their cloud platforms, Azure plays in his own league. Before a virtual machine can be imported it has to be prepared accordingly. That means the image has to be configured to use Ubuntu’s Azure repository, update it to use the latest Azure-tailored kernel and install Azure Linux tools, which include Hyper-V dependencies. Also cloud-init has to be installed and configured to use Azure as data source. This still can be done in VirtualBox but would work better in Hyper-V Manager.

Next, the virtual machine image needs to be converted to VHD or exported as VHD, because that’s the only format that is possible to import in Azure. A VHD-file can be exported from VirtualBox, nevertheless the size of the VHD can’t be defined. This brings up the next problem, when trying to upload the VHD-file: “BadRequestError: (BadRequest) The upload size in bytes <size> - 512 bytes for the VHD footer (<size> in this case) must be a multiple of MiB”. There are not many options to resize the VHD. One would be Hyper-V, but it can just be installed on Windows. Another option, suggested in the Azure documentation, is the Convert-VHD cmdlet but this also doesn’t work without the Hyper-V Module for Windows. So the experiment was stopped at this point, because

the needed tools were not available for Linux or Mac OS.

As a conclusion, what can be said about the import of a virtual machine to Azure? Definitely that it is not platform independent and that it can't be done without Hyper-V Manager or other Microsoft related products.

## 4.3 Docker

For the start of this experiment Docker is set up in the local environment, which is a virtual machine with Ubuntu 18.04. The container used here also contains ubuntu. But when comparing the uploaded images from docker and the virtual machine, one can see that the docker image is way smaller (about 80MB) than the virtual machine image (2GB or more). This is because docker does not virtualize a whole operating system.

As far as known there are no restrictions to which images can be uploaded to the registries. Only in Azure the Ubuntu container didn't run. After an image is created it usually needs to be tagged accordingly, before it can be pushed to a repository. But this and all other necessary steps to move a container to the cloud will be explained in this chapter for each cloud provider.

### 4.3.1 To GCP

To use Docker in GCP, an image has to be uploaded and deployed there. Before the upload can happen the image has to be tagged, this means adding the registry name. The statement has the following scheme: `docker tag [SOURCE_IMAGE] [HOSTNAME]/[PROJECT-ID]/[IMAGE]`. After this, the image can be pushed to the GCP registry. The registry doesn't have to be public, just the user who pushes the image there has to have the according rights. The two used statements for this process can be seen in figure 23. After running those statements the image is in the Container-Registry. There are three options to deploy the image: via Cloud Run, GKE (Google Kubernetes Engine) or GCE (Google Computing Engine). Cloud Run is a server less solution, which doesn't support all features that GKE supports. Nevertheless Cloud Run offers the most flexible pricing, since costs just arise when the service is used, not when the container is running. This means one just pays for a web application when a HTTP request is coming in but not for "only running it". Here the third option "deploy to GCE" was chosen. For this an instance has to be created, which can be done in the GCP-Console. Most settings are self-explaining, like choosing a region, zone and machine type. But one has to know to check the boxes for "Allocate a buffer for STDIN" and "Allocate a pseudo-TTY". without these settings the container will not start. The settings are the equivalent for the `-it` flag when using docker. "Allocate a buffer for STDIN" is `-i` and adds a buffer for STDIN, without that reading from STDIN will just give back EOF and no input to the terminal will be possible. Together `-i` and `-t` create sort of a terminal, which can be used for interacting with the container. Now the container can be used with CloudShell or with Cloud SDK.

```
gcloud beta compute ssh -zone "us-central1-a" "instancedocker1" -project  
"planar-osprey-277208"
```

is the statement which is used to connect to the instance; it can be found in the console when clicking on "View gcloud command". And with `docker start -i 1081b00d5405` the container can be started to work with it. 1081b00d5405 is the container-ID, the running container can be seen in figure 24.

```
osboxes@osboxes:~$ docker tag ubuntu gcr.io/planar-osprey-277208/ubuntu
osboxes@osboxes:~$ docker push gcr.io/planar-osprey-277208/ubuntu
The push refers to repository [gcr.io/planar-osprey-277208/ubuntu]
cc9d18e90faa: Layer already exists
0c2689e3f920: Layer already exists
47dde53750b4: Layer already exists
latest: digest: sha256:1d7b639619bdca2d008eca2d5293e3c43ff84cbee597ff76de3b7a7de3e84956 size: 943
osboxes@osboxes:~$
```

Figure 23: Tag a docker image and push it to the GCP repository.

```
osboxes@osboxes:~$ gcloud beta compute ssh --zone "us-central1-a" "dockergcp2" --project "canvas-adviser-294119"
#####[ Welcome ]#####
# You have logged in to the guest OS. #
# To access your containers use 'docker attach' command #
#####

osboxes@dockergcp2 ~ $ docker container ls
CONTAINER ID        IMAGE                                     COMMAND                  CREATED
9c6320abed27       gcr.io/stackdriver-agents/stackdriver-logging-agent:0.2-1.5.33-1-1  "/entrypoint.sh /usr..." 13 minutes ago
gent
1081b00d5405       gcr.io/canvas-adviser-294119/ubuntu:latest  "/bin/bash"              24 minutes ago
osboxes@dockergcp2 ~ $ docker start -i 1081b00d5405
root@dockergcp2:/#
```

Figure 24: Connecting to the container.

### 4.3.2 To AWS

Before moving a docker container to AWS, first a repository has to be created. In Amazon this is called ECR (Elastic Container Registry). This repository can be created in the AWS-Console under “Create Repository”. Here the name for the repository has to be chosen and it is advisable to enable tag-immutability. This ensures that no image is overwritten, if images with the same tags are pushed to the repository. Another option is to scan for vulnerabilities automatically while an image is pushed to ECR but this can also be left out or the scan can be started manually after the upload.

Now that the repository is created, the push commands recommended by AWS can be viewed. This is shown in figure 25. First an authentication token is needed, then the image needs to be tagged and finally it can be pushed to the repository.

To run the container the image needs to be deployed to an EC2 (Elastic Compute Cloud) instance. For this, so called cluster and task definitions have to be created in the AWS-Console. A cluster is a logical group of tasks or services. The task definition is needed to run the container in AWS; it defines things like networking mode, logging configuration, CPU usage and many more.

After the cluster and task definitions are created The instance can be found on the EC2-Dashboard under the point “Instance”. It is running and one can connect and work with it now. The connection can be established via SSH and a previously generated key pair. For this SSH has to be added in the security group. Then the following statement can be used to connect: `ssh -i "awsdocker.pem" ec2-user@ec2-52-57-114-32.eu-central-1.compute.amazonaws.com`

### Push commands for ubuntudocker ✕

**Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting Started with Amazon ECR](#).**

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry Authentication](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry.  
Use the AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
```

Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:

```
docker build -t ubuntudocker .
```
3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag ubuntudocker:latest 300513033571.dkr.ecr.us-east-1.amazonaws.com/ubuntudocker:latest
```
4. Run the following command to push this image to your newly created AWS repository:

```
docker push 300513033571.dkr.ecr.us-east-1.amazonaws.com/ubuntudocker:latest
```

Close

Figure 25: Suggested commands from AWS, to push an image to ECR.

### 4.3.3 To Azure

The first thing to do in Azure is creating a resource group for this project so that the billing can be done correctly. Next a container registry needs to be created. This can be done in the cloud-console. Some information about the project, like resource group, location and name of the registry have to be entered. After this, one can log in from the Azure-CLI with the following command: “`az acr login -n <registryname>`”. The image now needs to be tagged and then it can be pushed to the ACR (Azure Container Registry), the commands for that can be seen in figure 26.

```
osboxes@osboxes:~$ docker tag ubuntu azureregistry11.azurecr.io/ubuntu:loc:v1
osboxes@osboxes:~$ docker push azureregistry11.azurecr.io/ubuntu:loc:v1
The push refers to repository [azureregistry11.azurecr.io/ubuntu:loc]
cc9d18e90faa: Pushed
0c2689e3f920: Pushed
47dde53750b4: Pushed
```

Figure 26: Tag the image and push it to ACR.

Once the image is pushed to ACR it can be deployed to an Azure Container Instance. For this, the admin user has to be enabled in the registry and a container instance has to be created. Now the instance can be started. With the ubuntu image used here, the container shuts down immediately after the start, because there is no running process in

it. Also the `-it` flag doesn't help. If an image with `nginx` running is used the container keeps running and the web page can be visited with the provided IP-address.

## 4.4 Storage

File storage in the cloud is not just relevant for companies but also a lot of private users work with it. Dropbox and Google Drive are two examples of cloud storage for private users and small storage units are usually for free. Companies on the other hand have more data than private users and therefore need bigger storage units for which they have to pay. As will be seen, the upload functions of GCP, AWS and Azure differ from Dropbox, which uses drag-and-drop. This is the case because larger amounts of data have to be uploaded and also access control is needed. So, each cloud platform offers a kind of access control, but it is not possible to set access rights during the import, they have to be set afterwards manually.

### 4.4.1 To GCP

The Google Cloud Storage is structured very clearly. Different buckets can be created in which data can be saved. This data can be uploaded via drag-and-drop or a "upload"-button. In this way single files can be uploaded, as well as a whole structure of folders. Also `gsutil` can be used for the upload. The statement looks as follows: "`gsutil cp <filename> gs://upload`". So migrating data to GCP is easy but takes some time for big amounts of data. So called "Hold" can be set for every object, this means that this object can't be deleted until a certain date passed or until a certain event, like another document was deleted or updated, happened. Of course also access control can be managed, this happens with the help of access control lists. The access rights can be set for buckets, folders or single objects. These settings need to be made after the upload and can't be imported.

### 4.4.2 To AWS

The upload to AWS happens similar to GCP, first a bucket has to be created and then files and folders can be uploaded either via drag-and-drop or the "upload"-button. For objects which are larger than 160GB the AWS-CLI (AWS Command line interface) or AWS-SDK has to be used. Nevertheless the upload of the virtual machine image also needed to be done with the AWS-CLI. The command is "`aws s3 cp <filename> s3://vmupload1`". So apparently the AWS-Console does not work well with large objects. In AWS there is a upload-page, which displays the data that is going to be uploaded. Moreover the settings which can be made are shown. This includes access control, storage class (there are different classes depending on how often the data is used), encryption, versioning and tags. After checking all these setting the "upload"-button can be pushed. Then a status page appears and finally one is led back to the bucket and the uploaded data. There the data can be viewed and the access rights to buckets, folders or single objects can be changed with the help of an access control list.

### 4.4.3 To Azure

To use the Azure Blob Storage, first a Storage Account has to be created. This can be done in the cloud-console under “Storage Accounts”. For this account the subscription and resource group, the account name, a location and the kind of storage, which will be used has to be chosen. Here the V2 Storage for general purpose was chosen. There blob containers, file shares, queues and tables can be used. If the cloud-console is used, only single files can be uploaded, that’s why the Storage Explorer has to be downloaded and a connect statement to the password manager has to be executed. In Ubuntu this can be done with the following statements:

```
sudo apt install snapd
sudo snap install storage-explorer
snap connect storage-explorer:password-manager-service
:password-manager-service
```

After this, the storage explorer can be opened. First it has to be connected with the Azure account. The window for signing in appears automatically when opening the storage explorer. Now files and folders can be uploaded easily by clicking the “upload”-button. For access control there are the following options: Full public read access (everything can be read via anonymous requests), no public read access (just the container owner can read) and public read access for blobs only (blob data can be read by everyone, other container data not).

## 5 Migrating a system from a cloud platform to a local system

This chapter aims to clarify how the export from a cloud platform works and if a fully functional system can be exported and set up locally again. For this, first a new system is set up in the cloud. Later this system can be exported and set up in the local environment. As before, this is done for all parts of the test system. This can be seen as export, but also as the first step for a indirect migration between cloud platforms.

### 5.1 Databases

As in chapter 4, the first part of the test system to be examined is the database. For this the test database Sakila is created on every examined cloud platform. Besides the data, table structure, triggers, stored functions and procedures are exported from the different cloud platforms and set up in the local system. The local setup is performed by importing the mysqldump with the help of MySQL-Workbench to the local system.

#### 5.1.1 From GCP

Exporting the structure and data of the database is not difficult. GCP offers an export option and with just a few clicks the database can be exported as a CSV- or SQL-file. The CSV-file just contains the information of a SELECT query. The export mask can be seen in figure 27. First a file format has to be chosen, in this case it is SQL, so also the statements for creating tables and views will be exported. Then one has to select the database which should be exported, here it is Sakila. Finally the destination for the export has to be chosen. It can not be directly downloaded but has to be stored on the cloud storage in a bucket first. With these settings an SQL-file will be saved in the according bucket.

Unfortunately with the help of this export mask triggers, stored procedures and functions can't be exported. If they should be exported too, mysqldump needs to be used. To do so a MySQL-client is needed on a local computer or in an computing engine. Also the connection over public IP-addresses has to be configured for the instance in GCP. This can be done in the connection-settings of the instance. There the IP-address of the client has to be added to the authorised networks. Now a connection can be established from this client to the GCP instance. For the dump the following statement has been used:

```
mysqldump --databases sakila -h <mysql-server-ip> -u root  
-p --hex-blob --single-transaction --set-gtid-purged=OFF  
--default-character-set=utf8mb4  
--routines > sqldumpfile_sakila_fromGCP.sql
```

There <mysql-server-ip> is the public IP-address of the instance, to which the user wants to connect to export Sakila. `--routines` has to be used, otherwise the stored procedures and functions won't be exported. `--triggers` doesn't need to be included, because triggers are exported automatically. `--hex-blob` makes sure the binary strings are dumped in hexadecimal format. `--set-gtid-purged=OFF` does exclude information about executed transactions. After the statement was executed the dump can be found in a file named "sqldumpfile\_sakila\_fromGCP.sql". This file is saved locally in the directory from which the MySQL client was started.



### Source

Choose the format for your export and the data that you'd like to export from this instance. [Learn more](#)

#### File format

- SQL  
A plain text file with a sequence of SQL commands, like the output of mysqldump
- CSV  
Exports a plain text file with one line per row and comma-separated fields. Requires SQL SELECT query.

#### Data to export

- Entire instance
- One or more databases in this instance

Databases

- Offload export to a temporary instance  
Makes your export serverless to reduce strain on the source instance, allowing you to perform other operations while the export is in progress. Affects cost. [Learn more](#)

### Destination

Choose a Cloud Storage location to export into. Make sure you have the required permissions. [Learn more](#)

databases\_sakila/Cloud\_SQL\_Export\_2020-10-23 (14:23:29).sql BROWSE

Browse for a Cloud Storage location or enter the path to one

**i** When you export, a Cloud SQL service account will be granted write access to the selected bucket, which will be reflected in your permissions.

Figure 27: Mask for exporting a database from GCP.

Having MySQL Workbench installed, the import to the local system is no problem. Under “Server”, the option “Data import” can be found and then the file has to be selected and it will be imported.

It gets more tricky when the users should be transferred back to the local system. For this another export has to be done. This time the database “mysql” is exported. In the dump the following section can be found:

```
LOCK TABLES 'user' WRITE;
/*!40000 ALTER TABLE 'user' DISABLE KEYS */;
INSERT INTO 'user' VALUES
('%', 'root', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'mysql_native_password', '*81F5E21E35407D884A6CD4A731AEBFB6AF209E1B',
'N', '2020-10-22 12:48:50', NULL, 'N'),
('%', 'tanja', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
```



definers are not exported. Definers give the user who created the trigger, stored function or procedure. But in the new database this user doesn't exist, so there would be an error when importing the definers. Also importing the user first would not work, because the definers have the following format: <username>@<databasename> and here are different names for the databases used. Finally the exported SQL-file can be imported to the local system with the help of MySQL-Workbench.

In Azure the "mysql" database can't be exported, even the adminuser doesn't have access to it. So to get the user data, the statement "SELECT \* FROM mysql.user" is used. In this way the user data (name and permissions) can be exported, then the users have to be created manually in the local database.

## 5.2 Virtual Machines

When importing and exporting virtual machines one has to distinguish, whether an instance or an image should be moved. The instance is a dynamic system, a running virtual machine, which is usually started from an image. Here images and virtual appliances are exported from the different cloud platforms, depending on what is possible.

### 5.2.1 From GCP

The starting point here is the running instance. An image can be created based on a snapshot, another image, a virtual disk or a cloud storage file. This can be done in the cloud-console in the "Image"-section with the "Create Image"-button. After this happened the created image is listed with other image templates from GCP in the "Image"-section. There another button for the export can be found. The export needed almost 14 minutes, as can be seen in figure 28. Export means, saving the image as .vmdk to the cloud storage. From there it can be downloaded and then imported to the local virtual box again.

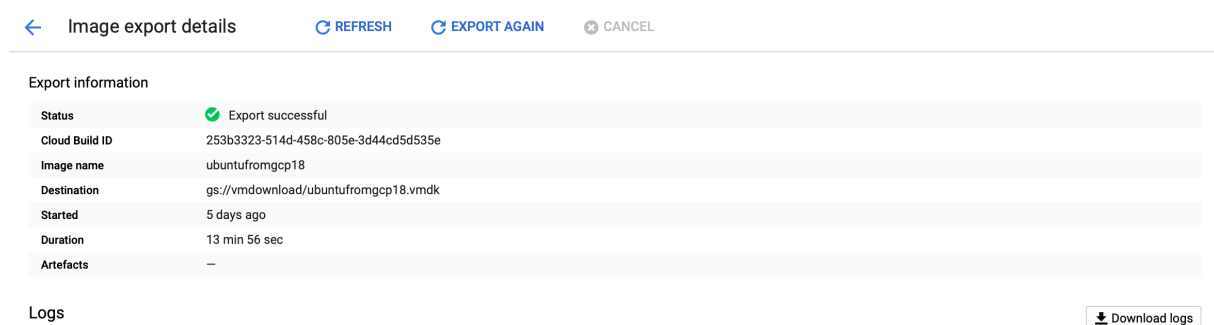


Figure 28: Export information screen.

Interesting is, that the VirtualBox guest additions didn't show when the virtual machine was running in GCP. Nevertheless they weren't deleted, because now, when the machine is running in a local environment again, the guest additions are there as well as the original GUI, which was also not able to run in the cloud environment.

A new user was created by GCP, this user was also exported. The username is `t_fraundorfer`, which is the login for GCP. To use this user in the local system, a password had to be set in the cloud instance before the export (`sudo passwd t_fraundorfer`). After setting

the local machine up, this user was used to change the password of the original ubuntu user, because the password of it didn't work anymore. After the password change both users could be used in the local virtual machine.

Also at the first login a "System program problem detected" error message appeared. It disappeared after updating the system, deleting the crash report file and restarting the machine. So the export for Ubuntu 16.04 LTS worked but has not been really smooth; there were some unexpected and inexplicable errors.

The export of Ubuntu 18.04 LTS, on the other hand, worked without problems. It was the same procedure, a user had to be created in the running cloud instance and the image was exported. After setting everything up locally, the user could be used for the login and everything worked without errors.

### 5.2.2 From AWS

The AWS-CLI is still installed from the import process, so there is just one thing left to prepare for the export and this is editing the access control list for the S3 bucket. Depending on the region in which the exported instance is running, an entry has to be added to the access control list of the target S3 bucket. From there the export is pretty straightforward. Only one statement and a json-file are needed. The very creative name (suggested by AWS) of the json-file is "file.json". It defines the format and target bucket of the export and can be seen in figure 29.

```
{  
  "ContainerFormat": "ova",  
  "DiskImageFormat": "VMDK",  
  "S3Bucket": "exportvm14",  
  "S3Prefix": "vms/"  
}
```

Figure 29: Json-file for exporting an instance.

```
aws ec2 create-instance-export-task -instance-id i-08eb724ac56d380d4  
-target-environment vmware -export-to-s3-task file://file.json
```

The statement above is used to start the export. It is comfortable that just one step is needed, no AMI (Amazon machine image) has to be created first. The export can be performed from a running instance and took a few hours to finish. It created a OVA-file in the according S3 Bucket and a snapshot of the instance could be found in the EC2 Dashboard.

Similar to the import, the status of the export can be monitored with the following statement: "aws ec2 describe-export-tasks -export-task-ids export-i-0e681c2c228d55741". This is no continuous information, but the statement has to be executed every time the state of the export needs to be known. It also doesn't show a progress, but states such as active, deleting, deleted (export is cancelled), updating (status is updated) and completed. The export information can be seen in figure 30. Finally a file named "export-i-0e681c2c228d55741.ova" is in the according S3 bucket. This can be downloaded and set up in a local system. After the appliance was imported to the virtual

box it started without any problems, this worked well for both Ubuntu 16.04 LTS and Ubuntu 18.04 LTS.

```

osboxes@osboxes:~/Desktop/vm$ aws ec2 create-instance-export-task --instance-id i-08eb724ac56d380d4
-----
|                               CreateInstanceExportTask                               |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                               ExportTask                                             |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                               ExportTaskId                                         | State | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| export-i-0e681c2c228d55741 | active |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                               ExportToS3Task                                         |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ContainerFormat | DiskImageFormat | S3Bucket | S3Key | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ova              | vmdk              | exportvm14 | vms/export-i-0e681c2c228d55741.ova |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                               InstanceExportDetails                                   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|                               InstanceId                                             | TargetEnvironment | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| i-08eb724ac56d380d4 | vmware |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
  
```

Figure 30: Export status information.

### 5.2.3 From Azure

Exporting a virtual machine from Azure is not complicated at all. After stopping the machine, a download link can be created under “Export Disk”. To create the link, only the expiry time has to be set. After this the VHD-file can be download from the link which is displayed (see figure 31) and imported to the local virtual box.

The following URL can be used to download the VHD file for this disk. Copy it and keep it secure, it will not be shown again.

<https://md-mp3fdr534hgx.blob.core.windows.net/bbtp1ktrks4z/abcd?sv=2018-03-28&sr=b&si=52b706d0-6e31-42b3-84e8-84cc29eaca08&sig=wAR14lgTE7MWYbdq9KvHj4aW7qXbCcFZnr...>

[Download the VHD file](#)

A SAS URL has been generated for this disk for export. While in this state, it can't be edited or attached to a running virtual machine. You can cancel the export by clicking the button below. This will revoke the SAS URL, and may cancel any in-progress transfers if the disk is currently being downloaded to another location.

[Cancel export](#)

Figure 31: Download link for the VHD-file.

The VHD-file (32.31GB) is way bigger than the VMDK-file downloaded from GCP (approximately 2GB) and the OVA-file downloaded from AWS (approximately 1GB), it took quite a while. But once it was downloaded everything started. Only some commands couldn't be executed due to a “unable to resolve host” - error. To solve this in /etc/hosts the line “127.0.0.1 ubuntuazure” needed to be added. This gives the local host name. Again, there was no difference in exporting Ubuntu 16.04 LTS and Ubuntu 18.04 LTS.

## 5.3 Docker

There are various reasons to run docker container in a local environment: they are small and therefore don't need much storage. Moreover Docker offers tools for Windows, Mac and Linux based operating systems, which are easy to install, to run and to work with.

The downloaded containers can be run on all operating systems on which Docker Desktop can be installed and on Linux, where the Docker Engine needs to be installed.

### 5.3.1 From GCP

The starting point for the export is a running container in an GCE instance. First an image has to be created from the instance. This happens with the statement: “`docker commit [ContainerID] dockergcp`”. Next, one has to authenticate to the container registry via a token and then the image has to be tagged, in the same way as in section 4.3.1. After this it can be pushed to the registry. The statements for tagging and pushing the image look as follows:

```
docker tag dockerubuntugcp2 gcr.io/canvas-adviser-294119/ubuntugcp
docker push gcr.io/canvas-adviser-294119/ubuntugcp
```

canvas-adviser-294119 is the GCP project name and gcr.io is the hostname, which gives the location of the image. In GCP each resource has to be allocated to a project for billing. The project name can be freely chosen or the suggestion of GCP can be accepted. For getting a docker image from the container registry of GCP only one statement is necessary and this statement is even given by GCP. When looking in the Container-Registry a click on the wanted container and the according three dots shows the option “show pull command”, this suggestion can be seen in figure 32. The according statement has to be copied to a terminal (of course docker and the Cloud-SDK have to be installed) and executed and then a container can be started based on this image.

#### Pull command

Use this command to pull the image by tag, [run in Cloud Shell](#)

```
$ docker pull gcr.io/planar-osprey-277208/ubuntu:latest
```

Use this command to pull the image by digest, [run in Cloud Shell](#)

```
$ docker pull gcr.io/planar-osprey-277208/ubuntu@sha256:1d7b639619bdca2d008eca2d5293e3c43ff84cbee597ff76de3b7a7de3e84956
```

[Container Registry reference](#)

Figure 32: Suggested pull command from GCP.

### 5.3.2 From AWS

Having a container running in an EC2 instance, one first needs to create an image by using `docker commit`. The authentication to ECR happens via a token and then the image can be tagged and pushed to the ECR Registry with the following statements:

```
docker tag dockerubuntuaws 300513033571.dkr.ecr.eu-central-1.
amazonaws.com/awsrepexport:latest
docker push 300513033571.dkr.ecr.eu-central-1.amazonaws.com/
awsrepexport:latest
```

For pushing the image to the registry, the user needs the right permissions. That's why the policy of the registry needs to be set accordingly. After this the image can easily be pulled from the registry to the local machine by using the following command: `docker pull 300513033571.dkr.ecr.eu-central-1.amazonaws.com/awsrepxport:latest`

### 5.3.3 From Azure

In Azure one can create a container instance from a docker image but this instance then can't be committed to an image again. So for the export here the starting point needs to be the image stored in ACR (Azure container registry). To export it from there not much needs to be done. Only permissions to execute the following pull command are needed: `“docker pull azuredockerprem.azurecr.io/ubuntu1604”`. So the image can be pulled to the local machine, where the image can be started.

## 5.4 Storage

Sometimes the confidentiality level of data changes and that makes it necessary to think about the data storage again. Critical data should not be stored in the cloud but must be stored on a secure server on the premises of the company. That's why it is examined here how data, which is stored in buckets, can be transferred from a cloud platform to a local system. In all three cloud platforms the data could be exported but the access rights couldn't.

### 5.4.1 From GCP

Downloading one object in GCP can be done in the cloud-console with the “Download”-button. This is quite impractical for large numbers of objects. So if more objects should be downloaded at once, e.g. a directory or bucket, the Cloud-SDK should be used. The statement `“gsutil cp -r gs://storage localDest”` downloads the whole bucket to the local system. If just parts should be downloaded, the desired objects can be marked in the cloud-console and when the download button is hit, the needed statement is displayed. To download a whole directory tree `-r` needs to be included in the statement. If just a subdirectory should be downloaded, the download location has to be appended to the bucket in the following way: `gs://storage/downloadfolder`.

### 5.4.2 From AWS

Downloading an object from AWS is not difficult, it can be done by selecting the object and then choosing “Download” in the menu. Nevertheless when there is a folder or more than one object is selected, the download-option in the cloud-console is not available. In this case the AWS-CLI has to be used. The command for downloading a whole bucket is the following: `“aws s3 sync s3:<source_bucket> <local_destination>”`. Also parts of the bucket can be downloaded by just appending the directory to the bucket.

### 5.4.3 From Azure

With the help of the storage explorer downloading objects is easy. After selecting the data which should be downloaded and clicking the “Download”-button the data is saved to the chosen folder. In the Console only single files can be downloaded.

## 6 Migration between cloud platforms

There are various reasons to change the cloud provider but thinking about migrating systems may let one hesitate to switch contracts. In this chapter it will be examined which knowledge and tools are necessary for moving systems between two cloud platforms. There are two options how this can be done. First the direct one, this means the data gets transferred directly from one platform to the other. The second one, is the indirect one. Here, the system first gets exported from the source cloud platform to the local system and then in a second step, it is imported to the destination cloud platform.

### 6.1 Direct – from cloud platform to cloud platform

Not all systems can be migrated directly between all cloud platforms. Usually there are special tools necessary to perform a direct migration. Unfortunately not all cloud platforms offer to import systems from other cloud platforms. Sometimes just the direct import from one certain cloud platform is possible. Sometimes the tools are more extensive and imports from various cloud platforms are possible. Sometimes a tool even offers a direct export option to other cloud platforms. So table 1 gives an overview of which direct transfers are possible and those will be discussed in this chapter.

Table 1: Possible direct transfers. For the database a direct user transfer is not possible, so here only the transfer of the structure, data, views, triggers, stored functions and procedures is examined.

From	To	Databases	Virtual machines	Docker container	Storage
AWS	GCP	yes	yes	yes	yes
Azure	GCP	yes	yes	no	yes
GCP	AWS	no	yes	yes	yes
Azure	AWS	no	yes	no	no
GCP	Azure	partly	no	yes	no
AWS	Azure	yes	yes	yes	yes

#### 6.1.1 Databases

Moving databases directly between the three cloud platforms is possible with an `mysqldump`, which is piped into the other database. This works for the database structure, the data, views and depending on the platform, also triggers, stored functions and procedures. Unfortunately a direct migration doesn't work for any users.

First, the platform where the database should be imported, needs to be prepared. This means for all three platforms, creating a database instance, enabling the `log_bin_trust_function_creators` flag and modifying the connection settings, so that a connection to the database can be established. The platform from which the export happens, doesn't need to be prepared in any other way than having a database, ready for the export.

When doing the `mysqldump`, definers are dumped too. To remove them, the dump is modified with "sed". An example of the full statement for the `mysqldump` looks as follows. In this example the database is moved from AWS to Azure.



```

mysqldump -h sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com
-u admin
-p<passwordAWS>
--port=3306
--single-transaction
--routines --triggers --databases
sakilaAWS | sed -e 's/DEFINER[ ]*=[ ]*[~]**\*/\*/' |
sed -e 's/DEFINER[ ]*=[ ]*[~]**PROCEDURE/PROCEDURE/' |
sed -e 's/DEFINER[ ]*=[ ]*[~]**FUNCTION/FUNCTION/' |
mysql -h azureserversakila.mysql.database.azure.com
-u tanjaadmin@azureserversakila.mysql.database.azure.com
--port=3306
-p<passwordAzure>

```

In the statement above the two hosts and the ports are given, as well as users with root privileges. `--single-transaction` makes sure changes, which are done in the database during the `mysqldump`, are excluded from the `mysqldump`. In this way data consistency is ensured. The parts with “sed” in it remove the definers in front of triggers, stored functions and procedures. For other combinations only the host and user have to be changed to the values shown in table 2. The port is the same (3306) for each cloud platform.

 Table 2: Connection information needed for the `mysqldump`.

Cloud Platform	User	Host
GCP	root	35.224.129.230
AWS	admin	sakilaaws.cmdy23z5qdep.eu-central-1.rds.amazonaws.com
Azure	tanjaadmin	azureserversakila.mysql.database.azure.com

When importing or exporting a database from Azure, nothing special has to be considered. Nevertheless, some things have to be considered when exporting it from GCP or importing the database to AWS. In GCP, when a `mysqldump` is created the flag `--set-gtid-purged` needs to be used in the following way: `--set-gtid-purged=OFF`. It excludes information about executed transactions. Otherwise the `mysqldump` can’t be imported to the other cloud platforms.

When moving databases to AWS, already in section 4.1.2 was discovered that triggers, stored functions and procedures can’t be imported. This is also the case when a direct transfer is done. It doesn’t matter if Azure or GCP is the cloud platform from which the database is exported. That means that only the database structure, the data and views can be migrated to AWS.

### 6.1.2 Virtual Machines

Fortunately the cloud platforms offer migration assistants to directly move virtual machines from one cloud platform to the other. In GCP it’s called `Migrate for Compute engine`, in AWS the `AWS SMS` (server migration service) is used and in Azure, `Azure Migrate: Server Migration`. With the help of these migration assistants, virtual machines can be transferred from one cloud platform to the other one without noteworthy downtimes. This is possible, because everything gets set up in the new environment while the

old machines are still running. Only once the new system is ready, the actual switch is done and the old system can be turned off. Unfortunately the costs for the needed resources are mostly not covered by the free trial credits, which are offered by the cloud providers. So this subsection just gives a theoretical overview of the possibilities.

**AWS and Azure to GCP:** The tool offered to migrate virtual machines from AWS and Azure to GCP is the Cloud Migrate for Compute Engine, which was formerly known as Velostrata. In [25] an overview of the migration process can be found. Basically a VPN for a secure connection between the cloud platforms is needed. Also the Migrate for Compute Engine Manager has to be configured in GCP. A Migrate for Compute Engine Importer is deployed on the source cloud platform during the migration. Finally the Migrate for Compute Engine package has to be installed on the virtual machines, which should be migrated. This package reconfigures the virtual machines for GCP.

**Azure to AWS:** On the base of [5] a short overview will be given, about how the migration from Azure to AWS works. For this migration the SMS (server migration service) connector of AWS is used. So virtual machines from Azure can be imported to AWS but no virtual machines from GCP. The SMS connector needs to be deployed in Azure. It coordinates the migration similar to the replication appliance server used in Azure. This SMS connector then needs to be configured; for this a web interface is provided. After this is done, the virtual machines which should be migrated can already be seen in the AWS-console in the server migration service. Now a replication job can be created, which then can be executed and results in an AMI (Amazon machine image) in AWS. This machine image finally can be deployed to an instance.

**GCP and AWS to Azure:** To move virtual machines to Azure, Azure Migrate: Server Migration is used. How this works is described in [18]. On this basis the process will be outlined here. The migration tool guides one through the steps of the migration and can be used for both GCP and AWS. First an Azure migration project has to be created. Next, in GCP (or AWS) a Windows Server 2016 has to be set up and the according replication appliance software has to be downloaded and registered. For this registration a key can be downloaded from Azure. The windows sever is a so-called replication appliance server. It will not be migrated but will coordinate the whole migration. Moreover an agent has to be installed on every virtual machine which should be migrated, so the machines can be discovered by the replication appliance server. Now only some details are missing, such as the name of the replication appliance server, which of the discovered machines should be migrated, the virtual network in Azure to where the virtual machines should be replicated and the size of the imported virtual machines (this doesn't have to match with the original size). Finally the replication can be started. Once a replication is in Azure, it can be deployed by clicking on the "Migrate"-button.

### 6.1.3 Docker

Also for docker not all options of direct transfers between the three cloud platforms are possible. But the ones which are possible are discussed here. Direct means here, directly pushing the docker image to the container registry of the other cloud. Once the image

is in the container registry of the target cloud, it can be run as described in chapter 4.3. There were no unexpected problems for the scenarios in this sub-chapter.

**AWS to GCP:** To move a docker container directly from AWS to GCP, first the image needs to be tagged accordingly. But before it can be pushed, one has to authenticate to the GCP container registry. This happens via a token, which is valid for one hour. After this, the image can be pushed to the GCP container registry. Once it is there, it can be used e.g. deploy it to GCE. The necessary steps (authenticating, tagging and pushing) can be seen in figure 33.

```
[ec2-user@ip-172-31-22-11 ~]$ docker login -u oauth2accesstoken https://gcr.io
Password:
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[ec2-user@ip-172-31-22-11 ~]$ docker tag ubuntu gcr.io/canvas-adviser-294119/awsubuntu
[ec2-user@ip-172-31-22-11 ~]$ docker push gcr.io/canvas-adviser-294119/awsubuntu
The push refers to repository [gcr.io/canvas-adviser-294119/awsubuntu]
```

Figure 33: Authenticating to GCP container registry, tagging and pushing the image.

**GCP to AWS:** Moving a docker container directly from GCP to AWS happens quite similar to moving it from AWS to GCP. In the running container instance one has to create an image and authenticate to AWS ECR with a one time token. Then the image can be tagged and pushed. The necessary commands can be seen in figure 34.

```
osboxes@instance-1 ~ $ docker login --username AWS 300513033571.dkr.ecr.eu-central-1.amazonaws.com
Password:
WARNING! Your password will be stored unencrypted in /home/osboxes/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
osboxes@instance-1 ~ $ docker tag ubuntu gcp 300513033571.dkr.ecr.eu-central-1.amazonaws.com/ubuntu gcp
osboxes@instance-1 ~ $ docker push 300513033571.dkr.ecr.eu-central-1.amazonaws.com/ubuntu gcp
The push refers to repository [300513033571.dkr.ecr.eu-central-1.amazonaws.com/ubuntu gcp]
```

Figure 34: Authenticating to AWS ECR, tagging and pushing the image.

**GCP and AWS to Azure:** To move a docker container from GCP or AWS to Azure, in Azure a premium container registry is necessary. Otherwise it is not possible to authenticate from other platforms to the Azure container registry, because token authentication is only a preview in Azure and not available in the basic option of the container registry. Besides that, the usual steps are necessary for the transfer: create an image, tag it, authenticate to ACR and then push the image to the according repository. This is the same for AWS and GCP. The code for GCP can be seen in figure 35. The docker login command is provided by Azure when creating the token.

```

osboxes@instance-2 ~ $ docker login -u tokengcp -p Z+qI8Z344VJ7j0kmyC05NVia3NMMWouU azureregprem.azurecr.io
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /home/osboxes/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
osboxes@instance-2 ~ $ docker tag gcr.io/canvas-adviser-294119/ubuntu azureregprem.azurecr.io/gcpubuntu
osboxes@instance-2 ~ $ docker push azureregprem.azurecr.io/gcpubuntu
The push refers to repository [azureregprem.azurecr.io/gcpubuntu]

```

Figure 35: Authenticating to Azure ACR, tagging and pushing the image.

#### 6.1.4 Storage

Moving the data from one cloud storage to another one directly is possible for four out of six combinations and fortunately quite uncomplicated.

**AWS and Azure to GCP:** To transfer stored data from another cloud platform to GCP, the function “Data transfer” in GCP can be used. It imports the data from an AWS S3 bucket or an Azure container. For this the bucket or container name has to be given, also the access key for AWS or a service SAS (shared access signature) for Azure. The mask for importing objects from AWS can be seen in figure 36. Also the destination bucket in the GCP cloud storage needs to be chosen and finally the time for the transfer can be set. Once this transfer job is created it can be run and saved to be used later again.

**AWS to Azure:** To move data from AWS to Azure, the tool azcopy can be used. After downloading and installing it, first the access key for AWS has to be set. Then the copy command can be used to copy a file, folder or bucket to Azure. In this statement, first the S3 bucket has to be defined and then the destination in Azure. Additionally a service SAS is needed, which is given as a string and appended to the destination. --recursive has to be set true, so that all sub folders are copied too. The statements needed to copy a bucket from AWS to Azure look as follows:

```

export AWS_ACCESS_KEY_ID=AKIAI5T55C5CEN3H33FA
export AWS_SECRET_ACCESS_KEY=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

azcopy copy 'https://s3.amazonaws.com/storagetest/' 'https://tanja.blob.
core.windows.net/teststoragesv=2019-12-12&ss=b&srt=sco&st=2020-12-10
T19%3A47%3A38Z&se=2020-12-11T19%3A47%3A38Z&sp=rwdx1&sig=%2FLfT500
xKz1EXTSunrYbL%2BT%2FPpW2CyPjdKNGwKkA7Is%3D' --recursive=true

```

**GCP to AWS:** To move data from GCP to AWS, the GCP-CLI can be used. With rsync a file, folder or bucket can be copied from the GCP Cloud Storage to AWS S3. The command looks as follows: “gsutil -m rsync -r gs://storagetest s3://storagetest”. The origin and destination have to be given and -m is for performing a parallel multi-threaded copy.

← Create a transfer job

1 Select source ↑

Google Cloud Storage bucket  
 Amazon S3 bucket  
 Microsoft Azure Storage container  
 List of object URLs

Enter the Amazon S3 bucket URL and access key. Key not required if bucket read access is set to Grant Everyone. [Amazon help](#)

Amazon S3 bucket

s3:// storagetest

**i** By providing your Amazon S3 credentials you acknowledge that Google Cloud Storage is your agent solely for the limited purpose of accessing your bucket for transfers.

Credentials

AWS Management Console instructions: Navigate to [Your Security Credentials](#) and open the Access keys section. Click [Create New Access Key](#). In the dialog that appears, click [Show Access Keys](#) to see your new access key ID and secret access key.

[More](#)

Access key ID

Secret access key

Show access key

[Specify file filters](#)

[Continue](#)

Figure 36: Data transfer mask, to import data from AWS.

## 6.2 Indirect – with the help of a local system

The indirect option of moving systems between two cloud platforms has the advantage that it works for more combinations than the direct transfer. And it is usually quite uncomplicated, if the export from and import to the different systems worked already. Of course different requirements and compatibilities have to be considered. For this option only the different steps of chapter 5 and chapter 4 have to be combined. So in here, it is highlighted if any special modifications need to be done to the system, which weren't necessary for the migration from a local system to a cloud platform.

### 6.2.1 Databases

Moving databases from one cloud platform to another, with the indirect version and by using mysqldump, means simply dumping the structure, data, views, triggers, stored functions and procedures to an SQL-file. This can be done via MySQL-Workbench or

with the `mysqldump` statement. After the structure and data is dumped to a file on the local infrastructure, it can be modified if necessary. Here usually the definers needed to be removed. Then it can be imported again to the other cloud platform. How the import is performed for each cloud platform can be seen in section 4.1.

To move users from one cloud platform to another, first the data has to be extracted as it was done for each platform in chapter 5.1 or simply with `“SELECT * from mysql.user;”`. Once this is done, the users can be created on the other cloud platform. Usually users can't be imported but have to be created manually. How this works for each cloud platform can be seen in chapter 4.1. There is also the option to write an SQL script and run it, which is probably less time consuming than creating every user in MySQL-Workbench (in GCP users have to be created with the according `gcloud` command). Because the indirect move is only a combination of export and import and saving the data locally in-between, there were no unexpected problems.

### 6.2.2 Virtual Machines

To move a virtual machine from one cloud platform to another indirectly takes quite some time. Big amounts of data have to be downloaded and uploaded again. This makes it quite time and bandwidth intensive. Also all import requirements of the new cloud platforms have to be considered. Moreover, the format of the exported image often needs to be changed, because not all formats can be imported and exported from all cloud platforms. An overview, of possible import and export formats, is given in table 3. Locally the formats can be changed with the help of VirtualBox or other tools. Another problem is, that the images are not standardised, so every cloud platform uses slightly different images. So they can have different device drivers and guest additions for example, which can cause problems during the migration.

Table 3: Possible import and export formats of the cloud platforms.

Cloud Platform	Import format	Export format
GCP	OVA, VHD, VMDK	VMDK, VHDX, QCOW2
AWS	OVA, VHD, VMDK	OVA, VHD, VMDK
Azure	VHD	VHD

How virtual machines can be exported from each cloud platform can be seen in chapter 5.2 and how they can be imported can be seen in chapter 4.2. For the transfer from GCP to AWS for example, either one imports the VMDK to Azure or changes it to an OVA with the help of VirtualBox. Here the second option was used. Unfortunately when trying to migrate a virtual machine from GCP or Azure to AWS, the following error occurred: `“ClientError: Multiple different grub/menu.lst files found”` and this error could not be resolved. The virtual machines exported from GCP and Azure were running locally without problems, but there was no indication on what to change so that the virtual machines could be imported to AWS.

Also the indirect migration from GCP and AWS to Azure was not possible. As can be seen in chapter 4.2.3, special preparations have to be done to import a virtual machine to Azure. These preparations have to be done locally, because at the end the kernel is changed and with an Azure tailored kernel the virtual machines neither work in the other

clouds nor can be exported from there. But as was also mentioned in chapter 4.2.3, the preparations can't be done without a Hyper-V manager.

At least the migration from AWS and Azure to GCP worked fine. For a faster upload the exported VHD-file from Azure was changed to an OVA-file, which then was imported to GCP.

### 6.2.3 Docker

The indirect version for moving docker images means, that images get pulled from the respective cloud platform to the local system. There they get tagged accordingly, then pushed to the other cloud platform and finally they can be deployed there. How the export works in detail for every cloud platform can be seen in chapter 5.3 and for import to the cloud the detailed information can be found in chapter 4.3. The indirect move works fine for every combination of cloud platforms.

### 6.2.4 Storage

Moving data between two cloud platforms indirectly works well. First the data has to be downloaded. How this works for each cloud platform is described in chapter 5.4. From there it can be imported to the next cloud, as described in chapter 4.4. This way data can be moved indirectly. Access permissions can't be exported or imported. A new access control list has to be created for each cloud platform.

## 6.3 Comparison

It can't be said, which method is better for transferring systems between two cloud platforms, the direct or indirect one. Both options have their advantages and disadvantages. While the indirect move is possible for more combinations of cloud platforms, the direct transfer usually is faster because there are fewer data transfers needed. Moreover, the connection between cloud providers is normally faster than the up- and download at home or work. Besides, for the direct option no local setup is necessary, which saves resources. Also it could be a lot of work to set up a local system, only for moving systems from one cloud platform to another. One more point to consider are the costs, if special hardware is needed for the indirect transfer.

Nevertheless, the direct option is usually only available, if the cloud providers support it. Most of the time, there are special tools needed for the direct transfer, like the GCP Transfer Service for data, the Cloud Migrate for Compute Engine for migrating virtual machines to GCP, the SMS connector of AWS or the Azure Migrate: Server Migration tool. An exception was the transfer of the database, there it worked with redirecting the mysqldump to the other cloud platform. But some things - like the users of a database - can't be transferred directly. That's because of insufficient permissions in the target cloud. However, when transfer services are used, one can never be sure which changes are done to the system. Often they offer a transfer where the system doesn't need to be stopped. So this is an advantage when the system needs to be available continuously.

Also for docker, the direct and indirect transfer option have advantages and disadvantages. While not everything can be transferred directly, it is quite convenient to directly push an image to the container registry of another cloud platform and use it there. This is definitely the fastest option. The advantage of the indirect move is, that no tokens

for the authentication have to be generated and disclosed. This authentication method should also not be taken as granted, because in Azure it is only available as a preview yet. The tokens are needed because the container instances, in which the docker containers are running, only have docker installed and nothing else can be installed there. But for the authentication to GCP, AWS or Azure usually gcloud, the AWS-CLI or Azure-CLI are necessary.

When it comes to virtual machines, a migration with the help of a migration manager is definitely more comfortable. Virtual machines have strict import requirements for each cloud platform. So it can be quite tricky to prepare the virtual machine for the import to another cloud platform. Also for the indirect transfer to Azure, special Microsoft tools are necessary, like the Hyper-V Manager. With a transfer manager, big numbers of virtual machines can be transferred at the same time, which is more convenient than preparing them all manually but therefore also costs for the transfer manager and other needed tools have to be considered. Virtual machines were also the only part of the test system, where the indirect transfer between the cloud platforms was not possible for all combinations. The import to Azure was not tested because no Hyper-V Manager was available. But also the migration from GCP and Azure to AWS did not work. This was surprising, because the import of a virtual machine, which was not exported from another cloud platform before, worked just fine.

Last but not least, when one already knows how to import and export systems from cloud platforms, then the indirect option has the advantage, that this knowledge usually can be used and no time for researching or to train people needs to be planned.



## 7 Summary and Conclusion

In this section a recapitulation and final conclusion is given, as well as a short outlook of possible future and related works.

### 7.1 Results

As mentioned in chapter 1 there are various reasons why systems should be migrated to the cloud or between clouds. So, in this thesis it was examined how a migration to and from the three chosen cloud providers works. The used cloud platforms were GCP (Google), AWS (Amazon) and Azure (Microsoft), because they are by far the largest. For the experiments a test system, consisting of a database, virtual machines, a docker container and storage was set up. Not just the import and export to the cloud platforms was tested but also the transfer of systems between the cloud platforms. For this last option the direct and indirect variant was examined. The goal of each experiment was a running system, either on a cloud platform or on a local system.

One thing that all tested cloud platforms had in common was, that a command-line interface or the cloud console could be used for import and export tasks. For GCP the command-line interface is called `gcloud`, for AWS it's the `AWS-CLI` and `Azure-CLI` is used for Azure. With the command-line interface, most import and export tasks could be completed. Sometimes it was only a choice of preference to use the cloud-console instead. Those command-line interfaces are available for different operating systems and installed within a few minutes. Installing them was usually the first step for a migration.

Migrating the MySQL database was not a problem, when just the structure of the database, the data and views were migrated. In GCP and Azure a special flag had to be set to also import triggers, stored functions and procedures. Contrary to that, in AWS it was not possible to import triggers, stored functions and procedures. Those had to be created after the database was already running. For the import to GCP, an SQL-file was uploaded to the cloud storage and imported from there, in AWS `mysqldump` was used and in Azure the database was connected with `MySQL-Workbench` and the import function of `MySQL-Workbench` was used. For the export, similar strategies were used: in GCP an SQL-file was exported with the help of the cloud console, in AWS `mysqldump` was used and in Azure the export was done via `MySQL-Workbench`.

However, the import process for database users is definitely something that should be improved by all tested cloud providers. The processes for that seems not suitable for large amounts of users. An automated import is usually not possible. In GCP, users have to be created manually in the cloud console, before their permissions can be changed in the database with `REVOKE` and `GRANT`. In AWS and Azure `CREATE` statements can be used. For exporting the users in GCP and AWS, the database "mysql" had to be exported with a `mysqldump`. There a section with user details can be found. This section includes the `INSERT` statements of the users, which were copied to the local database. In Azure the users and their permissions had to be exported with the `SELECT * FROM mysql.user` statement. With this information the users could be rebuilt in the local system. When migrating a database, the biggest problem in all three cloud platforms was the lack of permissions, even for the admin user.

Virtual machines were the part of the test system, which had the strictest requirements and regulations, no matter to which cloud platform they should be imported. The import and export formats are limited as well as the operating system versions and sometimes

even the kernel version of the virtual machine has to be considered. Regarding the import formats, Azure has the most limited selection. There only VHD files can be imported and exported. In GCP and AWS more formats are possible, so the import was done with virtual appliances (OVA). These needed to be uploaded to the cloud storage first, before a virtual machine could be created from them. In GCP and AWS the export worked similar, the generated file had first to be exported to the cloud storage and could then be downloaded from there. Azure, on the other hand, simply created a download link for the VHD file.

For uploading docker container to the cloud, the process for all three cloud platforms was quite similar. Each cloud platform offered a container registry, to where the image had to be pushed and from where it could be deployed to an instance. The export was the reverse process: creating an image in the cloud and then pulling it from the respective container registry.

To manage the storage, in Azure a special tool is used, the so called Azure Storage Explorer. With this tool, objects can easily be uploaded and downloaded from the cloud. Also, for GCP and AWS, simple drag and drop is not an option. One can upload objects via drag and drop but only download one object a time with this. So for large amounts of data the command-line interface must be used.

The next part of the thesis was about the migration between two cloud platforms. The advantage of the indirect migration was definitely, that it is possible for almost all combinations. The exception for this was the transfer of a virtual machine to AWS (an unsolvable error occurred) and to Azure (for that Hyper-V would be needed to prepare the machine accordingly). But for all other parts of the test system the steps from the export and import were combined with some small modifications of the system, while it was stored on the local system.

Nevertheless, the much faster and less resource intensive option is the direct one. Unfortunately this option is not available for all combinations of cloud platforms. Only the database structure, data and views can be transferred between all cloud platforms with the help of a direct mysqldump. Also from GCP and AWS the docker images can be directly pushed to another container registry and no local system is needed in-between. For a direct storage transfer GCP even offers a special data transfer option, where AWS and Azure can be used as a source.

Finally one thing has to be said: there is not only one right way to move a system to the cloud or between different cloud platforms. There are various ways, variants and options which may suit one system but do not work with other systems. Also, each of the tested cloud platforms has their strengths and weaknesses. While in AWS the import of a database was more problematic than in GCP and Azure, Azure definitely makes it the most difficult to import virtual machines. Regarding the migration process, it felt like the processes of GCP and AWS are more similar to each other and Azure just uses a completely different strategy.

This thesis was challenging in many different ways. While all the used tools were known before, it was interesting to look at them regarding the migration process. Some migration specific errors arose and to solve them it was necessary to gain a better understanding of the used systems. Sometimes it was also hard to decide which way to go. Usually there is more than one way to migrate a system and due to the limited time one way had to be chosen and examined. Nevertheless it was not always clear which variant is the best or most suitable one in a certain situation.

## 7.2 Implications and future directions

Cloud platforms are changing systems, they adapt to the users needs. That's why one possible future research could be to recreate the experiments from this thesis and have a look on how the import and export processes have changed. Maybe they will stay the same, because companies don't tend to change their cloud providers. Maybe only the transfer between clouds will change, because most systems in the future will be already running on a cloud platform and the import from a local system will not be as important. Maybe due to GAIA-X, a European cloud specification, cloud environments will be designed more similarly and migrations between different platforms can be done with one click. There are various options how this can develop and therefore for this topic, the time when the experiments are performed definitely matters.

One more option would be to look at other test systems. Here, the test system consisted of a database, a virtual machine, a docker container and a storage system but many more systems are used for daily businesses. Different development tools, monitoring tools or whole kubernetes environments are just a few examples, which weren't explored here.

Also in this thesis only the three biggest cloud platforms (GCP, AWS and Azure) were examined. So another idea would be to compare the import and export options of more or other cloud platforms. Maybe smaller cloud providers offer more individual import and export strategies or maybe systems even have to be transferred by their specialists and an external user can't do any imports or exports. Of course it could also happen, that another cloud provider gains more customers and then in a few years GCP, AWS and Azure are not the biggest ones anymore.

Moreover, systems where parts are on a cloud platform and the other part is either on a local system or on a different cloud platform can be examined. If those two or more systems have to communicate, which implications does this have on the import and export process? Are there certain permissions needed for setting something like this up? What if just a part of the system should be exported, can the other one still run in the cloud without complications?

A different approach to this topic, would be to try to develop a tool which automates the import and export process for all cloud platforms. This seems like a big challenge, because for now all cloud platforms have different processes and requirements. Maybe a first step towards this goal would be to work on a guideline with recommendations for standardized import and export processes.

As can be seen there are many more paths to explore, when it comes to cloud interchangeability. Import and export processes are definitely an interesting topic, where a lot of things can still be automated and optimized.

## References

- [1] Migrating to AWS: Best practices and strategies. <https://s3-ap-southeast-1.amazonaws.com/mktg-apac/Cloud+Migration+to+AWS+Campaign/AWS+ebook+Migrating+to+AWS.pdf>. (visited on 13.1.2021).
- [2] Inc Amazon Web Services. Angebotstypen. <https://aws.amazon.com/de/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc>. (visited on 14.10.2020).
- [3] Inc Amazon Web Services. Aws migration competency partners. <https://aws.amazon.com/migration/partner-solutions/?partner-solutions-cards.sort-by=item.additionalFields.partnerNameLower&partner-solutions-cards.sort-order=asc>. (visited on 15.1.2021).
- [4] Inc Amazon Web Services. Cloud-Produkte. <https://aws.amazon.com/de/products/>. (visited on 14.10.2020).
- [5] Inc Amazon Web Services. Install the server migration connector on Azure. <https://docs.aws.amazon.com/server-migration-service/latest/userguide/Azure.html>. (visited on 7.1.2021).
- [6] Inc Amazon Web Services. Migrate petabyte-scale data to the cloud. [https://aws.amazon.com/getting-started/hands-on/migrate-petabyte-scale-data/services-costs/?nc1=h\\_ls](https://aws.amazon.com/getting-started/hands-on/migrate-petabyte-scale-data/services-costs/?nc1=h_ls). (visited on 15.1.2021).
- [7] Inc Amazon Web Services. Unsere Geschichte: Was aus einer Garagen-Idee werden kann? <https://www.aboutamazon.de/über-amazon/unsere-geschichte-was-aus-einer-garagen-idee-werden-kann>. (visited on 14.10.2020).
- [8] Inc Amazon Web Services. What is AWS marketplace? <https://docs.aws.amazon.com/marketplace/latest/userguide/what-is-marketplace.html>. (visited on 14.10.2020).
- [9] Mohammad Ubaidullah Bokhari, Qahtan Makki Shallal, and Yahya Kord Tamandani. Cloud computing service models: A comparative study. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 890–895. IEEE, 2016.
- [10] Sneha Borge and Nidhi Poonia. Review on amazon web services, google cloud provider and microsoft windows azure. *Advance and Innovative Research*, page 53, 2020.
- [11] Microsoft Corporation. About microsoft - explore a timeline of microsoft’s journey. <https://news.microsoft.com/about/>. (visited on 14.10.2020).
- [12] Microsoft Corporation. Azure products. <https://azure.microsoft.com/en-us/services/#windows-virtual-desktop>. (visited on 14.10.2020).
- [13] Microsoft Corporation. Kostenloses Azure-Konto. <https://azure.microsoft.com/de-de/offers/ms-azr-0044p/>. (visited on 14.10.2020).

- [14] Microsoft Corporation. What is Azure? <https://azure.microsoft.com/en-us/overview/what-is-azure/>. (visited on 14.10.2020).
- [15] Microsoft Corporation. Windows virtual desktop. <https://azure.microsoft.com/en-us/services/virtual-desktop/#features>. (visited on 14.10.2020).
- [16] Microsoft Corporation. About azure migrate. <https://docs.microsoft.com/en-us/azure/migrate/migrate-services-overview>, 2020. (visited on 15.1.2021).
- [17] Microsoft Corporation. Agent-based migration architecture. <https://docs.microsoft.com/en-us/azure/migrate/agent-based-migration-architecture>, 2020. (visited on 7.1.2021).
- [18] Microsoft Corporation. Discover, assess, and migrate Amazon Web Services (AWS) vms to azure. <https://docs.microsoft.com/en-us/azure/migrate/tutorial-migrate-aws-virtual-machines>, 2020. (visited on 7.1.2021).
- [19] Oracle Corporation. Structure. <https://dev.mysql.com/doc/sakila/en/sakila-structure.html>. (visited on 16.10.2020).
- [20] Oracle Corporation. Welcome to virtualbox.org! <https://www.virtualbox.org>. (visited on 29.3.2021).
- [21] Tinankoria Diaby and Babak Bashari Rad. Cloud computing: a review of the concepts and deployment models. *International Journal of Information Technology and Computer Science*, 9(6):50–58, 2017.
- [22] Dijiang Huang and Huijun Wu. Chapter 3 - mobile cloud service models. In Dijiang Huang and Huijun Wu, editors, *Mobile Cloud Computing*, pages 65 – 85. Morgan Kaufmann, 2018.
- [23] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Cloud migration research: a systematic review. *IEEE transactions on cloud computing*, 1(2):142–157, 2013.
- [24] Muhammad Ayoub Kamal, Hafiz Wahab Raza, Muhammad Mansoor Alam, and Mazliham Mohd Su’ud. Highlight the features of aws, gcp and microsoft azure that have an impact when choosing a cloud service provider. 2020.
- [25] Google LLC. Getting started with migrate for compute engine. <https://cloud.google.com/migrate/compute-engine/docs/4.5/getting-started>. (visited on 15.1.2021).
- [26] Google LLC. About google cloud services. <https://cloud.google.com/docs/overview/cloud-platform-services>, 2020. (visited on 14.10.2020).
- [27] Google LLC. From the garage to the googleplex. <https://about.google/intl/en/our-story/>, 2020. (visited on 14.10.2020).
- [28] Google LLC. Google cloud free program. <https://cloud.google.com/free/docs/gcp-free-tier>, 2020. (visited on 14.10.2020).
- [29] Google LLC. Google cloud overview. <https://cloud.google.com/docs/overview>, 2020. (visited on 14.10.2020).

- [30] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.
- [31] Dana Petcu and Athanasios V Vasilakos. Portability in clouds: approaches and research opportunities. *Scalable Computing: Practice and Experience*, 15(3):251–270, 2014.
- [32] Aaqib Rashid and Amit Chaturvedi. Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering*, 7(2):421–426, 2019.
- [33] Derrick Rountree and Ileana Castrillo. Chapter 1 - introduction to the cloud. In Derrick Rountree and Ileana Castrillo, editors, *The Basics of Cloud Computing*, pages 1 – 17. Syngress, Boston, 2014.
- [34] Derrick Rountree and Ileana Castrillo. Chapter 3 - cloud deployment models. In Derrick Rountree and Ileana Castrillo, editors, *The Basics of Cloud Computing*, pages 35 – 47. Syngress, Boston, 2014.
- [35] Derrick Rountree and Ileana Castrillo. Chapter 4 - cloud service models. In Derrick Rountree and Ileana Castrillo, editors, *The Basics of Cloud Computing*, pages 49 – 94. Syngress, Boston, 2014.
- [36] Chellammal Surianarayanan and Pethuru Raj Chelliah. *Cloud Migration*, pages 221–240. Springer International Publishing, Cham, 2019.
- [37] Chellammal Surianarayanan and Pethuru Raj Chelliah. *Fundamentals of Cloud Computing*, pages 33–67. Springer International Publishing, Cham, 2019.
- [38] Chellammal Surianarayanan and Pethuru Raj Chelliah. *Introduction to Cloud Computing*, pages 1–32. Springer International Publishing, Cham, 2019.
- [39] Jun-Feng Zhao and Jian-Tao Zhou. Strategies and methods for cloud migration. *international Journal of Automation and Computing*, 11(2):143–152, 2014.

## List of Figures

1	The five essential characteristics of cloud computing [21]. . . . .	5
2	“Self-service based access” [37]. . . . .	6
3	“Resource Pooling and multi-tenancy” [37]. . . . .	7
4	Responsibilities in the three service models [22]. . . . .	8
5	Scheme of a hybrid cloud [37]. . . . .	12
6	Parts of a system, which can be migrated to a cloud platform [23]. . . . .	13
7	Migration strategies [36]. . . . .	14
8	Azure migration architecture [17]. . . . .	17
9	Market shares of GCP, AWS and Azure in 2020 [10]. . . . .	18
10	Logo of GCP [29]. . . . .	19
11	Logo of AWS [2]. . . . .	20
12	Logo of Azure [13]. . . . .	21
13	Scheme of Sakila (sample database) [19]. . . . .	25
14	Mask for exporting a database with MySQL-Workbench. . . . .	28
15	Database import screen of GCP. . . . .	29
16	Connection information of the created AWS RDS instance. . . . .	31
17	Mask for creating the database instance. . . . .	32
18	Server details of “azureserversakila”. . . . .	33
19	Output of the precheck-tool. . . . .	34
20	Contents of trust-policy.json. . . . .	35
21	Contents of role-policy.json. . . . .	35
22	Contents of containers.json. . . . .	36
23	Tag a docker image and push it to the GCP repository. . . . .	38
24	Connecting to the container. . . . .	38
25	Suggested commands from AWS, to push an image to ECR. . . . .	39
26	Tag the image and push it to ACR. . . . .	39
27	Mask for exporting a database from GCP. . . . .	43
28	Export information screen. . . . .	45
29	Json-file for exporting an instance. . . . .	46
30	Export status information. . . . .	47
31	Download link for the VHD-file. . . . .	47
32	Suggested pull command from GCP. . . . .	48
33	Authenticating to GCP container registry, tagging and pushing the image. . . . .	53
34	Authenticating to AWS ECR, tagging and pushing the image. . . . .	53
35	Authenticating to Azure ACR, tagging and pushing the image. . . . .	54
36	Data transfer mask, to import data from AWS. . . . .	55

# List of Tables

- 1 Possible direct transfers. For the database a direct user transfer is not possible, so here only the transfer of the structure, data, views, triggers, stored functions and procedures is examined. . . . . 50
- 2 Connection information needed for the mysqldump. . . . . 51
- 3 Possible import and export formats of the cloud platforms. . . . . 56



## SWORN DECLARATION

I hereby declare under oath that the submitted Master's Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.

The submitted document here present is identical to the electronically submitted text document.

Linz, 29.3.2021  
Place, Date

Tanja Fraundorfer  
Signature