Continuous Validation of Multi-Cloud Systems by Automated Test Scripting

JOSEFINE SANDSTRÖM
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Continuous Validation of Multi-Cloud Systems by Automated Test Scripting

Josefine Sandström
elt14jsa@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisors:
Christin Lindholm, Lund University
Richard Niklasson, Elastic Mobile
Richard Pålsson, Elastic Mobile

Examiner:
Christian Nyberg, Lund University

June 17, 2019

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Scalability, pay-as-you-go, and flexibility are some factors why more and more organizations are utilizing a cloud environment for their IT systems. Specifically, the trend is towards multi-cloud, where different cloud providers manage different applications and services. This environment is however a lot less deterministic than an on premise solution. For example, if an application from one cloud provider is updated, this might result in that another application from another cloud provider stops working. This Master's thesis investigates how a system owner can validate his/her cloud-based business system continuously, to be able too see that it functions as expected, as well as to spot anomalies. A measurement method is developed using Apache JMeter and crontab, with a setup using an Amazon EC2 instance and Raspberry Pis. To test this method, a multiple case study is performed on two different companies with two different cloud environments. Three tests are performed. One measures the login process, among other things, another measures the time required to generate a report of inventory values, and the third measures the response times of an ordering system. The method is successful for both environments, and all tests, as is shown by this thesis. For the first and third test, the response times are visualized in graphs. The result of the second test is shown in a table. The two companies are different in the sense that one is a large international company, while the other is a medium-sized company just located in Sweden. Also, the third test is performed because this company experienced its ordering system as slow, while the other company had no reported anomalies. Hence, this measurement method can be used to test different functions in a business system for both large and medium-sized companies. Furthermore, if a company suspects an error in their system, this method can be used to verify its existence, as well as verify if the error has been solved or not.

Popular Science Summary

It has become very popular for organizations to move their IT systems to the cloud. However, this environment can be quite unpredictable and changes constantly. How does this affect an organization's cloud based IT system? Well, the answer is to measure, to know.

Have you ever used Google's email service? Perhaps you have a few documents saved in Google Drive or use Dropbox as an additional backup for your photos? You access them through the Internet without having to maintain or support them yourself. This is because they are all applications based in the cloud.

Cloud based applications and services are maintained by cloud providers, such as Amazon, Microsoft, and Google. If an organization would use multiple cloud providers for their IT system, for example the storage service from Amazon and the analyzing tools from Microsoft, they are utilizing a multi-cloud environment. The providers are independent of each other. Thus, an IT system in this environment can stop working as expected if, for example, one provider adds a new version to its service, which is incompatible with the other providers' services. To be able to see how new changes affect a system, validation can be performed. Also, since the cloud changes constantly, this can with advantage be performed continuously.

This Master's thesis investigates how continuous validation can be executed, and a measurement method is developed. By performing measurements on a system continuously, this method can clarify what normal behaviour is and what is not. It is used for gaining knowledge about a system, for example measuring the time it takes to log in. It can also be used when trying to solve an error. For example, if users complain about a system being slow, this method can tell if the system has regained its speed again by measuring both before and after the potential solution took place.

A multiple case study is performed to evaluate the method, and for both test cases included, the method is considered successful. The method includes a lot of manual work, which might be a drawback for some organisations. However, this is considered to have less impact, comparing with the overall advantages with continuous validation.

Acknowledgements

First and foremost, I would like to thank my supervisors, Christin Lindholm, Richard Niklasson and Richard Pålsson, as well as my examiner Christian Nyberg, for their guidance and support throughout this thesis. A special thanks also goes to all colleagues at Elastic Mobile for providing a great work place, as well as to Kiviks Musteri AB and Company A for being part of this thesis' multiple case study. Finally, I would like to express my gratitude to my family, friends, and boyfriend for their great support during my time at LTH.

Table of Contents

Chapter 1
Introduction

More and more organizations are moving their IT systems to the cloud [42]. Cloud computing enables on-demand access to computing power, database storage, applications and other IT resources, often over the Internet. It can also be cheaper, removing the high costs of hardware, than for a company to own their servers themselves. The company can choose to adopt a single cloud environment, by using a single cloud provider to serve all their applications and services, or a multi-cloud environment, and the trend is towards multi-cloud [13]. In contrast to single clouds, different applications and services are provided by different cloud providers in the multi-cloud environment. It uses a mix of cloud offerings and this is often appealing for companies because of the many choices they are given. They can single out the service that is both cheapest and most suitable for their needs. However, multi-clouds are very complex. When companies maintain and house their own data centers, also called on premises data centers or "on prem", the company have full control over the infrastructure. This is not the case for cloud users, and the behavior in the cloud can be a lot less deterministic.

1.1 Background

A cloud provider is hosting applications and services for multiple users at the same time. The cloud is a multi-tenant environment and the users are sharing resources like bandwidth and CPU. If an application or virtual machine uses the majority of available resources, the others can experience performance issues. This effect is called noisy neighbor and can for example cause slow speeds and high *latency* [14]. For a user this means a worse user experience and possibly a decrease in productivity. The effect of noisy neighbor can change sporadically. Sometimes it might not be a problem, sometimes it might.

The multi-cloud contains of many different types of systems from different cloud providers. These systems are independent of each other and are therefore not updated at the same time. New features and new versions can for example be added to one system, which are not compatible with the others. All companies have a system owner who is responsible for their IT systems, or multi-cloud system in this case. For the system owner, also referred to as owner, all these scenarios usually result in complaints about how "nothing works". Hence, it is essential for the owner to know if the system will work satisfactory or not. The owner must

find out if something, perhaps a new feature or update, is causing problems. To gain this knowledge the system owner must validate the system to see if it meets the expectations and requirements that was once specified by the company.

1.2 Project Aim

The aim with this Master's thesis is to investigate how a system owner can perform continuous validation of his/her business system in a multi-cloud environment by using automated test scripting. By performing the validation continuously, the owner will get an overview of the system, trends can be spotted, and errors can be detected fast. During the validation, the system can be seen as a black box. A stimuli is submitted, which is dependent of what part of the system that is to be validated, and the response will be evaluated. A measurement method is developed, including three measurement setups. To test this method, a multiple case study is performed which includes two companies with two different cloud environments.

1.3 Limitations

This thesis aims to investigate *if* anomalies occur in a cloud-based business system, but not *where* those anomalies occur or how to solve them.

1.4 Research Questions

Questions which this thesis project aims to investigate and answer are as follows:

- Is it possible to perform continuous validation of a multi-cloud system by automated test scripting and if so, what advantages are there?
- How can continuous validation by automated test scripting be performed?
- What software tools can be used to perform continuous validation?
- What should be measured to know if the business system is working as specified for the system owner?

1.5 Disposition

The disposition of this thesis report is as follows. Chapter 2 discusses advantages of continuous validation and how others have tested their cloud systems. Information about cloud related technologies, technical components, and software programs is found in chapter 3. The multiple case study performed during this thesis can be read about in chapter 4. This chapter also discusses the different test cases, and how measurements were executed. Each test case is followed by a section where the results are addressed and analyzed. Chapter 5 then evaluates the measurement method, the used setups, and the requirements of the companies' systems. Conclusions and future work, as well as answers to the research

questions, can be found in chapter 6. Finally, chapter 7 includes explanations of words and expressions found in this report in italics.

Chapter 2

Related Research

This chapter aims to partially answer the research questions in section 1.4. Because there were not enough relevant articles found on continuous validation, this chapter will also look into non-continuous testing and validation. It will discuss what metrics one can measure to validate or test a cloud-based system or application, and what software tools that can be used for this.

2.1 Advantages of continuous validation

Section 1.1 mentions some reasons why continuous validation is of interest. N. Talens also writes in [1] that "software development has everything to do with validations". He asks for example if a newly added product feature can undo the effect of another, and if that wouldn't be valuable to know. Continuous validation is needed to make sure that software works as intended and that it keeps doing so [1]. Hence, it is desirable to know how new features or changes affect the system and, as Talens writes, "be able to act on them".

The company xLM, a subsidiary of ValiMation Inc, has introduced a continuous validation framework for cloud apps [2] and they define continuous validation as follows:

> Continuous validation is providing documented evidence to certify that an app not only met the pre-established acceptance criteria, but "continuous" to meet thus mitigating the risk of unknown changes.

They explain that continuous validation connects validation in different phases, for example initial validation or validation after upgrade, with continuous *smoke and regression testing*. Hence, this can confirm that an app continues to function as expected in the present, if it worked well in the past. Furthermore, they write that the risk of changes that can alter the behaviour of the app are mitigated. New changes are released in the cloud environment constantly, but the continuous validation framework can, for example daily, ensure that the requirements are met despite the changes.

2.2 Relevant metrics

The xLM continuous validation mentioned above consists of seven steps. The first is ”Requirements Definition”, during which the intended use of the cloud app is defined. Aspects as performance, security and disaster recovery are specified. ”Risk Assessments” is the second step, performed to ensure business continuity and regulatory compliance, among other things. This step is also performed to establish the testing models, for example to determine what features to test or what testing strategies to use. Next step is ”Specification Definition” where the app specifications are defined based on the intended use. The specifications include configuration, workflow, log management, etc. Next, various models are developed to validate the app. This step is called ”Test Automation Scripts”. Here, xLM uses their Model Based Test Automation framework, and a data designer is used to generate test data. Then ”Test Model Validation” is performed to ensure that the test automation model itself meets the specified requirements. The sixth step is ”Test Execution”. The model based test automation approach makes it possible to implement various types of tests, for example smoke, regression, load, or performance tests. The last step is ”Validation Reporting” where for example summary reports, and test deviation reports are provided by the xLM platform.

2.2.1 Performance evaluation

Performance evaluation of cloud computing is discussed in [12]. N. Khanghahi and R. Ravanmehr states that ”cloud computing resources must be compatible, high performance, and powerful”. Users and service providers are influenced by higher performance of services and anything cloud related. For them, therefore, performance evaluation is very important. Performance can be affected by several factors. One example is recovery, which is defined by the time required for errors, failures or lost data to be retrieved again. Other examples are network bandwidth, availability, number of users, scalability, and latency. Two different methods for evaluation are used in the article, evaluation based on criteria and characteristics, and evaluation based on simulation.

There are several criteria, or metrics, for evaluating the factors that affects the performance of cloud computing. There is average *response time* per unit time, which covers all factors mentioned above. There is network capacity per unit time, which mainly covers network bandwidth, availability and scalability. There is also throughput, average processing time, the number of requests executed per unit time, the number of rejected requests per unit time and so on.

For simulation the authors used the tool CloudAnalyst, which is further discussed in section 2.3. There are two main components in CloudAnalyst that are configurable. The first is data centers which shows the hardware configuration, such as processors, and bandwidth. They can also be defined in different geographic areas. The second component is users, which can symbolize a person, an organization or a group. For users, one can configure for example geographical area and number of requests per hour. Some of the metrics used in the simulations are overall response time, processing time in the overall data center and response time per user. Different scenarios are simulated where data centers, users

and geographical region are modified in different ways. Data centers are modified by changing the virtual machine, memory, and bandwidth. Users are modified by changing the number of users and volume of work. The geographical region is modified to see how the result changes when the data centers and users are in the same region compared to in different regions far away from each other.

2.2.2 Scalability testing

In [3], W.-T. Tsai, Y. Huang and Q. Shao perform scalability testing and they write that "the key in scalability testing is the metric used to measure the scalability". Two examples of metrics are speedup and efficiency. Speedup in this case refers to how an increasing number of processors also increases the rate of doing work, compared to one processor, and efficiency refers to the work rate per processor. According to the authors however, these metrics are not sufficient for evaluating cloud and Software as a Service (SaaS), see section 3.1.1 for definition. This is partly because these metrics doesn't consider the complexity in the cloud environment, and partly because the performance may differ depending on the workload on the cloud application.

The authors present other metrics for cloud applications or systems, for example performance/resource ratio (PRR). This metric describes the relationship between the performance of the system under test and the resources used, i.e. both the required computing time and the resources consumed in the process. Hence, the metric PRR is dependent of the waiting time, T_w, and the resource consumption, C_R. The waiting time for a resource is the sum of the queuing time, T_q, and the time it takes to execute it, T_e. The resource consumption is determined by the allocation of resource i, which is denoted R_i and can for example be CPU and memory usage, and the time resource i is used, T_i. The formulas are as follows:

$$T_w = T_q + T_e \tag{2.1}$$

$$C_R = \sum R_i * T_i \tag{2.2}$$

$$PRR = \frac{1}{T_w} * \frac{1}{C_R} \tag{2.3}$$

When the workload changes, the performance also changes and the scalability is measured by this change. How this performance change (PC) is calculated can be seen in equation (2.4) below, where t and t' represent different time periods. The workload at these time periods is denoted W.

$$PC = \frac{PRR(t)W(t)}{PRR(t')W(t')} \tag{2.4}$$

The ideal PC is equal to 1. Because of the complexity of the cloud system, the PC may vary between different test runs. Therefore it could be beneficial to also consider the performance variance (PV) when testing scalability. It can be computed by the standard variance of the PC in multiple runs of the same

workload, as shown in equation (2.5). A good scalability is shown by a PV close to 0.

$$PV = E[(PC_i - \frac{1}{n}\sum_{i=1}^{n} PC_i)^2] \tag{2.5}$$

2.2.3 Testing as a Service

One approach that has received wide attention when it comes to testing software based in cloud, is Testing as a Service (TaaS). It is a cloud-based service that provides a scalable testing environment, a reduction in cost, and on-demand testing services [4]. Gao et al. [5] proposes a TaaS infrastructure, also referred to as cloud-based TaaS or CTaaS, which supports automated testing of cloud-based software and SaaS applications. To support virtual test environments and the TaaS servers, the authors used Amazon's EC2 as the cloud infrastructure, which can be read about in section 3.3. The TaaS infrastructure consists of three layers. The UI-layer provides the TaaS user interfaces and the SaaS user interfaces. The Test Space layer refers to the cloud-based virtual test environment. It contains the SaaS under test and its supporting test frameworks, for example the simulation agent that simulates system load and the interactions between SaaS, among other things. The TaaS layer consists of a number of TaaS servers, for example on-demand test server or test simulation server. The servers can be individually instantiated in a cloud to support specific testing services by communicating with their corresponding agents.

SaaS performance validation is supported by CTaaS. The TaaS infrastructure has a performance validation component that supports several performance evaluation metrics. It does this by communicating with Amazon's EC2 CloudWatch *APIs*, a service for monitoring AWS instances [6]. The performance evaluation metrics that are mentioned by Gao et al. are Computing Resource Allocation Meter (CRAM), Computing Resource Utilization Meter (CRUM), System Performance Meter (SPM), and System Load Meter (SLM). For each SaaS and their instances, CRAM is the metric for monitoring allocated computing resources, like CPU or data storage. However, for monitoring the resource usage, the metric CRUM is used. SPM is used to evaluate and monitor the system performance in terms of system utilization, such as throughput ratio or *network utilization*. Lastly, SLM is used for evaluating the system load based on the user access load, the network load, and the data access load.

2.3 Cloud testing tools

There are several testing tools available for testing applications or systems in the cloud. One is Gatling [7], a tool designed for continuous *load testing* where the simulation scripts are written in Scala [8]. Gatling supports the HTTP protocol and can load test any HTTP server. An example of a test scenario can be made by recording a user's actions on a web application, and then launch this with an execution file provided by Gatling. Another testing tool is LoadStorm [11]. This

is also a load testing tool, designed to test web and mobile applications. Similar to Gatling it comes with the ability to record the actions on a web application. LoadStorm is a cloud-based platform which allows testing to be made from any computer with internet access. It also comes with the possibility to control the geographic distribution of traffic so that, for example, only traffic from the US will hit the application under test.

The tool CloudAnalyst [9] that is used in section 2.2.1 is built on top of a simulation toolkit called CloudSim [10]. This toolkit supports modeling and simulation of infrastructures including data centers, users, user workloads, and pricing models. CloudAnalyst utilizes the features of and extends some of the capabilities of CloudSim. It enables visual modeling and simulation of applications that are large-scale and distributed on cloud infrastructures. It provides the ability to describe application workloads, such as information about geographic location of users that generate traffic and data center locations, the amount of users and data centers, and the amount of resources in each data center. With this information, CloudAnalyst can generate information about the response time or processing time of a request, and other metrics.

Chapter 3
Technical Background

3.1 Cloud computing

Cloud computing has gained wide influence on IT systems in recent years [18], and organizations of all sizes adapts this technology to its business, whether they are small, mid-sized, or big [22]. The National Institute of Standards and Technology (NIST) define the concept as follows [19]:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

With cloud computing, an organization does not have to plan and obtain servers and other IT infrastructure weeks or months in advance. Instead, hundreds or thousands of servers can be set up in minutes in the cloud [20]. The infrastructure can be scaled up or scaled down considering the needs of the organization and typically the organization only pay for what they use. Cloud computing also allows for consumers to provision computing capabilities, for example network storage, as needed without human interaction with each service provider. There are three service models in the cloud [19], Software as a Service (SaaS), Platform as a service (PaaS), and Infrastructure as a Service (IaaS). There are also four deployment models. These are public cloud, private cloud, hybrid cloud, and community cloud.

3.1.1 Software as a Service

SaaS, or cloud application services, deliver applications managed by third party vendors to its users over the Internet. The majority of SaaS applications requires no downloads or installations on the client side. Instead they are run directly through the web browser, which simplifies for businesses by making the applications easily accessed from any computer. It is the vendors that manage all potential technical issues, like data, servers, and storage, as well as upgrading software. Examples of SaaS are Google Apps, and Dropbox [21].

3.1.2 Platform as a Service

PaaS, or cloud platform services, are used mainly for applications but also provides cloud components to certain software. With PaaS, developers obtain a framework which they can build upon and use to create customized applications. The developers can maintain management of the applications while servers, storage and networking can be managed either by the company itself or by a third-party provider. In contrast to SaaS, which delivers software over the Internet, PaaS provides a platform for software creation. This platform is delivered over the web and provides scalability, high availability, and simple and cost-effective development and deployment of applications. Examples of PaaS are AWS Elastic Beanstalk, and Google App Engine [21].

3.1.3 Infrastructure as a Service

IaaS, or cloud infrastructure services, deliver infrastructure components such as servers, network equipment, operating systems, and storage through virtualization technology. It provides the same technology and capabilities as a traditional data center, but the users do not have to maintain or manage all of it physically. The users have complete control over the entire infrastructure and can typically access and monitor it through a dashboard or an API. IaaS enables on-demand and as-needed purchase of resources, instead of having to buy hardware outright. Examples of IaaS are Amazon Web Services (AWS), Microsoft Azure, and Google Compute Engine [21].

3.1.4 Public cloud

Resources are offered as a service in the public cloud, often over the Internet and users pay only for what they use. The resources can scale depending on the user's needs, and services can be used without the purchase of hardware. Third party organizations owns public clouds and offers different cloud services. The public clouds are available to the general public and designed to be used by any user with an internet connection. All data a user creates and submits are typically stored on the servers of the third party vendor [22].

3.1.5 Private cloud

The purpose of private clouds is to use cloud services within an organization, instead of offering it to the general public. It is operated solely for the organization and can only be accessed by members of the organization and/or third parties with granted access. It can be managed either by the organization itself or by a third party, and it can be placed either on premise or off premise. Private clouds are more secure than public clouds, since it generally gives the organization more control over security parameters. It is, however, more expensive [22].

3.1.6 Hybrid cloud

A hybrid cloud is composed of at least one private cloud and one public cloud. Each cloud is a unique entity, but they are connected to each other by standardized or proprietary technology that allows transmission of data and application. Hybrid clouds offer benefits of both public and private clouds. For example they offer cost and scale benefits as public clouds does, but also security and control benefits like private clouds [22].

3.1.7 Community cloud

A community cloud lies somewhat between public and private clouds. It has similarities with private clouds but offers infrastructure and computational resources to two or more organizations with common privacy, security, and regulatory considerations, instead of a single organization. The bandwidth and data storage in the cloud is shared among all community members [22].

3.2 Single cloud vs Multi-cloud

According to [13], using a single cloud environment means that all applications and services an organization migrates to the cloud are served by a single cloud provider. Examples of cloud providers are Amazon, Microsoft and Google, and their cloud services are called Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform, respectively. When organizations employ a single cloud model, they often use the cloud for a single application or service, for example email, enterprise resource planning (*ERP*) or customer relationship management (CRM). The single cloud environment is suitable for smaller or less technically adept organizations that desire the benefits of the cloud without it becoming too overwhelming. It is also suitable for organizations who, for the time being, need less cloud resources but might need more in the future. When that day comes, and a single cloud server is not enough, the number of virtualized servers in the single cloud environment can be expanded.

In contrast to single cloud, a multi-cloud environment uses different cloud providers to serve different applications and services. By employing a multi-cloud model, an organization can choose the service that is both cheapest and most suitable considering the needs of the organization. A multi-cloud offers flexibility. It could be that one provider offers the best solution for storage, another offers the best solution for email, and a third offers the best solution for testing environments. Using a single-cloud environment, an organization is limited to use only that provider's services and applications. Similarly, multi-clouds also avoids *vendor lock-ins*. However, integration between the different cloud providers can be a problem, because of the complexity of the multi-cloud. For example, one must consider that different cloud providers have different *service level agreements*, and architectures, among other things [13]. Multi-clouds and hybrid clouds are often used interchangeably, but there are differences between them. A hybrid cloud has at least one private cloud and one public cloud and these are connected to each other. A multi-cloud typically consists of multiple public clouds but can also in-

clude private clouds. These may or may not be integrated with one another. A hybrid cloud is a single entity, but a multi-cloud is not [23].

For both test cases in this thesis, the definition of multi-cloud is extended to also include multiple clouds with different cloud account owners, within a cloud from one single cloud service provider. For example in section 4.3 there is one AWS cloud containing multiple clouds owned by two different AWS account owners.

3.3 Amazon Web Services

The cloud environments studied in this thesis is hosted by Amazon's cloud service AWS. AWS offers infrastructure services like computing power, databases, and options for storage, as well as tools to manage these resources, for example user access control, and monitoring. These services are accessed on-demand and users only pay for what they use. Resources can be located in different geographically isolated data centers which, among other things, increase availability [20].

Some AWS resources used in this thesis are VPC, and EC2. VPC stands for Virtual Private Cloud and is a customizable virtual network, logically isolated from other virtual networks in the AWS Cloud. It has close resemblance to a traditional network a user can operate in his/her own data center, but the VPC is for example scalable [27]. EC2 stands for Elastic Compute Cloud and offers a virtual machine in the AWS Cloud with scalable computing capacity. This can be used to launch virtual servers, configure security, or manage storage [28].

3.4 Apache JMeter

The load testing tool Apache JMeter, version 5.0, is used in this thesis for creating and executing test plans, as well as generating graphs of the results. It is an open source, Java based application designed to load test functional behavior and measure performance [16]. It is chosen because it is open source, it is a widely used tool and there exists much information and tutorials about it. With Jmeter one can for example create a test plan by recording a web application, simulate a heavy load, execute the test in command-line mode (CLI mode, non-GUI mode) and then generate an HTML report to visualize the results. With the HTML report, results are visualized as different graphs, for example latencies over time or bytes throughput over time.

JMeter allows for testing to be performed remotely, which makes it possible to test a business system from the views of a client. This also provides the ability to simulate a larger load since the test can be replicated across many low-end computers. One JMeter instance is the master that controls the tests, distributes them to all remote servers, also called slaves, and collects all data from them. The slaves are the ones that execute the tests and send requests to the business system, also referred to as target system. They are placed at the client whose business system is to be tested. An overview of the setup is shown in figure 3.1.

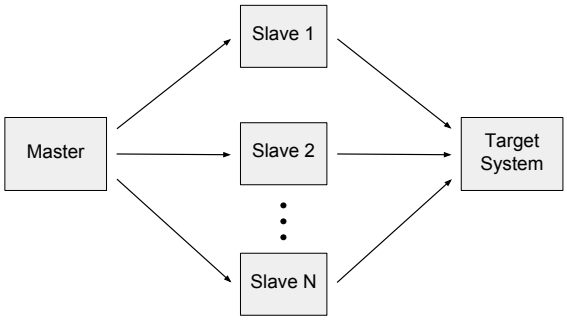

Figure 3.1: An overview of the setup used for remote testing in Apache JMeter.

JMeter uses Java Remote Method Invocation (RMI) to communicate between the master and the slaves. Java RMI allows Java virtual machines to invoke methods from other remote Java objects [17], and it requires that the JMeter master and all JMeter slaves are in the same subnet. This is not feasible if, for example, a client wants to execute a test from different offices. To solve this problem, the master and the slaves are instead communicating through SSH tunnels by using *SSH port forwarding*.

3.5 Raspberry Pi

With the aim to educate people in computing and to increase accessibility to computing education, the UK charity, Raspberry Pi Foundation, made a series of single-board computers called Raspberry Pi. These credit-card sized computers are open source and all models cost $35 or less. They are used for learning programming, build hardware projects, do home automation, and are also used in industrial applications [29] [30]. In this thesis, Raspberry Pis are used in the measurement setups because they are cheap, small, and easy to use.

3.6 Crontab

When it is desirable to schedule a task, for example a test run, crontab is used in this thesis. Crontab is a file containing instructions which are run by the "cron daemon" when specified. It is a Linux utility, that allows the "cron daemon" to run commands in the background at regular intervals, automatically [31]. In the crontab file, a user specifies a command, or a script containing several commands, to be executed at a certain time. This time is specified by the minute, hour, day of the month, month, and day of the week the command should be run. An asterisk is used to denote that the command should be run for example every hour or every day of the month. Crontab can also run commands every other minute, or every tenth minute, as well as specifying time intervals. An example is shown below:

```
*/30 8-10 * * THU /path/to/the/command
```

In this example, the command is run every month, every Thursday, and every 30 minutes between 8:00 and 10:00, that is 8:00, 8:30, 9:00, 9:30, and 10:00.

3.7 Reverse Proxy

A reverse proxy is a server, placed in front of one or more web servers, which forwards client requests to those web servers. Client requests can for example be requests from a web browser. The reverse proxy prevents direct communication from client machines to the web servers. There are many advantages using a reverse proxy. It can for example be used for load balancing and distribute requests to a web site evenly among several web servers so that no web server becomes overloaded. It can also protect against targeted attacks against the web servers, such as *Distributed Denial of Service (DDoS) attacks*. By using a reverse proxy, a web site does not have to reveal the IP address of their web servers. Attackers can therefore only target the reverse proxy, which has more resources to fend off a cyber attack [32]. In this thesis, this component is included in the system environment of Company A in section 4.3.

The reverse proxy is different from a forward proxy, where the proxy server instead is placed in front of a group of client machines. The forward proxy prevents direct communication from web servers to the client machines [32].

3.8 Apache Tomcat

The Apache Tomcat software, also referred to as Tomcat server, is used to develop web servers in the Java environment. It is an open source implementation of several Java EE [33] specifications, such as Java Servlet [34] and JavaServer Pages (JSP) [35]. Tomcat servers are widely used in different industries and organizations and powers numerous large-scale web applications [36]. In this thesis, a tomcat server is included in the system environment of Company A in section 4.3.

3.9 Internet Protocol Security (IPsec)

IPsec is a set of protocols which provides a secure exchange of data packets on the IP layer. The protocols define, among other things, cryptographic algorithms for encryption, decryption, and authentication of packets, as well as secure key exchange and key management [40]. In this thesis, an IPsec connection is included in the system environment of Company A in section 4.3.

3.10 Microsoft SQL server

The Microsoft SQL server is a database management system which uses the relational model for data management. Users can fetch and store data using T-SQL, a dialect of SQL. The requests can be made from the same computer, or from different devices across a network [39]. In this thesis, some SQL servers are included in the system environment of Company A in section 4.3.

3.11 Peering

Peering is a method of exchanging traffic directly between internet service providers (ISPs) instead of routing it through the Internet. Since the ISPs have a direct connection to each other, peering allows for fast traffic at low cost. There is no need to pay network service providers for access to the Internet backbone [37]. In this thesis, peering is used between two VPCs. This allows for instances in either VPC to communicate with each other as if they were within the same network [38].

Chapter 4
Multiple case study

This Master's thesis aims to investigate how continuous validation can be performed, and thus why and in which cases it can be used. Specifically, one method using JMeter and the setups in section 4.1 is developed, and how well this measurement method works in two test cases is investigated. According to Yin in [41], there are three situations where case study research is the preferred research method. The first is if the main research questions are "how" and "why" questions, like the ones above. The second is if the researcher has little or no control over behavioral events, and thirdly, if the focus on the study is a contemporary set of events, as opposed to historical. A case study is performed when a researcher wants to understand a real-world case, in depth [41]. In this thesis, a multiple case study is performed, covering two test cases with two different cloud environments. One is the cloud environment of Kiviks Musteri AB, and the other of an, in this thesis, anonymous company further referred to as Company A. This method is chosen because the three situations above occurs in each test case. The test case of Kivik is further branched out to two different test cases, however in the same cloud environment.

This thesis is done in collaboration with Elastic Mobile Scandinavia AB, a company which helps businesses to migrate parts of their IT systems to the cloud. They maintain these cloud environments and make sure that the clients can use them, and enter applications in them, without problems. Kivik and Company A are two clients of Elastic Mobile, and they are included in this thesis of different reasons. Kivik was eager to be a part of it so they could get a better system awareness. They also already had a Raspberry Pi for use in one of their offices, which seemed convenient. Company A got involved when an error occurred in their ordering system, see section 4.3. The employees of Elastic Mobile were ordered to examine this and the idea rose to also test the system continuously. Thus, the test case of Company A is based of an anomaly, whereas Kivik had not reported any such things.

4.1 Measurement method

Three different, but similar, measurement setups are used in this thesis. The first test case for Kivik uses the remote testing setup discussed in section 3.4. An Amazon EC2 instance is used as the JMeter master because it allows for testing

to be done continuously with an ”infinite” amount of resources. One Raspberry Pi is used as a JMeter slave because of its low price and user-friendliness. This Raspberry Pi acts as a user of Kivik's ERP system and is therefore placed at Kivik. The EC2 instance and the Raspberry Pi are communicating through SSH tunnels. This is configured by specifying which ports the master and slave should listen/send data to. The configuration is done in the respective property file. For the second test case for Kivik, tests were run from a Raspberry Pi placed at Elastic Mobile. This Raspberry is owned by Kivik and the original idea was to copy the correct test to it and then send it back. However, it was later decided that during the course of the thesis, the Raspberry should stay where it was. This meant that no new configurations needed to be done, and tests could be executed without interruption. The third setup, for Company A, also used the EC2 instance but the tests were executed directly from it and not remotely. The reason why no Raspberry Pi was used as a user in this case was that Company A had a problem which they needed knowledge about fast. There was no time to configure a remote setup and send the Raspberry Pi to them.

The tests are produced by performing, and simultaneously recording, the actions a user would do. JMeter uses a proxy to catch the HTTP requests made by the tester. A specific port is chosen and a JMeter certificate is added to the web browser used for accessing the web application. On the web browser, manual proxy configuration is chosen and the chosen port is entered. After the test has been recorded, small changes were made by manual scripting in Groovy, a dynamic programming language for the Java platform [26]. These changes could, for example, be an ID that is unique for a certain login session, and which is given to the user during the login process. Some results are shown in graphs as response times over time, and they are generated by JMeter as an HTML report. Graphs showing statistical values were made in Microsoft Excel. The tests were validated by comparing the HTTP requests recorded by JMeter with the requests shown using the web browser's developers tool. The tests were produced locally on a computer at Elastic Mobile, and later copied to the device that would execute it.

4.2 Test case 1: Kiviks Musteri AB

Kiviks Musteri AB started out as a small apple orchard in 1888, but is today a beverage company of 190 employees which produces both food and drinks [25]. This chapter describes Kivik's cloud and business system environment, as well as the two tests performed to increase system awareness about these.

4.2.1 The cloud environment

The business system, or ERP, used in this thesis is M3 by Infor. M3 is an application on the ”Infor cloud” and Infor is using AWS to support it. It offers functionality for a broad range of industries, for example food and beverage, chemicals, or fashion [15]. It offers a complete ERP suite and the functionality includes quality management, inventory management, risk management, supply chain planning, and so on. Kivik uses M3 for managing orders, inventory, invoicing, purchasing, manufacturing, logistics, and finance. This system is the main reason of why Kivik

started to use cloud. This is their most important system and they moved it to the cloud so that others, with specialized competence, could maintain it. Figure 4.1, see appendix A for a larger figure, shows the IT environment of Kivik. The different offices and on-prem components are shown at the bottom of the figure. The Raspberry Pi which executes tests and simulates a user is located at the office "Kivik". Above the offices is the AWS cloud, which includes the Infor cloud, and the cloud of Elastic Mobile. The latter includes, among other things, applications that exchange information with M3, but are not included in the Infor SaaS offer. One example is Opto (KMC-OPTO01) which is an invoice scanning system. Another example is TaGS (KMC-GS01), a production optimization tool. An employee from Kivik enters their M3 application via the Elastic Mobile Cloud, just like an employee from Elastic Mobile would do. Not included in the figure is an Azure cloud, containing SaaS applications for ordinary work tasks, such as Skype for business, and Office 365.

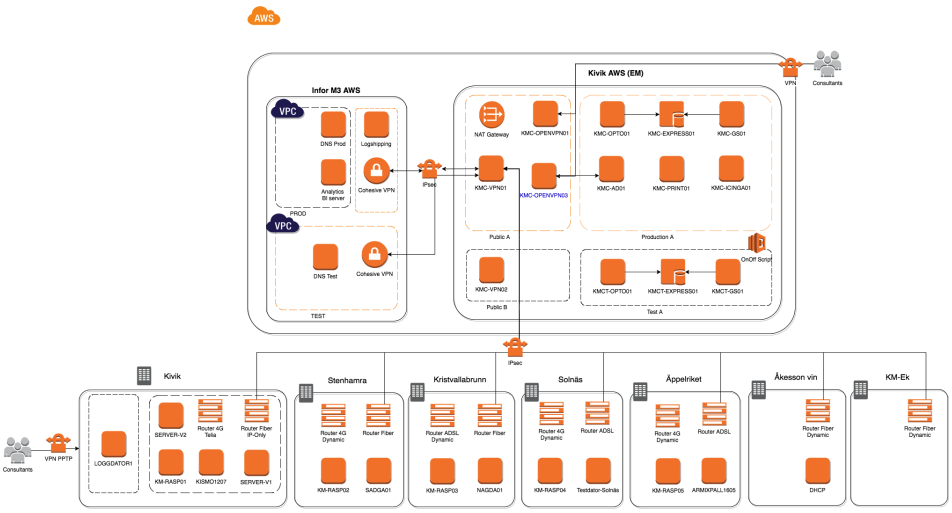

Figure 4.1: The cloud environment of Kiviks Musteri AB.

4.2.2 Login test

This test is done mostly to familiarize with the environments, but it also has another purpose. To access M3 and start utilizing its functions, a user has to login. If this takes too long, the user will complain about the system being slow and the user satisfaction decreases. Kivik does not have any documented requirements on the login process, but the expectations from the organization is a fast response. Besides the login process, this test also lists future transactions of pear juice concentrate. This is done by entering "Material plan". This M3 feature shows all planned transactions for the combination of item and warehouse. It also contains information about an item's on-hand balance, and safety stock level. The exact steps this test simulates are the following:

1. Enter credentials to M3 login window and click ”Log in” (M3 start page appears)
2. Click ”Material plan”
3. Enter the article number for pear juice concentrate, i.e. 23840 (list of planned transactions appears)
4. Exit ”Material plan”
5. Exit M3 and click ”Leave page”

As stated, the login process, i.e. step 1, is the interesting part of this test. It includes all steps from when a user clicks the login button to when the start page appears and has finished loading. Notice how step 4 says ”Exit M3, and click ”Leave page”, instead of, for example, ”Log out”. The M3 application used by Kivik does not have a logout function. However, if the user closes the web browser with the M3 application, and clicks ”Leave page” on the pop-up window, he/she have to enter his/her credentials again when entering M3 in a new browser window. The time it takes to perform each step above is measured and visualized in figure 4.2, in the result section below.

Results

Figure 4.2 below is a graph of the response times over time for each step in section 4.2.2 from May 4 to May 14. The login process is the red line, and have higher response times than the other steps. On weekends, the login times varies approximately between 3 s and 8 s. On weekdays they vary approximately between 3 s and 11 s. When measuring the time manually, a weekday at 10:12 and 10:33, it took approximately 7 s and 10 s, respectively, which are close to the times from JMeter. However, lower response times have also been manually measured during the measurement period. The lower times might sometimes be explained by the lack of log out function in Kivik's M3 application. If a user ”logs in” to M3 in a new browser window, while still signed in in another, the application does not have to re-load all data and the ”login process” becomes shorter.

The upper response times, i.e. 8 s and 11s, can seem like a long time for a login. However, a user typically only logs in once a day and then stays logged in until his/her working day has ended. Because of this, the login time has little importance compared to other actions a user performs in M3.

The most distinct peaks, for example during May 4, May 5, and May 12, occurred at 1:00 or 1:20 in the morning. Most peaks for entering ”Material plan” are also outside of working hours. These can for example be caused by scheduled updates and does not necessarily imply any anomalies. There is also a gap, i.e. no test results, between 15:40 - 17:40, as well as 18:00 on May 13. This is, with great probability, due to anomalies from the environment of Company A, see section 4.3.4. The response times of the tested system were too high, so another test started before the last one was finished. As can be seen, this prevented JMeter from executing the login test. It is not clear why this happened. The only thing that can be said is that somehow JMeter was too busy with the test at

Company A, to execute the test at Kivik. This has not occurred any other time during testing.

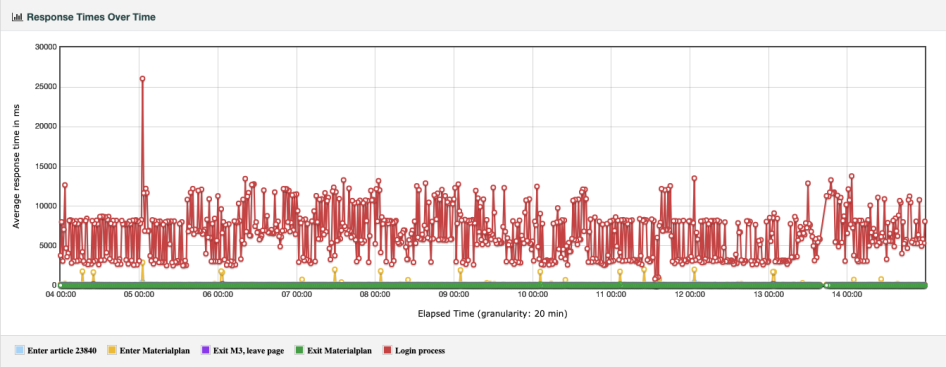

Figure 4.2: Response times over time for each step in the login test.

Below are graphs of how the average and median response time, as well as standard deviation, has changed per day during this measurement period. The graphs 4.3 - 4.7 show that the average response time for each step in section 4.2.2 is 6513 ms, 177 ms, 37 ms, 33 ms, and 62 ms, respectively. The average values are denoted with a blue line, the median values with an orange line, and the standard deviation with a gray line. Since the average times are so different, these graphs must be compared carefully. For example, at first glance it may look like the standard deviation in graph 4.5 varies more than in graph 4.3, which is incorrect. Except from the login time, the response times for the other steps are quite regular. The graph for the login process shows that the average response time increases during workdays, and decreases during weekends. For the other steps, the standard deviation shows some variations, but since the times are so low those variations have little impact on the user experience.

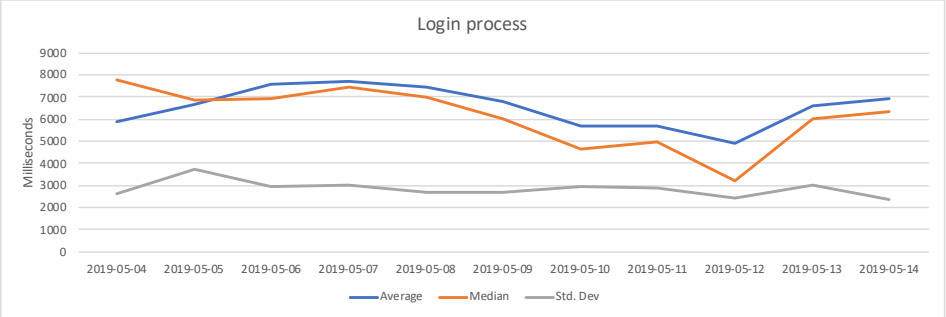

Figure 4.3: Daily changes of the avg. and mdn. response time, as well as std. dev., for the login process.

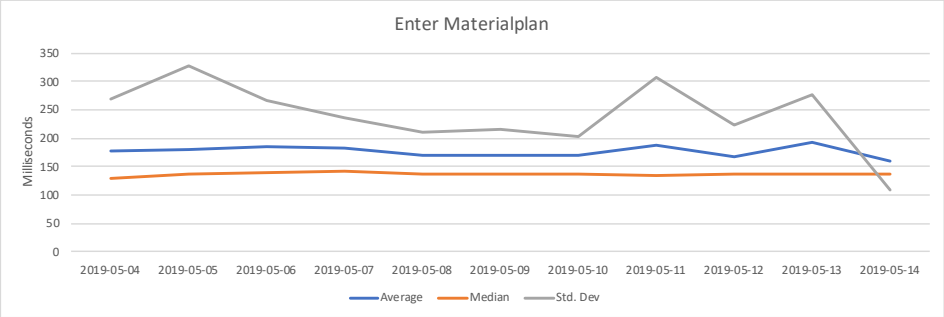

Figure 4.4: Daily changes of the avg. and mdn. response time, as well as std. dev., for clicking "Material plan".

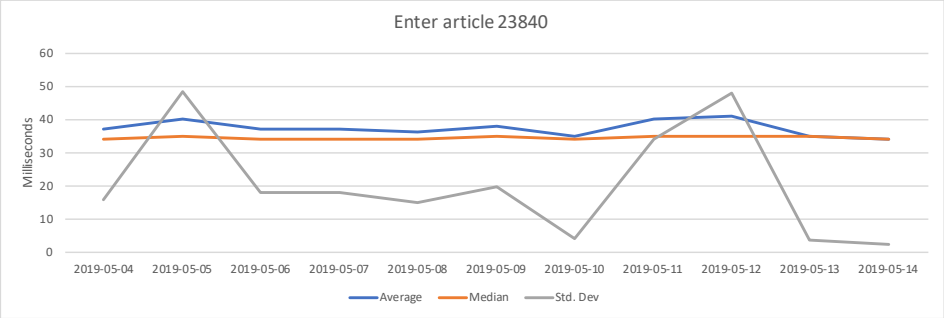

Figure 4.5: Daily changes of the avg. and mdn. response time, as well as std. dev., for entering the article number for pear juice concentrate (23840).

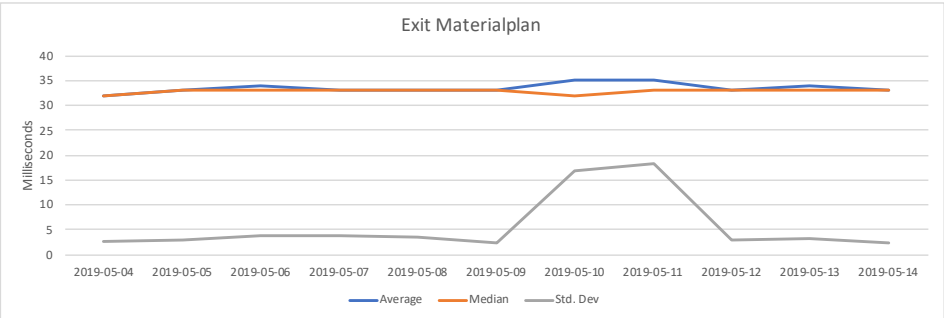

Figure 4.6: Daily changes of the avg. and mdn. response time, as well as std. dev., for exiting "Material plan".

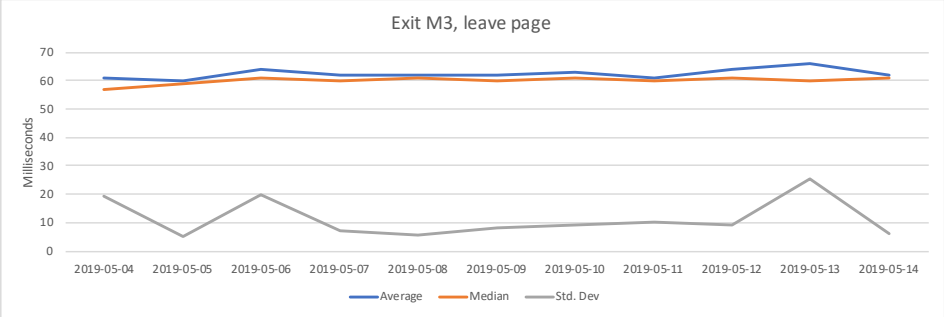

Figure 4.7: Daily changes of the avg. and mdn. response time, as well as std. dev., for exiting M3.

4.2.3 Generate report of inventory values

The other test performed in Kivik's environment generates a report of inventory values every Sunday at 15:00, and measures the time it takes to do so. The report is big, and it is therefore generated on Sundays so it does not interfere with ordinary work. The report contains inventory values of all items in the whole factory, including warehouses, which depend on what location and what stage of the production and logistic process the items are at.

This test was partially done for familiarization, for example it required more manual scripting than the login test. Some obstacles included calculating the time difference of when the report started to be generated until it was finished. The test was, however, mainly done for having the report generated automatically and scheduled. Previous to this test the report was generated manually and because of that, it was sometimes forgotten. This is of course irritating and results in having to generate two reports the next Sunday, which takes much longer time. This test therefore enables free time for the system owner and an assurance that the report will be generated.

Kivik does not have any requirements, considering time, on this report. Since it is generated when no other employees are using the system, it is acceptable if it takes a long time to generate. However, by knowing the time it usually takes to generate the report, Kivik will instantly know if something extends that time and can start investigating that. It also adds knowledge of when the data in the report can be visualized and analyzed.

The test creates a report by entering information about inventory valuation round, exchange rate type, name of report, inventory valuation date, whether it should include lines with zero quantity, usage of zero as actual value, whether it should exclude goods in transit, valuation type, report layout, and optional text for report header. The report is sent to the email address of the system owner and this is confirmed after entering the information above. Finally, the report is scheduled to "now" in M3. Crontab is used to schedule the report generation to every Sunday at 15:00. The measured times for generating reports on several Sundays can be read in table 4.1 in the result section below.

Results

The table below shows the total amount of time it took to generate the report of inventory values, as well as the date when each report was generated. The average generating time is 23 min and 40 s. When speaking to the system owner about this test, he said that it took approximately 20 minutes. The results given from the test is therefore considered reasonable and good. The generating time is similar on all Sundays, but it increases each week. This is most likely not an anomaly, but instead due to the fact that Kivik have the majority of their sales during summer. As summer approaches there occurs more transactions to be able to keep the warehouses full when summer has arrived. There are more data to process and this has an impact on the report generating time. However, since the results are analyzed the workdays after the report has been generated, it does not matter much if the generating time takes a few minutes longer or not. Also, as already has been mentioned, this report does not interfere with actual users of the system since it is generated on Sundays when no employees are working.

Date	**Total time**
2019-04-21	23 min 4 s
2019-04-28	23 min 24 s
2019-05-05	23 min 52 s
2019-05-12	23 min 57 s
2019-05-19	24 min 6 s

Table 4.1: The total time it took to generate reports of inventory values and what date the reports were generated.

4.3 Test case 2: Company A

Since Company A in this thesis is anonymous, not much can be revealed about it. However, what can be said is that it is a large company, much larger than Kivik, and it has several offices abroad. After an update of Company A's ordering system, the users said they experienced it as "very slow". Everyday tasks took very long to perform, but it was not slow constantly. This varied and therefore it was hard to be sure if the problem still existed or not. To see if there still was a problem, continuous testing was performed. The following sections will discuss the system environment, the test, and how continuous testing was used when trying to find a solution to the problem.

4.3.1 The system environment

Figure 4.8 below, see appendix A for a larger figure, shows the system environment after the update. This environment is hosted by AWS. When the tester wants to access the ordering system, which is done via a web browser, it starts off by entering the VPC in Dublin through a firewall. This VPC is part of AWS account 1,

owned by Company A. The user then enters subnet 1 and the reverse proxy. This component sends the user's request to the tomcat server in subnet 3, which provides the interface of the ordering system to the user. This component exchanges information with M3, and receives data about the articles' inventory balance, among other things. To receive this, the tomcat server sends requests to the IPsec in subnet 2. With the IPsec connection, the requests transfers from AWS account 1 and the VPC in Dublin, to AWS account 2, owned by Infor, and the Infor M3 VPC in Frankfurt. The requests are sent to the application server which communicates with the Microsoft SQL server and then sends back the requested information.

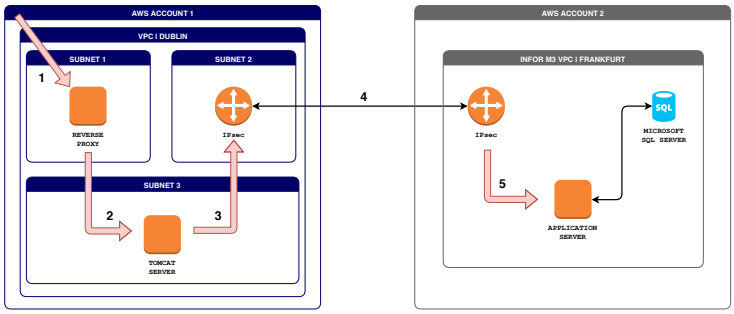

Figure 4.8: The system environment after Company A's update.

4.3.2 The test

The test simulates a user who logs in to the ordering system, creates an order of 52 articles, 5 of each, which ends up in a cart, deletes the order, and lastly logs out. After logging in, the user can choose to work in either the test environment or the production environment. Since this test is to simulate an actual user, the testing occurred in the production environment. The test was executed every 15 minutes, first only on evenings and during a weekend, to make sure that real employees were not disturbed. It was later also desirable to execute it during the day so that any potential anomalies would be spotted. No noticeable effects for real employees were reported when the test was executed during work hours. When trying to fix the problem, the test was run to immediately be able to tell if the fix was successful or not. The actual steps the simulated user performed are the following:

1. Enter URL to ordering system (login page appears)
2. Enter credentials and click "Log in"
3. Choose production environment (the start page appears)
4. Click items (all available articles are listed)
5. Write quantity 5 on the first 52 articles and click "Buy chosen" (the articles are placed in the cart)
6. Click on shopping cart (window with chosen articles appears)
7. Click "Confirm order" (order finalization appears)

8. Click ”Remove order”
9. Log out

4.3.3 The continuous testing process

While investigating all steps of the process, it was discovered that the application server was overloaded, so more memory was assigned to it. Employees who accessed M3 and the application server in Frankfurt directly said that they experienced faster responses. However, measurements show, see graph in figure 4.11, that this was not the case for employees who entered the ordering system and the tomcat server in Dublin. It is also important to note that in this case, the load on the system had no impact on the response times. The graph shows that the response times were basically the same during the afternoon, when the load is high, and during night, when the load is at its lowest. The slowness of the system was ”greater” than the impact of the load.

It was also discovered that there was a latency of approximately 23 ms between the tomcat server in Dublin and the application server in Frankfurt. A new VPC, located in Frankfurt, was created by AWS account 1 and the tomcat server was moved to this VPC. Figure 4.9, see appendix A for a larger figure, visualizes the system environment after the move. After entering the reverse proxy in the Dublin VPC, the user request reaches the tomcat server in the Frankfurt VPC via peering. A new IPsec connection was established between the VPCs in Frankfurt, which carried the tomcat server's requests to the application server. As can be seen by the graph in figure 4.12, this appeared to solve the problem. The response times became lower, especially for writing the quantity and clicking ”Buy chosen”.

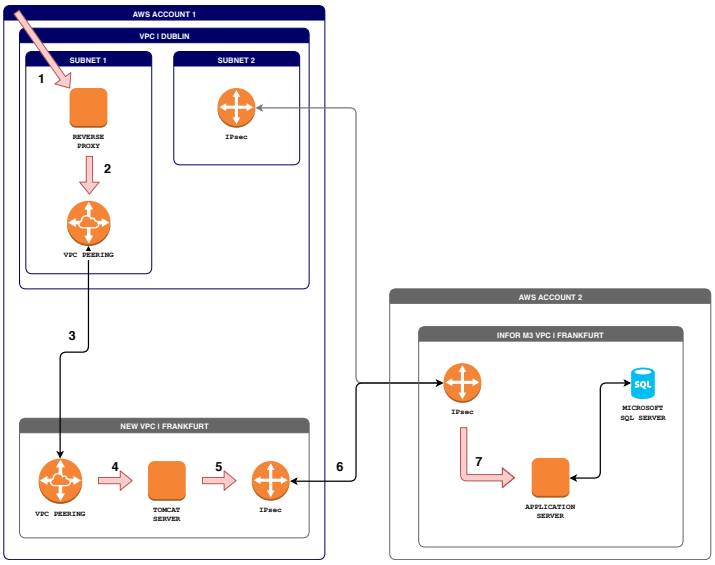

Figure 4.9: The system environment after moving the tomcat server to a new VPC in Frankfurt.

The test was first not allowed to be performed during the day when real employees used it so after the above result was generated, the testing stopped. However, when starting to use the system again, some employees still experienced slow response times. It turned out that there still were some dependencies left in Dublin for the tomcat server. A Microsoft SQL server remained, which provided user data and some frameworks to the tomcat server. For example, one framework created the grid used when listing the articles. The grid is then filled with data by the SQL server communicating with the application server. The remaining SQL server was also moved to the new VPC in Frankfurt, and the new layout of the system environment is shown in figure 4.10. A larger figure can be found in appendix A. The test was run during the move, but it was run every 5 minutes instead of every 15 minutes as before. The response times were so high that the test exceeded 5 minutes and another test started before the last one was finished. Therefore, the file with all results included results from a new test in between the results of an unfinished test. This caused an error in JMeter when trying to generate the graph. It was not known where the errors in the file occurred or their exact appearance, so troubleshooting had to be done manually. Since the resulting file included so much data it was not possible to go through it line by line and large parts of data were therefore removed without having been looked at first. This resulted in that only a few test runs were saved and generated. The graph in figure 4.13 shows the result from four test runs before the move. The graph in figure 4.14 shows results from after the move, but with 15 minutes intervals.

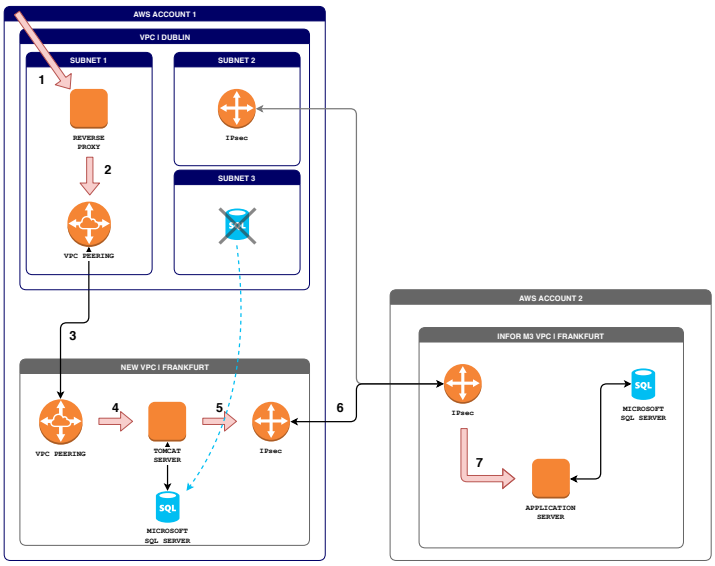

Figure 4.10: The system environment after moving the SQL server to the new VPC in Frankfurt.

4.3.4 Results

Below are graphs showing the continuous testing process, that is the response times over time when trying to fix the problem. Each graph shows a different time period of the process. Figure 4.11 shows when more memory was assigned to the application server. Figure 4.12 shows when the tomcat server was moved to Frankfurt. Figure 4.13 shows a few test runs before the remaining SQL server was moved to Frankfurt. Finally, figure 4.14 shows multiple test runs after the move was finished. These graphs were very helpful when trying to solve the problem. They show in a clear way if the response times have improved or not, and can give this answer ”immediately”. There is no need to involve users and see how they experience the system after a potential solution. It could also be that some users experience it differently than others, as was the case when assigning more memory to the application server. This test can confirm if the problem is solved or not. The three actions that take the longest to perform is ”Click items and list articles”, ”Write quantity and click Buy chosen”, and ”shows order finalization”. Therefore, it is the response times of these actions that were mostly analyzed.

This test case is only relevant because Company A has its ordering system in the cloud. If this was not the case, it would be extremely hard to put an application server in Frankfurt, if all other components are in Dublin, and not noticing it in time. In the cloud however, it is much easier to make such mistakes.

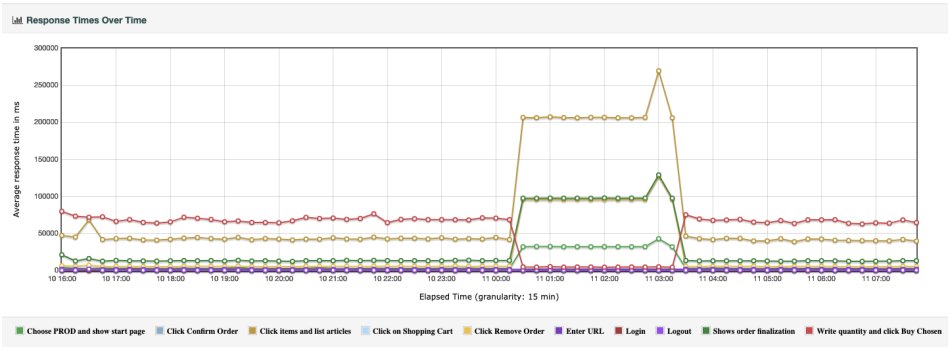

Figure 4.11: Response times over time after more memory was assigned to the application server.

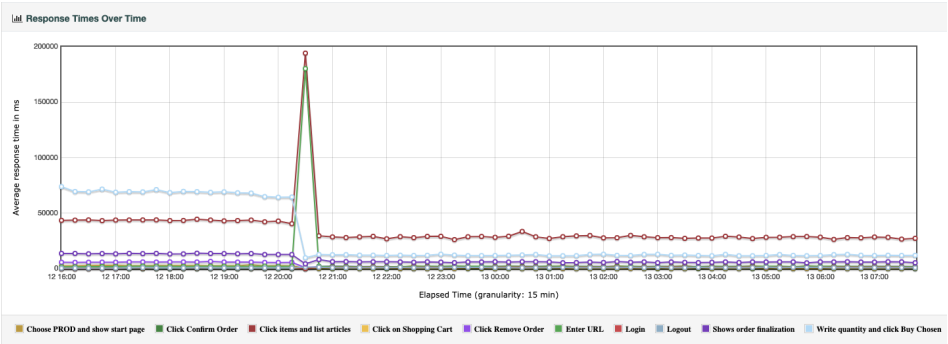

Figure 4.12: Response times over time after moving the tomcat server to Frankfurt.

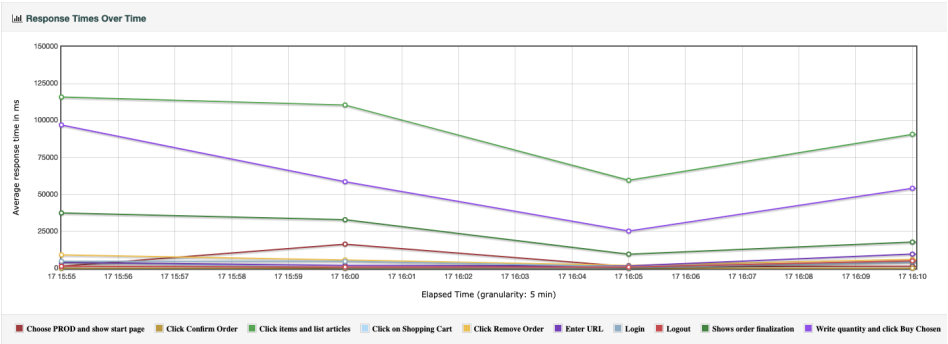

Figure 4.13: Response times over time before moving the remaining SQL server to Frankfurt.

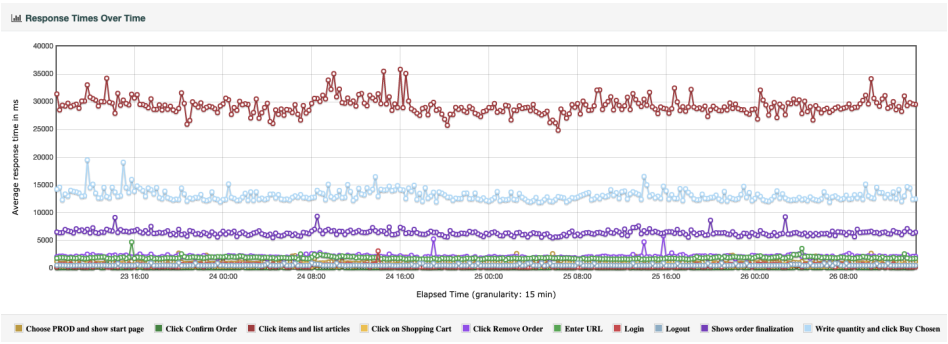

Figure 4.14: Response times over time after all fixes.

The graphs below, figure 4.15 - 4.24, show how the average and median response time, as well as standard deviation, changes daily for each step in section 4.3.2 after the last fix, namely between April 18 and May 12. All median response times are quite linear, the exception being for the login process. However, since the login times are so low, this has little impact on the user experience. There are some interesting peaks regarding the standard deviation during this period. The peak on April 26 in figure 4.17 and 4.18, is caused by one single peak at 18:15. Since there are no other peaks that day, this could be a coincidence. After this, however, an extra eye were kept on the ordering system's response times.

Next large change in standard deviation occurred on April 29, which was experienced for all steps. That day there were several peaks in response time, approximately between 15:30 and 17:45. For executing step 4, i.e. "clicking items and listing all articles", the average response time is 30 s. This day, however, the highest measured value was 518 s (approximately 8.6 min) and occurred at 16:15. This time is quite alarming. The rest of the week, the times were normal but next Monday, May 6, similar issues occurred again. There is one peak for performing step 4 at 13:15, and then there are several peaks approximately between 14:30 and 17:00. The highest response time was measured to 271 s (approximately 4.5 min) and occurred at 16:00. The time is lower but still alarming. Each workday that week has some period of peaks in response time, often around 16:00 or 17:00. The reason for these high values is unclear. It could be that more people are using the system at these times, perhaps because of an ordering deadline, but the times are still very high. Interesting is that no user have complained about it yet. Perhaps they experience it as temporary and want to wait awhile before reporting it.

On May 13, the response times were so high that they exceeded 15 minutes and caused an error when trying to generate the graph. Why the response times were so high is not clear, but it seems as if they were so high that they kept Jmeter from running the test for Kivik. This is something a system owner must have in mind. The conclusion to draw from this is that it is probably better, and safer, to execute one single type of test from one single type of device. To make the testing process even more secure, the JMeter AutoStop Listener can be used. This plugin can stop a test if it exceeds a chosen average response time, average latency, or error rate. The result from May 13 did not, however, have any seemingly impact on the results on May 14. For executing step 4, there are some peaks starting from 11:15 to 17:45, and the highest response time is 73 s, measured at 13:30, which is relatively low.

There are more peaks during the recent weeks compared to just after moving the remaining SQL server. The response times have not increased gradually, but by looking at the results from May 6 to May 10, it seems as if the peaks start to appear quite regularly. The graphs and response times will keep being analyzed and watched carefully, but until a user actually complaints, not much resources will be assigned to look into this anomaly.

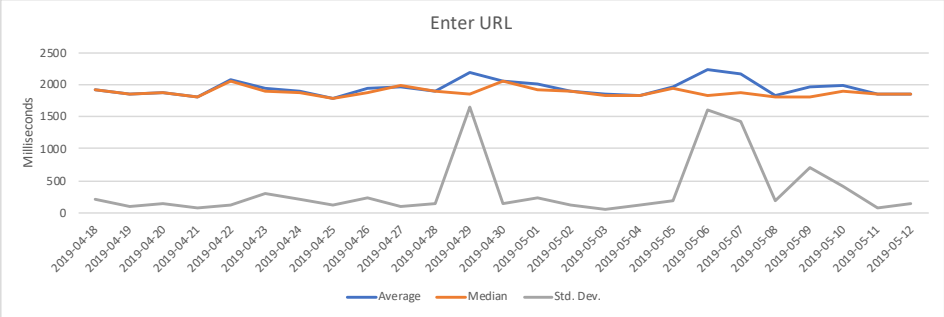

Figure 4.15: Daily changes of the avg. and mdn. response time, as well as std. dev., for entering the URL (and showing the login page).

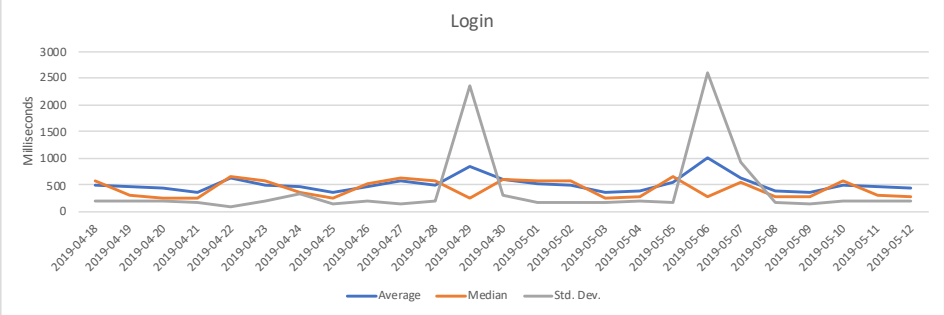

Figure 4.16: Daily changes of the avg. and mdn. response time, as well as std. dev., for logging in.

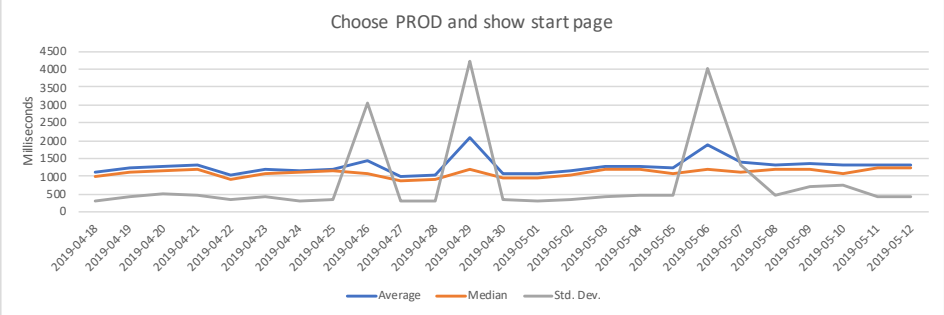

Figure 4.17: Daily changes of the avg. and mdn. response time, as well as std. dev., for choosing production environment (and entering start page).

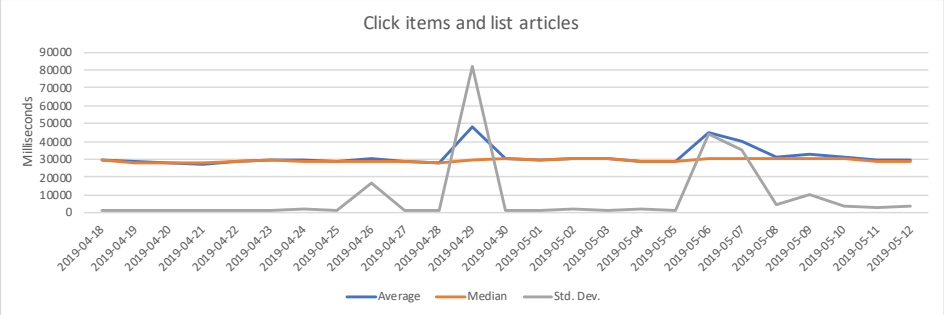

Figure 4.18: Daily changes of the avg. and mdn. response time, as well as std. dev., for clicking items (and listing all articles).

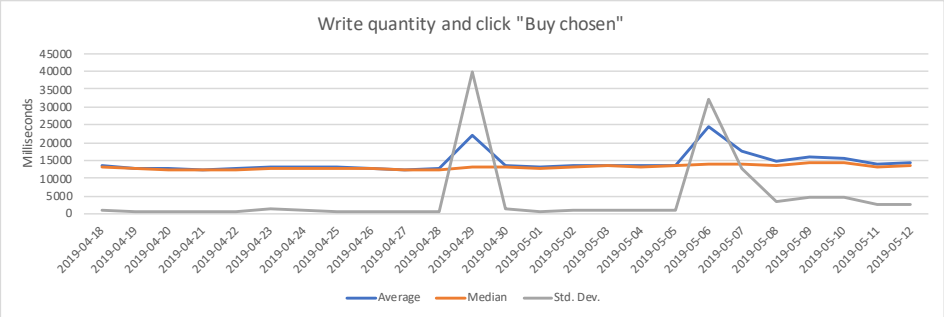

Figure 4.19: Daily changes of the avg. and mdn. response time, as well as std. dev., for writing quantity and clicking Buy chosen.

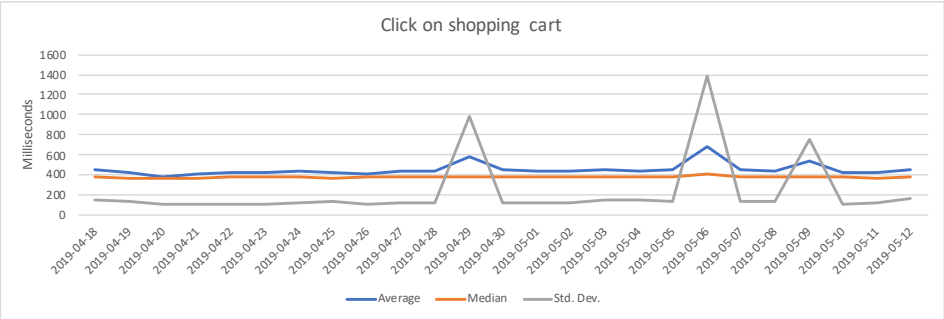

Figure 4.20: Daily changes of the avg. and mdn. response time, as well as std. dev., for clicking on shopping cart.

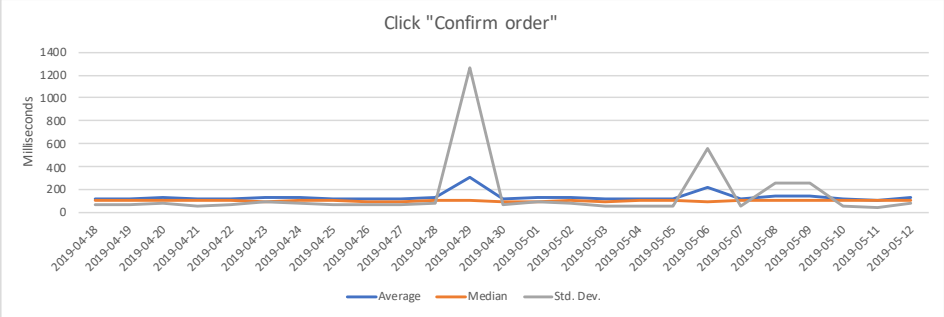

Figure 4.21: Daily changes of the avg. and mdn. response time, as well as std. dev., for clicking Confirm order.

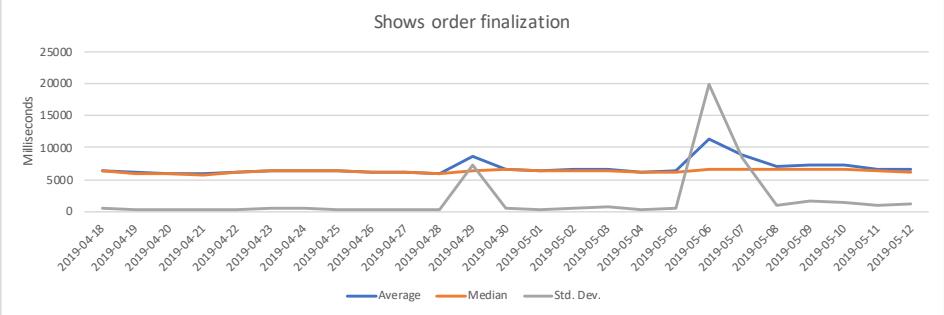

Figure 4.22: Daily changes of the avg. and mdn. response time, as well as std. dev., for showing the order finalization.

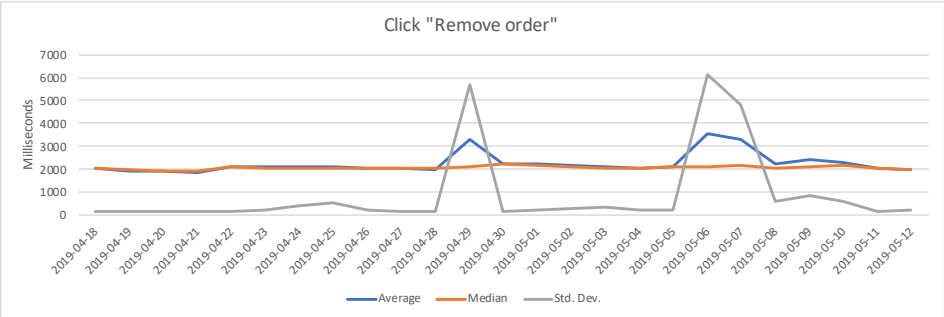

Figure 4.23: Daily changes of the avg. and mdn. response time, as well as std. dev., for clicking Remove order.

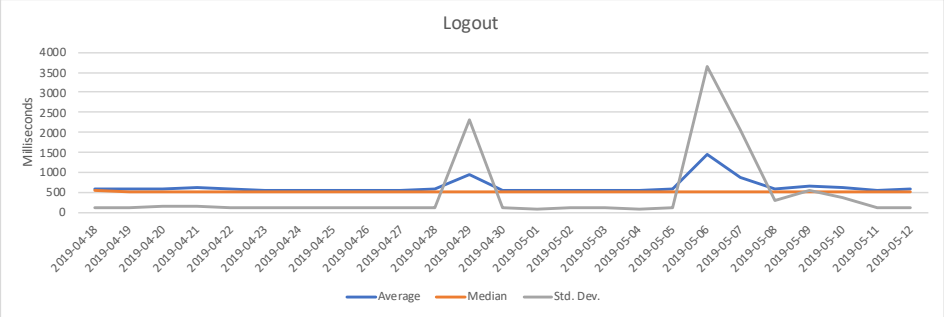

Figure 4.24: Daily changes of the avg. and mdn. response time, as well as std. dev., for logging out.

Chapter 5
Evaluation

This thesis work uses Apache JMeter to create tests for validation and crontab to run them continuously. Both of these programs are accessible in an easy way. JMeter is an open source application, free to download to any computer. Crontab is available for all Mac and Linux users. Both programs are experienced as user-friendly with much information and many tutorials available. Using the JMeter recording template was very convenient and a test could be produced in an easy way. However, the manual scripting was sometimes experienced as troublesome, for example to save a value to an already defined variable. This may be explained by inexperience in Groovy. It was also desirable to be able to validate the JMeter tests in an easier way, for example that JMeter played back what it had recorded. Using the developers tool in the web browser requires some understanding of the protocol HTTP, and all the manual work can be a drawback for some companies. For the multiple case study in this thesis, it did not give the desired certainty that the test actually performed the correct actions. When using crontab it is important to know what time zone the device executing the test is located in. For example, when first using crontab it was believed that it did not work properly since the test was not executed in time. Troubleshooting showed, however, that the clock of the device was 2 h behind.

This measurement method is considered easy to use and relatively easy to start using. However, there are a lot of manual steps included, for example when validating that the test is doing what is expected, or changing login IDs. It can take some time to have a validated and finished test ready. During the thesis, monitoring was encountered, particularly OpsRamp which another client of Elastic Mobile was going to start to use. This Monitoring as a Service is designed for IT in hybrid and multi-cloud environments. It can monitor applications, servers, networks, storage, and database instances, to capture behavioral and performance metrics [24]. For companies, who generally are looking for a tool which requires the least amount of work, perhaps OpsRamp had been better in that sense.

5.1 The measurement setups

Three setups are used in this thesis, where tests are executed either from an EC2 instance, from a Raspberry Pi, or from a Raspberry Pi using remote testing. All three setups can be useful for a system owner. The EC2 instance has a scalable size and can therefore carry multiple tests. More tests can also be added over time, and this setup does not require any extra physical device. However, as can be read in section 4.2.2 and 4.3.4, it is probably better to carry out only one single test on each single device. If the system or application to be tested is behind a lot of firewalls and similar network elements, it might be troublesome to execute tests from the cloud. Because of its size, price, and user-friendliness, the Raspberry Pi can be used instead. This device can easily be placed within the same network as the target and execute tests to it directly. However, it has limited memory and is not scalable. If the target is to be tested from several different offices or by a large load, the remote testing setup is the best choice. The JMeter master does not have to be an EC2 instance. It can, for example, also be a Raspberry Pi. However, since an EC2 instance can hold such a large amount of memory it can be better to use this instead of a device with limited memory. Of course one must remember that if more memory is assigned, it will also cost more money.

5.2 Requirements of the system

One starting point of this thesis was that companies have specified requirements of their systems, such as a maximum login time. It was soon realized that this was not the case. The companies in the test cases had no specific requirements considering response time on their systems other than that the users should be satisfied. If the system owner does not hear any complaints, he/she expects that the system is working. However, all complaints might not reach the system owner, for example if a user thinks its problem is minor. An anomaly can also reach the owner much later after its first occurrence, if a user first thinks it is temporary. This measurement method gives a company the opportunity to gain the knowledge of what response times a user can expect. The company gets a baseline of their system and can use this after updates or upgrades to validate that the system still works as expected. Instead of having employees saying that the system *feels* slow, this test can validate if this is the case or not. The system owner can also see anomalies directly and does not need to wait for a user to report it. Finally, as this thesis shows, this measurement method works both for large international companies, as well as medium-sized Swedish companies.

Chapter 6
Conclusion

In this thesis, a method to perform measurements on a business system in a multi-cloud environment, as defined by the last paragraph in section 3.2, is developed. To evaluate this method, a multiple case study is performed on two different companies. These are Kivik, a Swedish medium-sized company, and Company A, a large international company. The measurement method is considered successful for both test cases. It provides both companies with more knowledge of how their business systems are working, as well as knowledge that they keep working as expected. Anomalies can be detected fast, which provides the possibility that the anomaly can be resolved fast. Test case 2 shows that this measurement method also can be used in such a case. JMeter and crontab are used to construct the tests and run them continuously. They are relatively easy to use. The main problems encountered with JMeter are the manual scripting and the validation process, which can be explained by inexperience and uncertainty. All three measurement setups were successful, and easy to use. However, the testing process show that executing several tests from the same device can cause problems if one of the tests experiences errors. To further secure that no test interrupts another, the JMeter AutoStop Listener can be used where JMeter will stop the test on a certain runtime basis. In total, there were three different tests performed, one testing a login process, one generating a report of inventory values, and one testing an ordering system. The measurement method worked well for all tests. With this Master's thesis, the research questions in section 1.4 can be answered. These answers are found below.

Is it possible to perform continuous validation of a multi-cloud system by automated test scripting and if so, what advantages are there?

For the definition of multi-cloud stated in the last paragraph in section 3.2, this thesis shows that it is possible to perform continuous validation of a multi-cloud system by automated test scripting. Some advantages of continuous validation is mentioned in section 2.1. For example, continuous validation gives knowledge about how changes and new features affect a system. It also gives evidence that an application, or in this case a system, continues to work as expected in the present if it worked well in the past.

How can continuous validation by automated test scripting be performed?

This thesis presents a measurement method with three different, but similar, setups. JMeter and crontab are used together for creating tests and then run them continuously. The tests are either run from an EC2 instance in the AWS cloud, a Raspberry Pi or a combination of the two in a remote testing setup. Khanghahi and Ravanmehr in [12] measures performance based on simulation by using the tool CloudAnalyst. With this tool they configure data centers, geographic areas, and users, to be able to test the wanted scenario. Gao et al. [5] perform tests on cloud-based software using Testing as a Service. They use EC2 as the cloud infrastructure and by using Amazon's EC2 CloudWatch APIs they can use their TaaS infrastructure for performance validation.

What software tools can be used to perform continuous validation?

As stated, this thesis uses JMeter to create tests for validation and crontab to do this continuously. Another tool for continuous validation is the continuous validation framework from xLM, mentioned in Section 2.1. Section 2.3 also mentions several other software tools for creating tests, for example Gatling and LoadStorm which are similar to JMeter. However, in contrast to JMeter and Gatling, LoadStorm is cloud based which allows for testing to be done from any computer with internet access. The section also mentions CloudAnalyst which is built on top of the simulation toolkit CloudSim. More software tools exists however, and the ones mentioned here have been selected randomly.

What should be measured to know if the business system is working as specified for the system owner?

Basically, the only specification present in each test case is that the users should be satisfied and not complain. Measurements are done on a login process, the process of generating a report, and the process of laying an order in an ordering system. The response times are saved and visualized in graphs. These tests are measuring the performance. Other metrics one can measure to gain more information about the performance, as stated in section 2.2.1, are recovery, network bandwidth, availability, number of users, scalability, and latency. Section 2.2.2 gives more metrics to measure the scalability, for example performance/resource ratio, performance change, and performance variance. With the TaaS infrastructure by Gao et al. one can monitor allocated computing resources, like CPU or data storage. One can also monitor resource usage, and system performance in terms of system utilization. Finally, one can also evaluate the system load based on the user access load, the network load, and the data access load.

6.1 Future work

The next step for the measurement method would be to automate it more and decrease the manual work. A script would be written so that after a test have been executed several times, for example every half hour during a week, those results would be generated to graphs. To secure the testing process, different ways

to stop the test if it exceeds a chosen time would be investigated. If, for example, the JMeter AutoStop Listener would be the best choice, this would be implemented in the tests. Currently, the response times for different actions may be analyzed several days after the actions were performed. This could mean that potential anomalies are not spotted in time. This could be solved by sending some sort of warning to the tester if, for example, the test was stopped because it exceeded a chosen response time. If more time was assigned to this thesis, how a warning can be sent would be investigated. Also, more advanced tests involving more parts of the business systems would be performed to further test the developed measurement method and investigate its limits.

Chapter 7
Word Definitions

The following list contains definitions and explanations of expressions from this report in alphabetical order.

API stands for Application Programming Interface and is a set of commands, functions, protocols, and objects that can be used to create software or interact with external systems.

DDoS attacks denies access to a server, service, or network by overwhelming it with internet traffic so normal traffic is disrupted.

Enterprise Resource Planning (ERP) is an adaptive software system which unifies the main functional areas of an organization's business processes to one system.

Latency is measured from just before sending a request to just after the response has started to be received.

Load testing is the process of testing the behavior of a system when it is subject to a load close to the limits of its specifications.

Network Utilization is the amount of network traffic at a given moment, compared to the maximum amount of traffic the network can handle.

Regression Testing is the process of testing a system after changes have been introduced, to make sure that existing functionality is still intact.

Response Time is defined as the total amount of time it takes for a request to be sent until a response has been fully received.

Service Level Agreement is an agreement between a service provider and a client, containing requirements of a certain service, such as costs and speeds.

Smoke Testing is the process of ensuring that the most critical or basic functions of a system is working.

SSH Port Forwarding is a method for directing data from a client machine to a server machine, or vice versa, over an encrypted SSH connection.

Vendor Lock-in means that a customer of a product or service becomes dependent of the vendor offering it and can not, in an easy way, switch to the product or service of a competitor.

References

[1] N. Talens, 2015, *After continuous integration there is continuous validation*, Codecentric. `https://blog.codecentric.de/en/2015/08/after-continuous-integration-there-is-continuous-validation/` (Accessed: 2019-02-18)

[2] `https://www.continuousvalidation.com/` (Accessed: 2019-02-18)

[3] W.-T. Tsai, Y. Huang, Q. Shao, 2011, "Testing the Scalability of SaaS Applications", *Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*

[4] X. Bai, M. Li, B. Chen, W.- T. Tsai, J. Gao, 2011, "Cloud testing tools", *Proceedings of the 6th IEEE International Symposium on Service Oriented System Engineering (SOSE)*

[5] J. Gao, K. Manjula, P. Roopa, E. Sumalatha, X. Bai, W. -T. Tsai, T. Uehara, 2012, "A cloud-based TaaS infrastructure with tools for SaaS validation, performance and scalability evaluation", *Proceedings of IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom'12)*

[6] `https://aws.amazon.com/cloudwatch/` (Accessed: 2019-03-05)

[7] `https://gatling.io/` (Accessed: 2019-02-20)

[8] `https://www.scala-lang.org/` (Accessed: 2019-02-20)

[9] B. Wickremasinghe, R. N. Calheiros, R. Buyya, 2010, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications", *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*

[10] R. Buyya, R. Ranjan, R. N. Calheiros, 2009, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities", *Proc. of the 7th High Performance Computing and Simulation Conference (HPCS 09)*

[11] `https://loadstorm.com/` (Accessed: 2019-03-11)

[12] N. Khanghahi, R. Ravanmehr, 2013, "Cloud computing performance evaluation: issues and challenges", *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 3, no. 5

[13] S. Vonnegut, 2017, *Cloud or Clouds? How and Why to Choose a Single or Multi-Cloud Approach*, Stratoscale. https://www.stratoscale.com/blog/it-leadership/cloud-clouds-choose-single-multi-cloud-approach/ (Accessed: 2019-03-13)

[14] M. Rouse, 2014, *Definition: noisy neighbor (cloud computing performance)*, TechTarget. https://searchcloudcomputing.techtarget.com/definition/noisy-neighbor-cloud-computing-performance (Accessed: 2019-03-13)

[15] https://www.infor.com/products/m3 (Accessed: 2019-03-13)

[16] https://jmeter.apache.org/ (Accessed: 2019-03-14)

[17] https://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html (Accessed: 2019-03-14)

[18] M. Hauck, M. Huber, M. Klems, S. Kounev, J. Müller-Quade, A. Pretschner, R. Reussner, S. Tai, 2010, "Challenges and opportunities of cloud computing", *Karlsruhe Reports in Informatics 19*

[19] P. Mell, T. Grance, 2009, "The NIST definition of cloud computing", *National Inst. Standards Technol.*, vol. 53, no. 6

[20] https://aws.amazon.com/about-aws/ (Accessed: 2019-03-15)

[21] S. Watts, 2017, *SaaS vs PaaS vs IaaS: What's The Difference and How To Choose*, BMC. https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/ (Accessed: 2019-03-26)

[22] S. Goyal, 2014, "Public vs Private vs Hybrid vs Community - Cloud Computing: A Critical Review", *International Journal of Computer Network and Information Security (IJCNIS)*, vol. 6, no. 3

[23] S. D. Lowe, 2018, *Multi-cloud vs. hybrid cloud: Assessing the pros and cons*, TechTarget. https://searchstorage.techtarget.com/feature/Multi-cloud-vs-hybrid-cloud-Assessing-the-pros-and-cons (Accessed: 2019-04-05)

[24] https://www.opsramp.com/ (Accessed: 2019-04-05)

[25] https://www.kiviksmusteri.se/en/ (Accessed: 2019-04-08)

[26] http://groovy-lang.org/ (Accessed: 2019-04-08)

[27] https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html (Accessed: 2019-04-15)

[28] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (Accessed: 2019-04-15)

[29] https://opensource.com/resources/raspberry-pi (Accessed: 2019-04-26)

[30] https://www.raspberrypi.org/ (Accessed: 2019-04-26)

[31] https://www.adminschoice.com/crontab-quick-reference (Accessed: 2019-05-02)

[32] https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/ (Accessed: 2019-05-02)

[33] https://www.oracle.com/technetwork/java/javaee/overview/index.html (Accessed: 2019-05-02)

[34] https://www.oracle.com/technetwork/java/index-jsp-135475.html (Accessed: 2019-05-02)

[35] https://www.oracle.com/technetwork/java/javaee/jsp/index.html (Accessed: 2019-05-02)

[36] https://tomcat.apache.org/ (Accessed: 2019-05-02)

[37] https://www.techopedia.com/definition/2450/peering (Accessed: 2019-05-03)

[38] https://docs.aws.amazon.com/vpc/latest/peering/what-is-vpc-peering.html (Accessed: 2019-05-03)

[39] https://www.microsoft.com/sv-se/sql-server/sql-server-2017 (Accessed: 2019-05-03)

[40] M. Rouse, 2014, *Definition: IPsec (Internet Protocol Security)*, TechTarget. https://searchsecurity.techtarget.com/definition/IPsec-Internet-Protocol-Security (Accessed: 2019-05-03)

[41] R. K. Yin, 2014, "Case study research : design and methods", 5 ed., London: SAGE.

[42] M. A. AlZain, E. Pardede, B. Soh, J. A. Thom, 2012, "Cloud Computing Security: From Single to Multi-clouds", *Proceedings of The 2012 45th Hawaii International Conference on System Science (HICSS)*

Appendix A

This thesis' cloud environments

Below are larger figures of the cloud environments in this thesis, i.e. the environments of Kiviks Musteri AB and of Company A.

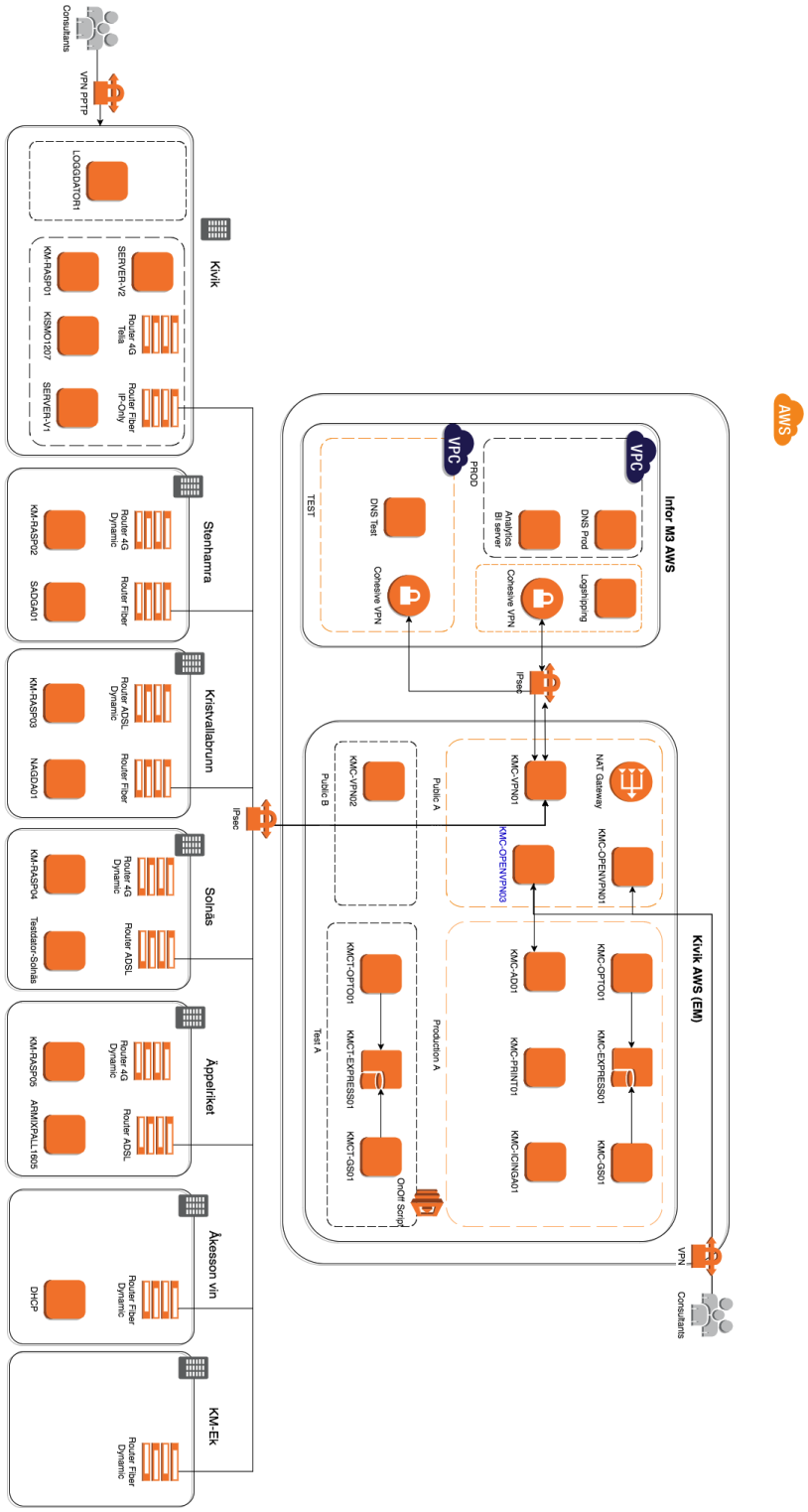

Figure A.1: The cloud environment of Kiviks Musteri AB.

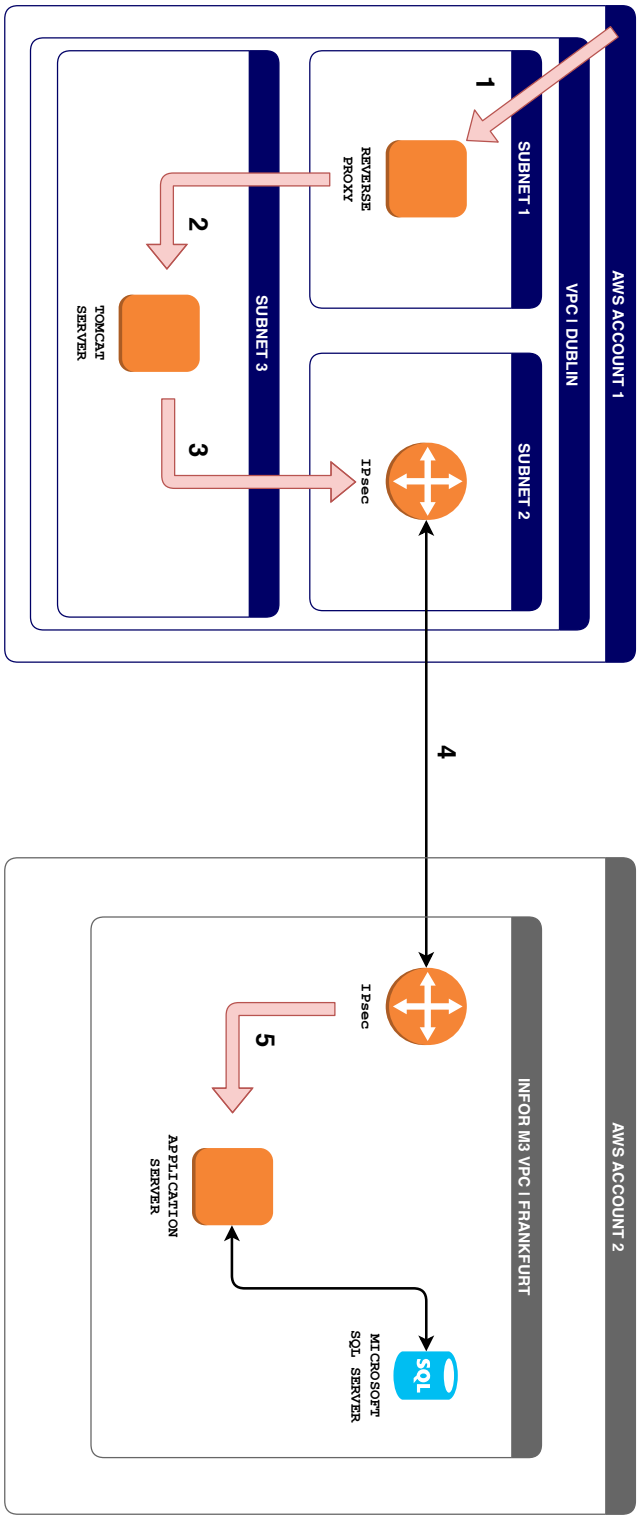

Figure A.2: Company A's ordering system environment after their update.

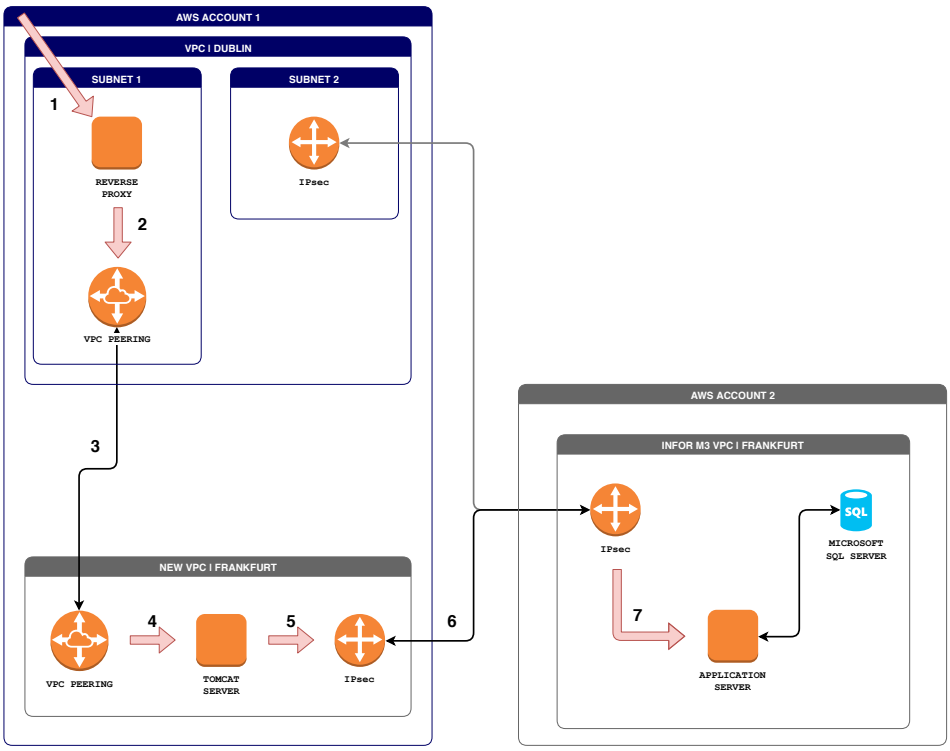

Figure A.3: Company A's ordering system environment after moving the tomcat server to a new VPC in Frankfurt.

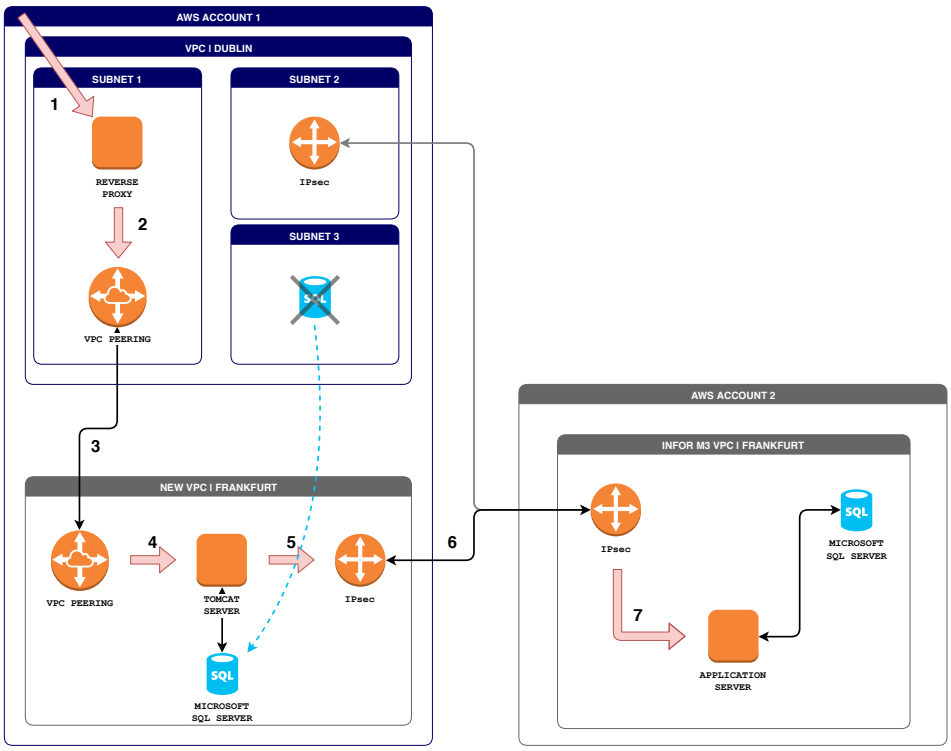

Figure A.4: Company A's ordering system environment after moving the SQL server to the new VPC in Frankfurt.

Printed by Tryckeriet i E-huset, Lund 2019

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-704
http://www.eit.lth.se