

ACIT5900
MASTER THESIS

in

Applied Computer and Information Technology (ACIT)
May 2023

Cloud-based Services and Operations

**Analyzing and Benchmarking the Performance of
Different Cloud Services for Agile App Deployment**

Mikkel Hiorthøy

Department of Computer Science
Faculty of Technology, Art and Design

OSLOMET

Abstract

Cloud computing has become a popular method for organizations to deploy and host websites. The cloud offers several innovative solutions and services for these tasks, but they also bring new challenges. Selecting the appropriate cloud platform and deployment method for a specific use case can be challenging due to the wide variety of options available in the market. The selection is also usually based on the specific requirements and technologies of the use case. This thesis aims to assist in the process of selecting a cloud platform and provide benchmarks and analysis of several cloud service providers (CSPs) and deployment methods. Two deployment methods are presented. The first method involves automatic deployment on CSPs through GitHub integration, while the second method utilizes a manual Dockerized approach with a virtual machine instance deployed in OpenStack. Analysis and benchmarks are conducted to obtain metrics based on resource usage and performance. The results obtained will be further analyzed to decide upon the CSP with the optimal performance. Huldra will be used as a test case for the deployment and hosting project. This is a React based survey framework developed by SimulaMet in Oslo, Norway. SimulaMet is currently exploring new approaches and services for deploying and hosting a Huldra survey in the cloud, which makes it a relevant use case. The thesis conclusion includes the approaches that can be applied to benchmark and analyze cloud services and deployment methods and assist in the selection of an alternative. The findings of the study are further presented which led to the conclusion that one CSP stood out as the optimal choice among the alternatives included.

Keywords: Cloud deployment, Cloud service providers (CSPs), Docker, React, OpenStack, Render, Netlify, Railway, Vercel

Acknowledgement

I would like to express my gratitude to my supervisors Cise Midoglu, Saeed Sabet, and Pål Halvorsen for the opportunity to participate in this project, and for the guidance they have provided. I would also like to thank my professors at OsloMet for valuable education throughout the master's degree.

Contents

- 1. Introduction 7
 - 1.1 Background and motivation 7
 - 1.2 Problem statement 9
 - 1.3 Scope and limitations 10
 - 1.4 Research methods 11
 - 1.5 Ethical considerations 11
 - 1.6 Main contributions 12
 - 1.7 Thesis outline 12
- 2. Background 14
 - 2.1 Cloud basics 14
 - 2.1.1 CSPs 14
 - 2.1.2 SaaS, IaaS, PaaS 15
 - 2.1.3 Benchmarking resources 16
 - 2.2 Related work 16
 - 2.2.1 Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with Cloud WorkBench (Tutorial) 16
 - 2.2.2 Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research 17
 - 2.2.3 Cloud Computing: Comparison and Analysis of Cloud Service Providers-AWs, Microsoft and Google 18
 - 2.3 Huldra 18
 - 2.3.1 Huldra front-end 21
 - 2.3.2 Huldra cloud 23
 - 2.4 Proposed cloud solutions 24
 - 2.4.1 OpenStack 25
 - 2.4.2 Render 26
 - 2.4.3 Netlify 27
 - 2.4.4 Railway 28

2.4.5 Vercel	28
3. Methodology and Implementation.....	30
3.1 Deployment requirements.....	30
3.2 Metrics.....	32
3.3 OpenStack	33
3.4 Render	36
3.5 Netlify	38
3.6 Railway	40
3.7 Vercel.....	43
4. Experiments and Results.....	44
4.1 OpenStack	44
4.2 Render	48
4.3 Netlify	50
4.4 Railway	51
4.5 Vercel.....	52
5. Discussion.....	55
5.1 Comparison of results	55
5.2 Other contributions.....	58
5.3 Challenges and future work	58
6. Conclusion.....	60
6.1 Thesis summary.....	60
6.2 Main Contributions	60
6.3 Revisiting the problem statement.....	61
References	64

List of Figures

Figure 2.1 Workflow of general IaaS benchmark	17
Figure 2.2 React – frontend	21
Figure 2.3 React element in the Huldra framework code	22
Figure 2.4 React components	23
Figure 2.5 Docker overview (Gaikwad, 2023, https://chaitannya.hashnode.dev/containerization-of-applications-docker-overview)	26
Figure 3.1 Optimal Time To First Byte (TTFB) measurements	33
Figure 3.2 Instance types in the OpenStack dashboard	35
Figure 3.3 Dockerfile content	36
Figure 3.4 Render instance pricing (Render, n.d., https://render.com/pricing#compute).....	37
Figure 3.5 Rewrite rule.....	38
Figure 3.6 Netlify redirect rule.....	40
Figure 3.7 Dockerfile detection in Railway	42
Figure 3.8 Dockerfile with environment variables.....	43
Figure 4.1 Initial deploy time	45
Figure 4.2 Redeploy time	45
Figure 4.3 OpenStack CPU test results	46
Figure 4.4 Google Cloud CPU test results	47
Figure 4.5 OpenStack memory test results.....	47
Figure 4.6 Google Cloud memory test results	48
Figure 4.7 OpenStack storage speed test results	48
Figure 4.8 Google Cloud storage speed test results	48
Figure 4.9 First time Render deployment.....	49
Figure 4.10 Redeployment of Render site	49
Figure 4.11 Render ping test.....	49
Figure 4.12 Netlify initial deploy/build time message.....	50
Figure 4.13 Netlify redeploy time	50
Figure 4.14 Successful Railway deployment.....	51
Figure 4.15 Railway network latency test.....	52
Figure 4.16 Vercel deployment summary.....	53
Figure 4.17 Vercel redeploy summary.....	53
Figure 4.18 Vercel ping test	53
Figure 5.1 Overview of free usage in Render	57
Figure 5.2 Monitoring instances in UpTimeRobot.....	58
Figure 5.3 Recorded monitoring events	58

List of Tables

Table 3.1 Render deployment plans	36
Table 4.1 Render metrics analysis with GTmetrix	50
Table 4.2 Netlify metrics analysis with GTmetrix	51
Table 4.3 Railway metrics analysis with GTmetrix.....	52
Table 4.4 Vercel metrics analysis with GTmetrix.....	54
Table 5.1 Comparison between the cloud alternatives.....	55
Table 5.2 Comparison of CSP features.....	56

1. Introduction

1.1 Background and motivation

Cloud computing has become a popular approach for deploying and hosting websites and web applications (Patel, 2021). There are numerous cloud solutions available for organizations to host and deploy their websites and apps, but there is an ongoing discussion of what the best practice is (Saraswat & Tripathi, 2020). With so many options to choose from, it can be a challenging task to decide which direction to choose, as there can be different consequences and outcomes from the chosen path (Saraswat & Tripathi, 2020). Numerous factors must be considered when determining the appropriate solution. Price, resources, features, and performance are some of the main aspects of a cloud deployment and hosting solution which can impact the consequences and outcome of the results.

Virtualization is a key concept when working with the cloud (Rashid & Chaturvedi, 2019). This is the process of creating virtualized resources from physical resources. A common practice is to generate multiple virtual machines (VMs) from a single server (Masdari et al., 2020). This provides the possibility to utilize a server for multiple purposes simultaneously, thereby increasing efficiency and reducing costs. For web hosting purposes, a CSP can be used. CSPs are services that typically utilizes virtualization to provide cloud-based resources and features directly to end users (Rashid & Chaturvedi, 2019). This solution can be referred to as an off-premise solution, as they offer services that are hosted and managed outside of an organization's own physical data center or IT infrastructure. Different pricing plans are usually offered in CSPs and usually determine the types of resources and features available for the end user (Saraswat & Tripathi, 2020). Deciding a pricing plan can be a time-consuming and cumbersome task, as the choice typically depends on the use case (Saraswat & Tripathi, 2020). The pricing plans can range from limited free, hobby plans to all-inclusive paid professional and enterprise plans. Trials of plans and features are also often offered. Pay-as-you-go is a pricing model many CSPs utilizes where customers are only charged based on their actual usage of the service (Duan et al., 2020). Performance and uptime are critical factors to consider when choosing a cloud platform (Dhingra & Rai, 2020). If a website or application fails to perform or

encounters frequent downtime, it can impact the users experience and in worst cases be critical for the survival of the site. To avoid these types of issues, a thorough review and analysis of the different CSPs and plans available should be conducted. A feature many CSPs offer is the ability for automatic scaling of resources, which can help address these issues (Singh et al., 2019).

The deployment and hosting of a website or app can be conducted in the cloud using a variety of methods (Lăcătușu et al., 2022). Deciding upon a method can among other factors depend on the technical skill of the person performing the deployment, and the configurations needed for the application. A simple alternative many CSPs offer today is the possibility to automatically deploy GitHub repositories (Render Docs, n.d.). Most of the commands and configurations executed during an automatic deployment is performed in the backend of the CSP, reducing the need for human input. With this alternative, getting a site live is a straightforward process that involves a few simple steps which can be achieved by users with little to no technical expertise. A more advanced method of deploying is using a manual Dockerized approach (Shah & Dubaria, 2019) by executing commands in a command line interface (CLI). Compared to an automatic deployment, this approach provides the possibility for more configuration and tweaking of the deployment but requires a higher skill level of the person conducting it.

Huldra is a framework for collecting crowdsourced feedback on multimedia assets (Hammou et al., 2022). The framework is written in ReactJS and is an ongoing project being developed at SimulaMet in Oslo, Norway. The framework aims to provide researchers with a new customizable way of collecting crowdsourced feedback with the use of online surveys. It was designed to overcome challenges related to gathering online survey participants and challenges related to already existing online survey tools (Hammou et al., 2022). Huldra currently provides guidelines on how to deploy and host a survey with Heroku (Simula, n.d.), which is a CSP, but are interested in exploring similar solutions which can work as an alternative for researchers that want to deploy and host Huldra surveys in the cloud.

1.2 Problem statement

As mentioned above, there is an ongoing discussion of the optimal approach for a cloud web hosting solution and companies face several challenges when selecting an alternative. To decide upon a solution, factors such as resources, pricing, and performance can be challenging to research. As Huldra is looking for an alternative to Heroku, it will be used as a test project to address these challenges. The thesis will present and test several solutions for deploying and hosting a website or application in the cloud and evaluate the differences in performance. To gain quantitative and measurable results, each deployment solution will be benchmarked and analyzed. The process will include gathering metrics concerning time to deploy, price/resource, and performance. Four different public CSPs will be included in the study. These will be tested using the provided user interface (UI) for automatic deployments of GitHub projects. One manual deployment option will also be included as a proposed solution. This solution will utilize a Dockerized deployment, and the steps for deploying will be conducted manually with terminal commands. To gain insight to the differences between the included solutions, the data findings of each solution will be analyzed and compared. The study aims to answer two main research questions, which are:

Research question 1: How can customer benchmark and analyze cloud computing resources and performance?

Research question 2: Based on the results obtained from the analysis and benchmarking presented, which of the selected CSPs and deployment methods are suitable for a use case like Huldra?

The objective for the first research question is to research different solutions for benchmarking and analyzing the performance of CSPs and deployment methods. To reach this goal, a benchmark of system resources will be conducted, along with other methods of performance data gathering for analyzation purposes. The second research question will investigate which of the cloud alternatives can potentially be the best fit for a use case like Huldra. To reach this objective, a comparison of metrics mentioned above will be conducted to examine which

metrics include outliers. If one solution has the most positive outliers, this will be deemed the most suitable solution.

1.3 Scope and limitations

The work conducted in this thesis has a relative short time span which results in some limits being set. As researching and testing different cloud services and deployment approaches can be time-consuming, the number of alternatives being included was set to 5. This number was decided upon in a testing phase where deployments in different CSPs was conducted, as well as the testing phase for the manual approach. Limiting the alternatives and methods ensured that there will be adequate time to execute a sufficient analysis for each alternative. The 5 alternatives are:

- OpenStack (manual)
- Render
- Railway
- Netlify
- Vercel

Another limitation of the research was using paid plans offered at cloud service providers. Testing out paid plans and features would be a costly undertaking and out of the scope for this study. Therefore, the cloud deployments were limited to free plans or trials. Benchmarking VM resources at CSPs using free plans also has limitations. To conduct a technical benchmark of the system resources of a VM, access to the operating system (OS) is normally needed. Such a benchmark should also be conducted on several similar instance types from different cloud services to be able to compare the results. With free plans it is normally not possible to access a VMs OS where benchmarking commands can be executed, so metrics must be gathered with other methods. This led to the manual deployment being mainly focused on system resource benchmarking, and the automatic deployments mainly focusing on metrics related to server response time and network latency.

1.4 Research methods

A quantitative and experimental approach is used as a research method to obtain objective results. This includes a benchmarking approach for evaluating system resources of a VM instance, to obtain measurable data. The approach includes using the sysbench tool.

Benchmarking is a method that can be used in a variety of ways (Bhutta & Huq, 1999). An article presented by Aslanpour et al. (2020) presents an overview of performance evaluation metrics that can be utilized when benchmarking cloud aspects. In this thesis, the benchmarking of the resources uses a straightforward method of executing commands with sysbench and obtaining quantitative output. The data findings of the benchmarking results are further analyzed and compared for two different VMs.

Deploy times for the automatic GitHub deployments are obtained by analyzing build and deploy logs to gather measurable metrics. For the manual deployment, Docker commands are executed to obtain the deploy times. Further data is collected by analyzing response time, network latency, and stability of the automatically deployed cloud hosted sites. To collect this data, performance evaluation tools are utilized, including UptimeRobot, GTmetrix, and ping. The results from analyzing and benchmarking each cloud solution and method will be compared to gain insight to differences and outliers, and an alternative deemed most preferable for a use case like Huldra will be discussed. Literature research in the form of examining CSP documentation will also be used as a method to analyze the solutions.

1.5 Ethical considerations

The benchmarking of system resources in this thesis includes accessing systems operated and owned by a third party. In this context, fair use is important to make sure that the cloud systems are not unfairly impacted by high resource usage. It is therefore important to research and follow guidelines and usage limits properly. Cloud computing can also have an environmental impact, which should be considered when exposing a system for excessive usage. To achieve the desired insights from a benchmark, the benchmarking should be limited to essential tasks for this study to avoid unnecessary usage of resources.

1.6 Main contributions

The main contribution of this thesis is providing approaches for analyzing and benchmarking cloud service providers and cloud deployment methods, to further assist in the selecting of a cloud service and deployment method. To research these concepts, analysis and benchmarking have been performed for cloud service providers to obtain results of deploy times, price, resources, and performance. Two cloud deployments have been presented and utilized in this thesis, in the form of manual and automatic. The automatic deployment involves the use of GitHub integration at CSPs to automatically build and deploy a project in the cloud. A VM launched in OpenStack has been utilized to present a manual Dockerized deployment. To test the cloud service providers and deployment methods, a React based survey framework has been utilized as a use case. The results obtained are metric based and are collected from the cloud services and the deployment process. To give an insight to the differences and evaluate if one solution has outperformed the others, the results are further compared. The conclusion indicates that one solution stood out as the most suitable approach. The findings and methods in this thesis can be useful in further research and for software developers who are in the process of selecting a cloud solution for their projects.

1.7 Thesis outline

The background chapter presents relevant background information needed for a better understanding of the subject of the thesis. Cloud computing is the initial focus of this chapter, with information about cloud concepts, features, and services. The use case is further presented, including background information about Huldra and the chosen solutions for the problem statement. Chapter 3 involves presenting the approaches and preparation for experiments with the solution, which is presented in the next chapter, Experiments and Results. This chapter presents the benchmarking and analysis experiments conducted and the results obtained. The discussion chapter includes comparison of the results from the different cloud service provider and cloud deployment solutions. It also presents other contributions provided from the research and discusses challenges from the work and potential future work. The conclusion chapter begins with a summary of the thesis. The main contributions are further presented, with an overview of the steps done to be able to present benchmarking of resources

and cloud solutions. Finally, the problem statement is revisited. This section examines how the problem statement has been addressed and the research question has been answered.

2. Background

In this chapter, relevant background information will be presented. Cloud computing basics, services, and features will first be introduced. Next, a literature review of similar research done on the topic will be presented. Further, Huldra will be discussed including the front-end, back-end, and the cloud aspects of the project, and finally, the selected cloud platforms and methods will be introduced.

2.1 Cloud basics

The cloud is a broad name for services, applications, and resources that are hosted and delivered over the internet (Mahmood, 2011). Traditionally, websites and apps have been hosted using an on-premises solution. With an on-premises solution, companies typically host sites and apps directly on their own servers on-site (Fisher, 2018). On-premises solutions is still used today in the age of cloud. Even so, many companies are shifting towards cloud-based solutions due to the flexibility, scalability, and cost-effectiveness they offer (M. Saraswat & Tripathi, 2020). With the cloud solution, businesses use a third-party cloud service for deploying their web services or apps in the cloud. With recent advances in technology, this is a solution that has seen a high rise in usage (Hassan et al., 2022) and is today a popular way of hosting webservices. On the market today there are various types of cloud services available for a variety of use cases. These services offer virtualized resources for other companies to host and deploy their apps and services (Rashid & Chaturvedi, 2019).

2.1.1 CSPs

Virtualization is a key concept when working with the cloud (Rashid & Chaturvedi, 2019). A CSP will normally have physical servers which they use to create virtual resources. These virtual resources use a portion of the physical servers' resources to create new, standalone systems called virtual machines (VMs) or instances (Masdari et al., 2020). A VM operate similar to and usually contain the same components as a personal computer. Just like a personal computer, a VM has its own operating system, memory, storage, and CPU. The resource specifications of the VMs offered by CSPs can greatly vary, often based on the CSP and what kind of pricing plan they offer (Zhou et al., 2019). CSPs typically offer a range of different plans, both free and paid. Free

plans or trials are often provided so companies can explore features and test out deployments before committing to the service. These plans are normally restricted in some form or another. These restrictions can include time limits, bandwidth limits, deployment limits, or include less features.

The more resources a customer requires, the more costly the service will be. The plans normally coincide with how much resources are allocated from the services infrastructure to the customer. The plans can be based on how many VMs are provided and how much resources each VM has. To choose a plan the customer should research what kind of resources their web service or app require to be both deployed and hosted with a CSP. The requirements behind the choice can be based on the size of the projects, how many estimated daily users it will generate, the storage needed, scaling options, collaboration options, bandwidth and build minutes. Before a plan is chosen, a customer should research which CSP will be best suitable for their use case, which will be discussed further in the methodology chapter.

2.1.1.2 SaaS, IaaS, PaaS

Three main types of cloud computing service models are often offered at third party CSPs. National Institute of Standards and Technology defines these three types as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) (Mell & Grance, 2011). The smallest model, SaaS, consists of an application software that is hosted on a company's servers and are available to use for customers remotely over the internet (Rani & Ranjan, 2014). This eliminates the need for customers to install the software on their own computers or devices. Well-known SaaS examples include Microsoft 365, Trello and Netflix. PaaS is a more complex service that can offer a platform for development, where the service offers access to tools to use in different layers of development and production (Rani & Ranjan, 2014). This allows customers to focus on their software development and deployment without worrying about the underlying infrastructure. The last service, IaaS, offers virtual infrastructure resources like storage, servers, and networking where customers can build their own computing infrastructure in the cloud (Rani & Ranjan, 2014). PaaS and IaaS have similar

features, and services that can be defined as PaaS often fall under the IaaS category as well. Two examples of such services are Amazon Web Services (AWS) and Microsoft Azure.

2.1.3 Benchmarking resources

Benchmarking the performance of system resources like CPU, memory, storage, and network on a VM can be performed by accessing the OS of the instance in a terminal. In the terminal, a tool like Sysbench can be used. Sysbench is a tool that offers a simple and efficient platform to obtain information about a system or testbed, and to evaluate its performance under a heavy workload (Gantikow et al., 2020). The tool is open-source and includes benchmarks to test various factors of resource performance and provide details and reports of findings. The result reports can be used to compare different types of instances performance. To benchmark network performance, network latency can be tested using a basic internet program like ping. Network latency shows the time that data takes to transfer across the network and determines the delay in network communication (Amazon Web Services, n.d.). A high latency means the network has a longer delay, while a low latency means a shorter delay. Websites, services, and apps want to reach as low latency as possible, as this means less delay and lag for end users. Latency is usually measured in milliseconds (ms) and Fitzgerald (2022) proposes that less than 100 ms is generally acceptable, with the optimal range being between 20 and 40 ms. Even so, the acceptable range can be dependent on the application, use case, or user preferences.

2.2 Related work

This section will present three articles of work relevant to this thesis.

2.2.1 Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with Cloud WorkBench (Tutorial)

An article written by Joel Scheuner and Philipp Leitner titled “Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with Cloud WorkBench (Tutorial)” presents a tutorial for a solution for benchmarking IaaS platforms (Scheuner & Leitner, 2019). The proposed solution in the study is a Web-based benchmarking tool called Cloud Workbench (CWB). This tool is defined at the GitHub repository as “a web-based framework that is grounded on the notion of Infrastructure-as-Code (IaC) to foster simple definition, execution, and repetition of

benchmarks over a wide array of cloud providers and configurations” (Sealuzh, n.d.). The article first describes how a benchmark of IaaS clouds are executed in general. Figure 2.1 displays the general workflow of such a benchmark presented in the article. The first step is for the benchmark manager to acquire the instances to be used in the benchmarking via the application programming interface (API) and configure them. Within the instance, the execution of a benchmark is started and returns results in form of metrics. Lastly, the instances are destroyed when the benchmark is complete. As one single lifecycle of such a procedure can result in unpredictable and uncertain results, there are usually several lifecycles executed to create a sample size which can be used to determine the actual performance. This form of benchmarking can be a tedious process, and the article further demonstrate how to modify a general benchmark and execute it in a public IaaS cloud using the CWB web interface.

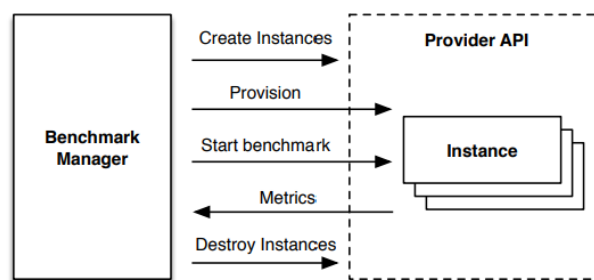


Figure 2.1 Workflow of general IaaS benchmark

2.2.2 Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research

This article written by Aslanpour et al. (2020) has relevant research that touches on the difficulties of benchmarking factors in cloud computing that can be useful for the research presented in this study. A taxonomy of the various real-world metrics to evaluate the performance of cloud computing is presented in the paper. The paper is further described as a comprehensive benchmark study for assisting developers and researchers to evaluate performance under realistic metrics and standards to ensure they will reach their objectives in a production environment. The list below displays the applicable cloud metrics defined and explained in the article.

- Throughput
- Network congestion/Traffic control
- Time to adaptation/Scaleability
- Cost/Profit
- Resource Utilization (Container, VM, host or data center)
- Resource Load (Container, VM, host or data center)
- Response time (second)
- Delay/latency time

According to the paper, the metrics presented have a various degrees of importance depending on the cloud model. The taxonomy can be further used for the research conducted in the analysis of cloud alternatives for deploying and hosting a use case like a Huldra survey.

2.2.3 Cloud Computing: Comparison and Analysis of Cloud Service Providers-AWs, Microsoft and Google

This study provides summarization and comparison of the features of Amazon Web Services (AWS), Microsoft Azure & Google Cloud Platform (GCP) to provide help to users and organizations for selecting the suitable features which will fulfill the long term requirements of the users (Saraswat & Tripathi, 2020). The article includes similar subjects as this thesis and presents relevant and useful information. While this thesis focuses on smaller services, the article presented in this section focuses on the three of the biggest competitors in the cloud computing industry.

2.3 Huldra

This thesis uses the repository of the Huldra framework project conducted at SimulaMet as a use case and will, as mentioned, research possible solutions for cloud-based deployment and hosting. The Huldra framework is a web-based survey framework which is designed to address challenges associated with user studies involving crowdsourced feedback collection (Hammou et al., 2022). The challenges stem from existing tools being difficult to customize, complicated, or designed for a specific use case. The project is open source, and the source code is freely available in a GitHub repository (Simula, n.d.) for developers and researchers to use the survey

framework for their own purposes and use cases. The goal of the project is to make an online survey more customizable for the creators. It can be used as an online survey tool to offer researchers feedback on multimedia assets which could relate to machine learning, audio, image and video quality assessment and quality of experience. Similar frameworks and services to Huldra exists. Some of the biggest competitors are Google Forms and Survey Monkey. These competitors offer many of the same features as Huldra, but they are services and are not open source. This potentially makes them a less useful tool for researchers that want to customize every aspect of their survey. With Huldra being open source, developers and researchers can use the whole code for their own use cases and configurations.

In recent times, the amount of online web sites and apps creating and delivering multimedia content have been rising (Hammou et al., 2022). There is a lot of content being produced in all kinds of topics and fields, and it can be difficult for content creators to understand and research what kind of content is relevant, in demand and that will generate regular user traffic. A tool like Huldra can therefore be useful for creators to get crowdsourced feedback from different sources and people on what the users would more specifically like to consume, as well as the usability and fidelity of the online systems generating and providing the content (Hammou et al., 2022). A paper written by Hammou et al. (2022) based on the Huldra framework elaborate on why the framework was created and the challenges associated with study participants. The following challenges and reasons for developing Huldra are listed in the article.

- Experts are a sparse resource.
- Complexity reduces participation.
- Increasing participant diversity requires universal accessibility.
- Human feedback is subject to bias.
- Multi-disciplinary competence is necessary.
- Commercial tools are not easy to customize or use.
- Academic institutions might incur additional costs.
- Privacy protection requires overall control.
- Continuous development is challenging.

With content creation, grabbing the consumers attention is an important part of the content being consumed. A video service like YouTube is a relevant example. Due to the vast content available on YouTube, users can tend to easily scroll past and ignore certain videos if they fail to capture their attention. The first impression of a video on YouTube is the title and the thumbnail. Here, Huldra provides assets where a researcher can present different thumbnails in correlation with the video and receive feedback on which thumbnail the participant prefers and in turn would also be more likely to click at. In this particular use case, the results from the survey can be further used by content creators to analyze and predict how a video should be presented.

Especially in high risk and high impact fields such as medicine, law, and industry, the testing of how certain multimedia content is perceived by end-users and domain experts can provide crucial information (Hammou et al., 2022). An example of the use of Huldra in such a use case is the possibility to evaluate the perceptibility of medical images in machine learning models. A paper by Hicks et al. (2022) conducting research on an artificial intelligence model used to explain the prediction of computer-vision-based deep learning models, uses Huldra to ask medical doctors their opinion on two different categories of explanation methods. The study was conducted using automatic detection of colon polyps as a use-case, where a deep learning-based model is tasked with classifying images as either containing a polyp or not. Huldra is used to gather important knowledge about which of the two explanations gives the clearest answer in what the picture represents and what the important part of the picture is.

As Huldra is a framework, it can be useful to understand what a framework entails. A framework is defined by Christensson (2013) as a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform. A framework takes care of low-level functions in a project and lets developers focus on high level tasks. In this case the framework is a survey-based project and the goal here is that researchers can customize their surveys on a high-level and not worry about the low-level aspects of the questions presented. More specifically for this case, the front-end and UI should be easily configurable without the need for tinkering with the back end. Even so, the

framework is open source, and the survey creators also have the possibility to deploy the project locally and edit and configure the back-end as well.

2.3.1 Huldra front-end

This section will briefly present the front-end code of Huldra and the basics behind ReactJS. The code behind Huldra, mainly ReactJS, is the foundation that will be used for the cloud deployments, and it can be useful to understand what it contains. Currently Huldra provides different types of question case layouts, where the elements and the user interface (UI) can be customized by the survey creators. Multimedia assets can be added and include images, videos, and audio, as well as hybrid cases. The front-end of the project is written in React, while the back-end utilizes NodeJS.

React is a popular frontend JavaScript library for web developers to build user interfaces. Generally, React is thought of as the view layer in an application, and changes what the user sees (Boduch & Derks, 2020). Figure 2.2 shows where React fits in front-end code. Data is first generated from some application logic. This data is further passed to a React Component that performs the job of passing HTML into the page. React components let developers split elements of an application into smaller isolated sections. These sections can then be reused in different parts of the code and project.

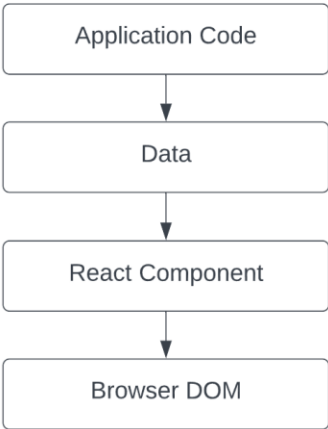


Figure 2.2 React – frontend

As React is the main technology used in the project, the JavaScript code will be examined further. The project uses a normal file structure for a React app project, with 'src' folder containing the main source code consisting of all JavaScript and CSS files.

The 'Public' folder contains the index.html file, which is the most important file in the folder as it is needed to successfully build and deploy the app in the cloud. Inside the 'src' folder, the subfolders 'major-components' and 'minor-components' are located. These folders contain .jsx files. JSX is a syntax extension to JavaScript and is in this case the technology that renders HyperText Markup Language (HTML), which is the standard markup language used for web pages. JSX produces React "elements", and React's documentation recommends using JSX with React to describe what the UI should look like (React, n.d.). React elements describe what will be seen on the screen or the front-end and allows for HTML to be written in React. The code in Figure 2.3 shows a React element for an input checkbox in the Huldra framework code and shows the HTML written in the element. The "const" in the code is the element, and the return statement returns HTML code with some JavaScript code. The last line in the figure exports the input checkbox as a component.

```
const InputTickbox = ({ id, Label, onChange, optional, type = "checkbox" }) => {
  return (
    <FormGroup check>
      <Label htmlFor={id} check>
        <Input
          id={id}
          onChange={(e) => onChange(e.currentTarget.checked)}
          type={type}
        />
        {Label} {!optional && <Asterisk />}
      </Label>
    </FormGroup>
  );
};

export default InputTickbox;
```

Figure 2.3 React element in the Huldra framework code

The code displayed in figure 2.3 is one of many minor components included in the code. Components are similar to and can be confused with elements, as a React element is what gets returned from a component. Figure 2.4 shows the component folders for the project, with minor and major components. The minor components cover smaller elements in the code, like

a generic button, which are independent and can be reused. The major components include the bigger elements the framework consists of, which is mainly the layout of the different question cases.

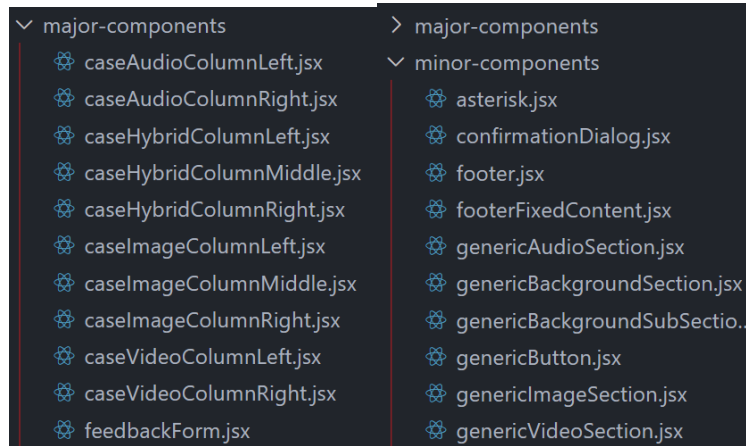


Figure 2.4 React components

2.3.2 Huldra cloud

Cloud features utilized for Huldra include storage, deployment, and hosting. For cloud storage Huldra uses Firebase. Firebase is a PaaS backed by Google and covers multiple areas and features of developing a backend for a web application in the cloud (Stevenson, 2018). The primary use of Firebase for Huldra is currently for storage, where the created survey pages/cases are stored. To set this up, survey creators need to set up the correct folder structure which is explained on the GitHub page for the project (Simula, n.d.). When this is taken care of, the multimedia assets corresponding to their desired cases needs to be prepared and uploaded. When the survey is rendered, it fetches the data from the storage and presents it on the page. The survey answers and results are also stored in Firebase and as raw data format. The reason for this is that the researcher and survey creators can decide for themselves how they want to analyze the results. With the raw data they have many possibilities to create visual representation of the result data in forms of graphs, diagrams, and other forms of analytic tools. One way to analyze the answer results is to use a Python notebook, which users of the framework have previously done.

NodeJS is as mentioned earlier the backend framework used for Huldra. This framework provides the ability to write JavaScript programs for the server-side (Nielsen et al., 2019). Processes that are executed on the web server, where websites and apps like Huldra is hosted, are referred to as server-side (Sigdestad, 2023). NodeJS must be installed when setting up the development environment for working on the framework. To connect the framework to the Firebase database, which is where the elements of the framework are stored, environment variables are needed. Environment variables in React are variables that are available through a global process.env Object, and that global Object is provided by the environment through NodeJS (Pakvis, 2020). The environment variable parameters should not be stored in configuration files for security reasons, as they contain sensitive data and must be secret. When deploying the app these variables must be added to successfully connect to the Firebase storage. There are various ways of adding these variables depending on which type of deployment service will be used. How to add the environment variables in the different cloud platforms tested will be discussed in chapter 3. When working on the project locally, the environment variables can be added to the Huldra project with .env file.

The cloud deployment and hosting of a survey is conducted by the researchers themselves, but Simula provides some guidelines on how to do it with Heroku (Simula, n.d.). Heroku is a PaaS that offers users the ability to build, run and operate applications in the cloud (Heroku, n.d.). To deploy a survey using Heroku, researchers are instructed to use Heroku's GitHub integration for automatic deployments (Heroku Docs, 2022). This is the main deployment and hosting service tested for the Huldra project. Heroku used to offer a free plan, but as of November 28, 2022 this plan has been removed (Heroku, n.d.).

2.4 Proposed cloud solutions

This section will present the cloud deployment and hosting solutions chosen for the study and their relevant technologies and features. The implementation and actual deployments will be conducted in chapter 3 and 4.

2.4.1 OpenStack

This solution consists of a manual deployment using OpenStack and the ALTO Cloud at OsloMet. According to OsloMets own website, ALTO cloud is currently the largest OpenStack deployment in production in Norwegian higher education with 1024 cores and 4TB of RAM (OsloMet, n.d.). OpenStack is an open-source cloud computing platform that provides assets for building and managing cloud computing platforms, including public and private clouds, created by a group of developers and cloud computing experts who work together globally (Kumar et al., 2014). For this deployment solution, a VM has been created in the ALTO cloud using OpenStack. A Huldra survey will be further deployed on this VM using Docker.

Docker is a technology for managing and building containers that has improved agility and efficiency in cloud-based software development and production. Docker was first introduced in 2013 and is currently one of the most popular containerization solutions (Lin et al., 2020).

When a webservice or app is deployed to a container, the container bundles everything needed for an app to run in an isolated environment (J. Shah et al., 2019). This includes code, dependencies, libraries, packages, and files. This isolated environment has its own operating system and gets a small amount of resources from the VM. Using Docker containers to share an app eliminates the need for recipients to install any dependencies or worry about whether the app will function correctly on their system.

To create a Docker container, a Docker image is used. A Docker image is a template that contains the needed instructions for creating a container (Lin et al., 2020). To create a Docker image, a Dockerfile is written which specifies the necessary instructions to build and run the image. Each layer in an image is created based on each instruction in the Dockerfile. When an instruction is changed, rebuilding the image will only modify the layer dependent on this instruction. This is according to Dockers own documentation one of the reasons for why images are so lightweight, small, and fast compared to other virtualization technologies (Docker, n.d.). Figure 2.5 shows an overview of the Docker technology, with client, server, image, and container. The figure also visualizes how an image can be distributed with Docker Registry. Dockers documentation explains the Registry as a stateless, highly scalable server-side

application that stores and lets users distribute Docker images (Docker, n.d.). Many CSPs offer services to help developers build, deploy, and run applications containerized with Docker (IBM, n.d.). This provides customers with a standardized and isolated environment for running applications on their platform. The technology is often built in in their virtualized environment and utilized when deploying a website or app.

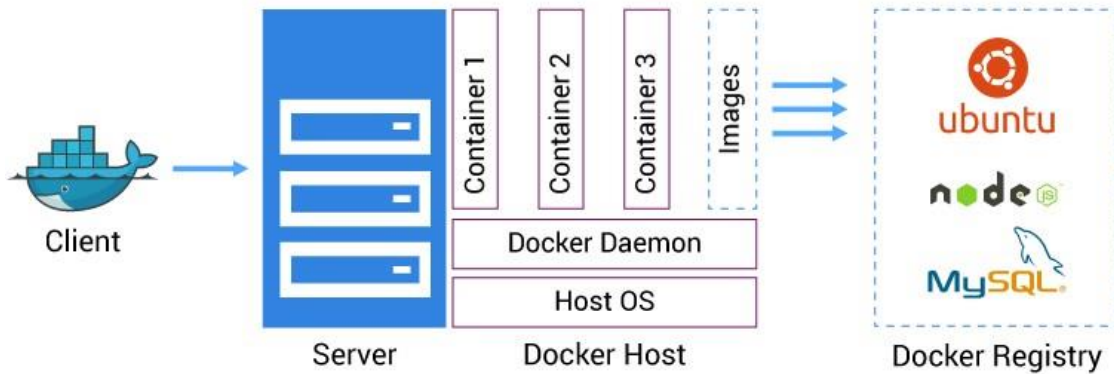


Figure 2.5 Docker overview (Gaikwad, 2023, <https://chaitannyya.hashnode.dev/containerization-of-applications-docker-overview>)

Contrary to a an automatic CSP deployment which provides a user interface (UI) for the deployment, this deployment will be conducted using commands in an Ubuntu command line. This requires some technical knowledge from the researcher that will deploy it. To log in to a VM, Secure Shell (SSH) will be used. SSH is a software package that enables secure system administration and file transfers over insecure networks, which is used in most data centers and large enterprises around the world (Ylonen, n.d.).

2.4.2 Render

Render is a PaaS and a direct competitor to Heroku. It offers a cloud platform where users can deploy sites and web apps in a few steps without much experience. Renders was launched in 2019 and according to their website they offer the best of both worlds when it comes to cloud infrastructure (Render, n.d.). Their website state that instead of developers having to be forced to pick between expense yet inflexible platforms that fail to scale or complex large clouds that impose steep learning curves and massive operations teams, they offer ease of use coupled

with immense power and scalability to power everything from a simple HTML page to complex applications with hundreds of microservices.

With Renders unified cloud, apps and websites can be built and run with free Transport Layer Security (TLS) certificates, a global Content Delivery Network (CDN), Distributed Denial of Service (DDoS) protection, private networks, and auto deploys from Git (Render, n.d.). These features and concept will be further described. AWS describe that an SSL/TLS certificate serves as a digital identity card that secures network communications, verifies the identity of websites on the internet, and authenticates resources on private networks (Amazon Web Services, Inc., n.d.). A big portion of the Internet's content is distributed by CDNs by caching and serving the objects requested by users (Kirilin, 2019). Caching is the process of storing portions of data temporarily, so it can be retrieved faster when needed. DDoS is a potential security threat all online systems should prepare and actively defend against. The concept of this security threat is an attack that overwhelms a target, for example a server, with countless request (Cui et al., 2019). When the target receives more request than it can handle, it can create critical performance issues and potentially shut down the system. This can be critical for systems that needs to stay online at all times, and DDoS protection is an important requirement for an online system.

2.4.3 Netlify

Netlify is a PaaS and a direct competitor to Heroku. Netlify offer a free plan with no credit card required. The free plan includes a single-member plan for personal projects, prototypes, or getting started, and should be a sufficient alternative for deploying and hosting Huldra.

Netlify enforces a strict concept of atomic deploys (Netlify Docs, 2023). Atomic deployments make updates available only when they are complete and totally in place (Hawksworth, 2021). When new files and configurations are pushed, the atomic deployment triggers a completely new redeploy of the site, but only after all changes have been successfully uploaded and are ready. This is enforced to avoid the site being deployed in a partially updated state. Just like Render, Netlify utilizes CDN. Once all changes are done and ready, the new version of the site goes live on the CDN immediately. The deployment testing of a Huldra survey will be conducted

with Netlify's provided GitHub integration (Williamson & Lengstorf, 2016). When changes are made in the GitHub repository, an atomic deploy will be triggered.

When deploying a site, Netlify provides a UI with different settings that can be configured by the user. Another solution to the UI provided settings is to use the Netlify configuration file, which can be used for more fine tuning of the deployment and build. To configure settings with this file, named `netlify.toml`, it is normally added to the root folder of the project (Netlify Docs, 2023). Netlify's full platform can also be accessed in a terminal (Netlify Docs, 2023). Here, building, deployment, and configurations can be done with commands. This is provided so more technically experienced users can use their own workflow and configure with commands as they go.

2.4.4 Railway

Railway is a PaaS with similar features to Netlify and Render and offers a free plan for deploying a site. For deployments in Railway, GitHub integration is available, and Railway uses Docker for its building and deployment infrastructure (Railway, n.d.). During the build of a deployment, Railway will attempt to create a Docker image of the provided code that is deployable. If the Docker build is successful, Railway will try to deploy this image. Other notable features included in the Railway platform are Railways CLI, deployment rollbacks and autoscaling. Railways CLI can be used to connect a projects infrastructure and secrets from any terminal in the world (Railway, n.d.). Deployment rollbacks is a useful feature provided in Railway, as well as other CSPs, to avoid keeping a faulty site up. The feature provides an instant rollback for every deployment change made (Railway, n.d.). Autoscaling provides scaling of apps to meet user demand automatically based on load (Clark, n.d.), which is a feature provided in several CSPs. The deployment test in Railway will consist of creating a new project in the Railway dashboard and connecting and deploying the project from a GitHub repository containing the Huldra framework.

2.4.5 Vercel

Vercel is a PaaS and an end-to-end platform for developers (Vercel Docs, 2023). It offers an infrastructure for users to create, build, deploy and host projects. Like the other CSP included in

this study, Vercel offers a UI for deploying webservices and apps. They also provide a CLI for more technical advanced users. Vercel offer a free plan, as well as a 14 days free trial of the pro plan.

A new feature to cover for Vercel, and CSPs in general, is the possibility to preview deployments. This feature allows users to preview changes done to an app in a live deployment without merging those changes to the Git projects production branch (Vercel Docs, 2023). The purpose of previewing deployments is to test and experiment with new features in a live environment without it being available to the public. Members of the Vercel team assigned to the project can view the changes and give feedback via comments, which is another (optional) Vercel feature. Comments can be added directly over the relevant areas on the site. The previews can also be shared with external people who are not a member of the team.

3. Methodology and Implementation

This chapter will cover the methodology, preparation, approach, and implementation for the testing of the different cloud platforms and methods chosen for the benchmark, with experiments and results being presented in the next chapter. A quantitative approach will be used in the experiments as this study aims to obtain objective results from the benchmarking, with analysis of the data findings.

3.1 Deployment requirements

When deciding upon a cloud solution for deploying and hosting a Huldra survey, several factors and criteria should be considered. The first factor to clarify is if a solutions infrastructure supports the technologies used in the Huldra framework. This primarily means a solution must support building and deploying Node.js projects. The next step is to determine the appropriate pricing plan and resources required for the survey to reach desired performance and uptime. This will depend on the content of the survey and the expected number of participants. If the survey needs to access large amounts of data for the content from the external database, it could consume a significant amount of network bandwidth and computing resources on the CSP's end. The participants of a survey will be the user that generates traffic to the site. The more users there are, the more network bandwidth will be consumed. Security is another requirement to consider. Lomas (2023) lists identity management, access controls, authentication, and data management as important factors when considering the security and reliability of a cloud solution. It should also be decided if custom configurations are needed beyond a simple and straightforward GitHub repository deployment. If custom configurations are needed, a manual Dockerized deployment or using the CLI provided by several CSPs could be good alternatives.

The Huldra code to be used for the deployments and benchmarking is the current internal version, which is under development, as this is the most up to date version and deployments of the public version ran into issues. The code includes a template survey, with placeholder text, sample images and some instructions on how a survey can be customized. The repository to be used is a new empty private repository with a clone of the internal repository. This is so new

files can be uploaded without interfering with the internal repository which is used by several developers.

To answer the problem statement and research question, deployments will be conducted with each cloud solution presented in the background chapter. After the deployments are completed, analysis and benchmarking of each alternative will be performed. The idea is that analyzing and benchmarking each solution will result in data that can be used to compare the solutions and answer if one solution is clearly superior to others, or if the results from each analysis and benchmark are so much alike that it will not make a big difference choosing either. Literature review of the services' documentation will also be used to analyze pricing plans and resources for each solution. The solutions were decided based on the following initial requirements.

Requirements for all solutions:

- The possibility to build, deploy and host the Huldra survey template in the cloud.
- Adequate resources and sustainability for deploying and hosting the template survey throughout the duration of analysis and benchmarking.

Requirements for automatic deployment at CSPs:

- A GitHub integration for automatic deployment of GitHub repositories.
- The possibility of creating or generating a unique URL domain for the survey, as buying a domain is an unnecessary expense for the survey creators.
- A UI for managing and deploying the project.
- Continuous deployment. This feature is a concept that falls under agile software development and production and provides the possibility for redeployments of the project whenever a change is done to the GitHub repository. When the main branch of a Huldra survey repository is updated, the cloud service provider should redeploy the project with the same configuration and domain as the first-time deployment. This

ensures that the survey is always up to date and the latest version can be accessed by the public.

- A free plan or trial.

3.2 Metrics

The first factors to include in the evaluation of a solution is the available pricing plans and resources. The benchmark will further analyze the time it takes to deploy the project. The preparation time for a deployment will be estimated, as well as the actual build and deployment time. The time it takes to redeploy the site after a change to the GitHub repository will also be included. Simplified steps taken to prepare and launch a deployment is covered in each solutions subsection. As these steps are included in the preparation time, it and can be useful for users to get an insight to the process when choosing a solution.

When the site is up and running, the network latency will be measured. Other metrics that will be experimented with for the automatic deployments are related to user experience and how quickly the server that is hosting the repository responds. The first metric to consider here is First Contentful Paint (FCP). FCP is a metric that determines the time it takes from when the page starts loading to when any part of the content of the page is rendered on the screen (Walton, P., 2022). One factor that can affect the speed of FCP is how fast a server responds to requests. Analyzing this metric can be useful in comparing the response times of different cloud solutions. In the same vein, Largest Contentful Paint (LCP) can be included as a metric. This determines the time it takes to load the main content of the page (Walton & Pollard, 2023) and is also dependent on the speed of the server. FCP and LCP are dependent on Time To First Byte (TTFB), which sums up the time of each request phase and estimates the time it takes for a browser to receive the first byte of response after it made a request to the server (Wagner & Pollard, 2023). This will also be measured and the requirements for a good TTFB is shown in figure 3.1. FCP, LCP and TTFB will be analyzed using GTmetrix which is a tool designed for analyzing the performance of websites. GTmetrix implements Google Lighthouse, another web performance tool, for their report results. For the manual dockerized solution, a benchmark of

the VM's system resources will be performed using the sysbench tool for Ubuntu. For this benchmark, the CPU, memory, and Disk I/O will be tested.

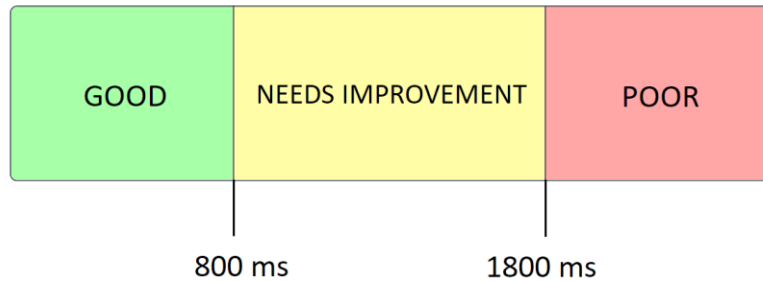


Figure 3.1 Optimal Time To First Byte (TTFB) measurements

To monitor the sites for performance analyzation purposes, UptimeRobot is utilized. UptimeRobot is according to their website the world's leading uptime monitoring service (UptimeRobot, n.d.). Each of the sites are added in the UptimeRobot dashboard with their URL. As the account used is free, the frequency of monitoring is set to a 5-minute interval. With a paid plan, the frequency of monitoring can be set to a 30 or 60-second interval. The findings from a week-long monitoring period will be discussed in the Experiment and Results chapter.

3.3 OpenStack

As previously mentioned, some technical understanding is needed when conducting a manual deployment compared to an automatic. This is a factor that should be taken into consideration when choosing a deployment method. The steps carried out to execute this deployment are described briefly. The reason for including the OpenStack deployment solution is that it can be more specifically configured and tweaked by having full manual control over the commands executed and the resources used. When using an automated deployment at a third-party CSP, these specific configurations are not always possible. As previously mentioned, the test will consist of deploying the framework using Docker on a VM that gets its resources from the ALTO Cloud.

The price of this deployment depends on what resources are available to the survey creator. If the survey creator owns his own server or has connections that can provide a server, there are no additional costs beyond server maintenance and keeping the server up and running, as

OpenStack is a free open-source platform. Without having access to a server, the creator of a survey can use a third-party service like OpenMetal to rent a server, but this is very costly and not relevant for this use case. A more relevant solution is using a third-party cloud service provider for the manual OpenStack deployment.

The deployment process involves creating and building a Docker image and running a container to host the application code from the repository. This deployment is conducted using cloud resources provided by OsloMet. These resources include an account with a set number of instances that can be configured with various specifications. To prepare a deployment, a VM is first launched in the OpenStack dashboard. Here there are several instance types available, with different resources included for each. Figure 3.2 shows the different types of instances available for this deployment. The smallest available is m1.micro with 1 VCPU, 64 MB of RAM and 0 GB total disk, and the biggest available is m1.xlarge with 8 VCPU, 15 GB of RAM and 160 GB total disk. For the deployment a medium instance is used, which includes two vCPUs, 4GB of ram and 40GB of hard disk. This should be sufficient to deploy and host the survey template, but smaller instances could work as well. The container where the site will be hosted has by default no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker does however provide ways to control how much memory, or CPU a container can use, (Docker, n.d.), but this test will use the default configuration.

NAME	VCPUS	RAM ^	TOTAL DISK	ROOT DISK	EPHEMERAL DISK	PUBLIC	
> m1.micro	1	64 MB	0 GB	0 GB	0 GB	Yes	+
> m1.tiny	1	512 MB	1 GB	1 GB	0 GB	Yes	+
> m1.512MB4GB	1	512 MB	4 GB	4 GB	0 GB	Yes	+
> m1.512MB	1	512 MB	3 GB	3 GB	0 GB	Yes	+
> m1.1GB	1	1 GB	5 GB	5 GB	0 GB	Yes	+
> m1.small	1	2 GB	20 GB	20 GB	0 GB	Yes	+
> m1.medium	2	4 GB	40 GB	40 GB	0 GB	Yes	+
> m1.large	4	8 GB	80 GB	80 GB	0 GB	Yes	+
> m1.xlarge	8	16 GB	160 GB	160 GB	0 GB	Yes	+

Figure 3.2 Instance types in the OpenStack dashboard

When launching the instance, it can be configured with different type of OSs. The OS of the instance is in this case set to Ubuntu 20.04 and an Ubuntu command line is used for all commands and configurations of the instance, which must be available on the PC of the person conducting the deployment. When the instance is up and running, two security rules must also be added in the OpenStack dashboard. One network port rule for port 22, which allows SSH access from the outside, must be added with an ingress network traffic direction. Another rule must be added to allow network traffic on the port which the survey site can be accessed on. Logging in to the instance requires adding a SSH key to the instance, which the user needs to provide from his own system/workstation. When the user is logged in, the Huldra framework is downloaded from GitHub to a user specified location on the instance. Docker is further installed on the VM. After it is installed, a Dockerfile can be created in the root folder of the project. As previously mentioned, a Dockerfile is a file that contain all the steps needed to customize and build a Docker image. The content of the Dockerfile is displayed in figure 3.3. The lines shown in the Dockerfile are equivalent to commands that will be executed to copy the framework and install all necessary dependencies for running the framework on a container. When deploying the site, environment variables also need to be added. This is achieved by creating a .env file in the root folder of the repository on the VM and adding the Firebase connection configurations.

The deployment has now been finalized and will be utilized for the experiments in the next chapter.

```

Dockerfile
1 FROM node as build
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 RUN npm run build
7
8 FROM nginx
9 COPY --from=build /app/build /usr/share/nginx/html
  
```

Figure 3.3 Dockerfile content

3.4 Render

Render offers four different plans which are designed based on the number of people who will use the account. The plans are Individual, Team, Organization, and Enterprise and the bandwidth and build minutes included in each plan is displayed in Table 3.1. All the plan includes the same service types but differ in bandwidth and build minutes. For this experiment the Individual plan will be used as one person will conduct the deployment. If a team of individuals are to be working on the deployment and hosting, the Team plan can be used.

	Individual	Team	Organization	Enterprise
Free Bandwidth	100 GB	500 GB	1 TB	Custom
Free Build Minutes	500/month	500 per user/month	500 per user/month	Custom

Table 3.1 Render deployment plans

For web services, Render offers seven different instance types from the lower tier, which is the free type, to the most extensive tier, Pro Ultra. They also offer a custom instance type. Each instance type includes a set of resources. Figure 3.4 shows the difference in price and resources between the instances available. The free instance includes 512 MB of RAM and 0.1 CPU, while Pro Ultra includes 32 GB of RAM and 8 CPU. With the custom instance customers can discuss the resources needed with Render. Web services on the free instance types has some

limitations (Render Docs, n.d.). They will automatically spin down after 15 minutes of inactivity, and spun up again if a request comes in. A request to the site after 15 minutes of inactivity has been reached will therefore cause a small delay to the response as the site needs to be spun up again, according to Render's own website. 750 hours of running time per month is included with the free instance type, and 100 GB of egress bandwidth for each service deployed. Free services are automatically suspended when the free usage limits are exceeded and can no longer serve traffic until the plan is upgraded or the monthly free usage is reset. The free usage is reset the 1st of every month. Render also has an option for deploying static sites. Static sites on Render are free, with no cost at all to you unless the site's monthly traffic exceeds 100 GB of bandwidth per month (Render Docs, n.d.). Disk size is not provided in Figure 3.4 as the default filesystem for services running on Render is ephemeral. This means they are temporary and application data isn't persisted across deploys and restarts (Render Docs, n.d.). If persistent storage is needed for a web service, this is also available at Render. Whether ephemeral or persistent storage is needed is something to consider when choosing a cloud solution for a website or application.


Instance Type	Pricing	RAM	CPU
Free	\$0/month with limits	512 MB	0.1
Starter	\$7/month	512 MB	0.5
Standard	\$25/month	2 GB	1
Pro	\$85/month	4 GB	2
Pro Plus	\$175/month	8 GB	4
Pro Max	\$225/month	16 GB	4
Pro Ultra	\$450/month	32 GB	8
Custom	Contact sales	Up to 512 GB	Up to 64

Figure 3.4 Render instance pricing (Render, n.d., <https://render.com/pricing#compute>)

The automatic GitHub deployment with Render will be conducted using the free Individual plan and the static site option. Creating a profile in Render is a straightforward process and does not

require adding a credit card when using the free plan. To start configuring the deployment, a new static site project is created in the Render dashboard. First the project needs to be connected to the relevant GitHub repository where the survey framework is located. After connecting to the relevant repository, environment variables can be added before deploying. These can also be added after the deployment.

During the deployment testing it was discovered that a redirect or rewrite rule is needed for the site. This is needed as static sites does not have a server-side component that can redirect HTTP requests. In the Render documents, the difference between redirects and rewrites is explained (Render Docs, n.d.). A redirect rule gives the browser instructions on how to change URLs and navigate to the rule's destination. A rewrite rule, on the other hand, does not change the original URL. It only serves the content of the rule destination at the original path. This makes it possible to display content from a different path or URL on any other path on a site, as the browser cannot tell that the content was served from a different path or URL. As Huldra is a React project and uses react-router and client-side routing, a rewrite rule is needed so it can handle all requests at index.html. Without this rule, the provided link for the site only works directly from Renders dashboard. When refreshing the link or accessing it from another browser tab, a '404 not found' message will be shown. Render offers a built-in option to add a rewrite or a redirect rule in the sites setting page. Here the following rule displayed in figure 3.5 is added. This rule will redirect the URL back to the homepage when the link is accessed or pasted into a browser tab. In the upcoming chapter, analysis will be carried out based on the completed deployment process.



Source	Destination	Action
/*	/index.html	Rewrite

Figure 3.5 Rewrite rule

3.5 Netlify

Netlify offers four different pricing plans which are Starter, Pro, Business, and Enterprise. Similar to Render, these plans are based on the size of the team or organization that will use the service. The starter plan is a single member plan for personal projects, prototypes and

getting started. This plans model is free to get started and then pay as you go. The Pro plan is intended for team collaboration for professional web projects, and the price starts at 19\$ per member a month. If advanced security and compliance for larger teams is needed, the business plan can be used and start at 99\$ a member per month. For the deployment test, the starter plan will be used. Per month, it includes 100 GB bandwidth and 300 build minutes (Netlify, n.d.). No specific instance specifications for the free plan is mentioned in Netlify's documentation. This is due to them using a serverless architecture to handle requests, which means the site is automatically deployed to a distributed network of servers and served from the location closest to the user (Varty, 2020).

Signing up for Netlify is a simple process, and no credit card is needed for the Starter plan. To deploy the site, a new site is added using the GitHub integration. In the settings before deploying the site, several configurations can be made. Before starting the deployment, the environment variables for connecting to the Firebase bucket are added and the deployment is started.

Shortly after the deployment has started, it fails due to several errors. In the deploy log, the output "Treating warnings as errors because of process.env.CI = true" can be seen. Some warnings are triggered when deploying the Huldra framework, which are related to the code. The output shows that these warnings are treated as errors, and this is what causing the deployment to fail. Pandey (2021) expands upon this in a Dev Community post and provides a solution. This is a result of a change Netlify started on June 15, 2020, which is adding the environment variable 'CI' with the value of 'true' to build environments. The reason for adding this change is that the environment variable 'CI', short for Continuous Integration, is commonly set in various CI environments and the cloud ecosystem has largely agreed to use this setting to detect when a build is executing in a CI environment, as opposed to a local development environment. This can create problems for some builds, just like it has with the Huldra project, but a simple configuration can be added to avoid the deployment to run this environment variable as true. In the basic build settings of the deployment, the build command can be changed to "CI= npm run build", which will unset CI for the NPM command. When redeploying

the site, the deployment is successful as the warnings are no longer treated as errors. The site can now be accessed by a generated URL under the site overview section. The domain will be the site name created during deployment followed by '.netlify.app'.

When accessing the link from the site overview it works as it should, but when refreshing this page or copying the link and pasting it in another browser tab a 404 page not found message is shown. This is the same issue that occurred with the Render test. Netlify doesn't provide a UI for adding rewrite or redirect rules like Render does but provides a solution where the rules can be changed by adding a file inside the build folder of the app named "_redirects" without a file extension (Netlify Docs, 2023). To include this file in the build folder when deploying, it must be added to the public folder of the source code. In the "_redirects" file, the rule in Figure 3.6 is added and the file is uploaded to the repository. Now the site is ready to be deployed and benchmarked.

```
≡ _redirects ×
public > ≡ _redirects
1 /* /index.html 200
```

Figure 3.6 Netlify redirect rule

3.6 Railway

The first thing to consider is Railways pricing. They offer three plans which are named Starter, Developer, and Team (Railway Docs, n.d.). The starter plan is a trial of the service designed to help a potential customer evaluate the platform and is fit for hobbyist projects. A trial plan gets \$5 of credits that do not accumulate, and a credit card is not needed to create an account. There is an execution limit to the trial plan and users get 500 execution hours per month. As long as a site is up and running, these hours will decline. Once an account on the free plan has run out of hours for a month, deployments will be suspended until the next month. The Starter plan includes 512 MB of RAM, 1 GB disk space, and shared CPU/container. Max 5 member can be a part of a project with a limit of max 3 concurrent deploys per user. Further there are a max total of 5 project, 5 services, and 5 database per user. The Developer plan is usage or prepaid based. This plan is meant for serious hobbyist use-cases and workloads. It has an unlimited

execution time and deploys remain online indefinitely. A Starter plan will be upgraded to a Developer plan when a credit card is added to the account or the user has a prepaid credit balance on the account. With this plan, any usage above \$5 will be billed for. The Developer plan include up to 8 GB of RAM, 100 GB of disk space, and 8 vCPU cores as well as max 5 members per project. The Team plan is designed to provide the most resources for all scaled needs. With this plan project usage is charged as well as the number of seats the team pays for. The price per seat is \$20. Up to 32 GB of RAM, 2 TB of disk space, and 32 vCPU is included in the Team plan.

After an account is created, a project can be created in the dashboard by connecting a GitHub repository with the framework. As in Render, environment variables can be added in the first step and the deployment is executed automatically. During the first deployment, it is observed that the CI environment variable is set to True as it was in Netlify. As explained in the Netlify deployment test, this causes warnings to be treated as errors and the deployment fails. The same fix for Netlify applies here, and the deployment is carried out with the build command `'CI=false npm run build'` and not stopped by warnings. Even so, the deployment crashes after some time with the log error message *'The build failed because the process exited too early. This probably means the system ran out of memory or someone called `kill -9` on the process.'* The reason for this is unknown as the same repository works with Render and Netlify. This could be caused by configurations in the Railway back end.

As the normal Railway deployment ran into issues, a deployment using a Dockerfile will be tested as an alternative. This can be a useful test as some users might want to fine tune a deployment more which can be done with a Dockerfile. Railway has a built-in automated discovery of Dockerfile, and simply uploading a Dockerfile to the repository will cause the deployment to restart using the Dockerfile as a configuration for the commands executed on the server. For a new initial deployment, it will also discover the Dockerfile and use it as the base for deploying. Figure 3.7 shows that the Dockerfile has been automatically detected and is used to deploy the site. The Dockerfile contains the same configurations used in Figure 3.3 in the OpenStack deployment. With these configurations the deployment is successful, and a

domain can be generated and accessed. As Railway needs an explicit port to listen on to expose the application to the internet, an environment variable with the variable 'PORT' and the value '80' is added in the variable settings provided in the Railway UI for the project. Port 80 is chosen as this is the default port for HTTP traffic.

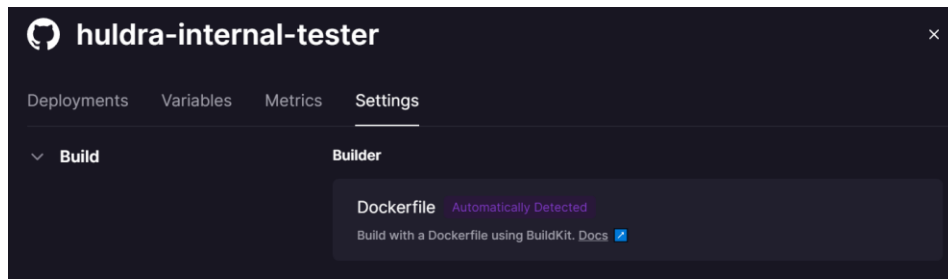


Figure 3.7 Dockerfile detection in Railway

To connect to the Firebase database, environment variables need to be added to the Railway project. As environment variables can be added with a Dockerfile, it is updated with new configurations. The updated Dockerfile is presented in Figure 3.8, with the implementation of environment variables. A solution for researchers who want to deploy their survey with a Dockerized railway deployment is to add the Dockerfile in their private repository. Another solution is to include the Dockerfile in the public repository with dummy values which the researchers can fill in with their Firebase configurations before deploying the site. The environment variables could also be added with the .env file. It should be noted that users should be careful with adding the environment variables to the Dockerfile, as the value of environment variables can inadvertently leak in logs and are revealed when running docker inspect (Amazon Web Services, Inc. (n.d.).

```
Dockerfile M X
Dockerfile
1 FROM node as build
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 ENV REACT_APP_FIREBASE_API_KEY="*****"
7 ENV REACT_APP_FIREBASE_AUTH_DOMAIN="*****"
8 ENV REACT_APP_FIREBASE_PROJECT_ID="*****"
9 ENV REACT_APP_FIREBASE_STORAGE_BUCKET="*****"
10 ENV REACT_APP_FIREBASE_MESSAGING_SENDER_ID="*****"
11 ENV REACT_APP_FIREBASE_APP_ID="*****"
12 ENV REACT_APP_FIREBASE_ROOT_DIRECTORY="*****"
13 RUN npm run build
14
15 FROM nginx
16 COPY --from=build /app/build /usr/share/nginx/html
```

Figure 3.8 Dockerfile with environment variables

3.7 Vercel

Vercel's pricing will first be analyzed. Hobby, Pro, and Enterprise are the plans offered (Vercel, n.d.). The Hobby plan is for personal or non-commercial projects and is free of charge forever. The pro plan is for team collaboration with advanced features and starts at \$20 per user a month. If a custom plan is needed, the Enterprise plan can be used. This is aimed at teams with more security, support, and performance needs. Every build deployed in Vercel is provided with 8 GB of RAM, 4 CPUs, and 13 GB of disk space, but some limits are set for the Hobby plan (Vercel Docs, 2023).

Preparing and deploying a Huldra survey with Vercel is a simple process. Once an account has been created, a new project can be added in the dashboard. Here the GitHub repository can be imported, and settings can be configured. In the settings a UI is provided for adding environment variables, where the Firebase connection parameters are added. Now the deployment can be executed. After the project has finished building and deploying, a new domain is provided where the site can be accessed. For a Vercel deployment, no extra configurations are needed.

4. Experiments and Results

This chapter will cover the experiments conducted to obtain measurable and quantitative data. The experiments will consist of benchmarks and analysis for each solution and deployment, and the result data will be presented after each experiment.

4.1 OpenStack

To begin the experiment, the Docker image is built from the prepared Dockerfile with the “docker build” command. When the image is created, a container can be created from the image with the “docker run” command. The command also includes a user specified port that a rule has been created for in the OpenStack dashboard. When the container is up and running the site can be accessed with the URL of the homepage together with the chosen port and IP address of the VM. Redeploying the site after a GitHub change will be a manual process compared to using a CSP, unless other configurations are done. When redeploying the site, the user has to access the folder of the cloned GitHub repository and execute the command ‘git pull’ with the URL of the project. When the pull is completed, the site can be redeployed by using the steps above.

The first metric to examine is the time it takes to deploy a survey with the OpenStack solution. From creating the VM instance to having the container up and running, the time used will depend on the technical knowledge of the person executing the deployment. This makes this a somewhat difficult metric to estimate. Even so, regardless of the person's technical expertise, the deployment can be quite time-consuming as some configurations and commands must be finished in order to create the Dockerfile, build the image and launch the container. It can span from 15 minutes to a couple of hours. If the person executing the deployment has experience working with OpenStack, or similar cloud platforms, and Docker, the time to deploy will most likely be reduced quite a bit. On the other hand, if the person starts fresh with these concepts the time to deploy will be more time-consuming. The time to deploy will also depend on the servers' specifications and in turn the VMs specification or the resources allocated to the VM from the server, as these factors are tied to the speed of which a deployment will be completed.

The time for building the image using the setup for this experiment will be further benchmarked to obtain quantitative metrics. To benchmark the build time for a fresh setup, all build cache is first deleted using the command 'docker builder prune'. When the caches are removed, the build can be timed using the command 'time docker build .'. The output is displayed in figure 4.1 which reports a total build time of 4 minutes and 19 seconds. The time it takes to create a container from this image is insignificant as it only takes a couple of seconds.

```
=> exporting to image
=> => exporting layers
=> => writing image sha256:e90ab29aa7fdacfcfc0b994cfa17dc9f34ee9a063af0c0fc3555a725a0eaeaeae
2.11user 0.84system 4:19.83elapsed 1%CPU (0avgtext+0avgdata 47588maxresident)k
```

Figure 4.1 Initial deploy time

To test the time for a redeploy, an update is first done to the repository and pulled to its location in the VM. A new image with a new name is then built with the updated code. The new build time is 1 minute and 41 seconds and can be seen in Figure 4.2. The figure also shows how step 2, 3, and 4 of the build has been cached, which reduces the time quite a bit. Before deploying a new container with the updated image, the old one is stopped and removed with the commands "docker stop <container-id>" and "docker rm <container-id>". Lastly, the new updated container is created. The total time for the redeploy process including was around 5 minutes. This process is also somewhat dependent on the technical skill of the person performing the task but shouldn't take more than 15 minutes.

```
ubuntu@master-huldra:~/huldra-internal-tester$ sudo time docker build .
[+] Building 100.7s (14/14) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 217B                                           0.0s
=> [internal] load .dockerignore                                               0.2s
=> => transferring context: 2B                                                 0.0s
=> [internal] load metadata for docker.io/library/node:latest                 1.1s
=> [internal] load metadata for docker.io/library/nginx:latest                1.1s
=> [build 1/6] FROM docker.io/library/node@sha256:0efc3ef3fea2822c9d16da084c40181ed7f74b6f45141100580f9887ccc8e9 0.0s
=> [internal] load build context                                              0.1s
=> => transferring context: 20.63kB                                           0.0s
=> [stage-1 1/2] FROM docker.io/library/nginx@sha256:480868e8c8c797794257e2abd88d0f9a8809b2fe956cbfbc05dcc0bca1f 0.0s
=> CACHED [build 2/6] WORKDIR /app                                           0.0s
=> CACHED [build 3/6] COPY package.json .                                     0.0s
=> CACHED [build 4/6] RUN npm install                                         0.0s
=> [build 5/6] COPY . .                                                       0.4s
=> [build 6/6] RUN npm run build                                             95.5s
=> CACHED [stage-1 2/2] COPY --from=build /app/build /usr/share/nginx/html 0.0s
=> exporting to image                                                         0.0s
=> => exporting layers                                                         0.0s
=> => writing image sha256:e90ab29aa7fdacfcfc0b994cfa17dc9f34ee9a063af0c0fc3555a725a0eaeaeae 0.0s
0.76user 0.37system 1:41.29elapsed 1%CPU (0avgtext+0avgdata 47940maxresident)k
```

Figure 4.2 Redeploy time

As access to the OS is available with this solution, a technical benchmark of the OpenStack VMs resources can be conducted. To get a more meaningful insight to the data, the benchmark should also be performed on another VM instance from another provider. To get access to another instance, Google Cloud Platform is used. Here an account is created which includes access to a VM. The two instances will be further used to compare results from benchmarking resources. Sysbench is used for the benchmarking commands and is installed on the VM. The first resource to benchmark is the CPU. This is done with the command “sysbench cpu --cpu-max-prime=20000 --threads=2 --time=60 run”. Figures 4.3 and 4.4 show the results from each instance. ‘Total number of events’ is the important metric here is. The OpenStack instance has a result of 43897 total number of events compared to Google Clouds instance with a result of 41629. The higher the value, the better the performance. This means the OpenStack instance has the better result.

```
ubuntu@master-huldra:~$ sysbench cpu --cpu-max-prime=20000 --threads=2 --time=60 run
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
  events per second:   731.53

General statistics:
  total time:          60.0028s
  total number of events: 43897

Latency (ms):
  min:                 2.67
  avg:                 2.73
  max:                 18.30
  95th percentile:   2.76
  sum:                 119852.65

Threads fairness:
  events (avg/stddev): 21948.5000/51.50
  execution time (avg/stddev): 59.9263/0.04
```

Figure 4.3 OpenStack CPU test results

```

mikkelhiorthoy@cloudshell:~ (storm-wall-385119)$ sysbench cpu --cpu-max-prime=20000 --threads=2 --time=60 run
sysbench 1.0.20 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 2
Initializing random number generator from current time

Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
  events per second:   693.76

General statistics:
  total time:          60.0029s
  total number of events: 41629

Latency (ms):
  min:                 2.57
  avg:                 2.88
  max:                 6.43
  95th percentile:    3.13
  sum:                 119952.80

Threads fairness:
  events (avg/stddev): 20814.5000/9.50
  execution time (avg/stddev): 59.9764/0.00

```

Figure 4.4 Google Cloud CPU test results

The next metric to benchmark is the memory throughput. This can be done with the command “sysbench memory --memory-oper=write --memory-block-size=1K --memory-scope=global --memory-total-size=4G --threads=2 --time=30 run”. The findings for each instance can be seen in Figure 4.5 and 4.6. The important numbers in the results are the total time taken as well as the latency (Ruostemaa, 2018). The lower the total time, the better the performance. The OpenStack instance performs the task in 1.7153s compared to Google Clouds 1.0216s. This indicates that the Google Cloud instance performs tasks dependent on memory faster.

```

General statistics:
  total time:          1.7153s
  total number of events: 4194304

Latency (ms):
  min:                 0.00
  avg:                 0.00
  max:                 3.48
  95th percentile:    0.00
  sum:                 2096.61

Threads fairness:
  events (avg/stddev): 2097152.0000/0.00
  execution time (avg/stddev): 1.0483/0.01

```

Figure 4.5 OpenStack memory test results


```

General statistics:
  total time:                1.0216s
  total number of events:    4194304

Latency (ms):
  min:                       0.00
  avg:                       0.00
  max:                       0.56
  95th percentile:         0.00
  sum:                       1429.74

Threads fairness:
  events (avg/stddev):       2097152.0000/0.00
  execution time (avg/stddev): 0.7149/0.01

```

Figure 4.6 Google Cloud memory test results

The next step is to benchmark the input/output performance of the storage system with the “sysbench fileio” parameter. The command used is “sysbench fileio --file-total-size=512M prepare”. The parameter “--file-total-size” is set to 512M which specifies that files with combined size of 512MB will be used as test sample for a transfer. The result for each instance is shown in Figure 4.7 and Figure 4.8. The OpenStack test takes approximately 30 seconds longer than Google Cloud, with over a 100 mb/s slower speed. This shows that the Google Cloud instance is much faster when it comes to storage input/output related tasks.

```
524288000 bytes written in 35.80 seconds (13.97 MiB/sec).
```

Figure 4.7 OpenStack storage speed test results

```
536870912 bytes written in 4.26 seconds (120.31 MiB/sec).
```

Figure 4.8 Google Cloud storage speed test results

4.2 Render

With the deployment prepared and ready for execution, an analysis of the performance can be conducted. The first metric to test is the time to deploy, with both initial deploy time and redeploy time after a GitHub change is made. In the menu for the site, events happening during deployment and building can be seen. Events from the first time deploy can be seen in Figure 4.9. As can be seen from the timestamps in the events, it takes 3 minutes from the deployment start to the site being live. To test the time it takes to redeploy the site when a GitHub change is performed, a file that is not being used is removed from the repository. When the file is removed from the repository, a redeployment of the site is immediately triggered in Render. The time it takes from the start of the redeploy to the site is live again takes 2 minutes which is shown in with the events figure 4.10. Thus, a redeploy takes 1 minute shorter compared to the

initial deployment which is insignificant and satisfactory. It can be concluded that a deployment in Render from start to finish is very fast as it takes around 5-10 minutes to prepare, with the actual deployment taking 3 minutes.

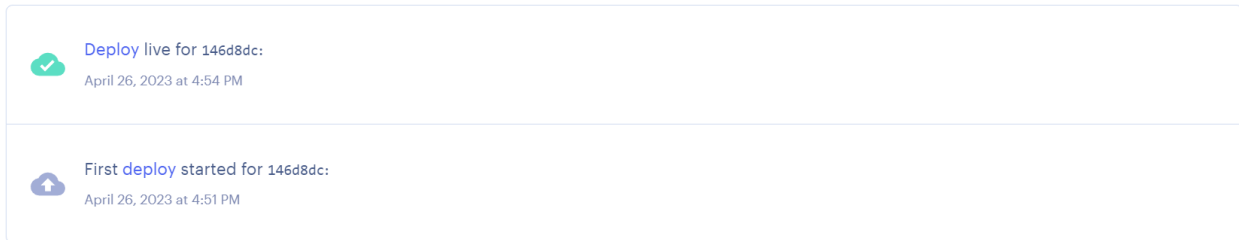


Figure 4.9 First time Render deployment

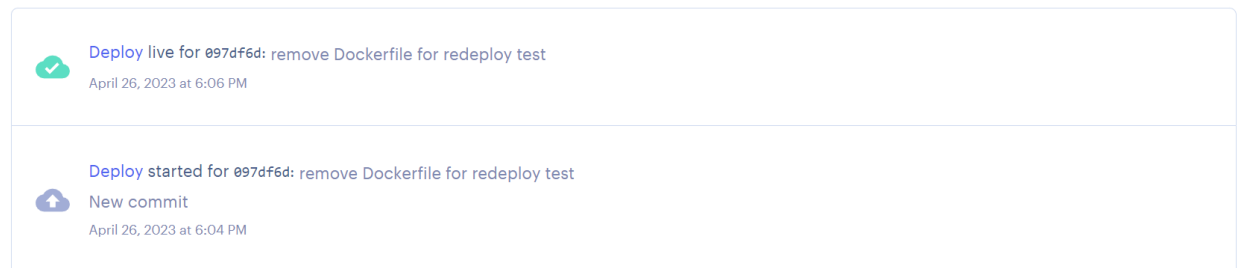


Figure 4.10 Redeployment of Render site

Now that the site is up some testing can be done to check the performance of Render's infrastructure. The first test is to ping the server and check the network latency. To ping the server where the site is hosted, the command "ping huldra-render-internal.onrender.com" is executed in a terminal. Figure 4.11 shows the result of the ping. The latency shows an average of 2 ms which is low and acceptable. This may change depending on what the survey contains, but the survey template hosted has a low latency.

```
C:\Users>ping huldra-render-internal.onrender.com

Pinging gcp-us-west1-1.origin.onrender.com.cdn.cloudflare.net [216.24.57.3] with 32 bytes of data:
Reply from 216.24.57.3: bytes=32 time=1ms TTL=55
Reply from 216.24.57.3: bytes=32 time=3ms TTL=55
Reply from 216.24.57.3: bytes=32 time=1ms TTL=55
Reply from 216.24.57.3: bytes=32 time=4ms TTL=55

Ping statistics for 216.24.57.3:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 4ms, Average = 2ms
```

Figure 4.11 Render ping test

The next factor to benchmark is the FCP, LCP and TTFB to analyze how fast the server hosted by Render responds. To check this, as mentioned, GTmetrix will be used. The result of the analysis is seen in Table 4.1. GTmetrix reports that all metrics are good and that there is nothing that can be improved in the code. All the metrics are satisfactory and will later be compared to the other cloud deployment alternatives.

Metric	Render
FCP	532ms
LCP	715ms
TTFB	90ms

Table 4.1 Render metrics analysis with GTmetrix

4.3 Netlify

To start the analysis, the site is first deployed using the settings and preparations explained in the previous chapter. Benchmarking the time to deploy is conducted during a new deployment. The initial deploy time for Netlify is presented in figure 4.12. It shows that the total deploy time took 1 minute and 2 seconds, while the build time took 1 minute and 1 second. A GitHub change to the repository is further made to get the redeploy time. The results are shown in Figure 4.13, with both build time and total deploy time being around 20 seconds faster than the initial deployment. The total deploy time with preparations of the deployment takes about 10-20 minutes, depending on the technical skill of the publisher.

```
Build time: 1m 1s. Total deploy time: 1m 2s
Build started at 10:39:06 PM and ended at 10:40:07 PM.
```

Figure 4.12 Netlify initial deploy/build time message

```
Build time: 39s. Total deploy time: 40s
Build started at 10:49:12 PM and ended at 10:49:51 PM.
```

Figure 4.13 Netlify redeploy time

Testing the network latency for the Netlify site is not possible with the ping command as the request times out. This might be due to some firewall or other security setting at Netlify's

backend. So, we move one to the FCP, LCP and TTFB test. GTmetrix is used again, and the results can be seen in Table 4.2. The report indicates that all metrics are good and there are no recommendations provided for change of code. This indicates, among other aspects, that the server responds quickly and without hiccups.

Metric	Netlify
FCP	687ms
LCP	831ms
TTFB	292ms

Table 4.2 Netlify metrics analysis with GTmetrix

4.4 Railway

With the Railway site ready to be deployed, the time it takes to deploy and redeploy can be evaluated. With the repository finished and ready for deployment it takes approximately 2 minutes to build and deploy the project with the results shown in the activity log displayed in Figure 4.14. A redeploy of the site after changes to the repository takes about 1 minute and 30 seconds. The total time of a deployment including preparations with Railway takes under 5 minutes, if the Dockerfile is prepared and configured. The time to deploy including configuring the Dockerfile depends on the configurations that a survey creator wants to include. If custom commands are to be included and tested, the time to deploy can be slower as testing may be time-consuming.

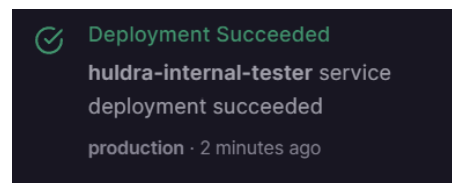


Figure 4.14 Successful Railway deployment

The network latency of the server will now be tested using the ping command in a terminal. The result of the test is displayed in Figure 4.15. The average latency is 160ms, which is relatively high, and above the acceptable limit described in the background chapter.

```
C:\Users>ping huldra-internal-tester-production.up.railway.app

Pinging huldra-internal-tester-production.up.railway.app [104.196.232.237] with 32 bytes of data:
Reply from 104.196.232.237: bytes=32 time=159ms TTL=102
Reply from 104.196.232.237: bytes=32 time=161ms TTL=102
Reply from 104.196.232.237: bytes=32 time=161ms TTL=102
Reply from 104.196.232.237: bytes=32 time=161ms TTL=102

Ping statistics for 104.196.232.237:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 159ms, Maximum = 161ms, Average = 160ms
```

Figure 4.15 Railway network latency test

The metrics that includes server response will be measured using GTmetrix. The results are displayed in Table 4.3. The report indicates all metrics are good and no immediate improvements are needed.

Metric	Railway
FCP	876ms
LCP	949ms
TTFB	98ms

Table 4.3 Railway metrics analysis with GTmetrix

4.5 Vercel

The time it takes for the initial deployment with preparations is approximately 5 minutes. Preparation of the deployment, which included importing the repository and adding the environment variables, takes about 3 minutes. The actual building and deployment take 1 minute and 20 seconds which can be seen in the upper right corner of Figure 4.16. Figure 4.17 shows the summary of a redeploy of the site after a change to the repository. The redeploy takes 39 seconds.

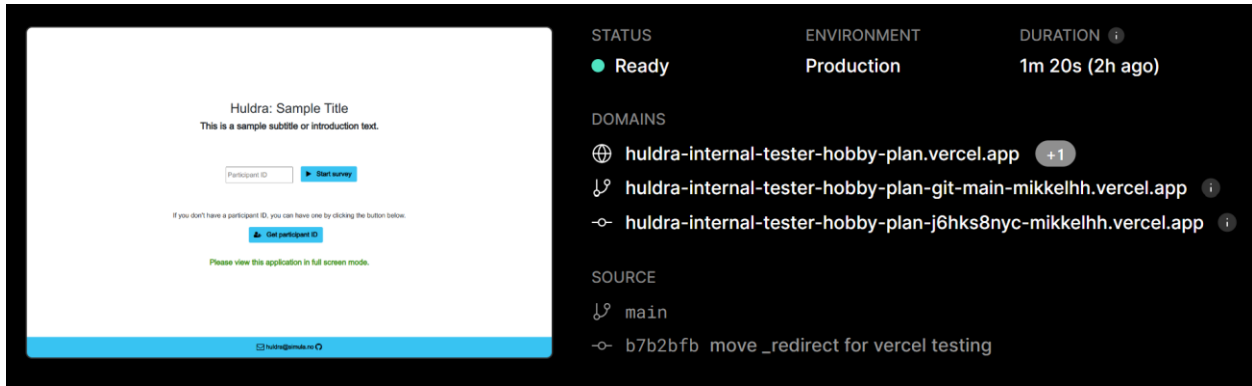


Figure 4.16 Vercel deployment summary

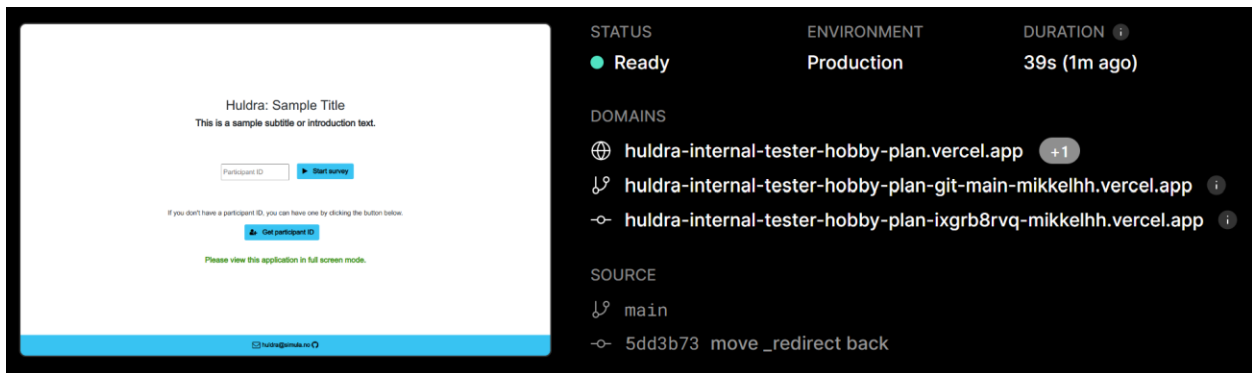


Figure 4.17 Vercel redeploy summary

Now that the survey is up and running, a metrics analysis can be performed. The first metric to examine is the network latency. Figure 4.18 shows the result of a ping test of the site. The ping test results with an average latency of 8 ms, which can be classified as a low and acceptable latency.

```
C:\Users>ping huldra-internal-tester-hobby-plan.vercel.app

Pinging huldra-internal-tester-hobby-plan.vercel.app [76.76.21.93] with 32 bytes of data:
Reply from 76.76.21.93: bytes=32 time=8ms TTL=117
Reply from 76.76.21.93: bytes=32 time=7ms TTL=117
Reply from 76.76.21.93: bytes=32 time=8ms TTL=117
Reply from 76.76.21.93: bytes=32 time=9ms TTL=117

Ping statistics for 76.76.21.93:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 7ms, Maximum = 9ms, Average = 8ms
```

Figure 4.18 Vercel ping test

Now the FCP, LCP and TTFB can be measured. Table 4.4 shows the relevant metrics from GTmetrix. The report further states that all metrics are good, and no changes to the code is needed. This also indicates that the server responds quickly.

Metric	Vercel
FCP	543ms
LCP	833ms
TTFB	51ms

Table 4.4 Vercel metrics analysis with GTmetrix

5. Discussion

This chapter will first compare the results of the data obtained during deployments and experiments. It will further mention other contributions, and lastly, discuss challenges found throughout the project and potential future work.

5.1 Comparison of results

Table 5.1 displays a comparison between the different cloud platforms with performance metrics. The time to deploy does not include preparation of the deployment, but the actual build and deploy time executed by the cloud infrastructure. Google Cloud is included in the table even though a deployment was not executed on this platform. This is to get a better overview of the performance of OpenStack resources compared to another VM instance.

	Initial deploy time	Redeploy time	CPU (higher is better)	Memory (lower is better)	I/O (higher is better)	FCP	LCP	TTFB	Latency
OpenStack (VM instance)	4m 19s	1m 41s	43897	1.7153s	13.97 MiB/s	N/A	N/A	N/A	N/A
Google Cloud (VM instance)	N/A	N/A	41629	1.0216s	120.31 MiB/s	N/A	N/A	N/A	N/A
Render	3m	2m	N/A	N/A	N/A	532ms	715ms	90ms	2ms
Netlify	1m 2s	40s	N/A	N/A	N/A	687ms	831ms	292ms	N/A
Railway	2m	1m 30s	N/A	N/A	N/A	876ms	949ms	98ms	160ms
Vercel	1m 20s	39s	N/A	N/A	N/A	543ms	833ms	51ms	8ms

Table 5.1 Comparison between the cloud alternatives

Netlify offers the fastest initial deploy time, whereas OpenStack has the slowest, with the remaining solutions falling in between these two. The redeploy time is fastest with Vercel and slowest with Render. With no obvious outlier in the data for deploy times, and the deploy times being generally fast, it can be concluded that this metric is not vital when choosing between these solutions. The results from the benchmarking of CPU, Memory, and file I/O are exclusive to OpenStack and Google Cloud. The data obtained from the OpenStack instance compared to the Google Cloud instance indicates that Google Cloud has a faster instance and could be a better solution than using the infrastructure provided by OsloMet. The metrics concerning

server response time (FCP, LCP, TTFB) and network latency are exclusive to the automatic deployments at public CSPs. The results for response time do not show an obvious outlier and are also based on other factors than server response time. This indicates that these metrics should not make a big difference when choosing a solution. The network latency shows a negative outlier for Railway with an average of 160ms. This latency can be defined as relatively high, which is a factor to consider when selecting an alternative.

The differences in features for each CSP will be further discussed. The OpenStack solution is difficult to compare with the other solutions as this alternative can be customized in various ways. With the OpenStack solution, the resources and features will depend on the infrastructure used. It is therefore not included in Table 5.2, which displays a comparison of features included in the free plans of each CSP.

	Bandwidth	Build execution /month	Uptime hours	Ram	Disk	CPU
Render	100 GB	500 mins	750/month	512 MB	Ephemeral	0.1
Netlify	100 GB	300 mins	N/A	N/A	N/A	N/A
Railway	100 GB	N/A	500/month	512 MB	1 GB	Shared
Vercel	100 GB	100 hrs	N/A	8 GB	13 GB	4

Table 5.2 Comparison of CSP features

Bandwidth refers to how the amount of data that can be transferred between the hosted website and its visitors. Each solution comes with the same bandwidth allocation of 100 GB. Some differences can be seen under build execution. Build execution refers to how many minutes the solution allows for building and deploying sites and services. Vercel has a positive outlier with a 100 hours of build time per month, and if a lot of building and deploying is a requirement this could be a good choice. It should be noted that Vercel documentation states that you are limited to build a maximum of 32 deployments per hour and 100 deployments per day (Vercel Docs, 2023). Uptime hours refer to how long a free site is online each month. Render and Railways uptime are limited to 750 hours and 500 hours. This is something to

consider when choosing a service. If the site should be online over longer periods, these services free solutions are not sufficient. For Render's static site option however, the uptime is unlimited, and a static site hosted on Render can be always online. Figure 5.1 shows a screenshot of the free usage for the Individual plan at Render, after the Huldra site has been hosted for several weeks as a static site. Netlify and Vercel does not specify limits on the amount of time a site can be hosted per month on their free plans. Table 5.2 also shows some differences in VM resources offered at each solution, but Vercel again has a positive outlier with its 8 GB of memory, 13 GB disk, and 4 CPUs. If the deployment includes resource heavy websites or apps, Vercel could be a good solution.

Free Usage

Review your free Render usage this month.

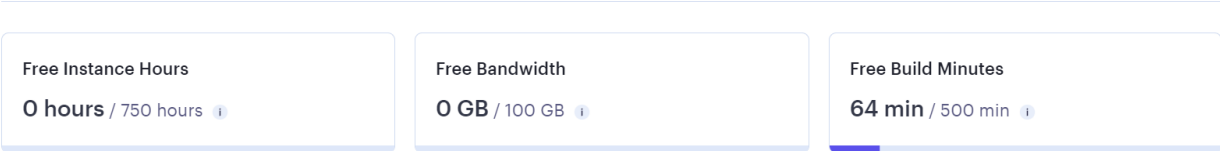


Figure 5.1 Overview of free usage in Render

After a week of monitoring, the recorded uptime in UptimeRobot is reviewed. The findings are displayed in Figure 5.2 and Figure 5.3. Figure 5.2 displays each monitor instance, with the bar on the right side indicating the recorded uptime for the last 24 hours. As all bars are green, no downtime was recorded for the last 24 hours. By inspecting each monitoring instance, the findings from the start of the monitoring to present moment can be reviewed. Figure 5.3 displays the results for Render, which indicates no downtime during a week of monitoring. The other sites have the same results for the monitoring events displayed in Figure 5.3, with no downtime recorded. This indicates that all CSPs have maintained persistent uptime. It should be noted that that due to the limitation of updates occurring every 5 minutes, it is possible for downtime to occur in intervals shorter than this.

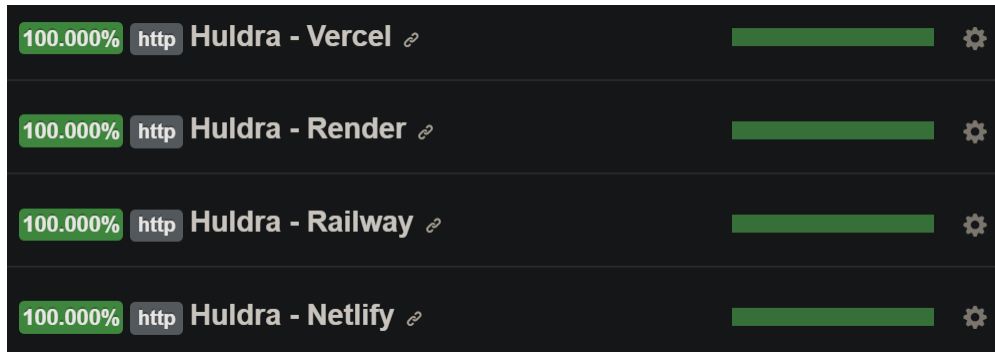


Figure 5.2 Monitoring instances in UpTimeRobot

Latest Events (up, down, start, pause) [Export Logs](#)

Event	Date-Time	Reason	Duration
↑ Up	2023-05-06 21:31:29	OK (200)	138 hrs, 25 mins
▶ Started	2023-05-06 21:30:51	Started	0 hrs, 0 mins

Figure 5.3 Recorded monitoring events

5.2 Other contributions

- Insight to the steps required for both an automatic and manual cloud deployment and the differences in complexity.
- Configurations and commands needed to deploy a site on each CSP included. The configurations and commands were specific for the Huldra use case but can also be helpful for similar projects.
- Dockerfile configurations for a Dockerized deployment.
- How to include environment variables in a cloud deployment.

5.3 Challenges and future work

During the writing of this thesis, some challenges were met. The first challenge was benchmarking resources at free plans offered by CSPs, where no credit card was mandatory. These services did not provide an access to the OS and file system for specific instances, which made it impossible to benchmark system resources in the same manner as the OpenStack instance. As it was important to be able to compare the resource performance of two instances, research was conducted to find a comparable service. Google Cloud offered access to

an instance where, even though a credit card was needed, a free trial could be used, and the account would only be charged if manually upgrading to a paid account. This provided the solution for comparison between to instance performances. For future work, a benchmarking comparison of several instances could be conducted by using paid plans or features.

Another challenge was benchmarking the performance of the sites being hosted at the CSPs under stress and real-world use. Getting enough real users to the visit the sites for it to be put under stress was out of the scope for this thesis. Further work that can be conducted regarding this is analyzing a site that is operating with real users. For Huldra, this would mean analyzing the hosting of an actual survey with active participants. Another potential solution for future work related to this could be to perform load testing. Load testing is the process of using specific software that increases the load on a website or application (Wang et al., 2013). This method is used for performance evaluation and allows for monitoring a website or app's performance under heavy usage, with simulated users.

The Dockerized OpenStack deployment was conducted using manual commands. An alternative approach for this deployment could involve automating these manual commands. This would eliminate the need to input the same commands repeatedly during deployment testing, thus saving time. In future work, a script could be developed for these commands to create an automated Docker deployment.

The monitoring of the sites for detecting downtime was performed in a relatively short period. Ideally, this should have been done earlier since a website typically needs to remain online for more than a week. This would have provided a more thorough evaluation of the network and infrastructure stability of each CSP. In future work, sites deployed with CSPs could be monitored over a longer period. Other tools could also be utilized to provide additional analysis of stability.

6. Conclusion

6.1 Thesis summary

This thesis has conducted an analyzation and benchmarking of five cloud deployment and hosting solutions, with one manual and four automatic deployments approaches. Firstly, relevant background information has been presented. This chapter involves concepts, technologies, and features of cloud computing and CSPs. The next chapter presents the ideas, solutions, and implementations for addressing the problem statement and research questions. This includes what requirement should be considered when choosing a CSP and deployment method, as well as the approach and metrics to be used for the benchmarking experiments. The fourth chapter consists of experiments and results. Here, benchmarking experiments are conducted, and the results are presented. The results are further discussed and elaborated in the next chapter, and the data findings of the different solutions are compared.

6.2 Main Contributions

The thesis involves research and analysis of CSPs and cloud deployment approaches to understand the differences in performance and features. The objective is to address the challenges of selecting a CSP and cloud deployment method, and to provide solutions for deploying and hosting websites or applications in the cloud. To evaluate the proposed solutions, Huldra was used as a test subject. As Simula is exploring options for hosting their application, a subgoal of the study was to identify an alternative to their current solution. Multiple proposed solutions were included to conduct a comparison and determine the most suitable option for use cases such as Huldra. To address the problem statement and research questions, a quantitative analysis and benchmarking of the proposed solutions and approaches has been conducted. Two methods for deploying a site were presented to examine and highlight the differences in deployment time and complexity. The first approach involved automated deployment of a GitHub repository provided in the CSPs dashboard, and was tested with four different CSPs. The second approach consists of a manual Dockerized deployment, executed with commands in a terminal, and was implemented using a VM deployed in OpenStack.

The deployment process for each solution has been briefly described, and the deployment times have been benchmarked. For the automatic deployments conducted at CSPs, metrics such as response time and network performance of the deployed site were obtained using GTmetrix and ping. The metric results from these automated deployments were further analyzed and compared to identify any anomalies and determine if a particular solution outperforms the others. For the manual OpenStack deployment approach on a VM with OS access, benchmarking of the VM's resources was performed. To analyze the results, a benchmark of an instance in Google Cloud was also conducted. The results and data findings from each instance were further analyzed and compared to the performance of the OpenStack instance.

The knowledge gained from this work is the approaches and methods for analyzing a cloud service provider and deployment method, which can be further used when selecting a cloud service provider and deployment method for a similar use case to Huldra. The discussion and comparison of the results demonstrates how benchmarking and analysis of performance can be effective in researching cloud solutions. The conclusion present which of the solutions is the most optimal for the use case and gives an overview of how the research questions are answered.

6.3 Revisiting the problem statement

Research question 1: How can customer benchmark and analyze cloud computing resources and performance?

To find answers to the research question stated above, several solutions has been presented and tested. Conducting a benchmark of cloud resources has been explored by utilizing the sysbench tool in a VM deployed in OpenStack. Here, approaches for benchmarking the CPU, memory and disk have been described and tested. These are straightforward steps and should provide customers with a solution to benchmarking the system resources of a VM. This method can be done with several instances and the results from each can be further compared.

Pricing and resources offered at CSPs has been presented by analyzing the pricing plans offered at the CSPs included in the study. The overview of the pricing plans can be found on the

websites of each CSP and can usually be analyzed further by reviewing the documentation. The analyzation conducted in this thesis indicates that prices among CSPs are similar. The pricing plans offered at Render, Netlify, and Vercel has the smallest option as a free starter/individual plan and the most complex plan as an enterprise option. Railway differs somewhat with its Trial and Team plan. The resources included in the free plans varies. Vercel showed a positive outlier here, with its 8 GB of RAM, 4 CPUs, and 13 GB of disk space. For a more resource heavy project where a free plan is preferred, the Hobby plan for Vercel could be a good alternative.

A metric included for the benchmarking of performance was time to deploy and redeploy. These were recorded by analyzing build and deployment logs. The output of the logs and messages usually reports the time it takes to build and deploy a project, and they can be inspected to acquire time related metrics. The results from analyzing the deploy times for the different cloud solutions included in this thesis showed no obvious outlier. Further benchmarking was conducting regarding response time and network latency for the cloud hosted sites. The response time analysis showed no obvious bottlenecks in the systems, and the sites respond similar. For network latency, Railway had a negative outlier with 160ms. This is generally considered a high latency and should be considered when concluding on a solution. Monitoring the sites for potential stability issues and downtime can be used with a tool like UpTimeRobot. Here a straightforward process of adding the site to the dashboard can be implemented for free. The review of each site's stability was presented in 5.1 and shows no recorded downtime.

Research question 2: Based on the results obtained from the analysis and benchmarking presented, which of the selected CSPs and deployment methods included are suitable for a use case like Huldra?

This objective consists of obtaining a solution that would be the most suitable for a use case like Huldra. The selection of a solution should be based on the analysis and benchmarking results, as the obtained metrics and their comparison provide an indication of the solution with the most optimal performance. As stated above, every build in Vercel is provided with 8 GB of memory, 4 CPUs, and 13 GB of disk space. This is compared to the resources provided in the

other CSPs a positive outlier. The build execution times provided in Vercel also has a positive outlier with 100 hours per month. It should be noted that there are some limitations, with 32 deployments per hour and 100 deployments per day. During building and deployment testing, it was observed that no additional configurations were needed for Vercel in form of redirect rules or build commands. Compared to the other services, this is positive aspect as it leads to faster and simpler deployments. Railway, for example, had compared to Vercel various configurations needed, and ran into issues. Both the 14 days Pro trial, and the Hobby plan was tested for Vercel, and no difference in deployment performance was observed. Based on these findings, it can be concluded that an automatic deployment with Vercel would be the most optimal solution for a use case like Huldra among the CSPs included in this study. By extending the research and implementing the relevant work presented in section 5.3, this could be further confirmed.

References

- Amazon Web Services, Inc. (n.d.). Secrets management. *AWS Docs | Security*. URL: <https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/security-secrets-management.html>
- Amazon Web Services, Inc. (n.d.). What Is An SSL/TLS Certificate? *AWS – Security, Identity, & Compliance*. URL: <https://aws.amazon.com/what-is/ssl-certificate/>
- Amazon Web Services, Inc. (n.d.). What Is Network Latency? URL: <https://aws.amazon.com/what-is/latency/>
- Aslanpour, M. S., Gill, S. S. & Toosi, A. N. (2020). Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. *Internet of Things*, 12. URL: <https://doi.org/10.1016/j.iot.2020.100273>
- Bhutta, K. S. & Huq, F. (1999). Benchmarking – best practices: an integrated approach. *Benchmarking: An International Journal*. pp. 254-268. URL: <https://doi.org/10.1108/14635779910289261>
- Boduch, A. & Derks, R. (2020). React and React Native: A complete hands-on guide to modern web and mobile development with React.js. *Packt Publishing Ltd*. pp. 10 URL: https://books.google.no/books?hl=no&lr=&id=XCLhDwAAQBAJ&oi=fnd&pg=PP1&dq=react+js&ots=Buat-yyrfO&sig=O0pv-vhct8LBEA8OT7xR7SLiEjE&redir_esc=y#v=onepage&q=react%20js&f=false
- Christensson, P. (2013). Framework. *TechtermsTerms | Software Terms*. URL: <https://techterms.com/definition/framework>
- Clark, J. (n.d.). Heroku vs Railway - Which is the best option? *Back4App Blog | Learn*. URL: <https://blog.back4app.com/heroku-vs-railway-which-is-the-best/>
- Cui, Y., Qian, Q., Guo, C., Shen, G., Tian, Y., Xing, H. & Yan, L. (2021). Towards DDoS detection mechanisms in Software-Defined Networking. *Journal of Network and Computer Applications* vol. 190. URL: <https://doi.org/10.1016/j.inca.2021.103156>
- Dhingra, A. K., & Rai, D. D. (2020). Study on the Performance Analysis Tools and Techniques of Cloud Computing. *Journal of Critical Reviews*, 7(15). URL: <https://jcreview.com/admin/Uploads/Files/61f2b656875437.04815083.pdf>
- Docker. (n.d.). Docker overview. *Docker Docs*. URL: <https://docs.docker.com/get-started/overview/#docker-objects>

Docker. (n.d.). Docker Registry. *Docker Docs | Open-source projects*. URL: <https://docs.docker.com/registry/>

Docker Docs. (n.d.). Runtime options with Memory, CPUs, and GPUs. *Docker Docs | Docker Engine*. URL: https://docs.docker.com/config/containers/resource_constraints/

Duan, W., Hu, M., Zhou, Q., Wu, T., Zhou, J., Liu, X., Wei, T. & Chen, M. (2020). *Reliability in cloud computing system: a review*. *Computer research and development, Vol 57*. pp. 102-123
URL: https://dro.deakin.edu.au/articles/journal_contribution/Reliability_in_cloud_computing_system_a_review/20706409

Fisher, C. (2018). Cloud versus On-Premise Computing. *American Journal of Industrial and Business Management, 08(09)*. URL: <https://doi.org/10.4236/ajibm.2018.89133>

Fitzgerald, A. (2022, July 12). What Is Latency & How Do You Improve It? *Hubspot*. URL: <https://blog.hubspot.com/website/what-is-latency>

Gantikow, H., Walter, S. & Reich, C. (2020). Rootless Containers with Podman for HPC. *Lecture Notes in Computer Science*. pp. 343–354 URL: https://doi.org/10.1007/978-3-030-59851-8_23

Hammou, M., Midoglu, C., Hicks, S. A., Storås, A., Sabet, S. S., Strümke, I., Riegler, M. A. & Halvorsen, P. (2022). Huldra: a framework for collecting crowdsourced feedback on multimedia assets. *Proceedings of the 13th ACM Multimedia Systems Conference (MMSys '22)*. pp. 203–209. URL: <https://doi.org/10.1145/3524273.3532887>

Hassan, J., Shehzad, D., Habib, U., Aftab, M. U., Ahmad, M., Kuleev, R. & Mazzara, M. (2022). The Rise of Cloud Computing: Data Protection, Privacy, and Open Research Challenges—A Systematic Literature Review (SLR). *Computational Intelligence and Neuroscience*, vol. 2022. Article ID 8303504, 26 pages. DOI: 10.1155/2022/8303504. URL: <https://pubmed.ncbi.nlm.nih.gov/35712069/>

Hawksworth, P. (2021, February 23). Terminology explained: Atomic and immutable deploys. *What are Atomic deploys? Immutable deploys? Learn here! Netlify Blog*. URL: <https://www.netlify.com/blog/2021/02/23/terminology-explained-atomic-and-immutable-deploys/>

Heroku. (n.d.). Removal of Heroku Free Product Plans FAQ - Heroku Help. *Heroku Support*. URL: <https://help.heroku.com/RSBRUH58/removal-of-heroku-free-product-plans-faq>

Heroku. (n.d.). What is Heroku. *Heroku*. URL: <https://www.heroku.com/what>

Heroku Docs. (2022, November 16). GitHub Integration (Heroku GitHub Deploys). *Heroku Dev Center | Deployment*. URL: <https://devcenter.heroku.com/articles/github-integration>

Hicks, S., Storås, A., Riegler, M., Midoglu, C., Hammou, M., de Lange, T., Parasa, S., Halvorsen, P. & Strümke, I. (2022). Visual explanations for polyp detection: How medical doctors assess intrinsic versus extrinsic explanations. *arXiv preprint arXiv:2204.00617*. URL:

<https://arxiv.org/abs/2204.00617>

IBM. (n.d.). What is Docker?. *IBM | Cloud*. URL: <https://www.ibm.com/topics/docker>

J. Shah & D. Dubaria. (2019). Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform. *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. DOI: 10.1109/CCWC.2019.8666479. URL:

<https://ieeexplore.ieee.org/document/8666479>

Kirilin, V., Sundarrajan, A., Gorinsky, S. & Sitaraman, R. K. (2019). RL-Cache: Learning-Based Cache Admission for Content Delivery. *NetAI'19: Proceedings of the 2019 Workshop on Network Meets AI & ML*, pp. 57–63. URL: <https://doi.org/10.1145/3341216.3342214>

Kumar, R., Gupta, N., Charu, S., Jain, K. & Jangir, S. K. (2014). Open source solution for cloud computing platform using OpenStack. *International Journal of Computer Science and Mobile Computing*, Vol. 3(5). Pp. 89-98. DOI: 10.13140/2.1.1695.9043. URL:

https://edisciplinas.usp.br/pluginfile.php/318402/course/section/93669/V315201427_3.pdf

Lăcătușu, M., Ionita, A. D., Anton, F. D. & Lăcătușu, F. (2022). Analysis of Complexity and Performance for Automated Deployment of a Software Environment into the Cloud. *Applied Sciences*, 12(9), 4183. MDPI AG. URL: <http://dx.doi.org/10.3390/app12094183>

Lin, C., Nadi, S. & Khazaei, H. (2020). A large-scale data set and an empirical study of docker images hosted on docker hub. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. pp. 371-381. DOI: 10.1109/ICSME46990.2020.00043. URL:

https://ieeexplore.ieee.org/abstract/document/9240654?casa_token=Bw3WT09jGsQAAAAA:SoS5x07JZrvQyYzg5o3tpmg6RcwEm3hx-a-NbAbfpOsTUCOHhFqSm_xh6AeUvOwWivzjqxcbxEg

Lomas, A. (2023, January 25). How to Choose the Right Cloud Service Provider? *Net Solutions | Cloud Services*. URL: <https://www.netsolutions.com/insights/how-to-choose-cloud-service-provider/>

Masdari, M. & Zangakani, M. (2019). Green Cloud Computing Using Proactive Virtual Machine Placement: Challenges and Issues. *J Grid Computing* 18, pp. 727–759. URL:

<https://doi.org/10.1007/s10723-019-09489-9>

Mell, P. & Grance, T. (2011). The NIST definition of cloud computing. URL:

<http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf>

Netlify. (n.d.). Netlify Pricing. *Netlify website*. URL: <https://www.netlify.com/pricing/>

Netlify Docs. (2023, May 5). File-based configuration. *Netlify Docs | Configure builds*.

URL: <https://docs.netlify.com/configure-builds/file-based-configuration/>

Netlify Docs. (2023, May 4). Get started with Netlify CLI. *Netlify Docs | CLI*.

URL: <https://docs.netlify.com/cli/get-started/>

Netlify Docs. (2023, April 3). Redirects and rewrites. *Netlify Docs | Static routing*. URL:

<https://docs.netlify.com/routing/redirects/>

Netlify Docs. (2023, May 4). Site deploys overview. *Netlify Docs | Site deploys*.

URL: <https://docs.netlify.com/site-deploys/overview/>

Nielsen, B. B., Hassanshahi, B. & Gauthier, F. (2019). Nodest: feedback-driven static analysis of Node.js applications. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. pp. 455-465 URL:

https://dl.acm.org/doi/abs/10.1145/3338906.3338933?casa_token=UBW7DoPq7FsAAAAA:g0tcbkTbTNhzPMnhHkdWseljDVnE3bo9Urr1224tJO4fDKd5wtl70TNIcVMayIPFo1N-BLu7OZI3fuc

OsloMet. (n.d.). Autonomous Systems and Networks (ASN). *OsloMet - Storbyuniversitetet*.

URL: <https://www.oslomet.no/en/research/research-groups/autonomous-systems-networks>

Pakvis, S. (2018, May 25). Using environment variables in React. *Medium*.

URL: <https://trekinbami.medium.com/using-environment-variables-in-react-6b0a99d83cf5>

Pandey, K. (2021, February 5). [Solved] "Treating warnings as errors because of process.env.CI = true". *DEV Community*. <https://dev.to/kapi1/solved-treating-warnings-as-errors-because-of-process-env-ci-true-bk5>

Patel, B. (2021). Cloud Computing Deployment Models: A Comparative Study. *International Journal of Innovative Research in Computer Science & Technology*. Vol 9. DOI:

10.21276/ijrcst.2021.9.2.8. URL:

https://www.researchgate.net/publication/350721171_Cloud_Computing_Deployment_Models_A_Comparative_Study

Rani, D. & Ranjan, R. K. (2014). A comparative study of SaaS, PaaS and IaaS in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6). URL:

https://d1wqtxts1xzle7.cloudfront.net/49900472/cloud_iaas_saas_paas-libre.pdf?1477545579=&response-content-disposition=inline%3B+filename%3DA_Comparative_Study_of_SaaS_PaaS_and_IaaS.pdf&Expires=1684067423&Signature=DeANefdqJ5f6OC2efDmUqyFEZtPoKDLf~lDtdk02sML9O88RdYo-ygvIFjIXGWKVJ7fNN4CNbkm661GrL6-

[Kww3MCKqjGXM9kArf5nb6tJien3c0v6ZN0istReqpt9W0Esnd18d0IjzAZUaeBRmT2BDQV3xzNBZE1~lz0X7IfQZNx0SchtvuDwozImiyHAW6q9kUII9uYj1vwtQziTESke4Gjp8RYo~NW0rASJsEh1fTGN184oCMDEgCtShvVsQIRdsajYY6rbXWdhIvgABkTrOIvePYJ-5V38lx0oFrS6iKJo8y3Z3MhtN8H2Zwp~K72o7e~g-MSQoCctmozW7tA &Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA](https://kww3mckqjgxm9kArf5nb6tJien3c0v6ZN0istReqpt9W0Esnd18d0IjzAZUaeBRmT2BDQV3xzNBZE1~lz0X7IfQZNx0SchtvuDwozImiyHAW6q9kUII9uYj1vwtQziTESke4Gjp8RYo~NW0rASJsEh1fTGN184oCMDEgCtShvVsQIRdsajYY6rbXWdhIvgABkTrOIvePYJ-5V38lx0oFrS6iKJo8y3Z3MhtN8H2Zwp~K72o7e~g-MSQoCctmozW7tA &Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)

Railway. (n.d.). Bring your code, we'll handle the rest. *Railway website*. <https://railway.app/>

Railway Docs. (n.d.). Deployments. *Railway Docs | Deploy*.

URL: <https://docs.railway.app/deploy/deployments>

Railway Docs. (n.d.). Plans. *Railway Docs | Reference*. URL:

<https://docs.railway.app/reference/plans>

Rashid, A. & Chaturvedi, A. (2019). Cloud Computing Characteristics and Services: A Brief Review. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*. 7. pp. 421-426. DOI: 10.26438/ijcse/v7i2.421426. URL:

https://www.researchgate.net/publication/331731714_Cloud_Computing_Characteristics_and_Services_A_Brief_Review

Ruostemaa, J. (2018, December 19). Evaluating cloud server performance with sysbench.

UpCloud | Long reads. URL: <https://upcloud.com/blog/evaluating-cloud-server-performance-with-sysbench>

Render. (n.d.). Cloud Application Hosting for Developers. *Render*. <https://render.com/>

Render. (n.d.) A Cloud for the New Decade. *Render | About*. URL: <https://render.com/about>

Render Docs. (n.d.). Disks. *Render Docs | Disks*. URL: <https://render.com/docs/disks>

Render Docs. (n.d.). Free Instance Types. *Render Docs | Free Instance Types*. URL:

<https://render.com/docs/free>

Render Docs. (n.d.). How Deploys Work. *Render Docs | Deploy*. URL:

<https://render.com/docs/deploys>

Render Docs. (n.d.). Redirects and Rewrites. *Render Docs*. URL:

<https://render.com/docs/redirects-rewrites>

Render Docs. (n.d.). Static Sites. *Render Docs | Service Types*. URL:

<https://render.com/docs/static-sites>

React. (n.d.). Introducing JSX. *React Docs*. URL: <https://legacy.reactjs.org/docs/introducing-jsx.html>

Saraswat, M. & Tripathi, R. C. (2020). Cloud Computing: Comparison and Analysis of Cloud Service Providers-AWs, Microsoft and Google. *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*. pp. 281-285. DOI:

10.1109/SMART50582.2020.9337100. URL:

https://ieeexplore.ieee.org/abstract/document/9337100?casa_token=jybUV72fnDwAAAAA:7cUbE3eCYmZ8_k3Q5CE0e7q3Gzz9PRP9pQ3dk6Pe7RD7aQ5NSNsCuPRfZlopbgaxoZBGX3yhFUM

Scheuner, J. & Leitner, P. (2019). Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with Cloud WorkBench (Tutorial). *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems*. pp. 257-258, DOI: 10.1109/FAS-W.2019.00070.

URL:

https://ieeexplore.ieee.org/abstract/document/8791993?casa_token=lpLLDNa5RNsAAAAA:7UvBhY6CUhEecf9qxiZNfXfSqY-NZB_vuO0suR_wwk2nEzQlogA90rgqw5bHI_nCM4ysu8LBL4

Sealuzh. (n.d.). Cloud WorkBench (CWB). *GitHub - sealuzh/cloud-workbench*.

URL: <https://github.com/sealuzh/cloud-workbench>

Sigdestad, T. (2023, January 26). *What is the difference between server-side and client-side?*

Enonic. URL: <https://enonic.com/blog/what-is-the-difference-between-server-side-and-client-side>

Simula. (n.d.). Huldra: A Framework for Collecting Crowdsourced Feedback on Multimedia Assets. *GitHub - simula/huldra: HOST Department Huldra project*. URL:

<https://github.com/simula/huldra>

Singh, P., Gupta, P., Jyoti, K. & Nayyar, A. (2019). Research on Auto-Scaling of Web Applications in Cloud: Survey, Trends and Future Directions. *Scalable Computing: Practice and Experience*, 20(2). pp. 399–432. URL: <https://doi.org/10.12694/scpe.v20i2.1537>

Stevenson, D. (2018, September 25). What is Firebase? The complete story, abridged. *Medium*.

URL: <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>

UptimeRobot. (n.d.). UptimeRobot: Free Website Monitoring Service. *UptimeRobot*. URL:

<https://uptimerobot.com/>

Varty, J. (2020, August 4). What is Netlify and What are its Benefits? *Agility Blog | Jamstack, Netlify, Headless CMS*.

<https://agilitycms.com/resources/posts/what-is-netlify-and-why-should-you-care-as-an-editor>

Vercel. (n.d.). Find a plan to power your projects. *Vercel | Pricing*. URL:

<https://vercel.com/pricing>

Vercel Docs. (2023, February 6). Free Trial Limits. *Vercel Documentation* | Payments & Billing. URL: <https://vercel.com/docs/concepts/payments-and-billing/free-trials>

Vercel Docs. (2023, February 6). Getting Started with Vercel. *Vercel Documentation* | Get Started. URL: <https://vercel.com/docs/concepts/get-started>

Vercel Docs. (2023, March 6). Limits. *Vercel Documentation* | Limits. URL: <https://vercel.com/docs/concepts/limits/overview>

Vercel Docs. (2023, February 15). Preview Deployments Overview. *Vercel Documentation* | Deployments. URL: <https://vercel.com/docs/concepts/deployments/preview-deployments>

Wagner, J. & Pollard, B. (2023, Januar 19). Time to First Byte (TTFB). *web.dev* | Metrics. URL: <https://web.dev/ttfb/>

Walton, P. (2022, October 19). First Contentful Paint (FCP). *web.dev* | Metrics. URL: <https://web.dev/fcp/>

Walton, P. & Pollard, B. (2023, May 4). Optimize Largest Contentful Paint. *web.dev* | Fast load times, Core Web Vitals. URL: <https://web.dev/optimize-lcp/>

Wang, X., Zhou, B. & Li, W. (2013). Model-based load testing of web applications. *Journal of the Chinese Institute of Engineers*, 36(1). pp. 74–86. URL: <https://doi.org/10.1080/02533839.2012.726028>

Williamson, E. & Lengstorf, J. (2016, September 29). A Step-by-Step Guide: Deploying on Netlify. *Netlify Blog – Guides & Tutorials*. URL: <https://www.netlify.com/blog/2016/09/29/a-step-by-step-guide-deploying-on-netlify/>

Ylonen, T. (n.d.). What is SSH (Secure Shell)? *SSH Academy*. URL: <https://www.ssh.com/academy/ssh>

Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T. & Hu, S. (2019). Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Generation Computer Systems*, 93. pp. 278–289. URL: <https://doi.org/10.1016/j.future.2018.10.046>

Z, Mahmood. (2011). Cloud Computing: Characteristics and Deployment Approaches. *2011 IEEE 11th International Conference on Computer and Information Technology*. pp. 121-126, DOI: 10.1109/CIT.2011.75. URL: https://ieeexplore.ieee.org/abstract/document/6036733?casa_token=z30X1pP_bI0AAAAA:0c86QJRpoq12iiKXhi19PRP7copPIaCOG9RJ5Pf3yLn57tTGuhm8rbevzfkLCTXBr81WXch6vUc