

Cloud Operations to Support UX and Accessibility for Crowdsourced Online Survey Framework Deployment

Bjørge Seim Øvstedal



Thesis submitted for the degree of
Master in Applied Computer and Information Technology (ACIT)
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2023

Cloud Operations to Support UX and Accessibility for Crowdsourced Online Survey Framework Deployment

Bjørge Seim Øvstedal

© 2023 Børge Seim Øvstedal

Cloud Operations to Support UX and Accessibility for Crowdsourced Online Survey
Framework Deployment

<http://www.oslomet.no/>

Printed: Oslo Metropolitan University

Abstract

This study investigates different cloud service providers (CSPs) and their deployment platforms in terms of user friendliness and experience. With the help of Docker virtualization, allowing to create an isolated environment, the application deployment process can be replicated on different platforms and prevent technical issues linked to the underlying technical configurations of a CSP. It further allows for non-technical users to choose their CSP based on requirements other than the necessary technical aspects and deploy their applications. To ensure a higher level of user experience (UX) and accessibility for application deployment, we investigate ways of optimizing cloud operations.

Docker is used to create a container image along with Docker Compose to define multi-container applications. The Docker Compose files are created as base files for further development with a production and development environment. Using these files, an evaluation is conducted on the features, services, and deployment processes of different CSPs. We conducted a deployment test using both a web-based GUI and a terminal for the CSP, which resulted in a consistent and comparable deployment. As discovered through testing, using the developed Dockerfiles prevent certain issues of the technical infrastructures of CSPs.

As part of the thesis, a subjective user study is conducted consisting of a pre-questionnaire, a deployment guide and a post-questionnaire involving System Usability Scale (SUS). This user study uses the Huldra application for deployment on a CSP named Render. The pre-questionnaire responses regarding familiarity with computer science, application deployment, and Docker did not appear to significantly impact the deployment time. We also analyzed the relationship between the variables in terms of time and difficulty level of each deployment step where we found few significant trends. We also found low ratings of difficulty throughout the user study and a high mean score in participants responses with respect to user experience.

Keywords— Cloud Service Provider (CSP), Deployment Platform, Docker, Docker Compose, Virtualization, Container

Acknowledgments

First and foremost, I would like to thank my principal supervisor Pål Halvorsen for his guidance and contributions during this thesis. I would also like to thank my co-supervisors, Cise Midoglu and Saeed Sabet, for their astonishing support and feedback throughout this thesis. Without them, this thesis would not be possible.

Contents

- Abstract** **i**

- Acknowledgments** **iii**

- 1 Introduction** **1**
 - 1.1 Motivation 2
 - 1.2 Problem Statement 4
 - 1.3 Scope 5
 - 1.4 Research Methods 6
 - 1.5 Ethical Considerations 6
 - 1.6 Main Contributions 7
 - 1.7 Thesis Outline 7

- 2 Background and Related Work** **9**
 - 2.1 Subjective User Studies 9
 - 2.2 Huldra Framework 10
 - 2.2.1 Huldra Pages 10
 - 2.2.2 Huldra Frontend with React 12
 - 2.3 Cloud Fundamentals 14
 - 2.3.1 Infrastructure as a Service 15

2.3.2	Platform as a Service	15
2.3.3	Software as a Service	16
2.4	Relevant Concepts and Technologies	16
2.4.1	Virtualization	16
	Hypervisor Type 1	17
	Hypervisor Type 2	17
2.4.2	Containerization	17
	Docker	19
2.4.3	Application Deployment	20
	Deployment without Technical Expertise	21
2.4.4	DevOps	22
	Software Development Lifecycle	22
2.5	Deployment Platforms	24
2.5.1	Fly	24
2.5.2	Railway	24
2.5.3	Render	25
2.6	Related Work	25
2.7	Chapter Summary	28
3	Methodology	29
3.1	Proposed Approach	29
3.2	Plan for Subjective Study	31
3.2.1	Method	31
3.2.2	Participant Recruitment	31
3.2.3	Study Material	32

3.3	Chapter Summary	36
4	Implementation	37
4.1	Containerization of Huldra	38
4.1.1	Multi-stage Builds	39
4.1.2	Dockerfile Production Environment	39
4.1.3	Try Files for Nginx	40
4.1.4	Dockerfile Development Environment	41
4.1.5	Docker Compose files	43
4.2	Alternative Deployments	47
4.2.1	Huldra Deployment using ALTO Cloud	47
	Production Environment	47
	Development Environment	50
	Container Solution for Huldra	51
4.2.2	Huldra Deployment using Render	51
4.2.3	Huldra Deployment using Railway	60
4.2.4	Huldra Deployment using Fly	66
4.3	Chapter Summary	70
5	Results	73
5.1	Objective Results	73
5.2	Subjective User Study	77
5.2.1	Data Cleaning	77
5.2.2	Study Results	78
5.3	Chapter Summary	83

6 Discussion	85
6.1 Addressing the Research Questions	85
6.2 Lessons Learned	87
6.3 Other Contributions	88
6.4 Limitations	89
6.5 Future Work	89
7 Conclusion	91

List of Figures

2.1	One-way vs. two-way data binding	13
4.1	ARG and ENV availability	38
4.2	Dockerfile for production environment	40
4.3	Try_files	41
4.4	Dockerfile for development environment	42
4.5	Base Docker Compose file	43
4.6	Docker Compose production file	44
4.7	Docker Compose development file	45
4.8	Docker Compose start command for production environment	47
4.9	Docker Compose production environment output	48
4.10	Production container successfully created	48
4.11	List containers and Docker Compose down	49
4.12	Docker Compose starting a development environment	50
4.13	Docker volume and location	50
4.14	Available Render services	52
4.15	Render repository connection	53
4.16	Render Deployment configuration	54
4.17	Render Deployment advanced configuration	55

4.18	Render secret file	56
4.19	Render deployment build	57
4.20	Dockerfile build stage commands executed in Render	57
4.21	Build commands output, creation of production build	58
4.22	Successful build stage	58
4.23	Output from Nginx on Render	59
4.24	Render Web Service dashboard	59
4.25	Railway project selection	61
4.26	Railway variables selection	61
4.27	Railway raw editor with an example of Firebase connection parameters	62
4.28	Railway variables filled including Firebase connection parameters, port and Dockerfile path	62
4.29	Dockerfile detection	63
4.30	Render build stage complete	64
4.31	Railway separate logs, Nginx output	64
4.32	Railway environment settings	65
4.33	Railway email preferences	65
4.34	Fly personal dashboard	67
4.35	Fly configuration file	68
4.36	Flyctl secrets list	69
4.37	Flyctl deployment output	70
4.38	Fly application dashboard	70
5.1	Outliers based on total time per step after sign in	77
5.2	User study gender results	78

5.3	User study age results	79
5.4	Difficulty level per Deployment Step	80
5.5	Correlation Matrix, pre-Questionnaire and time	80
5.6	Correlation Matrix, pre-Questionnaire and step difficulty	81
5.7	Render - System Usability Scale (SUS) scoring of the participants	82
6.1	Grep command and found file	86
6.2	Firebase api key location	86
6.3	Firebase api key using web developer tool	87
7.1	Render - user study pre-questionnaire	94
7.2	Render - user study step 1	95
7.3	Render - user study step 2	96
7.4	Render - user study step 3	97
7.5	Render - user study step 4	98
7.6	Render - user study step 5	99
7.7	Render - user study step 6	100
7.8	Render - user study step 7	101
7.9	Render - user study post-questionnaire	102
7.10	Fly - user study pre-questionnaire	103
7.11	Fly - user study step 1	104
7.12	Fly - user study step 2	105
7.13	Fly - user study step 3	106
7.14	Fly - user study step 4	107
7.15	Fly - user study step 5	108

7.16 Fly - user study step 6	109
7.17 Fly - user study step 7	110
7.18 Fly - user study step 8	111
7.19 Fly - user study post-questionnaire	112

List of Tables

- 5.1 Mean and standard deviation of the familiarity questions. 79
- 5.2 Mean and standard deviation of the SUS and QoE questions, first five questions. 82
- 5.3 Mean and standard deviation of the SUS and QoE questions, last six questions. 83

Chapter 1

Introduction

Simula was established as a project under the University of Oslo in 2001 and was founded as a limited liability company in 2002. Simula is owned and managed by the Norwegian Ministry of Education and Science. The research laboratory focuses its research towards communication systems, scientific computing, software engineering, cybersecurity, and machine learning. With research in both the industry and public sector, Simula also works in collaboration with partner universities which allows the education of students and for them to finish their degree by offering projects for master thesis with supervision.

The Department of Holistic Systems (HOST) at Simula started a project called Huldra [20] in 2021 as a framework for deploying custom online surveys, which facilitate conducting subjective user studies in many fields of research. Since its inception, Huldra has had many contributors such as master students and interns. As a framework for collecting crowdsourced feedback on multimedia assets, Huldra has been used for two surveys until 2022, one involving the collection of feedback from medical experts about how they perceive different eXplainable Artificial Intelligence (XAI) methods demonstrated on images from the gastrointestinal (GI) tract [22], and another involving the collection of feedback from the general public about how they perceive alternative thumbnails for a given soccer video clip [23].

Huldra is an open-source framework [20] developed to address the challenges of other traditional survey frameworks or applications. With commercial platforms providing little configuration regarding both the survey as a whole and the backend solution, the making of such surveys can be tiresome although with available tools. Without the use of online survey platforms or tools, the expertise and knowledge within development for a good user interface and interaction is needed as well

as a secure backend depending on the information gathered from the survey. For example, some of the available survey tools require publicly available content to be used within the survey. This can for example be videos which are needed to be uploaded to YouTube or other video sharing websites in order to be used within a survey. A survey creator should not be obligated to upload its survey content publicly in order to be used in a survey as the survey and the content may or may not be used internally in an organization where the information should not be made public for security reasons or otherwise. Regarding privacy and security, other platforms are also not clear where the participant data and responses are stored.

With focus on the participants regarding design of a study, the framework aims to reduce the complexity of such studies to make participants complete the studies and provide feedback. A study should not feel like a chore and a cumbersome task and this is what Huldra tries to prevent along with the easy creation of a study for the creator. It also focuses on diversity with universal accessibility. For the creator of a study, it is important that the survey is diverse and inclusive as it gives a more accurate result and clarification of the data. The framework assists this issue with accessibility in mind with 4 alternative formats for the case pages.

1.1 Motivation

Cloud computing has its roots in the concept of time-sharing, which emerged in the 1960s as a way for multiple users to access a single computer simultaneously. DARPA, the Defense Advanced Research Projects Agency, funded Massachusetts Institute of Technology (MIT) for the project called Project on Mathematics and Computation (Project MAC) [31]. From this project and the use of a single computer where a couple of people could gain access simultaneously, full-time-sharing solutions were not available before the early 1970s. Followed by the 1990s, virtualization came to life which paved the way for the development of cloud computing later on. Although virtualization failed to gain widespread adoption, the groundwork of cloud computing was set and by the early 2000s the technology was in place to launch modern cloud services as we know it today.

The idea behind cloud computing and its goal was for businesses to easily gain access to and use computing resources. By delivering computing resources over the internet, these resources could be scaled and accessed on an on-demand basis. As the first cloud-based service, Salesforce introduced Software as a Service (SaaS) in 1999 by delivering enterprise applications via a simple website. Followed by SaaS,

Infrastructure as a Service (IaaS) was pioneered by Amazon Web Services (AWS) in 2006 with the launch of Elastic Compute Cloud (EC2) service. Shortly after, Platform as a Service (PaaS) was introduced in 2008 by Google App Engine [52].

Since the initial goal and focus on allowing businesses to rent computing resources provided by IaaS, cloud computing has expanded over time to include a wider range of services and applications. Including specialized services such as machine learning, data analytics and other areas, cloud computing is also increasingly being used closer to the devices generating it. The overall goal of cloud computing has shifted from simple on-demand access to a more comprehensive set of solutions and services.

Cloud computing has become the 21st century wonder that holds its importance in almost every field. With the increase of collaboration and remote work, cloud computing has become important in today's digital workplace by providing a centralized platform for data access and sharing. Along with scalability, cloud computing brings important but necessary benefits that are needed in a constantly changing world with new technology. Many associate cloud computing with organizations and businesses, but it has become useful for the average person as well. As an essential part of many aspects in a daily life, cloud computing is hard to avoid on a daily basis. From storage such as Google Drive and mobile applications to entertainment such as Netflix, cloud computing is hard to avoid in everyday interaction with the world.

With the growing popularity of cloud computing, deployment and management has become easier for its users. Cloud providers are developing services and tools to reach a wider audience, increasing the accessibility of deployment and management of applications on the cloud. Without extensive technical knowledge, tools are designed to simplify the deployment process which allows users to not be heavily invested in technical understanding. Further third-party tools and services also help users deploy and manage applications on the cloud. For example, containerization platforms like Docker allow users to package their application into portable containers that can be easily deployed on any cloud platform that supports containerization.

As an open-source survey framework, Huldra is available for use by others. The framework should be able to deploy on any deployment platform of choice provided by a CSP. Regardless of the technical expertise of the users, the users should be able to deploy the Huldra framework without any further technical issues caused by the chosen CSP.

1.2 Problem Statement

In a digital world with rapid growth in cloud computing, this thesis aims to explain how tools and services provide accessibility in terms of deployment and management of applications on cloud platforms. By explaining the fundamentals of cloud computing and the underlying technologies, the knowledge is presented to enlighten the process of deploying through existing tools and services along with an investigation of availability of cloud services.

As an important technology in development and deployment of applications or software, containerization is one of the most significant technological advancements in cloud computing. Although as a technological advancement, containerization may imply the complexity of configuring and managing containers. Further explanation aims to get an overall view of for what reason it is increasing in popularity and the current state of containerization in cloud-based deployment.

How can cloud operations be optimized to ensure a high level of user experience and accessibility for application deployment?

- **RQ1:** How can the accessibility of application deployment on cloud be improved through the use of containerization technology and simplify the process of deploying applications on the cloud using cloud service providers?
- **RQ2:** How does cloud service providers facilitate its features and provide access to a wider range of people regardless of technical expertise?

Implemented and tested with the Huldra framework, a containerized solution will be developed together with researching existing cloud-based services to combine and improve accessibility and enhance the user experience regarding application deployment. Containerizing the application keeps the environment in the same state and prevents failures on different platforms it is deployed. For example, if the application is deployed outside of a container on one deployment platform and it works fine, it may not work on another deployment platform. This may happen when a platform uses a different operating system, software libraries, or hardware configurations that does not correspond with the deployed application as it relies on specific dependencies that are not present or configured on the given deployment platform. Containerization helps to address the issue by containerizing the application as it isolates and provides a consistent runtime environment for the application to run in. This allows the application to be deployed without issues

of different configurations on different deployment platforms and enhances the experience of application deployment without the underlying technical expertise.

Deploying an application on the cloud can be difficult for people lacking technical expertise. This technical aspect can hinder some individuals from deploying their applications on the cloud, as they may not be familiar with the deployment process. To simplify the deployment process, many CSP's provide step-by-step guidance to deploy using their platform. Also, many of them offer a web-based integration of the deployment process which the users can follow to successfully deploy an application.

The complexity of the deployment process can also pose a challenge for those with limited technical knowledge, especially when the application has many dependencies that require proper management and updating. Also, dependencies may require specific configuration such as a specific operating system or network settings for the application to function properly. To simplify the deployment process, containerization technology like Docker can bundle the application and its dependencies into a single container that can run anywhere. This eliminates issues that may arise from deploying the application on different platforms and makes troubleshooting and reconfiguration less complex based on the platform the application is being deployed on.

1.3 Scope

The goal of this thesis is to implement a containerized solution of Huldra that standardise the application environment to prevent issues regarding the deployment process. With focus on accessibility, the containerized solution is deployed on several CSP's to observe the deployment in regards to required technical expertise.

The implementation of the Docker image is done through the use of OsloMet Alto Cloud, where it is also tested based on application dependent configurations. Once the Docker image is successfully deployed, further deployment tests are made using Render, Railway and Fly. These tests are made to see how the use of a Docker image is used within the deployment platform, and to see if further configurations are needed beyond the Docker image.

To understand and evaluate the accessibility of CSP, we present a user study of a deployment using one of the CSP called Render. The user study aims to discover difficulties and issues of the deployment process based on various levels of technical expertise. With a pre and post-questionnaire along with the actual guide of

deployment, we will see the correlations between the respondents technical expertise in regards to the guided deployment and the participants UX based on usability.

1.4 Research Methods

This thesis aims to investigate the accessibility of application deployment using cloud service providers with focus on Docker image implementations. To get a better understanding of this topic, a mixed-methods approach combining both quantitative and qualitative data collection methods is utilized. The study involves the creation of Docker images, which are used to evaluate a containerized deployment using various CSP's. The research design also includes a subjective user study that is conducted online using Google Forms with a shared account for Render. The participants are asked to complete a user study from different backgrounds to evaluate the accessibility of application deployment using the CSP.

The mixed-methods approach combining quantitative [3] and qualitative [19] data collection methods is chosen to provide a understanding of the evaluation of cloud service providers and the UX of application deployment. Qualitative research methods will provide insights into the users experiences of using the cloud service providers, which cannot be captured by quantitative data alone. The subjective user study is conducted to evaluate the UX of application deployment and to gather qualitative data on the difficulty of application deployment. By combining both quantitative and qualitative data, the study aims to provide a more complete picture of the evaluation of cloud service providers and to better understand the factors that affect the UX of application deployment.

1.5 Ethical Considerations

In the context of our user study, the collection and handling of sensitive information related to the participants was treated with great care and responsibility. To ensure the protection of the participants privacy, identifying information such as names or contact details was not collected. Further, no questions regarding metadata was required. This allowed participants to provide information at their own will. Furthermore, Self-described and other options were provided for metadata questions, ensuring that participants could provide information at their own discretion. These measures were implemented to respect the participants autonomy and maintain their

privacy throughout the study.

1.6 Main Contributions

In this thesis we provide insight into how the use of containerization technology improves the accessibility of application deployment. We develop Dockerfiles that creates a production build of the Node application Huldra. We also developed Docker Compose files and a Dockerfile for a developer environment were the Docker Compose files further defines the application stack. Using the developed Dockerfile for production, we explored different deployment platforms to examine their features and application deployment using containers. The tested CSP's provided similar features in terms of deployment, but differentiated based on the available deployment methods using either a web-based GUI or via a terminal. We identified several CSP's that did not successfully deploy the application without the use of the developed Dockerfile.

We also conducted a subjective user study to evaluate accessibility of application deployment. The study consisted of a pre-questionnaire with questions regarding familiarity with technical competence, web application deployment and Docker. After the pre-questionnaire, the participants was asked to follow a guide to deploy Huldra on the CSP. The user study ends with the participants answering questions using SUS. Based on the results we determine that the participants in any case based on their technical background, found the CSP to be user-friendly and reported a high level of satisfaction with the deployment procedure.

1.7 Thesis Outline

The rest of this thesis is organized as follows:

- Chapter 2 (Background and Related Work) introduces the Huldra framework and the React library that it is built on. It further explains cloud and its main services before introducing relevant concepts and technologies for the thesis followed by related work.
- Chapter 3 (Methodology and Implementation) explains the containerization of Huldra and shows the deployment of the application on different cloud service providers (CSP).

- Chapter 4 (Results) presents the objective results of the tested CSP's and subjective user study results.
- Chapter 5 (Discussion) addresses the research questions and limitations of the thesis.
- Chapter 6 (Summary and Conclusions) summarizes and concludes the thesis by the results from previous chapters in addition to providing suggestions for future work.

Chapter 2

Background and Related Work

2.1 Subjective User Studies

User studies are commonly used in research to gain insights into user behavior, preferences, and attitudes towards a product or service. These studies can be controlled, where researchers carefully design and execute the study with a specific set of participants and conditions [44], or crowdsourced, where researchers rely on a large and diverse group of participants who volunteer to provide feedback on a product or service. Controlled studies allow researchers to carefully control and manipulate variables to test specific hypotheses, while crowdsourced studies provide a larger and more diverse set of data, but with less control over participant selection and experimental conditions [26].

Subjective user studies are used to gain a better understanding of the opinions, attitudes, and experiences of their users when interacting with a product or service [28]. These studies aim to gather qualitative data on how users perceive the product or service. By collecting feedback directly from users, subjective user studies can provide insights that may not be apparent through other forms of research, such as quantitative data analysis. Subjective user studies are important because they provide insights into the users' needs, expectations, and behaviour. By collecting users feedback, designers and developers can identify the strengths and weaknesses of a product or service and make informed decisions to improve its quality, usability, and user satisfaction [14].

2.2 Huldra Framework

Huldra is a React-based framework built to conduct online surveys for collecting crowdsourced feedback on multimedia assets. The Huldra framework uses React, a JavaScript library, for web development to build interactive elements with encapsulated components that manage their own state and are composed to make more complex user interfaces. React aims to provide a modular and configurable framework for adjusting the user interface. React also allows for easy scalability due to its components which allows for complexity [29]. Along with React, NodeJS is used for Huldra's backend. As a framework of JavaScript, NodeJS is working as the backend of the application with the job of communicating between the actual survey and its storage. Storage in this case, is provided by a Google Cloud Platform (GCP) S3 bucket. The storage holds the content of a survey as well as the responses of participants and is accessed via Firebase. With a folder structure in Firebase, the cases are constructed by uploading the multimedia assets.

Huldra uses Heroku to deploy its applications with triggering of automatic deployments from a GitHub repository. As an option, the use of ALTO Cloud should allow to develop further solutions to Heroku as Huldra's third-party integrations [20] to deploy the Huldra framework on different deployment platforms. This would also give a more secure solution as the data is under the user's control. It could be a better solution as cloud platforms may experience downtime and vulnerabilities such as data theft, data loss, data leakage and more [33]. It further allows to customize the servers for the user's needs.

As a part of the Salesforce Platform, Heroku eliminated its free services in November 2022. Free product plans and data services were stopped, and inactive accounts got deleted along with associated storage for accounts that had been inactive for over a year [55]. For testing purposes, free cloud platform alternatives had to be found. Research resulted in 5 potential cloud platforms including Render, Cyclic, Railway, Deta and Fly. Other Heroku alternatives also included Vercel, Netlify and GitHub pages. With the most similarity to Heroku, Render along with Railway and Fly was chosen for deployment testing of a containerized environment.

2.2.1 Huldra Pages

Huldra offers the overall functionality of registration and login, introduction, questionnaire and summary and feedback. The registration and login functionality

allows participants to collect a new participant ID to be able to participate in a given survey. The registration page also allows the collection of information from the participants. Further, the introduction functionality offers information to the participants on how the expected survey is held. This includes instructions, demonstrations audio and video checks. Followed by the introduction pages, the questionnaire functionality is presented. The questionnaire pages hold the core functionality of the framework where survey questions are introduced to the participants. As explained in the introduction pages, the questionnaire pages are where the participants answer the given survey questions for the several multimedia content. These survey questions are presented in the four available formats of audio, hybrid, image and video. Lastly, the summary and feedback functionality involve a page where the summary of the survey questions and the participant answers are presented. This page also provides a feedback form for participants to provide information regarding the particular study along with information regarding the survey framework itself.

For a survey, the survey questions or cases are presented in the four available formats. For an audio case, the page is constructed by two columns. A case description in the left-middle column with two audio assets for answer options. The most rightward column holds and displays the participant answer. Next, the hybrid case holds a mix of video and image assets and is constructed into three columns. The left column holds the video and the case description. The middle column holds two image assets from the video as answer options, and the rightmost column holds the participant answer. The next available format, image, is used for image assets and the page is constructed of three columns. The left column holds the main image of the case with two image assets in the middle column as answer options. The right column displays the participant answer. Lastly, the video case is constructed of two columns. The left columns hold the case description and two video assets that can be selected. The selected videos, in order, are shown in the right column as the participant answer.

Other available frameworks such as SurveyMonkey and Google Forms are difficult to customize for a particular study and are often designed to work with a certain type of content like audio or video. Survey Monkey for example, does not allow for more than 10 survey questions with 100 results for free. Exceeding this, a monthly payment is needed. Further, Survey Monkey is limited when it comes to customization were formatting a survey can for some people be challenging.

Since the beginning of the coronavirus pandemic, online survey-based studies have increased. With the reduced ability to conduct social surveys and restricting other

social data collection methods, online surveys have since been increased and are now more important [27]. These changes mean that social data collection methods, as an alternative, can be held online through a survey. The conversion of social data collections to an online survey can be tricky and time consuming [4]. Huldra addresses this issue with dynamic and easy setup of surveys with multimedia assets in mind. The framework aims to address this task along with other boundaries when it comes to the creation of a survey and the ability to scale the survey to the needs of the creator towards its participants.

A search of existing popular online survey platforms resulted in a wide range of platforms regarding features, design, customization, data collection and analysis. While some of these platforms are free, some offer a paid plan with more advanced features. Many of these online survey platforms are tailored to a specific type of study. For example, Google Surveys can be used for many different types of studies, but other survey platforms such as SurveyGizmo are popular for market research as it offers features like conjoint analysis and other features regarding marketing. Other examples include Culture Amp which focuses on employee engagement. This survey platform is popular for employee experience and satisfaction as it offers features specifically designed for HR and company leaders. Further categories of survey platforms developed to serve specific studies include academic research, customer satisfaction and healthcare research. Based on the specific needs will be dependent on the survey that is being conducted as well as what kind of data that should be collected and budget.

2.2.2 Huldra Frontend with React

Built and maintained by Facebook, React is one of the most popular frontend frameworks and is used by many, including Apple and Netflix [46]. React offers a lot of benefits when it comes to development. The frontend framework offers speed in which the developers utilize individual parts of the application which ultimately would not cause problems regarding the logic of the application. With high performance in mind, React was designed to use server-side rendering and the use of virtual DOM (Document Object Model). Virtual DOM provides a high level of abstraction from the actual DOM which in return makes it faster and easier to update the UI of an application.

React can use two-way data binding by the use of `LinkStateMixin` instead of one-way where the overall structure flows from parent to child. As the only option, and

close competitor, Angular offers two-way binding in which a changed value in an input box will automatically update the added property value of the component class [29]. This creates a real-time synchronization of data between model and view.

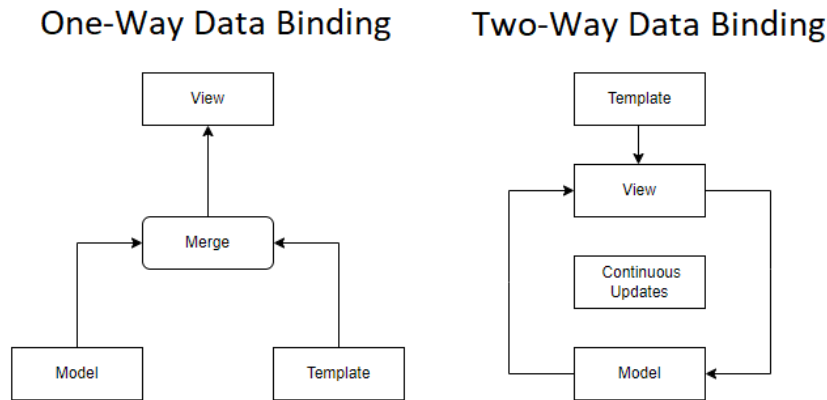


Figure 2.1: One-way vs. two-way data binding

Figure 2.1 displays the difference between one-way and two-way data binding in React. One important performance metric is the speed of updating the real DOM, which can be slow if the whole tree structure needs to be refreshed even for a single change. To avoid this problem, React uses a virtual DOM. When there is a state change, React updates a virtual copy of the real DOM instead of talking to the real DOM directly. React then compares this updated virtual DOM with a snapshot of the virtual DOM right before any changes were made to determine which component needs to be updated in the real DOM. React updates only the changed objects in the real DOM, making the process much faster than manipulating the real DOM directly. This approach is called reconciliation, where a virtual representation of the user interface (UI) is kept in memory and manipulated before updating the ReactDOM [9].

As an alternative option for a front-end development framework, Angular is close in popularity to React. Supported and led by Google, Angular is used widely by big companies such as Google, Microsoft, Samsung and PayPal. Released a few years before React, Angular has its popularity regarding single-page application and more complex applications. Angular provides a fully fledged framework for software development. Additional libraries are usually not required since functions such as data binding, component-based routing, dependency injection and form validation are included within Angular. With these libraries, more complex applications are easier to build. On the other hand, React requires implementation of libraries, but gives more freedom regarding organization of code. With a simpler architecture,

React is easier to learn and understand [57]. There are benefits from both frameworks, but some features should determine the framework depending on development needs.

2.3 Cloud Fundamentals

As an incredibly popular and increasingly used tool, the term “cloud” in the context of technology refers to the concept of delivering computing resources such as servers, storage, databases, networking, software, analytics and intelligence over the Internet [45]. These computing services offer faster innovation, flexible resources and economies of scale. Instead of having to own and maintain physical hardware and software on-premises, users can access these resources remotely from a third-party provider’s infrastructure.

For better understanding, a simple example of cloud computing is using an online file storage service such as Dropbox or Google Drive. Instead of storing files locally on a device, the files are uploaded to a cloud storage service like Dropbox or Google Drive. These services store the files on their servers, providing accessibility over the internet. The files uploaded can be accessed by any device anywhere and can be shared for others to read and write [38]. With the level of accessibility, the ability of collaborations and remote work are increased. For a project or remote employees, this is very beneficial for businesses.

By storing files in the cloud, local storage and its capacity is removed. As an important feature of cloud storage, scalability comes easily where the storage capacity can be increased or decreased deepening on the user’s needs. For businesses, this feature becomes useful when undergoing growth and expansion. Along with scaling the storage capacity, the cost also scale. In cloud and cloud computing, single users and businesses only pay for what they use. More directed towards businesses, flexibility regarding computing resources can be scaled up or down depending on demand [45]. Investing in hardware and software for an on-premises solution is often very expensive and brings along additional drawbacks such as staff and maintenance. Also, a combination of technical and administrative controls is needed within on-premises security to achieve an acceptable standard of security measurements [12]. By using a cloud service, the security is typically managed by the cloud providers in which the cloud provider is responsible for securing the cloud infrastructure.

There are generally three types of clouds, public, private and hybrid clouds. Public cloud refers to a cloud infrastructure that is owned and operated by a third-party provider. Examples of a public cloud provider are Amazon Web Services, Microsoft Azure and Google Cloud, all of which are accessible to anyone. Unlike public clouds, a private cloud is exclusively used by a single user, organization or company. Private clouds can be a company's own on-premises server or be located in a datacenter where the computing resources are not shared with other organizations or the public [1]. Benefits of private clouds include security, low latency and greater control over data governance.

As the name suggests, hybrid clouds refer to a combination of public and private clouds that are integrated and work together to provide a unified computing environment. In a hybrid cloud, organizations can run certain workloads or applications in the public cloud while keeping others in their private cloud. For example, running public cloud services on privately owned infrastructure. Cloud computing is a broad spectrum that encompasses a wide range of technologies and services. These services, provided and maintained by cloud service providers, can be divided into three main categories [45].

2.3.1 Infrastructure as a Service

Infrastructure as a Service (IaaS) is where the cloud service provider manages the entire infrastructure which includes data storage, servers, network and virtualization [53]. With IaaS, users or businesses use these resources to build and run their own application on hardware provided by the cloud service provider. Users of IaaS have complete control over the operating system, applications and software that runs on the virtual machines. Examples of IaaS providers are Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP).

2.3.2 Platform as a Service

Platform as a Service (PaaS) is where the cloud service provider offers a complete platform for developing, running and managing applications [56]. In contrast to IaaS, PaaS gives users more flexibility and ease of use in which IaaS provides more control in terms of the infrastructure. With higher simplicity, users of PaaS do not need to worry about the underlying infrastructure, operating system, middleware and tools.

2.3.3 Software as a Service

Software as a Service (SaaS) is commonly used daily by millions of people. As an increasingly popular choice for businesses, SaaS provided a flexible way to access software applications and services. SaaS allows users to access software applications over the internet, typically accessed through a web browser or a mobile app, without downloading or installing the software on their own devices [45]. Some examples of popular SaaS applications used by many are Google Drive, Zoom, Dropbox, Microsoft Teams and OneDrive. These services are managed by the cloud service provider which maintains the servers, databases and other hardware to assure a consistent delivery of the product.

2.4 Relevant Concepts and Technologies

2.4.1 Virtualization

A key technology used in cloud computing is virtualization which is a foundational technology that enables the development of clouds. Virtualization enables single physical instances to be shared to its users by allowing multiple operating systems to run on the same physical machine. Resources such as CPU, memory, storage and network interface get shared among the users. The physical machine is called the host and the virtual machines running on the physical machine are called guests [40].

As a software-based computer, virtual machines (VM) function like a physical machine with their own operating system and applications. On the same host, via the hypervisor, different operating systems can be used on different virtual machines. As they are independent, a virtual machine is portable where it can be moved from one hypervisor to another hypervisor on a different physical machine. This, along with other benefits of using virtualization, provides speed and efficiency at which virtual machines can be deployed [39]. Starting a virtual machine is quick and is simpler than creating an entire new environment for developers. For example, if a development team needs a testing environment one could simply create or start a new virtual machine designated for that purpose. Virtualization is also beneficial regarding uptime where virtual machines can easily be redeployed by being moved to another physical server or host if that host has crashed. Another key benefit of using virtualization is low costs. Reducing the need for additional hardware along with power, cooling and maintenance, virtualization lowers the cost by running

multiple virtual environments from one piece of infrastructure [40].

Hypervisor Type 1

Installed on the physical machine, the hypervisor is responsible for allocating resources to the virtual machines. Most commonly used are the hypervisor 1, also known as a bare-metal hypervisor, which runs directly on the host hardware. As a separate environment, a guest is allocated its own resources and is isolated from other guests. The isolation ensures that none of the virtual machines can interfere with each other like access to data or resources belonging to each individual guest which improves security and reliability [10]. Examples of Type 1 or bare-metal hypervisors are VMware ESXi, Microsoft Hyper-V and open source KVM.

Hypervisor Type 2

Hypervisor Type 2, in contrast to Type 1, has a layer of host OS that sits between the physical server and the hypervisor. Also known as “hosted” hypervisor, hypervisor Type 2 runs on top of an operating system which is installed on the host computer. With direct access to the hardware resources, Type 2 hypervisors run on an existing operating system and provide virtualization as an application. They are less common and mostly used for end-user virtualization working as a regular application running within the host [10]. Examples of Type 2 or hosted hypervisors are Oracle, VirtualBox and VMware Workstation.

2.4.2 Containerization

Containerization holds a big part in future technological advancements of application deployment and management. The technology allows applications to be packaged along with its dependencies, libraries and configuration files into a single package called a container. As more organizations move their applications to the cloud, containerization has become a critical part of their strategy to build and deploy applications faster and more efficiently [37]. It also provides a standardized deployment process, making application deployment more accessible for non-technical people.

Containerization can prevent certain issues that often appear when deploying and running applications in a traditional environment. These issues often involve

compatibility and dependency conflicts, but may include other issues such as performance issues, security vulnerabilities, deployment and management complexity [7]. For example, for a developer who is developing an application on his local machine, and everything is working as it should, the application can run and function the same way on other machines. A typical scenario is where an application functions as it should for one developer but does not function on another machine for another developer. This can happen when files are missing, software version mismatches or due to different configuration settings like environmental variables. With containerization the application is within its own package with everything it needs to run. For an application consisting of multiple services such as a database, monitoring or backup, each service is isolated in its own container with their needed dependencies [7]. The isolated environment also allows for multiple applications to use different versions of a software side by side where the same dependencies are used but with different versions. These containers are isolated from each other and from the underlying host operating system which then again can run the same containerized application on any system that supports containerization technology. With the use of containerization, one can consistently build, run and ship applications.

Containerization provides multiple benefits including portability, isolation, scalability, consistency and more. As containers are able to run on any system that supports containerization, moving applications from one environment to another is made easy. For example, from a developer's laptop to a production server. Further, isolation and scalability offer reduced conflict between application as well as improved security with the ability to scale up or down the containers based on changes in application demand [41]. Overall, containerization provides a convenient, efficient, and reliable way to run and manage applications.

To use containers, certain components are necessary. Firstly, a host operating system is required, which can be either a physical or virtual machine. This operating system can either be a physical machine or a virtual machine as well as being compatible with the container runtime. Containers typically runs on virtual machines as they provide additional advantages such as isolation between the containers and the host operating system, resource management, portability, compatibility as in integration of containers into existing environment and security. Further components include container runtime like Docker and container images.

Docker

As the prominent containerization platform, Docker is one of the most widely used platforms for containerization with 40 percent of enterprises using it and 30 percent more planning to do so [60]. With its popularity and community support, Docker is well supported and is constantly developed by developers and other contributors in an active community. Docker has gained widespread adoption due to its portability, flexibility, performance and available tools which makes it an attractive option to containerize applications [6]. It also brings tools and services for container orchestration, image management, continuous integration and deployment.

A container may look similar to a virtual machine in terms of virtualization, but as virtual machines virtualize entire machines, containers virtualize software layers above the operating system. VM's is the isolation of machines, while containers are the isolation of processes [5]. Depending on the use case, virtual machines are usually used when there is a need for different operating systems and varying processing power to run multiple applications simultaneously. As containers focus more on packaging applications rather than simulating physical machines, it solves some of the difficulties by using VM's. Combining VM's and containers and running them together and not as competing technologies, brings isolation and management benefits of VM's, and the resource efficiency and flexibility of containers [34].

As a crucial part of the Docker architecture, the Docker Daemon is responsible for managing Docker images, containers, networks and storage volumes. Provided by Docker, the Docker Daemon is accessible and can be interacted with using REST API through the Docker CLI. The Docker Daemon constantly listens for Docker API requests which are made through the Docker CLI.

To create a container, a Docker image must first be created which works as a template that contains the instructions for creating a container. A docker image is a lightweight, stand alone, executable package that includes everything needed to run a piece of software, including the code, libraries, environment variables and configuration files. This set of instructions which specify how to build the image, are defined in a file known as a Dockerfile. The Dockerfile contains information such as which base image to start from, which software packages to install, how to configure the environment, and what command to run when a container is started from the image. In other words, the Dockerfile is used to build the Docker image which is a pre-packaged environment. It is this image that is being executed as a container on a system that supports containerization. The image can then be pushed to some cloud-based registry for easy access by users and other applications.

2.4.3 Application Deployment

As part of the software development process, application deployment is an important stage and encapsulates the process of installing, configuring, updating and monitoring [36]. Also known as software deployment, the process validates an application and makes it available for use by its intended audience. In response to customer preferences, requirements and demand, methods used to build, test and deploy new code are heavily impacted. Along with updates and the delivery of new features, software development has become important in terms of workflows that enables frequent deployment of updates to the production environment.

Deploying the application typically involves transferring application files to the server using File Transfer Protocol (FTP) [16] client or a Command-Line Interface (CLI) [51] tool. FTP is a simpler and easier way of transferring files from a local machine to a remote server using a graphical user interface. It is preferred over CLI for those who are not familiar with CLI but are slower in terms of transferring files and automation of the deployment process. CLI is more flexible but requires direct interaction with the server through a terminal window. With the use of commands, the deployment process can be automated and is more secure compared to FTP as it uses protocols like secure shell (SSH) and SSH File Transfer Protocol (SFTP). Depending on the hosting provider, Git also offers a way to transfer files for deployment. As a version control system, Git has become a common way of deploying an application directly on to the server. With Git installed on the server, developers push their code to a remote repository where the app gets updated by pulling those changes to a directory on the server which contains the application.

An important process after an application is deployed, is the process of monitoring is implemented. Regular checks of performance, security and availability are done to ensure correct functionality and that the application delivers what is intended for end-users [54]. To help identify and resolve issues before they cause any downtime or other inconveniences, monitoring the application after deployment is critical. Key aspects of monitoring include performance in which resource usage is monitored, along with other metrics, to identify issues affecting speed and reliability. As its own aspects of monitoring, availability monitoring ensures accessibility to its users. By checking the uptime of infrastructure components such as servers and application, availability monitoring makes sure that technology and services are in operation and available for users to access. Further, security monitoring is implemented as an automated process [54] of collecting and analyzing signs of potential threats. Events and activities are monitored and alerted when certain anomalies are detected. Other

monitoring aspects involve logs and user experience monitoring.

Deployment without Technical Expertise

Deploying applications and configuring cloud-based services can be a complicated task, especially for non-technical people. For those who are not familiar with cloud infrastructure and deployment process, cloud deployment can be overwhelming. With the complexity of configuring and managing cloud-based services, individuals or smaller businesses which do not have a dedicated IT department may find it overwhelming [36]. As for smaller businesses and individuals, cloud service providers remove a lot of these complications by offering management and maintenance of the underlying infrastructure.

Cloud service providers proceed to develop, implement and improve their services to make them easier to use and reach a wider audience. As a non-technical user, cloud service providers have user-friendly interfaces that aid non-technical users in the deployment process [2]. With an intuitive interface, the deployment process offered by a graphical representation and instructions helps users understand what to do. The interfaces are simple with intuitive design making the layout and its content not difficult to navigate. Deployment offered by cloud service providers is often well documented and outlines the deployment process by a step-by-step procedure. The documentation often includes images and examples, making it easier to follow and perform certain tasks. Along with the detailed documentation, many cloud service providers offer tutorials as part of their get started programs. To troubleshoot issues, users may use the provided documentation and tutorials, and as an option use support services such as email or phone. Not all cloud service providers have support services, but those who do can offer a more specific guidance to help identify and fix problems. In addition to the Graphical User Interface (GUI) provided by the cloud service providers, some cloud service providers also offer CLI tools. For more advanced users, CLI allows scripts and automation which is often needed for more complex deployments [51].

Customers or users of cloud service providers choose the provider based on factors such as cost, reliability and features. Flexibility in terms of cloud service providers refers to the ability to choose from multiple cloud service providers. In a flexible approach, selecting a provider that does not lock the user and provides vendor independence is beneficial [42]. Furthermore, many cloud service providers offer different features and as a customer, these specific features or strengths provided by each of the providers should be highly considered.

As part of many cloud service providers and deployment platforms the implementation of Continuous Integration and Continuous Deployment (CI/CD) [58] can help deploy applications more easily and reliably. Implementing (CI/CD) pipelines to automate deployment will remove the technical aspect of deployment and human interaction. As a key benefit, (CI/CD) pipelines remove technical details of deploying an application which means non-technical people such as project managers, analytics etc. can deploy without the underlying technical details of the deployment process. As a key component of DevOps, (CI/CD) pipelines help deploy and release new versions but also improve the overall quality and efficiency of the deployment process.

2.4.4 DevOps

Application deployment and monitoring is an essential element in DevOps which encompasses the combination of software development and operations. To help organizations deliver software and services with higher quality and reliability, DevOps aims to emphasize the collaboration and communication between development teams and IT operations [11]. With the goal of automating the processes and creating a more efficient way of developing software, DevOps shifts from traditional software delivery by encouraging an integrated and collaborative approach [25]. DevOps focuses on CI and CD which means that with a central code repository, developers continuously integrate their code which is automatically tested and deployed to production.

As a critical part of DevOps, the software development lifecycle (SDLC) lies in focus where DevOps aims to streamline and optimize its process to facilitate efficiency in terms of application development and delivery [32]. DevOps practices play a key role in CI/CD which relates to many of the stages in SDLC. By integrating development, testing, and deployment processes, DevOps aims to optimize the SDLC to enable faster, more efficient, and higher quality software development and delivery. Collaboration and communication between development and operations teams throughout the SDLC are also emphasized, enabling teams to work together more efficiently and effectively.

Software Development Lifecycle

The software development lifecycle or SDLC, defines the steps that are taken to build software and provides a well-structured flow that allows for high quality, well tested

and quick development. It also defines the responsibilities for team members during each step of the phase [17]. The typical SDLC stages consists of a planning stage, requirement stage, design stage, development stage and testing stage as described by [49].

The planning phase involves gathering requirements from the stakeholders and identifying the scope of the project. The project goals, timelines, resources, and risks are identified, and a feasibility study is conducted to determine the project's viability.

The analysis stage is where the detailed requirements are defined and documented in the Software Requirement Specification (SRS) [21] document. The SRS document outlines the functional and non-functional requirements of the software product, including user interface design, database design, and system performance requirements.

In the design stage, the high-level design of the software system is created based on the requirements gathered in the previous stage. Different software architectures are designed, reviewed, and evaluated based on factors such as risk assessment, robustness, design modularity, time, and cost. The best design is selected, and a detailed design is created.

The development stage the actual development of the software product begins. The software developers use different tools and programming languages to generate the code based on the design. The code is developed, tested, and reviewed to ensure that it meets the requirements outlined in the SRS document.

Lastly, in the testing stage, the developed product is tested to ensure that it meets the user's requirements outlined in the SRS document. The testing process involves identifying software defects, reporting them, tracking them, fixing them, and retesting them to ensure that the product meets high quality standards.

Using SDLC provides a clear plan and structure for the entire development process, ensuring that all necessary stages and activities are accomplished. It helps to ensure that requirements are well-defined and that the development team has a clear understanding of what needs to be built. The process also enables better collaboration between development teams and stakeholders, ensuring that everyone is aligned on project goals and timelines [17].

2.5 Deployment Platforms

2.5.1 Fly

Fly is a platform designed to provide an easy way to run full-stack applications and databases. The platform utilizes Firecracker virtualization technology to enable micro-virtual machines that are located closer to end-users around the world. Fly aims to make it easier for developers to manage complicated self-service infrastructure without abstracting away important details. The platform features a purpose-built cloud that runs physical servers in cities close to the users. It supports multiple programming languages and frameworks, such as Ruby, Laravel, Python, Go, and Dyno, as well as Docker for increased flexibility [13]. Additionally, Fly offers an automated process for creating Fly Postgres databases, complete with extensions to simplify the management process. Fly's Postgres offering includes features such as CPU and memory allocation, metrics/alerts, load balancers, and SSL for secure access.

Fly offers extensive documentation regarding their features and services. The documentation involves everything from how to start a project and working with application within Fly, to pricing and support. Guides for language and frameworks are also provided such as running a Node application, Go application, Python application and many others. These guides are listed based on their extensive documentation and are categorized as "Comprehensive Guides" and "Starter Guides". Other guides such as production, custom domains, deploying with GitHub actions, running multiple processes, UDP services and others are also provided in the documentation of Fly [13].

2.5.2 Railway

Railway is a platform as a service (PaaS) tool designed to make app development more accessible for developers. Railway allows developers to choose their own preferred database, such as MySQL, Redis, or Postgres, and offers a variety of templates for building apps from scratch. To deploy an app on Railway, all you need is a GitHub account. The interface is straightforward and allows developers to add database tables and make queries using a web form. Railway also offers features like autoscaling, which scales the application based on user demand, and real-time metrics for debugging issues [43]. Unlike some other PaaS providers, Railway allows

apps to run indefinitely, either using a Procfile or by deploying a Docker file.

Railway offers detailed documentation on every available feature and service offered by Railway. This documentation involves development of projects, services, variables, environments and development using CLI. Further documentation involves deployment where commands, networking, integrations, and Dockerfiles are displayed. The documentation also includes diagnostics, the use of different databases, troubleshooting and references such as pricing and plans [43].

2.5.3 Render

Render provides all the necessary features and functionalities that a backend application needs, providing fast and secure development, deployment, and hosting services. Render comes with SSL certificates and network configurations, enhancing the security of a deployed applications. Containers, API's, Web Services, Static Sites, and others can be hosted using Render where the feature of auto-deploy from Git is offered to automatically deploy changes in the remote repository [47]. Render offers several features, including DDoS attack protection, SSD storage, database experience, and instant deployment. Its PostgreSQL database provides easy configuration and management with daily database backup for at least 7 days.

Render offers a high level of flexibility through its various features and services. For example, Render allows responses to take up to 100 minutes for HTTP requests. In terms of performance and reliability, the focus is shown in several of its features and services. One of these features is HTTP requests where Render serves all requests over HTTP/2 (and HTTP/3 where available), which reduces page load times and minimizes simultaneous connections to your Render apps [47].

With documentation on the use of the platform, users can find information about the different services, custom domains, integrations and more. All documentation includes images and steps regarding certain applications. Render also supports private assistance beyond community support.

2.6 Related Work

Along with the increased popularity of cloud computing, cloud service providers have emerged as a competitive participant to the market. As competitors, they all

strive to be the best regarding price, including discounts and promotions as well as pricing models. Different features also differentiate cloud service providers where specialized or unique services are offered along with a more user-friendly interface. Other features regarding security also differentiate cloud service providers where for example encryption and access controls are offered. Other competing factors include reliability, performance and customer support. With the amount of available cloud service providers and more to come, the choice of which one is difficult.

Lang [30] conducted a Delphi study to identify and rank Quality of Service (QoS) attributes used by customers within a cloud computing environment. Based on previous studies, Lang [30] identified four changes within the cloud market. These include an increase in the importance of data protection, cloud customers continue to seek value co-creation with CSP, a decrease in the possibility of CSP opportunistic behaviour and the continuation of product uncertainty as a major problem during CSP selection. Identifying relevant QoS attributes is difficult, and to choose the correct CSP, both technical and managerial QoS attributes are considered. The study approaches the identification of QoS attributes based on earlier research and their own panel of professionals with experience in the cloud computing field.

The earlier studies included 4 managerial QoS attributes and 21 technical QoS attributes by Saripalli and Pingali [50]. Repschlaeger [48] identified 8 managerial QoS attributes along with a growing number related to data protection issues, and 13 technical QoS attributes.

Lang's [30] panel selection involved the recruitment of professionals with significant work experience in the cloud computing field. The chosen professionals were asked predefined questions regarding their cloud experience and use of cloud deployment and cloud delivery methods. Novice persons, private and hybrid cloud users and cloud service users who use cloud services less frequently than once a day were excluded from the panel selection.

The exploratory interview phase brought 31 unranked QoS attributes for CSP selection in combination with the earlier studies. 14 new QoS attributes were identified by the panel as not mentioned in previous studies and it is suggested that new QoS attributes have appeared during the past few years, mainly managerial QoS attributes. As the first major topic, the new QoS attributes involves reliable CSP with good financial performance to assure long-term relationships. As part of the long-term relationships, lock-in effects come as a second major topic. Flexibility in terms of exit strategies and standardized operating environment as changes in business strategies may frequently occur are highly requested as a result. The third major

finding of new QoS is directed against data protection capabilities as laws force cloud customers to take responsibility for the cloud services in use and data transferred.

Based on the opinion of the professionals with experience in cloud computing, the top 5 QoS attributes that were repeatedly ranked highest among the professionals were functionality, legal compliance, contract, geolocation of server and flexibility. The study interpreted these attributes as a universal indicator of the most important QoS attributes during CSP selection decisions. Further ranking and classification based on the interviews, the QoS attributes were classified based on technical and managerial QoS attributes. The technical QoS consisted of functionality, flexibility, integration and control which relates to the operational aspects of the cloud service. The managerial QoS attributes consisted of contract, legal compliance, geolocation of services, transparency of activities, certification, monitoring, test of solution and support. Lang [30] states that the most important QoS attributes were ranked by also considering dependencies between the attributes.

In conclusion, their results show that cloud customers require a variety of managerial QoS attributes during CSP selection. Further, based on previous studies, their study aligned with the following four key changes within the cloud market. First, the importance of increasing data protection. Second, value co-creation pursued by the customer of cloud service providers. Third, the possibility of a decrease in CSP's opportunistic behavior and lastly, product uncertainty.

Containerization holds a big role in continuous deployment (CD), and in a paper published by Zhang [59], the study on CD workflows caused by containerization is presented. The paper investigates Docker-enabled CD workflows by collecting and combining lessons learned and offering data-driven evidence from different CD implementations. With data collection from GitHub and Docker Hub, the paper addresses the motivations and differential benefits of Docker-enabled workflows.

Using an online form, the study conducted a survey of developers who had source code repositories on GitHub. Based on 168 responses, common CD implementations involved pushing images to Docker Hub while other projects used their own scripts to deploy images. They also found two prominent Docker image deployment workflows. A Docker Hub auto-builds Workflow where the registry itself builds the image automatically when GitHub source files change, and a CI-based Workflow where CI tools build images during the build and test stage followed by publishing it to Docker Hub.

Based on the first question regarding motivation, the survey found that 60 out of

140 respondents valued automatic deployment instead of manual deployment as the motivation of CD workflows. Further, 27 respondents said that their motivation for using CD workflows was that it gave them smoother and easier deployments. In a question about current Docker workflow, the developers reported using either Docker Hub auto builds or CI-based workflows. Using Docker Hub workflow for automated builds, the GitHub source files are automatically built into a Docker image. The new build, triggered by new code that is pushed, produces a Docker image that is further pushed to Docker Hub. Respondents of Docker Hub workflows specified advantages of Docker Hub workflows to reduce the time spent on setting it up and to deploy more frequently. As another reason, respondents said that it is free and ready to work with GitHub projects, and that it is easy to share with Docker users.

2.7 Chapter Summary

This chapter explained the Huldra framework and its features and how it compares to other online survey platforms. We also examined the technology which Huldra is built on before going into the concept of Cloud and other technologies such as virtualization and containerization. In addition, we looked at related work that identified QoS attributes when choosing a CSP and we looked at a study of CD using containerization.

Chapter 3

Methodology

3.1 Proposed Approach

This thesis aims to explain the investigation of cloud service providers with focus on deployment platforms in terms of deployment accessibility for non-technical people. Existing research shows how important cloud computing have become and its relevance for many fields. As the amount of CPS's are growing, choosing the correct CSP can be difficult. New QoS attributes show that data protection and long-term relationships have become important in the choice of CSP. Many CSP's that provide deployment platforms have become easier to use due to their improved tools and GUI for deployment which allows less technical knowledge to deploy an application.

Depending on the needs, size and requirements of a project, it is essential to understand what is offered by the cloud service provider in terms of features. With many different cloud service providers and platforms to deploy on, the ability to choose which type of cloud service provider should not be based on the technical aspect but rather the performance and features proved by the platform. To assist the deployment process and create an isolated environment, the use of container images limits the technical issues associated with deployment on different platforms as they may differentiate in operating systems or software configurations. This can cause issues for the deployed application as the environment it is deployed in can vary.

The implementation and design encompass the development of Dockerfiles for production and a development environment. It also includes the development of Docker Compose files for both environments to facilitate further development and

easier configuration of services within an application. As a tool for defining and running multi-container Docker applications, Docker Compose allows users to define the services and dependencies of an application in a single file. Using the specified Docker Compose files for a specific environment, the users can define new services and its connection between them as a single application. For example, creating a storage system or database for the application which would be defined in the Docker Compose file. As a higher-level abstraction tool, Docker Compose is implemented to bring simplicity to the defining of the environments using YAML. As a YAML file, the environment can be effortlessly launched by other people using Docker Compose. Docker Compose replaces the Docker run commands which are executed repeatedly when rebuilding and redeploying containers, for example when changing code for an application in a development environment.

Docker was chosen as the containerization platform due to its popularity and support along with Docker Compose for defining multi-container applications. For better control and security, the containerized solution was developed and tested with OsloMet ALTO Cloud before it was further tested and deployed using different deployment platforms provided by CSP. The development and testing was utilized on the OsloMet ALTO cloud using a virtual machine including the following specifications:

- Ubuntu 18.04 instance
- 1 virtual Central Processing Unit (CPU)
- 2 gigabytes of Random Access Memory (RAM)
- 20 gigabytes of disk space

The GitHub repository was cloned and added to the virtual machine using the "git clone" command. The files for Docker, Docker Compose and Nginx was then added at the root of the application directory. Using Docker commands, the application was then built as an image and started as a container using the Dockerfiles. During build time and runtime the logs was examined, and for any occurrence of errors the application was shutdown and reconfigured based on the error messages. Once the application was successfully deployed using the virtual machine and Docker, the application was tested in the browser using an demo survey to verify that it worked correctly.

As many deployment platforms differ in terms of the deployment process and technical control, the development using the ALTO Cloud provided full access to

files, VM's, containers, images, logs, configurations and other management of the container. This allowed for better trouble shooting during the development of the Docker, Docker Compose files and the containerized deployment of the application. The application was then deployed on different platforms including Render, Railway and Fly where the use of their web-based deployment process was used. Fly does not support any form of GUI when deploying on their platform and must instead use command line interface to interact with their platform. This requires some technical knowledge over the use of the platforms GUI.

3.2 Plan for Subjective Study

The motivation of this user study is to gather feedback and insights from the participants about the experience of deploying the Huldra application using Render. The study consists of three parts where the first part involves a pre-questionnaire to gather information about the participants followed by a guided deployment, and ends with a post-questionnaire with questions based on System Usability Scale (SUS) [8] and a QoE question.

3.2.1 Method

To understand the difficulty or simplicity of deploying an application to the web using CSP's, we proposed a user study to gather data from participants with varying level expertise in application deployment and with technical expertise in general.

3.2.2 Participant Recruitment

Application deployment is not common knowledge and is not a task that is done on a regular basis. Therefore, the survey was distributed to as many people as possible to ensure a diverse range of participants with varying levels of technical expertise and to maximise the number of participants. The study were conducted online using Google Forms were participants completed the survey at their own time using their web browser.

3.2.3 Study Material

The subjective user study begins by giving the participants an introduction of the study containing the purpose and contents. Followed by the introduction, the participants are asked to answer some questions before starting the deployment.

Pre Questionnaire

The pre-questionnaire asks the participants to select their age in range of:

- less than 18
- 18 to 29
- 30 to 39
- 40 to 49
- 50 or above

Followed by the age selection, the participants were asked to select their gender with the options of choosing:

- Female
- Male
- Non-binary
- Other

As part of the study, the participants were asked to range their familiarity with:

- Computer science
- Web application deployment
- Docker virtualization

These familiarity's were ranked on a scale from 1, indicating very low familiarity, to 5 indicating very high familiarity. Lastly in the pre-questionnaire, the participants can enter their profession as an optional text input.

Step 1

After the pre-questionnaire, the guided deployment begins with logging in to Render. Two bullet points are provided as follows:

- Go to <https://render.com/>
- Enter e-mail (huldra@simula.no) and password (*****) and click Sign In

In this step there are also provided a screenshot of the sign in page on Render. The participants enters the local time at completion of this step and also ranks the difficulty level of this step by using a likert scale from 1 to 5. The participants can also choose to give a comment on this step.

Step 2

Step 2 of the guide includes the creation of a Web Service where the participants are given four bullet points and two screenshots. The provided bullet points include:

- Click New from the top right, and select Web Service
- After clicking Web Service, navigate to Public Git repository at the bottom of the next page
- Enter <https://github.com/BjorgeO/docker-deploy> in the input field as the public GitHub repository address
- Click Continue

The first bullet point is also provided as a screenshot, showing the participant where to click and choose the Web Service. Another screenshot is also provided, showing the Render page where the GitHub repository address is added. As the previous step and further steps of the guide, we asked the participants to enter their local time when completing this step and also choose the difficulty level of this step using a likert scale from 1 to 5.

Step 3

Followed by step 2 of the guide, step 3 included the Web Service setup. Five bullet points are provided including:

- Choose a Name for the web service (you can pick any name), and enter it in the relevant input field
- Choose a Region for the web service (preferably Frankfurt (EU Central)) using the relevant dropdown menu
- Set the Branch to be main

- Leave the Root Directory blank
- Set the Runtime to be Docker

As Render automatically sets the runtime to Docker due to the detection of a Dockerfile in the GitHub repository and branch is set to main by default, only the name and region is changed by the participant. Branch is by default set to "main" and the root directory should remain blank. A screenshot is also provided, showing the correct input in the different fields.

Step 4

Step 4 of the guide consists of five bullet points for setting advanced options of the Web Service. These bullet points includes:

- Navigate to the bottom of the page, and click Advanced
- Inside the advanced options, click on Add Secret File
- Enter .env as Filename
- Copy and paste the following lines into File contents
 - `REACT_APP_FIREBASE_API_KEY=*****`
 - `REACT_APP_FIREBASE_AUTH_DOMAIN=*****`
 - `REACT_APP_FIREBASE_PROJECT_ID=*****`
 - `REACT_APP_FIREBASE_STORAGE_BUCKET=*****`
 - `REACT_APP_FIREBASE_MESSAGING_SENDER_ID=*****`
 - `REACT_APP_FIREBASE_APP_ID=*****`
 - `REACT_APP_FIREBASE_ROOT_DIRECTORY=*****`
- Click Save, this will close the Secret File window.

This step asks the participants to navigate to the bottom of the page and clicking the "Advanced" button that opens further configuration of the Web Service. The participant then gets asked to click on "Add Secret File" that opens a popup window where the user inputs filename and file contents. The filename and file contents are provided in the guide and as a bullet points we ask the participant to copy and paste the lines from the guide into the file contents of the popup window. We also provided a screenshot of the popup window with the correct filename and file contents. The last bullet points asks the participant to save and close the "Secret File" window.

Step 5

Step 5 of the guide consists of three bullet points:

- Navigate to the bottom of the page, and click Create Web Service
- The deployment takes a few minutes
- The web app is deployed when the In progress indicator changes to Live

Step 6

The last bullet point is also visible in the provided screenshot of this step. As the next step in the guide, we ask the participant to go to the deployed application to verify that it is working. Two bullet points are provided to help the user find the URL of the deployed application and to inform the participant to confirm the deployed application by provided screenshot:

- The URL for the deployed app is displayed at the top left, below the web service name (looks like: `https://<web service name>-<random string>.onrender.com`, for example `https://mywebservice-pa5m.onrender.com/`)
- Click on the URL, and check if the Huldra app is properly deployed (it is enough if you confirm that you see the Huldra homepage as indicated below)

Step 7

As the last step of the guide we ask the participant to delete the Web Service. Three bullet points are provided with a screenshot. These bullet points include:

- Go to the Settings tab on the left side
- Scroll all the way down in the page, and click Delete Web Service
- Type or copy-paste the red text into the input field and click Delete Web Service

This step concludes the guided deployment and the participants ends the user study by answering questions related to SUS.

Post Questionnaire

The post-questionnaire uses the SUS [8] to measure the usability of Render and includes a QoE question. The questions were presented with a likert scale were the participants could choose from 1 as strongly disagree to 5 as strongly agree. Eleven questions were presented as follows:

1. I think that I would like to use this system frequently.

2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.
11. My overall quality of experience with the deployment procedure was high.

The participants could also give general comments at the end.

3.3 Chapter Summary

We have in this chapter looked at the aim of thesis, investigating how difficult it is for people with less technical expertise to deploy applications on different CSP's. We also looked at the approach of developing a containerized solution for Huldra that by using Docker, should remove issues related to application deployment on deployment platforms. This study tests the deployment process using a specific cloud service provider and then tests the same process on different providers.

We also looked at the plan of the subjective user study, containing the deployment steps, pre and post-questionnaires. The user study is conducted to understand the difficulties of deploying an application using CSP's, and are conducted to estimate the deployment process based on different backgrounds and familiarity's of the participants.

Chapter 4

Implementation

The following chapter includes the containerization of the application Huldra and the deployment on different cloud service platforms including a deployment on OsloMet ALTO Cloud.

Huldra uses Firebase which provides a set of backend services used serve a specific survey. For an application to be connected to Firebase, certain connection parameters must be set in order for it to communicate with the Firebase project. These parameters include an API key, authentication domain, project ID, storage bucket, messaging sender ID, application ID and root directory. For security reasons, these parameters are not added in code, but are passed through either as an environment file or passed as environment variables through the CSP interface. It is arguably safer to provide the Firebase connection parameters as environmental variables within the instance the service is running, but as a containerized solution the Docker container is isolated from its host operating system and is not able to access these variables. As a solution, the environmental variable values can be passed to the Dockerfile at build-time.

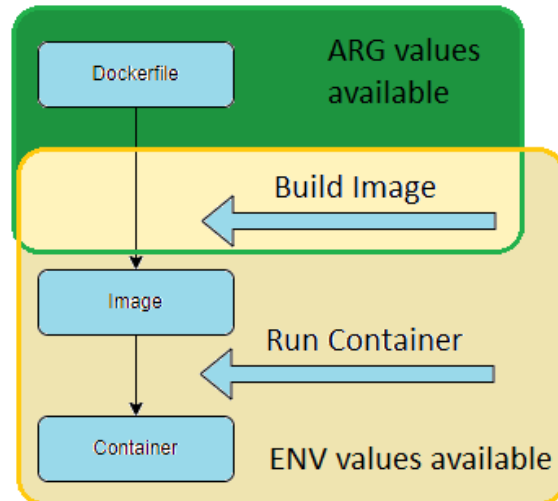


Figure 4.1: ARG and ENV availability

With the use of ARG instruction and ENV instruction, the environmental variables defined at the host OS can be passed through to the Docker container. At build-time, the variables and its values are defined using ARG. These variables are not accessible by a running container and must be defined within a ENV instruction. The ENV instruction sets the environment variable in the Docker image which will be available in the running Docker container. By passing the host OS environment variables as ARG and then using the ARG variable in the specified ENV instructions, the environment variables and their values specified in the interface of the deployment platform are passed through to the container.

Using ARG and ENV instructions as build-time variables for passing data makes it visible to any user of the image. As an option, secrets can be added in the Dockerfile as secret mounts. Although this option also can result in data being stored in the image, it is considered as a better practise than adding data in plain sight.

4.1 Containerization of Huldra

For further development of the framework and as an option for the users of Huldra, a Docker image is developed to support a containerized deployment. With the implementation of Docker image and the containerization of the application, the framework can operate on many different CSP platforms including on-premises servers and other platforms supporting containerization. By using containerization to deploy the framework the code does not need to change along with the required

services for deployment. With an image, the created containers will work on multiple operating systems where Docker is more versatile in deployment.

For testing and development of the Docker configurations, a virtual machine (VM) was created in ALTO Cloud at OsloMet for fast and easy deployment. The VM was created to test a containerized environment using different Dockerfile configurations for deployment of the Huldra framework. Since Huldra uses React, different methods of deployment were tested. As a possibility, a development environment along with production environment can be containerized. A development environment is not available to the end-user and is created for developers to ensure that the application functions as intended before moving it to the production environment. With a containerized development environment, the entire development stack, including the operating system, runtime and dependencies are wrapped into containers which provides containerized benefits. Since developers of the Huldra framework do local testing running ReactJS in localhost, a containerized development environment may not be needed but is added as an option. Running the React project locally also helps detecting and fixing problems as React provides warnings and tools for eliminating potential bugs. These development features increase the project size and as a result slow down the app. To prevent a slow running app in production, a production-ready version is created where the source code is compiled, optimized and bundled into a set of static files which can be served via a web server. For production, a multi-stage docker build is used to break the steps in building a Docker image into multiple stages.

4.1.1 Multi-stage Builds

Multi-stage builds help optimize Dockerfiles both in readability and maintenance. It allows the use of multiple images within the same Dockerfile to build the finishing image. Keeping the image as small as possible or optimized is desirable as they decrease potential security vulnerabilities and surface area of attack. Only what the application needs should be defined to run in a production environment [15].

4.1.2 Dockerfile Production Environment

Multi-stage builds eliminate the use of multiple single-stage builds which is implemented through the use of multiple Docker files. For example, the following figure displays a multi-stage build Dockerfile which is used to build a React project

for a production environment and served by a Nginx webserver.

```
GNU nano 2.9.3 Dockerfile.prod Modified
FROM node:alpine AS build
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run build

FROM nginx
COPY nginx.conf /etc/nginx/conf.d/default.conf
COPY --from=build /app/build /usr/share/nginx/html
```

Figure 4.2: Dockerfile for production environment

Figure 4.2 displays a multi-stage Dockerfile of building and serving a React project by Nginx. This file is used to serve the Huldra framework as a containerized application. With the command "npm run build", the executed command returns a couple of files which then needs to be served in a Docker container. A Nginx server must also be created to serve those files. This is the reason for multi-stage builds as it allows to run the FROM command multiple times, defining the base image of each build stage of the Dockerfile. By default, the stages are not named, but can be named as shown in figure 4.2 by adding an "AS <name>" to the FROM instruction. The first stage of the multi-stage docker build includes the installation of node, copying the "package.json" file, installation of dependencies, copying source code and executing the command "npm run build". This stage returns the static files in a build folder which is then used in the next stage, stage 2. Stage 2 starts by installing the official Nginx image and copying a Nginx configuration file named "try_files" to the container which holds a new parameter. Lastly, the last line in the Dockerfile 4.2 copies the build folder from the first stage into the directory of Nginx. Nginx will then serve the copied files in the HTML directory as a web server.

4.1.3 Try Files for Nginx

The "try_files" is a directive used in the configuration file of the Nginx web server. It is used to define a set of files or paths that Nginx should try to serve in case the requested file or resource is not available or cannot be found.

```
GNU nano 2.9.3      nginx.conf      Modified
server {
  location / {
    root    /usr/share/nginx/html;
    index  index.html index.htm;
    try_files $uri /index.html;
  }

  error_page 500 502 503 504 /50x.html;
  location = /50x.html {
    root    /usr/share/nginx/html;
  }
}
```

Figure 4.3: Try_files

Figure 4.3 displays the Nginx configuration file which allows users to refresh the website. When the React application loads, the routes are handled on the frontend by the “react-router”. For example, when a user is on a page routed at “http://huldra.com” and navigates to “http://huldra.com/registration”, the route change is handled in the browser itself. When the user refreshes the URL at “http://huldra.com/registration” the request for that URL is sent to Nginx where the specific route does not exist. As a result, the user gets a 404-error message which indicates that the requested page is not available or does not exist. To prevent the error, the configuration surrounded in red in figure 4.3 is added. The “try_files” directive is used to specify a set of files to look for when a URL is requested. In sequence, these files are to be tried when a client request is received. Further, the “uri” variable contains the requested URI (Uniform Resource Identifier) for a particular request. It is a string that represents the path component of the requested URL. In action, the user first makes a request to the web server which receives the requested URI. The “try_files” directive in Nginx specifies a list of files to look for in order to serve the requested content. If the requested URI is not found, Nginx sends “index.html” back to the browser. With this Nginx configuration, the users can refresh the webpage and get the correct requested URL back.

4.1.4 Dockerfile Development Environment

Together with the production environment, a containerized development environment is also created. As shown in figure 4.2, the filename of the Dockerfile is “Dockerfile.prod”. Two files are made for each environment where the Dockerfile for the

development environment is called "Dockerfile.dev". The development Dockerfile holds different configuration relative to the production file as it does not require to be built.

```
GNU nano 2.9.3 Dockerfile.dev Modified
FROM node:alpine
WORKDIR /app
COPY package.json .
RUN npm install
COPY . ./
EXPOSE 3000
CMD ["npm", "start"]
```

Figure 4.4: Dockerfile for development environment

Figure 4.4 displays the Dockerfile, called "Dockerfile.dev", used for a development environment. Like the Dockerfile for production, the file starts with the "FROM" command to initialize the base image for subsequent instructions. Base images can be official Docker images or custom images. In figure 3 the Alpine Linux distribution is specified to build the Docker image. When not defining any specific image, the latest version of node will be installed. The command "node" is an alias to "node:latest" meaning that when only specifying "node", the latest version will be installed. For the task of creating a container used for React, a smaller image is used. Depending on the image, more or less dependencies are installed along with a smaller vulnerability surface. With a complete package repository, Alpine is significantly smaller than the default node image where a lot of libraries are removed which are not necessary for this solution. A smaller image gives benefits such as less memory, better performance, security and maintainability. The second line in the "Dockerfile.dev" file 4.4, the working directory of the Docker container is set by using the "WORKDIR" command. Any "RUN", "CMD", "ADD", "COPY" or "ENTRYPOINT" command used inside Dockerfile succeeding this line will be executed in the specified working directory. The working directory is specified as "app", but should be specified to match the host folder structure. The following command copies the "package.json" file into the container and further runs npm to install the required dependencies. To prevent recopying and reinstalling dependencies, the "package.json" file and the installation of dependencies is done before the rest of the react application is copied over. All commands is executed within the working directory, specified by the dotted path. Finally the CMD command is executed to run the start script in the "package.json" file. The docker

CMD command specifies instructions to be executed when a Docker container starts, in this case “npm start” as shown in figure 4.4.

4.1.5 Docker Compose files

With the Docker files for development and production, three docker-compose files are created to support, maintain and scale the application when needed. Although the react project only consists of one container, docker-compose allows to change the configuration easily and add more containers if necessary. It also provides the organization of volumes and it can be used to look at current configuration of the containers.

Instead of a single docker-compose file, three files are created since there are multiple environments. A single docker-compose file is usually used per service as multiple services should not be dependent on different docker-compose files. For example, in a scenario where multiple services are using a common database. If one of the services causes the database to go down, then other services that rely on the database are also affected. The same problem occurs when one service overloads the database so that it goes down or reduces the performance. With those conditions in mind, a single docker-compose file is created per environment. The third docker-compose file is created to share configuration between the two docker-compose files and works as a base file for those services. The base docker-compose file is called “docker-compose.yml” and does not hold that much configuration other than the commonalities between the production and development environment.

A screenshot of a terminal window showing the content of a base Docker Compose file named docker-compose.yml. The terminal title is "GNU nano 2.9.3 docker-compose.yml". The file content is:

```
version: "3"
services:
  react-app:
    build:
      context: .
```

Figure 4.5: Base Docker Compose file

Figure 4.5 displays the base docker-compose file which holds the shared configuration between the two docker-compose files for a production and development environment. The file starts by specifying the version of the Compose file format. Compose file format version depends on the features needed for the Docker Compose

files. The base Docker Compose file and the files for production and development environment does not require any special features and are the reason for the chosen version. The version can be changed easily and can be done so accordingly depending on further development of the containerized application. In the next line the service is specified. The service along with Docker, represents the container. This service is named react-app as shown in the third line in figure 4.5 as the service will hold the react application. Within the specified service name, the configuration for that service is specified. As this solution uses a custom image, the build parameter is added with a context. The build parameter specifies that “react-app” service should be built using a Dockerfile located in the current directory. The context parameter also specifies the build context in which the directory contains the files that will be used to build the image. In this case, the same directory is specified. Both Docker files for production and development are located at the same directory as the path are defined as “.”.



```
GNU nano 2.9.3 docker-compose.prod.yml
version: "3"
services:
  react-app:
    image: app-prod-i
    build:
      dockerfile: Dockerfile.prod
    container_name: app-prod-c
    ports:
      - "8080:80"
    environment:
      - NODE_ENV=production
```

Figure 4.6: Docker Compose production file

Like in the base Docker Compose file, the services are defined in the production Docker Compose file. Along with the specified version, the service name must be declared the same as in the base Docker Compose file 4.5. Further, with the image property the specified name of the image which is used to create the container is defined. The image is declared as “app-prod-i” shown in figure 4.6 as it holds the production environment instruction for a container of the application and are specified with an “i” as in “image”. Along with specifying the path to the “Dockerfile.prod” file, the name must also be specified. This is done by the property “dockerfile” which when combined with base Docker Compose file “docker-compose.yml” 4.5, builds the image using the context of the base Docker Compose file and the Docker File “Dockerfile.prod”.

Further, within the created react-app service, a name is given to the container which is created from the image. The container is given a name by the "container_name" instruction where the name is specified as "app-prod-c", "c" for container. Used to map the exposed ports of a container to the host machine's ports, the "ports" property is used. When a container is started, Docker creates a network interface to allow communication between the container and the host machine. These interfaces are isolated by default from the host machine and need port mapping to expose the container's services to the host machine or to the outside world. The ports specified are "8080:80" which means that the container's port 80 is mapped to the host machine's port at 8080.

Lastly, in figure 4.6, the environment property is defined. Used to set environment variables for a container, the environment property defined at the service level implies that the environment variables are applied to all containers created from that service. In this case, the environment variables are only applied to that specific container. The only environment variable specified in "docker-compose.prod" is the variable "NODE_ENV". It defines the environment in which the NodeJS application is running, in this case the production environment. The NODE_ENV environment variable is commonly used in NodeJS applications to enable or disable certain features based on the environment. By setting NODE_ENV to production, features like debugging or logging is disabled, improving performance and reduces security risks.

```
docker-compose.dev.yml Modified
version: "3"
services:
  react-app:
    image: app-dev-i
    build:
      dockerfile: Dockerfile.dev
    container_name: app-dev-c
    volumes:
      - ./:/app
      - node_modules:/app/node_modules
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
volumes:
  node_modules:
```

Figure 4.7: Docker Compose development file

Similar to the Docker Compose production file, the Docker Compose development file holds the configuration of a development environment. With most of the same configuration settings as the Docker Compose production file, the Docker Compose development file mainly focuses on the Docker volumes. Docker volume is an independent file system that is entirely managed by Docker. The volumes keyword defines a persistent data storage for a container and stores data outside of a container's file system. As for a development environment, persistent data storage allows data to persist even if the container is deleted or recreated.

The specified volumes in figure 4.7 show that the host directory is `./` and that the Docker directory is `"/app"`. Whenever anything is changed inside the client directory on the host machine, these changes will be replicated in the Docker container. Another volume is also defined as a named volume defined at the bottom of the Docker Compose file 4.7. The named volume references the container directory to a name which is applied with `"node_modules"`. This prevents the node modules folder from the host environment to overwrite the node modules folder within the container. Used to share data between containers or to store data that needs to persist across containers restarts, the named volume prevents issues for node modules if the container architecture is different than the host operating system. For example, different dependencies will be installed regarding NPM on a container that uses MacOS versus a container that uses Ubuntu. Node modules are defined outside of the services with their own volumes keyword and mounted to the app directory inside the react-app container. Any data written to this directory by the react-app container will be persisted on the host system even if the container is deleted or reacted.

In difference from the Docker Compose production file the ports, image, container name and the node environment variable properties are set to different values. Ports are set to `"3000"` as the default port of a ReactJS application in development mode runs on port `"3000"`. Image and container name are changed to development rather than production, and the environment variable `"NODE_ENV"` are set to development.

4.2 Alternative Deployments

4.2.1 Huldra Deployment using ALTO Cloud

Production Environment

Tested and built on ALTO Cloud the Docker files including “Dockerfile.prod”, “Dockerfile.dev”, “docker-compose.yaml”, “docker-compose.dev.yaml” and “docker-compose.prod.yaml” are executed on a virtual machine to simulate a real deployment of each environment. The virtual machine is created using Ubuntu 18.04 instance with 1 virtual Central Processing Unit (CPU), 2 gigabytes of Random Access Memory (RAM) and 20 gigabytes of disk space.

```
root@master: /home/ubuntu/huldra# docker compose -f docker-compose.yaml -f docker-compose.prod.yaml up
[+] Running 0/1
# react-app Warning
[+] Building 321.9s (15/15) FINISHED
=> [internal] load build definition from Dockerfile.prod
=> => transferring dockerfile: 275B
=> [internal] load .dockerignore
```

Figure 4.8: Docker Compose start command for production environment

Figure 4.8 displays the execution of the Docker Compose command used to build and start a production environment. Surrounded in a red box, the Docker Compose command uses two Docker Compose files. The base Docker Compose file which holds similar configuration of both environments and the Docker Compose production file. The command “docker compose up” is used to start and run Docker containers as defined in the Docker Compose files. By reading the configuration defined in these files, Docker creates and starts the containers specified in the file. If the containers do not already exist, Docker builds them on the configuration defined for each service. It is also visible as the last line of figure 4.8 that the “dockerignore” file is loaded. With the “dockerignore” file the build process gets optimized as it excludes unnecessary files and directories from the build context. It reduces the size of the build context and speeds up the build process.

Executing the “docker-compose up” command for the production environment outputs the following messages displayed in figure 4.9. The output messages in figure 4.9 show that “Dockerfile.prod” has been executed and are following the instruction defined in the file to create its container. The Dockerfile for each environment is specified in its matching Docker Compose file. Each line in figure 4.9 represents the commands used in the “Docker.prod” file and are executed

```

=> [build 1/6] FROM docker.io/library/node:alpine@sha256:d3a3d691797cef0b70e361788a2aeb9dd7925112996719628d4bcd4 0.25
=> => resolve docker.io/library/node:alpine@sha256:d3a3d691797cef0b70e361788a2aeb9dd7925112996719628d4bcd4d2708 0.25
=> [stage-1 1/3] FROM docker.io/library/nginx@sha256:aa8afebbb3cfa473899a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5 0.05
=> [internal] load build context 0.45
=> => transferring context: 2.05MB 0.15
=> CACHED [build 2/6] WORKDIR /app 0.05
=> CACHED [build 3/6] COPY package.json 0.05
=> CACHED [build 4/6] RUN npm install --legacy-peer-deps 0.05
=> [build 5/6] COPY . 0.65
=> [build 6/6] RUN npm run build 297.35
=> CACHED [stage-1 2/3] COPY nginx.conf /etc/nginx/conf.d/default.conf 0.05
=> CACHED [stage-1 3/3] COPY --from=build /app/build /usr/share/nginx/html 0.05
=> exporting to image 0.25

```

Figure 4.9: Docker Compose production environment output

systematically.

```

[*] Running 2/2
  * Network huldra_default Created 0.65
  * Container app-prod-c Created 0.45
Attaching to app-prod-c
app-prod-c | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
app-prod-c | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
app-prod-c | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
app-prod-c | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
app-prod-c | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
app-prod-c | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
app-prod-c | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
app-prod-c | /docker-entrypoint.sh: Configuration complete; ready for start up
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: using the "epoll" event method
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: nginx/1.23.3
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: OS: Linux 4.15.0-46-generic
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: start worker processes
app-prod-c | 2023/03/14 15:23:51 [notice] 1#1: start worker process 27

```

Figure 4.10: Production container successfully created

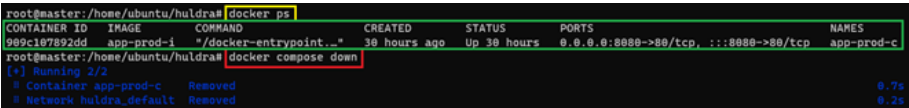
Visible in figure 4.10, the production environment is successfully created and started. The first line “Network huldra_default Created” indicates that Docker created a new network with the specified name based on the name of the directory it exists in. Since no networks were specified in the Docker Compose file, the network name is taken from the directory name where the Docker Compose file is located. The created network is isolated and only used by the containers specified in the Docker Compose file. Other containers on different networks cannot communicate with each other unless network forwarding or routing between the networks is configured.

Next, the “Container app-prod-c Created” message indicated that Docker has created a new container with the name specified in the Docker Compose file. Docker follows by attaching the console output to the terminal, indicated that it has been successfully created and are running. Further messages are logs from the container, and as the last messages visible in figure 4.10 there are two messages output as “start worker processes” and “start worker process 27”. These messages are related to the command being executed inside the container and not Docker itself. As the production environment uses Nginx, the “start worker processes” message indicates that the Nginx web server is starting and is starting worker processes to handle incoming requests.

The worker processes are a critical component of the Nginx architecture. As a high-

performance web server, Nginx uses an asynchronous event-driven architecture to improve performance and handle multiple connections simultaneously. With one master process that evaluates configuration files and maintains worker processes, Nginx is flexible and lightweight when it comes to resource usage. Assigned by the master process, an incoming client request is assigned to an available worker process which handles the request and serves the correct response. Depending on the available system resources, the number of worker processes are determined by the specified configuration file. By default, the number of worker processes are set to the number of CPU cores on the server.

Redeployment with Docker Compose. If changes to the “docker-compose” files are made, using the command “docker compose up” with the corresponding files will delete and recreate affected containers. As the “up” command is idempotent, none of the services and its configuration will be duplicated but instead overwritten or recreated. Services can be stopped by “docker-compose stop” command. This stops the services but does not delete them. The “docker-compose down” command will both stop and remove the services including its containers, networks etc. Images and volumes can also be removed by including “-volumes” or “-rmi” at the end of a “docker-compose down” command, for example “docker-compose down -volumes”.



```
root@master:/home/ubuntu/huldra# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
389c187892dd  app-prod-i    "/docker-entrypoint_..." 30 hours ago  Up 30 hours  0.0.0.0:8080->80/tcp, :::8080->80/tcp  app-prod-c
root@master:/home/ubuntu/huldra# docker compose down
[*] Running 2/2
  Container app-prod-c   Removed
  Network huldra_default Removed
```

Figure 4.11: List containers and Docker Compose down

Displayed in figure 4.11 are the container created for the production environment named “app-prod-c”. With the docker command “docker ps” as visible in figure 4.11 surrounded in yellow, all running containers are displayed. Only one container is running, which is displayed in a green box where some information about the container is also displayed. The container in figure 4.11, named “app-prod-c”, has been up for 30 hours which is when it was created. It also shows the image used to create the container named “app-prod-I”. Further, figure 4.11 displays the use of the command “docker compose down” surrounded in a red box. The output of the command is displayed in blue text, showing that both the container and the network created for the service are removed.

Development Environment

The containerized development environment is made using the same Docker Compose command to create and start the container as for the production environment. Using a different Docker Compose file, the command to start a development environment is “docker compose -f docker-compose.yml -f docker-compose.dev.yml up”. When using files together with the Docker Compose command, the option “-f” needs to be specified before each file. Further, the same base Docker Compose base file is used, but the file “docker-compose.dev.yml” is used instead.

```
root@master:/home/ubuntu/huldra# docker compose -f docker-compose.yml -f docker-compose.dev.yml up
[*] Running 2/2
[*] Network huldra_default Created
[*] Container app-dev-c Created
```

Figure 4.12: Docker Compose starting a development environment

As displayed by figure 4.12, the command and files used to create and start the development environment is surrounded in a red box. The container and network are created, marked in a yellow box in figure 4.12, where the container name is “app-dev-c” indicating that it is a development container. With different configurations than the Docker Compose production file, the development file is configured using Docker volumes. When the “docker-compose.dev.yml” is executed, a Docker volume is also created.

```
root@master:/var/lib/docker/volumes# ls
5dda17d4f38e45009d8e339452ba39a151be3f10aa7def2fb05fd58d6e3d8bc8
856b2280a6f37a252d8c237f6c470ae4bd4996852232a2a43c64f962bfe480b1
backingFsBlockDev
huldra_node_modules
metadata.db
```

Figure 4.13: Docker volume and location

The Docker volume is located at the path “/var/lib/docker/volumes/” as shown by figure 4.13 surrounded in a yellow box. By using the command “ls” to list files and directories, the Docker volume is visible named “huldra_node_modules” surrounded in a red box in figure 4.13. The default Docker volume name is based on the folder of which the Docker files and Docker Compose files are executed from. Much like the default network name, the Docker volume name becomes the folder name in addition to the specified folder of the volume.

Container Solution for Huldra

The solution for Huldra consists only of one container depending on the environment that is running. As it uses Heroku for deployment and Firebase for storage, only one container which is served and running is sufficient. If multiple containers and services were to be served for the application, Docker Compose would be an option for such an application. If Huldra had a database as part of the application, these two services would be defined and served in separate containers. In order to create these containers which holds each service, a Dockerfile is created to specify these services and create the image. Much like the single-stage Dockerfile, one would like to avoid such development structures as it inhibits control and maintenance by having multiple files for each service along with running a container one by one. To solve the problem of having multiple containers for a single service, the tool Docker Compose is available and facilitates multiple services which are running simultaneously [35].

Docker Compose is used to manage and deploy multi-container Docker applications and simplifies the deployment process by defining and running multiple containers as a single application. Not all scenarios benefit from Docker Compose, but it is useful when running an environment which consists of multiple services such as a web server, a database server or a caching server for a development environment. Other examples include test environments consisting of multiple containers where each container represents different aspects of the application. Developers can easily run automated tests against the application in these environments which helps the automation of continuous integration workflows. Further examples include the production environment which ultimately consists of multiple containers which allow the deployment and management of the entire application as a single unit [24].

4.2.2 Huldra Deployment using Render

Given as one of the cloud service provider alternatives to Heroku, Render is first tested as an alternative over Heroku and used for testing as a cloud service provider along with OsloMet ALTO Cloud. Render focuses on beating Heroku on aspects such as flexibility, performance, reliability, developer experience, pricing and customer focus. Render itself provides a comparison over Heroku where features within said aspects are listed and compared against Heroku. Render also provides a guide to migrate a Heroku application to Render, making it a clear competitor for Heroku.

At the time of developing and testing a containerized environment using Docker, Render does not support Docker Compose. The containerized deployment solution uses Docker Compose as part of the configuration and support for later services but cannot be used on Render. Instead, the individual Dockerfiles are used without the Docker Compose files.

Render offers deployment through CLI or by using their web-based interface. This deployment will be using the web-based interface.

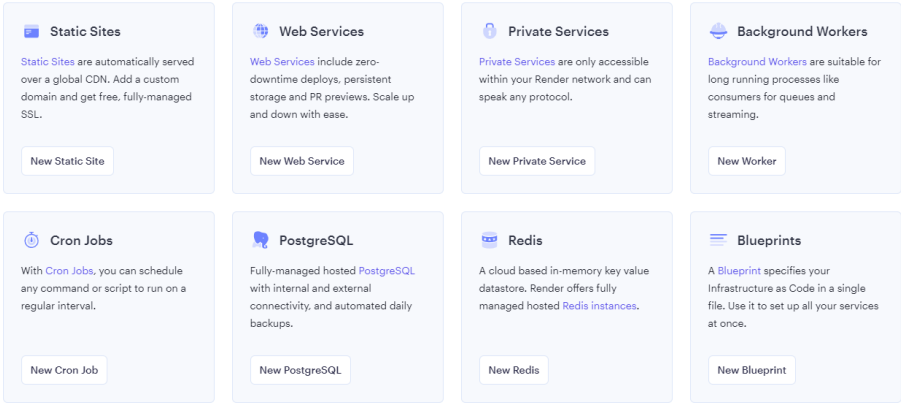


Figure 4.14: Available Render services

As displayed in figure 4.14, Render offers services including static sites, web services, private services, background workers, cron jobs, PostgreSQL, Redis and blueprints. For a containerized deployment using Docker, a web service is chosen which automatically builds and deploys the service every time code gets pushed to the selected repository. After selecting and clicking on “New Web Service”, a repository must be connected to further proceed with the deployment.

Create a new Web Service

Connect your Git repository or use an existing public repository URL.

The screenshot displays the 'Create a new Web Service' interface. It is divided into two main sections: 'Connect a repository' and 'Public Git repository'.
1. 'Connect a repository': This section features a search bar and a list of repositories. A green box highlights the 'Connect' buttons for three repositories: 'Bjorge0 / huIdra' (42 minutes ago), 'Bjorge0 / rd-test' (7 days ago), and 'Bjorge0 / docker-webapp' (a month ago). A green arrow points from the first repository to its 'Connect' button.
2. 'Public Git repository': This section includes a text input field containing the URL 'https://github.com/render-examples/express-hello-world' and a 'Continue' button. A purple box highlights the input field, and a purple arrow points down to it.
3. Account Configuration: On the right side, there is a 'GitHub' account section with a 'Configure account' link highlighted in a red box. Below it is a 'GitLab' section with a 'Connect account' link. A red arrow points up to the 'Configure account' link.

Figure 4.15: Render repository connection

Displayed in a green box in figure 4.15, the available repositories are listed based on the connect GitHub account. By connecting a GitHub account to Render, specific selected repositories or all repositories will be available for connection to a service. Repository access can be changed by navigating to account configurations surrounded in a red box named "Configure account". The link takes the user to GitHub where all or selected repositories can be chosen for Render to access. If the repository is public, the repository URL can be added in the input field at the bottom of the page marked in a purple box in figure 4.15. Using the URL of a public repository disables certain features like PR Previews and Auto-Deploy as the repository has not been configured for Render.

You are deploying a web service for [BjorgeO/huldra](#).

Name
A unique name for your web service.

Region
The region where your web service runs.

Branch
The repository branch used for your web service.

Root Directory Optional
Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.

Runtime
The runtime for your web service.

Please enter your payment information to select an instance type with higher limits.

Instance Type	RAM	CPU	Price
<input checked="" type="radio"/> Free	512 MB	0.1 CPU	\$0 / month

Figure 4.16: Render Deployment configuration

Once the correct repository is selected, a few configurations must be selected displayed in figure 4.16. First, the name of the web service is set, in this case it's called "docker deployment". Next the region is chosen where the web service is hosted. For services to communicate privately, they must be deployed in the same region. The region chosen for this web service is Frankfurt (EU Central). After selecting the region, the repository branch used for the web service is chosen. In a text field the branch is written, and in the deployment displayed by figure 4.16, the main branch is chosen. Second to last configuration of the web service is the root directory specification. The path of which the root directory is specified where Render runs all the commands. Changes made outside the specified root directory are ignored. Finally, the runtime is chosen which specifies the runtime of the web service. Depending on the selected runtime, further configuration must be made such as build command and start command. Other runtimes include Node, Python, Go, Ruby, Elixir and Rust. As a containerized deployment using Docker, the runtime Docker is chosen as displayed by figure 4.16. The figure also displays that this web service is using a free instance, limiting the resources available and performance.

As designed by Render to explore, build personal projects and get a preview of the Render developer experience, the free web service instance type automatically goes down after 15 minutes of inactivity. When new requests for the web services are detected, Render spins the service back up so it can process the requests. Render states that this can cause a response delay of up to 30 seconds for the first request that comes in after a period of inactivity. The free instance type further allows only 750 hours of uptime per month across all free web services on the account and 100 GB

of egress bandwidth for each free service. Egress bandwidth meaning the amount of traffic that gets transferred from the hosted service to external networks.

Since Render does not support Docker Compose, further configurations are made for the deployment. In the deployment page there is a dropdown menu of advanced options.

The screenshot shows a configuration form with the following fields and values:

- Health Check Path:** /healthz
- Docker Build Context Directory:** .
- Dockerfile Path:** ./Dockerfile.prod.yaml
- Docker Command:** (empty)
- Auto-Deploy:** Yes

Figure 4.17: Render Deployment advanced configuration

Figure 4.17 displays some of the advanced configuration. The only configuration used is the Dockerfile path. Since the Docker Compose files are not available for use, the individual Dockerfiles are instead defined explicitly and used to build the containers for the deployment of the application. All the Dockerfiles and Docker Compose files are located at the root directory and the path are therefore specified as “./Dockerfile.prod.yaml”. For this deployment the production environment is deployed as specified by Dockerfile named “Dockerfile.prod.yaml”, prod being production. Other configurations can be set such as a Health Check Path to which the load balancer sends health check requests. Next is the Docker Build Context Directory where the specified path of the Docker build context is defined. This is specified in the Docker Compose files, but since they are not used, they must be defined explicitly. By default, Render uses the root directory unless specified otherwise and is where the build context is located. Further configuration includes Docker Command which overrides Docker CMD and Entrypoint which specifies the executable to be invoked when the container is started. This optional command is not set. Lastly the Auto Deploy is set to yes, indicating that Render automatically deploys on every push to the repository or changes made to the service. Other advanced configurations include environment variable, secret file and build filters.

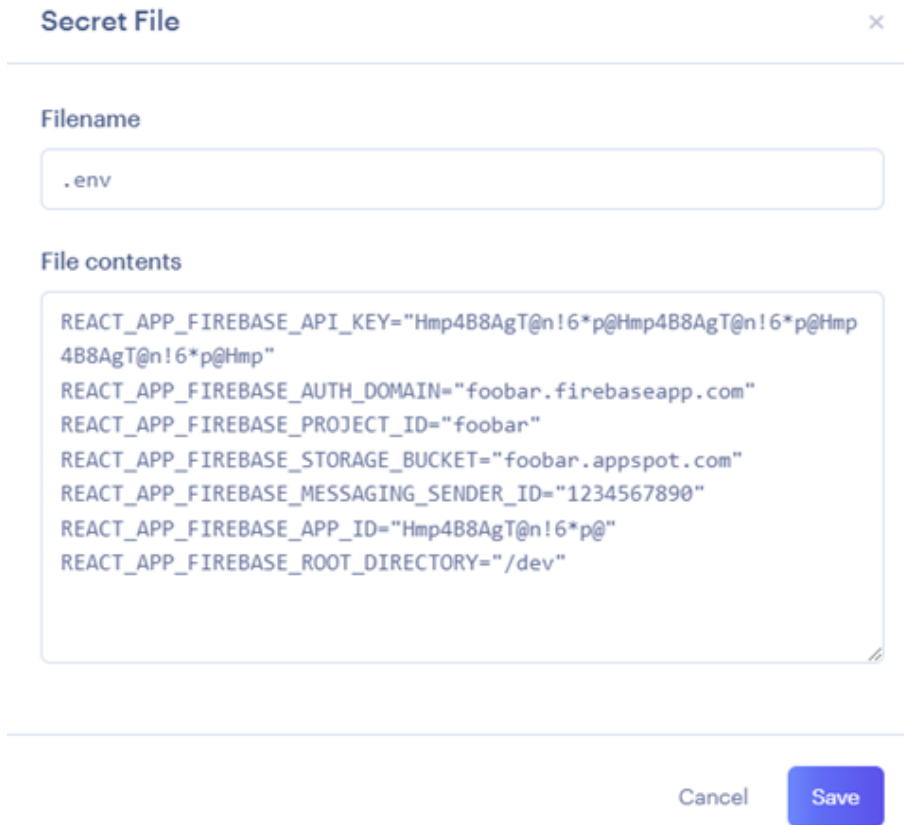


Figure 4.18: Render secret file

The secret file configuration is used where a small window is prompted asking for file name and file contents. The filename is specified as “.env” and the file contents will be the Firebase configuration parameters as displayed by figure 4.18. These files can be accessed during builds and in the code just like regular files. All secret files created are available to read at the root of the repo or Docker context. This allows the container to access the defined environmental variables.

When deploying a web service on Render, the platform performs port detection to determine the appropriate incoming traffic forwarding for the service, including those that utilize custom Dockerfiles. Although setting a PORT environment variable can expedite the port detection process, it is not mandatory. Render does not require exposing port 80 or 443, and it can locate the open port for HTTP traffic on which the server listens. Render’s services operate exclusively over HTTPS, with HTTP requests being redirected to HTTPS. The load balancer terminates TLS for the service, and traffic is then transferred over to the HTTP service on a private network. Therefore, it is possible to serve content over HTTP using any desired port as Render automatically encrypts all traffic.

When done with the configuration of the deployment, a button at the bottom of the page which says “Create Web Service” is clicked and the user gets moved to another page where the deployment is taking place. During deployment, a console windows displays the commands that are being executed and other information.

```
March 21, 2023 at 1:50 AM Live
40ccde2 Update Dockerfile.prod.yaml Removed --legacy-peer-deps

Search logs Search Maximize Scroll to top

Mar 21 01:50:04 AM => Cloning from https://github.com/Ejorge0/huldra...
Mar 21 01:50:05 AM => Checking out commit 40ccde2f1f7012029e39e494f9338207de55cc6c in branch main
Mar 21 01:50:08 AM #1 [internal] load build definition from Dockerfile.prod.yaml
Mar 21 01:50:08 AM #1 transferring dockerfile: 261B done
Mar 21 01:50:08 AM #1 DONE 0.0s
Mar 21 01:50:08 AM
Mar 21 01:50:08 AM #2 [internal] load .dockerignore
Mar 21 01:50:08 AM #2 transferring context: 67B done
Mar 21 01:50:08 AM #2 DONE 0.0s
Mar 21 01:50:08 AM
Mar 21 01:50:08 AM #3 [internal] load metadata for docker.io/library/node:alpine
Mar 21 01:50:08 AM #3 ...
Mar 21 01:50:08 AM
Mar 21 01:50:08 AM #4 [auth] library/node:pull token for registry-1.docker.io
Mar 21 01:50:08 AM #4 DONE 0.0s
Mar 21 01:50:08 AM
Mar 21 01:50:08 AM #5 [auth] library/nginx:pull token for registry-1.docker.io
Mar 21 01:50:08 AM #5 DONE 0.0s
Mar 21 01:50:08 AM
Mar 21 01:50:08 AM #6 [internal] load metadata for docker.io/library/nginx:latest
Mar 21 01:50:09 AM #6 DONE 1.3s
Mar 21 01:50:09 AM
Mar 21 01:50:09 AM #3 [internal] load metadata for docker.io/library/node:alpine
Mar 21 01:50:09 AM #3 DONE 1.3s
Mar 21 01:50:09 AM
Mar 21 01:50:09 AM #7 [internal] load build context
Mar 21 01:50:09 AM #7 transferring context: 1.91MB 0.0s done

Scroll to bottom
```

Figure 4.19: Render deployment build

Based on the last commit to the main branch of the repository, Render deploys or re-deploys the web service. It starts by cloning the repository before loading the build definition from “Dockerfile.prod.yaml”, the “dockerignore” file and build context. These steps are displayed in figure 4.19.

```
Mar 21 01:50:13 AM #10 [build 1/6] FROM docker.io/library/node:alpine@sha256:a67a33f791d1c86ced985f339fa160f6188f590ebbe963fe1cc00adc971fa41
Mar 21 01:50:13 AM #10 resolve docker.io/library/node:alpine@sha256:a67a33f791d1c86ced985f339fa160f6188f590ebbe963fe1cc00adc971fa41 done
Mar 21 01:50:13 AM #10 sha256:c8fd6d18c4e07741813297d22a4b571d0a72a7bebe60a80935aa471e737ae1d1 2.35MB / 2.35MB 0.1s done
Mar 21 01:50:13 AM #10 sha256:eb5638384261ec0a8b95b825a372c46dfe0c89749eaca0e47cd75d09fc7ff51 447B / 447B 0.1s done
Mar 21 01:50:13 AM #10 sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb60cf064a4dc21c 3.37MB / 3.37MB 0.1s done
Mar 21 01:50:13 AM #10 extracting sha256:63b65145d645c1250c391b2d16ebe53b3747c295ca8ba2fcb60cf064a4dc21c 0.1s done
Mar 21 01:50:13 AM #10 sha256:bc59db64d63d87a2863457b909c9a6ca8772a8f04b85959ce5bd955b0ebfcfb 48.23MB / 48.23MB 0.4s done
Mar 21 01:50:13 AM #10 extracting sha256:bc59db64d63d87a2863457b909c9a6ca8772a8f04b85959ce5bd955b0ebfcfb 3.0s done
Mar 21 01:50:13 AM #10 extracting sha256:c8fd6d18c4e07741813297d22a4b571d0a72a7bebe60a80935aa471e737ae1d1 0.1s done
Mar 21 01:50:13 AM #10 extracting sha256:eb5638384261ec0a8b95b825a372c46dfe0c89749eaca0e47cd75d09fc7ff51 done
Mar 21 01:50:13 AM #10 DONE 3.5s
Mar 21 01:50:13 AM
Mar 21 01:50:13 AM #11 [build 2/6] WORKDIR /app
Mar 21 01:50:13 AM #11 DONE 0.0s
Mar 21 01:50:13 AM
Mar 21 01:50:13 AM #12 [build 3/6] COPY package.json .
Mar 21 01:50:13 AM #12 DONE 0.0s
Mar 21 01:50:13 AM
Mar 21 01:50:13 AM #13 [build 4/6] RUN npm install
```

Figure 4.20: Dockerfile build stage commands executed in Render

The instructions defined in the “Dockerfile.prod.yaml” file can be visible in the logs of the Render deployment. Figure 4.20 represents the executed instructions defined in the Dockerfile where the “FROM node:alpine” command, working directory command, copying of packages and NPM install command is visible. These commands are surrounded by a red box in figure 4.20.

```
Mar 21 01:51:59 AM #14 [build 5/6] COPY . .
Mar 21 01:51:59 AM #14 DONE 0.5s
Mar 21 01:51:59 AM
Mar 21 01:51:59 AM #15 [build 6/6] RUN npm run build
Mar 21 01:52:00 AM #15 0.526
Mar 21 01:52:00 AM #15 0.526 > haidra@1.0.0 build
Mar 21 01:52:00 AM #15 0.526 > react-scripts build
Mar 21 01:52:00 AM #15 0.526
Mar 21 01:52:01 AM #15 2.079 Creating an optimized production build...
Mar 21 01:52:33 AM #15 33.35 Compiled with warnings.
```

Figure 4.21: Build commands output, creation of production build

Figure 4.21 displays further logs, continuing the build stage from the Dockerfile. Step 5 copies all the files from the repository to the container. As the last step of the build stage, the command “npm run build” is executed which creates build directory with the production build of the application. As mentioned, the Dockerfile for production is a multistage Dockerfile including a build stage which is used within the stage of Nginx. The build stage creates the image and builds the application before the next stage for Nginx is executed.

```
Mar 21 01:52:33 AM #15 33.40 The build folder is ready to be deployed.
Mar 21 01:52:33 AM #15 33.40 You may serve it with a static server:
Mar 21 01:52:33 AM #15 33.40
Mar 21 01:52:33 AM #15 33.40 npm install -g serve
Mar 21 01:52:33 AM #15 33.40 serve -s build
Mar 21 01:52:33 AM #15 33.40
Mar 21 01:52:33 AM #15 33.40 Find out more about deployment here:
Mar 21 01:52:33 AM #15 33.40
Mar 21 01:52:33 AM #15 33.40 https://cra.link/deployment
Mar 21 01:52:33 AM #15 33.40
Mar 21 01:52:34 AM #15 DONE 34.5s
Mar 21 01:52:35 AM
Mar 21 01:52:35 AM #16 [stage-1 3/3] COPY --from=build /app/build /usr/share/nginx/html
Mar 21 01:52:36 AM #16 DONE 0.4s
Mar 21 01:52:36 AM
Mar 21 01:52:36 AM #17 exporting to docker image format
Mar 21 01:52:36 AM #17 exporting layers
Mar 21 01:52:36 AM #17 exporting layers 0.4s done
Mar 21 01:52:36 AM #17 exporting manifest sha256:330723f3bd1c0efc56192f899081d31a2835277e28c47f1018628b052cc56a18 done
Mar 21 01:52:36 AM #17 exporting config sha256:7e8099bf9340c23d2dfccc7a543263133556394683467da65748e568a88fb26 done
Mar 21 01:52:37 AM #17 DONE 1.6s
Mar 21 01:52:37 AM
Mar 21 01:52:37 AM #18 exporting content cache
Mar 21 01:52:37 AM #18 preparing build cache for export
Mar 21 01:53:10 AM #18 DONE 33.2s
Mar 21 01:53:11 AM # Pushing image to registry...
Mar 21 01:53:13 AM # Upload succeeded
```

Figure 4.22: Successful build stage

As displayed by figure 4.22, the output of the following stage succeeded and is followed by the message “the build folder is ready to be deployed” surrounded in a yellow box. The build folder from the previous stage is then copied to the Nginx

folder located at “/usr/share/nginx/html” for hosting. This command is visible in figure 4.22 by a red box. The stage is completed by pushing the image to the container registry and the message “upload succeeded”.

```
Mar 21 01:53:18 AM /docker-entrypoint.sh: Configuration complete; ready for start up
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: using the "epoll" event method
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: nginx/1.23.3
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: OS: Linux 5.15.0-1031-aws
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker processes
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 27
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 28
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 29
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 30
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 31
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 32
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 33
Mar 21 01:53:18 AM 2023/03/21 00:53:18 [notice] 1#1: start worker process 34
```

Figure 4.23: Output from Nginx on Render

Figure 4.23 shows the given output when starting a Docker container that is running Nginx. The line “using the epoll event method” indicates that Nginx is using the “epoll” event method, which is a scalable I/O event notification mechanism that is available on Linux systems. It further shows the version of Nginx that is running, in this case the version is “1.23.3”. The “built by gcc 10.2.1” line indicates the compiler used to build Nginx and its version. Furthermore, the operating system that the container is running on is shown, in this case Linux 5.15.0-1031-aws. Next, the resource limit is displayed by the line “getrlimit (RLIMIT_NOFILE)”. This line shows the maximum number of open files allowed by the system. After these messages, Nginx starts the worker processes where each worker is allocated an ID.

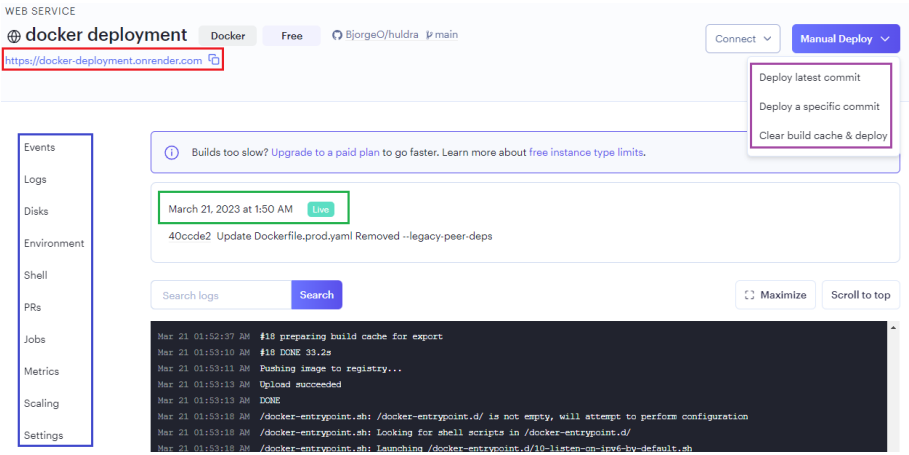


Figure 4.24: Render Web Service dashboard

When a deployment is successful, the time of the deployment and service availability is visible in the dashboard. This is displayed in figure 4.24 surrounded in a green box. The link for the website is at the top left of the page surrounded in a red box. The domain name will be the web service name followed by “.onrender.com”. In this case the name would be “docker-deployment.onrender.com”. One can manually deploy by using the button at the top right. With a drop-down menu, three options of deployment can be chosen as visible in figure 4.24 surrounded in a purple box. A manual deployment can be done using the latest commit, a specific commit or clear build cache and deploy. Lastly, different features are available to the left of the dashboard. Surrounded in a blue box, features such as logs, metrics, scaling and settings can be found. By using a free instance type, not all features will be available.

4.2.3 Huldra Deployment using Railway

As a relatively new infrastructure platform, Railway is a deployment platform inspired by its pioneer Heroku. Much like Render, Railway is presented as an option to Heroku and is a close competitor. At the time of developing a containerized environment of Huldra and deploying using Railway, Docker Compose is not supported. Instead of using Docker Compose for deployment, individual Dockerfiles are used and for the deployment of the production environment, “Dockerfile.prod.yaml” is used. Further, Railway does not provide any features regarding build secrets or secret files. Instead, the only option to include environment variables defined at the host is to implement ARG and ENV instructions in the Dockerfile. Railway states in their guide of deploying using Docker that environments variables defined in the web-based interface must be combined with ARG instructions in the Dockerfile in order to be available at build time.

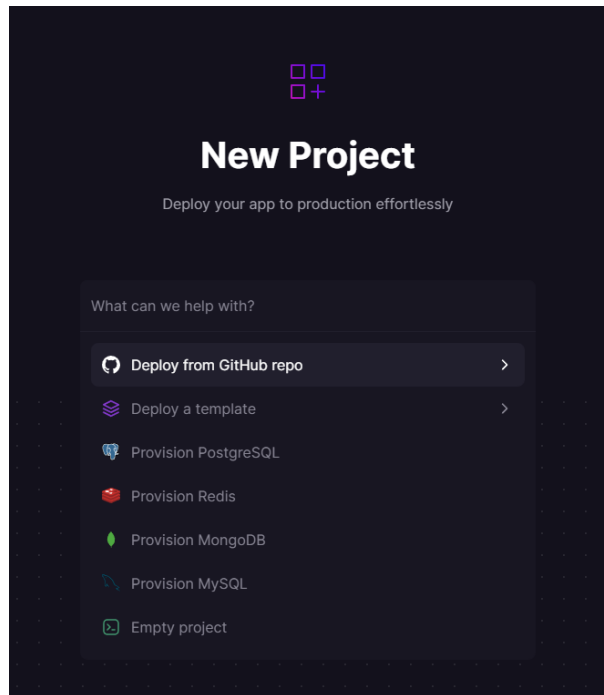


Figure 4.25: Railway project selection

When deploying on Railway, the option of which deployment method is presented in figure 4.25. By clicking “Deploy from GitHub repo”, one is prompted to choose from the available repositories associated with the GitHub account. Railway requires a linked GitHub account if deployment is made using templates or GitHub repositories. After choosing the correct repository for deployment, an option to either deploy or add variables is prompted.

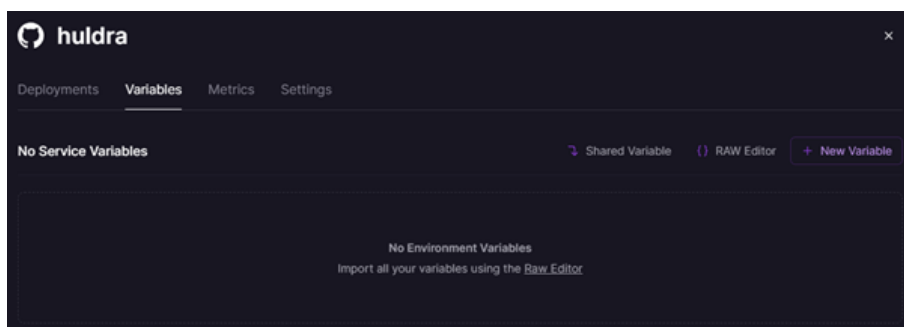


Figure 4.26: Railway variables selection

After choosing the variables option when deploying, the environment gets created and the variable section is displayed. Single variables can be added by clicking the “New Variable” button or it can be added using the raw editor. It can also be added using shared variables which are defined within project settings. These variables can be referenced by multiple services within an environment.

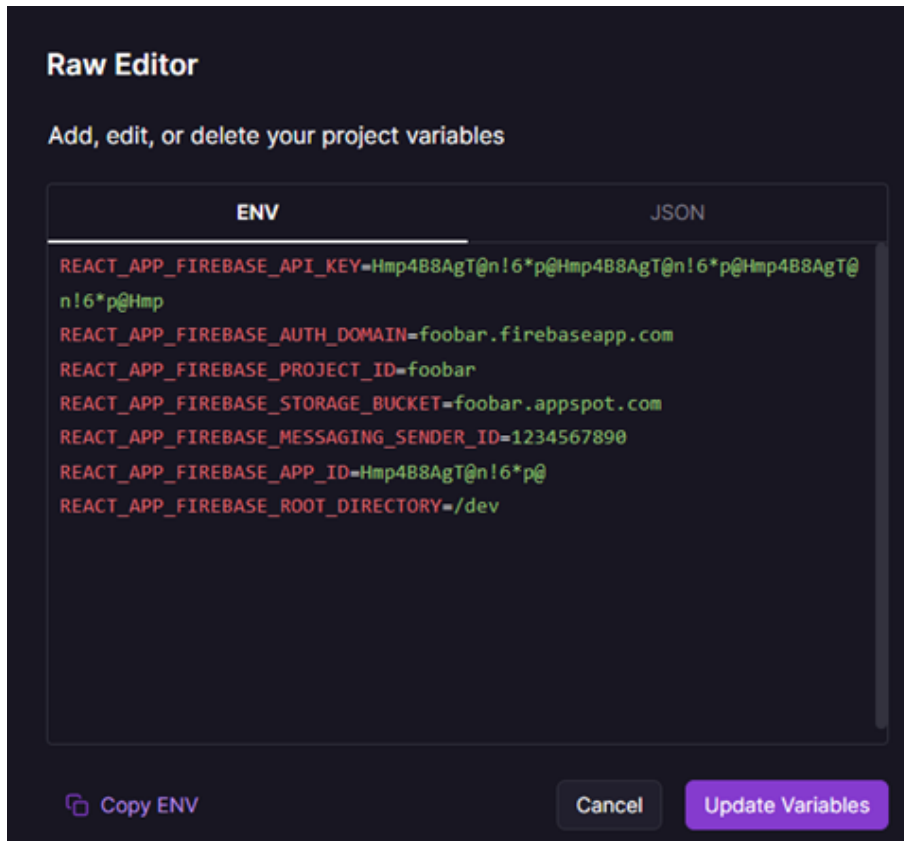


Figure 4.27: Railway raw editor with an example of Firebase connection parameters

The Firebase connection parameters are added using the raw editor as displayed by figure 4.27. The values displayed in figure 4.27 are dummy variables and are replaced with actual Firebase connection parameters which are generated when creating a Firebase project. Further, two more variables are added to ensure a containerized deployment of the production environment served by Nginx.

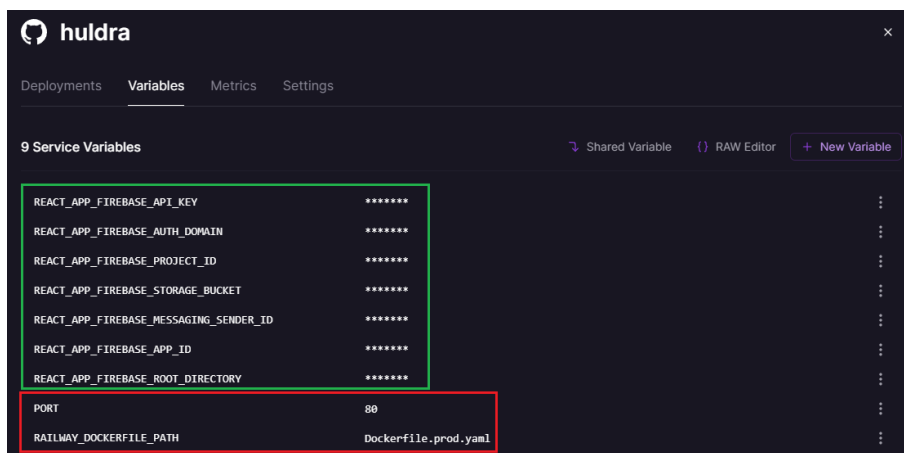


Figure 4.28: Railway variables filled including Firebase connection parameters, port and Dockerfile path

Figure 4.28 displays the service variables for the deployment of Huldra. In a green box the Firebase connection parameters are displayed, and in a red box are the two specified service variables that enable the containerized deployment using Docker and Nginx are displayed. Railway does not store secret files like Render and suggests using ARG instructions at build time to pass the defined environment variables to the Docker container. This is the only option of passing environment variables to the container other than including them as a file in the GitHub repository.

Railway automatically detects and uses a Dockerfile at the services root if it exists, but as the developed Dockerfiles are not named "Dockerfile" they must be specified explicitly. By using the service variable "RAILWAY_DOCKERFILE_PATH", the correct Dockerfile is specified named "Dockerfile.prod.yaml". The port of which the application is listening on is also specified, in order to expose it to the internet. As by default, the Nginx HTTP server listens for inbound connections and connects to port "80" and are specified using the "PORT" service variable.

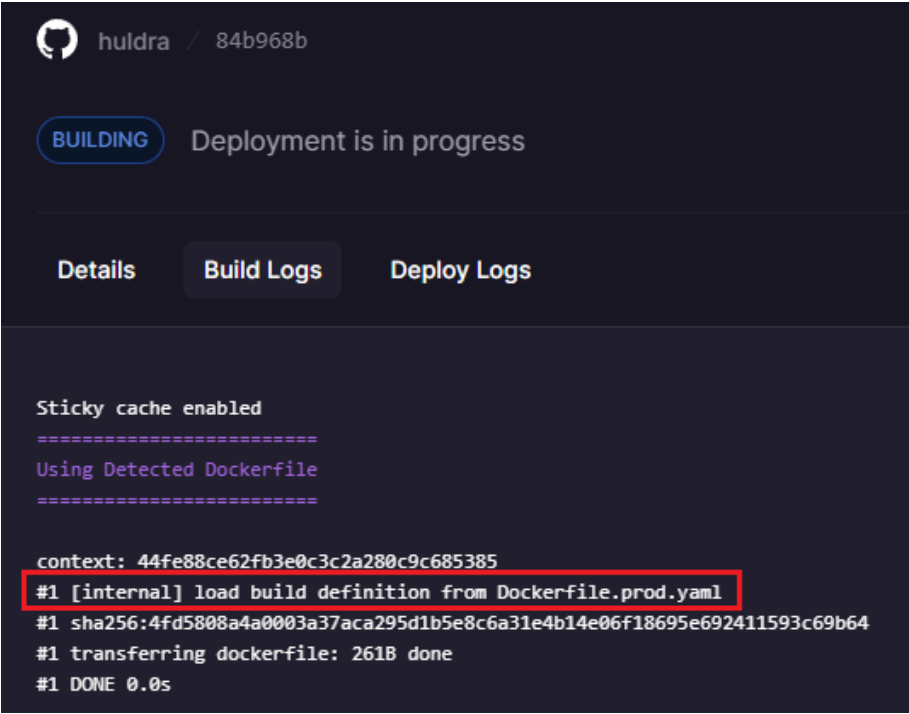


Figure 4.29: Dockerfile detection

When service variables are set, a re-deployment of the service starts. Displayed by figure 4.29, Railway detects and uses the specified Dockerfile named "Dockerfile.prod.yaml". The build process continues to execute the instructions specified in the Dockerfile, completing both stages.

```

#13 45.09 The build folder is ready to be deployed. #14 [stage-1 3/3] COPY --from=build /app/build /usr/share/nginx/html
#13 45.09 You may serve it with a static server: #14 sha256:8b26bd7f7349b1bf3efcfb4eeff9481b8cc7d3d0d5d79f2e8bad35e7da193c5c
#13 45.09 #14 DONE 0.1s
#13 45.09 npm install -g serve #15 exporting to image
#13 45.09 serve -s build #15 sha256:e8c613e07b0b7ff33893b694f7759a10d42e180f2b4dc349fb57dc6b71dcab00
#13 45.09 #15 exporting layers
#13 45.09 Find out more about deployment here: #15 exporting layers 0.1s done
#13 45.09 #15 writing image sha256:5d1058c8e5d9b3b6decf1ea9857e6771602759ebd531348eb4b)
#13 45.09 https://cra.link/deployment #15 naming to us-west1-docker.pkg.dev/railway-infra/railway-docker-users/pro
#13 45.09 #15 DONE 0.1s

#13 DONE 45.3s

Build time: 106.17 seconds

*****
Publishing Image
*****

```

Figure 4.30: Render build stage complete

Figure 4.30 shows the log from the completion of the build stage. As the production environment is built, a new folder is created with the corresponding files for production. This is displayed in figure 4.30 surrounded in a red box. Further, the build folder is copied to the Nginx folder to be served as a web application. This is surrounded by a green box in figure 4.30. Output from the log also shows the build time, in this case 106 seconds.

```

Details    Build Logs    Deploy Logs

/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/03/28 21:02:12 [notice] 1#1: using the "epoll" event method
2023/03/28 21:02:12 [notice] 1#1: nginx/1.23.3
2023/03/28 21:02:12 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/03/28 21:02:12 [notice] 1#1: OS: linux 4.19.0-21-cloud-amd64
2023/03/28 21:02:12 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/03/28 21:02:12 [notice] 1#1: start worker processes
2023/03/28 21:02:12 [notice] 1#1: start worker process 28
2023/03/28 21:02:12 [notice] 1#1: start worker process 29
2023/03/28 21:02:12 [notice] 1#1: start worker process 30

```

Figure 4.31: Railway separate logs, Nginx output

In a separate window, the deploy logs are found as an option named "Deploy Logs" at the top of a specific deployment. Figure 4.31 presents the deployment logs of the service where the configuration of Nginx is displayed in a red box. These configurations are explained in subsection 4.2.2 of the Render deployment. The configuration includes the use of the "epoll" event method followed by the version of Nginx, the compiler, operating system on the container and resource limit. Finally, Nginx starts the worker processes where each process will handle the connection and incoming requests.

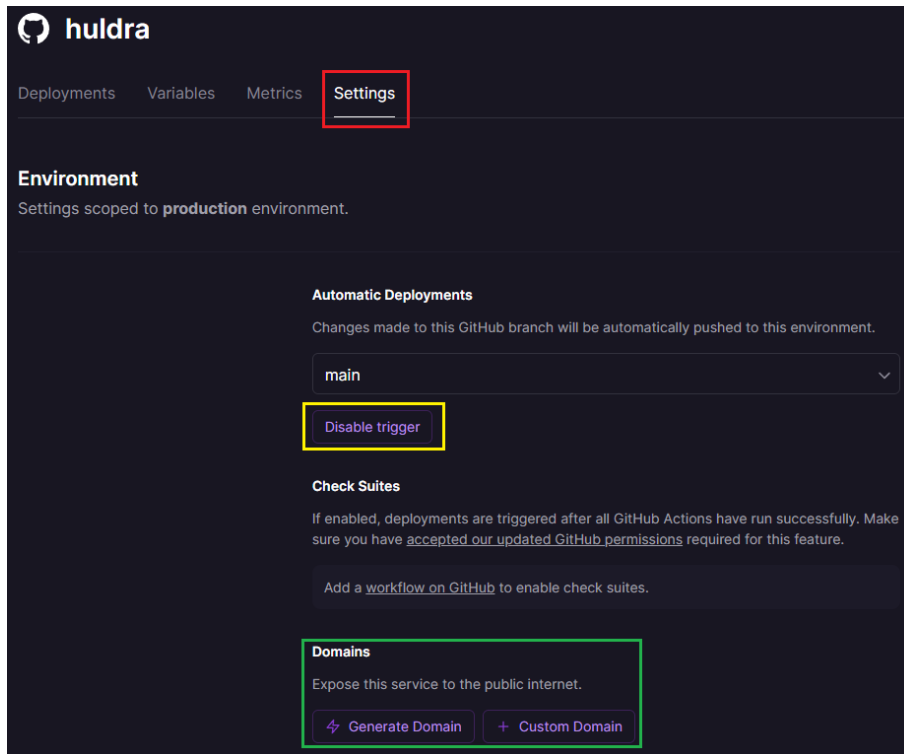


Figure 4.32: Railway environment settings

Before the service can be exposed to the public and accessed by a URL, the domain must be specified. Surrounded in a green box in Figure 4.32, the choice of a generated domain or custom domain is made. If choosing to create a custom domain, the CNAME record must be added to the DNS settings of the domain. In this case, the generated domain name is used and will be displayed within the green box when generated. Also visible in figure 4.32 are the automatic deployments. Based on the selected branch of the connected GitHub repository, the service automatically re-deploys if it detects any changes made within the branch. This can be disabled using the button displayed by a yellow box in figure 4.32.

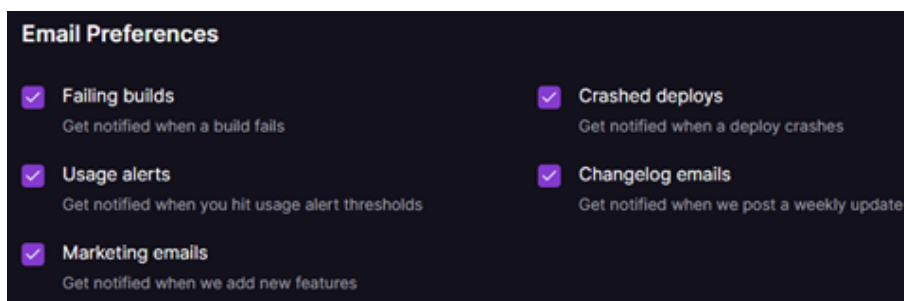


Figure 4.33: Railway email preferences

Railway also provides a set of email preferences where notifications are sent based on

certain scenarios. Displayed in figure 4.33, these notifications include failed builds, crashed deploys, usage alerts, changelog emails and marketing emails. With a simple GUI, these notifications can be toggled based on which notifications one would like to receive. Railway also provides a restart policy where the selected restart option is executed at stopped services. The options include failure, always or never. As the default settings, the option “on failure” is selected where Railway restarts the service if it has stopped due to an error. By selecting “always” the service is restarted if it has stopped for any reason. Lastly, selecting “never” means that Railway will not automatically restart the service and is therefore done manually.

Like Render, Railway compares itself to Heroku and provides an own article of comparison found in their documentation [43]. They claim to have focus on support and developer experience where the Railway team will help scale based on the deployment. Further, Railway offers PR deployments, variable management, rapid builds and local development flows. Railway also uses autoscaling to meet user demands and bills for computes on the platform. Compared to Heroku and Render, the deployed application stays up and is not shut down when there is no activity. Another important difference is that Render offers hidden files whereas Railway does not provide any features for uploading hidden files. This in regard to the Firebase connection parameters for the Docker container, must be passed by using the environmental variables in Railway and implemented within the Dockerfile using ARG and ENV instructions. This prevents a standardize Dockerfile which can be used on other platforms as it is tailored specifically for Railway.

4.2.4 Huldra Deployment using Fly

Render and Railway both provide a deployment platform with a GUI and focus more on simplicity in regard to the deployment process and overall technical aspects. With its simplicity and reduced technical overhead, the platforms are more user-friendly and accessible. As a more advanced deployment platform, Fly does not provide an automated deployment process via GitHub like Render and Railway. Instead, Fly utilizes CLI and relies heavily on application configuration using a “fly.toml” file. Entering the CLI of Fly can be done through the browser using their Web CLI or by installing the command-line utility “flyctl” on the local machine using the command “iwr https://fly.io/install.ps1 -useb | iex”. Further instructions are then provided to sign in or sign up if not already registered.

When using the Web CLI, a Fly Machine is launched with an instance that has “flyctl”

installed. The presumed Fly Machines are Firecracker VMs which are developed by Amazon Web Services. The Firecracker VM is different than typical VMs as it is optimized for running micro-VMs that are extremely small and fast, and used to isolate workloads or processes. Similarly, when deploying an application from a local device which is installed and connected with “flyctl”, Fly Machine is used to provision and manage that application.

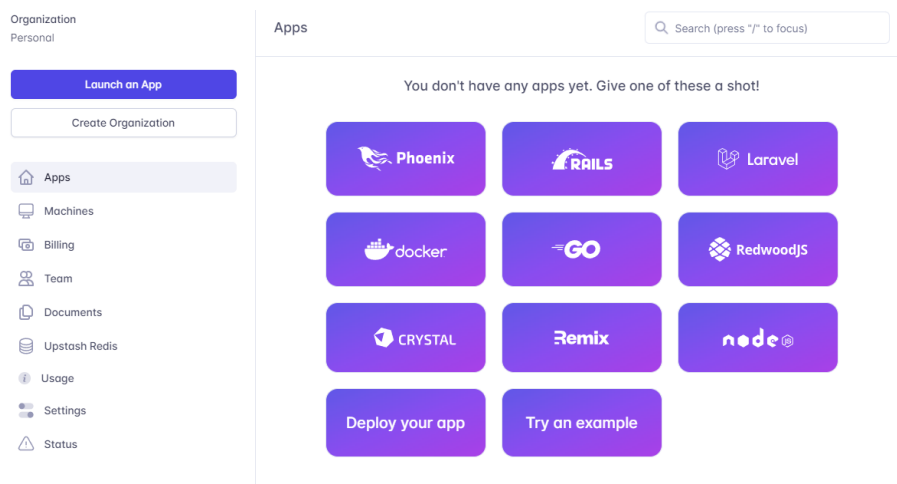


Figure 4.34: Fly personal dashboard

At the dashboard, when logged in, one is prompted with different deployment options. Each of the deployment options provides a detailed description of the process and what commands to execute followed by examples. At the left side of the dashboard, standard navigation such as deployed apps, machines, billing, usage and other settings is located.

Fly does not support Docker Compose as of current deployment and specific Dockerfiles are instead used depending on the environment that is deployed. In this case, the production environment is deployed and specified in the launch command as “fly launch –dockerfile Dockerfile.prod.yaml”. The fly launch command would otherwise automatically detect a Dockerfile, but since there are two Dockerfiles for each environment named differently the filename is defined explicitly in the launch command. If Docker is running locally, the container is built on that machine and then deployed. If Docker is not installed, it builds the container on a Fly build machine.

```

# fly.toml file generated for long-star-7892 on 2023-04-06T16:33:25Z

app = "long-star-7892"
kill_signal = "SIGINT"
kill_timeout = 5
mounts = []
primary_region = "iad"
processes = []

[build]
  dockerfile = "Dockerfile.prod.yaml"

[[services]]
  internal_port = 80
  processes = ["app"]
  protocol = "tcp"
  [services.concurrency]
    hard_limit = 25
    soft_limit = 20
    type = "connections"

  [services.ports]
    force_https = true
    handlers = ["http"]
    port = 80

  [services.ports]
    handlers = ["tls", "http"]
    port = 443

```

Figure 4.35: Fly configuration file

The “fly.toml” file is used to configure an application for deployment. Figure 4.35 displays a generated “fly.toml” file that is automatically generated when launching an application. When executing the command “fly launch” followed by the defined Dockerfile, one is prompted to fill in an application name. The name can be filled in manually or dynamically generated. For this deployment, the name was generated. The chosen name, or generated name, will also be used to create the host name that the application will use by default. Further configuration displayed by figure 4.35, the “kill_signal” option is set to “SIGINT” by default that signals a running process to shut down when an Fly app instance is shutting down. Further, the “kill_timeout” option specifies how much time an instance has to close down before it gets killed. The “primary_region” option specifies which region the application is hosted, in this case Ashburn, Virginia (US). The website provides a list of all available region locations with corresponding region ID.

Further, the specified Dockerfile is automatically defined within the build section of the file as it is defined in the launch command. Other configuration can be made within the build sections such as specifying a certain stage within a multistage Dockerfile and specifying Docker build arguments. The internal port of the services is set to "8080" by default and needs be changed to "80" as Nginx by default listens on port 80.

```
/app/bin/projects/huldra $ flyctl secrets list
Update available 0.0.488 -> v0.0.509.
Run "flyctl version update" to upgrade.
NAME                                DIGEST                                CREATED AT
REACT_APP_FIREBASE_API_KEY          92f7f058e1e329ed                     32s ago
REACT_APP_FIREBASE_APP_ID            e3b97a10aec264b9                     28s ago
REACT_APP_FIREBASE_AUTH_DOMAIN       45cb2eac93ec0f5f                     31s ago
REACT_APP_FIREBASE_MESSAGING_SENDER_ID 032e7273441dab79                     29s ago
REACT_APP_FIREBASE_PROJECT_ID        454dd2cf38ed2bf5                     30s ago
REACT_APP_FIREBASE_ROOT_DIRECTORY    3e1239c28b3e11a0                     28s ago
REACT_APP_FIREBASE_STORAGE_BUCKET    a2d24230c8dc6c15                     29s ago
```

Figure 4.36: Flyctl secrets list

Fly allows secrets to be set using the command “flyctl secrets set <name=value>”. Each individual connection parameter are then set as a secret and staged for the first deployment. One can check the secrets by executing the command “flyctl secrets list” as seen in figure 4.36 below.

However, this approach does not allow the secrets to be accessed by the container as they function as environment variables to the application and are not available at build time. Fly offers another option by using build secrets where the secrets are mounted to the Dockerfile by using a RUN command. The secret values are then provided at the deploy command. To prevent the Dockerfile from being changed only to work with Fly, the Firebase connection parameters are provided as an environment file.

Deploying the application is done using the “flyctl deploy” command. One can use “fly” or “flyctl” as “fly” is a symbolic link to “flyctl”. The image will then be built using a remote builder as Docker is not installed. The remote builder works by using a series of Docker images and build packages to create a build environment that is tailored to the application’s needs. It then uses this environment to compile the application code and create a runnable artifact that is deployed to the Fly platform.

Figure 4.37 displays the deployment output of the application on the Fly platform. When the image is pushed, the application is provisioned an IP address. Provisioning an IPs refers to the process of assigning a unique IP address to a device, in this case the machine, in a network. This allows the machine to communicate on the network. As seen by the deployment output, the application “long-star-7892” is not linked to a machine as it is the first deployment and is therefore automatically allocated a machine. The name of the application is randomly generated hence the name “long-star-7892”. Further, the deployment finished with the output “Finished deploying”.

```

--> Building image done
==> Pushing image to fly
The push refers to repository [registry.fly.io/long-star-7892]
36b467300fef: Pushed
3c197a4304e6: Pushed
ff4557f62768: Pushed
4d0bf5b5e17b: Pushed
95457f8a16fd: Pushed
a0b795906dc1: Pushed
af29ec691175: Pushed
3af14c9a24c9: Pushed
deployment-01GXBWSZJJJ9X8GNZ5207V9CYM: digest: sha256:ec41c697995bb153c92b70
--> Pushing image done
image: registry.fly.io/long-star-7892:deployment-01GXBWSZJJJ9X8GNZ5207V9CYM
image size: 151 MB
Provisioning ips for long-star-7892
  Dedicated ipv6: 2a09:8280:1::1c:83e
  Shared ipv4: 66.241.124.213
  Add a dedicated ipv4 with: fly ips allocate-v4
No machines in long-star-7892 app, launching one new machine
Machine 328744e3b0e258 [app] update finished: success
Finished deploying
/app/bin/projects/huldra $

```

Figure 4.37: Flyctl deployment output

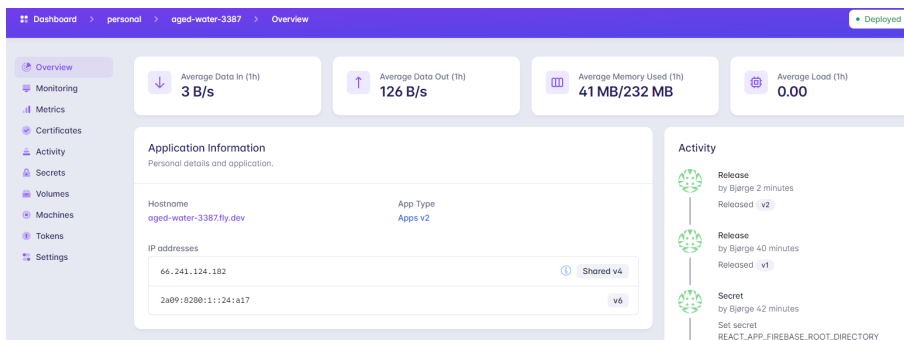


Figure 4.38: Fly application dashboard

When the application is deployed it can be accessed in the personal dashboard of the Fly platform under the Apps section as displayed by figure 4.38. Metrics such as data transfer and memory usage of the VM are visible in the dashboard along with logs produced by the virtual machine which is hosting the application. These web pages are available to the left of the dashboard as separate navigational options. Further, at the top right of the application overview the state of the application is visible. The activity of the application is also visible in the application overview where new deployments and activities executed on the VM are put.

4.3 Chapter Summary

In this chapter we have looked at the implementation of Dockerfiles and Docker Compose files for the containerization of Huldra. None of the tested CSP's supported

Docker Compose which resulted in only using the Dockerfile for a production environment. We also looked at ways to supply the Firebase connection parameters in regards to the different CSP's. Not every CSP provides the same functionality in terms of secrets implementation, and we have seen how these methods of implementing those secrets are done.

Chapter 5

Results

5.1 Objective Results

In this chapter we will examine the range of features provided by Render, Railway and Fly. Additionally, we will investigate the process of deploying an application on each platform. In relevance to usability, the provided feature of deploying an application gives flexibility in terms of choosing the deployment option that suits the user. Further in this section, we highlight the features of each platform that are relevant to usability and compare the technical expertise required to deploy an application on each platform.

Render

With a user-friendly interface for deploying and managing applications, Render makes it easy for users with varying levels of technical expertise to deploy and manage an application. Render supports multiple deployment options including Docker, Git and CLI with automatic scaling based on traffic. As another feature, Render is integrated with GitHub, GitLab and Slack which makes it easy for deployment workflows as new changes to the repository are automatically deployed. It is also worth mentioning that all applications deployed on Render are automatically assigned with a TLS certificate for the subdomain of Render or custom domains.

Using their web-based GUI for deployment requires few clicks and provides step-by-step guidance for deploying different types of services including Web Services, Static Sites, Redis, Private Services, PostgreSQL, Cron Jobs and Background Workers. Through the GUI, a wide range of configuration options is also offered depending on

the chosen service, including the configuration of environment variables, automatic deployment, runtime, start and build commands which are clearly described. To test how easy the deployment process is using Render, we created an account and connected a private GitHub repository. Further, we selected Web Service to be deployed and followed the guided steps provided by the GUI to configure the deployment settings. The Firebase connection parameters were incorporated using Render's secret file feature, which functions similarly as adding an environment file at the root of the repository. With an intuitive description of each available configuration, choosing Docker runtime and configuring the deployment settings for a containerized deployment was straightforward. Based on the walkthrough of a deployment on Render, the deployment process would require little to no prior knowledge of server administration or application deployment. We also found that the provided monitoring tools of Render provided insights into the applications performance via the dashboard and live logs from the application to troubleshoot bugs.

We also found that without the use of the developed Dockerfiles, the application did not work. This test was executed following the same steps and guidelines provided by Render, but instead of using the Docker runtime we chose Node runtime. The application did not successfully deploy where the logs displayed several NPM errors, none of which was troubleshooted. This may be a result of the stated differences in the underlying software of the deployment platform which interfere with the dependencies required for the application to run. The containerized solution therefore prevents this issue by isolating the application in its own environment.

In terms of security regarding the Firebase connection parameters, Render offers the definition of a secret file at build time outside of the build directory. This prevents including a secret file in the GitHub repository and in the root folder for a local deployment using the CLI which can result in unauthorized access to its values.

Railway

Railway offers a modern interface with Git integration and the possibility of using Docker. As with Render, Railway can be used using CLI to interact and create projects from a local directory. With very few clicks a project is created and deployed using a GitHub repository or deployed using a chosen template. Railway offers numerous templates to choose from and automatically creates a GitHub repository when using a template. It also provides the necessary environment variables at creation when using the templates, for example when using a Discord Bot template, it requires a Discord token that identifies the Discord account.

These features together with the simple GUI allow users to deploy their application quickly and efficiently. There is very little room for error when deploying on Railway, but it lacks the overall management and control of the deployment process which is heavily automated. Through testing using the containerized version of Huldra, we discovered that Railway only offers the implementation of environment variables for a containerized application using build arguments. As stated, using build arguments for sensitive information should be prevented as it leaves traces of the information in the image.

We also tested a deployment without the use of the Docker image to test how the platform succeeded by automatically detecting a Node application. Following the same guidelines as with a containerized deployment, the deployment failed with compile errors and build errors. These errors were not further investigated and it is likely that they were related to the internal build commands used by Railway.

Fly

Based on the previous platforms, Fly requires some technical expertise as it does not provide any GUI for application deployment. With similar features as Render and Railway in terms of scaling and monitoring, Fly also allows for a wider range of commands through the CLI including SSH connection to a running instance of an application. This feature makes it easier to debug potential issues rather than relying on logs and monitoring through the dashboard.

We tested the containerized solution using the CLI, deploying Huldra from a local directory. To avoid the need for an environment file containing Firebase connection parameters, Fly offers the use of secrets. These secrets can be added either through the dashboard of the application or via the Fly CLI using their Fly secret command. This provides a more secure way to manage sensitive information, such as connection parameters, without exposing them in the environment file. Through testing, the Firebase connection parameters were added as secrets during deployment of Huldra, but as a result the Fly secrets were not available by default at build-time during the deployment. The error suggested that the React application did not find any environment variables in order to connect to Firebase. After some troubleshooting, we tried adding a new configuration file for Nginx as by default, nginx removes all environment variables inherited from its parent process. The new directive for Nginx included the Firebase connection parameters which propagated the environment variables. This solution did not work either as the same errors appeared.

We also tested a deployment without the use of the developed Dockerfile using Fly's CLI. As we did not include any Dockerfile, Fly made a default Dockerfile in the

deployed directory which consisted of a multi-stage build. The generated Dockerfile was defined with Debian Bullseye as base image followed by an environment variable with the path to the Node binary files. Further, the generated Dockerfile executed a RUN command with packages and tools to build the Node application. It then removed the build tool after the installation was complete. The generated Dockerfile also created a directory called "app" and set it as the working directory. It then copied the application content into the directory. It then executed NPM install NPM run build command to install dependencies and build the application.

The second part of the multi-stage also used the Debian Bullseye base image, but with a smaller version. It starts by copying the previous built "node_modules" and the installed version of Node to the new image along with the application. It further sets the working directory and creates two environment variables. One is set to "production" and the other is a path set to the Node binary files. The generated Dockerfile finishes by using a CMD command that is executed when the container is started. This command is "npm, run, start" which starts the Node application.

When trying to deploy with the generated file, the building of the image succeeded and the application got deployed. It deployed without any errors, but when trying to enter the deployed application in the web-browser we got a 502 bad gateway error indicating a response error from the server when trying to request the website content. Based on the logs in the Fly dashboard, the process was killed due to memory outage. Although the same application was deployed with the use of the developed Dockerfile instead of the generated one, this error seemed strange and was not investigated further.

During the testing phase of the deployment process for each platform, Docker Compose was not supported by any of them. Therefore, all tests were conducted using the production Dockerfile instead of the implemented Docker Compose files. This meant that the service definition and configuration provided by the Docker Compose files were not utilized. However, since the Compose configuration for production only consisted of specific image and container naming and port mapping, the deployment was not affected. For CSP's that supports Docker Compose, the Docker Compose files should be utilized. As an option to Docker Compose, all of the tested deployment platforms provided a distinct configuration file which offers more specific configuration options for a particular service. These files are strictly linked to each of the deployment platforms and are created in the root directory or repository. For example, Render provides the option to use a "render.yaml" file that stores service configuration for a deployment. This file is automatically detected when deploying on Render and it uses the defined configuration for the service

which is being deployed. Similarly, for each corresponding deployment platform, a similar file needs to be created and configured to specify and configure a service further. Thus, these files were not tested or included in the development because the containerized solution is meant to be platform-neutral and readily deployable without any additional configuration from users on various platforms.

5.2 Subjective User Study

5.2.1 Data Cleaning

The dataset from the survey was manually cleaned and re formatted due to its small size. We also used Python with the Pandas and Plotly library for further analysis and removal of outliers. By using the Python libraries we can visually represent data analysis techniques.

During the deployment guide, the participants was asked to enter their local time when completing each step. This value was re formatted given the time after sign in as reference and changed the following times per step relative to that value. This means that at the last step, deleting the Web Service, the total time is registered from after the sign in step. Since the participant choose an age range, we had to choose a method of handling the value of a given range. The chosen method involves taking the middle value from the start and end points of a give range. Therefore, the 18-29 range becomes 24, 30-39 becomes 35 and 40-49 becomes 45. For the range less than 18, we subtract 5 as average number of the other ranges, and add 5 for the 50 and up range as there are not likely to be any respondents over 60.

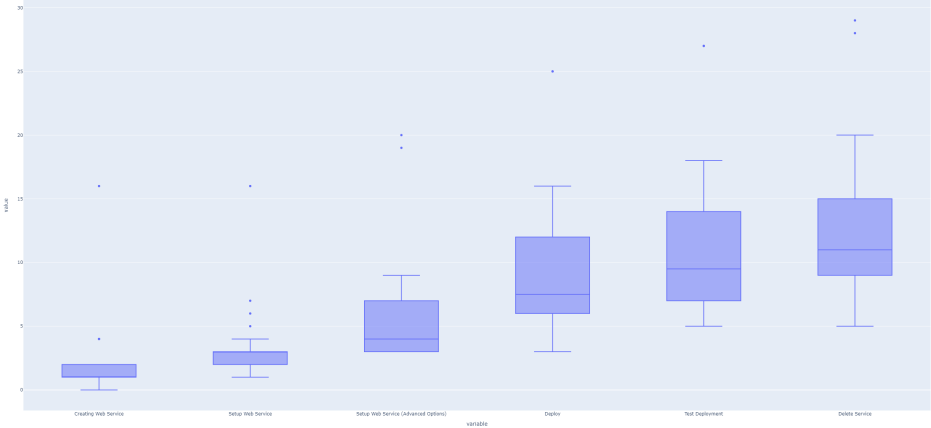


Figure 5.1: Outliers based on total time per step after sign in

Through cleaning and re formatting the dataset, we identified outliers linked to the given time at each step of the guided deployment. These outliers are removed as they may represent human errors of incorrect inputs, distractions or pauses of the participants during the user study. Only the outliers are removed, not the data row associated with the outliers as the rest of the responses from the participant is still used. The outliers are replaced with the mean of the column it is removed from.

5.2.2 Study Results

The sample consisted of 22 responses were 14 (63,6%) of the responses were male and 8 (36,4%) were female. This is visible in figure 5.2 were male is represented in red and female represented in blue.

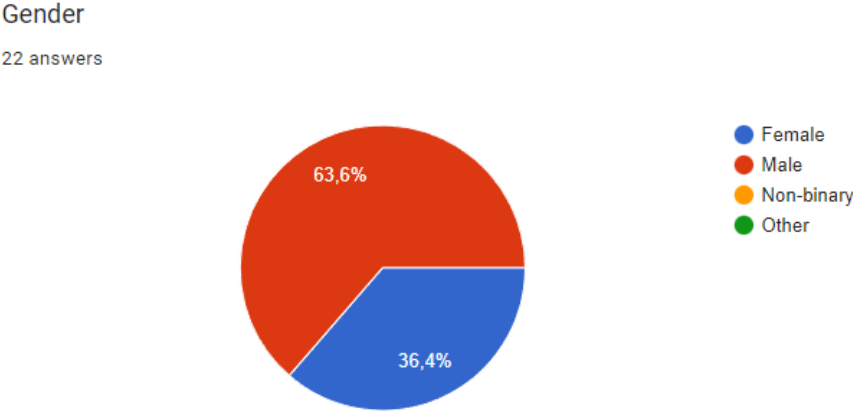


Figure 5.2: User study gender results

Based on the method of handling the age range, the mean age was 28,8 years were 16 (72,7%) of the participants were between the age of 18-29, 4 (18,2%) of the participants between the age of 30-39, and 2 (9,1%) of the participants were of the age 50 or higher. This is visible in figure 5.3.

Value Description	Familiarity with computer science	Familiarity with web app deployment	Familiarity with web Docker virtualization
Mean	2.318	1.864	1.227
SD	1.323	1.207	0.528

Table 5.1: Mean and standard deviation of the familiarity questions.

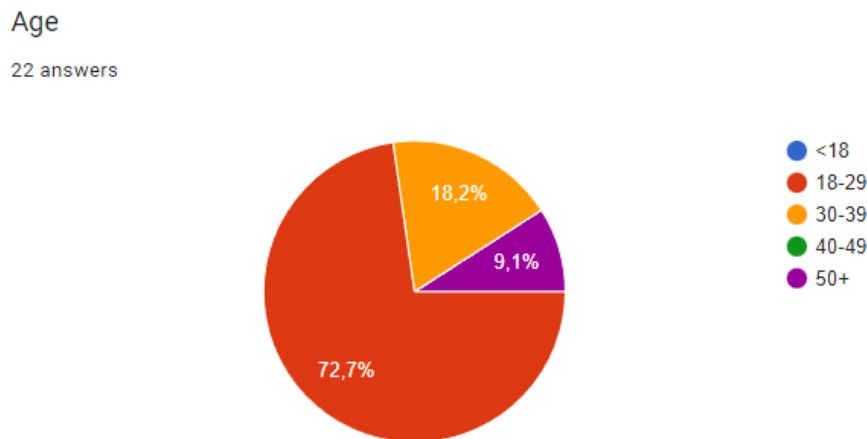


Figure 5.3: User study age results

The standard deviation of the dataset according to age is 9.5. A standard deviation of approximately 9.5 indicates that the ages in the dataset vary quite a bit from the mean age of 28, and are probably a result of the participants who are older than the mean age.

The standard deviation and mean of the familiarity questions asked in the pre-questionnaire are displayed by table 5.2.2. The table shows a relatively large amount of variation in terms of "Familiarity in computer science" with a value of 1.323. This value indicates that familiarity with computer science or technical competence is to some degree diverse among the participants although the mean of 2.318 indicates that the participants has somewhat low technical competence based on the range 1 to 5.

Further, the standard deviation for "Familiar with application deployment" is 1.206, indicating that familiarity with application deployment is less diverse than computer science. Lastly, the standard deviation for "Familiar with Docker" is 0.528, indicating that respondents have the most similar level of familiarity with Docker.

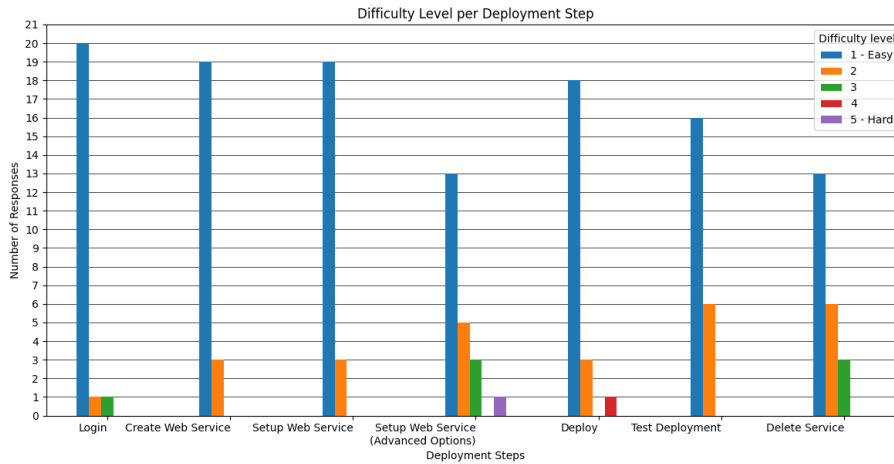


Figure 5.4: Difficulty level per Deployment Step

To get a better overview of each step in the deployment guide, a grouped bar chart is made. Figure 5.4 displays the number of participants in the y-axis and each deployment step in the x-axis. From a scale of 1 to 5, the blue bars indicate a difficulty level of 1. The orange bars displays the number of participants who answers the difficulty level of 2. Difficulty level of 3 is displayed by green bars, 4 by red and purple indicating that the deployment step was hard. We can see that the step that most of the participants struggled with was the "Setup Web Service (Advanced Options)" step were the participants had to create an environment file and copy paste the file contents from the guide. The following steps also proved to be harder than the first three steps, deleting the service as the second hardest step to complete. Overall as indicated by figure 5.4, the blue bars are prominent and are showing that most participants found each deployment step easy to accomplish.

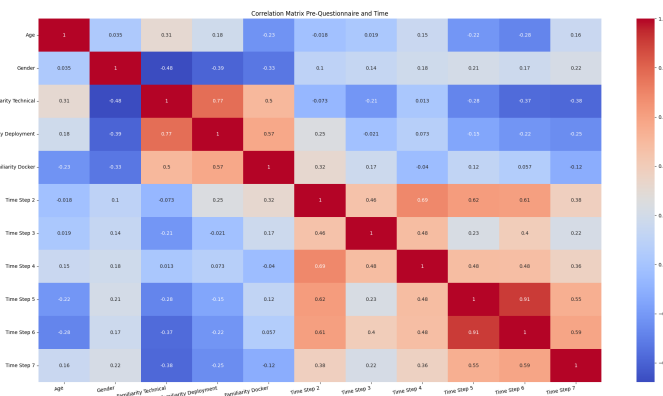


Figure 5.5: Correlation Matrix, pre-Questionnaire and time

To understand the relationship between the pre-questionnaire questions and time spent on each deployment step, a correlation matrix is made displayed by figure Figure 5.5. In this matrix, each row and column represents a variable, and the values in each cell indicate the correlation coefficient between the row and column variables. When analysing, a score closer to 1 and -1 indicates a strong relationship between two variables. A value of 1 indicates a perfectly positive linear correlation and are visible as red in figure 5.5. A value of -1 indicates a perfectly negative linear correlation and are visible as blue in figure 5.5. Each square of the correlation coefficient gradually changes colour to either blue or red based on positive or negative correlation. This means that when there is a positive correlation, the dependent variable increases as the independent variable increases in value. When we have a negative correlation, the dependent variable decreases in value as the independent variable increases.

Comparing the technical, deployment and Docker familiarity with time of each deployment step, we see based on the correlation matrix that there are slightly negative to no correlation.

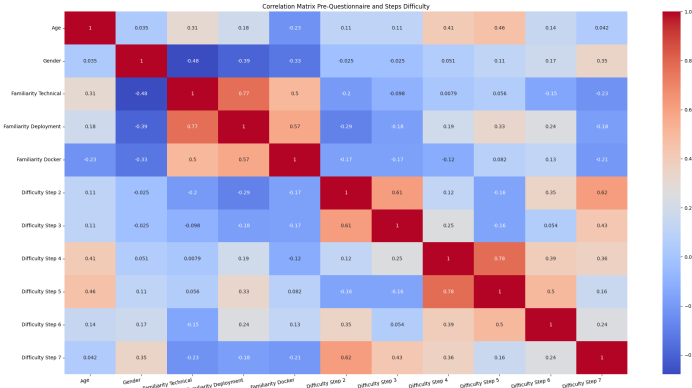


Figure 5.6: Correlation Matrix, pre-Questionnaire and step difficulty

With the same analysis method, we made a correlation matrix of the pre-questionnaire questions and difficulty level of each deployment step. Based on the results visible in figure 5.6, there was a slightly negative correlation for "Familiarity with computer science" and "Familiarity with web Docker virtualization". With the variable "Familiarity with web app deployment", there was a moderate correlation with a value of 0.33 in step five of the deployment guide.

We also see that gender has a negative correlation on the technical, deployment and Docker familiarity's with technical familiarity's being highest with a value of -0.48. We also found that age had a moderate to high correlation with the difficulty of

Value Description	Would like to use this system frequently	System unnecessarily complex	System was easy to use	Need the support of a technical person to be able to use this system	Various functions in this system were well integrated
Mean	2.681	2	4	2.681	3.818
SD	1.249	1.023	0.975	1.323	0.906

Table 5.2: Mean and standard deviation of the SUS and QoE questions, first five questions.

step 4 and 5 with a correlation coefficient of 0.41 in step 4 and a coefficient of 0.46 in step 5. The variable "Familiarity with computer science" had a high correlation with "Familiarity with web app deployment" with a correlation coefficient of 0.77 which indicates that participant with technical competence also had familiarity's with application deployment.

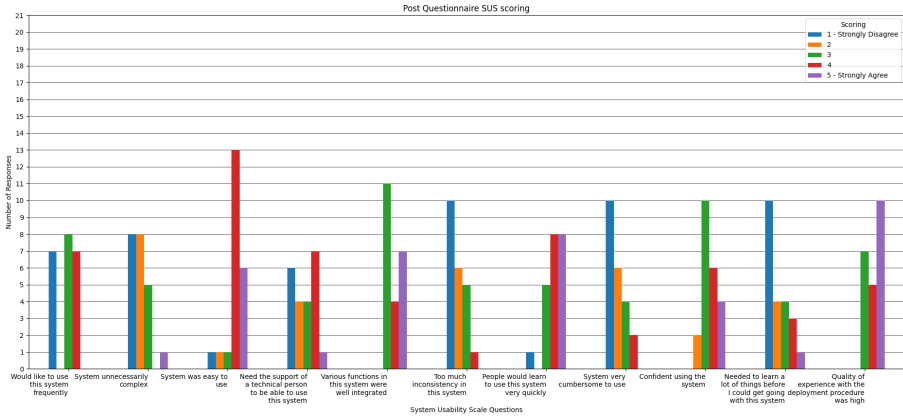


Figure 5.7: Render - System Usability Scale (SUS) scoring of the participants

Based on the post-questionnaire containing the SUS questions and QoE question 3.2, we see that there is a variety of answers ranging from 1 to 5 as visible in figure 5.7. Each bar and its colour represents the total number of answers within the given range of agreement. Blue (1) indicates strongly disagree followed by orange (2), green (3), red (4) and purple (5) which indicates strongly agree.

With a mean of 4.136 and a standard deviation coefficient of 0.888, "Quality of experience with the deployment procedure was high" ranked the highest in terms of the mean 5.2.2. Both "System was easy to use" and "People would learn to use this system very quickly" questions had a mean of 4, indicating that the majority of the participants found the system easy to use and thought that people would quickly learn how to use the system. With the lowest standard deviation coefficient of 0.911, the question "I felt confident using the system" indicates that the ratings are relatively

Value De- scription	Too much inconsistency in this system	People would learn to use this system very quickly	System very cumbersome to use	Confident using the system	Needed to learn a lot of things before I could get going with this system	Quality of experience with the deployment procedure was high
Mean	1.863	4	1.909	3.545	2.136	4.136
SD	0.940	1.023	1.019	0.911	1.283	0.888

Table 5.3: Mean and standard deviation of the SUS and QoE questions, last six questions.

gathered around the mean of 3.545 5.2.2. The mean further indicates that in average, the participants answered just above mid in regards to confidently using the system.

5.3 Chapter Summary

Through the results we have discovered based on the objective results, similarities of the available features provided by the CSP's. As stated in the implementation chapter 4, the implemented Docker Compose files where not used. We discovered unique configuration files similar to Docker Compose of each platform that could be used as a substitute. We also discovered that without the implemented Dockerfiles, a deployment using the default deployment configurations for Node provided by the CSP's, the deployment failed.

The subjective user study showed that participants found the deployment process fairly simple with most of the participants scoring each deployment step the lowest score, indicating that the step was easy. Based on the presented correlation matrices, no significant correlations was found based on time and difficulty level linked to the pre-questionnaire questions and the deployment process. On average, the participants also found the system easy to use.

Chapter 6

Discussion

6.1 Addressing the Research Questions

In this chapter, we will address the research questions outlined in the introduction of the thesis. These questions guided our investigation into the accessibility of application deployment using Cloud Service Provider (CSP) and the use of containerization in CSP environments. Our goal was to identify best practices and potential challenges associated with these technologies to ensure that cloud-deployed applications are accessible to users with less technical expertise regarding application deployment.

At the beginning of this thesis, we formulated specific research questions to guide our investigation. These research questions was introduced in section 1.2.

RQ1. How can the accessibility of application deployment on cloud be improved through the use of containerization technology and simplify the process of deploying applications on the cloud using cloud service providers? Using containerization technology has proven to improve accessibility of cloud deployment by providing a standardized environment, resolving the problem of technical issues linked to the underlying infrastructure of a CSP. Improving the accessibility by removing the technical expertise acquired to address technical issues that appears when deploying on different cloud platforms. To simplify the deployment process, the use of a Docker image which contains the necessary configuration for the application to be built and run are removed. The image separates the configuration needed from the users to a ready to deploy image that is deployed.

RQ2. How does cloud service providers facilitate its features and provide access to a wider range of people regardless of technical expertise? Based on the observations and testing of different CSP's regarding application deployment, the observed/tested deployment process through the web-based interface typically involved guided steps. In cases where the deployment was done through the CLI, the corresponding platform provided documentation on how to install and execute commands based in order to achieve the necessary outcome. Despite the fact that the platforms that were tested had varying templates available, they all shared the same fundamental functionality of deploying applications from either a personal directory or repository. Based on the accessibility, the web-based GUI is user-friendly and designed for people with less technical expertise. The use of the various features offered by the CSP is presented in a way that does not require knowledge of the underlying technical implementation.

The user study's relatively small sample size and shortcomings in regards to age increases the concern of potential age bias. With 16 participants in the age range of 18-29, 2 participants in the age group of 50 or above, and no participants in the 40-49 age range, the study's results may not be representative of the older population's perspectives and experiences. Furthermore, the absence of participants below the age of 18 further limits the generalizability of the findings. By focusing on a more diverse sample size, future studies could get a better representation from various age groups to ensure findings that are more generalizable. This way, the user study and research is not biased against specific age groups.

```
# cd /usr/share/nginx/html
# grep -rl test16682
static/js/main.faaade15.js
```

Figure 6.1: Grep command and found file

```
delegate,e),this}},e.prototype.setMaxUploadRet
n=t.instanceIdentifier,r=e.getProvider("app").
,mr,"PUBLIC").setServiceProps(t).setMultipleIn
SH: !0, REACT_APP_FIREBASE_API_KEY: "test16682"),
REACT_APP_FIREBASE_API_KEY,authDomain:wr.REACT
rk((function e(){var t;return Oe().wrap((funct
(). mark((function e(t){var n;return Oe().wrap
```

Figure 6.2: Firebase api key location


```

}
,
e
});
function mr(e, t) {
  var n = t.instanceIdentifier
  , r = e.getProvider("app").getImmediate()
  , o = e.getProvider("auth-internal")
  , i = e.getProvider("app-check-internal");
  return new hr(r,new ar(r,o,i,new on,n,Rt.SDK_VERSION))
}
!function(e) {
  var t = {
    TaskState: tn,
    TaskEvent: en,
    StringFormat: $t,
    Storage: ar,
    Reference: dr
  };
  e.INTERNAL.registerComponent(new nt("storage",mr,"PUBLIC")
  e.registerVersion("@firebase/storage", "0.7.1")
}(Rt);
var vr = n(4569)
, yr = n.n(vr)
, gr = n(763)
, br = n.n(gr)
, wr = {
  NODE_ENV: "production",
  PUBLIC_URL: "",
  WDS_SOCKET_HOST: void 0,
  WDS_SOCKET_PATH: void 0,
  WDS_SOCKET_PORT: void 0,
  FAST_REFRESH: !0,
  REACT_APP_FIREBASE_API_KEY: "test16682"
},
_r = wr.REACT_APP_FIREBASE_ROOT_DIRECTORY
, kr = {
  apiKey: wr.REACT_APP_FIREBASE_API_KEY,
  authDomain: wr.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: wr.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: wr.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: wr.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: wr.REACT_APP_FIREBASE_APP_ID
},
Sr = function() {
  Rt.apps.length ? Rt.app() : Rt.initializeApp(kr)
}

```

Source map detected. Don't show again [Learn more](#) ×

REACT_APP_FIREBASE_API_KEY 1 of 6 Aa .* Cancel

26 characters selected Coverage: n/a

Figure 6.3: Firebase api key using web developer tool

6.2 Lessons Learned

According to the documentation [18], React incorporates environment variables into the generated HTML/JS files, resulting in the inclusion of the Firebase connection parameters in the image. As a result, anyone who can access the deployed app will be able to view them. Since all the code runs on the client side, there is no other

way to prevent this. Build secrets cannot protect the environment variable values. To verify this, we set a build secret to a searchable value and search for it using the command “`grep -rl test16682`” after running `npm run build`. The build secret was set using Docker mount where a file or directory on the host machine is mounted into a container.

This command revealed a file containing the secret value, which can be confirmed by viewing the source of those files using a web browser. Although Firebase allows exposing the API key as documented by Firebase, relying solely on build secrets to protect sensitive information can be dangerous.

To confirm that the secrets are being sent to the client even with build secrets, we navigated to the known file that contained the secret by viewing it in the web browser using the developer tool.

With the known generated file we also entered the URL that points to the generated file where we further searched for the environment variable.

To clarify, while the general advice to avoid embedding secrets in images is valid, it does not apply to the specific case of React and Firebase. While build secrets are a better option than using an environment file or ARG, it can still be challenging to ensure that secrets are not embedded in the image.

6.3 Other Contributions

Other contributions, after main contributions in section 1.6 related to research questions, are as follows:

- **Source code:** The application used for deployment testing and as the application used in the subjective user study can be found publicly at <https://github.com/simula/huldra>
- **Research artifacts:** The developed Dockerfiles for a development environment 4.4 and production environment 4.2, Docker Compose file for production 4.6 and development environment 4.7, and Nginx configuration 4.3 files can be found in chapter 4.
- **Guidelines:** The guidelines of the subjective user study can be found in the appendix where each page of the Google Forms are listed. These includes the pre-questionnaire, each step of the guided deployment and post-questionnaire.

6.4 Limitations

The objective of this study was to evaluate the accessibility of deployment platforms using Docker Compose for containerized application deployment. One of the significant limitations of this research was the lack of availability of cloud service providers that support the use of Docker Compose. Although Google Cloud and Amazon Elastic Container Service (Amazon ECS) are two popular deployment platforms that offer Docker Compose support, they were not evaluated due to the requirement of payment method in order to use their platform. As a result, the study had to rely on other CSP's that provide free trials but did not support Docker Compose. However, the study was able to successfully test and deploy the Huldra application using Docker Compose on the OsloMet Alto Cloud, which indicates the advantages and usefulness of the approach.

In terms of the user study, a small representative of older participants were limited. This limitation of older participants impacts the age bias of the study.

As another limitation due to time constraints, the planned subjective user study for Fly was not conducted. This study would have involved the collection of data and feedback from participants, like the study for Render, to gather information about the experience of deploying an application using Fly. It would also identify issues and challenges related to the deployment. While this study could have provided valuable insights, as a deployment would be more technical on Fly, it was not feasible within the remaining time. The lack of this user study may have limited the analysis presented in this thesis as a comparison of time and likert scalings would further show relevant findings regarding accessibility in terms of application deployment.

6.5 Future Work

This thesis has accomplished the implementation of Dockerfiles and Docker Compose files that creates and builds a Node application for a production and development environment for the Huldra application. We also managed to conduct a subjective user study for one of the CSP's with 22 responses. Due to time constraints, the following tasks represents areas of potential future work:

- Identifying additional CSP's that support Docker Compose were the application is deployed with focus on accessibility. Examine the overall technical aspect of

deploying on such platforms and evaluate the technical expertise acquired to deploy.

- Conduct a subjective user study on Fly. The deployment guide for Fly is made and as future work, should be conducted and compared against the previous user study on Render.
- Conduct a user study on deployment accessibility using Docker Compose on platforms that supports it. This would further evaluate accessibility and user experience linked to application deployment. It would also discover any difficulties of application deployment in regards to technical knowledge.

Chapter 7

Conclusion

In this thesis we have looked at the different types of clouds and how they operate. With the continuous growth and utilization of cloud services, we examined the use of cloud and how it leverages virtualization to manage resources and isolate environments. We have analyzed the differences between containers and virtualization, as well as how these technologies can be effectively utilized together. Containerization, although not a new technology, has gained significant popularity in recent years due to its efficiency and flexibility in managing and deploying applications in the cloud.

With an increasing number of people using the cloud and the emergence of more cloud service providers, we have developed a containerized solution for Huldra to facilitate deployment on any chosen CSP. The solution is implemented using Docker and Docker Compose. The files we have developed consist of a production environment and a developer environment. By utilizing multi-stage builds, our Dockerfiles contain the essential commands to build a production ready Docker image of React. Additionally, we have created Docker Compose files to allow users to easily configure their service deployment by editing the appropriate files based on the environment, whether it is production or development.

Our deployment testing of various platforms has demonstrated that a functional containerized environment leads to consistent and comparable deployments with previous deployments on other platforms. This is particularly helpful for users with limited technical expertise as it eliminates technical issues related to the platform on which an application is deployed. We conducted a deployment test using both a web-based GUI and a terminal. The GUI proved to be user-friendly and easy to understand, as the platform provided clear and simple steps for deploying and

configuring a deployment. However, deploying through the terminal requires some technical knowledge but offers additional configuration options including deployment using local directories.

After conducting numerous iterations of testing using various approaches to implement Firebase connection parameters, we discovered that regardless of the method used, the values of these parameters were accessible to users. This is true even in a non-containerized environment since the Firebase connection parameters are embedded in the client-side application rather than the server-side as discussed in lessons learned 6. We also explored different methods to exclude the Firebase connection parameters from a container image, including using the secret implementation of CSP's and Docker mount with secrets during build time. None of these approaches proved to work as the build secrets were available in the browser by using the developer tool. Although build secrets offer better protection than using a .env file to hide the secrets in the image, it remains difficult to ensure that the secrets are not embedded within the image.

The user study showed that were little affect in terms of familiarity with computer science, application deployment and Docker with the time used for each deployment step. Based on difficulty, the study also found a moderate correlation between technical knowledge and step 4 (advanced configuration of the web service) and step 5 (deploying the application). The majority of participants rated the difficulty level of each step as 1, suggesting that application deployment using the CSP is overall accessible based on the participants in the study. These findings suggest that the CSP deployment process is intuitive and user-friendly, even for those without a background in computer science or application deployment. In regard to user experience (UX), all participants rated the statement "My overall quality of experience with the deployment procedure was high" on the System Usability Scale (SUS) with a score of 3 or higher on the 5-point agreement scale ranging from 1 (strongly disagree) to 5 (strongly agree). These results suggest that the deployment procedure was generally well-received by participants and that the UX was considered to be of high quality.

Appendix

Huldra Deployment User Study - Render

The user study for Render consists of the pre-questionnaire followed by seven steps of the deployment and the post-questionnaire 3.2. The results of this study can be found in section 5.2. Figure 7.1 represents the pre-questionnaire questions presented to the participants. Figure 7.2 to figure 7.8 represents the deployment steps the participants had to accomplish. As the final section of the user study, the post-questionnaire is displayed by figure 7.9.

Huldra Deployment User Study - Fly

We also present the Google Forms pages of the Fly user study. The pre-questionnaire can be found in figure 7.10. Figure 7.11 to figure 7.18 displays each step of the deployment process for the participant on Fly. This user study also consist of a post-questionnaire found in figure 7.19.


Huldra Deployment User Study - Render


We are conducting research on deploying web applications using cloud platforms, and today we will be deploying an application called Huldra using a cloud platform called Render.

The purpose of this study is to gather information about your experience from deploying an application using Render, and to identify any issues or challenges you may have encountered while deploying Huldra.

The pre questionnaire below will ask you a few questions about your technical competence and experience with application deployment in general. The following steps will walk you through the deployment process. The post questionnaire will ask you to reflect on your experience and provide feedback on the deployment process.

Thank you for your time and participation in this survey.

bjorge7@hotmail.com [Bytt konto](#) 

 Ikke delt

Age

<18

18-29

30-39

40-49

50+

Gender

Female

Male

Non-binary

Other

Please rate from 1 (very low) to 5 (very high)

	1	2	3	4	5
Familiarity with / technical competence in computer science?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Familiarity with web app deployment	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Familiarity with web Docker virtualization	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

What is your profession?

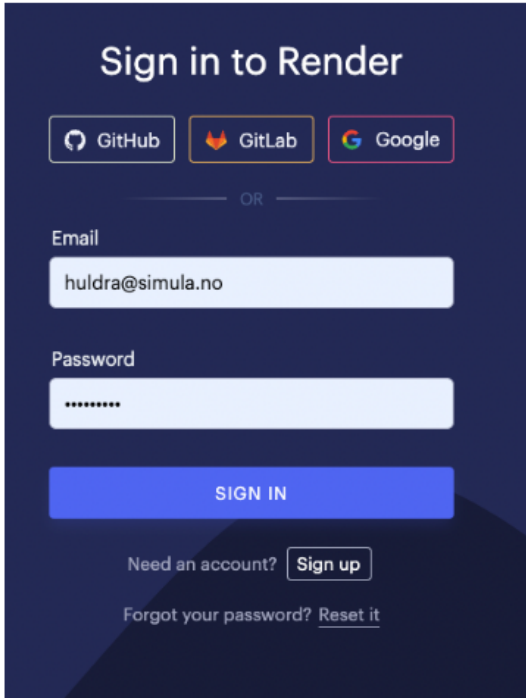
Svaret ditt _____

Figure 7.1: Render - user study pre-questionnaire

Step 1. Login

- Go to <https://render.com/>
- Enter e-mail (*huldra@simula.no*) and password (████████) and click **Sign In**

Example screenshots



Please enter the local time when you complete this step (not the duration, but the * current time)

Tid

__ : __

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Comments about this step (optional)

Svaret ditt _____

Figure 7.2: Render - user study step 1

Step 2. Create Web Service

After you login:

- Click **New** from the top right, and select **Web Service**
- After clicking **Web Service**, navigate to **Public Git repository** at the bottom of the next page
- Enter <https://github.com/BjorgeO/docker-deploy> in the input field as the public GitHub repository address
- Click **Continue**

Example screenshot (create Web Service)

The screenshot shows the Render 'New +' dropdown menu. The 'Web Service' option is highlighted in blue. Other options include Static Site, Private Service, Background Worker, Cron Job, PostgreSQL, Redis, and Blueprint.

Example screenshot (GitHub connection)

The screenshot shows the 'Create a new Web Service' page. Under 'Connect a repository', there are buttons for 'Connect GitHub' and 'Connect GitLab'. To the right, there are links to 'Connect account' for both GitHub and GitLab. Below, there is a 'Public Git repository' section with an input field containing 'https://github.com/BjorgeO/docker-deploy' and a 'Continue' button.

Please enter the local time when you complete this step (not the duration, but the ^{*} current time)

Tid
 __ : __

How easy was this step in your opinion? ^{*}

Very Easy 1 2 3 4 5 Very Hard

○ ○ ○ ○ ○

Comments about this step (optional)

Sivarett ditt

Figure 7.3: Render - user study step 2

Step 3. Setup Web Service

After you create the web service:

- Choose a **Name** for the web service (you can pick any name), and enter it in the relevant input field
- Choose a **Region** for the web service (preferably *Frankfurt (EU Central)*) using the relevant dropdown menu
- Set the **Branch** to be *main*
- Leave the **Root Directory** blank
- Set the **Runtime** to be *Docker*

Example screenshot

render Dashboard Blueprints Env Groups Docs Community Help Free 1 Profile

You are deploying a web service for BjorgeO/docker-deploy.

You seem to be using Docker, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name
A unique name for your web service.

Region
The region where your web service runs. Services must be in the same region to communicate privately and you currently have services running in Frankfurt.

Branch
The repository branch used for your web service.

Root Directory Optional
Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory, and ignores changes outside the directory.

Runtime
The runtime for your web service.

Please enter your payment information to select an instance type with higher limits.

Instance Type	RAM	CPU	Price
<input checked="" type="radio"/> Free	512 MB	0.5 CPU	\$0 / month
<input type="radio"/> Starter	512 MB	0.5 CPU	\$7 / month
<input type="radio"/> Standard	2 GB	1 CPU	\$35 / month
<input type="radio"/> Pro	4 GB	2 CPU	\$55 / month
<input type="radio"/> Pro Plus	8 GB	4 CPU	\$75 / month
<input type="radio"/> Pro Max	16 GB	4 CPU	\$225 / month
<input type="radio"/> Pro Ultra	32 GB	8 CPU	\$450 / month

Need a custom plan? We support up to 512 GB RAM and 64 CPUs.

Unlike paid services, free services scale down when inactive. They also have slower build times. [Learn more about free instance type limits.](#)

Please enter the local time when you complete this step (not the duration, but the **current time**)*

Tid
: _____

How easy was this step in your opinion? *

1 2 3 4 5
Very Easy Very Hard

Comments about this step (optional)

Svaret ditt _____

Figure 7.4: Render - user study step 3

Step 4. Setup Web Service (Advanced Options)

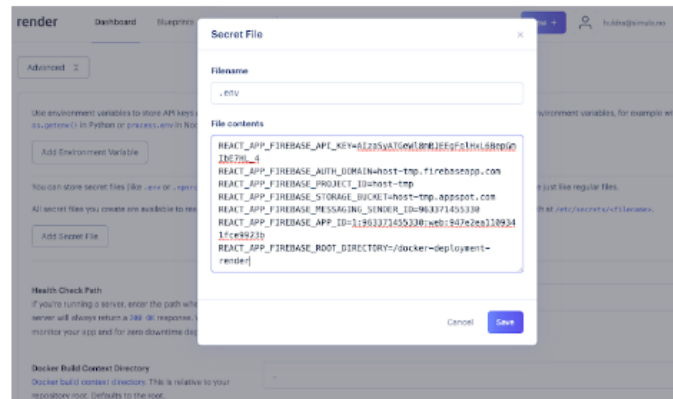
Continue setting up the web service:

- Navigate to the bottom of the page, and click **Advanced**
- Inside the advanced options, click on **Add Secret File**
- Enter **Filename** as `.env`
- Copy and paste the following lines into **File contents**

```
REACT_APP_FIREBASE_API_KEY=[REDACTED]
REACT_APP_FIREBASE_AUTH_DOMAIN=[REDACTED]
REACT_APP_FIREBASE_PROJECT_ID=[REDACTED]
REACT_APP_FIREBASE_STORAGE_BUCKET=[REDACTED]
REACT_APP_FIREBASE_MESSAGING_SENDER_ID=[REDACTED]
REACT_APP_FIREBASE_APP_ID=[REDACTED]
REACT_APP_FIREBASE_ROOT_DIRECTORY=[REDACTED]
```

- Click **Save**, this will close the **Secret File** window.

Example screenshot



Please enter the local time when you complete this step (not the duration, but the * current time)

Tid

__ : __

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Comments about this step (optional)

Svaret ditt

Figure 7.5: Render - user study step 4

Step 5. Deploy

After you click **Save**:

- Navigate to the bottom of the page, and click **Create Web Service**
- The deployment takes a few minutes
- The web app is deployed when the **In progress** indicator changes to **Live**

Example screenshot

The screenshot shows the Render dashboard for a user named 'haldre@simulino'. The main section displays a 'Test Web Service' with a 'Manual Deploy' button. Below this, there are several sections: 'Events' with a warning 'Builds too slow? Upgrade to a paid instance type to go faster.', 'Logs' showing a successful deployment on May 4, 2023 at 4:34 PM, 'Environment' with 'Delete README.md', 'Shell' with a search bar, 'Jobs' with a search bar, 'Metrics', 'Scaling', and 'Settings'. A terminal window shows the deployment logs, including the command 'git init' and the start of 34 worker processes.

Please enter the local time when you complete this step (not the duration, but the current time)

Tid
: _____

How easy was this step in your opinion?

Very Easy 1 2 3 4 5 Very Hard

Comments about this step (optional)

Svaret ditt _____

Figure 7.6: Render - user study step 5

Step 6. Test Deployment

Once the web app is deployed:

- The URL for the deployed app is displayed at the top left, below the web service name (looks like: `https://<web service name>-<random string>.onrender.com`, for example <https://mywebservice-pa5m.onrender.com/>)
- Click on the URL, and check if the Huldra app is properly deployed (it is enough if you confirm that you see the Huldra homepage as indicated below)

Example screenshot



Please enter the local time when you complete this step (not the duration, but the * current time)

Tid
__ : __

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Comments about this step (optional)

Svaret ditt

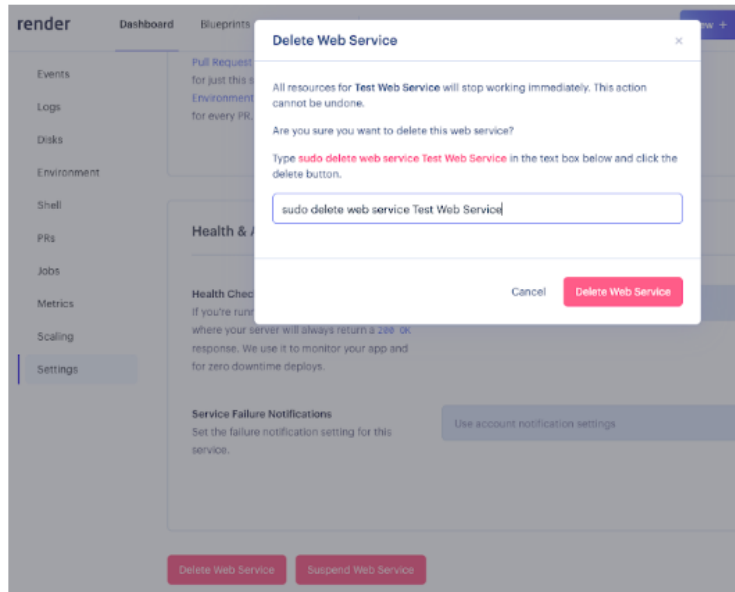
Figure 7.7: Render - user study step 6

Step 7. Delete Service

To delete the service you deployed:

- Go to the **Settings** tab on the left side
- Scroll all the way down in the page, and click **Delete Web Service**
- Type or copy-paste the red text into the input field and click **Delete Web Service**

Example screenshot



Please enter the local time when you complete this step (not the duration, but the * current time)

Tid

__ : __

How easy was this step in your opinion? *

1 2 3 4 5

Comments about this step (optional)

Svaret ditt

Figure 7.8: Render - user study step 7

Post Questionnaire

Please rate from 1 (strongly disagree) to 5 (strongly agree) *

	1	2	3	4	5
I think that I would like to use this system frequently	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system unnecessarily complex	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought the system was easy to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would need the support of a technical person to be able to use this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the various functions in this system were well integrated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought there was too much inconsistency in this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I would imagine that most people would learn to use this system very quickly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system very cumbersome to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt very confident using the system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I needed to learn a lot of things before I could get going with this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
My overall quality of experience with the deployment procedure was high	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

General comments (optional)

Svaret ditt _____

Figure 7.9: Render - user study post-questionnaire

Part 1 of 11

Huldra Deployment User Study - Fly

We are conducting research on deploying web applications using cloud platforms, and today we will be deploying an application called Huldra using a cloud platform called Fly.

The purpose of this study is to gather information about your experience from deploying an application using Fly, and to identify any issues or challenges you may have encountered while deploying Huldra.

The pre questionnaire below will ask you a few questions about your technical competence and experience with application deployment in general. The following steps will walk you through the deployment process. The post questionnaire will ask you to reflect on your experience and provide feedback on the deployment process.

Thank you for your time and participation in this survey.

Age

<18

18-29

30-39

40-49

50+

Gender

Female

Male

Non-binary

Other

Please rate from 1 (very low) to 5 (very high)

	1	2	3	4	5
Familiarity with...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Familiarity with...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Familiarity with...	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

What is your profession?

Short reply text

.....

Figure 7.10: Fly - user study pre-questionnaire


Part 2 of 11

Step 1. Login

- Go to <https://fly.io/>
- Enter e-mail (*huldra@simula.no*) and password (██████████) and click **Sign In**

Example screenshots

Sign in to Your Account

 Sign in with Github

or

Email Address

Password

[Forgot Your Password?](#) [Need an Account?](#)

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Figure 7.11: Fly - user study step 1

Part 3 of 11

Step 2. Open Terminal

After you login:

- Navigate to the Dashboard and click **Launch an App**
- Click **Web CLI** to open a terminal, this will open a new browser tab

Example screenshot (open terminal)

The screenshot shows the Fly.io dashboard. On the left is a sidebar with navigation items: Organization, Apps, Machines, Billing, Team, Documents, Upstash Redis, Usage, Settings, Status, and Support. The 'Launch on App' button in the sidebar is highlighted with a red box and a red arrow pointing down. The main content area is titled 'Launch on App' and contains a 'Web Launchers' section with the subtext 'Launch apps without leaving your browser'. Below this, the 'Web CLI' option is highlighted with a red box and a red arrow pointing down. Other options include 'Turboku', 'Code Server', and a 'Frameworks' section with 'Phoenix' and 'Django'.

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Figure 7.12: Fly - user study step 2

Part 4 of 11

Step 3. Clone Repository/Project

✕
⋮

After you open the terminal:

- Click the button **Launch Web CLI**
- In the terminal enter the command `git clone https://github.com/BjorgeO/docker-deploy` (to copy text into the terminal, mouse right-click and choose paste)
- To navigate to the cloned directory, enter the command `cd docker-deployment`

Example screenshot

Want to launch a GitHub App?

Connect your accounts.

Connect your GitHub account to your Fly.io account to access your apps and choose one to launch.

🔄 Sign in with GitHub

Try Our CLI from Your Browser

Launch a terminal with one click.

Want to try us out without having to download anything? We use Fly Machines to launch a terminal with an app that has `flyctl` installed and ready to go.

🚀 Launch Web CLI

⌚ try it later

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid 🕒

How easy was this step in your opinion? *

1

Very Easy

2

3

4

5

Very Hard

Figure 7.13: Fly - user study step 3

Part 5 of 11

Step 4. Prepare Application for Deployment

In order to prepare the application for deployment:

- Execute the command `flyctl launch`
- When prompted for app name, press **enter** to generate name
- Type "n" for the following questions

Example screenshot

```
/app/bin/docker-deploy $ flyctl launch
Creating app in /app/bin/docker-deploy
Scanning source code
Detected a Dockerfile app
? Choose an app name (leave blank to generate one):
automatically selected personal organization:
Created app 'broken-firefly-246' in organization 'personal'
Admin URL: https://fly.io/apps/broken-firefly-246
Hostname: broken-firefly-246.fly.dev
? Would you like to set up a Postgresql database now? No
? Would you like to set up an Upstash Redis database now? No
Wrote config file fly.toml
? Would you like to deploy now? No
Your app is ready! Deploy with `flyctl deploy`
/app/bin/docker-deploy $
```

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Figure 7.14: Fly - user study step 4

Part 6 of 11

Step 5. Edit Fly file

After you click **Save**:

- Enter the command `"vim fly.toml"`
- Press `i` on your keyboard to enter editing mode
- Navigate using your arrows keys on the keyboard and edit **internal_port** variable from `8080` to `80`
- Save and exit by pressing `esc` on your keyboard followed by typing `"x"` and pressing `enter`

Example screenshot

```

app = "broken-firefly-246"
kill_signal = "SIGINT"
kill_timeout = 5
mounts = []
primary_region = "iad"
processes = []

[[services]]
internal_port = 80
processes = ["app"]
protocol = "tcp"
[services.concurrency]
hard_limit = 25
soft_limit = 20
type = "connections"

[[services.ports]]
force_https = true
handlers = ["http"]
port = 80

[[services.ports]]
handlers = ["tls", "http"]
port = 443

```

Please enter the local time when you complete this step (not the duration, but the current time)

Tid

How easy was this step in your opinion?

Very Easy 1 2 3 4 5 Very Hard

Figure 7.15: Fly - user study step 5

Step 6. Create .env File

Once the web app is deployed:

- Inside the terminal, enter the command "vim .env"
- Copy and paste the Firebase connection parameters into the new file in the browser (note that **CTRL+V** does not work in the terminal, use mouse right-click and paste)
- Save and exit by pressing **esc** on your keyboard followed by typing "x" and pressing enter

Example screenshot

```

REACT_APP_FIREBASE_API_KEY="Hmp4B8AgT€n!6*p€Hmp4B8AgT€n!6*p€Hmp4B8AgT€n!6*p€Hmp
REACT_APP_FIREBASE_AUTH_DOMAIN="foobar.firebaseio.com"
REACT_APP_FIREBASE_PROJECT_ID="foobar"
REACT_APP_FIREBASE_STORAGE_BUCKET="foobar.appspot.com"
REACT_APP_FIREBASE_MESSAGING_SENDER_ID="1234567890"
REACT_APP_FIREBASE_APP_ID="Hmp4B8AgT€n!6*p€"
REACT_APP_FIREBASE_ROOT_DIRECTORY="/dev"
~
~
~
~
~
~
:~
    
```

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Figure 7.16: Fly - user study step 6

Part 8 of 11

Step 7. Deploy the Application with CLI

- Type `flyctl deploy` to deploy your application
- After a few minutes the application is up, and you will see a success message (see screenshot)
- Go to your dashboard in the browser at <https://fly.io/dashboard/personal>
- Navigate to the **Apps** section on the left and you will see your application
- Click on the application, in this example `winter-sun-8649`
- There you will find the Hostname and URL of your application
- The URL will be your application name + `.fly.dev`

Success message

```
No machines in group 'app', launching one new machine
Machine 3d8d991db01389 [app] update finished: success
```

Apps section

Application information

Application Information
Personal details and application.

⚠️ Your app is running on 1 machine. We'd strongly recommend running at least 2 instances to ensure high availability. Check out the [documentation](#) for more details on scaling.

Hostname	App Type
winter-sun-8649.fly.dev	Apps v2

IP addresses

66.241.124.20	Shared v4
2a09:8280:11::15:de6	v6

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

Very Easy 1 2 3 4 5 Very Hard

Figure 7.17: Fly - user study step 7

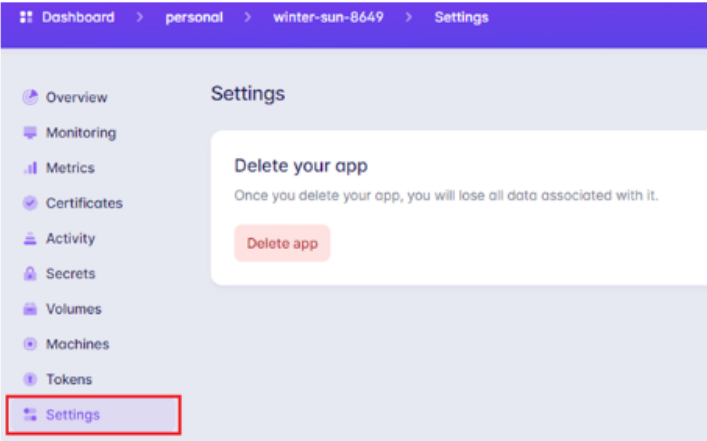
Part 9 of 11

Step 7. Delete the Application

To delete the application:

- Go to the settings section of the application and click the **Delete app** button

Example screenshot



The screenshot shows a web interface with a navigation menu on the left containing: Overview, Monitoring, Metrics, Certificates, Activity, Secrets, Volumes, Machines, Tokens, and Settings. The 'Settings' item is highlighted with a red box. The main content area is titled 'Settings' and contains a white card with the heading 'Delete your app' and the text 'Once you delete your app, you will lose all data associated with it.' Below this text is a red button labeled 'Delete app'.

Please enter the local time when you complete this step (not the duration, but the current time) *

Tid

How easy was this step in your opinion? *

1 2 3 4 5

Figure 7.18: Fly - user study step 8

Post Questionnaire

Please rate from 1 (strongly disagree) to 5 (strongly agree) *

	1	2	3	4	5
I think that I would like to use this system frequently	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system unnecessarily complex	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought the system was easy to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I think that I would need the support of a technical person to be able to use this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the various functions in this system were well integrated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I thought there was too much inconsistency in this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I would imagine that most people would learn to use this system very quickly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the system very cumbersome to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I felt very confident using the system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I needed to learn a lot of things before I could get going with this system	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
My overall quality of experience with the deployment procedure was high	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

General comments (optional)

Svaret ditt _____

Figure 7.19: Fly - user study post-questionnaire

Bibliography

- [1] Mahyar Amini, Nazli Sadat Safavi, Seyyed Majtaba Dashti Khavidak and Azam Abdollahzadegan. 'Types of cloud computing (public and private) that transform the organization more effectively'. In: (May 2013). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2270660.
- [2] Andreea Andrei. 'Industries that Need the Accessibility of Cloud Computing'. In: (2022). URL: <https://www.cloud-awards.com/industries-that-need-the-accessibility-of-cloud-computing/>.
- [3] Oberiri Destiny Apuke. 'Quantitative Research Methods : A Synopsis Approach'. In: *Arabian Journal of Business and Management Review (kuwait Chapter)*. 6 (Oct. 2017), pp. 40–47. DOI: 10.12816/0040336. URL: https://www.researchgate.net/publication/320346875_Quantitative_Research_Methods_A_Synopsis_Approach.
- [4] Helen L. Ball. 'Conducting Online Surveys'. In: *Journal of Human Lactation* 35.3 (2019). PMID: 31084575, pp. 413–417. DOI: 10.1177/0890334419848734. URL: <https://doi.org/10.1177/0890334419848734>.
- [5] Rabindra K. Barik, Rakesh K. Lenka, K. Rahul Rao and Devam Ghose. 'Performance analysis of virtual machines and containers in cloud computing'. In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*. 2016, pp. 1204–1210. DOI: 10.1109/CCAA.2016.7813925. URL: <https://ieeexplore.ieee.org/document/7813925>.
- [6] Babak Bashari Rad, Harrison Bhatti and Mohammad Ahmadi. 'An Introduction to Docker and Analysis of its Performance'. In: *IJCSNS International Journal of Computer Science and Network Security* 173 (Mar. 2017), p. 8. URL: https://www.researchgate.net/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance.
- [7] Ouafa Bentaleb, Adam S. Z. Belloum, Abderrazak Sebaa and Aouaouche El-Maouhab. 'Containerization technologies: taxonomies, applications and challenges'. In: *The Journal of Supercomputing* (2022). DOI: 10.1007/s11227-021-03914-1. URL: <https://doi.org/10.1007/s11227-021-03914-1>.

- [8] John Brooke. 'SUS: A quick and dirty usability scale'. In: *Usability Eval. Ind.* 189 (Nov. 1995). URL: https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale.
- [9] Ravi Chandola. 'React Reconciliation Algorithm'. In: (2023). URL: <https://medium.com/javarevisited/react-reconciliation-algorithm-86e3e22c1b40>.
- [10] Ankita Desai, Rachana Oza, Pratik Sharma and Bhautik Patel. 'Hypervisor : A Survey on Concepts and Taxonomy'. In: (2013). URL: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=650fe84a42c2914fb94c282cb3736c48e506d462>.
- [11] Jessica Díaz, Daniel López-Fernández, Jorge Pérez and Ángel González-Prieto. 'Hypervisor : A Survey on Concepts and Taxonomy'. In: *Empir. Softw. Eng.* 2 (2021). DOI: 10.1007/s10664-020-09919-3. URL: <https://link.springer.com/article/10.1007/s10664-020-09919-3>.
- [12] Cameron Fisher. 'Cloud versus on-premise computing'. In: *Am. J. Ind. Bus. Manag.* 08.09 (2018), pp. 1991–2006. URL: https://www.scirp.org/html/7-2121263_87661.htm.
- [13] Fly. 'Fly.io Docs'. In: (n.d.). URL: <https://fly.io/docs/>.
- [14] Asbjørn Følstad. 'Users' design feedback in usability evaluation: a literature review'. In: *Hum.-centric Comput. Inf. Sci.* 7.1 (Dec. 2017). URL: <https://hcis-journal.springeropen.com/articles/10.1186/s13673-017-0100-y>.
- [15] Engy Fouda. 'Image Creation, Management, and Registry'. In: *A Complete Guide to Docker for Operations and Development*. Apress, 2022, pp. 15–34. URL: https://link.springer.com/chapter/10.1007/978-1-4842-8117-8_3.
- [16] Michel Gien. 'A File Transfer Protocol (FTP)'. In: *Computer Networks (1976)* 2.4 (1978), pp. 312–319. ISSN: 0376-5075. DOI: [https://doi.org/10.1016/0376-5075\(78\)90009-0](https://doi.org/10.1016/0376-5075(78)90009-0). URL: <https://www.sciencedirect.com/science/article/pii/0376507578900090>.
- [17] Gagan Gurung, Rahul Shah and Dhiraj Jaiswal. 'Software Development Life Cycle Models-A Comparative Study'. In: *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* (July 2020), pp. 30–37. DOI: 10.32628/CSEIT206410. URL: https://www.researchgate.net/publication/346819120_Software_Development_Life_Cycle_Models-A_Comparative_Study.
- [18] Joe Haddad. 'Adding Custom Environment Variables'. In: (May 2020). URL: <https://create-react-app.dev/docs/adding-custom-environment-variables/>.

- [19] K. Hammarberg, M. Kirkman and S. de Lacey. 'Qualitative research methods: when to use them and how to judge them'. In: *Human Reproduction* 31.3 (Jan. 2016), pp. 498–501. ISSN: 0268-1161. DOI: 10.1093/humrep/dev334. eprint: <https://academic.oup.com/humrep/article-pdf/31/3/498/7066330/dev334.pdf>. URL: <https://doi.org/10.1093/humrep/dev334>.
- [20] Malek Hammou, Cise Midoglu, Steven A Hicks, Andrea Storås, Saeed Shafiee Sabet, Inga Strümke, Michael A Riegler and Pål Halvorsen. 'Huldra: A Framework for Collecting Crowdsourced Feedback on Multimedia Assets'. In: ACM, June 2022. DOI: 10.1145/3524273.3532887. URL: <https://dl.acm.org/doi/pdf/10.1145/3524273.3532887>.
- [21] Eko Handoyo, R Rizal Isnantoa and Mikhail Anachiva Sonda. 'SRS Document Proposal Analysis on the Design of Management Information Systems According to IEEE STD 830-1998'. In: *Procedia - Social and Behavioral Sciences* 67 (2012). 3rd INTERNATIONAL CONFERENCE ON E-LEARNING, ICEL 2011, pp. 123–134. ISSN: 1877-0428. DOI: <https://doi.org/10.1016/j.sbspro.2012.11.313>. URL: <https://www.sciencedirect.com/science/article/pii/S1877042812052998>.
- [22] Steven Hicks, Andrea Storås, Michael Riegler, Cise Midoglu, Malek Hammou, Thomas de Lange, Sravanthi Parasa, Pål Halvorsen and Inga Strümke. *Visual explanations for polyp detection: How medical doctors assess intrinsic versus extrinsic explanations*. 2022. arXiv: 2204.00617 [eess.IV]. URL: <https://arxiv.org/abs/2204.00617>.
- [23] Andreas Husa, Cise Midoglu, Malek Hammou, Pål Halvorsen and Michael A Riegler. 'HOST-ATS: automatic thumbnail selection with dashboard-controlled ML pipeline and dynamic user survey'. In: ACM, June 2022. DOI: 10.1145/3524273.3532908. URL: <https://dl.acm.org/doi/10.1145/3524273.3532908>.
- [24] Md Hasan Ibrahim, Mohammed Sayagh and Ahmed E Hassan. 'A study of how Docker Compose is used to compose multi-component systems'. In: *Empir. Softw. Eng.* 26.6 (Nov. 2021). DOI: 10.1007/s10664-021-10025-1. URL: <https://link.springer.com/article/10.1007/s10664-021-10025-1>.
- [25] Ramtin Jabbari, Nauman bin Ali, Kai Petersen and Binish Tanveer. 'What is DevOps? A Systematic Mapping Study on Definitions and Practices'. In: *Proceedings of the Scientific Workshop Proceedings of XP2016. XP '16 Workshops*. Edinburgh, Scotland, UK: Association for Computing Machinery, 2016. ISBN: 9781450341349. DOI: 10.1145/2962695.2962707. URL: <https://doi.org/10.1145/2962695.2962707>.

- [26] Aniket Kittur, Ed H. Chi and Bongwon Suh. 'Crowdsourcing User Studies with Mechanical Turk'. In: CHI '08. Florence, Italy: Association for Computing Machinery, 2008, pp. 453–456. ISBN: 9781605580111. DOI: 10.1145/1357054.1357127. URL: <https://doi.org/10.1145/1357054.1357127>.
- [27] Rosaline de Koning, Abdullah Egiz, Jay Kotecha, Ana Catinca Ciuculete, Setthasorn Zhi Yang Ooi, Nourou Dine Adeniran Bankole, Joshua Erhabor, George Higginbotham, Mehdi Khan, David Ulrich Dalle, Dawin Sichimba, Soham Bandyopadhyay and Ulrick Sidney Kanmounye. 'Survey Fatigue During the COVID-19 Pandemic: An Analysis of Neurosurgery Survey Response Rates'. In: *Frontiers in Surgery* 8 (2021). ISSN: 2296-875X. DOI: 10.3389/fsurg.2021.690680. URL: <https://www.frontiersin.org/articles/10.3389/fsurg.2021.690680>.
- [28] Robert Kosara, Christopher Healey, Victoria Interrante, David Laidlaw and Colin Ware. 'Thoughts on User Studies: Why, How, and When'. In: *IEEE CGA* 23 (Mar. 2004). URL: https://www.researchgate.net/publication/2897870_Thoughts_on_User_Studies_Why_How_and_When.
- [29] Anurag Kumar and Ravi Kumar Singh. 'COMPARATIVE ANALYSIS OF ANGULARJS AND REACTJS'. In: (n.d). DOI: 10.21172/1.74.030. URL: <https://www.ijltet.org/journal/148051944230.1245.pdf>.
- [30] Michael Lang, Manuel Wiesche and Helmut Krcmar. 'Criteria for Selecting Cloud Service Providers: A Delphi Study of Quality-of-Service Attributes'. In: *Information & Management* 55.6 (2018), pp. 746–758. ISSN: 0378-7206. DOI: <https://doi.org/10.1016/j.im.2018.03.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0378720617303142>.
- [31] J. Lee and R. Rosin. 'The Project MAC Interviews'. In: *IEEE Annals of the History of Computing* 14.02 (Apr. 1992), pp. 14–35. ISSN: 1934-1547. DOI: 10.1109/MAHC.1992.10024.
- [32] Lucy Ellen Lwakatare, Terhi Kilamo, Teemu Karvonen, Tanja Sauvola, Ville Heikkilä, Juha Itkonen, Pasi Kuvaja, Tommi Mikkonen, Markku Oivo and Casper Lassenius. 'DevOps in practice: A multiple case study of five companies'. In: *Information and Software Technology* 114 (2019), pp. 217–230. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2019.06.010>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584917302793>.
- [33] Santosh Kumar Majhi and Sunil Kumar Dhal. 'A Study on Security Vulnerability on Cloud Platforms'. In: *Procedia Computer Science* 78 (2016). 1st International Conference on Information Security & Privacy 2015, pp. 55–60. ISSN: 1877-0509.

DOI: <https://doi.org/10.1016/j.procs.2016.02.010>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050916000120>.

- [34] Ilias Mavridis and Helen Karatza. 'Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing'. In: *Future Generation Computer Systems* 94 (2019), pp. 674–696. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2018.12.035>. URL: <https://www.sciencedirect.com/science/article/pii/S0167739X18305764>.
- [35] Cameron McKenzie. 'Docker run vs docker-compose: What's the difference?'. In: (2022). URL: <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Docker-run-vs-docker-compose-Whats-the-difference>.
- [36] Marco Miglierina. 'Application Deployment and Management in the Cloud'. In: *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. 2014, pp. 422–428. DOI: 10.1109/SYNASC.2014.63. URL: <https://ieeexplore.ieee.org/abstract/document/7034713>.
- [37] Oswaldo Moscoso-Zea, Joel Paredes-Gualtor, Pablo Saa and Fanny Sandoval. 'Moving the IT Infrastructure to the Cloud'. In: 9 (Mar. 2018), pp. 79–89. DOI: 10.29019/enfoqueute.v9n1.219. URL: https://www.researchgate.net/publication/331009708_Moving_the_IT_Infrastructure_to_the_Cloud.
- [38] M. Lakshmi Neelima and M. Padma. 'A STUDY ON CLOUD STORAGE'. In: 3 (May 2014), pp. 966–971. URL: <https://ijcsmc.com/docs/papers/May2014/V3I5201499a81.pdf>.
- [39] Volker Nissen and Henry Seifert. 'Virtualization of Consulting – Benefits, Risks and a Suggested Decision Process'. In: Jan. 2015. URL: https://www.researchgate.net/publication/291679483_Virtualization_of_Consulting_-_Benefits_Risks_and_a_Suggested_Decision_Process.
- [40] Awodele Oludele, Emmanuel Ogu, Shade Kuyoro and Chinecherem Umezuruike. 'On the Evolution of Virtualization and Cloud Computing: A Review'. In: *Journal of Computer Science and Applications* 2 (Dec. 2014), pp. 40–43. DOI: 10.12691/jcsa-2-3-1. URL: https://www.researchgate.net/publication/270162063_On_the_Evolution_of_Virtualization_and_Cloud_Computing_A_Review.
- [41] Claus Pahl, Antonio Brogi, Jacopo Soldani and Pooyan Jamshidi. 'Cloud Container Technologies: A State-of-the-Art Review'. In: *IEEE Transactions on Cloud Computing* 7.3 (2019), pp. 677–692. DOI: 10.1109/TCC.2017.2702586.

- [42] Morteza Rahimi, Nima Jafari Navimipour, Mehdi Hosseinzadeh, Mohammad Hossein Moattar and Aso Darwesh. 'Toward the efficient service selection approaches in cloud computing'. In: *Kybernetes* 51.4 (Mar. 2022), pp. 1388–1412. DOI: 10.1108/K-02-2021-0129. URL: <https://www.emerald.com/insight/content/doi/10.1108/K-02-2021-0129/full/html>.
- [43] Railway. 'Introduction'. In: (n.d.). URL: <https://docs.railway.app/>.
- [44] Jorge Ramìrez, Marcos Baez, Fabio Casati, Luca Cernuzzi and Boualem Benatallah. 'Challenges and strategies for running controlled crowdsourcing experiments'. In: (Nov. 2020). URL: <https://arxiv.org/pdf/2011.02804.pdf>.
- [45] Aaqib Rashid and Amit Chaturvedi. 'Cloud Computing Characteristics and Services: A Brief Review'. In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* 7 (Feb. 2019), pp. 421–426. DOI: 10.26438/ijcse/v7i2.421426. URL: https://www.researchgate.net/publication/331731714_Cloud_Computing_Characteristics_and_Services_A_Brief_Review.
- [46] Mukthapuram Reddy. 'Analysis of Component Libraries for React JS'. In: *IARJSET* 8 (June 2021), pp. 43–46. DOI: 10.17148/IARJSET.2021.8607.
- [47] Render. 'Quickstarts'. In: (n.d.). URL: <https://render.com/docs>.
- [48] Jonas Repschlaeger, Stefan Wind, Ruediger Zarnekow and Klaus Turowski. 'Decision Model for Selecting a Cloud Provider: A Study of Service Model Decision Priorities'. In: (2013). Americas Conference on Information Systems. URL: <https://core.ac.uk/download/pdf/301359419.pdf>.
- [49] Shylesh S. 'A Study of Software Development Life Cycle Process Models'. In: (June 2017). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2988291.
- [50] Prasad Saripalli and Gopal Pingali. 'MADMAC: Multiple Attribute Decision Methodology for Adoption of Clouds'. In: *2011 IEEE 4th International Conference on Cloud Computing*. 2011, pp. 316–323. DOI: 10.1109/CLOUD.2011.61.
- [51] Michael Schröder and Jürgen Cito. 'An empirical investigation of command-line customization'. In: *Empir. Softw. Eng.* 27.2 (Mar. 2022).
- [52] Jatankumar Sedani and Minal Doshi. 'Cloud Computing: From The Era Of Beginning To Present'. In: (2015). URL: <https://www.noveltyjournals.com/upload/paper/Cloud%5C%20Computing%5C%20From-286.pdf>.
- [53] Mohammed Suliman. 'A Brief Analysis of Cloud Computing Infrastructure as a Service(IaaS)'. In: 6 (Feb. 2021), pp. 1409–1412. URL: https://www.researchgate.net/publication/349297686_A_Brief_Analysis_of_Cloud_Computing_Infrastructure_as_a_ServicelaaS.

- [54] Damian A. Tamburri, Marco Miglierina and Elisabetta Di Nitto. 'Cloud applications monitoring: An industrial study'. In: *Information and Software Technology* 127 (2020), p. 106376. ISSN: 0950-5849. DOI: <https://doi.org/10.1016/j.infsof.2020.106376>. URL: <https://www.sciencedirect.com/science/article/pii/S0950584920301452>.
- [55] Kyle Wiggers. 'Heroku announces plans to eliminate free plans, blaming 'fraud and abuse''. In: (Aug. 2022). URL: <https://techcrunch.com/2022/08/25/heroku-announces-plans-to-eliminate-free-plans-blaming-fraud-and-abuse/>.
- [56] Robail Yasrab. 'Platform-as-a-Service (PaaS): The Next Hype of Cloud Computing'. In: (Apr. 2018). URL: https://www.researchgate.net/publication/324859738_Platform-as-a-Service_PaaS_The_Next_Hype_of_Cloud_Computing.
- [57] Ilham Yusron and Antoni Wibowo. 'A Performance Analyst Comparison of ReactJS and AngularJS in the Front-End Website'. In: (Aug. 2020). URL: <http://www.warse.org/IJSAIT/static/pdf/file/ijisait01942020.pdf>.
- [58] Fiorella Zampetti, Salvatore Geremia, Gabriele Bavota and Massimiliano Di Penta. 'CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study'. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021, pp. 471–482. DOI: 10.1109/ICSME52107.2021.00048. URL: <https://ieeexplore.ieee.org/document/9609201>.
- [59] Yang Zhang, Bogdan Vasilescu, Huaimin Wang and Vladimir Filkov. 'One Size Does Not Fit All: An Empirical Study of Containerized Continuous Deployment Workflows'. In: *ESEC/FSE 2018*. Lake Buena Vista, FL, USA: Association for Computing Machinery, 2018, pp. 295–306. ISBN: 9781450355735. DOI: 10.1145/3236024.3236033. URL: <https://doi.org/10.1145/3236024.3236033>.
- [60] Hong Zhu and Ian Bayley. 'If Docker is the Answer, What is the Question?'. In: *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. 2018, pp. 152–163. DOI: 10.1109/SOSE.2018.00027. URL: <https://ieeexplore.ieee.org/abstract/document/8359160>.