Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Anders Kampesæter Slaaen
Jørgen Mo Opsahl
Marius Raes
Martin Kristensen Eide

Platform for Secure Data Management

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Jia Chun Lin

May 2022

NTNU
Kunnskap for en bedre verden

Anders Kampesæter Slaaen
Jørgen Mo Opsahl
Marius Raes
Martin Kristensen Eide

Platform for Secure Data Management

Bachelor's thesis in Digital Infrastructure and Cyber Security
Supervisor: Jia Chun Lin
May 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU

Kunnskap for en bedre verden

Platform for secure data management

Anders Kampesæter Slaaen
Jørgen Mo Opsahl
Marius Raes
Martin Kristensen Eide

Preface

We were given this bachelor project from the IT Division at NTNU, and we are grateful for being given the opportunity to to get the chance to contribute to such an important project at NTNU. Throughout the project we have learned a lot both when it comes to new technologies, academic writing, and teamwork. We got introduced to many new technologies and fields in IT management and we would not have succeeded if it was not for the help we received from all the people that supported us this period. We would like to give a big thank you to our contact person at the IT-Division, Eigil Obrestad, who were very eager to help us with the project throughout the whole period.

There were also two individuals that were absolutely extraordinary with helping and guiding us throughout the project and there is no doubt that we would not have made it if it were not for them. We would therefore like to give a big thank you to our supervisor Jia-Chun Lin who spent an enormous amount of time guiding us through this project and we are extremely grateful to have been given her as our supervisor.

The second person we want to thank is Lars Erik Pedersen, who took the time to help us numerous times throughout the project giving us exactly the tips and help we needed when we were stuck.

Lastly we want to thank friends, family and lecturers who spent their time giving us feedback, tips and participate in user tests.

Summary

Date:	20.05.2022
Title:	Platform for Secure Data Management
Authors:	Anders Kampesæter Slaaen Jørgen Mo Opsahl Marius Raes Martin Kristensen Eide
Supervisor:	Jia-Chun Lin
Employer:	NTNU IT Division
Contact Person:	Eigil Obrestad, eigil.obrestad@ntnu.no, 61135143
Keywords:	Cloud, API, Load Balancing, Virtual Workstations, File Server, Active Directory, Windows, OpenStack, Infrastructure as Code, Docker, Web Server.
Pages:	75
Attachments:	4
Availability:	Open

Abstract:

The NTNU IT Division is responsible for IT services provided to nearly 50.000 students at NTNU. NTNU currently has no good solution for sharing and managing highly confidential data, so the IT division has decided to implement a new platform for secure data management. The task for this bachelor project was to make certain modules for such a platform, including orchestration of virtual machines and the network architecture and a back-end API to order secure digital infrastructure for research projects. This thesis first begins with defining the requirements, followed by the technical design of the platform and the implementation process. Through a questionnaire, the platform was evaluated by both students and lecturers at NTNU, and the feedback we received were mostly positive.

Sammendrag

Dato:	20.05.2022
Tittel:	Plattform for Sikker Databehandling
Deltakere:	Anders Kampesæter Slaaen Jørgen Mo Opsahl Marius Raes Martin Kristensen Eide
Veileder:	Jia-Chun Lin
Oppdragsgiver:	NTNU IT-avdelingen
Kontaktperson:	Eigil Obrestad, eigil.obrestad@ntnu.no, 61135143
Nøkkelord:	Sky teknologier, API, Lastbalansering, Virtuelle Arbeidsstasjoner, Fil Server, Active Directory, Windows, OpenStack, Infrastruktur som kode, Docker, Web Server.
Antall sider:	75
Antall vedlegg:	4
Publiseringsavtale:	Åpen

Sammendrag: IT avdelingen på NTNU er ansvarlige for IT tjenestene NTNU tilbyr til sine rundt 50.000 studenter. I dag har NTNU ingen god løsning for å sikkert dele og arbeide fortrolig og strengt fortrolig data. IT avdelingen på NTNU har derfor bestemt seg for å lage en ny plattform for sikker databehandling. Oppdraget for denne bacheloroppgaven var å utvikle enkelte moduler for en slik løsning. Dette inkluderer orkestrering av virtuelle arbeidsstasjoner, nettverksarkitektur og back-end API for å bestille sikker infrastruktur til forskningsprosjekter. I denne rapporten vil kravspesifikasjonene, tekninsk design og implementasjon av produktet være en sentral del. Gjennom en spørreundersøkelse ble plattformen evaluert av både studenter og ansatte på NTNU, og tilbakemeldingene viste seg hovedsakelig å være positive.

Table of Contents

Figures

Tables

Code Listings

Acronyms

AD Active Directory. vi, vii, 15, 32, 53, 56, 66, 71, 74, 75

AD DS Active Directory Domain Services. 15, 16, 56, 57, 71

API Application Programming Interface. iii, vi, 4, 5, 13, 27, 36, 39, 40, 43, 55, 58, 59, 61, 62, 65, 71

CPU Central Processing Unit. 72

DC Domain Controller. vii, 16, 56, 57

DHCP Dynamic Host Configuration Protocol. 15

DIGSEC Digital Infrastructure and Cyber Security. 5

DNS Domain Name System. 15, 16, 56, 71, 72

FEIDE Felles Elektronisk Identitet. 5

GPO Group Policy Object. 56

GUI Graphical User Interface. 29, 43

HPC High performance computing. 18

HTTP Hypertext Transfer Protocol. xv, 41

HTTPS Hypertext Transfer Protocol Secure. xv, 71, 75

IaaS Infrastructure as a Service. vi, 7, 8

IaC Infrastructure as Code. vi, viii, 12, 71, 74

IDI Department of Computer Science. 34

IDS Intrusion Detection System. 8

IIK Department of Information Security and Communication Technology. 9

IP Internet Protocol. 27, 30–32

IT Information Technology. 5, 19, 34, 63

LDAP Lightweight Directory Access Protocol. 16, 66, 70

NTNU Norwegian University of Science and Technology. vi, 1–5, 16, 18, 19, 23–32, 34, 43, 56, 57, 59, 70, 75

POC Proof of concept. xi, 32, 56, 57, 71, 72

RDP Remote Desktop Protocol. vi, 4, 9, 10, 19, 23, 28, 30, 40, 47, 49, 56, 57, 65, 66, 69, 70

SAFE Secure Access to Research Data and E-infrastructure. 19

SFTP SSH File Transfer Protocol. ix, 29, 30, 47, 53–55

SSH Secure Shell. 29, 53, 54

TCP Transmission Control Protocol. 47

TSD Services for Sensitive Data. 18, 19

UDP User Datagram Protocol. 47

UIB University of Bergen. 19

UIO University of Oslo. 3, 18

USIT University Centre for IT. 18, 19

VPN Virtual Private Network. 8, 19, 59, 69

WSGI Web Server Gateway Interface. vi, 13, 14

YAML Yet Another Markup Language. 12, 15, 44

Glossary

200 HTTP OK The 200 HTTP code indicates that the request sent by the user was processed successfully. 37

400 HTTP Bad Request The 400 HTTP code indicates that the request sent by the user was not processed, because it appear to be malformed or invalid . 37

500 HTTP Internal Error The 500 HTTP code indicates that by processing the request, something went wrong on the server side. 37

HTTP DELETE request A HTTP DELETE request is a request sent by a user to delete a resource from a server . 38

HTTP GET request A HTTP GET request is a request sent by a user to retrieve a resource from a server . 38

HTTP POST request A HTTP POST request is used to send data to a server . 37

abstraction is a generalization of what is common in a category. For instance both cats and dogs are animals. Animal is an abstraction of cat. 8

agile An iterative development approach that delivers products faster to customers, by delivering work in smaller increments. 33

authentication Authentication means providing information to make sure you are who you say you are. 75

authorization Authorization means defined access rights . 75

automated Something automated doesn't require human interaction. 3

bloatware Unnecessary preloaded software. 29

client - server model A structure where tasks are separated into those who provide a service (servers), and those that requests a service (clients) . xv

daemon A data program that runs as a background process rather than being under the control of a user. 8

decorator A decorator is a function that both takes a function as parameter, and returns a function back as return value [2]. 36

framework A framework, in a software context, is a collection of pre-written code that you can utilize in a software solution, without having to worry about the underlying logic.. 13

HTTP HTTP is a communication protocol mainly associated with web-traffic in a client - server model relationship . xv

HTTPS Hypertext Transfer Protocol Secure (HTTPS) is the secure version of HTTP. 75

modular When something is modular it only relies on itself, but is (normally) part of a bigger item. 3, 13

open-sourced Published under a licence, allowing users to examine, use, change, and distribute the software. 10

production A production environment is an environment where code is considered "finished" and providing value. For instance being accessible to your customers. 14

regular expression (regex) Regular expression is a string of characters used as search pattern for a text. These search patterns can be rather complex and quite hard to understand. 39

scalability Scalability refer to the ability to adjust the provided resources based on required workload. xv, 14

scalable Something scalable implements scalability. 3, 13

SMB Service Message Block is a communication protocol that Microsoft created for providing shared access to files and printers across nodes on a network. 11

virtual machine A machine resource that uses software to run programs and apps "on top" of an physical computer. 8, 9

Chapter 1

Introduction

This chapter will be an introduction to this bachelor thesis and it will cover the background for the project, the goals for the project, project scope, description of the project group as well as a description of the thesis. Our project is available at `https://git.gvk.idi.ntnu.no/mariurae/orchestrationapi`.

1.1 Background

1.1.1 Our client NTNU

Norwegian University of Science and Technology (NTNU) is the largest university in Norway [3] with a history dating back to 1760. The university is divided into nine faculties and spread over three different campuses: Gjøvik, Trondheim and Ålesund. There were 44 747 students registered in 2021. Eight percent of which were international students, originating from 121 different countries. About 50 percent of the students study technical and natural sciences, while the rest is spread over other sciences such as health, education, economics, social sciences etc. The budget of the university was 10 billion NOK in 2021 [4].

Table 1.1: NTNUs data classification system [5]

Open	This information is available for everyone even without authentication.
Internal	This information is available for certain internal and external users. Most information goes under this category.
Confidential	This information require strict authentication and authorization. This is sensitive information that may hurt whoever the information is about, if it were to be released to the public.
Strictly confidential	This information require strict authentication and authorization to an even higher degree. This is extra sensitive information that could severely hurt whoever the information is about, if it were to be released to the public.

1.1.2 Current solutions

NTNU has their own system for classifying data based on how confidential and sensitive it is. As seen in Table 1.1 this system is divided into four classes. NTNU has access to a range of both external and internal services for storing and accessing sensitive data. However, these solutions have some issues, and have proved inadequate to meet the needs of NTNU. For instance providing insufficient functionalities, giving users too much responsibility and freedom, or being outside of NTNUs control. When interacting with data trough personal devices, most of the solutions allows for copying the data locally. This puts the confidentiality of the data at risk in the case of theft or the personal device being compromised. Table 1.2 lists the current solutions that are allowed for storing data classified as internal or stricter.

Table 1.2: NTNUs list of current solution for storing classified data

NTNUs home directory	This is allowed for all classifications, even Strictly confidential as long as the data is encrypted. This however gives the user full control over the data.
NTNUs shared network directory	This is allowed for data that is classified as Open or Internal. This gives the user full control over the data.
NTNU administered Dropbox	This is allowed for data that is classified as Open or Internal. This gives the user full control over the data.
NTNU-Box	This is allowed for data that is classified as Open or Internal. This gives the user full control over the data .
Office 365	This is allowed for data that is classified as Open and Internal, even Confidential as long as the data is encrypted. This gives the user full control over the data.
NTNUs NICE-1	A similar solution as NTNUs shared network directory, but with stricter authentication. This is allowed for all classifications, even Strictly confidential as long as the data is encrypted. This gives the user full control over the data [6].
HUNT Cloud	This is allowed for all classifications, even Strictly confidential on a individual basis.
UiO TSD	This is allowed for all classifications. This service is provided by a third-party (University of Oslo (UIO)) [7].
NIRD	This is allowed for data that is classified as Open or Internal. This service is provided by a third-party (Sigma2) [8].

1.1.3 Requirements

NTNU handles a large quantities of sensitive data for research and education. The confidentiality, integrity and availability of this data is paramount. Any data leak could have catastrophic consequences for the affected individuals, as well as legal and reputational repercussions for NTNU. Therefore a good solution for interacting as well as storing this data is needed, while maintaining a high level of confidentiality and integrity. NTNU has therefore decided that these services should be provided in-house. The solution to this should be scalable, automated and modular, allowing for future improvements. The service also needs to provide a safe way to import the data to the file storage, possibly exporting data (or certain parts of the data), and a secure virtual workstation for interacting with the data.

1.2 Project Goals

The purpose of this bachelor project is to meet the aforementioned requirements. This would entail creating a system that allows sensitive data to be stored and accessed in a more secure manner than the existing NTNU solutions.

The desired solution must serve as an infrastructure for project members to securely work on highly confidential data. Hence, various capabilities are crucial goals for such a solution. The goals include:

- **Automation:** The solution should be completely automated, making it simple to enter and exit production.
- **Scalability:** The solution should be scalable in a way that it can be expanded easily if needed.
- **Modularity:** The solution should be modular, allowing NTNU to employ relevant components and simply incorporate them into their work.

Other goals that do not serve as specific capabilities in the solution include ensuring that the solution adheres to suggestions and requirements from relevant standards and guidelines. Because this is a solution for handling sensitive data at NTNU, the Norwegian health norm for information security and privacy [9] is a critical guideline to follow, as health data might be extremely private.

Other guidelines the solution aims to satisfy is NTNUs guidelines for information security, including NTNUs guideline for information classification [10] and NTNUs guideline for access control [11].

1.3 Project Scope

The final version of this project would be a secure and user-friendly web platform where authenticated and authorised NTNU members can order a virtual environment for their research project. The platform should provide a method for a project owner to upload and download files to and from the platform. Project members should be able to read and edit the files using virtual desktop clients available through Remote Desktop Protocol (RDP), which provides users with a graphical interface to connect to another computer over a network connection [12].

However due to resource and time constraints, it is not feasible for us to fully implement all of the features in our final product. Our focus would be implementing the logic that orchestrates the virtual environments based on API calls from the web platform. In addition our group have a strong focus on securing the project environment, as well as our runtime environments. Our product will include our own logging server to monitor and debug the system, although in the final product logs should be integrated into NTNUs central logging service. Similarly we were not given access to NTNUs Active Directory. This led to our group having

to create our own Active Directory domain with users to test the domain controller that is used for authentication on the Windows clients.

We decided to not fully design and implement the front-end website where project owners can order and manage their projects. We did however implement a very simplified version of the website, mostly for demonstration purposes. This decision was made mostly due to time constraints and because no one on the group has a lot of skill or passion when it comes to front-end web development. We will implement the back-end API; however, we will not implement authentication to use this API which should probably be done with Felles Elektronisk Identitet (FEIDE) and ideally using two-factor Authentication. We chose not to implement this because you need to be FEIDE-administrator to configure login with FEIDE [13], we are not authorized for this. NTNU has experience and know about how to set up FEIDE authentication so it should be relatively easy for them to implement.

1.4 Project Group

Our group consists of four bachelor students enrolled in the Digital Infrastructure and Cyber Security (DIGSEC)) [14] program at NTNU Gjøvik. This program has provided us with a wide range of skills within development, operations of digital infrastructure. A common theme throughout most of our courses was security, abuse potential and prevention. The courses *Infrastructure: secure core services*, and *Robust and scalable services* provided us with knowledge about OpenStack orchestration, that proved valuable for our project work. The courses *PROG2053 - WWW-teknologier* and *IDATG2204 - Datamodellering og databasesystemer* provided us with the knowledge needed to create APIs and and web-development.

The project owner is Eigil Obrestad on behalf of NTNU IT Department. Jia-Chun Lin, assistant professor at NTNU is our supervisor.

1.5 Thesis Structure

- ***Chapter 1 - Introduction:*** Description of the background of the thesis, project goals, project scope and the structure of the thesis.
- ***Chapter 2 - Background:*** Introduction to OpenStack and privacy requirements for health and research data.
- ***Chapter 3 - Related Work:*** Similar solutions found elsewhere.
- ***Chapter 4 - Requirement Specification:*** Requirements for the developed architecture
- ***Chapter 5 - Technical Design:*** Design of our infrastructure and technical description.

- ***Chapter 6 - Development Process:*** The development process throughout the project.
- ***Chapter 7 - Implementation:*** Covers how the group implemented the required features in our solution.
- ***Chapter 8 - Evaluation:*** Covers our risk analysis of the infrastructure, and the testing performed on it.
- ***Chapter 9 - Discussion:*** Discussion on the work done for the project.
- ***Chapter 10 - Closing Remarks:*** Future improvements and conclusion.

Chapter 2

Background

This chapter will cover technologies and techniques used for developing an automated infrastructure, how they are used as well as their advantages and disadvantages.

2.1 Infrastructure as a Service (IaaS)

For developing the product of this bachelor project, Infrastructure as a Service (IaaS) is used. IaaS is a service used for cloud computing where the consumer gets resources in form of storage, network, instances (servers, clients etc) on a cloud. The consumer then has full control over these resources while the provider is responsible for physical maintenance of the cloud [15], as illustrated in Figure 2.1.

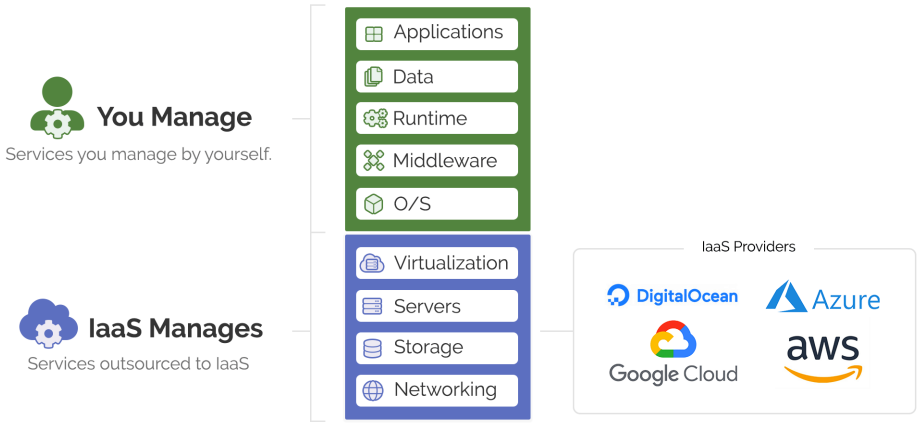


Figure 2.1: The fundamentals of IaaS [16].

In this project it is necessary to use instances, networks and storage to host the platform. This makes it logical to take use of a cloud environment, and IaaS is a service that gives the project group a lot of control over the product. The disad-

vantages of using such a service is the fact that control over the physical hardware is lost. The provider has the responsibility to maintain, secure and virtualize (see the paragraph below) the physical hardware. Fortunately for this project group, the cloud we use is SkyHiGh [17] which makes it possible to have more control over the physical resources, since it is located at the same university that the project are being developed on. There will be a more detailed description of SkyHiGh in Section 2.1.2.

Virtualization is a concept and technology that optimize the use of physical resources. Virtualization makes it possible for hardware to divide its resources to multiple virtual machines that runs its own operating system and acts as independent computers [18]. It makes it possible to easily upscale and downscale resources depending on how much is needed. This is one of the core functionalities of IaaS where cloud development is made easy by providing necessary resources on demand to the consumer.

2.1.1 OpenStack

OpenStack is a free cloud computing platform originating from NASA and Rackspace Hosting [19]. OpenStack controls large pools of compute, storage, and networking resources which can be managed by either an API or a graphical interface. OpenStack works by creating a layer of abstraction over the hardware resources available for the entire infrastructure. This ensures that when a user requests resources for their project, the system administrator for the OpenStack infrastructure can assign resources from the different resource pools available for them. Of the six main services that OpenStack offers, four will be applicable for our platform. These handle computing (Nova), networking (Neutron), orchestration (Heat), and storage (Cinder). These service are introduced as below.

Nova

Nova [20] is the OpenStack service that provides users the ability to start and manage virtual machine servers. The nova service runs as a set of daemon on a linux server. Nova is designed to scale horizontally, which means that rather than switching to larger servers when needing more resource, it creates additional servers to tackle the increased traffic.

Neutron

Neutron [21] is the OpenStack service providing network connectivity to devices running in a OpenStack environment. Neutron manages all parts of the network management and allows users to create complex virtual network topology. OpenStack can deploy additional network services such as firewalls, Virtual Private Network (VPN), and Intrusion Detection System (IDS).

Cinder

Cinder [22] is the OpenStack service providing storage and volume management to Nova virtual machines and containers. The Cinder volume devices provides persistent storage options for virtual machines managed by OpenStack.

Heat

Heat [23] is the OpenStack service to orchestrate multiple cloud applications using templates. Heat can be used to set up multiple different scenarios and environment by using templates to design the system. A typical template will contain the network, subnets and router, as well as the virtual machines

2.1.2 SkyHiGh

SkyHiGh is NTNU Gjøvik's OpenStack installation, hosted by the Department of Information Security and Communication Technology (IIK). The platform is used for both education and research purpose. Both Gjøvik and Trondheim have their own dedicated OpenStack installations on-site. Students, lecturers and researchers can apply to be assigned resources in SkyHiGh for their projects or courses.

2.1.3 Load Balancing RDP sessions

Load balancing is a technique used to distribute network traffic between a set of servers [24], referred to as a server pool. A back-end server have a limited amount of resources and it is therefore a limit to how much network traffic one server can handle. To solve this problem, multiple identical servers can be used where users are distributed between the servers, making the service provided by the servers much more robust.

To distribute users between the servers, a load balancer can to be used. A load balancer is a server that are placed in front of the server pool which tries to solve this problem by sending users to different servers in the server pool. This means that two users trying to connect to for example a website, could be sent to two different servers while the content of the websites are totally identical. Notice how several users are connected to the same load balancer and then are distributed to different servers is illustrated in Figure 2.2.

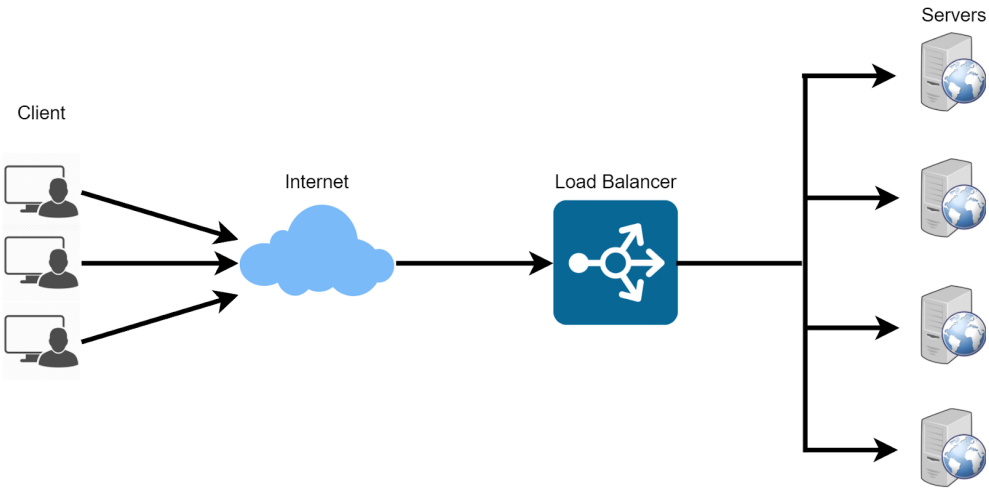


Figure 2.2: Illustration of how a load balancer works [25]

Load balancing is in this project used to distribute Remote Desktop Protocol (RDP) sessions to a server pool of windows machines, users are distributed to different desktop clients with identical functionality. By doing this, the infrastructure solution is made quite robust with a server pool that dynamically change in size according to the number of users.

Remote Desktop Protocol (RDP) is a protocol developed by Microsoft, which allows users to remotely connect to the graphic user interface of a computer (usually windows) through a network [12]. This makes it possible to use a client that are situated in a cloud infrastructure by establishing a RDP session from a personal machine. Said briefly, a user gets the IP address of the load balancer that they connect to, that then establishes a connection with one of the servers in the server pool, trying to maintain a balance of users distributed between the different clients.

Octavia

Octavia is an open-sourced load balancer developed to work with OpenStack [26]. For this bachelor project, this is the load balancer to be used. The reason Octavia is a great load balancer for this bachelor project is because it is optimized to run on an OpenStack cloud environment and SkyHiGh has implemented this load balancer.

2.1.4 Cloud-Init

Cloud-init is a method to initialize cloud intances, and is considered the industry standard [27]. Cloud-init identifies the cloud an instance is booted on and gather the clouds metadata to initialize the instance with potentially storage, network and SSH access. It also allows for more data to be specified with the instance

before boot [27]. All instances booted on the SkyHiGh cloud uses cloud-init to ensure proper configuration of SSH access, network configuration and storage mounting. Cloud-init also allows for the creation of boot scripts that are run on first boot, giving detailed control of the servers configuration.

2.1.5 Storage

Samba [28] is an implementation of Microsoft's SMB networking protocol for UNIX machines. Samba provides file and print service for Windows clients, and it can also integrate into Windows Server domain and act as an domain controller. This enables file sharing between multiple devices, both Linux and Windows with ease. All of the Samba configuration is located in the `/etc/samba/smb.conf` file on the Linux instance. The code listing below shows a simple layout of a file sharing server using Samba. It includes a global configuration for the Samba service, and then the configuration for the file share in the section called *[share]*.

```
1  [global]
2      interfaces = lo ens3
3      bind interface only = true
4      security = share
5      guest account = nobody
6
7  [share]
8      comment = Samba share
9      path = /path/of/share
10     read only = no
11     browsable = yes
12     guest ok = yes
```

Code listing 2.1: Example of a simple Samba configuration

An alternative to the Samba service that we considered was NitroShare [29]. NitroShare is also a cross-platform file transfer that promises gigabit transfer speeds with easy to use software. NitroShare differs from Samba as they only provide users to share files to one computer at a time. This makes it much more troublesome for a project administrator to share files with multiple virtual workstations. NitroShare does not provide the ability to share specific directories and changes done to files are not synchronized to other workstations automatically. Another adverse side of NitroShare is that the source code has not been maintained or developed since June 2019 [30].

We chose Samba as our software for file sharing and storage because it provides consistent and fast file sharing between both the Windows virtual workstations and our Linux file server. Samba also provides consistent synchronization between all workstations.

2.1.6 Infrastructure as Code (IaC)

What is crucial when the cloud infrastructure is created, is automating the process, meaning an infrastructure easily could be taken down and booted back up again. Another key feature of cloud infrastructure is the ability to modify the the infrastructure based on a set of parameters. In our project for example, one project's infrastructure would need to be modified compared to another based on the number of virtual machines. Both automation and modification of the infrastructure can be solved by Infrastructure as Code (IaC).

Infrastructure as Code refers to the management of digital infrastructure. Digital infrastructure is written in code and infrastructure components are placed in a descriptive model [31]. In this project, infrastructure components are placed in what is called a Heat template, using YAML syntax. In the sections below, it is described how API calls are used to pass parameters to the Heat templates. By doing this it is possible to automatically apply and take down an infrastructure with specified requirements.

Code Listing 2.2 shows how a simple Windows server could be represented in code in an OpenStack Heat template.

```
1  #This is a comment
2
3  #'resources' defines all instances in the file
4  resources:
5    #This is the name om the resource
6    server:
7      #This is the type of the resource
8      type: OS::Nova::Server
9      #'properties defines the properties of the resource'
10     properties:
11       name: Windows_Server
12       flavor: m1.small
13       image: "Windows␣10␣21H2␣Enterprise␣[Evaluation]"
14       key_name: Important_Key
15       networks:
16         - network: Main_Network
17       user_data_format: RAW
18       #This is data that are sent with Cloud-Init
19       #when the instance is booted. Here a script
20       #is used, where some parameter is used.
21       user_data:
22         str_replace:
23           template:
24             get_file: bootup.ps1
25           params:
26             float: {get_param: filserver_float}
```

Code listing 2.2: Example of a simple Windows instance

2.2 Application Programming Interface

For users to create a new project, they will need to have an interface to interact with the system. This is where an Application Programming Interface (API) comes into play. An API is an essential part of how modules, services and systems interact with each other. APIs enable different sub-parts of something to be modular, allowing abstraction of the underlying logic of other modules, services and systems that its interacting with [32].

2.2.1 Web API

A Web API is an API that is accessible via a network, for instance the Internet. This allows for communication between computers and servers, whether they are in the same building, or on different continents [32], it does not matter. This communication and the roll of the API is illustrated in the Figure 2.3. The illustration shows a web application ran in a browser communication with an API. Whats behind the API is important and hidden away from the user, who may not be aware that there is a database in the background.

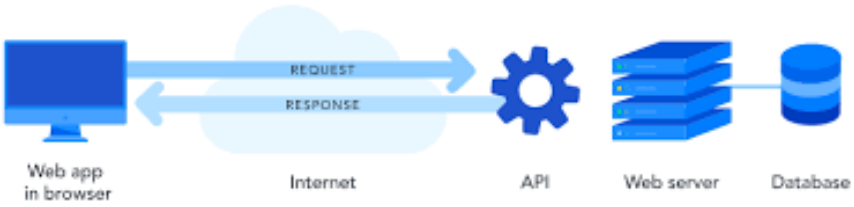


Figure 2.3: Illustration of an API [33]

2.2.2 Flask

Flask is a web framework for the python programming language. Flask is considered to be very lightweight and easy to use, while being scalable and powerful. It is considered lightweight because it is up to the developer to choose what extra functionality should be added, instead of adding a lot of functionality by default that's rarely going to be used [34]. This freedom allows the developer to be creative with what dependencies to be reliant on for functionalities, and the complexity and size of the project can vary quite a lot.

2.2.3 Web Server Gateway Interface

To run the python flask server, making it available to your network, you utilize something called Web Server Gateway Interface (WSGI). A WSGI is what actually

runs your python code and making it available to your network [35]. Flask comes with its own WSGI server, however it is not created, and therefore not recommended, for a production environment. The Flask WSGI server should only be used for development and testing. A good WSGI is important mainly for scalability. As it is supposed to be able to handle hundreds or possibly thousands of requests at the same time. Waitress is a production-quality WSGI server written in Python. It is a good choice of WSGI server because of its performance and its ease of use [36].

2.3 Docker

Docker [37] is a set of software packages that provide an API for creating, running and managing Docker containers. According to the Docker, inc. "A container is a standard unit of software that packages up code and all its dependencies" [38]. Unlike Virtual Machines containers do not need to run on top of an hypervisor, nor do they need to contain an entire operating system to be able to run [39]. Instead, largely thanks to isolation features on modern Linux kernels, like cgroups and namespaces, containers are able to share resources made available by the host machine. This allows makes them much more lightweight and nimble while still being largely isolated from each other [40].

Since Docker containers contain the application and everything it depends on, applications can easily be moved from one host to another while guaranteeing it will still work [39]. This is particularly useful when developing an application, because you can develop and test it in a container on your own machine and then ship the entire container to your production environment.

Another advantage containers provide compared to running applications directly on a machine is security. Since containers are isolated from the host that they are running on, an attacker who compromises a Docker container will only have access to the content inside said container. For example if an attacker manages to get a shell inside a Docker container they will have no way of accessing the files or settings of the host system. Containers therefore provide a great way to reduce the damage potential attacker can cause.

2.3.1 Dockerfile

Docker containers start as Docker images. To create a custom Docker image a *Dockerfile* is used. A *Dockerfile* is a text file containing step by step instructions for Docker to build an image. *Dockerfiles* starts with the FROM keyword, specifying the base image the new image is based on. Instructions after the FROM keyword will be run on top of the base image and each other, as Docker creates intermediary images for each instruction being run. These intermediary images are cached, meaning that if changes are made to the end of a *Dockerfile*, the image does not

need to be built from scratch again.

The simple *Dockerfile* example illustrates some of the most important keywords in *Dockerfiles*. The *COPY* keyword copies a file or folder from the host into the docker image. The *RUN* keyword specifies shell commands that are run when building the image. This keyword is usually used for installing packages or creating files or directories. Finally the *CMD* keyword, while it may seem similar to the *RUN* keyword it differs in that instead of being executed when the image is built it is instead executed once the container is ran. Every *Dockerfile* should contain exactly one *CMD* keyword, because if none are listed nothing will happen once the container is ran, and the container will exit immediately.

```
1 # This is a comment in a simple Dockerfile
2 FROM ubuntu:latest
3
4 COPY ./src /src
5 RUN apt-get update -y
6 RUN apt-get install -y cmatrix
7
8 CMD echo "Hello␣World!"
```

Code listing 2.3: Simple Dockerfile example

2.3.2 Docker Compose

While it is possible to start Docker containers manually through the Docker command line client, Docker Compose is preferred when starting multiple containers that depend on or interact with each other. Docker compose works by creating a *docker-compose.yml* file that by using YAML syntax defines services. Each service runs in a container based on the specified docker image.

Docker Compose provides a lot of functionality for Docker volumes, virtual Docker networks, container port mapping, environment variables and many other powerful Docker features. Once the *docker-compose.yml* file is created this makes it possible to create complex Docker environments using just one command "docker-compose up". A typical Docker workflow would therefore be defining Docker images with *Dockerfiles*, defining the services that are part of application with *docker-compose.yml*, running the services by calling "docker-compose up" [41].

2.4 Active Directory

Active Directory (AD) [42] is a Microsoft product providing a collection of tools and services needed in a big enterprise network. The most essential services provided are Active Directory Domain Services (AD DS), Domain Name System (DNS), and Dynamic Host Configuration Protocol (DHCP).

AD DS is a product for storing information about users, workstations and servers in an hierarchical structure, also called a data store. AD DS is often at the core of most big enterprise networks, for instance NTNU [43].

Domain Name System (DNS) [44] is a suite of protocols used for name resolution in a network. This functionality is an essential part of how humans interact with the internet. It is a lot easier for humans to remember a domain name (e.g. ntnu.no) than its actual IP address (129.241.160.102 as of 23.03.2022 1:27 PM). Translating domain names to IP addresses is the main task of the DNS, as seen in Figure 2.4. DNS also provides the translation of an IP address to a domain name, also called a "reverse lookup". When a DNS is required to translate a domain name to an IP address, the server looks in what is referred to as a "Forward Lookup Zone". When an IP address is unknown, and a translation to a domain name is required, the server looks in what is referred to as a "Reverse Lookup Zone" [45].

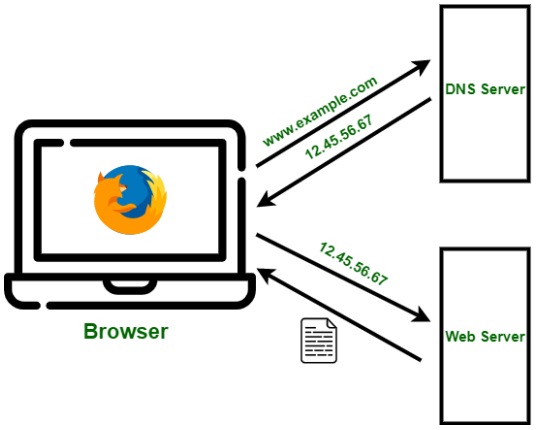


Figure 2.4: DNS-client communication illustration [46].

The servers actually running these services are called Domain Controller (DC). The DC is the core of a computer domain, and at the top of the hierarchy as seen in Figure 2.5. Its main task is authentication and authorization for users within the domain using Lightweight Directory Access Protocol (LDAP) [47].

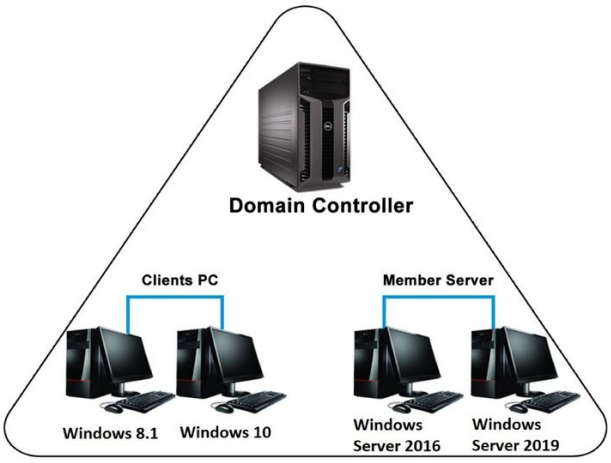


Figure 2.5: Illustration of a domain hierarchy [48].

Chapter 3

Related Work

This chapter will cover related work to our thesis. NTNUs need for a secure environment to store and edit files is not unique. Other national and international universities have implemented and published white papers for services similar to what is proposed in this thesis. While these white papers allow insight into some of their design, a lot details remain obscure, as these services are largely closed source. This is most likely due to the security risks associated with publishing source code in such a way that it is available to bad actors.

Services for Sensitive Data (TSD)

TSD is an expansive digital infrastructure developed and maintained by University Centre for IT (USIT) at the University of Oslo (UIO) [49]. Preliminary development and research started in 2008 and the service launched in 2014 [49], and has since hosted over 2000 research projects [50]. UIO allows other research institutions including NTNU to use TSD [51].

TSD is a feature rich service that supports multiple authentication methods, High performance computing (HPC), direct data collection with "nettskjema" and different client operating systems [49]. While these features make it a more powerful tool to the end users, it can create additional security risks, according to a security principle called economy of mechanism. As described by Saltzer and Schroeder in their foundational 1975 paper "The Protection of Information in Computer Systems" economy of mechanism means "Keep the design as simple and small as possible. This well-known principle applies to any aspect of a system, but it deserves emphasis for protection mechanisms..." [52]. Simply put, a larger set of features mean a larger surface for malicious actors to exploit. For example as TSD has two methods of authentication a malicious actor would only need to compromise one of them to gain unwanted acces to the system.

As mentioned previously NTNU has access to TSD, however this access is limited to only 60 projects that NTNU has purchased from UIO [51]. This limited

number of projects is most likely not enough to support NTNUs needs in the long run, as NTNU is the largest university in Norway, with many students studying in fields like medicine or psychology that regularly deal with sensitive personal information. In addition, we have heard anecdotally from staff at NTNU that TSD projects can be difficult to start and manage. This is supposedly due to projects having to be manually reviewed before being created, and a lot of project managment having to be done through administrators at USIT. The solution proposed in this paper and future efforts by NTNU should seek to learn from and remedy these issues.

Secure Access to Research Data and E-infrastructure (SAFE)

SAFE is a service for research groups to store and edit sensitive research data. It was developed by the IT department at University of Bergen (UIB) and it is available to faculty, researchers and students at UIB. Details about the implementation of SAFE are limited, and most of our information is gathered from a open document describing SAFE and its use [53].

SAFE functions by having projects, and each project has its own infrastructure. SAFE projects has two kinds of members: project managers and project users [53]. Managers can request a project using UIB's ticket system. A SAFE administrator will then set up a project for them. This seemingly requires a decent amount of manual work for the administrator, and may be very tedious if SAFE projects are created frequently. Project managers are also responsible for creating an access document that contains the name, user ID, cell phone number, and access level of all project members. This document is submitted along with the ticket to create the project, and it is resubmitted if they want to change the access level of a project user. It is clear that this requires a lot of work from both the SAFE administrators and project managers [54].

SAFE utilises a fairly simple architecture and can be considered a less feature rich solution compared to TSD. The architecture primarily consists of a single terminal server and two file servers, one outside the project network and one inside it that is connected to the terminal server [53]. The terminal server can host multiple RDP sessions concurrently. To gain access to the terminal server, users connect through a VPN and log in using their user name and password as well as a one time code from their phone. To upload files to the project, any project member can place files on the external file server these files will then periodically automatically be transferred from the external file server to the internal one.

It is unclear from the publicly available documentation how users authenticate to upload files to the file server and how files are encrypted while stored and transferred. To export files you have to be granted access by the project manager, files are then placed in a special folder and similarly to importing files they are

periodically transferred to the external folder. When files are exported they are encrypted using AES-256 with a password that is generated and placed in the internal export folder [53]. The project owner then has to collect the files from the external file server and manually decrypt them with the generated password, probably using a program like 7-Zip. This does indicate that users are required to manually encrypt files when importing them, which could be a problem as user errors or laziness could lead to files being weakly or not encrypted at all on import. Enforcing encryption and doing it automatically is therefore probably preferable.

Chapter 4

Requirement Specification

This chapter will define and describe all requirements for the service developed in this bachelor project. Every requirement will be categorised as either functional or non-functional. A functional requirement is a specific requirement to the service and refers to pure functionality of the service [55]. A non-functional requirement does not refer to specific service functionality, but rather to the performance of the service [55].

4.1 Description of our Service

Before identifying all requirements, it is necessary to know how our service can be used by whom.

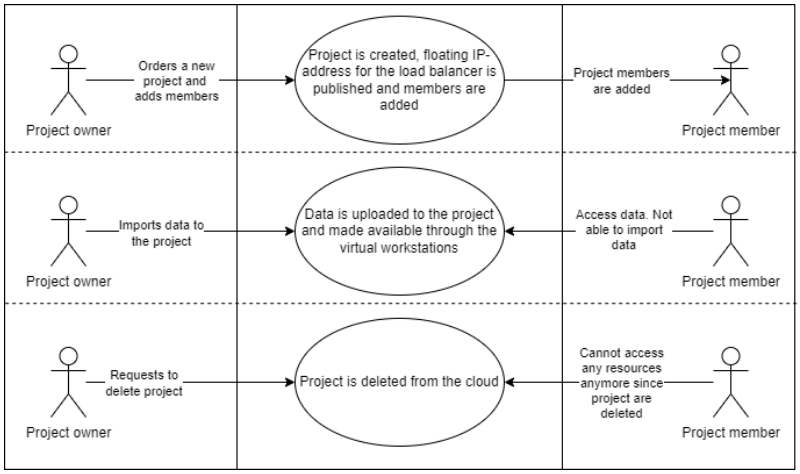


Figure 4.1: Graphical description of different use scenarios provided by our service

When a student/professor wants to share data with others, he/she requires a secure environment for managing and sharing his/her data. This could also apply if the student/professor is a project member to a project and wants to access project data outside of NTNU. In these cases, our service can be used.

To create a project environment on our solution, a project owner can use the web interface of the service to do it and add users (i.e, project members) to the project. An infrastructure for the project will then be booted on a cloud, with a number of virtual workstations that is equal to the number of the project members. All members will be allowed to access the project environment and an IP address will be provided for them to connect to the virtual workstations on the cloud.

When a project is set up, users will be able to use the virtual workstations situated on the cloud platform. They will then be able to manage data stored on the platform based on their rights. Some users will be able to read, write and extract data while others may only be able to read the data. The project owner should be able to import data to the system. When the resources allocated to the project is no longer needed, the project owner can delete the whole project from the cloud.

4.2 Requirements

In this section, requirements for functionality, performance and requirements specified by NTNU will be listed and described.

4.2.1 Functional Requirements

FR1: A project owner should be able to add project members while requesting to create a project
A project owner should be able to add members to a project on a graphical user interface before the project is created. Members are added by writing the university email of each member.

FR2: The project owner should specify themselves to the owner of the project
Before creating a project, the project owner should also write in their email or another email if they is not the owner. This is done so an owner of the project is specified when creating the project.

FR3: A project should be created by a click on a button
A project owner should be able to press a button on the graphical user interface to create a project when all preferred members are added, making them project members.

FR4: A project owner should be able to retrieve the IP address for virtual workstations

A few seconds after a project owner creates a project, the project owner should be able to retrieve an IP address on the graphical user interface by clicking a button to connect to the virtual workstations. This is the address all project members should use to connect to the virtual workstations.

FR5: Project members should be able to connect to virtual workstations
When a member are added to a project and the project is created, the member should be able to use the retrieved IP address to access the project clients and log in with their personal username and password, gaining privileges which were set by the project owner in the graphical user interface.

FR6: Project members should be able to access data
When a project member has connected to a project clients, they should be able to manage data from a file server represented as a network drive on the client. The project member should then be able to execute actions on the shared data based on their privileges granted by the project owner.

FR7: The system administrator should be able to review logs from the project
All actions performed in the project should be logged and presented to the system administrator on a graphical interface, presenting the data in a statistic manner. This logging solution could also be used by the it department of the university/-company that use this service.

FR8: The project owner should be able to delete a project
When there is no longer need for the project, the project and all of its files will be deleted. This is done by using the graphical user interface, specifying the name of the project, and press a delete button.

FR9: The project owner should be able to add files to and download files from the file server
To create a file share that project members can access through the window clients, the project owner should be able to upload files to the file server. To this the need to be authenticated as the owner of the project.

FR10: The system should support RDP session session persistence
If a project members disconnects from a session how they should be connected to the same virtual workstation the next time they log on. That way they are able to pick up their work from where they left off without losing any progress.

4.2.2 Non-functional Requirements

A number of the requirements below is gathered from NTNUs guidelines for access control [11], because this bachelor project mainly is meant to be used by NTNU,

and therefore needs to follow NTNUs requirements. The Norm for Information security and Privacy in the Health and Care Sector [9] will also been referred to since this system has to satisfy requirements for management of health data.

NFR1: Requirements for access to information and information systems
NTNUs guidelines for access control [11] specifies that resource domains should be used for access control where access is granted based on the groups users belongs to. Microsoft Active Directory and LDAP are used for access control in resource domains. The system owner should define roles with specified access control and permissions to storage. There is also a requirements to logging activity related to access and storage.

NFR2: The Health Norms requirements for user insight
The Norm for Information Security and Privacy in the Health and Care Sector [9] specifies in Section 4.2.3 that any individual which has his health data stored should have the possibility to get insight in his/hers own data.

NFR3: Access control for source code
The source code for the system should be available for personnel with correct permissions. Access and changes to source code must be logged. This is in accordance to NTNUs guidelines for access control [11].

NFR4: The system should satisfy NTNUs requirements for the highest level of confidentiality
If leaked information could result in remarkable damage for interests, NTNU, persons or collaboration partners, the information should be classified as strictly confidential [10]. Since information stored in this system could contain details about peoples health, business secrets, etc, it is important that information in this system should be treated as strictly confidential. Information should only be available for users with strictly controlled permissions and that are required to have access to the information.

NFR5: The system should satisfy NTNUs requirements for the highest level of integrity
It is highly crucial that the data in the system always is authentic and correct. Compromised data could result in wrong decisions, errors in treatments if the data is related to health, or constructing errors if the data is related to building plans. This are just a few of many examples. It is therefore crucial that the system satisfy NTNUs classification of integrity for level 4 [10].

NFR6: The system should satisfy NTNUs requirements for the highest level of availability
Since information stored on the system could be highly important, it is crucial that it delivers extremely good availability. Very short down times could be cata-

strophic since the system may contain time sensitive data. Therefore, the system must satisfy NTNUs classification of availability for level 4 [10]. It is also crucial that the system delivers good availability in form of quick responses in retrieval of the clients IP address and creation of the project infrastructure. Because of this high prioritisation, the infrastructure should be fixed immediately if a problem occurs.

Chapter 5

Technical Design

In this chapter, the design and functionality of our platform will be explained. It will also explain the system architecture and its sub-components.

5.1 System Architecture

Figure 5.1 illustrates the current system architecture of the platform for secure data management. As shown the platform is hosted on SkyHiGh in its entirety, and it is only available to users on NTNUs internal network.

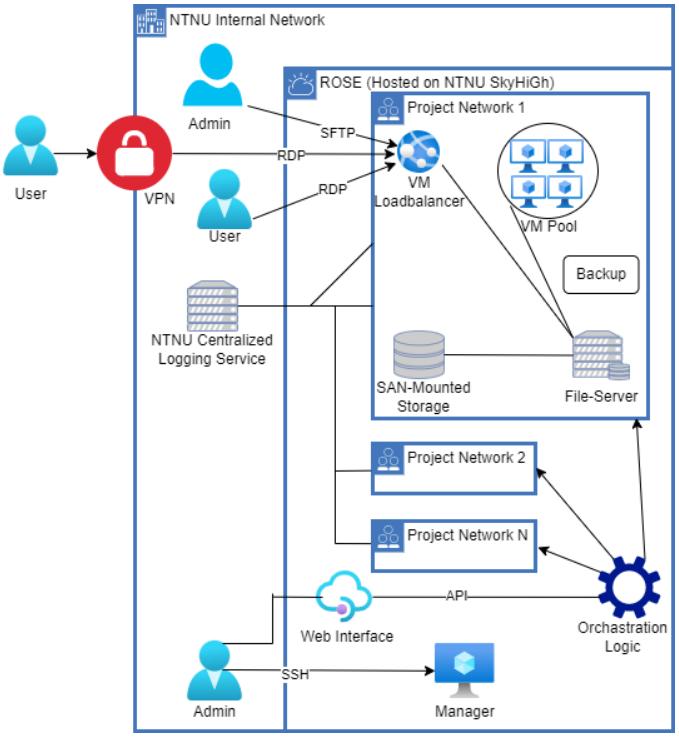


Figure 5.1: Overview of technical design

5.1.1 Network Design

SkyHiGh is hosted on NTNUs internal servers, as such any virtual networks created in OpenStack are only available through NTNUs internal network. Our design consists of two types of networks: *administrator network* and the *project network*. There is only one administrator network, and it is responsible for hosting the orchestration API that creates and manages projects. Each project has its own project network, that contains the domain controller, Windows clients and file server, for project owners and project members to connect to.

Both the administrator network and the project networks are private *192.168.0.0/24* networks that are completely separate from each other. In other words they are only connected to the NTNU internal network through a virtual router and not each other in any way. An alternate solution would be to have each project network as a subnetwork of a larger network, or to have each project network be connected to the admin network through via a router. However our solution of completely separate identical networks offer several benefits as listed below.

Benefit 1: Security
Having networks that are completely separate from each other means that if an attacker somehow gains access to one network, it offers them no advantages in trying to compromise another network. As all network traffic between project or admin networks would have to be routed through the NTNU internal network anyway.

Benefit 2: Scalability
With each network using the same *192.168.0.0/24* address space, we could have an infinite number of project networks each containing up to 250 clients without running out of IP addresses. This theoretical infinite scalability would of course in practice be limited by SkyHiGh's resource constraints.

Benefit 3: Simplicity
When the project networks are totally logically and physically separate, it makes creating managing and deleting projects considerably more simple. For example, if each project network was just a subnetted part of a larger network, we would need to keep track of the subnetworks that had already been allocated to a project. When creating projects, the subnetwork that was being used would need to be marked as "USED", and conversely when deleted it would need to be marked as "FREE". This data would then need to somehow be persistently stored and checked whenever a new project was created.

5.1.2 API

An important part of any interconnected system are APIs. An API is the medium between the user and the system itself and handles all of the requests of the user

and initiate the process of creating a project that the user can use. The API is separated into two logical units: The API itself, which handles the requests, and the orchestrator, which interact directly with OpenStack and the infrastructure. These services are hosted in a docker-container running Ubuntu to simplify the deployment process, accessible only with a floating IP.

5.2 Components

5.2.1 Virtual Workstations

When a project is created on the system, virtual workstations are created based on the number of project members. A project member then use the Remote Desktop Protocol to connect and log in to these virtual workstations with one IP address. All workstations run the *Windows 10 21H2 Enterprise (Evaluation)* operating system. All project members should have their private username and password, which they use as login credentials for the virtual workstations. The virtual workstations are then used to access data that are imported to the project storage through the file server.

Access Control

Access to the workstations and permissions related to project data are all decided by access control mechanisms. When creating a project, project members are specified through their NTNU mail address. The virtual workstations are then added to NTNUs Active Directory tree, adding NTNU users with corresponding mail addresses to the virtual workstations. This means that only the specified NTNU users are able to log in to the virtual workstations. In a filnal version of the platform, the project owner would also has the opportunity to decide permissions for different users on the virtual workstations by adding them to different groups with different group policies. The permissions decide what the users are allowed to do with the data uploaded to the project. There are groups for the following permissions:

- Read
- Read and write
- Read and extract
- Read, write and extract

5.2.2 Storage and File Server

The data and information each project wants to work on need to be stored on a secure location and needs to be quickly accessible. When a project is created, a Linux server and a storage volume is created inside the project network. The file server is a *Ubuntu Server 18.04 LTS (Bionic Beaver) i386* image from SkyHiGhs selection of instances, and the storage volume is a block storage device that is directly attached to the file server, giving it additional volumes of storage. The

reason for choosing a Linux instance for our file server, is that we want the server to be quickly accessible for the project administrator to upload their data. The image chosen is of small size with a minimal amount of bloatware preinstalled, which makes it faster to start up.

Samba

To allow the Windows clients to access the data stored on the file server, a Samba share is created on the file server after it is generated. This Samba share will allow computers located inside the network to connect to the file server to access the data uploaded.

Other options for file sharing

When we explored what technologies to use for file sharing, the conclusion quickly landed on using the Samba service. Since platform-users are planned to be students from different faculties with varied knowledge and abilities to use the command line, we decided that software with a Graphical User Interface (GUI) was a strong requirement. We found no other service that could provide the same level of availability and speed with an easy to use graphical interface.

5.2.3 File Imports and Exports

Without a method to import and exports files to and from the file server, it is useless. Any method used for this purpose needs to support two key features; encryption while files are in transit, and authentication. It is for this reason we chose to use SSH File Transfer Protocol (SFTP).

SFTP

SFTP is as the name suggests a file transfer protocol that leverages SSH. Like SSH it supports public key authentication, where the public key of the person who wishes to upload files to a server is placed on said server. Now if and only if a person has the corresponding private key they are allowed to upload files the server. In our design the public key is supplied by the project owner when creating the project, and placed on the file server. Public key authentication works because it is impossible to generate the private key from the public key. Therefore if someone is able to decrypt data that has been encrypted using the public key, they prove that they are the owner of the private key [56].

After the public key authentication has succeeded an encrypted channel is opened, and data can be sent securely. SFTP is originally a command line program, but the protocol has been integrated with some graphical programs like FileZilla. A benefit of using FileZilla is that NTNU already has a guide on how to use SFTP with FileZilla [57].

While SFTP does support password authentication, it can be configured to only work with public key authentication, as we will. This offers two major security benefits; no insecure user passwords and no possibilities password leaks.

Users are notoriously bad at generation passwords. Short or simple passwords as well as password reuse are common issues. By using public key authentication weak passwords are entirely avoided. Instead a unique 2048 bit key that is impossible to brute force or guess is used.

When using password authentication they have to be transported and stored securely to avoid password leaks. This can be very difficult to implement successfully and it is generally a bad idea to make your own implementation of password management. Creating a solution where user generated passwords are used for sftp authentication on the file server, while guaranteeing the confidentiality of said password would be infeasible for us. Conversely when using public key authentication there is no need to protect the public key, as it can be shared freely without concern of exposing the private key.

There are some disadvantages to using sftp, the main one being difficulty of use. The public key has to be generated using a terminal. Although it only takes a single command that can be copied and pasted, it might be an issue for many users, especially because the primary user base for the platform is non IT studies.

5.2.4 Load balancing

The system offers virtual workstations for project members, the amount of workstations is based on the number of project members. This pool of workstations can then be placed behind a load balancer, which is given a floating IP. In order to connect to the virtual workstations, the floating IP address of an load balancer is used. The file server is also placed behind the load balancer. The load balancer will then redirect connections based on the protocol, if it is SFTP it will be redirected to the file server, and if it is RDP it will be redirected to one of the virtual workstations.

This solution results in one universal IP address for a project, that is used for both RDP sessions to the workstations and for the SSH File Transfer Protocol (SFTP) for the file server. This means that the load balancer IP address is the only information the project owner and project members has to know, making use of the platform simpler for the end users. It would also make a future integration with dns simpler.

Another benefit this solution provides is conservation of floating IP addresses. Floating IPs are needed to connect to a SkyHiGh resource from the NTNU internal network. The floating IPs in SkyHiGh are analogous to public IP addresses on the

internet, however instead of being actual public IP addresses they are private IPs on the NTNU internal network. As such they need to be used somewhat sparingly in order to conserve the limited amount of IP addresses on NTNUs internal network. The administrators of SkyHiGh therefore limit the amount of floating IPs each OpenStack project is allotted, for our project we were limited to 50 floating IPs. By using a load balancer that functions as a proxy for the file server and the windows clients we do not need to give each client or file server their own floating IP, this greatly reduces the amount of floating IPs we use. The logic of the load balancing infrastructure is illustrated in Figure 5.2.

The technology used for the load balancing service is the OpenStack Heat Octavia [26] load balancer. OpenStacks Octavia load balancer makes it possible to define listeners that filter and redirect incoming traffic based on the packet's metadata, it also has features to support session persistence and client health checks. Using OpenStacks own load balancer instead of integrating a generic load balancer like HAProxy offers the benefit of not having to create a separate instance to host the load balancer. This VM would be a single point of failure and it would be more difficult to configure as we would have to inject configuration files at boot, compared to the Octavia load balancer which can be configured entirely through HEAT templates.

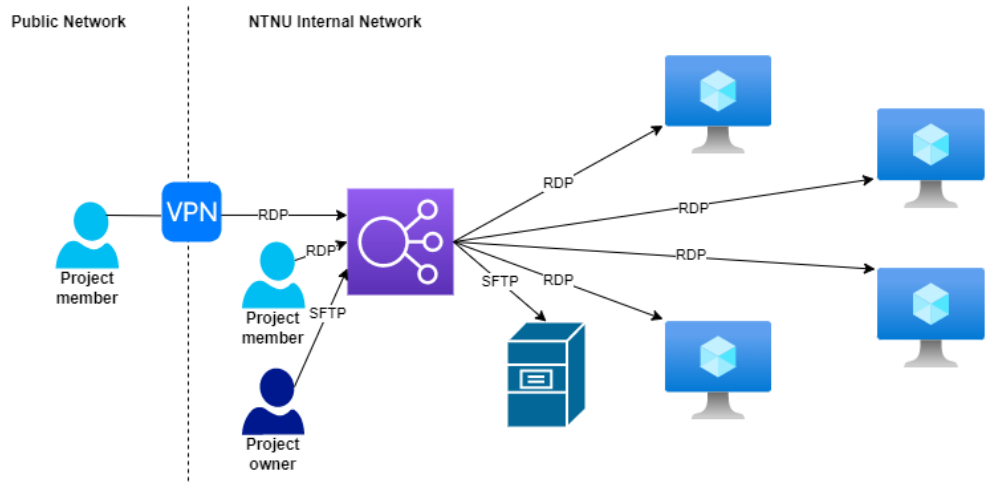


Figure 5.2: Load balancer logic

5.2.5 Logging Service

For a system like ours that is supposed to handle highly sensitive personal data logging is extremely important to ensure accountability, and as a way to detect security events. In a final version of this system logs produced should be pushed to NTNUs centralized logging service. We were not given access to this logging service and instead we implemented a rudimentary logging server using Grafana

Loki. Due to time constraints, we were only able to implement logging of creation, deletion, and collecting of IPs of the projects. In a final design, Windows event logs and file access logs should definitely be included.

5.2.6 Active Directory

The entire system is supposed to be integrated with NTNUs Active Directory for authentication and authorization, to give NTNU control over who gets to access to the different projects and their data. Due to our limited access to NTNUs AD, we had to create our own solution used for testing and as a POC. This is implemented on each individual network for simplicity.

Chapter 6

Development Process

This chapter covers the teams developments process and approach for the suggested solution. First comes the description of the development model used. After that comes the documentation of the project and the routines used when developing the solution.

6.1 Development Model

At the start of the project, in January, when creating the GANTT chart seen in the project plan in appendix A, the group tried to plan out the whole timespan of the project. None of the groups members had taken on a project of this size before, but the group was confident that the suggested time schedule would suffice for the project scope. The proposed GANTT chart gave enough leeway if unexpected issues arose throughout the semester that would use up our time. The group chose to use the development model Kanban [58] when developing the different compartments of the solution. Kanban is a Agile development model but differ from other agile models by the absence of clear iterations in the development process. In kanban, work is distrubuted in a kanban board shown in Figure 6.1.

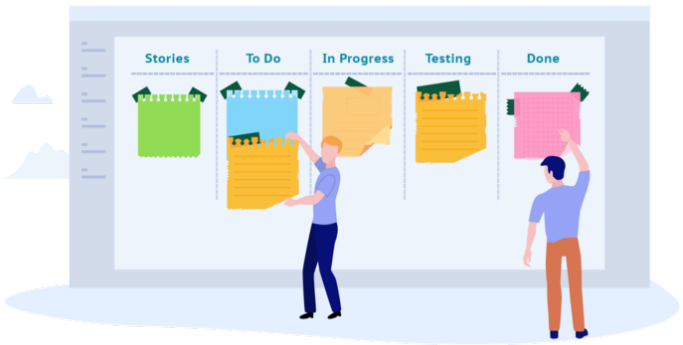


Figure 6.1: Kanban board [59].

The kanban board helps visualize the work that needs to be done, and maximizes efficiency. The team uses Gitlabs [60] built in issue board to manage our tasks, this gives us a clear overview over what we are working on, and what's remaining. It was decided to have a kanban board with four sections when developing the solution:

- Open (tasks available for group members to take on)
- In progress (tasks being worked on)
- Peer review (tasks ready to be peer review by the other team members)
- Closed (tasks successfully peer review and finished)

6.2 Documentation

Report writing

The report is written using the open source document preparation tool LaTeX [61]. LaTeX is the recommended academic writing tool since it gives the user a great deal of customization and control when writing a document.

Source code

The group uses NTNU Gjøvik's Department of Computer Science (IDI) internal GitLab server to upload and keep track of the source code for the solution. This made it easy to keep track of version control when multiple group members worked on the same file at the same time.

Status meetings

From early January, the group agreed with supervisor Kelly to have weekly meeting with her to keep information flow and progress updates regular. This is beneficial for the group and Kelly as she can give constructive feedback more regularly and she stays up to date with the progress the group does. The group also wanted to keep NTNU IT up to date and wanted the ability to discuss ideas and solutions.

Meeting references

As the weekly meetings with Kelly were our main meetings throughout the project period, this is where most of our meeting references stem from. Ahead of our meetings with Kelly, she requested that we prepared a short note with what we had done since last meeting, what problems arose, and if we were still on track with our Gantt chart from the project plan. As for meetings with Eigil, the topic of these were more related to the functionality of the platform. All the feedback from these meetings were well documented, and acted as great guidance when working on out platform.

Time management

As for time management and logging, the group decided on Clockify [62] for keeping track of what task each group member was working on, and how much time was spent each day / week. Early on, in the project plan, the group concluded that each group member should work approximately 30 hours each week, and using Clockify made it easy to keep track of that.

6.3 Routines

From early on in the project, we agreed to follow certain routines both internally within the group and also with Eigil and Kelly.

6.3.1 Tools

Communication

When communicating with other group members a Facebook Messenger [63] chat was created. Digital meetings within the group was held using the free VOIP platform Discord [64]. When having meetings with Kelly, the meetings were held on the online collaborative platform [65], Teams was also used to communicate and share files with both Eigil and Kelly.

Work management

As stated earlier in the report we used GitLab's integrated Kanban board to keep track of the tasks needed to be done.

Report

The report was written using the website Overleaf [66]. This website was chosen because it allow for simultaneous collaboration for all group members, and NTNU has an agreement with Overleaf that provides all students with premium features.

Chapter 7

Implementation

This chapter will cover the groups process of implementing and developing the platform.

7.1 API

The API is handling all of the requests sent by the user. The API is the first point of interaction between the user and the infrastructure, and initiating the process of creating a project by retrieving and validating request parameters and sending valid data to the orchestator.

7.1.1 Flask

The API is implemented in python and the Flask web framework. Flask makes it easy to create endpoints which can be used by a user. To create a route, all we need is to add the Flask.route method as a decorator, as seen in Code listing 7.1.

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route("/") # Create route
6 def hello_world(): #Handle route
7     return "<p>Hello, World!</p>"
```

Code listing 7.1: Basic endpoint for Flask [1]

7.1.2 Handling Requests

By using the method as seen in Subsection 7.1.1 we could create the needed endpoints seen in Code Listing 7.2.

```
1 @app.route('/api/new-project', methods = ['POST'])
2 def newProjectEndpoint():
3    requestData = request.get_json() # JSON data
4
```

```
5     if(requestData): ## Theres json in req. body
6        emailsOfUsers = requestData.get('users') ## Email of clients
7        ownerMail = requestData.get('owner')
8        #print(str(emailsOfUsers))
9        if(emailsOfUsers and ownerMail): # Make sure 'users' exists.
10          try:
11             return project.newProject(emailsOfUsers, ownerMail), Const.HTTPOK
12          except Error.OutOfRangeError as err:
13             return err.message, Const.HTTPBADREQUEST
14          except Error.InvalidEmailError as err:
15             return err.message, Const.HTTPBADREQUEST
16          except (TypeError, ValueError) as err:
17             print(err.args)
18             return "Malformed request", Const.HTTPBADREQUEST
19          except Error.ProjectCreateFailedError as err:
20             return err.message, Const.HTTPINTERNALERROR
21          except Exception as err:
22             return "Something went wrong :(", Const.HTTPINTERNALERROR
23
24       else:
25          return "Missing parameters", Const.HTTPBADREQUEST
26    else:
27       return "Missing body", Const.HTTPBADREQUEST
```

Code listing 7.2: The code handling the HTTP POST request for the /api/new-project enpoint

This function handles all HTTP POST requests requests to the */api/new-project* path. The main purpose of this method is to make sure that the 'owner' and 'users' parameters are present, and handle any possible exceptions thrown because of a request with wrong parameter types. The parameters are sent to the the newProject function. The name of the project along with a 200 HTTP OK is returned to the user, indicating that everything went well. If there was an issue, the user would be given an error message with a relating status code, like the 500 HTTP Internal Error or the 400 HTTP Bad Request.

```
1  def newProject(userMails, ownerMail, publicKey):
2
3      for mail in userMails: # Validate all user emails
4          if not Util.validateEmail(mail):
5              raise Error.InvalidEmailError(mail)
6
7      if not Util.validateEmail(ownerMail):       # Make sure that the owner email is
           ↪  valid
8          raise Error.InvalidEmailError(ownerMail)
9
10     owner = Util.getUserNameFromEmail(ownerMail) # Get username out of email
11     if not Util.validateUsername(owner):
12         raise Error.InvalidUsernameError(owner)
13
14     if not Util.validatePublicKey(publicKey): # Validate the public SSH key
15         raise Error.InvalidPublicKeylError()
16
17     orchestrator = Orchestrator()
18     name = orchestrator.createProject(userMails, owner, publicKey) # Create
19
```

```
20      if name: # Ok
21          comfirmProject.delay(name)
22          return name
23      else:
24          raise Error.ProjectCreateFailedError
```

Code listing 7.3: The logic initializing a new project

Code Listing 7.3 makes sure that the emails are validly formatted using regular expression, including the owner of the project, and interacts with the orchestrator. Resulting in the creation of the project network and the client workstations. The function returns the name of the project, which again is returned to the user by the API. This name is used at different endpoints, like retrieving the IP address that users can connect to workstations, and to delete the project later.

```
1  @app.route('/api/project/<name>', methods = ['GET'])
2  def getIPEndpoint(name):
3     if (name): # Is present
4        try:
5           return project.getIP(name) , Const.HTTPOK # OK
6        except Error.ProjectNotFoundError as err:
7           return err.message, Const.HTTPNOTFOUND
8        except Exception as err:
9           return Const.HTTPINTERNALERROR
10
11    return Const.HTTPBADREQUEST
```

Code listing 7.4: The code handling the HTTP GET request for the /api/project/<name> enpoint

Code Listing 7.4 handles all HTTP GET requests to the */api/project/<name>* path. This endpoint returns the floating IP for a given project to the user. This IP is then later used to access the workstations.

```
1  @app.route('/api/project/<name>', methods = ['DELETE'])
2  def deleteProjectEndpoint(name):
3     if (name): # Present
4        try:
5           return project.deleteProject(name), Const.HTTPOK # Delete Ok
6        except Error.ProjectDeleteFailedError as err:
7           return err.message, Const.HTTPINTERNALERROR
8        except Exception as err:
9           return Const.HTTPINTERNALERROR
10
11    return Const.HTTPBADREQUEST
```

Code listing 7.5: The code handling the HTTP DELETE request for the /api/project/<name> enpoint

Code Listing 7.5 handles all HTTP DELETE requests to the */api/project/<name>* path. This endpoint deletes a a project based on project name, when it is no longer needed. The name is validated to see if the name actually exists. Authenticating the DELETE-request is considered out of our scope.

```
def validateEmail(email):
    #https://www.geeksforgeeks.org/check-if-email-address-valid-or-not-in-python/
    regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
    if(re.fullmatch(regex, email)): # Is valid email
        return True
    return False
```

Code listing 7.6: Function making sure all emails are validly formated

All input provided by a user has to be validated, with no exception. The reason for this is that the user might misspell something, or the input could be malicious in an attempt at attacking the service and its users. As seen in Code Listing 7.6 the email is validated using regular expression (regex) borrowed from the website GeeksForGeeks, a popular website in the computer science field.

7.2 Orchestration Logic

The orchestration logic is written in python and communicates with the OpenStack API to create, delete and manage project stacks that are defined using HEAT templates.

7.2.1 Orchestrator

The core of the orchestration logic is the Orchestrator class defined in *orchestrator.py* file. At the creation of a orchestrator object the Orchestrator constructor is automatically called. This constructor initializes the *connection* object from the OpenStack python module. The *connection* object is an authenticated connection to the OpenStack API. It is initialized in the constructor because it takes some time and this way the same connection can be used multiple times, saving time and computing resources. Code listing 7.7 shows the constructor of the orchestrator class.

```
def __init__(self):
    # Initialize and turn on debug logging if wanted
    openstack.enable_logging(debug=False)
    # Initialize connection
    self.conn = openstack.connect()
```

Code listing 7.7: Orchestrator constructor method

Once the *connection* object has been initialized it is possible to make OpenStack API calls by calling its methods. This is illustrate in Code Listing 7.8, here the method gets the IP address of the load balancer for a given project, which can then be used to connect to all the virtual workstations. It takes the project name or id and as a parameter and returns the IP address if the project exists. Notice that the resolve output parameter is *True* because the load balancer IP address is defined as an output in the HEAT template.

```
# returns None if something goes wrong in the query or if the stack does not exist
# Else the IP is returned
def getLBIP(self, nameOrID):
    try:
        stack = self.conn.get_stack(nameOrID, resolve_outputs=True)
    except:
        logger.exception("Api call on get_LBIP for raised an exeption", extra={"
            ↪ tags": {"project": nameOrID}})
        return None

    if stack:
        logger.info("Loadbalancer IP {} was collected".format(stack.outputs[0]["
            ↪ output_value"]), extra={"tags": {"project": nameOrID}})
        return (stack.outputs[0]["output_value"])
    else:
        logger.error("Attempted to get ip of loadbalancer that does not exist",
            ↪ extra={"tags": {"project": nameOrID}})
        return None
```

Code listing 7.8: Method for getting the IP address of a project load balancer

The two methods responsible for creating and deleting projects are *createProject()* and *deleteProject()*. As shown in Code Listing 7.9 below, *createProject()* creates a random name for the project and calls the *create_stack()* method on the *connection* object. It passes the name of the HEAT file the stack is based on (*project.yaml*), the project name, the number of windows clients, the project owner, the project owner's public key, and a Powershell script *RDPMembers* that allows supplied project members to RDP into the clients. All these variables are defined as these are defined as parameters in the HEAT file, and their values are what makes one project different from another.

If the stack create OpenStack API call succeeds it returns the name of the newly created project. Similarly the *projectDelete()* method takes the name of the project that is to be deleted and passes it on to the *delete_stack()* method of the *connection* object. If the OpenStack API call succeeds True is returned if not False is returned.

```
def createProject(self, members, owner, publicKey):
    name = self.createRandomName()

    # ensures client count is positive
    # stops user from making more than 15 clients
    clientCount = len(members)
    if clientCount > 15:
        clientCount = 15

    RDPMembers = self.allowRDP(members)

    try:
        self.conn.create_stack(name, template_file='orchestrationLogic/HEAT/project
            ↪ .yaml', rollback = False, project_name = name, key_name = 'mankey',
            ↪  windows_count = clientCount, project_owner = owner, public_key =
            ↪ publicKey, RDP_members = RDPMembers, tags = owner)
    except:
```

```
15          logger.exception("Stack␣create␣API␣call␣failed", extra={"tags": {"owner":
                ↪ owner}})
16          return None
17
18      logger.info("Project␣create␣api␣call␣succeeded" , extra={"tags": {"owner":
            ↪ owner, "project": name}})
19      return name
20
21  def deleteProject(self, nameOrID):
22      try:
23          result = self.conn.delete_stack(nameOrID)
24      except:
25          logger.exception("Stack␣delete␣API␣call␣failed.", extra={"tags": {"project"
                ↪ : nameOrID}})
26          return False
27      if result:
28          logger.info("Project␣delete␣api␣call␣succeeded", extra={"tags": {"project":
                ↪  nameOrID}})
29          return True
30      else:
31          logger.error("Attempted␣to␣delete␣project␣that␣does␣not␣exist", extra={"
                ↪ tags": {"project": nameOrID}})
32          return False
```

Code listing 7.9: Create and delete project methods of orchestrator

7.2.2 Background Tasks

Even if *create_stack()* succeeds it does not necessarily mean that the project infrastructure will be created, equally if *delete_stack()* succeeds the project infrastructure is not guaranteed to actually be deleted. It simply means that the stack has entered a status of *CREATE_IN_PROGRESS* or *DELETE_IN_PROGRESS*, in other words it has been scheduled for creation or deletion but the task has not actually completed. However it may take several minutes before the create or delete actually completes and they may never complete, due to several possible reasons like running out of resources in OpenStack or internal errors.

This is an issue because the project owner who ordered the project needs to be given a response within 30 seconds, as that is the HTTP timeout limit, and the project owner should obviously be given a response way before that for a good user experience. However we can not simply trust that a project create or delete completes. Because if a create fails the parts of the project infrastructure that were successfully created need to be deleted, and if a delete fails the incident needs to be logged so that an admin can step in. Hence the need for background tasks, we implemented background tasks using celery which allows us to ensure the creation and deletion of projects and log any errors. An overview of what tasks are run in the foreground and background can be seen in Figure 7.1.

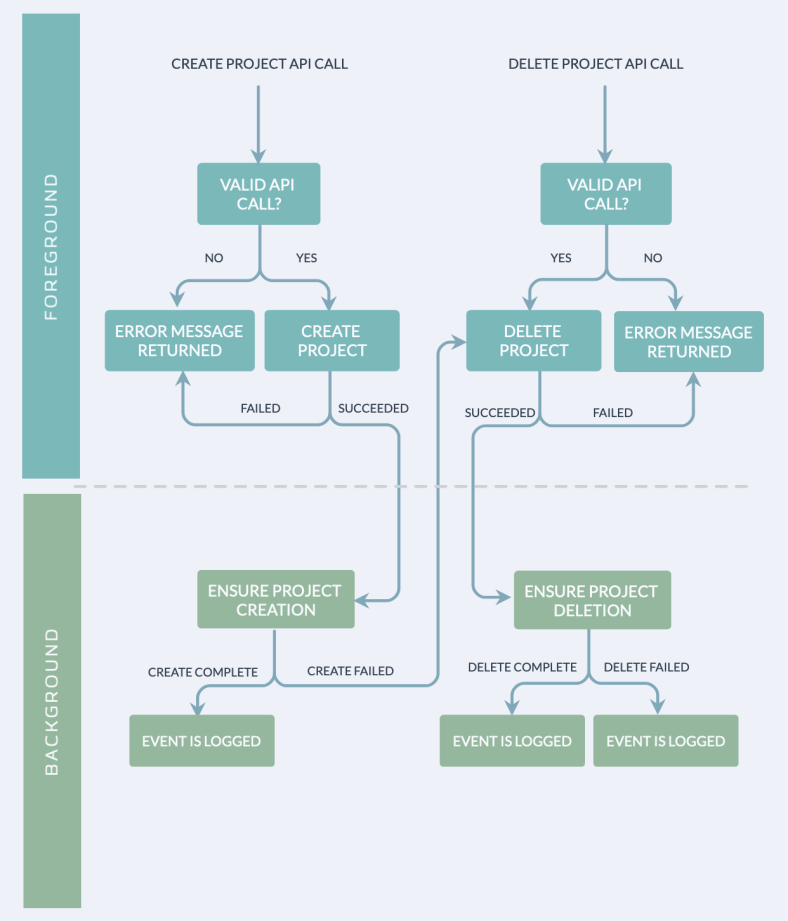


Figure 7.1: Graphic describing the orchestration logic

Celery needs to be imported and configured at the root of the Python project. Celery background tasks can then be defined using the *@celery.task* Python decorator, as demonstrated in Code Listing 7.10, they can then be called using the *.delay* method. The minimal requirements for Celery to work is to have a task queue database and a Celery worker. The Python application will add Celery tasks to the task queue and the worker will fetch the tasks and run them. More information on how we configured Celery, the task queue database, and the Celery worker will be given in Section 7.5.

```
@celery.task
def comfirmProject(nameOrID):
    orchestrator = Orchestrator()
    orchestrator.ensureProjectCreation(nameOrID)
@celery.task

def comfirmDeletion(nameOrID):
    orchestrator = Orchestrator()
    return orchestrator.ensureProjectDeletion(nameOrID)
```

Code listing 7.10: Defining Celery task

7.2.3 Logging

Every action done by the Orchestrator is logged. As mentioned previously, a final version of our platform would be connected to NTNUs centralized logging server. We were not given access to this server and we therefore implemented our own simple logging server using Grafana Loki. Grafana Loki are actually two separate servers; Loki that collects data from log sources and Grafana that filters and displays that data. In practice they function as a single server and they will be referred to as if they were from now on, for simplicity's sake. We did not bother to secure our logging server by using communication over SSL and proper authentication, as the server would not be used in a final product, its primary purpose for us is debugging and as a proof of concept. The Grafana Loki logging server has two exposed ports *3100* and *3000*. The former is used to push logs to the server, the latter is used to filter and view logs through a web GUI. More details about the logging server will be given in Section 7.5.

Python does not support Grafana Loki out of the box, so the module *logging_loki* has to be imported. A custom log handler and a *logger* object can then be created. We decided implement our custom logging as a python module, meaning that we created a file *customLogger/__init__.py*. The *customLogger* module can then be imported anywhere and the *logger* object can be used to push logs to our logging server. Grafana loki supports a lot of useful features like tags and log levels, making it easy to filter and analyze logs. We always sent the project name as a tag so we could easily see exactly what happened to a project and when it happened.

Figure 7.2 show the Grafana Loki web interface. As shown it is simple to filter for a specific project, in this case *united_rook*. Following the log messages we can read that the initial project create API call succeeded, and the loadbalancer's IP address was collected soon after. However the create eventually failed due to our OpenStack enviroment running out of memory. The project was then automatically scheduled for deletion and the log ends with the project being confirmed as successfully deleted.

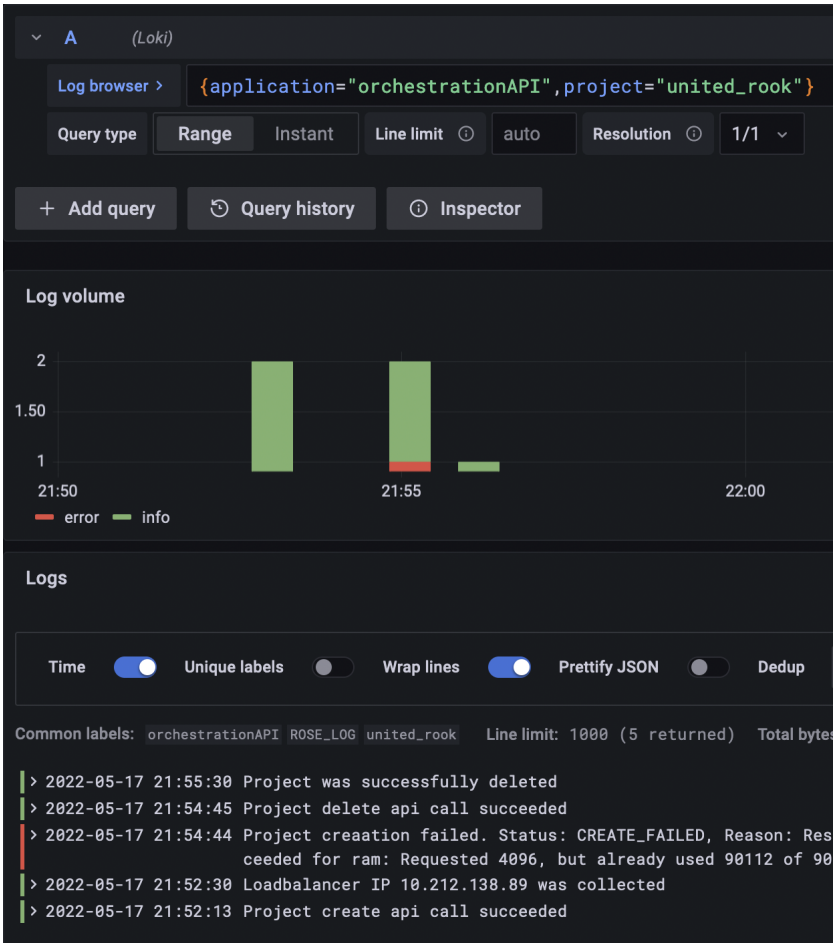


Figure 7.2: Screenshot from Grafana Loki

7.3 Heat Templates

The heat templates are descriptive YAML [67] files that orchestrates and builds different components that together makes up a stack. For us, one stack and its components make up one projects infrastructure. We have implemented our heat templates in such a way that they entirely are responsible for the configuration of the project infrastructure, requiring no manual intervention by administrators, or post boot configuration.

A heat template file is separated into three main sections:
Parameters: Variables passed to the heat file which can be to customize the stack created.
Resources: The objects created by heat as a part of the stack.
Outputs: Values that are passed by heat after creating the stack.

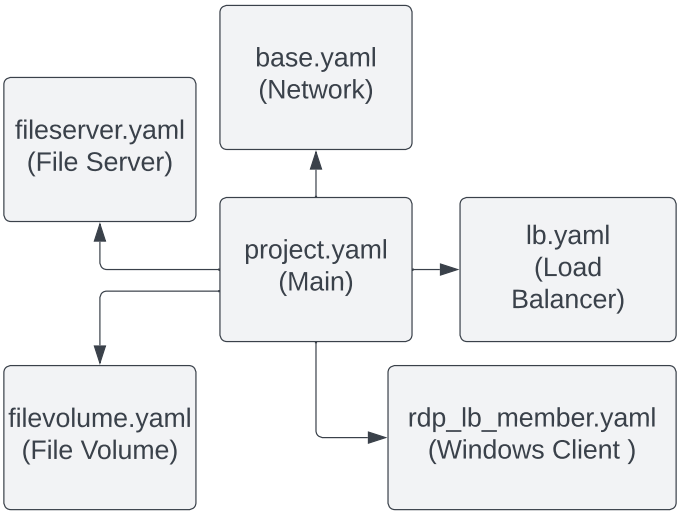


Figure 7.3: Graphic of our heat file layout

We chose to build a nested layout for our heat templates. A nested layout will allow for a more modular composition and the different components can more easily be modified and possibly replaced if needed in future versions. In Figure 7.3 we have illustrated the hierarchical heat file-structure that we chose to use. As we can see in the figure above, the *project.yaml* file is the main component of the structure, which then calls on other yaml files to create different components of the stack. Code Listing 7.11 contains parts of the resource definition in the *project.yaml* file. Here we can see an example of the hierarchical structure, as seen on line 7, when creating the resource **load_balancer**. In the field *type:* we specified that the file *lb.yaml* should be used to create the new resource. The values under *properties:* are parsed as the new *parameters* for the newly created resource group.

```
1  resources:
2    base:
3      type: base.yaml
4      properties:
5        project_name: { get_param: project_name }
6
7    load_balancer:
8      type: lb.yaml
9      properties:
10       project_name: { get_param: project_name }
11       heat_network_subnet: { get_attr: [ base, resource.heat_network_subnet ] }
12
13   clients:
14     type: OS::Heat::ResourceGroup
15     properties:
```

```
16      count: { get_param: windows_count }
17      resource_def:
18        type: rdp_lb_member.yaml
19        properties:
20          server_name:
21            list_join: [ '-', [ { get_param: project_name }, 'win%index%' ] ]
22          lb_pool_name: { get_attr: [ load_balancer, resource.rdp_pool ] }
23          network_name: { get_attr: [ base, resource.heat_network ] }
24          subnet_name: { get_attr: [ base, resource.heat_network_subnet ] }
25          key_name: { get_param: key_name }
26          volume_id: { get_attr: [ filevolume, resource.volume ]}
27  ...
```

Code listing 7.11: Part of resource definition of *project.yaml*

7.3.1 Base

Every project network has the same base network topology. This network consist of a private network with a subnet resource assosiated with it. As seen in Code Listing 7.12 the resources **heat_network**, **heat_network_subnet**, **heat_router** and **router_interface** gets created. All project networks have a network address of *192.168.0.0/24* giving place for over 250 possible devices to connect to. As seen on line 17 the allocation pool for assigning of IP addresses go from *192.168.0.4* to *192.168.0.254*. This is because we have reserved the three first IP addresses for specific devices. The first address is for the default gateway of each network, as seen on line 14 of Code Listing 7.12 on page 46. The second address reserved is for the projects file server, as seen in Code Listing 7.15 on page 50, and the third address reserved is for the domain controller, as seen in Code Listing 7.19 on page 57.
Furthermore the resources **heat_router** and **router_interface** is created. The router is the default gateway for the network and the router interface links each isolated network to NTNUs internal network.

```
1   ...
2   resources:
3     heat_network:
4       type: OS::Neutron::Net
5       properties:
6         admin_state_up: true
7         name:
8           list_join: [ '_', [ { get_param: project_name }, 'NET' ] ]
9
10    heat_network_subnet:
11      type: "OS::Neutron::Subnet"
12      properties:
13        cidr: 192.168.0.0/24
14        gateway_ip: "192.168.0.1"
15        ip_version: 4
16        network: { get_resource: heat_network }
17        allocation_pools: [{"start": "192.168.0.4", "end": "192.168.0.254"}]
18
19     heat_router:
```

```
20      type: OS::Neutron::Router
21      properties:
22        admin_state_up: True
23        name:
24          list_join: [ '_', [ { get_param: project_name }, 'Router' ] ]
25        external_gateway_info: { network: ntnu-internal}
26
27    router_interface:
28      type: OS::Neutron::RouterInterface
29      properties:
30        router_id: { get_resource: heat_router }
31        subnet: { get_resource: heat_network_subnet }
```

Code listing 7.12: Code snippet from base.yaml

7.3.2 Load balancing RDP sessions

We automated the load balancing infrastructure in a Heat Orchestration Template. The OpenStack resource Octavia [26] was used to create the load balancing infrastructure, as well as OpenStack Neutron [21] for creating and associate the floating IP-Address for the load balancer. In order to make the load balancing infrastructure, the following resources had to be added in the Heat Template.

- **Octavia::LoadBalancer:** This is the load balancer itself where it is associated with the project subnet.
- **Octavia::Listener:** Two different listeners were created for RDP and SFTP where both are associated to the load balancer created above. The job of the listener is to listen to traffic of a specific protocol and a specific port. The listener for RDP was set to listen to the Transmission Control Protocol (TCP) and port 3389 which is a TCP and User Datagram Protocol (UDP) port used for the RDP service.
  The listener for SFTP was set to listen to TCP and port 22 which is a TCP and UDP port which in this circumstance is used for safe file transfer.
- **Octavia::Pool:** A load balancer pool is a collection of instances (server resources) which are gathered in one common group [68]. One pool are created for RDP and one pool are created for SFTP. The pools are then associated with the listener for their respective use. The algorithm the load balancer should use to distribute resources to users are set to be round robin [69] and the protocol the resources should operate on is set to be TCP. The property called *session persistence* is set to be of the type *SOURCE IP*. This means that when a user previously has connected to a resource, he/she should be assigned the same machine the next time he/she connects to the resource based on the IP-address of the device he/she used to connect to the resource.
- **Octavia::PoolMember:** Each pool has a set of pool members. For each resource to be load balanced, the *PoolMember* resource is used to assign a resource to a pool. It is then required to use the IP-address of the resource to be balanced, and specify which port a connection should be established on.

- **Octavia::HealthMonitor:** A health monitor is used to determine the health of the load balancer pool members [70]. It is configured to determine how much delay an instace is allowed, how many retries it is allowed and when to time out.
- **Neutron::FloatingIP:** This is the public IP-address that should be used to connect to the load balancer. This IP-address can be reached by the network specified.
- **Neutron::FloatingIPAssociation:** The IP association is used to associate the floating IP-address to the load balancer.

Below is the code that creates the load balancer for a project. Note that the windows clients are not associated in this file. The logic for the pool of windows clients will be described below Code Listing 7.13.

```
heat_template_version: 2018-08-31

parameters:
  project_name:
    type: string
    description: Navn på prosjektet
    default: Prosjekt_X

  heat_network_subnet:
    type: string

resources:
  rdp_balancer:
    type: OS::Octavia::LoadBalancer
    properties:
      name:
        list_join: [ '-', [ { get_param: project_name }, 'RDP_LB' ] ]
      vip_subnet: { get_param: heat_network_subnet }

  rdp_listener:
    type: OS::Octavia::Listener
    properties:
      loadbalancer: { get_resource: rdp_balancer }
      protocol: TCP
      protocol_port: 3389

  sftp_listener:
    type: OS::Octavia::Listener
    properties:
      loadbalancer: { get_resource: rdp_balancer }
      protocol: TCP
      protocol_port: 22

  rdp_pool:
    type: OS::Octavia::Pool
    properties:
      lb_algorithm: ROUND_ROBIN
      protocol: TCP
      listener: { get_resource: rdp_listener }
      session_persistence: { "type": SOURCE_IP }

```

```
42   sftp_poolmember:
43     type: OS::Octavia::PoolMember
44     properties:
45       address: "192.168.0.2"
46       protocol_port: 22
47       subnet: {get_param: heat_network_subnet}
48       pool: {get_resource: sftp_pool}
49
50   sftp_pool:
51     type: OS::Octavia::Pool
52     properties:
53       lb_algorithm: ROUND_ROBIN
54       protocol: TCP
55       listener: { get_resource: sftp_listener }
56       session_persistence: { "type": SOURCE_IP }
57
58   health_monitor:
59     type: OS::Octavia::HealthMonitor
60     properties:
61       delay: 5
62       max_retries: 4
63       timeout: 10
64       type: TCP
65       pool: { get_resource: rdp_pool }
66
67   lb_floating:
68     type: OS::Neutron::FloatingIP
69     properties:
70       floating_network: ntnu-internal
71
72   lb_floating_association:
73     type: OS::Neutron::FloatingIPAssociation
74     properties:
75       floatingip_id: { get_resource: lb_floating }
76       port_id: { get_attr: [ rdp_balancer, vip_port_id ] }
```

Code listing 7.13: The contents of the file *lb.yaml*

The number of windows clients for a project is decided based on the amount of project members. In order to make the load balancing pools dynamic, the windows clients and the pool members resource for the clients are declared in a separate file, rdp_lb_member.yaml. In this file, the windows clients are declared with all required configuration, and also associated as a pool member for the RDP pool.

```
1  resources:
2    server:
3      type: OS::Nova::Server
4      properties:
5        name: { get_param: server_name }
6        flavor: m1.small
7        image: "Windows␣10␣21H2␣Enterprise␣[Evaluation]"
8        key_name: { get_param: key_name }
9        networks:
10         - network: {get_param: network_name}
11       user_data_format: RAW
12       user_data:
13         str_replace:
```

```
14          template:
15            get_file: bootup.ps1
16          params:
17            <RDPMembers>: {get_param: RDP_members}
18
19    poolmember:
20      type: OS::Octavia::PoolMember
21      properties:
22        address: { get_attr: [ server, first_address ] }
23        pool: { get_param: lb_pool_name }
24        protocol_port: 3389
25        subnet: {get_param: subnet_name}
```

Code listing 7.14: Parts of the resources in the file *rdp_lb_member.yaml*

Creating the windows clients pool members from the file above is done in the file project.yaml. This is done by using the OpenStack Heat resource, *ResourceGroup* where the clients with pool member are created a number of times based on the parameter *count*. This can be seen in Code Listing 7.11.

7.3.3 File server

The file server is hosting all the data that the project administrator uploads for students to work on. In order to allow file sharing, certain packages need to be installed, and the Samba configuration needs to be set up correctly.

Base configuration for the file server

As stated in Section 5.2.2, the file server image is that of a Ubuntu server with minimal components preloaded to it. On boot up of the Linux instance, the resource type *OS::Heat::CloudConfig* and *OS::Heat::SoftwareConfig* gets created. The CloudConfig resource represents cloud-init cloud configuration. Cloud-init is a service for customizing Linux-based operating systems [71]. With cloud-init we get complete freedom to customize our file server to our needs. The SoftwareConfig resource is a more general resource for storing software configuration for a wider range of operating systems.

As seen in Code Listing 7.15 on line 7 we specified that the package for samba should be installed on creation. From line 24 to 28 we create a new disk partition in path */dev/vdb*. The reason for creating this disk partition is that we now can manage the separated partition as we please. Further we specify on line 29 that we want to create a file system in the newly created disk partition. With this section we specify the type of file system we want to exist on the partition.

```
1  resources:
2    cloudconf_samba:
3      type: OS::Heat::CloudConfig
4      properties:
5        cloud_config:
6          packages:
```

```
7           - 'samba'
8         write_files:
9           - path: /etc/samba/smb.conf
10            content: |
11                [global]
12                    workgroup = WORKGROUP
13                    server string = Samba Filserver
14                    security = user
15                    map to guest = bad user
16
17                [share]
18                    comment = Samba share
19                    path = /opt/data/shared
20                    read only = no
21                    guest ok = yes
22                    guest only = yes
23                    create mask = 777
24        disk_setup:
25          /dev/vdb:
26            table_type: gpt
27            layout: true
28            overwrite: false
29        fs_setup:
30          - filesystem: "ext4"
31            label: "datapartition"
32            device: "/dev/vdb"
33            partition: "auto"
34
35    script_filserver:
36      type: OS::Heat::SoftwareConfig
37      properties:
38        group: ungrouped
39        config:
40          str_replace:
41            template:
42              get_file: fileserver.sh
43            params:
44              <owner>: {get_param: project_owner}
45              <publicKey>: {get_param: public_key}
46
47    cloudconf_fileserver:
48      type: OS::Heat::MultipartMime
49      properties:
50        parts:
51          - config: { get_resource: cloudconf_samba }
52          - config: { get_resource: script_filserver }
53
54    heat_filserver:
55      type: OS::Nova::Server
56      properties:
57        key_name: { get_param: key_name }
58        name:
59          list_join: ["-", [{ get_param: project_name }, "heat_filserver"]]
60        flavor: { get_param: flavor }
61        image: { get_param: image }
62        networks:
63          - port: { get_resource: heat_port }
64        user_data_format: RAW
65        user_data: { get_resource: cloudconf_fileserver}
66    heat_port:
```

```
67      type: OS::Neutron::Port
68      properties:
69        network: {get_param: heat_network}
70        fixed_ips:
71          - ip_address: "192.168.0.2"
72  ...
```

Code listing 7.15: Cloud config snippet from *fileserver.yaml*

The resource *script_filserver* on line 35 is a bash script which contains more configuration of the file server. A snippet of the contents of the bash file can be found in Code Listing 7.16. The first command of the code listing appends the string "*/dev/vdb1 /opt/data ext4 defaults,comment=cloudconfig 0 0*" to the */etc/fstab* file on the file server. The function of the */etc/fstab* file is to keep track of all disk partitions that are not based on physical disks. With the newly added line in the *fstab* file, the server has now mounted the new disk partition created earlier to the local path of */opt/data*. In the lines following, the directory */opt/data* and */opt/data/shared* is created and the "*mount /dev/vdb1 /opt/data* command connects the new mount to file tree-structure of the server.

```
1  #!/bin/bash
2  ...
3  echo "/dev/vdb1␣/opt/data␣␣␣␣␣␣␣ext4␣␣␣␣defaults,comment=cloudconfig␣␣␣␣0␣␣␣␣␣␣␣0"
       ↪ >> /etc/fstab
4  mkdir /opt/data
5  mount /dev/vdb1 /opt/data
6  mkdir /opt/data/shared
7  ...
```

Code listing 7.16: Bash script snippet from *fileserver.sh*

The complete content of both *fileserver.yaml* and *fileserver.sh* can be found in the appendix.

Samba configuration

As stated in Section 2.1.5, the configuration file of the Samba service is located in the */etc/samba/smb.conf* file. The standard configuration file for Samba is quite long, spanning over 260 lines. For our area of use we can create our own configuration file without all of the unnecessary data. As seen in Code Listing 7.15 from line 11 to 23 this is the new content of the configuration file. In the indented part after the *[global]* section we declare global settings to Samba. Line 12 specifies the windows workgroup that the file server will join. The server string variable on the next line specifies the Samba service name that will show up on file explorer on the Windows clients. The line *security = user* is one of the most important lines in the configuration file. With *user* as the value the clients are expected to log-in with a username and password. This, in combination with the line *map to guest = bad user* will make user who log in with the wrong password log in as a guest user.

On line 17 the new share is defined, this is where project related data will be uploaded by the project administrator. The next line gives a short description of what the share is being used for. The line *path = /opt/data/shared* is the path on the file server the share will point to when clients try to connect to the Samba share. The *read only = no* line determines that users are allowed to not only read, but also write to the files inside the share. This is essential for allowing students to manage the data as they wish. On the next line, the *guest ok = yes* allows guest users to log on to the share without a password. This option is enabled since we were not allowed to integrate our platform with the NTNU AD, and as this is only a proof of concept, user authentication inside the Samba share is outside the scope. The next command; *guest only = yes* allows only guest connections to be established and this, in combination line 14 in the *global* section will force the use of guest users. All configuration of the Samba configuration file are inspired by Ubuntu's website on Samba configuration [72] and Sambas own manual page [73]. The line create mask = 777 gives any user the ability to read, write and execute file that are located in the directory */opt/data/shared*.

SFTP

SFTP is our chosen method for importing and exporting files to and from the file server. Public key authentication is used, meaning that for SFTP to work a user on the file server needs to be created, and the project owners public key needs to be associated with that user. As the name suggest SFTP is closely related to SSH. A user who can SFTP to a server can normally also SSH into it as well. For security reasons only want give project owners SFTP access while disallowing them from actually getting a shell. We also want to limit them to only accessing files inside */opt/data*.

Code Listing 7.17 is an excerpt from the file server boot script that securely configures SFTP. The main actions taken by part of the scrip is:

- Create a user for the project owner with the name they provided.
- Inject the project owner's public key into *.ssh/authorized_keys*, thereby allowing them to authenticate using their private key.
- Create a two groups *allowssh* and *sftponly*.
- Using the *sshd_config* file:
    - Only allow public key authentcation for member of *allowssh*.
    - Change the root directory to */opt/data* for members of *sftponly*. Meaning they cant access any file outside that directory.
    - Disable all forwarding for members *sftponly*.
    - Allow only internal SFTP for members *sftponly*.
- Add the admin user *ubuntu* to *allowssh*.
- Add the project owner user to *allowssh* and *sftponly*, thereby limiting access.

```
# Everything below is based on the great top answer for this stack exchange question
# https://unix.stackexchange.com/questions/503312/is-it-possible-to-grant-users-sftp-access-without-shell-access-if-yes-how-is-i
# adds two groups one fro users who are allowed to ssh (ubuntu) and one
# for user who are only allowed to sftp
addgroup --system allowssh
addgroup --system sftponly

# creates a user with no no password
adduser --disabled-password --gecos "" --home /home/project_owner <owner>

mkdir /home/project_owner/.ssh/
touch /home/project_owner/.ssh/authorized_keys
echo '<publicKey>' > /home/project_owner/.ssh/authorized_keys

chown <owner>:allowssh /opt/data/shared
chmod 775 /opt/data/shared

adduser <owner> allowssh
adduser <owner> sftponly
adduser ubuntu allowssh

echo 'PermitRootLogin no
PubkeyAuthentication no
PasswordAuthentication no
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding yes
PrintMotd no
AcceptEnv LANG LC_*
Subsystem    sftp    /usr/lib/openssh/sftp-server

# all users need to be in this group to ssh or sftp
# only pubic key authentication is allowed
Match Group allowssh
    PubkeyAuthentication yes

# users who are only allowed to ssh is added to this group aswel
# Croot keeps them form accessing anything outside /opt/data
# disables forwarding and only allows sftp
Match Group sftponly
    ChrootDirectory /opt/data
    DisableForwarding yes
    ForceCommand internal-sftp' > /etc/ssh/sshd_config


# restart ssh to apply changes
sudo systemctl restart ssh.service
```

Code listing 7.17: Bash script snippet from *fileserver.sh*

Figure 7.4 shows how SFTP is used to upload files to the file server. As shown you authenticate using the project owner's username (mariurae) and private key (id_rsa). Also notice how SSH is not allowed for the project owner.

```
❯ sftp -i id_rsa mariurae@10.212.139.196:/shared
Connected to 10.212.139.196.
Changing to: /shared
sftp> put very_secret_file
Uploading very_secret_file to /shared/very_secret_file
very_secret_file
sftp> exit
❯ ssh -i id_rsa mariurae@10.212.139.196
This service allows sftp connections only.
Connection to 10.212.139.196 closed.
~ ❯
```

Figure 7.4: Example showing use of SFTP as configured on the file server

7.3.4 Windows Clients

The creation of the windows clients are done in the file *rdp_lb_member.yaml* file, where the operating system, specifications, name, security key, network and bootup script are specified. This can be seen in Code Listing 7.14. The script that runs on the windows clients can be seen below in Code Listing 7.18. This Powershell script does the following;

- Allows the virtual workstation to connect to the file server by editing Registry keys
- Sets the virtual workstation to use the domain controller as the DNS server
- Creates a credential object for joining the domain
- Tries to join the domain (It tries up to 25 times for robustness)
- The items above only happens the first time the instance is booted.
- The second time the machine is booted, it tries to add members to the domain (It tries up to 10 times)

<RDPMembers> is a script that adds members to the domain which is generated in *orchestrator.py* based on the list of project members sent through the API.

```
1  #ps1_sysnative
2  echo custom-script
3  if( !(Test-Path -Path "C:\flag.txt")) {
4      New-Item -Path "C:\flag.txt"
5      echo "if"
6
7      New-ItemProperty -Path "REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\
           ↪ LanmanWorkstation\Parameters" -Name AllowInsecureGuestAuth -
           ↪ PropertyType "DWORD"
8      Set-ItemProperty -path "REGISTRY::HKLM\SYSTEM\CurrentControlSet\Services\
           ↪ LanmanWorkstation\Parameters" -Name "AllowInsecureGuestAuth" -value "1"
9
10     Set-DNSClientServerAddress -InterfaceIndex (Get-NetAdapter).InterfaceIndex -
           ↪ ServerAddresses 192.168.0.3
11
12     $dc = "rose.local"
13     $pw = ConvertTo-SecureString "Rosetest1" -AsPlainText -Force
14     $usr = "$dc\Administrator"
```

```
15      $creds = New-Object System.Management.Automation.PSCredential($usr,$pw)
16
17      $joined = $true
18      $i = 0;
19      do {
20          try {
21          Add-Computer -DomainName $dc -Credential $creds -ErrorAction Stop
22          echo "`n`n␣DOMAIN␣JOIN␣SUCCEEDED`n`n"
23          exit 1003
24
25          } catch {
26          $joined = $false
27          $i = $i + 1
28          echo "`n`n␣DOMAIN␣JOIN␣FAILED␣RETRYING␣IN␣1␣MINUTE␣`n`n"
29          Start-Sleep -s 60
30          }
31      } while ((-Not $joined) -And ($i -lt 25))
32
33  }else {
34      $added = $true
35      $i = 0
36      do {
37          try{
38          <RDPMembers> # Adds users to the rdp member group
39          } catch{
40              $added = $false
41              $i = $i+1
42          }
43
44      } while($i -lt 10)
45  }
```

Code listing 7.18: Bootup script for the virtual clients

Mounting Samba Drive on Virtual Workstations

After joining the AD domain, virtual workstation need to connect the Samba share created in Section 7.3.3 to a file path on their system. This proved to be quite the challenge since we had to create a new domain controller for each project network. If we had access to NTNUs AD we could easily create only one group policy where every time a user connects with RDP, the network drive gets connected [74]. With a domain controller in every network this gets more complicated as it is not possible to create the GPO automatically with PowerShell, and if the process can not be automated upon creation it can not be used for our platform.

7.4 Domain Controller

Because of our limited access to NTNUs Domain Controller, we had to implement our own version. We did this mainly for a working Proof of concept (POC) as a demonstration, as interacting with the Domain Controller (DC) is an important part. To do this we had to implement a Domain Controller running AD DS, as well as a DNS server, which is needed. One set of these servers are running in each

network as a stand-in for the NTNU Domain Controller created from the HEAT template seen in Figure 7.19.

The Domain Controller is an essential element in the system, mainly for its role in authentication. Our POC contain a couple of domain users, which are used as test users. After the workstation has been added to the domain, the domain user is added to a local Remote Desktop Protocol group. This allows for RDP to the workstation while authenticated as a domain user, hardening the confidentiality of the sensitive data.

The time between the creation of the domain controller and when the domain controller is ready was an issue. Creating an instance of the workstation is relatively quick, however the workstation is dependant on the domain controller. This is solved with a while-loop in the workstation startup script.

```
...
resources:
  dc:
    type: OS::Nova::Server
    properties:
      name:
        list_join: ["-", [{ get_param: project_name }, "domain_controller"]]
      image: Windows Server 2022 Standard [Licensed]
      flavor: m1.medium
      key_name: mankey
      networks:
        - port: { get_resource: dc_private_port }
      user_data: {get_file: dc-init.ps1}
  dc_private_port:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_name}
      fixed_ips:
       - ip_address: "192.168.0.3"
```

Code listing 7.19: The yaml file creating the domain controller instance used as a POC

The initialization script itself is separated to its own file as seen in 7.20. This script is ran on boot. The first time running it creates a file used as a logical flag and installs AD DS and DNS. It is then rebooted as is recommended [75]. The domain is created next. After yet another reboot, the test users are created. For each of these steps a file is created and checked. These files are used as flags for progress tracking purposes, as these steps has to be done in a certain order.

```
#ps1_sysnative
New-Item -Path "C:\test.txt"

echo "kjort" >> C:\test.txt

if( !(Test-Path -Path "C:\flag1.txt")) {
    echo "if"
    New-Item -Path "C:\flag1.txt"
```

```
    # Use self as DNS
    Set-DNSClientServerAddress -InterfaceIndex (Get-NetAdapter).InterfaceIndex -
        ↪ ServerAddresses 127.0.0.1
    #Install ADDS
    Install-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools
    # Set Administrator password
    net user Administrator Rosetest1
    Import-Module ADDSDeployment
    exit 1003

}elseif(!(Test-Path -Path "C:\flag2.txt")){
    echo "elseif"
    New-Item -Path "C:\flag2.txt"
    $secureSafeModePwd = ConvertTo-SecureString "Rosetest1" -AsPlainText -Force
    # Setup domain
    Install-ADDSForest -DomainName "rose.local" `
        -DomainNetbiosName ROSE `
        -SafeModeAdministratorPassword $secureSafeModePwd `
        -InstallDns -NoRebootOnCompletion -force
    exit 1003

}else {
    echo "else"
    New-Item -Path "C:\log.txt"

    $dc = "rose.local"
    $pw = ConvertTo-SecureString "Rosetest1" -AsPlainText -Force
    $usr = "$dc\Administrator"
    $creds = New-Object System.Management.Automation.PSCredential($usr,$pw)

    New-ADUser -Name "bojack" -Accountpassword (ConvertTo-SecureString "
        ↪ HorsemanPass1" -AsPlainText -Force ) -Enabled $true -Credential $creds
    New-ADUser -Name "diane" -Accountpassword (ConvertTo-SecureString "NguyenPass1"
        ↪  -AsPlainText -Force ) -Enabled $true -Credential $creds
    New-ADUser -Name "todd" -Accountpassword (ConvertTo-SecureString "ChavezPass1"
        ↪ -AsPlainText -Force ) -Enabled $true -Credential $creds
    New-ADUser -Name "sarah" -Accountpassword (ConvertTo-SecureString "LynnPass1" -
        ↪ AsPlainText -Force ) -Enabled $true -Credential $credst
    New-ADUser -Name "hollyhock" -Accountpassword (ConvertTo-SecureString "
        ↪ ManheimPass1" -AsPlainText -Force ) -Enabled $true -Credential $creds
    New-ADUser -Name "wanda" -Accountpassword (ConvertTo-SecureString "PiercePass1"
        ↪  -AsPlainText -Force ) -Enabled $true -Credential $creds
    New-ADUser -Name "pinky" -Accountpassword (ConvertTo-SecureString "PenguinPass1
        ↪ " -AsPlainText -Force ) -Enabled $true -Credential $creds
}
```

Code listing 7.20: Script setting up the domain controller

7.5 Deployment

Our entire infrastructure is deployed using Docker. Each service is hosted in it own Docker container, giving us a total of 6 containers; one for the demo website, one for the orchestration API, one for the Celery task queue, one for the celery worker, one for Grafana and one for Loki. We used two Docker compose files one for the logging service and one for everything else, because we view the logger service as

a standalone service that should be easy to replace. The infrastructure could be deployed anywhere that has access to NTNUs internal network either physically or through a VPN. However for us it made the most sense to host it within SkyHiGh on a Virtual Machine, so we made a administrator network and a Ubuntu server for this purpose.

The website is deployed using a standard *ubuntu/apache2* Docker container with barely any configuration needed to be done by us. It will therefore not be covered in any detail in this section, we will however describe all the other containers.

Orchestration API

This Docker container is responsible for hosting the API that receives user input, and the orchestration logic that creates a project infrastructure based on the user input. The Docker container is defined using a Dockerfile, shown in Code Listing 7.21.

```
1  FROM ubuntu:21.04
2
3  ENV DEBIAN_FRONTEND noninteractive
4
5  COPY creds/servicebruker.sh /creds/
6  COPY creds/worker.key /ssl/private/
7  COPY creds/worker.crt /ssl/cert/worker.crt
8
9  RUN apt update
10 RUN apt upgrade
11 RUN apt-get -y install python3-pip
12 RUN pip3 install waitress
13 RUN pip3 install redis
14 RUN pip3 install celery
15 RUN pip3 install Flask
16 RUN pip3 install python-openstackclient
17 RUN pip3 install python-logging-loki
18 RUN pip3 install flask-cors
19 RUN pip install hupper
20
21 CMD . /creds/servicebruker.sh && rm /creds/servicebruker.sh && cd /source && hupper
       ↪  -m waitress --port=8080 app:app
```

Code listing 7.21: Dockerfile for the orchestration API

The Dockerfile starts by copying 3 files from the *creds* folder of the host to the Docker container. The first file is a *serviceuser.sh*. This file generated by SkyHiGh and can be downloaded from the web interface, it is used to authenticate to the OpenStack API. It therefore needs to be sourced before starting the app for it to work, as is done in the last line of the Dockerfile, using the *CMD* keyword. Immediately after being sourced it is deleted as it is no longer needed, this provides some extra security in the case of a hacker getting a shell inside the Docker container. This way he would not be able to read the content of the *serviceuser.sh* file, which contains sensitive data.

The two next files that are copied are the SSL private key *worker.key* and public key *worker.crt*. These are used to sign and verify Celery tasks that are to be run by the worker. By default the worker trusts any task that is in the task queue, which can be exploited by attackers to run malicious tasks. By signing and verifying tasks using the SSL key pair, the worker can trust that the task is from the *orchestration_api* Docker container. There is a shell script in the *creds* folder that generates the key pair with the correct names, making it easier to deploy.

After copying the files all the packages we use in our python project is installed using *apt* and *pip*. Lastly *Waitress* is used to serve the API on the port *8080*. We used Waitress in conjunction with *Hupper*. *Hupper* automatically detects file changes to the source directory and restarts Waitress whenever it does so. This makes development a lot easier as you don't need to restart the entire container to see changes take effect.

Celery

Celery is deployed using two Docker containers one standard *Redis* container that is the task queue, and one container running the Celery worker. Since we use the standard redis container as the task queue it is defined entirely by the *docker-compose.yaml* file as seen in the excerpt below.

```
1  web:
2    build:
3      context: ../
4      dockerfile: ./docker/web/Dockerfile
5    ports:
6     - "8080:8080"
7    depends_on:
8      - redis
9    volumes:
10     - ../source:/source
11   networks:
12     - proxy_net
13 worker:
14   build:
15     context: ../
16     dockerfile: ./docker/worker/Dockerfile
17   volumes:
18     - ../source:/source
19   networks:
20     - proxy_net
21 redis:
22   image: redis:alpine
23   ports:
24     - "6379:6379"
25   networks:
26     - proxy_net
```

Code listing 7.22: Part of the main Docker Compose

The redis container has port *6379* exposed, this port will be used by the orchestration API (just called web in the Dockerfile), to push Celery tasks to the task queue. Subsequently the port will be used by the worker to fetch those tasks. Notice also how the container for the orchestration API, the Celery worker and the Redis task queue are all on the Docker network *(*proxy_net), as they need to communicate with each other.

The *Dockerfile* for the celery worker is almost identical to the Docker file for the orchestration API. As it also needs the SSL keypair, the python packages, and the file. The only difference is that instead of using Waitress to serve the API on the last line a worker is created using the command "*celery -A app.celery worker*". As shown in the *Dockerfile* we can see that it also needs to have the source directory mounted as a volume just like the orchestration API.

The Code Listing 7.23 shows how Flask and Celery with SSL message signing where configured in the *__init__.py* file at the root of our python project. Notice how the shorthand *redis:* can be used to give the IP address of the Redis container, this is because it is on the *proxy_net* along with the worker and the orchestration API

```
1  from flask import Flask, request
2  from celery import Celery
3  from secrets import token_hex
4
5  app = Flask(__name__)
6  app.secret_key = token_hex() # Secret key for sessions
7
8  # Informs celery of the broker
9  app.config['CELERY_BROKER_URL'] = 'redis://redis:6379/0'
10 celery = Celery(app.name, broker=app.config['CELERY_BROKER_URL'])
11
12
13 # enables message signing for celery
14 celery.conf.update(
15     security_key='/ssl/private/worker.key',
16     security_certificate='/ssl/cert/worker.crt',
17     security_cert_store='/ssl/cert/*.crt',
18     security_digest='sha256',
19     task_serializer='auth',
20     event_serializer='auth',
21     accept_content=['auth']
22 )
23 celery.setup_security()
24
25 from app.Controller import ProjectController
26 from app import Const, Routes
```

Code listing 7.23: *__init__.py file showing configuration of Celery and Flask*

7.5.1 Logging

As mentioned previously we use Grafana Loki for our logging server. Grafana Loki is designed to be used in Docker, and they provide a *docker-compose* template that we only made minor modifications to. The template file originally defines three Docker containers; Grafana, Loki and Promtail. We had no use for Promtail so we simply commented that section out. The other modification we made was to define a Docker network that connects the containers define in the logging *docker-compose* and the containers defined in the main *docker-compose*.

7.5.2 Deployment Step by Step

This step by step assumes that you have access to an OpenStack environment.

- Clone the *OrchestrationAPI* Git repo
- Install Docker on your host
- Download the *serviceuser.sh* script from SkyHiGh and place it in the *creds* folder.
- Create the SSL key pair by runnning *creds/creds.sh*
- Start the logging service by running "*docker compose up -d*" inside the *docker/logger* folder.
- Start the website, the orchestration API and Celery by running "*docker compose up -d*" from inside the *docker* folder.

Chapter 8

Evaluation

This chapter will include the evaluation of the platform developed during this bachelor project period.

8.1 User Testing Results

A broad selection of individuals was chosen to evaluate this bachelor project product. Individuals with different sets of skills was chosen so a more general overview of the project was made possible. This collection consists of individuals with and without cloud technology skills, and individuals studying IT, nursing and design.

There are mainly two different users testing this product - project owners and project members. The project owners was presented with the graphical web interface, and asked to make an project. It is important to note that the web interface is not a part of the project, but it is used to demonstrate the API functionality of the project. The project owners were also given an description of how the system works and presented results from project creations so that they can evaluate the whole system.

The project members were given the IP address to connect to the virtual workstations and instructions to do so. An existing project was used with existing data on for the users to test.

The requirements specified in Chapter 4 was evaluated:

FR1: Members should be added to project before creation
A project owner should be able to add members to a project on a graphical user interface before the project is created. Members are added by writing the university email of each member.
Evaluation results:
When using the graphical user interface, members are added before creating the

project. This part is quite strait forward. The testing results showed that it was quite easy for the participants to understand this part, since it is the first action to be done before hitting the create button. The design of the graphical user interface is out of this projects scope, but an improvement would be giving even clearer instructions on the interface.

FR2: It should be possible to add a project owner
Before creating a project, the project owner should also write in his/her email or another email if he/she is not the owner. This is done so an owner of the project is specified when creating the project.
Evaluation results:
Every participant in the testing were able to add a project owner's email and public SSH key. Some users found it difficult to find the SSH key. The concept of a public and private key is something not everyone knows, as well as the creation of a key pair and where it is stored. It was therefore necessary to give some additional instructions.

FR3: A project should be created by a click of a button
A project owner should be able to press a button on the graphical user interface to create a project when all preferred members are added, making them project members.
Evaluation results:
After inputting the list of project members, project owner, and the project owners public SSH key, the project can be created by clicking the button for project creation. It was quite clear to the participants that it was necessary to click the button to create a project.

FR4: A project owner should retrieve IP address for project clients
A few seconds after a project owner creates a project, the project owner should be able to retrieve an IP address on the graphical user interface by clicking a button to connect to the virtual workstations. This is the address all project members should use to connect to the virtual workstations.
Evaluation results:
In all the user tests, the IP address was received after approximately 10 seconds. This indicates that this function works as it is intended to.

FR5: Project members should be able to connect to project clients
When a member are added to a project and the project is created, the member should be able to use the retrieved IP-address to access the project clients and log in with their personal username and password, gaining privileges which were set by the project owner in the graphical user interface.
Evaluation results:
In all the tests, project members were able to connect to the project clients when they were ready. It takes about 15 minutes for the windows clients to be ready for

access, so this is one of the drawbacks from the tests. However, there is nothing to be done to improve this since this is a problem of the OpenStack cloud the solution runs on.

FR6: Project members should be able to access data
When a project member has connected to a project clients, he/she should be able to manage data from a file server represented as a network drive on the client. The project member should then be able to execute actions on the shared data based on their privileges granted by the project owner.
Evaluation results:
After a RDP session was established, all participants were able to access the data from the file server through the windows clients.

FR7: The system administrator should be able to review logs from the project
All actions performed in the project should be logged and presented to the system administrator on a graphical interface, presenting the data in a statistic manner. This logging solution could also be used by the it department of the university/company that use this service.
Evaluation results:
This is implemented and works fine. The user testing proved that the output from the logs presented in a graphical user interface were easy to understand. However, only API actions are logged, meaning activity within a project is not logged. This is something that should be improved on in the future. We will still consider this requirement as satisfied since logging is implemented. In the future the logging functionality should also be integrated with NTNUs centralized logging service.

FR8: The project owner should be able to delete a project
When there is no longer need for the project, the project and all of its files will be deleted. This is done by using the graphical user interface, specifying the name of the project, and press a delete button.
Evaluation results:
Deleting projects works well, and every tester where able to delete a project. However, the website made for a proof on concept that the orchestration API works is out of scope for this project, meaning that it does not include any authorization for access, which means anyone that has access to the website could delete a project.

FR9: The project owner should be able to add files to and download files from the file server
To create a file share that project members can access through the window clients, the project owner should be able to upload files to the file server. To this the need to be authenticated as the owner of the project.
Evaluation results:
Project owners are able to add and download files, either via CLI or FileZilla. This

proved to be a little challenging since not everyone had that much experience with the command line, and FileZilla is also a little complicated to use.

FR10: The system should support RDP session session persistence
If a project members disconnects from a session how they should be connected to the same virtual workstation the next time they log on. That way they are able to pick up their work from where they left off without losing any progress.
Evaluation results:
The system did support session persistence where the address of the device used is remembered, and connection from the same device will be established to the same virtual workstations each time. However, this did not work if a different device were used to connect to the virtual workstations. Therefore, this requirement is not quite satisfied yet.

NFR1: Requirements for access to information and information systems
NTNUs guidelines for access control [11] specifies that resource domains should be used for access control where access is granted based on the groups users belongs to. Microsoft Active Directory and LDAP are used for access control in resource domains. The system owner should define roles with specified access control and permissions to storage. There is also a requirements to logging activity related to access and storage.
Evaluation results:
The project group were not allowed to integrate to system with NTNUs LDAP and AD services. This means that the product does not satisfy NTNUs requirements for access to information and information systems. The project group did however implement a local AD domain and this secures access control and permission control for the different projects. Therefore this requirement can be considered satisfied, but optimally the project workstations should be joined to the AD domain of NTNU.

NFR2: The Health Norms requirements for user insight
The Norm for Information security and Privacy in the Health and Care Sector [9] specifies in section 4.2.3 that any individual which has his health data stored should have the possibility to get insight in their own data.
Evaluation results:
This is a requirement regarding privacy regulations, and the product does not have an direct solution to this requirement. This requirement could however be satisfied if a project owner shares information to the concerned party regarding only that person. A person not included in a project should not have access since a project may contain sensitive information regarding several individuals.

NFR3: Access control for source code
The source code for the system should be available for personnel with correct permissions. Access and changes to source code must be logged. This is in accordance to NTNUs guidelines for access control [11].

Evaluation results:
The source code for this service is stored on NTNUs in-house GitLab as a private project. This means that access permissions are strictly controlled. access can be granted to required parts satisfying this requirement.

NFR4: The system should satisfy NTNUs requirements for the highest level of confidentiality
If leaked information could result in remarkable damage for interests, NTNU, persons or collaboration partners, the information should be classified as strictly confidential [10]. Since information stored in this system could contain details about peoples health, business secrets, etc, it is important that information in this system should be treated as strictly confidential. Information should only be available for users with strictly controlled permissions and that are required to have access to the information.
Evaluation results:
This requirement is not quite satisfied in this version of the project. The fact that each project is segregated in its own private network and infrastructure combined with access and permission control should make the system quite secure. There is however quite a bit of security mechanisms that should be implemented to satisfy this requirement. This includes mechanisms such as two factor authorization, firewalls, more levels of restricted networks, stricter policies, and integration with NTNUs active directory services.

NFR5: The system should satisfy NTNUs requirements for the highest level of integrity
It is highly crucial that the data in the system always is authentic and correct. Compromised data could result in wrong decisions, errors in treatments if the data is related to health, or constructing errors if the data is related to building plans. This are just a few of many examples. It is therefore crucial that the system satisfy NTNUs classification of integrity for level 4 [10].
Evaluation results:
In regards to integrity, the system does not provide any mechanisms to ensure that data holds its integrity. This could be improved with a good backup solution. Logging would also play a central role into this requirement, where changes done to data would be logged. Other than these mechanisms, one would assume that the original data is stored offline and can be reviewed to make sure the data holds its integrity. Permissions is another spectre that can affect integrity relying on which individuals has permission to edit data. This would be the responsibility of the project owner to manage.

NFR6: The system should satisfy NTNUs requirements for the highest level of availability
Since information stored on the system could be highly important, it is crucial that it delivers extremely good availability. Very short down times could be catastrophic

since the system may contain time sensitive data. Therefore, the system must satisfy NTNUs classification of availability for level 4 [10]. It is also crucial that the system delivers good availability in form of quick responses in retrieval of the clients IP-address and creation of the project infrastructure. Because of this high prioritisation, the infrastructure should be fixed immediately if a problem occurs.

Evaluation results:

The product delivers good availability in some areas, but lacks availability in others. As soon as a project owners orders a project, the creation process starts, and the IP address is ready in about 5 seconds (The website waits 10 seconds to retrieve to ensure the retrieval of the IP address). Unfortunately, the Windows workstations takes quite a long time to get ready. This means it takes about 15 minutes before the virtual workstations are ready. The domain controller also uses about 15 minutes to be ready. For the virtual workstations to join the domain, they have to be restarted. This means that it takes about 20/25 minutes before the whole infrastructure is ready. This is unforunately out of our control, as this is an issue related to the products the OpenStack cloud runs on. There are also no implemented mechanisms to check if parts of a project are down, and then fix it as fast as possible. Because of this, the system does not satisfy this requirement, and only some of the issues can be fixed in the future.

8.1.1 Evaluation Conclusion

The evaluation proved to be useful, since some issues where discovered in the process of evaluation and some of the requirements were not quite satisfied. This means there is a lot of room for improvement. The evaluation also proved that there was a lot of the requirements that were satisfied, and therefore there was a lot of successful work in the project.

Chapter 9

Discussion

In this chapter several security issues and challenges from the project period are addressed.

9.1 Risks and Security Aspects

More Levels of Restricted Networks

Right now, the infrastructure is designed in such a way that a user has to be inside the NTNU internal network either physically or through a VPN connection. From here, the user then connects to a project directly with RDP and the load balancer floating IP address. This means that the IP address for the projects can be seen from within the NTNU internal network. The NTNU internal network is not to be considered safe because of the great amount of student, employees and scientists that has access to it. It would therefore be much more secure if there was a private segregated network between the NTNU internal network and the project networks.

This can be thought of as zones, where the NTNU internal network is represented as a green zone, meaning no sensitive data should be stored directly on this network. A private network within this network could be considered as a yellow zone, meaning some sensitive data could be stored here, but not the most confidential data because this network can be seen from the NTNU internal network. If another private network could be reached from this previous private network, it would be considered much more secure, since it can not be seen from the NTNU internal network. This network would then act as a black zone, meaning strictly confidential data could be stored here.

In order to make the project network suitable for strictly confidential data it would therefore be optimal to add one more layer of network on the infrastructure. This would mean that users would have to access an additional private network from the NTNU internal network that is restricted to the project members, and from

there establish a RDP session to the private project networks. It will then be necessary to remove any users not in a project when projects are deleted to make sure as few users has access to this *yellow zone* network as possible.

Two Factor Authentication

We have not integrated two factor authentication with any of the authorization mechanisms in our project. This means the interface for creating a project and the RDP connections to the virtual workstations. It is important to note that the graphic user interface is out of scope for this project so no security mechanisms are implemented to the web solution and therefor no two factor authentication. Implementing a system for two factor authentication on the connection to the virtual workstations is a little more tricky, and not implemented.

Stricter policies

For the system to work, policies for project networks were quite open, meaning it would allow most traffic. For security reasons it would be best to stricktly limit what kind of traffic is allown on the project network. This can easily be fixed by closing down what traffic is allowed using an Access Control List on router interfaces.

Windows Client Hardening

At the moment the Windows clients are and the users on them are fairly unrestricted. More strictly limiting what users can and can not do on the clients would improve security. This is something the IT department at NTNU has already done on physical machines on campus. If our system is integrated with NTNUs Active Directory, a lot of the same Group Policy Objects could be leveraged and reused.

Active Directory

This project was originally meant to be integrated with NTNU, and therefore it was logically to integrate the product with NTNUs active directory and LDAP. This project group was not allowed to work with these technologies since these are crucial services for NTNU. Because of this, the product does not have functional access control and permission control. We did however implement our own domain controller in each project to prove our concept of access control. This is not a safe solution, but its purpose is to be a proof of concept. Integrating this product with NTNUs Active Directory and LDAP would make the access and permission control way more secure and effective since it already has NTNU users, making it so we can specify which users to be allowed on a project and then those users will be authorized on NTNUs access and permission control mechanisms.

Hypertext Transfer Protocol Secure (HTTPS)

HTTPS is not implemented. The security measurement should be implemented between the user and the API to protect the confidentiality of the sensitive data during transit from anyone listening in on the traffic. Without HTTPS it is possible to listen in on the traffic between one project owner and the API. The attacker could for instance steal session cookies and delete a project.

9.2 Challenges During the Project

Restricted Knowledge Before Project

This Bachelor project required a lot of skills within Infrastructure as Code. The bachelor group consists of bachelor students in digital infrastructure and cybersecurity. This bachelor study includes a lot of it management courses, which are very relevant for this project. However, non of the group members had taken the course in Infrastructure as Code, which meant there was a somewhat restricted level of knowledge before the project. This proved not to be a major problem, since it was quite easy to find good resources for development with IaC. Since the group developed the product on an OpenStack cloud, the OpenStack Heat documentation was rapidly used for creating the OpenStack Heat orchestration Templates. There was also a small lack in knowledge of API development, where the group only had some experience from the databases course. This did also show to not be any problem because of good self study.

Manilla

Manila is a OpenStack service that provides a file server service. It is for creating a file sharing environment for virtual machines on a cloud infrastructure and also has integrated API. With OpenStack Manila it is easy to make and manage file shares [76]. Unfortunately, this service was not included in SkyHiGh which made the group unable to use this service, which could have made the project mush easier.

NTNU Active Directory

Due to our lack of access to NTNUs active directory we had to create our own step-in as a POC. This gave us a couple of challenges. Our first idea was to have a Windows server running AD DS within our administrator network with an interface exposed to the rest of NTNUs network accessible by a floating IP. The administrator network is in parallel with the different project-networks.

However this provided unexpectedly difficult, mainly due to DNS issues. It goes against AD "nature" to be exposed to outside networks directly due to its importance and normal tasks. In this case the rest of NTNUs local network.

Whenever you set up a domain controller, you also need to create a DNS server. This DNS server resolves the domain controller name its own private IP. This private address is inaccessible because its in a completely separate network. We were not able to make this resolve to the floating IP, because it would be reverted back to the private IP, possibly as a security mechanism from Microsoft to make sure people aren't doing what we are trying to do to make this POC work.

A possible workaround to this would be to create another DNS server, independent of the domain controller, which the workstations could use, resolving to the domain controller's floating IP. Due to this not being an important part of the task, we did not prioritize this solution. In the end we added a domain controller in each individual network, avoiding the aforementioned issues completely.

Small Amount of Resources

At the start of the project we were provided with some resources on the SkyHiGh cloud. These resources however was not sufficient. There is one larger server placed in its own network, which is tasked with hosting the website, the API itself, the Celery worker and the Redis store in docker containers. This server has eight virtual CPUs, 16 GB RAM, and 40 GB storage. Creating one project stack also require quite a lot of resources: six virtual CPUs, 16 GB of RAM, and 120 GB of storage. This became an issue when we had multiple stacks running at the same time during testing and development. We did get access to more resources upon request.

Chapter 10

Closing Remarks

In this chapter we will talk about the results of working on this project. We will discuss our learning outcome this semester and talk about future improvements for the platform. We will also come with a conclusion based on all previous chapters.

10.1 Learning Outcome

General

This project has thought us a lot. We have used new technologies, gone more in depth in technologies we have learned about earlier, but most importantly we have learned to work as a team. We have also learned that communication is paramount to make teamwork actually work.

Communication and Teamwork

Throughout the project period we learned how to work as a group and delegate tasks between the group members. Working as a group can be both challenging and rewarding. We have also learned the value of communication and planning ahead. We have learned that investing time and energy into communication and planning early saves a lot of time and pain later on. When multiple people work on different tasks that are dependant on each other you need communication to reduce idle time.

We have also learned that even with good communication and planning there will be hiccups and issues during the development progress; issues your group didn't think of will emerge, bugs that require a lot of time and attention, and even miscommunication.

Thesis Writing

Four people writing on the same thesis has also been challenging; however we have learned a lot. We have learned that communication is just as important when

it comes to thesis writing as it is during the development of a product or service. Both the finished thesis and language will become inconsistent if there is no communication during the thesis writing. We have learned the value of structuring acronyms and glossaries, and that thesis writing require quite a lot more time than you might initially think it does.

10.1.1 API

Although we have created simple APIs before, nobody in our group had any prior experience with the Python's Flask framework. Python is regarded as a programming language that is easy to use, and this includes the Flask framework as well. Therefore getting started was rather easy thanks to good documentation. Learning Python and Flask is useful, as these technologies open up a lot of opportunities. Python is used for all kinds of tasks; everything from the development of large software products to smaller scripts.

10.1.2 Infrastructure as Code

Nobody in our group had any significant experience when it came to IaC before starting this project. This proved to be a great learning experience for all of us, as we learned best practice for creating and managing cloud architecture.

10.1.3 Windows

During this project we have interacted a lot with Windows, both client operative systems as well as Windows server. At the start of the project we assumed that the work needed to be done on these Windows nodes would be rather easy. However, they gave us a lot of problems, resulting in us learning a lot about them. We learned a lot about Active Directory, Microsoft domains, and the Windows Registry.

10.2 Conclusion

The main goal of the project was to develop certain components for a secure platform for data sharing and editing. We started off researching what technologies to use and how the infrastructure would be structured. By the end of the project, we had created a platform where all of the requirements were met to some degree, however some aspects still needs improvement and some finishing touches.

10.3 Future Improvements

We have achieved a lot this semester, however there are still a lot to be improved. Due to limited access and time we had to prioritize the different aspects of this

system that we were able to implement. We were not given access to NTNUs Active Directory (AD) and their centralized logging service. A future improvement would therefore be to more tightly integrate the platform with NTNUs IT system as a whole.

In Chapter 9, a lot of important aspects of improvement are discussed. These items are all important candidates for future improvements. Since the goal of this bachelor project was to implement certain components of the platform, there is a lot of additional work that has to be done to complete such a solution. In relation to the components developed by us during the project, the future improvements mainly relies on security aspects. As discussed in Chapter 9, the main items that should be improved on or implemented in the future are; More levels of restricted networks, two factor authentication, stricter policies, NTNU Active Directory and LDAP, and HTTPS.

All communication between the user the API should be transferred securely over HTTPS. This is important to hide sensitive information during transit. In this case the most sensitive information are cookies, like the session cookie. These session cookies should be used for authentication and authorization on the more sensitive endpoints. For instance only the project owner should be able to delete a project.

Centralized logging is an important part of bigger networks. When there are so many nodes connected that manually checking each one is close to impossible, it is paramount to keep them in one place where you can utilize tools to search for important information. This has to be done in the future due to our limited access to NTNUs internal systems.

Backup is also an important implementation that did not get included in this project. With a good backup solution, both the integrity and availability of the platform would be improved.

There are also several points for improvement when it comes to SkyHiGh and its resources. OpenStack Manila is a technology that absolutely should be integrated in SkyHiGh, making it much easier to manage shared environments. All of these items are discussed in depth in Chapter 9.

Bibliography

[1] 'A minimal application.' (), [Online]. Available: https://flask.palletsprojects.com/en/latest/quickstart/#a-minimal-application.

[2] 'How to use decorators in python, by example.' (), [Online]. Available: https://towardsdatascience.com/how-to-use-decorators-in-python-by-example-b398328163b.

[3] M.-L. Lervåg. 'Dette er de største studiestedene i norge.' (2021), [Online]. Available: https://www.ssb.no/utdanning/hoyere-utdanning/statistikk/studenter-i-universitets-og-hogskoleutdanning/artikler/dette-er-de-storste-studiestedene-i-norge.

[4] NTNU. 'Ntnu i tall og fakta.' (2021), [Online]. Available: https://www.ntnu.no/tall-og-fakta.

[5] 'Informasjonsklassifisering - informasjonssikkerhet.' (2022), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/Norsk/Informasjonsklassifisering+-+informasjonssikkerhet.

[6] NTNU. 'Nice-1.' (), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/Norsk/NICE-1.

[7] 'Tsd - tjenester for sensitive data.' (2022), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/Norsk/tsd+-+tjenester+for+sensitive+data.

[8] 'Nird data storage.' (2022), [Online]. Available: https://www.sigma2.no/data-storage.

[9] D. for e-helse. 'Norm for informasjonssikkerhet og personvern i helse- og omsorgssektoren.' (2020), [Online]. Available: https://www.ehelse.no/normen/normen-for-informasjonssikkerhet-og-personvern-i-helse-og-omsorgssektoren/_/attachment/inline/ab1b230b-eb1f-47f8-b716-aae86ed34a8f:085dc760fecbf9141ee59f446495c41b1a73346f/Normen%20versjon%206.0.pdf.

[10] NTNU. 'Informasjonsklassifisering - informasjonssikkerhet.' (2018), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/Norsk/Informasjonsklassifisering+-+informasjonssikkerhet#section-Informasjonsklassifisering+-+informasjonssikkerhet-Arkivn%C3%B8kkel.

[11] 'Retningslinje for tilgangskontroll.' (2018), [Online]. Available: https://docplayer.me/107083611-Ntnu-retningslinje-for-tilgangskontroll.html.

[12] Microsoft. 'Understanding the remote desktop protocol (rdp).' (2021), [Online]. Available: https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol.

[13] FEIDE. 'Feide-administrator.' (2022), [Online]. Available: https://www.feide.no/feide-administrator.

[14] NTNU. 'Digsec course page.' (2022), [Online]. Available: https://www.ntnu.edu/studies/bdigsec.

[15] I. C. Education. 'Iaas (infrastructure-as-a-service).' (2019), [Online]. Available: https://www.ibm.com/cloud/learn/iaas.

[16] G. Batschinski. 'Iaas vs paas – who wins this fight?' (), [Online]. Available: https://blog.back4app.com/iaas-vs-paas/.

[17] 'Openstack at ntnu.' (2022), [Online]. Available: https://www.ntnu.no/wiki/display/skyhigh.

[18] I. C. Education. 'What is virtualization?' (2019), [Online]. Available: https://www.ibm.com/cloud/learn/virtualization-a-complete-guide.

[19] Wikipedia. 'Openstack.' (), [Online]. Available: https://en.wikipedia.org/wiki/OpenStack.

[20] Wikipedia. 'Compute (nova).' (2022), [Online]. Available: https://en.wikipedia.org/wiki/OpenStack#Compute_(Nova).

[21] Wikipedia. 'Networking (neutron).' (2022), [Online]. Available: https://en.wikipedia.org/wiki/OpenStack#Networking_(Neutron).

[22] Wikipedia. 'Block storage (cinder).' (2022), [Online]. Available: https://en.wikipedia.org/wiki/OpenStack#Block_storage_(Cinder).

[23] Wikipedia. 'Orchestration (heat).' (2022), [Online]. Available: https://en.wikipedia.org/wiki/OpenStack#Orchestration(Heat).

[24] NGINX. 'What is load balancing?' (), [Online]. Available: https://www.nginx.com/resources/glossary/load-balancing/.

[25] M. Evans. 'What is a load balancer and its types?' (2021), [Online]. Available: https://www.cloud4u.com/blog/what-is-a-load-balancer-and-its-types/.

[26] Openstack. 'Introducing octavia.' (2020), [Online]. Available: https://docs.openstack.org/octavia/queens/reference/introduction.html.

[27] Cloud-Init. 'Cloud-init documentation.' (2020), [Online]. Available: https://cloudinit.readthedocs.io/en/latest/.

[28] Wikipedia. 'Samba (software).' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Samba_(software).

[29] 'Nitroshare wiki.' (2022), [Online]. Available: https://github.com/nitroshare/nitroshare-desktop/wiki.

[30] 'Nitroshare wiki.' (2022), [Online]. Available: https://github.com/nitroshare/nitroshare-desktop.

[31] Microsoft. 'What is infrastructure as code?' (2021), [Online]. Available: https://docs.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code.

[32] Redhat. 'What is an api?' (2017), [Online]. Available: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.

[33] 'What is an api? how apis work, simply explained.' (), [Online]. Available: https://www.contentful.com/blog/2021/08/12/what-is-an-api/.

[34] T. P. Project. 'Flask.' (), [Online]. Available: https://palletsprojects.com/p/flask/.

[35] F. S. Python. 'Wsgi servers.' (), [Online]. Available: https://www.fullstackpython.com/wsgi-servers.html.

[36] Zope. 'Waitress wsgi server.' (), [Online]. Available: https://pypi.org/project/waitress/.

[37] 'What is docker?' (2022), [Online]. Available: https://opensource.com/resources/what-docker.

[38] i. Docker. 'What is a containeri?' (), [Online]. Available: https://www.docker.com/resources/what-container.

[39] P. Rubens. 'What are containers and why do you need them?' (2017), [Online]. Available: https://www.cio.com/article/247005/what-are-containers-and-why-do-you-need-them.html.

[40] D. Merkel. 'Docker: Lightweight linux containers for consistent development and deployment.' (2014), [Online]. Available: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.

[41] i. Docker. 'Overview of docker compose.' (), [Online]. Available: https://docs.docker.com/compose/.

[42] 'Active directory domain services overview.' (2022), [Online]. Available: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[43] Microsoft. 'Active directory domain services overview.' (), [Online]. Available: https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview.

[44] Microsoft. 'Domain name system (dns).' (), [Online]. Available: https://docs.microsoft.com/en-us/windows-server/networking/dns/dns-top.

[45] A. Saputra. 'Understanding forward and reverse lookup zones in dns.' (2019), [Online]. Available: https://www.mustbegeek.com/understanding-forward-and-reverse-lookup-zones-in-dns/.

[46] GeeksForGeeks. 'Working of domain name system (dns) server.' (), [Online]. Available: https://www.geeksforgeeks.org/working-of-domain-name-system-dns-server/.

[47] P. Loshin. 'Domain controller.' (), [Online]. Available: https://www.techtarget.com/searchwindowsserver/definition/domain-controller.

[48] G. Singh. 'What is the difference between a domain controller and a dns server?' (), [Online]. Available: https://www.quora.com/What-is-the-difference-between-a-domain-controller-and-a-DNS-server.

[49] L. du Toit and G. Thomassen. 'Services for sensitive data (tsd) whitepaper v6.0.' (2019), [Online]. Available: https://www-int.uio.no/english/services/it/research/sensitive-data/about/whitepaper_tsd_v6.0_2020.pdf.

[50] U. of Bergen IT Department. 'Projects in tsd.' (), [Online]. Available: https://www.uio.no/english/services/it/research/sensitive-data/about/projects_in_tsd/projects_in_tsd.md.

[51] NTNU. 'Tsd - tjenester for sensitive data.' (2021), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/Norsk/tsd+-+tjenester+for+sensitive+data.

[52] J. H. Saltzer and M. Schroeder, 'The protection of information in computer systems,' *Communications of the ACM*, vol. 17, no. 7, p. 8, 1975.

[53] 'Safe for decision makers.' (), [Online]. Available: https://www-int.uio.no/english/services/it/research/sensitive-data/about/whitepaper_tsd_v6.0_2020.pdf.

[54] 'Safe (secure access to research data and e-infrastructure.' (8.04.2022), [Online]. Available: https://www.uib.no/en/it/131011/safe-secure-access-research-data-and-e-infrastructure.

[55] Altexsoft. 'Functional and nonfunctional requirements: Specification and types.' (2021), [Online]. Available: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.

[56] 'What is ssh public key authentication?' (2021), [Online]. Available: https://www.ssh.com/academy/ssh/public-key-authentication.

[57] 'Secure ftp - sftp.' (2013), [Online]. Available: https://i.ntnu.no/wiki/-/wiki/English/Secure+FTP+-+SFTP.

[58] 'Kanban, a brief introduction.' (2016), [Online]. Available: https://www.atlassian.com/agile/kanban.

[59] 'What is kanban.' (), [Online]. Available: https://d30s2hykpf82zu.cloudfront.net/wp-content/uploads/2018/11/Kanban-for-Lean-Agile-Teams.png.

[60] 'Iterate faster, innovate together | gitlab.' (2022), [Online]. Available: https://about.gitlab.com.

[61] 'Latex, a document preparation system.' (2022), [Online]. Available: https://www.latex-project.org.

[62] 'Clockify.' (2022), [Online]. Available: https://clockify.me.

[63] 'Facebook messenger.' (2022), [Online]. Available: https://www.messenger.com.

[64] 'Discord.' (2022), [Online]. Available: https://discord.com.

[65] 'Microsoft teams.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Teams.

[66] 'Overleaf.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Overleaf.

[67] 'Yaml.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/YAML.

[68] 'Pools.' (2022), [Online]. Available: https://developers.cloudflare.com/load-balancing/understand-basics/pools/.

[69] 'Round-robin scheduling.' (2022), [Online]. Available: https://en.wikipedia.org/wiki/Round-robin_scheduling.

[70] 'Configure health checks for your classic load balancer.' (2022), [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/elb-healthchecks.html.

[71] 'Cloud-init.' (2022), [Online]. Available: https://cloudinit.readthedocs.io/en/latest/index.html.

[72] 'Install and configure samba.' (), [Online]. Available: https://ubuntu.com/tutorials/install-and-configure-samba#1-overview.

[73] 'Smb.conf - samba.' (), [Online]. Available: https://www.samba.org/samba/docs/current/man-html/smb.conf.5.html.

[74] 'How to map network drives with group policy (complete guide).' (2022), [Online]. Available: https://activedirectorypro.com/map-network-drives-with-group-policy/.

[75] 'Creating a windows active directory domain controller in oracle cloud infrastructure.' (), [Online]. Available: https://blogs.oracle.com/cloud-infrastructure/post/creating-a-windows-active-directory-domain-controller-in-oracle-cloud-infrastructure.

[76] 'Manila - shared filesystems service.' (2021), [Online]. Available: https://docs.openstack.org/kolla-ansible/latest/reference/storage/manila-guide.html.

Appendix A

Project Plan

NTNU
Kunnskap for en bedre verden

Department of Information Security and Communication Technology

DCSG2900 - Bachelor in Digital Infrastructure and Cyber Security

---

Project Plan - Platform for Secure Data Processing ROSE

Written by:
Martin Kristensen Eide
Jørgen Mo Opsahl
Marius Raes
Anders Kampesæter Slaaen

January, 2022

Table of Contents

List of Figures

List of Tables

1 Background and Goals

1.1 About Us

ROSE stands for Raes, Opsahl, Slaaen, Eide, which are the four group members who's currently in the final semester of the 2019 digital infrastructure and information security bachelor degree. Our interests are wide spread. Where some of our members are more interested in the operations aspect of Information Technology, others are more interested in the development aspect. However all members are interested in the cybersecurity aspect, which is paramount in this thesis.

1.2 Background

Norwegian University of Science and Technology (NTNU) handles a lot of sensitive information of varying degrees in research and academia. NTNU is currently dependent on external services for storing and interacting with this data. This can be problematic because of the nature of the data and poor experiences with the other proprietary alternatives. This is why NTNU has decided that they want to have their own infrastructure providing these services.

1.3 Project Goal

The goal of this project is to create a virtual infrastructure that automatically deploys a scalable network with virtual machines for a group of endx— users and storage to share, access and process their project data on a secure platform. This system allows authorized users to edit and analyze stored data, with different levels of authorization by interacting with FS (Felles Studentssystem) for students and PAGA (HR portal for employees). The project will reflect our thought process and why we decided to solve the problems the way we did in a final report.

We will not solve all the issues tied to this project, and will focus our scope to a few components. See section 2.3.

2 Scope

2.1 Subject Area

The nature of the project will cover a wide range of IT focus areas. This include network management, cloud management, risk management and infrastructure as code for managing and scaling our virtual infrastructure.

2.2 Task Description

Our task will be to design, risk-assess and designing components of a virtual infrastructure that will allow students and employees on NTNU to work together on sensitive data for their projects. We will not design all of them, due to our limited time and the share size of the project. Our solution must empathize the sensitivity of data shares and include access control of who is allowed to read or write different data sets. This is specially important for health data that is sensitive. Our employer needs the infrastructure to be completely automated, so that project managers can order resources without needing to contact support. Projects must be kept separate from each other to ensure data integrity. Users should be able to work on datasets, viewing and editing, without being able to export it. It should also be possible for users to work on different data sets within the infrastructure without being allowed to extract the data. The infrastructure will be hosted on NTNU's SkyHiGh cloud. Specific tasks for the bachelor project is as follows:

- Design the infrastructure for such an solution
- Carry out a risk assessment of the designed infrastructure
- Authenticate and authorize users of the system based on FS (Felles Studentssystem) for students and PAGA (HR portal for employees).
- Implement some crucial modules for the infrastructure:
  - Automate the creation of a OpenStack Network for each project, and their components: Virtual Machines (VM) used to interact with the data, logging server for centralized logging, Storage Area Network (SAN) for storage, file server for interacting with the SAN, and backup file server.
  - Develop a system for allocating VMs to different users.
  - Create the infrastructure for logging access history of the projects. This should interact with NTNU's centralized logging service.

– Create an administration component, where the administrator can add isolated project networks, manage users, etc.
– Design secure solutions for importing shared data.
– Implement import of data (Low priority)
– Host a Jupyter hub in each project network for statistic analysis of the data sets (Low priority)

The following figure shows the first draft of how we imagine our infrastructure will look like. Note that some of the components will not be prioritized, as specified in the "Limitations" section.

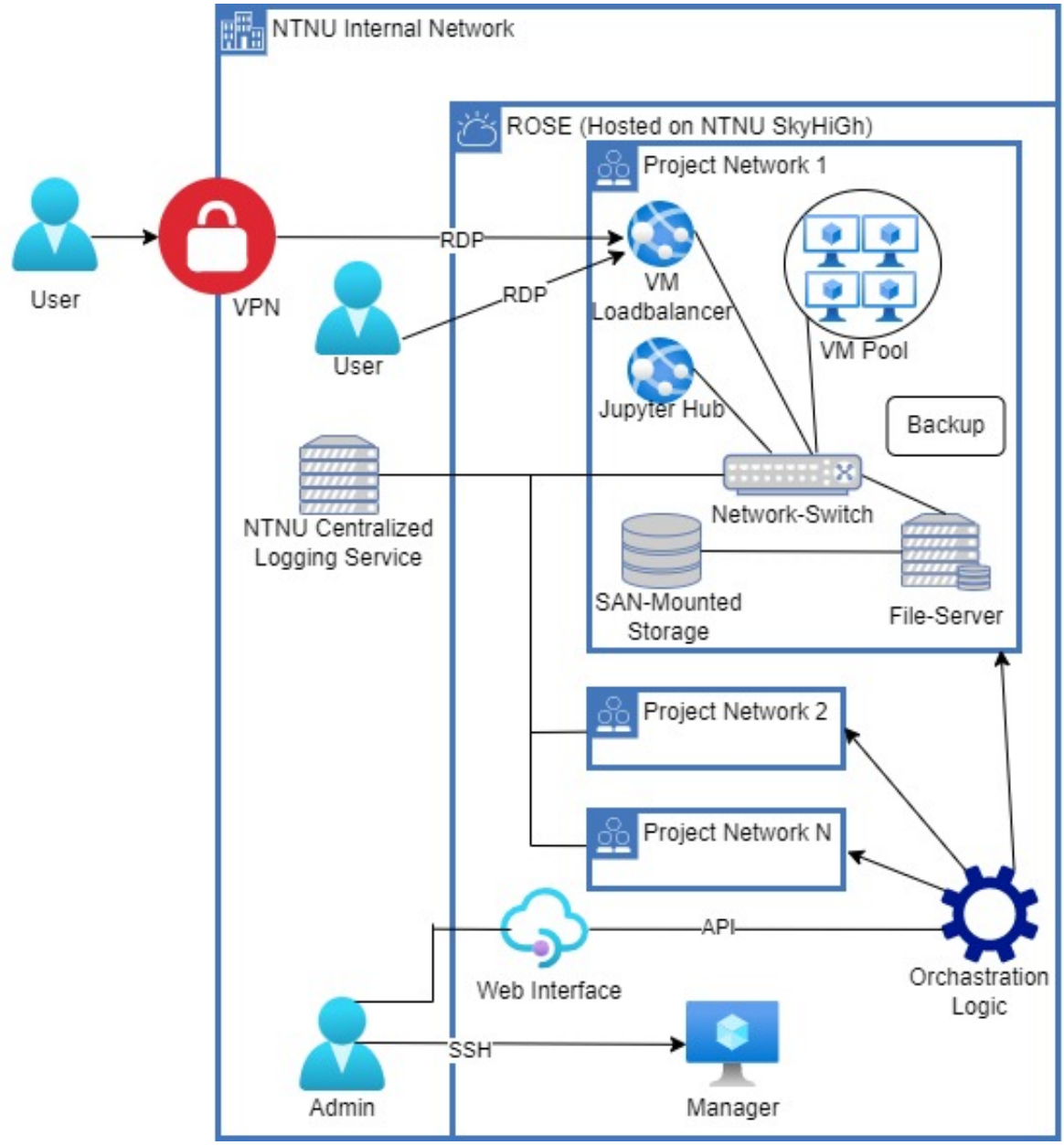


Figure 1: Draft for the cloud infrastructure

2.3 Limitations

The long term goal for NTNU is for this platform to provide a complete cloud platform where project owners can easily order virtual environments for their users to upload, store, process and extract data if the user is authorized for it. Implementing all these features is not possible in our bachelor project due to limited time and resources. Our group will therefore focus on certain parts of the infrastructure. We will try to make everything we do implement easily maintainable and extendable for future efforts.

Although the infrastructure is built to handle sensitive data, it is important to note that it is the responsibility of those that process the data to risk assess the processing of said data for the given purpose. This risk assessment must be done before any collection and storage of data begins. It is the responsibility of the data owners to make sure use of the ROSE infrastructure is in accordance to NTNU guidelines and external regulations.

The parts we are NOT going to focus on includes:

- Optimize the SAN implementation
- Backup of data stored.
- Authentication for the creator of a new project.
- Export of data
- Frontend-web-portal, however we will create the backend API to handle the requests and a basic frontend.

3 Project Organizing

3.1 Responsibilities and Roles

Project Leader: Anders Kampesæter Slaaen
Vara Leader: Jørgen Mo Opsahl
Secretaries: Martin Kristensen Eide, Marius Raes

3.2 Workflow and Group Rules

3.2.1 Workflow

For filesharing, code and documents used in our thesis, we have created a GitLab repository for keeping track of all our work. We use the GitLab issue board to keep track of our tasks for the bachelor thesis. We have four boards, open tasks, tasks in progress, one for peer review and one for closed tasks. Each member will move their task, as the status of the tasks changes. After a team-member is done with their tasks, they will move the issue from "In progress" to "Peer review". Thereafter the other group members will review the code, and deploy the final edition.

Each member will log their time spent working on the project by using the website Clockify.me. We have created different categories in Clockify to keep track of how much time is spent on each task throughout the project period.

We have created a Facebook groupchat that we use for text communication and a Discord channel for voice communication when we have digital meetings. For communication with our supervisor and client we have created Teams channel, where we can share files and notes.

3.2.2 Group Rules

Group Meetings and Designated Work Sessions
Each week, the group will meet on given times to work on the bachelor task. These meetings will be as following:

In addition to the planned meetings and sessions, we will distribute work each week during the Monday meeting. Every member must work on their assigned tasks, which can be done in groups or individually.

Monday	14.15 - 15.00 (Digital team meeting)
Tuesday	12.15 - 18.00 (Physical working session)
Wednesday	11.15 - 18.00 (Physical working session)
Friday	13.15 - 16.00 (Digital working session)

Table 1: Meeting days and meeting times

Absence
If a member is unable to join the scheduled meeting or work session, he is obligated to notify the rest of the team as soon as absence is known in written form either through mail or designated group Messenger chat.

If the member is absent for a prolonged period of time and is assigned a time sensitive task, the rest of the group has to delegate this task to another member if at all possible.

Logging
Members are required to log their own time from meetings, group work sessions, and individual work sessions via the groups' project on Clockify.

Workload
Each member is expected to be able to work for up to 30 hours each week until the project deadline.

Violations
Group rule violations will result in a written warning. Three written warnings to one individual group member will result in a meeting with the groups' supervisor (Kelly), with all members present.

4 Planning and Reporting

4.1 Main Project Sections

4.1.1 Development Model

We will use the Kanban approach from the Lean method for software development as our main development model to structure our work. To realize the model we use the Issue Board feature offered in the GitLab solution hosted by the Department of Computer Science on NTNU Gjøvik. We consider this to be a good approach since it will give a good overview of all the tasks to be done during the course of the project. We will follow the concept of Divide and Conquer, where we will divide the big tasks into several smaller tasks. In our Kanban board, we have sections for tasks classified as open, in progress, and closed. We prefer this approach, since it will make it easy for us to assign tasks between the different members, and make us focus on the tasks that are weighted with higher priority.

4.1.2 Method and Approach

The final product should include working modules for a secure file sharing and editing system. Since such a system can be quite large and complex, containing a lot of different modules, protocols and services. We will therefore design our solution and illustrate the design before starting to develop. It will be a complex and thorough illustration with reasoning and explanation of every functionality. We will then submit this model to our client and our supervisor and improve it on feedback. We will also perform routinely risk assessments of our design and evaluate the confidentiality, integrity and availability of the design.

When it comes to development of the technical infrastructure, we will use a test environment in SkyHiGh for testing our solution with mock-up data. We will have to be careful with how we manage this data, since it should simulate real sensitive data with a high score in confidentiality. This will be an iterative process, meaning we most likely will have to roll out several versions on the test environment before achieving the preferred result. In similarity to the development of the technical infrastructure, the illustrative design of the infrastructure and the creation of the risk assessment should be iterative processes where we expect a number of versions.

4.2 Plan for Status Meetings and Decision Points for the period

On every Wednesday during the project period, we will have a short meeting with our supervisor Jia-Chun Lin, referred to as Kelly, from 10.00 - 10.30. In these meetings Kelly will give us feedback so we can improve our work, by providing Kelly a status report of the work done in the last week as well as plans for further work. In addition to these status meetings, we do have internal status meetings each Monday from 14.15 to 15.00 where we review our work. In these meetings we will also review and assign further work, assigning specific tasks to each team member. This will act as guiding for work done by the group during the week.

We do also have an agreement with our client, where we are to contact him every time we have a clear plan for specific modules and designs we plan to develop. We do also have a direct contact line with him where he is part of our Microsoft Team, making it is possible to ask questions without necessarily arrange a physical or digital meeting.

5 Quality Control

5.1 Documentation

All source code and configuration files should be well documented both in the actual document with comments and trough other documents describing the code, how to use it and deploy it. Web APIs created should be documented and examples should be given on how to interact with them.

The strucure of modules and the system as a whole should be documented using UML and data flow diagrams, to describe how they work and how users interact with them. All documentation files should be available in the Gitlab repository.

5.2 Code review and proofreading

Every document we deliver should be proofread at least once by all group members to look for and correct mistakes. In addition the main report will be reviewed by our supervisor Kelly. We will deliver a first draft the 31st of March and Kelly will help us continuously improve the document before the final delivery.

Code written by a team members should be reviewed by at least once by another member to make sure it works as expected and is free of bugs and security flaws. This also encourages good documentation and helps other members gain a better understanding of the project implementation.

5.3 Project Progress Risk Analysis

5.3.1 Consequence and Probability Category Explanation

Consequence	
High	A high consequence for this project is defined as something that's detrimental to the completion or success of our project.
Medium	A Medium consequence for this project is defined as something that would hinder our group in completing the project to a satisfying level of completion.
Low	A Low consequence for this project is defined as something that would not hinder our group in completing the project to a satisfying level of completion.

Table 2: Consequence Category Explanation

Probability	
High	A high probability for this project is defined as something that's very likely/guarantied to occur during the project period.
Medium	A Medium probability for this project is defined as something that's likely to occur during the project period.
Low	A Low probability for this project is defined as something that's unlikely to occur during the project period.

Table 3: Probability Category Explanation

5.3.2 Risk Analysis of Project Completion

Risk Scenario 1
Action: Our scope is to wide
Consequence: Medium
Probability: Medium
Preventive measures: Address Eigil about the size of our scope.

Risk Scenario 2
Action: Project member suffers prolonged issues
Consequence: Medium
Probability: Low
Preventive measures: Inform other members early so plans can be made too work around the issues

Risk Scenario 3
Action: The group goes off track and loses its focus, resulting in a low quality product
Consequence: Medium
Probability: Low
Preventive measures: Continuous communication and status update with Eigil Obrestad

Risk Scenario 4
Action: Not meeting deadlines set in Gantt chart
Consequence: Low / Medium
Probability: High
Preventive measures: Weekly meetings with Kelly to keeps the project on track

Risk Scenario 5

Action: Disagreement within group
Consequence: Medium
Probability: High
Preventive measures: Open communication within the group to catch issues early. A meeting with Kelly will be called if issues can't be resolved within the group.

6 Plan for Project Execution

6.1 Work Structure

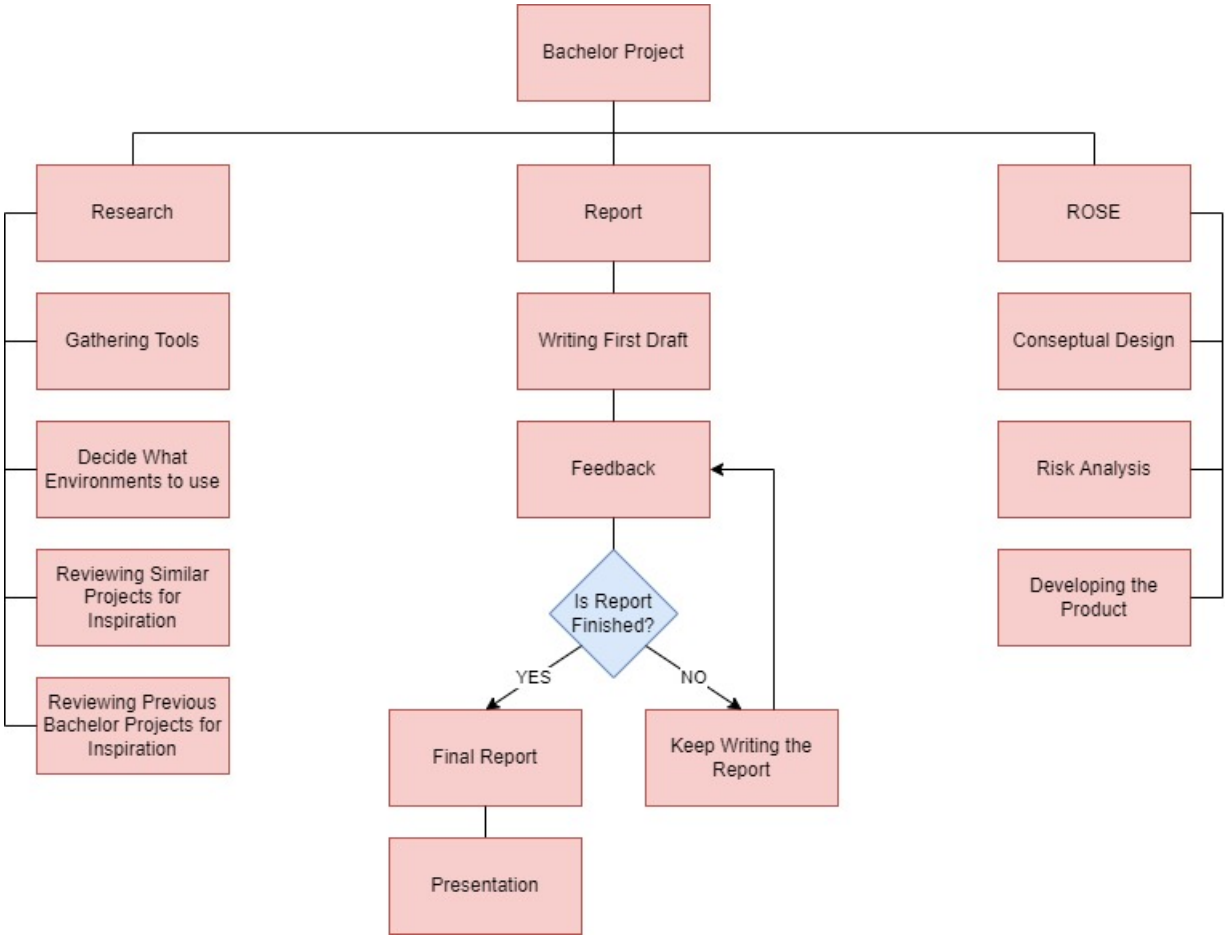


Figure 2: Breakdown of Work Structure

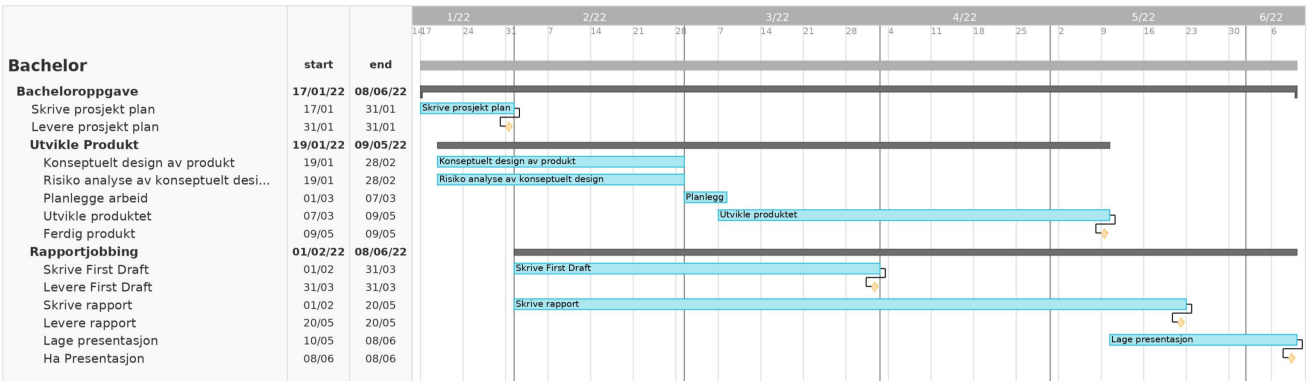


Figure 3: Gantt chart

6.2 Deadlines

- **Project Agreement:** January 31st
- **Project Plan:** January 31st
- **First draft thesis:** March 31st
- **Thesis:** May 20th

7 Confirmation

I have read the project plan and agree with its content.

Martin Kristensen Eide

Jørgen Mo Opsahl

Marius Raes

Anders Kampesæter Slaaen

Appendix B

Project Agreement

NTNU

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt: INSTITUTT FOR INFORMASJONSTEKNOLOGI OG KOMMUNIKASJONSTEKNOLOGI
Veileder ved NTNU: JIA-CHUN LIN e-post og tlf. jia-chun.lin@ntnu.no, 45849 287
Ekstern virksomhet: NTNU IT-DRIFT Ekstern virksomhet sin kontaktperson, e-post og tlf.: EIGIL OBRESTAD eigil.obrestad@ntnu.no, 61135143
Student: Jørgen Mo Opsahl Fødselsdato: 16.09.1997
Ev. flere studenter[1] Marius Raes 03.05.2000 | Martin Eide 15.04.2000
Anders Slaaen 29.07.1999

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato:
Sluttdato:

Oppgavens arbeidstittel er: Plattform for sikker databehandling

[1] Dersom flere studenter skriver oppgave i fellesskap, kan alle føres opp her. Rettigheter ligger da i fellesskap mellom studentene. Dersom ekstern virksomhet i stedet ønsker at det skal inngås egen avtale med hver enkelt student, gjøres dette.

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven:

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[2]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten

[2] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

X	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:
Veileder ved NTNU: Dato:
Ekstern virksomhet: Eigil Obrestad, 28/1-2022 Dato:
Student: Dato: Marius Raes, 26.1.2022
Ev. flere studenter Anders Slaaen, Jørgen Mo Orsahl,
Martin Kristensen Eide

Appendix C

Project Description

Platform for sikker databehandling

NTNU IT - Drift - Server

October 19, 2021

Oppdragsgiver

Oppdragsgiver er seksjon for IT Drift i IT-avdelingen på NTNU. Kontaktperson er Eigil Obrestad.

Bakgrunn

På NTNU jobber studenter og forskere med å analysere ulike datasett. En del av disse datasettene er mer sensitive enn andre. For eksempel er det i helsesammenheng analysert datasett som inneholder helseopplysninger, i informasjonsikkerhetssammenheng analysert datasett som inneholder biometriske opplysninger og så videre. Når slike datasett behandles og lagres er det spesielt viktig at dette skjer på en sikker måte.

NTNU har i dag noen løsninger hvor slike sensitive data kan lagres, og man har avtaler med ulike eksterne tjenester hvor slik informasjon kan lagres og behandles på en trygg måte. Disse tjenestene fungerer bra i noen sammenhenger, samtidig som at de kommer litt til kort i andre sammenhenger. Det er derfor startet et prosjekt med å lage en trygg infrastruktur hvor man både kan lagre og behandle sensitive data.

En slik løsning har krav som at den skal automatiseres, ulike prosjekter skal holdes separate, og man skal tilby brukerne gode muligheter til å jobbe på data uten at man nødvendigvis har muligheten til å hente data ut. Målet er å bygge en løsning som er modulær, og kan utvides i fremtiden for å møte fremtidige behov.

Oppgaven

Denne oppgaven vil gå ut på å designe, risikovurdere og bygge enkelte komponenter av en slik løsning. Disse komponentene kan realiseres på en rekke ulike måter, men felles stikkord er "automatisering", "infrastruktur som kode" og "openstack". Eksempler på komponenter man kan fokusere på kan være:

- Sikker importering/eksportering av data
- Sikre virtuelle arbeidsstasjoner for å jobbe med data.
- Sentral administrasjonskomponent
- Selvhjepsløsning for å administrere løsningen.

Lista er kun for inspirasjon, og på ingen måte uttømmende eller begrensende. NTNU IT legger opp til at studentene skal få rom til å forme oppgaven mot de deler studentene finner mest spennende.

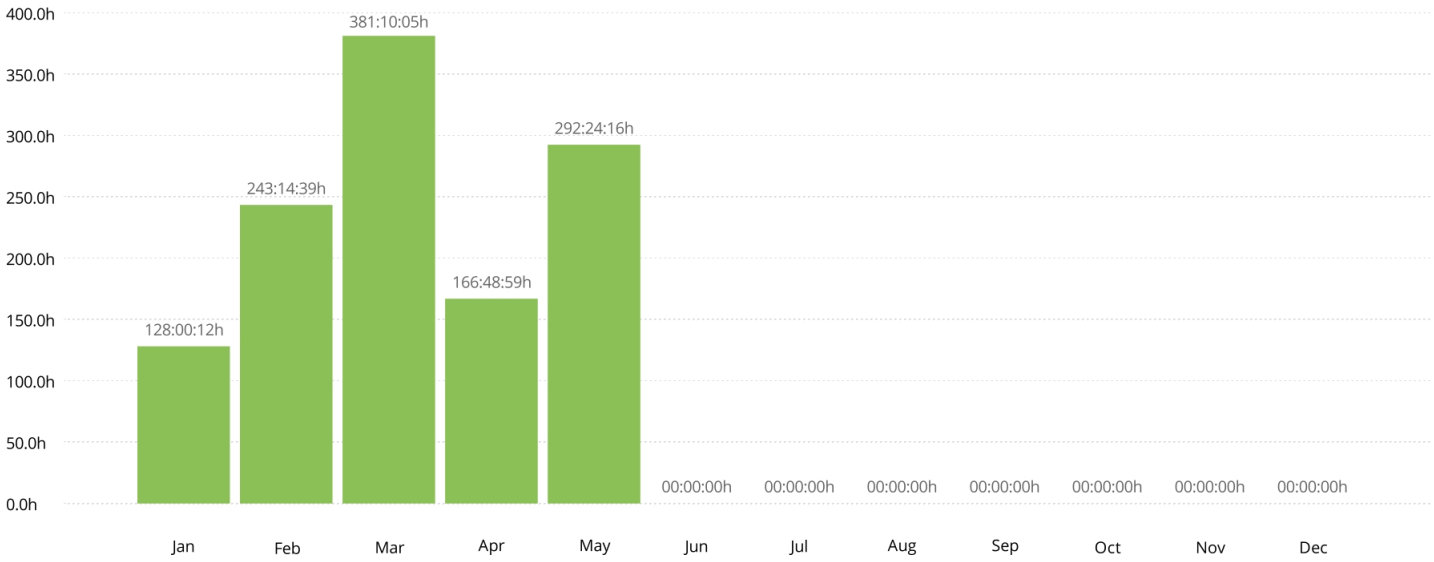
Appendix D

Time tracking

Summary report

01/01/2022 - 31/12/2022

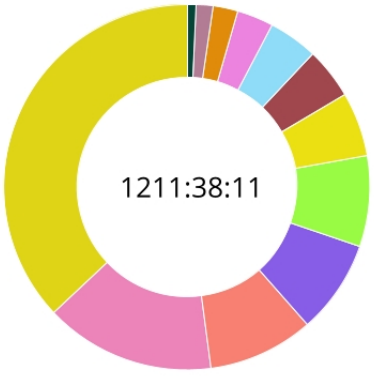
Total: 1211:38:11 Billable: 1211:38:11 Amount: 0.00 USD



Project

bachelorjobbing	1211:38:11	100.00%

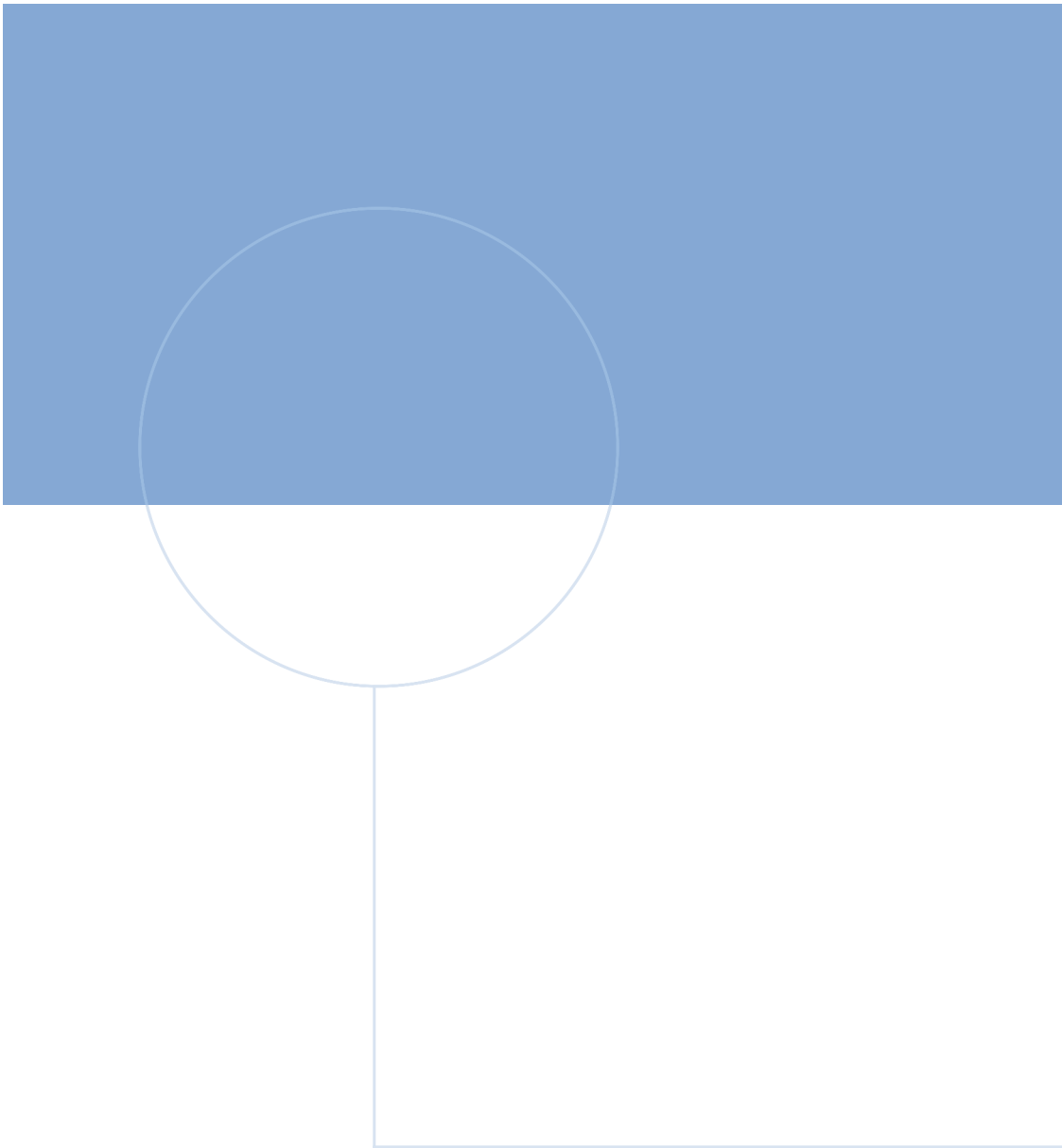
Task



Report - bachelorjobbing	448:46:59	37.04%
heat-template - bachelorjobbing	182:24:49	15.05%
project_plan - bachelorjobbing	113:07:28	9.34%
orchestration logic - bachelorjobbing	100:06:43	8.26%
research - bachelorjobbing	97:29:08	8.05%

●	dc - bachelorjobbing	69:15:39	5.72%
●	møte - bachelorjobbing	54:07:24	4.47%
●	API - bachelorjobbing	52:46:50	4.36%
●	Access control - bachelorjobbing	39:12:26	3.24%
●	Infrastructure_design - bachelorjobbing	26:34:35	2.19%
●	web gui - bachelorjobbing	18:01:19	1.49%
●	logging - bachelorjobbing	09:44:51	0.80%

Project / Task	Duration	Amount
bachelorjobbing	1211:38:11	0.00 USD
Report	448:46:59	0.00 USD
heat-template	182:24:49	0.00 USD
project_plan	113:07:28	0.00 USD
orchestration logic	100:06:43	0.00 USD
research	97:29:08	0.00 USD
dc	69:15:39	0.00 USD
møte	54:07:24	0.00 USD
API	52:46:50	0.00 USD
Access control	39:12:26	0.00 USD
Infrastructure_design	26:34:35	0.00 USD
web gui	18:01:19	0.00 USD
logging	09:44:51	0.00 USD



NTNU

Kunnskap for en bedre verden