

Marius Fylling and Lars Klubbenes

Automating and simplifying the creation of virtual machines for students

An on-premises cloud

Candidate 10013 and 10020, 91 pages including frontpage

Bachelor thesis
for the degree of baccalaureatus

Aalesund, June 2018

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences



NTNU

Norwegian University of Science and Technology

Bachelor thesis
for the degree of baccalaureatus

Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

© 2018 Marius Fylling and Lars Klubbenes. All rights reserved

Bachelor thesis at NTNU

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	<input checked="" type="checkbox"/>
2.	Jeg/vi erklærer videre at denne besvarelsen: <ul style="list-style-type: none"> ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. ikke refererer til andres arbeid uten at det er oppgitt. ikke refererer til eget tidligere arbeid uten at det er oppgitt. har alle referansene oppgitt i litteraturlisten. ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse. 	<input checked="" type="checkbox"/>
3.	Jeg/vi er kjent med at brudd på ovennevnte er å <u>betrakte som fusk</u> og kan medføre annullering av eksamen og utestengelse fra universiteter og høyskoler i Norge, jf. Universitets- og høyskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§14 og 15.	<input checked="" type="checkbox"/>
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert i Ephorus, se Retningslinjer for elektronisk innlevering og publisering av studiepoenggivende studentoppgaver	<input checked="" type="checkbox"/>
5.	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens studieforskrift §31	<input checked="" type="checkbox"/>
6.	Jeg/vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider	<input checked="" type="checkbox"/>

Publiseringsavtale

Studiepoeng: 20

Veileder: Mikael Tollefsen

Fullmakt til elektronisk publisering av oppgaven

Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten ([Andsverkloven §2](#)).

Alle oppgaver som fyller kriteriene vil bli registrert og publisert i Brage HiM med forfatter(ne)s godkjenning.

Oppgaver som er unntatt offentlighet eller båndlagt vil ikke bli publisert.

Jeg/vi gir herved NTNU i Ålesund en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:

ja nei

Er oppgaven båndlagt (konfidensiell)?

ja nei

(Båndleggingsavtale må fylles ut)

- Hvis ja:

Kan oppgaven publiseres når båndleggingsperioden er over?

ja nei

Er oppgaven unntatt offentlighet?

ja nei

(inneholder taushetsbelagt informasjon. [Jfr. Offl. §13/Fvl. §13](#))

Dato: 31. Mai 2018

Contents

1	Abstract	1
1.1	Abstract	2
2	Introduction	3
2.1	What we need to create	4
2.2	Current situation	4
2.3	Challenges	4
2.4	Hypothesis	5
2.5	Acronyms	5
3	Theoretical baselines	7
3.1	Introduction	8
3.2	Precursors	8
3.3	Virtualization: A look into the virtual world of servers	8

3.3.1	History	8
3.3.2	Hypervisor	10
3.3.3	Hypervisor technologies	11
3.4	Stateless APIs	11
3.5	Stateless	12
3.5.1	The session state	12
3.5.2	The stateless constraint	13
3.5.3	Authentication and authorization	13
3.6	Technologies to choose from	15
3.6.1	Perl	15
3.6.2	Python	15
3.6.3	Django	15
3.6.4	Flask	15
3.6.5	PHP	16
3.6.6	Node.js	16
3.6.7	GoLang	16
3.6.8	React	17
3.6.9	Angular	17
3.7	Data storage	17
3.7.1	Data mapping	17
3.7.2	A database	18

3.7.3	Data stores	18
3.7.4	Database schema	19
3.8	Hardware	20
3.8.1	Existing hardware	20
3.8.2	Improvements to existing hardware	20
4	Technical Documentation	21
4.1	Introduction	22
4.2	The big picture	22
4.2.1	The infrastructure	22
4.3	The API	23
4.3.1	API Endpoints	24
4.4	Authentication	25
4.4.1	How does JWT work?	26
4.4.2	Why use JWT?	28
4.4.3	Authentication directories	31
4.5	Data storage	33
4.5.1	Structured data storage	33
4.5.2	File storage	33
4.6	Creating a virtual machine	34
4.6.1	The VM Template	34
4.6.2	The VM Startup Agent	35

4.6.3	The Queue Manager	35
4.6.4	Creating a VM - Diagram	37
4.7	Foreign keys in VMWare	37
4.8	The user panel - Front end	38
4.8.1	Quick overview of front ends	38
4.8.2	Angular front end	38
4.8.3	Front end API calls	42
5	Manual	45
5.1	User manual	46
5.2	Introduction	46
5.2.1	Logging in to AutoDeploy for the first time	46
5.2.2	Creating your first Virtual Machine	47
5.2.3	Connecting to the virtual machines	51
5.2.4	Managing the virtual machine	54
5.3	Administration of AutoDeploy	55
5.3.1	Managing the database	55
5.3.2	Starting, stopping and managing the AutoDeploy services	55
5.3.3	Managing the virtual machines	56
5.3.4	Creating a new template	58
5.3.5	Disabling a template	62

6	Research Strategy	63
6.1	Research methods	64
7	Discussion	67
7.1	Introduction	68
7.2	Front-end	68
7.3	Queue Manager	68
7.4	Deployment of VMs	69
7.5	Programming languages	69
7.6	Planned functions	70
7.7	The future	71
8	Conclusion	73

Chapter 1

Abstract

1.1 Abstract

This thesis is based on a project in cooperation with NTNU.

Most people will at some point have sat down in front of a computer, performing some task, and realized: "This isn't going to work. I have a hammer, but right now I need a screwdriver."

Because so many different operating system platforms exist, many tools will be incompatible no matter which OS you choose. From Office on windows, to Linux specific server utilities, not having access to these tools present a problem when trying to learn how to use them, or use them to learn.

But what if we didn't have to choose at all? Just get a Windows PC when we want to use Microsoft proprietary software or a quick Ubuntu machine when we want to host a website? This is actually something we can do with virtualization today. But doing it on consumer grade hardware is often painfully slow. So, why not just use the much more appropriate servers at NTNU?

So we were tasked with creating a solution that enables NTNU students and faculty to easily create and maintain their own virtual machines. The solution we created contains an easy to use website front-end, and a separate back-end that controls the logic of creating, updating, and controlling the VMs. The system is user friendly, accessible everywhere, reliable, and created with cutting edge programming techniques in mind.

Chapter 2

Introduction

2.1 What we need to create

The delivery for NTNU is a system that the students can use to maintain and create/deploy virtual machines for use in school. The system must be user friendly, easy to use and stable. Technically, the system must be stateless and de-coupled.

2.2 Current situation

Traditional ways to create and maintain servers are fading fast. Most companies today use cloud based services and cloud servers, and the old "on-premises"-way of thinking is going away. [1]

Today, the cost of starting a service or starting a company is, in IT-perspective, a lot cheaper than it used to be. You don't need to purchase a brand new, state of the art server. All you need is your credit card, and a cloud service will get you the services you need for a very small fee, often as little as \$5 USD.

Students already have access to a server service hosted at NTNU Aalesund, but it is very limited. It's very hard for the administrator to maintain, because the virtual machines all have client software installed that is almost impossible to update. Also it's slow when creating new VMs, and crashes often.

2.3 Challenges

We need to provide students with a real world or "cloud-feel". But with on-premises servers. The project is an improvement of an already existing system. In short, the system should allow the students to create and maintain their servers, while the servers themselves are hosted in the local VM datacenter in Aalesund. [2]

One of the big challenges in this is to create a system that is not only versatile and stable, but also dynamic and maintainable. The system API has to be stateless, and the VM clients must be able to have updates installed from the outside.

We will need to get the current virtual machines merged over to the new system from the old one. Because these contain important user data.

Finding the tools best suited for this project will require research. This is a large project, so finding tools that interact well with each other is imperative.

2.4 Hypothesis

What can we do to improve the use of virtual machines to aid students in higher education?

- What tools do we need to achieve this?
- What are the major challenges of implementing such a system?

The new world of IT consists of major use of cloud services, and the students need to be able to use a as close to as possible-experience system without major costs from renting cloud VMs for all students. The system that we are creating should give a student the needed experience with very basic cloud services.

2.5 Acronyms

This section lists various acronyms used throughout this thesis (in alphabetical order).

- **AJAX**: Asynchronous JavaScript and XML.
- **ADP**: AutoDeploy
- **API**: Application Programming Interface.
- **CLR**: Common Language Runtime.
- **CMS**: Console Monitor System. (May also refer to Content Management system)

- **CP:** Control Program.
- **CSS:** Cascading Style Sheets.
- **DOM:** Document Object Model.
- **HTML:** Hypertext Markup Language.
- **HTTP:** Hypertext Transfer Protocol.
- **JSON:** JavaScript Object Notation.
- **JVM:** Java Virtual Machine.
- **MVC:** Model View Controller.
- **VDI:** Virtual Desktop Interface.
- **VM:** Virtual Machine.

Chapter 3

Theoretical baselines

Tools: Programming theories, languages and biases.

3.1 Introduction

In this chapter we will go through the theoretical baselines. This includes programming theories, languages, technical information about the current situation and some history about virtualization.

3.2 Precursors

NTNU made some prerequisites for the project. They wanted it to be stateless and be based on a front end framework that was widely used, relatively new and it should be maintained in the foreseeable future by their creators. We did not get any requests for specific implementations so we included this choice as a part of our research.

The system was requested to be stateless, so that the back end and front end can be decoupled. This will ease the amount of work required to maintain the system. It also makes upgrading either part of it much easier since they are completely separate.

The old system has a lot of VMs running on VMware ESXi, and these would have to be handled without any added data loss or downtime.

3.3 Virtualization: A look into the virtual world of servers

3.3.1 History

In the very early days computers were huge behemoths taking up entire floors of buildings. They cost millions and needed near constant maintenance. They were operated by mysterious men in lab coats and you were never allowed to be anywhere near it. However, if you had a very good reason you could write your program on punch cards and hopefully get the results back the following day. This is called a batch processing system. It is how computation was done for a long time in the early days.

The next leap forward was multi user systems. This meant that more than one simultaneous user would be possible. But it presented a new challenge in how they would interact with the system. The old way of punch cards would not work for more than one user. IBM's solution to the problem was in two parts. The first was CMS. CMS stands for console monitor system and was a small single user operating system. CMS handled the user interaction. The other was CP, or Control program. CP created virtual machines running CMS. Together they formed CP/CMS. Along with Multics (another time-sharing operating system), they started the idea of virtualization. [3]

Many of the ideas from Multics are still present in operating systems to this day. It is a spiritual successor in many ways.

On a number of points we were influenced by Multics, which suggested the particular form of the I/O system calls and both the name of the Shell and its general functions. [4]

Though Unix itself does not run any virtual operating systems. It virtualizes the user space instead. This means it allows system resources to be shared amongst users. It was easily ported to different hardware platforms, so you could run the same software on them.[5] However, this required the software to be compiled for the platform as well. [3]

For the same software to run truly independently from the platform then application virtualization is required. The first of which was oracle's java. It combines many of the earlier concepts by running a "java virtual machine" every time a java program is executed. That way they have a small operating system all to itself. This means it can run anywhere oracle has ported the java runtime environment. This can be anything from phones to cars or datacenters. The main drawback is some tasks are much slower, such as those who are heavily hardware dependent.[6][6]

In more recent years we see a shift towards virtual machines. Now in the form of “VDI” or virtual desktop infrastructures. VDI allows users to access a full desktop with the same resources from any place or device. This of course means it is very flexible. Files do not need to be stored locally which is good from both a security and performance perspective. And, resources like CPU and memory be can shared when not in use. This is very much a resurgence of the original 1960’s idea of a centralized server running full virtual operating systems. [7]

The first concept of virtualization is usually has its origins around the Mainframe days, mid and late 1960. This is when IBM invested a lot of time to create good time-sharing solutions. Time-sharing sparked the possibility for small businesses, and even individuals, to use a computer - without actually owning one. This is often a very similar to what is driving virtualization ahead today. The capacity of one single server is often so large that is it very inefficient to be used for one single purpose, and there the concept of time-sharing comes back. A really good way to improve utilization of the server is through virtualization.[8] [9] [7]

3.3.2 Hypervisor

A hypervisor is the piece of software that controls and allows you to run multiple operating-systems on one computer. This hypervisor creates a virtual platform, and on top of this platform you create the virtual machines. The hypervisor then manages and monitors the VMs. There are two fundamental types of hypervisors, and they are similar in many ways, but they are actually quite different. [10]

Classification	Characteristics and Description
Type 1: native or bare metal	Native hypervisors are software systems that run directly on the host's hardware to control the hardware, and to monitor the guest operating systems. Consequently, the guest operating system runs on a separate level above the hypervisor. Examples of this classic implementation of virtual machine architecture are Oracle VM, Microsoft Hyper-V, VMWare ESX and Xen.
Type 2: hosted	Hosted hypervisors are designed to run within a traditional operating system. In other words, a hosted hypervisor adds a distinct software layer on top of the host operating system, and the guest operating system becomes a third software level above the hardware. A well-known example of a hosted hypervisor is Oracle VM VirtualBox. Others include VMWare Server and Workstation, Microsoft Virtual PC, KVM, QEMU and Parallels.

Table 3.1: LLC [10]

3.3.3 Hypervisor technologies

There are, as indicated before, quite a lot of different hypervisors to look at. The benefits from native or bare metal versus hosted hypervisors are also quite comparable.

3.4 Stateless APIs

What does it mean that something is stateless?

Put in simple terms, stateless means that every request must contain all the information needed for the API or server to understand the request. [11]

You must treat the server like it has no memory, it will never remember the client or the clients information. In the design process of the API or server, you can never

use sessions or similar methods of containing information about a client.

A way to solve this is to use tokens. A token can be generated to contain certain information, and be time-based. If used correctly, this can be a perfect way to communicate with a stateless API.

A stateless API is often referred to as a REST-API, or RESTful API.

3.5 Stateless

3.5.1 The session state

Traditionally, a web application would use *remote sessions*. In this approach, you would save the application state on the server, and the client would not need to worry about this. See the following quote from Roy T. Fielding's dissertation:

The remote session style is a variant of client-server that attempts to minimize the complexity, or maximize the reuse, of the client components rather than the server component. Each client initiates a session on the server and then invokes a series of services on the server, finally exiting the session. Application state is kept entirely on the server. [11]

While this approach is great in many cases and does indeed introduce some advantages, it does reduce the scalability of the server:

The advantages of the remote session style are that it is easier to centrally maintain the interface at the server, reducing concerns about inconsistencies in deployed clients when functionality is extended, and improves efficiency if the interactions make use of extended session context on the server. The disadvantages are that it reduces scalability of the server, due to the stored application state, and reduces visibility

of interactions, since a monitor would have to know the complete state of the server. [11]

3.5.2 The stateless constraint

Using the REST architectural style you need to follow a set of constraints that include the *statelessness of the server*. According to Fielding, the REST *stateless constraint* is defined as the following:

5.1.3 Stateless

Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

This constraint brings in the properties of *visibility*, *reliability*, and *scalability*:

Visibility is improved because a monitoring system does not have to look beyond a single request datum in order to determine the full nature of the request. Reliability is improved because it eases the task of recovering from partial failures. Scalability is improved because not having to store state between requests allows the server component to quickly free resources, and further simplifies implementation because the server doesn't have to manage resource usage across requests. [11]

3.5.3 Authentication and authorization

Well, if you don't keep anything on the client. How do you intend to authenticate the client?

If the client requests protected resources that requires authentication, every request must, as Fielding wrote, contain all necessary data to be properly authenticated/authorized. A quote from RFC 7235 [12]:

HTTP authentication is presumed to be stateless: all of the information necessary to authenticate a request **MUST** be provided in the request, rather than be dependent on the server remembering prior requests. [12]

And authentication data should belong to the standard HTTP Authorization header. From the RFC 7235:

4.2. Authorization

The Authorization header field allows a user agent to authenticate itself with an origin server – usually, but not necessarily, after receiving a 401 (Unauthorized) response. Its value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested. [12]

In the end, for authentication, you could use Basic HTTP Authentication. This will transmit the username and password to the server encoded using Base64:

Authorization: Basic <credentials>

This is of course insecure, especially if it is sent over normal unencrypted HTTP. Instead, one would use something like a token. The token can be generated based on metadata, an expiration date and other information that might be considered important for the application:

Authorization: Token <token>

3.6 Technologies to choose from

3.6.1 Perl

Perl (Practical Extraction and Report Language) is really an old and true programming and scripting language. The Hypervisor we intend to use is has a long and trusted use of this language to script and automate the processes. [13]

3.6.2 Python

Python by it self is a very good programming language with very versatile options. Although it is often better to use frameworks to deal with web requests instead of "re-inventing the wheel".

3.6.3 Django

Django is a Python framework. Django deals with the HTTP and Web requests and has a lot of built in features.

This is a very popular choice now a days. With a very versatile way to work and an easy workflow this would probably be a good choice. The issue here is that this takes a normal approach where backend and frontend is not decoupled, and the system will therefore also not be stateless.

You could have used Django for the RESTful API, but this would be very overkill and thus created a system that would probably be a memory hogger[14] if you were to compare this to other frameworks.

3.6.4 Flask

Flask is also a Python framework. Flask is what you would call a lightweight framework created with APIs and simple websites in mind. The framework is optimized to use little resources when it runs, and it is very simple to work with.

3.6.5 PHP

PHP is a programming language that gets a lot of hate, but also a lot of love. Today it is still the most used web programming language out there,[15] and with the proper tools the programming language can be used for both decoupled purposes and for normal coupled purposes. PHP is usually either used as a functional based or an object-oriented programming language, and it is also normal to implement the MVC-method of thinking.[14]

However, PHP is also considered a dinosaur. This is a programming language that, even though it is very well maintained, it is old and dusty. There is a lot of deprecation, which can break your project down the line. [16]

3.6.6 Node.js

Node.js is a runtime environment for JavaScript execution. It lets you use your JavaScript knowledge to create backend-applications. Node is sadly not always a good choice for backend applications because it is single threaded, so all instructions must be executed in sequence. This leads to bad scaling. [17]

3.6.7 GoLang

GoLang is a programming language created by Google, and can in many ways remind you of C. In fact according to "The Go Programming Language"

Go is sometimes described as a "C-like" language," or as "C for the 21st century"

[18]

It is considered to be one of the fastest programming languages, and is a very good choice for backend services. But, since it is so new there are things missing like documentation and some libraries. It can also be hard to maintain since not many use it in production at the moment. [18]

3.6.8 React

React is a front-end framework based on TypeScript and JavaScript and can be a really good choice for front-end applications. React is simple to work with and will give great results. The React frontend can be used in combination with a RESTful API based on any technology you would wish for. [19]

3.6.9 Angular

Angular is a front-end framework similar to React. It uses TypeScript, which it compiles to become JavaScript. The compiler is based on Node.js, but after you compile it you can take Node.js completely out of your stack.

Angular is a great choice for frontend, it is very versatile and is easy to customize. It works perfectly as a de-coupled frontend together with a RESTful API.[20]

3.7 Data storage

There is a lot of options when it comes to storage. First, we need to map what data we are going to store.

3.7.1 Data mapping

A few of the things we are storing are:

- User information (no passwords)
- Virtual Machine information
- Virtual Machine Queue
- Template information

Now that we have mapped all the data that has to be stored, we can start looking at solutions that would give good enough performance and good data reliability.

3.7.2 A database

With a database you will have a well-organized collection of data, great for storing user information or similar data with relations.

3.7.3 Data stores

All of our data is relational data, with potential foreign keys in almost all tables. The natural data storage method would be a relational database, but the data could also be stored in flatfile or similar.

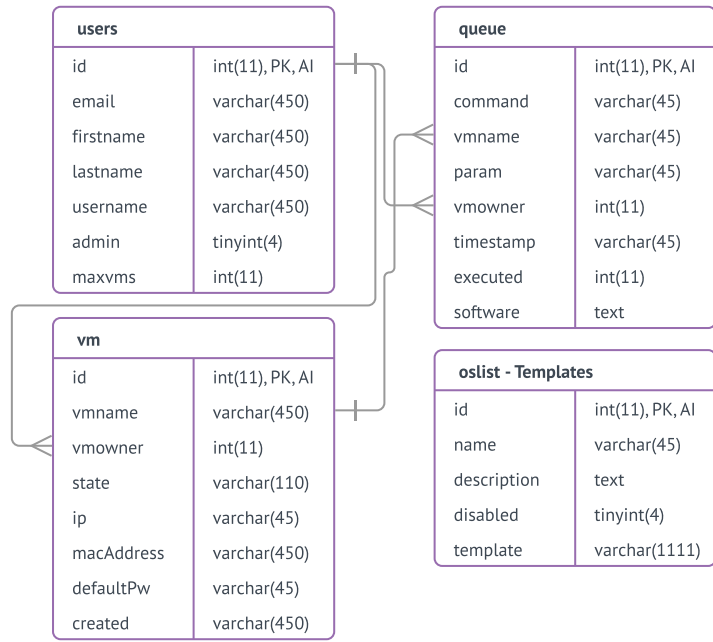
Flatfile database

A flat file database is normally a database that consists of files stored directly on the hard-drive. There are many downsides to this, among them are bad performance and no integrity. [21]

Relational databases

Relational databases is normally based on tables with information that contains structured data with pre-defined data-fields. A relational database is easy to keep the integrity of, and normally has very good performance. Examples of free relational databases are MySQL and MariaDB. [22]

3.7.4 Database schema



This is a theoretical implementation of a database schema with a relational database, in this case MySQL, as the database system manager. [22]

3.8 Hardware

Since the hardware is pre-existing infrastructure, we are only going to describe the hardware needed. For virtualization, you need hosts that can be the hypervisor. For stability and high availability we need a few servers, such that in a case where one server goes down there is not a major outage in the virtualization. [23]

3.8.1 Existing hardware

The existing setup includes 4 VMWare ESXi hosts, 40TB of storage and 2x10Gbit/s networking. This setup is good, but not perfect. A few of the flaws include:

- Next to no High Availability setup, if a host goes down your server goes down with it.
- Single-point-failure for storage, a VM is only stored one place.
- No automatic backup system.

3.8.2 Improvements to existing hardware

The hardware should be improved with better high availability setup, including 2 or 4 (in pairs) servers that are completely the same. The reason they should be the same is so that they are good matches when it comes to configuring high availability. [24]

The storage solution is also too slow. A solution to better this is to add SSD disks for cache. Also, the storage has the same weakness as the servers - single-point-failure will take it down. Adding another storage solution that mirrors the first might be a solution.

Chapter 4

Technical Documentation

4.1 Introduction

In this chapter we will take a look at the technical documentation of the finished product, as well as some of the issues and challenges we had. This chapter contains for the most part only statements based on our experience with the software involved and the development performed.

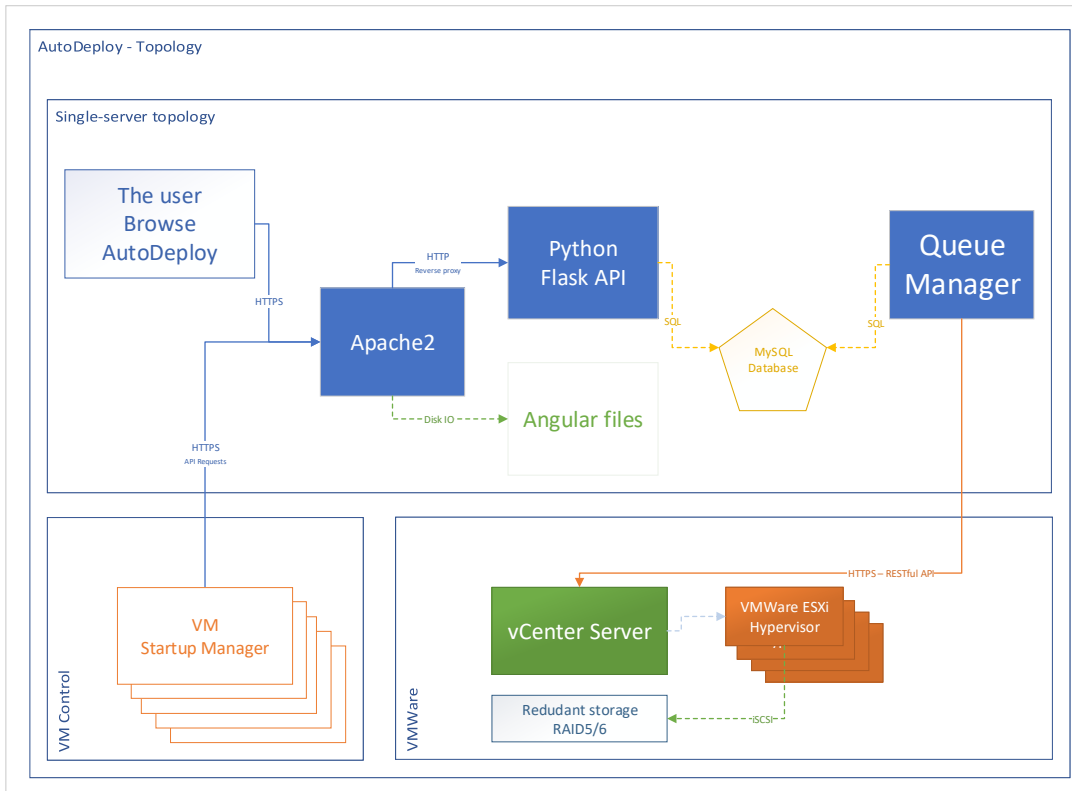
4.2 The big picture

The application we created is based on de-coupling the front-end and back-end. This means in practice that we are developing two, or more, individual systems that will communicate with different techniques.

4.2.1 The infrastructure

The application is incredibly easy to create a good and stable infrastructure for. The front-end needs a standard HTTP-server, such as Apache2 or nginx. The back-end needs Python and a few dependencies.

Below is the topology for the service in production. This is very generalized, but it shows how the traffic egresses through the topology, and how things are connected. The illustration below is for a single-server topology. The application is very much able to handle scaling with a multi-server topology, or with containerization of the service, but this is out of scope.



The production environment for this application is based upon a single Ubuntu Linux server with Apache2 and MySQL installed. The AutoDeploy-API and AutoDeploy-Queue is also installed as systemctl service on this server.

4.3 The API

The AutoDeploy-API is a Python Flask application that handles all API requests for this application. It is also the only place in the web-application that talks to the SQL server. The API handles requests from users through the Angular front-end, and requests from the VM Startup Manager that runs on the provisioned virtual machines. The API is RESTful and stateless.

4.3.1 API Endpoints

Table 4.1: API Endpoints

Endpoint	POST Data	Headers	Response	Protected
Authentication				
/login	username, password	none	user (array), token(string)	No
/verify	none	Auth	information	Yes
Getters				
/getuser	None	Auth	user (array)	Yes
/getvms	None	Auth	vms (array)	Yes
Deleters				
/deletevm	id (VM id)	Auth	information	Yes
Adders				
/addqueue	command, vmname, param, software	Auth	information	Yes
/createvm	vmnwame, template, software (json)	Auth	information	Yes

The information response looks like this:

Response: error or success, like this:

```
{ 'failed':false, 'info': 'Echoable text to show user.' }
```

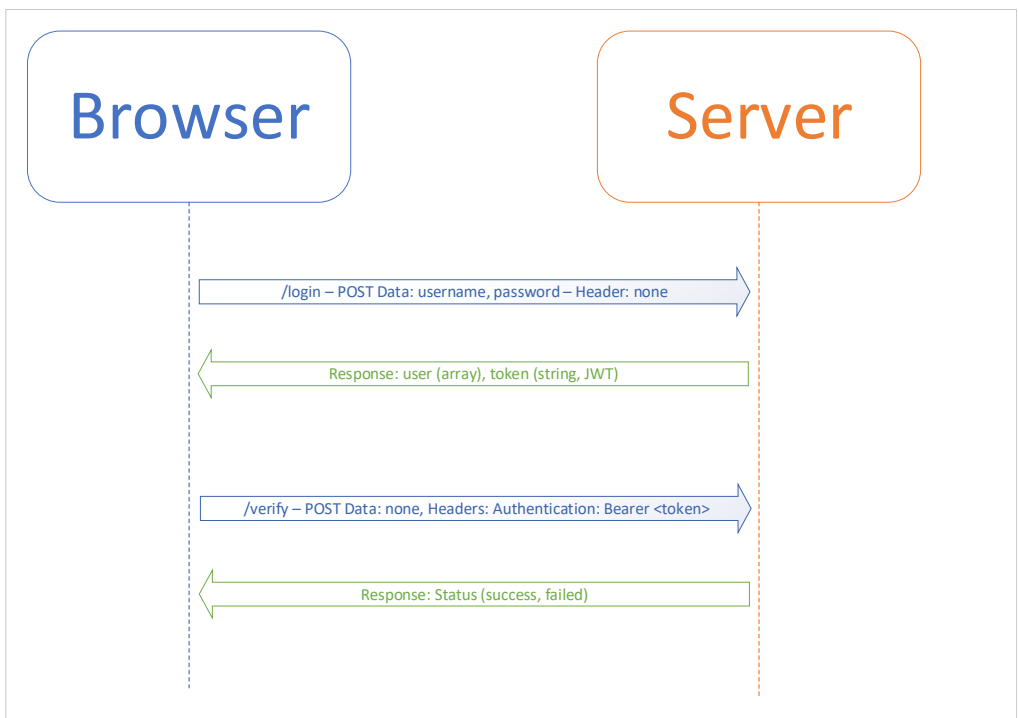
4.4 Authentication

Because the API is stateless, the implementation of authentication must be done correctly and with precision. A good token-implementation is required.

The application implements JSON Web Token. This is a token algorithm that lets you create a token with a payload. When the user successfully logs in using their NTNU credentials, a JSON Web Token will be returned and must be saved locally, either in local storage or in a cookie.

Every time the user wants to communicate with the API, the user agent (usually the browser) will need to include the authentication header in the request. The Authentication header is explained in more detail in section 3.4.3.

Example of communication between the browser and the API Server:



In the first step, the user agent sends a login request to the `/login` route of the API.

This endpoint requires POST data to be submitted with the username and password of the user. This route is not protected by any token requirements because of the nature of the route. In the response of this request, if the user authenticated successfully, there will be returned an JSON Web Token.

4.4.1 How does JWT work?

The structure of JWT is quite simple. The token contains 3 parts, separated with dots (.). They are[25]:

- Header
- Payload
- Signature ...

A typical JWT token looks something like this:

```
xxxxxx.yyyyy.zzzzz
```

The header

Consists of two parts - the type of token, and the hashing algorithm being used (such as HMAC SHA256 or RSA).

```
{ "alg": "HS256", "typ": "JWT" }
```

The JSON is Base64Url-encoded to form the first part of the JWT.

The payload

The payload consists of claims. Claims are data about an entity, usually the user and related metadata. There are three types of claims; registered, public and private claims.

- **Registered claims:** Predefined claims, things like issuer, expiration time, subject and audience.
- **Public claims:** Information that is defined by those using JWTs.
- **Private claims:** Custom claims, created to share information between parties.

An example payload could be:

```
{ "sub": "1234567890", "name": "John Doe", "admin": true }
```

The payload is then, like the header, Base64Url encoded.

It is important to know that even though the information in the JWT token is tamper-proof, it can be read by anyone that has obtained the token. Do not use the payload to transverse secret information without encrypting the information.

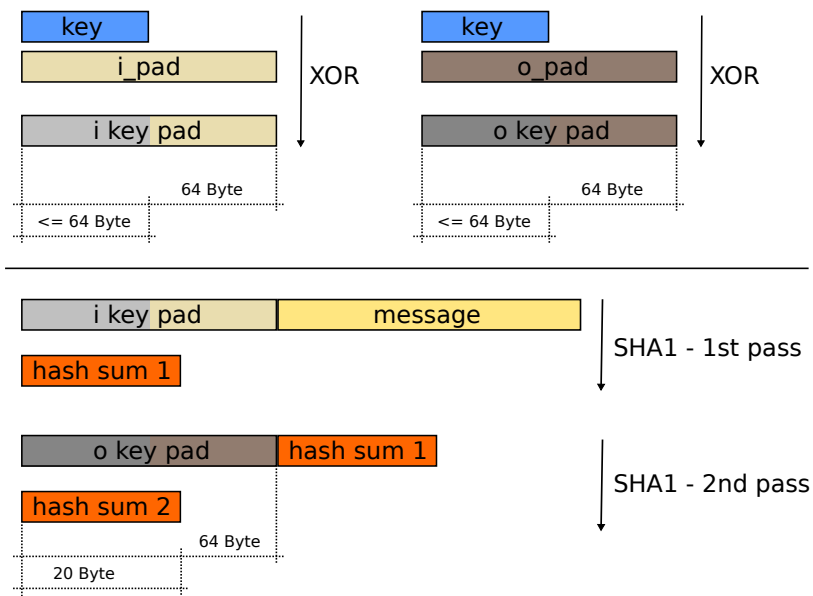
The signature

The signature is used to verify that the token is not tampered with. The signature in our implementation of JWT is a HMAC SHA256 algorithm.

Implemented like this:

```
HMACSHA256 base64UrlEncode(header) + "." +  
base64UrlEncode(payload),  
secret
```

The HMAC SHA256 algorithm can be illustrated like this:



Please understand that the algorithm is out of scope of this paper.

The end result

The output of the three parts are Base64-URL strings separated by dots. It is easy to work with, and can easily be passed in HTML and HTTP applications.

```

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
    
```

4.4.2 Why use JWT?

There are many benefits from using JWT, instead of something like Security Assertion Markup Language (SAML). One of them is size. JSON is less verbose than XML, and that makes it so that when JSON is encoded its size is smaller. This makes JWT a good choice to be passed in HTML and HTTP environments.

[25]

SAML and JWT use the same signing-method, where you use a public/private key pair in the form of a X.509-certificate for signing. This is also a place where JWT comes out ahead of SAML.[25]

Size comparison

JWT

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMRhHDcEfxjoYzgeFONFh7HgQ
```

Decoded

HEADER: ALGORITHM & TOKEN TYPE
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>
PAYLOAD: DATA
<pre>{ "sub": "1234567890", "name": "John Doe", "admin": true }</pre>
VERIFY SIGNATURE
<pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret) <input type="text"/> secret base64 encoded</pre>

SAML

SAML

[Follow @auth0](#) 3,150 followers

Debugger

SAML ENCODED

SAML DECODED

Prettyfy (not editable) Expand

```
PHNhbWxwOUIic3BvbmlHhthbG5zOnNhbWxwPSJ1cm46b2FzaXM6b
mFlZXM6dGM6U0FNTDoyLjA6cHJvdG9jb2wllEIEPSJlNjlxYzRjNGFIN
WQ2MGM3NjhjYzllCBWZkzJaW9uPSlyLjAilElzc3VlSW5zdGFudD0iMj
AxNC0xM0C0xNFQxND0zMjoxNtoilCBZEXN0aW5hdGlvbjoiaHR0cHM
6Ly9hcHAuYXV0aDAuY29lL3Rlc3Rlc3Rlc3Rlc3Rlc3Rlc3Rlc3Rlc3
WVYlHhthbG5zOnNhbWw9InVybjpYXNpczpuYWllc2p0YzptQU1MOJlu
MDphc3NlcnRpb24lPnVybjpYXR1Z2l0LmFldGgwLmNvbTwwc2FtbDpJ
c3N1ZlXI+PHNhbWxwOIN0YXR1cz48c2FtbHA6U3RhdHVzQ29kZSBW
YWx1ZT0idXJlOm9hc2lzOm5hbWVzOnR5b291bnR5b291bnR5b291bnR5
c2pTdWVhZjZlbnR5b291bnR5b291bnR5b291bnR5b291bnR5b291bnR5
aW9uHhthbG5zOnNhbWw9InVybjpYXNpczpuYWllc2p0YzptQU1MOJlu
uMDphc3NlcnRpb24lPnVybjpYXR1Z2l0LmFldGgwLmNvbTwwc2FtbDpJ
VWtrYVZlZyZyNGJyRjRBERzVFM1g3NllgSjNzdWVJbnN0YV50PSlyMD
E0LTEwLkE0VDE0OjMyOjE3LjltMVoipjxzYWt5Okkzc3Vlcj5lcm46bWF
0dWdpdC5hdXR0MjC5jb208L3NhbWw6SjNzdWVYpXzTaWduYXR1cm
UgeG1sbmM9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNp
pZyMipXzTaWduZWRJb2VpXzYV5vbmliYXNpczpuYWllc2p0YzptQU1MO
kIEF5Z29yXRobT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMjM0NTY3
WwtZXh1LWwMxNG4jllB+PFNpZ225hdHVyU1ldGhVZCBBbGdvcml0aG0
9Imh0dHA6Ly93d3cudzMub3JnLzlwMDAvMDkveG1sZHNpZ225hdHVyU1
2hhMSVpXzSZWZlcmVvY2UgVWVJPSijXzVW5zdWVdG6G6G1Va2thUXV
XNlOYnJGJMERHNUUzWdCz4JH4VJHbnNmb3J3c3Rlc3Rlc3Rlc3Rlc3Rlc3
```

```
1 <saml:Response
2 xmlns:saml="urn:oasis:names:tc:SAML:2.0:protocol" ID="_621c4
3 <saml:Issuer
4 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">urn:matu
5 </saml:Issuer>
6 <saml:Status>
7 <saml:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status
8 </saml:Status>
9 <saml:Assertion
10 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="
11 <saml:Issuer="urn:matugit.auth0.com"><saml:Issuer>
12 <Signature
13 xmlns="http://www.w3.org/2000/09/xmldsig#"
14 <SignedInfo>
15 <CanonicalizationMethod Algorithm="http://www.w3.org/2
16 <SignatureMethod Algorithm="http://www.w3.org/2000/0
17 <Reference URI="#_5VK7LT7FiiUkkaQuW6i4brFODGSE3
18 </Transforms>
19 <Transform Algorithm="http://www.w3.org/2000/09/x
20 <Transform Algorithm="http://www.w3.org/2001/10/xr
21 </Transforms>
22 <DigestMethod Algorithm="http://www.w3.org/2000/09
23 <DigestValue>ZdKfG03H1Tu50hawzQVjjsACzJwc=</Di
24 </Reference>
25 </SignedInfo>
26 <SignatureValue>fGpt7AaHcMe2GtA158achvGQVqDwHSH
```

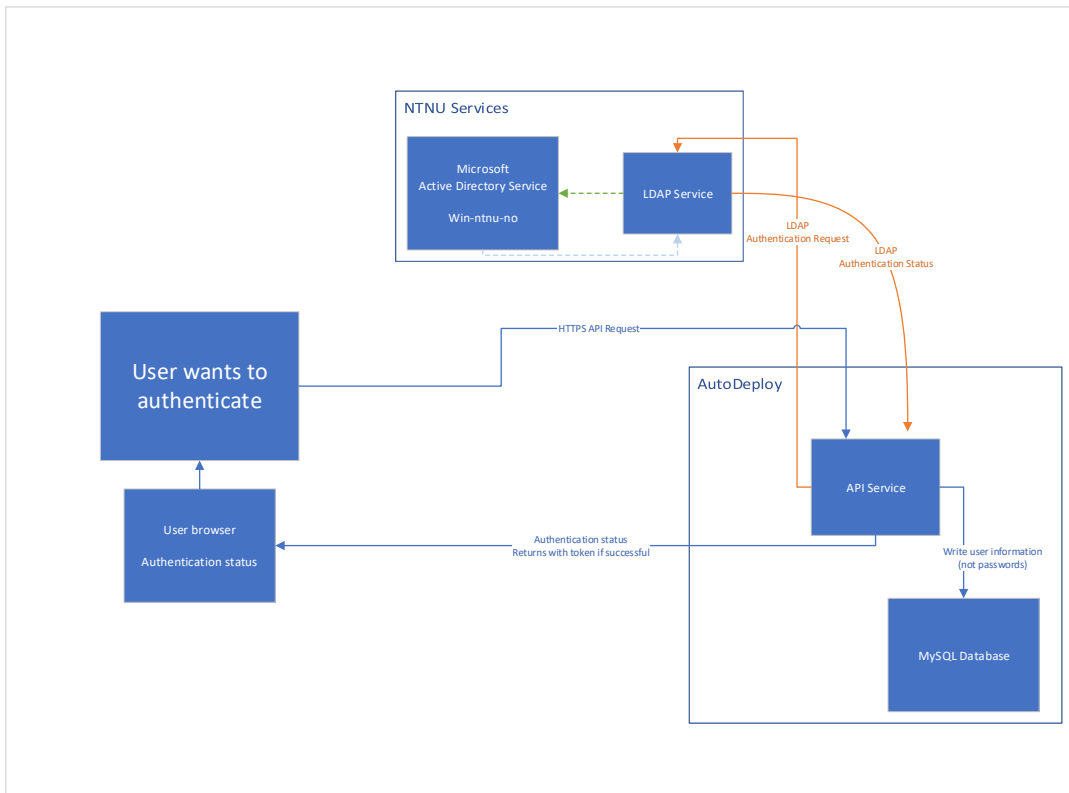
The SAML version is 6856 characters, and the JWT version is 155 characters.

Quite the difference! Imagine every single HTTP request has to carry this weight. JWT is quite a lot lighter to use.

4.4.3 Authentication directories

Actual authentication is handled by a user directory. We are using Lightweight Directory Access Protocol, LDAP, to speak with a Microsoft Active Directory server. This is because we want the users of NTNU to authenticate with their normal passwords. We do not store the passwords themselves, only a user reference and some cached user information.

The implementation of this is as follows:



Alternative authentication methods

We could also have implemented other user directories or other methods of authentication, like Feide. Feide was not implemented because of the very time-consuming implementation period, and would have added a complexity that is not needed for this project. [26]

But as of now, the only way to authenticate is via LDAP.

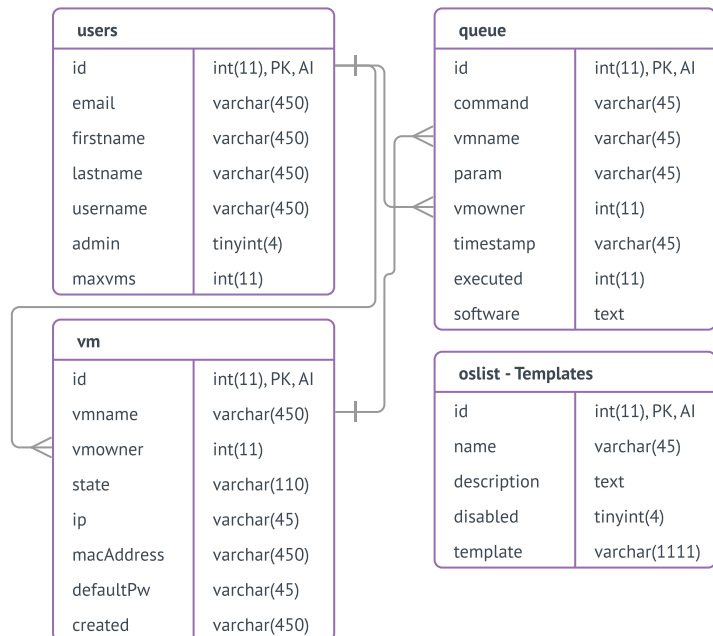
4.5 Data storage

4.5.1 Structured data storage

For the structured data, we are using a relational database. We are using MySQL as the relational database systems manager. We chose a relational database because of the nature of our data. [22]

Database structure

As referenced earlier in the paper, this is the database structure that is implemented in this project:



4.5.2 File storage

There are not a lot of file storage used in the main solution of AutoDeploy, except for the templates and the virtual machines.

The templates and virtual machines are all stored on the VMWare Storage Solution, which is one or more iSCSI targets mounted on all the ESXi hosts. The templates are not stored on the webserver and the webserver can normally not reach these templates.

4.6 Creating a virtual machine

There are a few things that are key elements involved in creating a virtual machine, excluding AutoDeploy. These are:

- The hypervisor and all related services
- The VM template
- The VM startup agent
- The QueueManager

4.6.1 The VM Template

Normally, in VMWare, you can create a virtual machine using 2 methods. Blank virtual machine and manually installing an operating system, or via a virtual machine template.

A virtual machine template in VMWare is in short a virtual machine that has been installed with the needed software, and then converted into a template - then a template can be used to create many clones. The process to create new virtual machines from this template is called Cloning, and in practise is basically like cloning a virtual machine.

Because we are not creating standard virtual machines in VMWare, we need some specialized software installed, in addition to the standard applications installed. One of the applications installed on all the templates, and therefore also on all Linux AutoDeploy Virtual Machines, is Fail2Ban. Because of the fact that all the

Virtual Machines are assigned a public IP address, we employ fail2ban to make sure that there is no easy way to brute force the SSH connection. With the standard configuration of fail2ban, you will be blocked permanently from connecting over SSH after the third failed login attempt. Harsh, but necessary.

The specialized software installed is the VM Startup Agent, an application developed to handle simple requests from AutoDeploy.

4.6.2 The VM Startup Agent

A startup agent is something that is very common among virtual machines. Usually this is an agent that gives the hyper-visor or the management software access to manage the virtual machines operating system. [27]

AutoDeploy has to be able to control the deployment of the virtual machines, and this is done via this agent. After a virtual machine is created, the queue manager will populate the MAC Address table and from there the virtual machine will be able to pull its config from the API. The Agent makes sure that there is no manual action required from system administrators before the user requesting the virtual machine can log in and start utilizing the resources assigned to the machine.

In addition to the AutoDeploy agent we have the standard VMWare Tools installed, this is mainly to make sure the virtual machine has the required drivers installed, but also so that the hypervisor can shutdown and control the virtual machine better. Without VMWare Tools, the hypervisor has only one option to shut down the VM, and that is to kill it.

4.6.3 The Queue Manager

When a user clicks a button in the UI, usually the action that is performed is a queue action. This means that the UI/API creates a queued task, and it is the QueueManagers job to crawl these tasks and perform them. In short, the queue-manager is the Perl script that communicates with VMWare via the VMWare Perl

SDK.

The basic process of the queuemanager is a loop that waits for tasks that are not executed. If there exists such tasks, the task will be executed in the manner described in the description of the task.

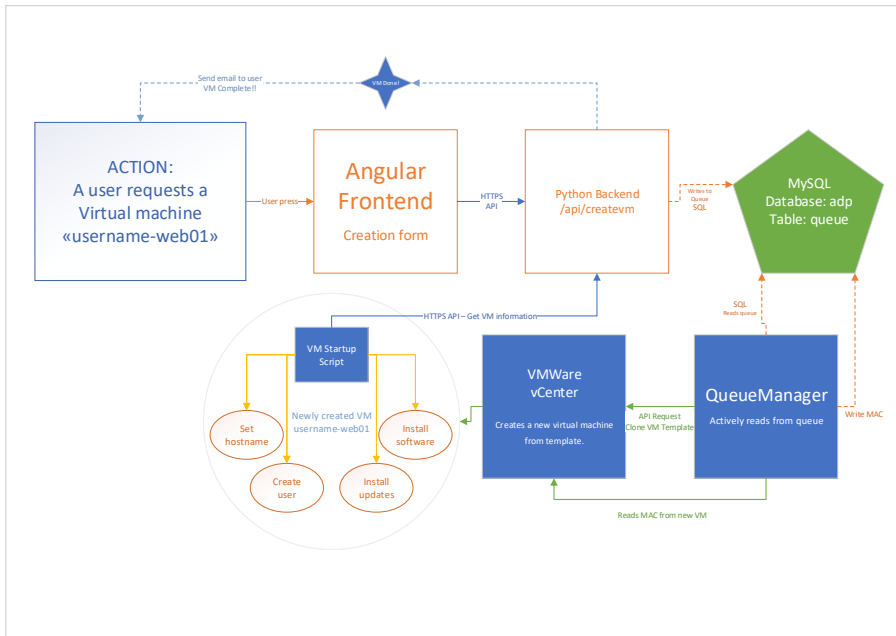
A Queued task

A queued task contains the following information:

- Command - what command to perform
- VM Name - what virtual machine is the command to be performed on.
- param - Additional parameters, like template name.
- Status - Is the task executed?
- Software - What software should be installed?

The status is set by the QueueManager when the task is executed.

4.6.4 Creating a VM - Diagram



4.7 Foreign keys in VMWare

Our implementation of deployment in VMWare uses the virtual machines name and MAC address as foreign keys.

The only identification from the AutoDeploy QueueManager and the vCenter API is the name of the virtual machine. It is because of this that we have to be very careful not to have any possibility of two virtual machines being created with the same name. This would result in double action.

The other foreign key is the MAC Address of the main ethernet interface of the deployed virtual machine. We remove the separators and we are left with a unique string.

Example of MAC address with separators removed:

E4A4715DE968

The MAC address is used as foreign key when the AutoDeploy Startup agent communicates with the API. This is because it is possible to retrieve the MAC from the VMWare API before the VM is completely deployed.

4.8 The user panel - Front end

4.8.1 Quick overview of front ends

The front end in software development is the part the end user will see and interact with. When using the front end, it will make requests to the back end which handles the logic of the application. The received data is then presented to the user. [28]

Front ends are at their core html. It is the base of any website regardless of other tools. It describes the structure of webpages and links data together with hyper-text. [12] Then a CSS file is added to control the look of the site, for example the background and fonts. But neither are very good at handling dynamically updating content. We solve this by using another cornerstone of web development, JavaScript.

JavaScript is a high-level scripting language that enables dynamic updating of the content being presented. It controls multimedia, animation and is a staple of the web. But since most sites use similar elements it makes sense to not reinvent the wheel every time one is made. Therefore, we add a framework on top of JavaScript so we can create a beautiful site without thinking too much about setting up build systems etc. But in the end the code is compiled to JavaScript, and everything is made up of these 3. (Html,stylesheet, js). [29]

4.8.2 Angular front end

The frontend is written in Typescript and uses Angular 4 as a framework. Typescript is a superset of JavaScript meant to make writing faster, and the code more readable. We use Angular because it does a lot of work in the background to make life

easier for the developer.

Angular uses what we call modules too provide context for components such as application domain. Meaning you can tie together a component on your site, like a form or a text field, to a service such as our API. [29]

All Angular applications have a root module. this is the entry point; all the subsequent functionality is imported to this module. When you navigate within the site, for example /dashboard in our project, then the dashboard module is loaded by a service called router. The router is responsible for navigation within the pages of the application.

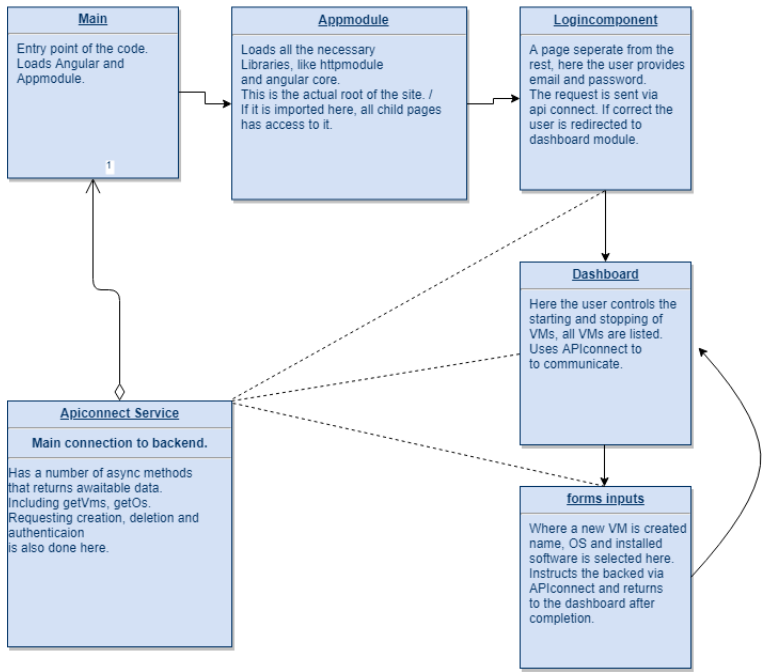
Each "page" has its own html, css/scss files. These are injected into the main html file so that the other components do not have to be implemented for each page. Having a menu, header and imports for each page is less effective than just putting the page elements you wish to display at a given moment into a single view. [29]

To visualize this, one can think of it like a collection of webpages put into a single webpage. And you can modify which pages are viewable as you please.

When the user wants to create a VM we simply remove the table containing the VMs and present a form with options instead. Nothing else has changed, and the time spent getting the other components is saved. When the form is filled and the user clicks the create button a call is made to the server via the apiconnect class, containing the options for the new VM.

This is done asynchronously so the rest of the page is not "frozen" until we get a response. When the response does arrive, give the user a "toast" notification and send them back to the dashboard.

The dashboard of course has its own functions that are executed when it is loaded. These include retrieving the VMs and displaying them in a table. Each table item has start, stop and delete functions.



Here we see an overview of the flow of the main components of the program. First, logging in. Dashboard. And VM creation. The arrow from the "forms inputs" represents getting sent back to the dashboard after completing a VM.

AutoDeploy

A system for NTNU Students, by NTNU Students.

Create your own virtual machine out of templates, and use them for school projects.

NTNU LDAP Account

We'll never share your email with anyone else.

Password

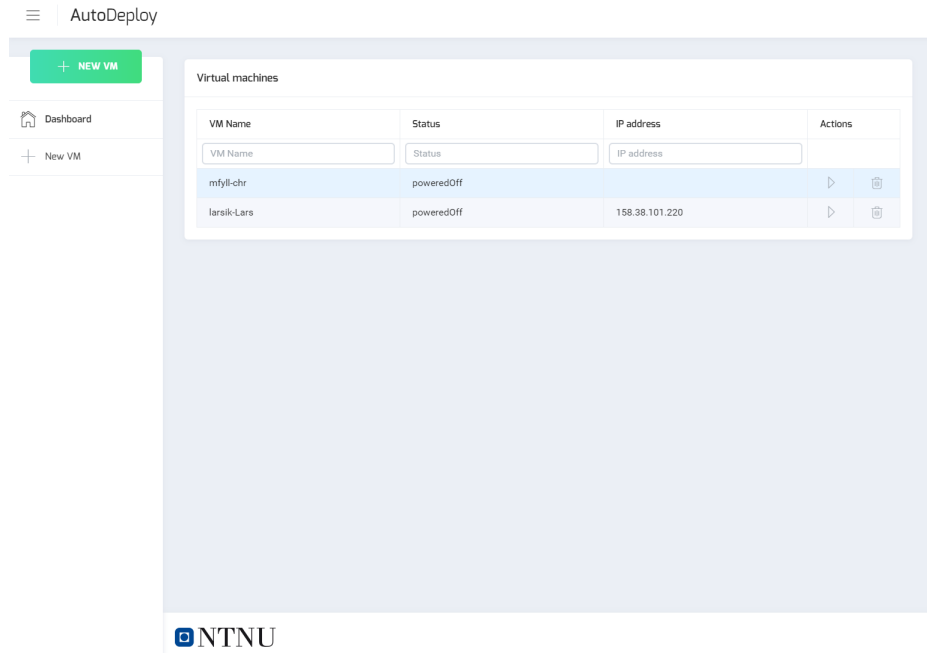
We do not store your password here, it is only used for authentication against NTNU servers.

If you have forgotten your password, or are unable to log in, please contact NTNU Orakel for support.

LOG IN

The first page presented to the user is the logincomponent. Here the user provides a LDAP account name and password. This class is encapsulated and not accessible

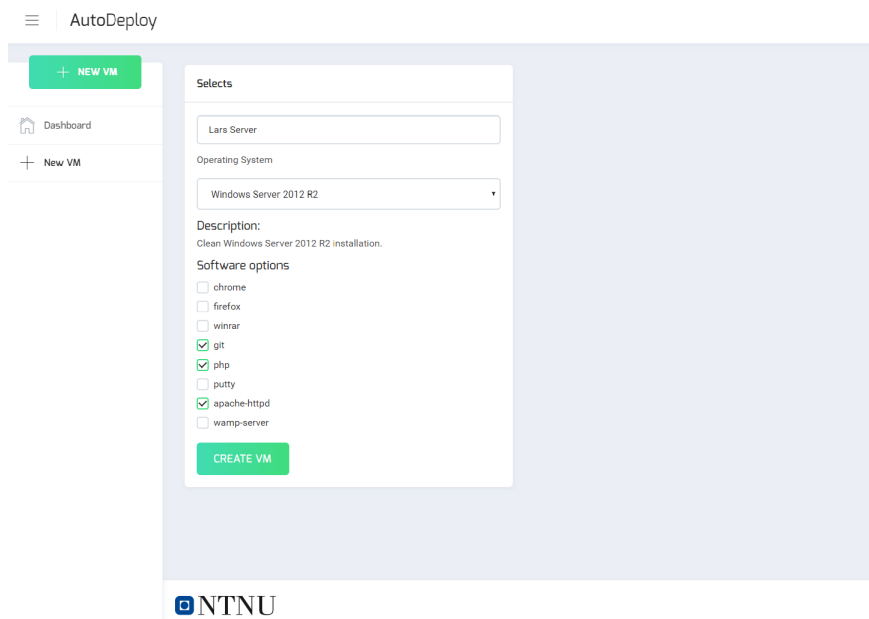
to other components.



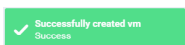
The screenshot shows a web interface for 'AutoDeploy'. On the left is a sidebar with a 'NEW VM' button and navigation links for 'Dashboard' and 'New VM'. The main area is titled 'Virtual machines' and contains a table with columns for 'VM Name', 'Status', 'IP address', and 'Actions'. The table lists two VMs: 'mfyll-chr' with status 'poweredOff' and 'larsik-Lars' with status 'poweredOff' and IP '158.38.101.220'. Each row has a play button and a trash icon in the 'Actions' column. The NTNU logo is at the bottom left.

VM Name	Status	IP address	Actions
<input type="text" value="VM Name"/>	<input type="text" value="Status"/>	<input type="text" value="IP address"/>	
mfyll-chr	poweredOff		▶ 🗑️
larsik-Lars	poweredOff	158.38.101.220	▶ 🗑️

Here we see the dashboard which the user will be sent to after the login completes. The table displays all the VMs the user has access to. The top of each column has a search field so you can search by name, ip and status. The far right column houses the action bar. Here you can start, stop and delete the existing vms. Pressing the "new vm" buttons brings up the vm creation interface.



The new vm interface (forms.inputs in flow overview diagram) lets the user set the create vms with a specified set of preferences. First, the name of the vm. Then the user can set the operating system he or she wants, and select software to be pre-installed from a list of choices. The available software of course varies based on the operating system selected. When the server responds after creating the vm and installing the software, a "toast" message is displayed to the user and the dashboard replaces the new vm interface.



(Toast message)

4.8.3 Front end API calls

To be able to display any data, you need to fetch it somehow. So the front end has a set of calls that it communicates with the back end with. It requests and modifies it with a set of "Application Interface Calls" that are all defined and executed from our "API connect service" class.

Table 4.2: API Service methods

Request Method	POST Data	Headers	Response	HTML Method
Authentication				
login	username, password	none	user (array), token(string)	Post
verify	none	Auth	information	Post
Getters				
getOs	None	Auth	user (array)	Get
getVms	None	Auth	vms (array)	Get
Delete				
deleteVm	id (VM id)	Auth	information	Post
Adders				
addQueue	command, vmname, param, software	Auth	information	Post
createVm	vmnwame, template, software (json)	Auth	information	Post

Chapter 5

Manual

User, administration and maintenance manual

5.1 User manual

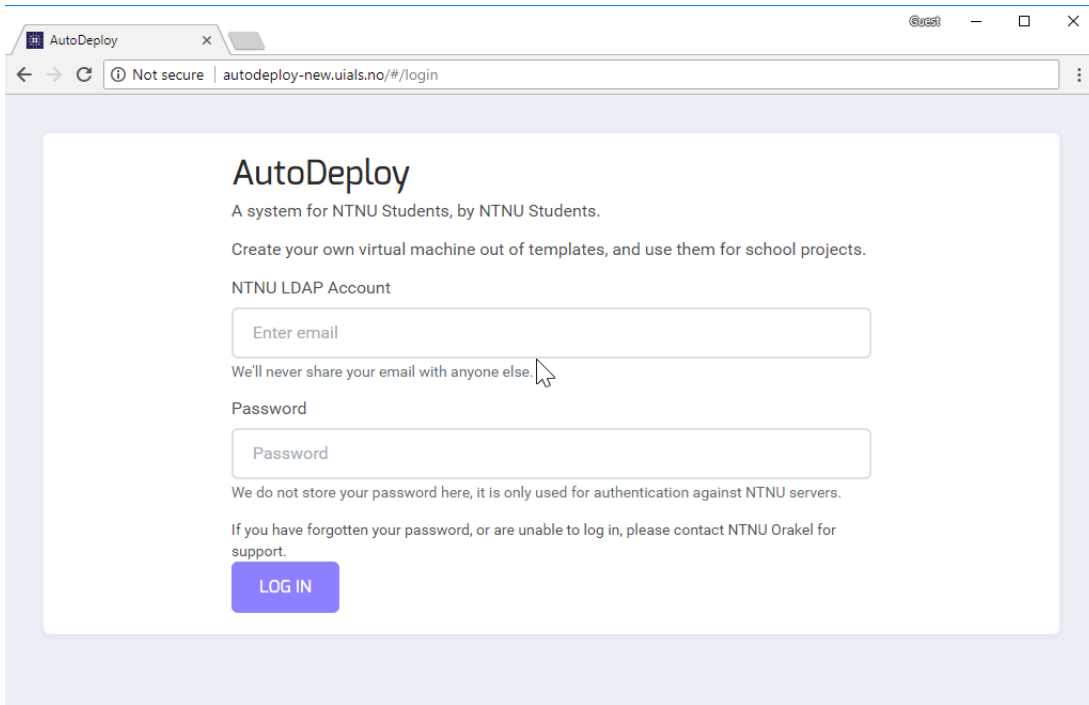
5.2 Introduction

This chapter contains the user documentation, administration documentation and maintenance manual for the AutoDeploy system. A short summary of how to use the system, and how to do things like creating new templates or maintaining the adp-server.

5.2.1 Logging in to AutoDeploy for the first time

Navigate your browser to **<https://autodeploy-new.uials.no/>**. The browser will ask for your NTNU credentials. Enter your NTNU username and password. You are now logged in.

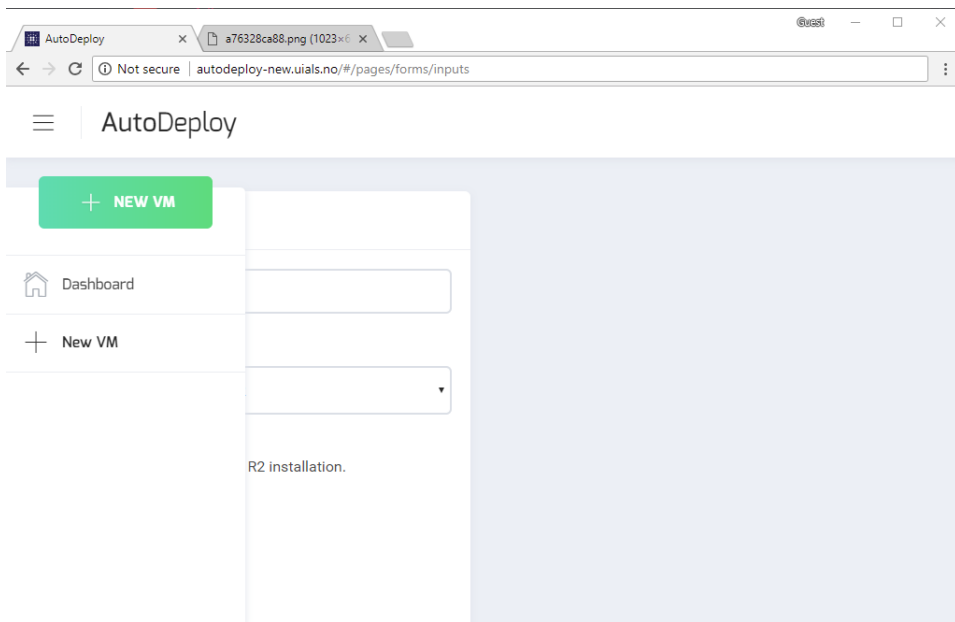
The NTNU username and password is the same as when you log in to Eduroam, or when you use FEIDE. The username normally is the same as the prepending part before the @ in your NTNU email address.



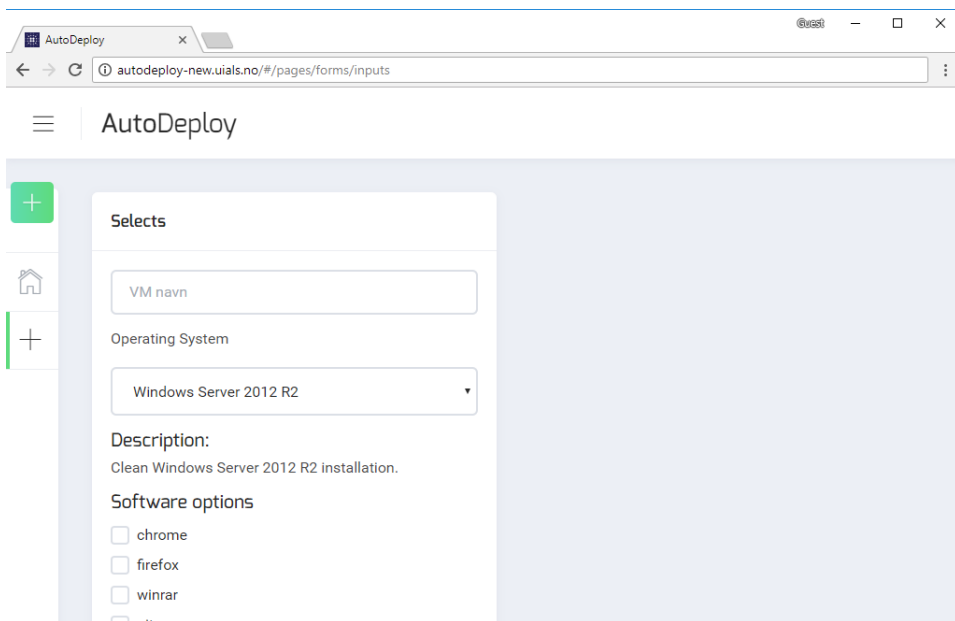
5.2.2 Creating your first Virtual Machine

After you have logged in, you can start creating virtual machines. Normally, you can create a maximum of 5 virtual machines, but this can be adjusted on a per-user basis.

To create your first virtual machine, you click the plus-icon on the left-hand side. This will, if you are on mobile or on a device with low screen resolution, bring up a menu.



Click on **New VM**. This will bring you to a form that you will be able to fill in to create your virtual machine.



First, you fill in the name of your new virtual machine. This can be anything, as

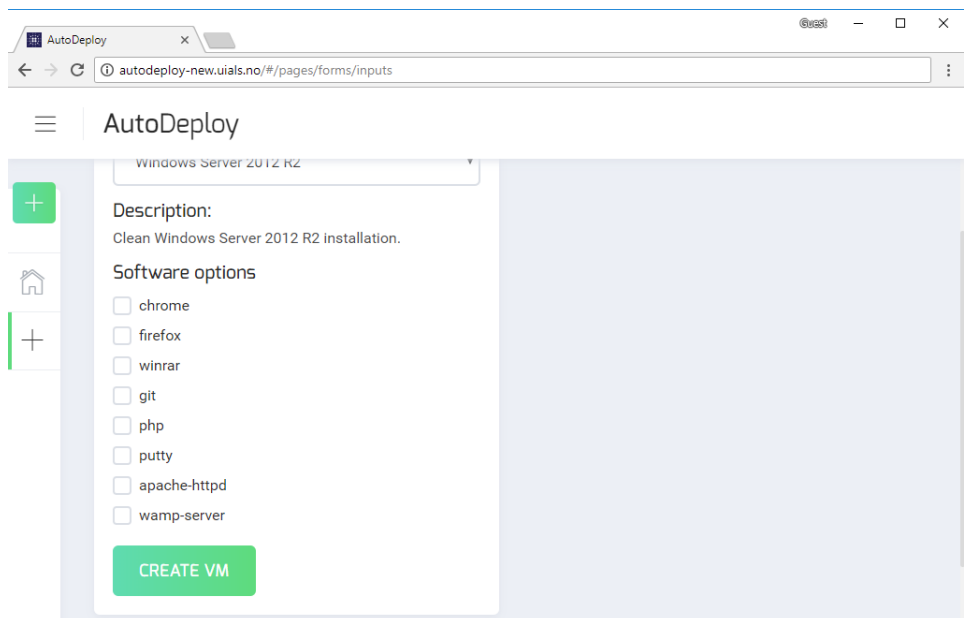
long as it is without spaces and only use the characters [a-Z0-9]. The resulting name will be appended to a prefix containing your NTNU username - like this: **olanord-testvm2**, where testvm2 is the name entered in the form and olanord is the NTNU username.

Operating System

The next thing you will have to select is the operating system template. As of now, there is only Windows Server 2012 R2 and Ubuntu Server 16.04 available for selection. It is very possible to supply more templates, but this must be developed by the data-lab team.

Some OS templates will also come pre-installed and pre-configured with some kinds of software. Examples for the future might be Microsoft SQL Server or Docker.

Software options



Next, you have the option to select software that shall be installed on your virtual

machine. This is a very simple process, so no configuration will be done to the software. There can be a wide range of selections, and they also vary with the kind of template you have selected.

Click **CREATE VM** to create the virtual machine.

VM Creation complete

Finally, the virtual machine you have worked so hard to get is finally under way. The creation process can take anywhere from 10 minutes to 2 hours, depending on the load of the servers and the size of the template.

When the Virtual Machine is finished and ready for use, an email will be sent to your student email address containing the IP address, the password and other detail regarding the virtual machine. The username for authentication will always be your NTNU username.

Example email

AutoDeploy

Hello!

Your AutoDeploy-VM mariusbf-testvm2 is ready for use!

IP: 158.38.101.222

Username: The same as your NTNU username.

Password: [REDACTED]

Please change your password at first log-in. Log in to AutoDeploy for more information.

Please note: **NEVER** save data on the virtual machine without proper backups. **DO NOT** use the virtual machine as backup-location. Always treat the virtual machine and its resources as volatile.

5.2.3 Connecting to the virtual machines

In the previous section we described how you can create your own virtual machine. In the end, you would receive an email with the details of the virtual machine; an IP and a password. How do you use this information to connect and use the virtual machine?

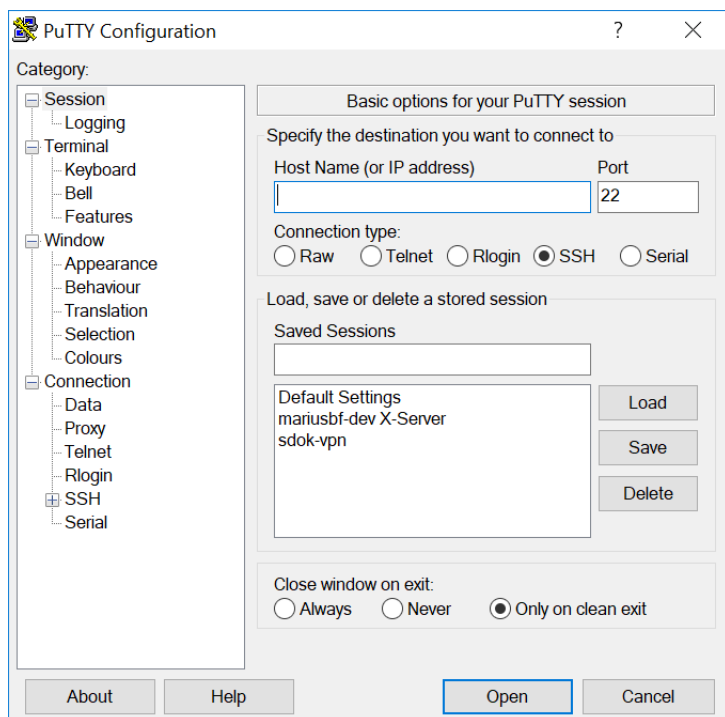
Connecting to a Linux VM

The connection to a Linux VM is done via Secure Shell, or SSH. This is a secure and encrypted way to communicate with the terminal sessions of your server. To connect to your server via this protocol you will need a SSH client.

Connecting from a Windows PC

The Windows PC does not come with a SSH client, so we need to retrieve a free and good SSH client from the internet. We recommend PuTTY. PuTTY can be retrieved from <https://www.putty.org/>. Please navigate there with your browser and download PuTTY before continuing.

Open PuTTY, and enter the IP address sent to you in the email in the **Hostname (or IP address)** field.



After entering the IP, make sure PuTTY is set to use SSH. You can select SSH under **Connection type**.

Click **Open**. If everything is configured correctly, you will get a black window asking for a username. Enter your NTNU username, for example **olanord**. Press enter, and you will be asked for a password. Enter the password you received in the email.

The first time you log in to a Linux VM it will ask you to change your password immediately after login. You have to first supply with the password you received in the email, and then enter a good and secure password twice. This will **disconnect** PuTTY, do not worry. Make sure you use the new password when you re-connect. 3 failed password attempts will block your IP.

Connecting from Mac or Linux

Mac and Linux both have built in SSH clients. Open Terminal or similar terminal emulators and enter the following command. **Replace the username and IP, of course:**

```
ssh olanord@158.38.101.xxx
```

This will bring you to a prompt asking for password. Enter the password supplied in the email and press enter.

The first time you log in to a Linux VM it will ask you to change your password immediately after login. You have to first supply with the password you received in the email, and then enter a good and secure password twice. This will **disconnect** SSH, do not worry. Make sure you use the new password when you re-connect. 3 failed password attempts will block your IP.

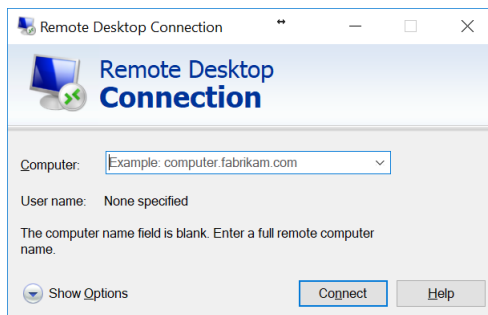
Connecting to a Windows machine

The Windows virtual machines use Remote Desktop Protocol to deliver you a remote desktop.

Connecting from Windows

Open **Remote Desktop Connection**, enter the IP or hostname of your virtual machine in the **Computer**-field. Click **Connect**. It will ask for your username and password. Enter your NTNU username, for example **olanord**, and the password you have received in the email.

After logging in to the Windows virtual machine, please change the log-in password as soon as possible. Do not use the default password.



Connecting from other operating systems

You will need a compatible RDP client. On Mac, the recommended client is **Microsoft Remote Desktop**, available in the App Store. On Linux, any RDP client can be used. A good example is Remmina, available on the website <https://www.remmina.org/>. The connection is performed in a very similar way as from Windows, but it might of course vary from client to client.

5.2.4 Managing the virtual machine

It is important that you regularly run software updates on the virtual machine. You do this with **apt-get update; apt-get upgrade;** on Linux, and with **Windows Update** on Windows. It is also very important that you regularly check that the software running on the VM is the intended software.

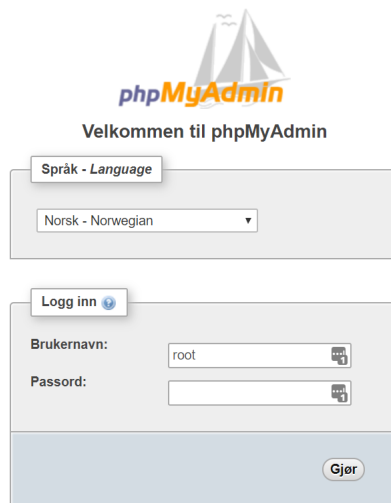
Never use simple passwords or deploy un-safe applications. **It will get hacked.**

5.3 Administration of AutoDeploy

There are some aspects of the administration of AutoDeploy that we sadly will not be able to cover in this section, but the most important parts are covered.

5.3.1 Managing the database

To manage the database, navigate to **<http://autodeploy-new.uials.no/phpmyadmin>**. The username is **root** and the password is the standard UIALS password (without). From this page you can manage the **adp** database and perform the required operations.



The screenshot shows the phpMyAdmin interface. At the top center is the phpMyAdmin logo, which consists of a stylized sailboat icon above the text 'phpMyAdmin'. Below the logo is the text 'Velkommen til phpMyAdmin'. Underneath this is a language selection section with a dropdown menu currently set to 'Norsk - Norwegian'. Below the language section is a login form. The login form has a 'Logg inn' button with a blue arrow icon. Below the button are two input fields: 'Brukernavn:' with the value 'root' and 'Passord:'. At the bottom right of the login form is a 'Gjør' button.

5.3.2 Starting, stopping and managing the AutoDeploy services

The AutoDeploy server has two services installed specially from AutoDeploy, as well as MySQL and Apache2.

1. autodeploy-api
2. autodeploy-queue

The autodeploy-api service is the Web RESTful API that is available through

apache reverse proxy at `autodeploy-new.uials.no/api` (reverse-proxyed to `127.0.0.1:5000/`).

The `Autodeploy-queue` service is the queue manager that works the queue and performs VMWare actions. If this service stops, you will not be able to power off, power on or create virtual machines.

Managing the services

To start or stop a process, you can enter:

```
sudo service autodeploy-api start
sudo service autodeploy-api stop
```

Also, if you want to view the status of a service, you type this:

```
sudo service autodeploy-api status
```

The logs from the services are written to `syslog`. To view the logs raw, enter this:

```
sudo less /var/log/syslog
```

You can also tail the log and view only items from a selected service with `tail` and `grep`, like this:

```
tail -f /var/log/syslog | grep autodeploy-api
```

5.3.3 Managing the virtual machines

There are a few tasks that might become relevant, such as renaming a virtual machine or trouble-shooting.

Renaming a virtual machine

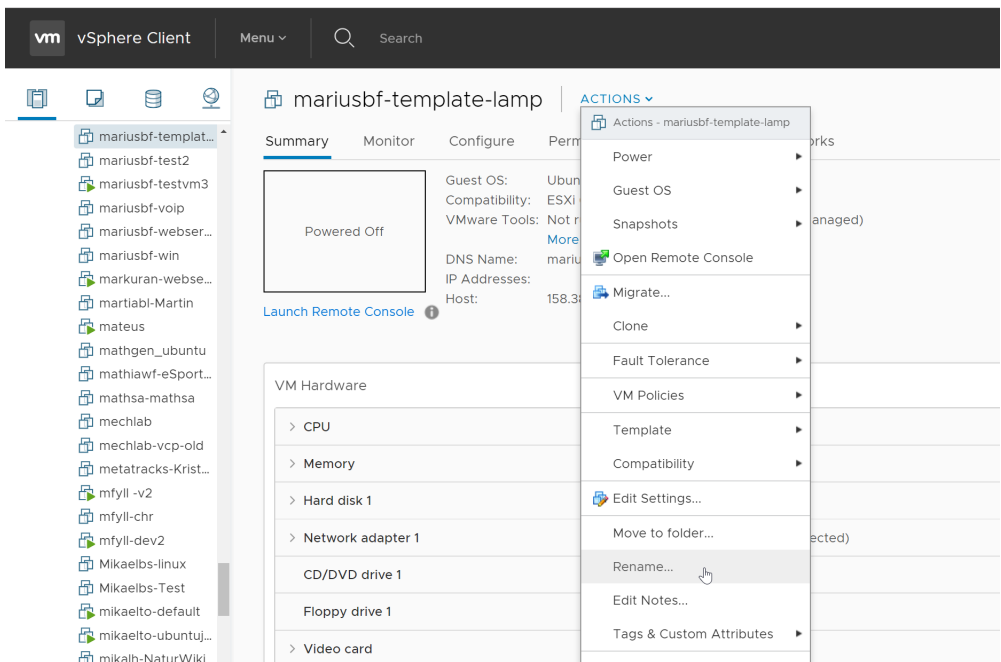
The first step is to rename the virtual machine in the database. You do this by connecting to the database with `phpMyAdmin` (described above) and navigating

to the **adp.vm** table. From there you can search for the VM by name using the **Search** tab. Re-name it accordingly.

The next step is to rename the virtual machine in VMWare. You do this by logging in to VMWare with your UIALS credentials, find the VM and rename it. The name in the database and in VMWare has to be 100% the same.

The last step is to either re-boot the re-named VM, the start-up script should take care of the renaming. The other way to do this to avoid downtime is to do it manually in the command line. Like this, replacing **newhostname**:

```
sudo hostname newhostname
sudo echo newhostname > /etc/hostname
sudo echo '127.0.0.1 newhostname' >> /etc/hosts
```



Troubleshooting the startup script

Sometimes, a virtual machine might not be configured correctly by the startup-script. Most of the time this is solved by rebooting the VM. You do this in VMWare vCenter.

If this does not solve the issue, you can try to re-deploy a new Virtual Machine, maybe with a new and different name?

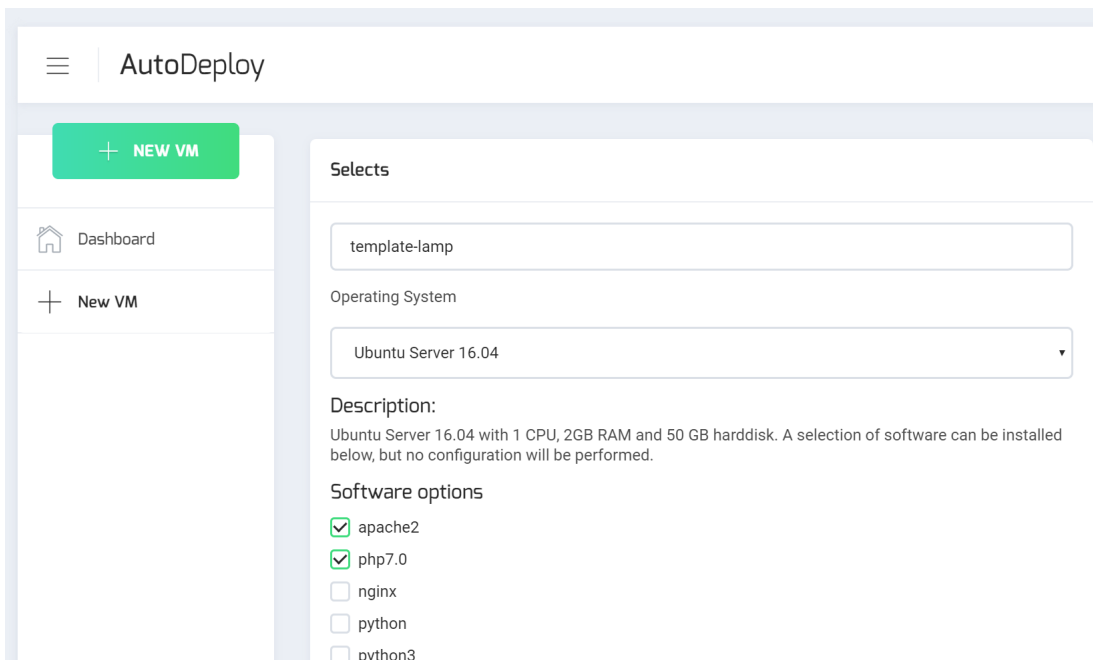
5.3.4 Creating a new template

It is important to first understand what a template is before you set out to create a new template. In short, a template is a virtual machine that is configured to be cloned and deployed with a start-up agent. The concept of templates is described in more detail earlier in this paper.

Step 1: Cloning an existing template

The easiest way to create a new template is to clone an existing virtual machine, and the easiest way to do this is to simply deploy a virtual machine with AutoDeploy. Follow the guide above to create a virtual machine.

There is one pre-cursor for this to work - the desired operating system has to be available for deployment already. If you intend to create a new template with a different OS, for example CentOS or FreeBSD, you need to create a new template and create a modified startup-script for this purpose.



The screenshot shows the AutoDeploy web interface. On the left is a navigation sidebar with a hamburger menu icon, a 'NEW VM' button, and links for 'Dashboard' and 'New VM'. The main content area is titled 'Selects' and contains a dropdown menu with 'template-lamp' selected. Below this is an 'Operating System' dropdown menu with 'Ubuntu Server 16.04' selected. A 'Description' section follows, stating: 'Ubuntu Server 16.04 with 1 CPU, 2GB RAM and 50 GB harddisk. A selection of software can be installed below, but no configuration will be performed.' Under 'Software options', there are five checkboxes: 'apache2' (checked), 'php7.0' (checked), 'nginx' (unchecked), 'python' (unchecked), and 'python3' (unchecked).

Step 2: Modifying the VM

Make sure you do all modifications with the **hialsvmadmin/Administrator** accounts, depending on operating system.

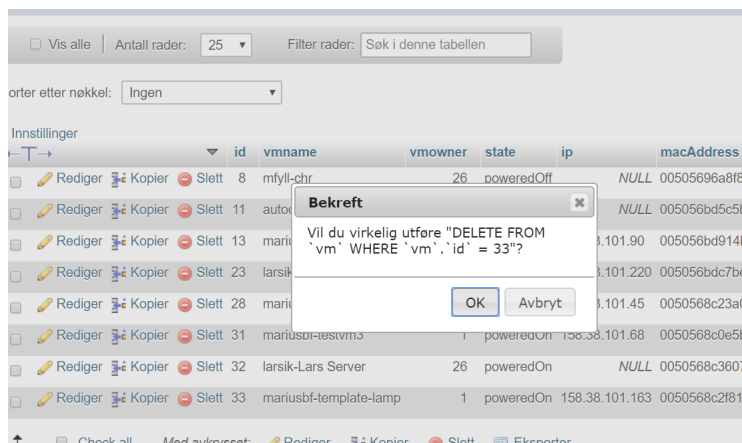
So, one of the best reasons to create a new Virtual Machine template is to have pre-configured software installed to save time. When you have the virtual machine ready, all you have to do is make sure that all software you install is for **ALL** users. You can do all necessary configuration, but this is also where you have to remember to configure everything for all users.

Step 3: Re-naming the virtual machine

Follow the guide above to rename the virtual machine. The recommended template name format is **autodeploy-opeartingsystem-software**. The name **must** be unique. An example can be: **autodeploy-windows2012r2-mssql**

Step 4: Clear the template

Delete the user that was created when the template was deployed the first time. Also, navigate to `adp.vm` in the database and remove the VM from AutoDeploy.



Step 5: Add the template to AutoDeploy

Connect to the database with phpMyAdmin, described above. Navigate to the table `adp.oslist`. There are 5 fields you need to fill in. Name and description are user-friendly names and descriptions of the template. Options is available software options for the template. The template field is where you enter the name of the template. It is very important that this field is correct. Also, if you want to disable a template you can set the disabled to 1.

Example options field: `["apache2", "php7.0", "nginx", "python", "python3", "python-pip", "python3-pip", "default-jre", "default-jdk", "docker", "postgresql", "tmux"]`

The screenshot shows the phpMyAdmin interface for the 'oslist' table. The table structure is as follows:

id	name	description	options
3	Windows Server 2012 R2	Clean Windows Server 2012 R2 installation.	["chrome","firefox"]
4	Ubuntu Server 16.04	Ubuntu Server 16.04 with 1 CPU, 2GB RAM and 50 GB ...	["apache2","php"]

The interface includes a search bar, sorting options, and a list of actions for each row (Edit, Copy, Delete, Export).

The screenshot shows the 'oslist' table field definitions in phpMyAdmin. The fields are defined as follows:

Kolonne	Type	Funksjon	Null	Verdi
id	int(11)			
name	varchar(45)			Ubuntu Server 16.04 with LAMP
description	text			Ubuntu Server with Apache, MySQL and PHP. Note: MySQL root password is entay .
options	longtext			
disabled	tinyint(4)			0
template	varchar(1111)			autodeploy-linux-lamp

The interface includes a 'Gjør' button at the bottom right.

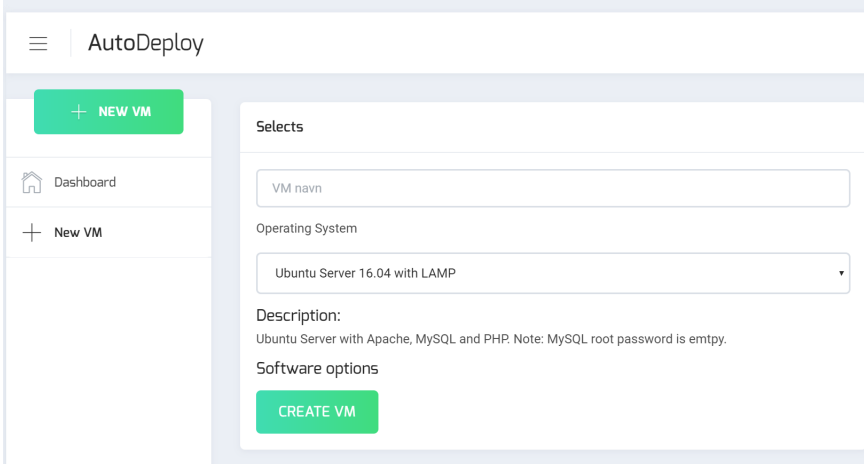
Step 6: Test the template

Make sure to test the template a few times to make sure that the template works as desired.

Make sure to reboot and SWITCH OFF the VM Template after you are finished.

Complete!

After testing, the template should be ready for use. The AutoDeploy users can now take advantage of the template immediately.



The screenshot shows the 'AutoDeploy' web interface. On the left is a sidebar with a menu icon, a 'NEW VM' button, and links for 'Dashboard' and 'New VM'. The main content area is titled 'Selects' and contains a form with the following fields:

- 'VM navn' (text input)
- 'Operating System' (dropdown menu showing 'Ubuntu Server 16.04 with LAMP')
- 'Description:' (text area containing 'Ubuntu Server with Apache, MySQL and PHP. Note: MySQL root password is empty.')
- 'Software options' (text area)
- 'CREATE VM' (green button)

5.3.5 Disabling a template

Connect to the database with phpMyAdmin, described above. Navigate to the table **adp.oslist**. Find the template you wish to disable and set the field "disabled" to 1.

Chapter 6

Research Strategy

6.1 Research methods

To measure our success we need to look at our research questions. They aim to find a solution to a specific problem and are therefore exploratory as opposed to explanatory or descriptive. Also both qualitative and quantitative research methodology can fit our research questions. Our main question can be answered qualitatively by interviews with users or studying system interaction. Thus we get more of an insight in what this system has *meant* to them during their studies. [30]

Qualitative data, with their emphasis on peoples lived experiences, are fundamentally well suited for locating the meanings people place on events, processes, and structures of their lives and for connecting these *meanings* to the *social world* around them. [30, p. 11]

We questioned fellow students to supplement our own understanding and experience using the old system. Events like crashes, data-loss and general downtime was noted. Also general "feelings" of the students experiences such as frustrations provided a good qualitative understanding of the old system. We plan on doing the same when the new system is fully operational.

Identifying challenges is a major part of software development. Therefore we wanted to explore how our initial thoughts were changed over the course of the project. We set out knowing some of the challenges beforehand, and the rest were discovered by testing, implementing and research. Most of the literature for the development part of a project like this is invariably technical documents from developers like Oracle and Google.

We also wanted to set aside some time for research into the history and origins of vitalization to get a better understanding of what we are actually making. Most of these documents are dissertations and historical documents from the depths of the archives at IBM and Bell labs.

We used Google scholar so we could confirm that our sources are highly cited and trustworthy. Zotero was used to keep track of references. We used BibTeX to format the reference bibliography in our LaTeX document.

Chapter 7

Discussion

7.1 Introduction

In this chapter we will discuss what we have done, and how we could improve on what we have done, the errors we have done and what we could have done better.

Our findings suggest that it is very possible to offer a effective solution to the problem of incompatible platforms used by students. And, more importantly, make one that is easy enough to use for everyone, not just the computer engineering students.

Our choices of tools were for the most part good, but of course no project turns out exactly how we envisioned it when we set out.

7.2 Front-end

There were some hurdles that could have been avoided in the front end. In hindsight a lot of issues arose from utilizing a bad template and not building our site from the ground up. We spent a little too much time reverse engineering because of the poor documentation for the template. But, in the end this led to insight into how Angular is used in a professional setting, and the end result is very user friendly and stable.

7.3 Queue Manager

In the backend-section some compromises had to be made. One of them is the queue management. There is a lot of software out there designed to be queue managers, like RabbitMQ[31] or ZeroMQ.[32] This is software designed to deliver messages between software. Had we used something like this instead, the system would probably have been better. But designing a new one or implementing one correctly could have been a thesis in itself. A good implementation of a queue should also include error-handling in the queue. Most queue-managers have this, but the one implemented in AutoDeploy does not handle errors thrown by external

APIs (including VMWare). The implementation is somewhat basic and only takes a very strict and defined set of commands. In the future, the queue manager can probably be replaced with a out of the box solution like the ones mentioned above.

7.4 Deployment of VMs

The deployment solution itself is not perfect, but it does everything we need it to. In most cases the deployment works without hitch, and gives the user a virtual machine within minutes. But, the use of a startup-script instead of a virtual machine agent is something that was not the best of decisions. This has, in the past, caused a lot of headache. Some times the virtual machine does not get its configuration when it is asked for, and then the VM is stuck in a ghost-type state until it gets rebooted. This could probably have been solved by utilizing a VM agent instead of a startup script. A lot of popular VM host providers, like Digital Ocean and Azure, use VM Agents.[33] This is a program or utility that always runs on the virtual machine. It communicates with control software via either proprietary channels or via HTTP RESTful APIs. This would enable the control software itself to take control over ghosted VMs, and make the deployment process more agile. In the future it is possible to replace the VM Startup script, just by utilizing the Script update functions.

7.5 Programming languages

In the end, the project has a lot of quirks and funky workarounds to make this work. We did encounter a lot of problems with the VMWare implementation itself as well, both because the API changes with time, and because after a lot of trial and error we decided that the Python SDK was not good enough and we switched back to Perl. This is why the Queue Manager is written in Perl. Perl is a good language, no doubt, but we did have a vision to keep the number of programming languages to a minimum. We ended up with 3 major languages, and that is not too bad (Python, Perl and TypeScript (Angular)).

7.6 Planned functions

There were a some planned functions that sadly did not make it in to the current version, some of them is were not implemented because of technical issues and others because of time constraints. A function that we used a lot of time to try and implement was to beeing able to show the user a percentage on triggered processes. For example, we could show the user that the VM they created is 67% complete. After a lot of work, we had to conclude that this function was not technically feasible because of the shortcomings of the VMWare SDK. Today you will get a toast notification saying "Creating VM..." in a message pop up when the VM is being created. This you also get an email when the VM is ready, so that you don't have to wait around for the creation to finish.

Another planned function was having a web VNC view of the VM. This would mean no ssh client would be needed, and a full terminal window would be accessible right in the web interface. This would enable total control of the VM, with no software at all installed on their local machine. This could have been possible with a lot of trial and error, but in the end we decided to postpone this function because of time constraints - and because of the risk of creating a security hole if not implemented correctly.

We also did look into better templating. A vision we had was to use a script language called Ansible to create templates automatically. This would enable a teacher or, possibly, a student, to create templates very fast. All you would have to do is to create an Ansible-playbook and add this to an OS template. Sadly, this was a function that we were not able to implement due to time constraints, but also because of the way templating works in the current version. This might be a fine project for students coming after us to tackle.

7.7 The future

All in all, the project turned out great. We have learned a lot from the mistakes. Not only is it important reach for the impossible, but it is also important to sometimes allow yourself to fail. The best ways to learn is to challenge yourself. Sadly, in this project, a lot of the challenges were due to technical limitations in the VMWare Hypervisor and its API / SDK. There is a consensus in our team that replacing VMWare with another hypervisor like Xen or OpenVZ would open up a lot more opportunities and possibilities - but it would also require a complete rewrite of AutoDeploy back-end.

We believe that the new AutoDeploy will give students access to a broader set of computer tools, by allowing them to easily create Virtual Machines. The VMs they create can be used to learn, explore and discover new things. Who knows, maybe the new big thing on the internet one day will be developed by a student at NTNU Aalesund, using the endless opportunities you get with a VM from AutoDeploy.

Chapter 8

Conclusion

In this project we wanted to create a better VM solution for NTNU. We believe we succeeded in doing that, since the system we delivered is much more user friendly and stable. Even though there are still some issues that will have to be corrected, we have created a platform that the students can use in many years to come, and other students can build upon.

We managed to keep the front- and back-end separate, which was an important goal for both us and NTNU. This means if either the site or back-end crashes the other will not, though you will not be able to log in. It also means it's possible to work on either one without knowing anything about the other.

Through research we found the best tools to be a mix of Angular, Python, and Perl. They provide performance, maturity and good documentation.

In the end this project was very much a success. We are very pleased with successfully tying together two very different areas of computer science. One very user oriented, the other traditionally reserved only for the most dedicated of the IT community. Presenting this very intimidating software in an approachable way; was perhaps the biggest challenge we faced.

Bibliography

- [1] Karissa Miller and Mahmoud Pegah. ‘Virtualization: Virtually at the Desktop’. In: *Proceedings of the 35th Annual ACM SIGUCCS Fall Conference*. SIGUCCS ’07. New York, NY, USA: ACM, 2007, pp. 255–260. ISBN: 978-1-59593-634-9. DOI: 10.1145/1294046.1294107.
- [2] S. Agrawal, R. Biswas and A. Nath. ‘Virtual Desktop Infrastructure in Higher Education Institution: Energy Efficiency as an Application of Green Computing’. In: *2014 Fourth International Conference on Communication Systems and Network Technologies*. Apr. 2014, pp. 601–605. DOI: 10.1109/CSNT.2014.250.
- [3] IBM. *Virtualization*. en. a key to virtualization. Google-Books-ID: BNS7KNMAa_IC. PediaPress, 2014. URL: <http://pediapress.com/books/show/virtualization-a-key-to-virtualization-wo/>.
- [4] Dennis M Ritchie, Ken Thompson and Bell Laboratories. ‘The UNIX Time-Sharing System’. en. In: *University of California Berkeley 17.7 (1974)*, p. 11.
- [5] Dennis M. Ritchie. ‘The evolution of the unix time-sharing system’. en. In: *Language Design and Programming Methodology*. Ed. by G. Goos et al.

- Vol. 79. Berlin, Heidelberg: Springer Berlin Heidelberg, 1980, pp. 25–35. ISBN: 978-0-13-110362-7. DOI: 10.1007/3-540-09745-7_2. URL: http://link.springer.com/10.1007/3-540-09745-7_2 (visited on 22/05/2018).
- [6] James Gosling et al. *The Java® Language Specification*. en. Specification. Oracle, LLC., Feb. 2018, p. 772.
- [7] Melinda Varian. ‘VM and the VM Community: Past, Present, and Future’. en. In: *Princeton University* (Aug. 1997), p. 70.
- [8] Oracle LLC. *1.1.1. Brief History of Virtualization*. 2012. URL: https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html (visited on 10/04/2018).
- [9] Oracle LLC. *Oracle® VM*. 2012. URL: https://docs.oracle.com/cd/E26996_01/E18549/html/index.html (visited on 10/04/2018).
- [10] Oracle LLC. *User’s Guide for Release 3.0.3 - 1.1.2. Hypervisor*. 2012. URL: https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1011.html (visited on 10/04/2018).
- [11] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (visited on 10/04/2018).
- [12] Julian F. Reschke and Roy T. Fielding. *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. en. Specification. World Wide Web Consortium, 2014. URL: <https://tools.ietf.org/html/rfc7235> (visited on 10/04/2018).
- [13] The Perl Foundation. *Perl.com*. en-us. 2018. URL: <https://www.perl.com/about> (visited on 10/04/2018).
- [14] Scott Trent et al. ‘Performance Comparison of PHP and JSP as Server-Side Scripting Languages’. en. In: *Middleware 2008*. Ed. by Valérie Issarny and Richard Schantz. Vol. 5346. Berlin, Heidelberg: Springer Berlin Heidelberg,

- 2008, pp. 164–182. ISBN: 978-3-540-89855-9. DOI: 10.1007/978-3-540-89856-6_9. URL: http://link.springer.com/10.1007/978-3-540-89856-6_9 (visited on 22/05/2018).
- [15] World Wide Web Consortium et al. *Usage Statistics and Market Share of Server-side Programming Languages for Websites, May 2018*. Tech. rep. World Wide Web Consortium, 2018. URL: https://w3techs.com/technologies/overview/programming_language/all (visited on 22/05/2018).
- [16] *Comparing Node.js vs PHP Performance*. en. Mar. 2015. URL: <http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/> (visited on 22/05/2018).
- [17] *Golang vs Node.js Comparison: Performance, Speed, Scalability and Other | DA-14*. Jan. 2018. URL: <https://da-14.com/blog/golang-vs-nodejs-comparison-performance-speed-scalability-and-other> (visited on 22/05/2018).
- [18] Alan A. A. Donovan and Brian W. Kernighan. *The Go Programming Language*. en. Google-Books-ID: SJHvCgAAQBAJ. Addison-Wesley Professional, Nov. 2015. ISBN: 978-0-13-419056-3.
- [19] Facebook. *React - A JavaScript library for building user interfaces*. en. Specification. Facebook, Inc., 2018. URL: <https://reactjs.org/index.html> (visited on 23/05/2018).
- [20] Angular.IO. *Angular - Architecture overview*. 2018. URL: <https://angular.io/guide/architecture> (visited on 11/05/2018).
- [21] Indiana University. *What are flat file and relational databases?* 2017. URL: <https://kb.iu.edu/d/ahrp> (visited on 04/05/2018).

- [22] S. Sumathi and S. Esakkirajan. *Fundamentals of Relational Database Management Systems*. en. Google-Books-ID: MQNtCQAAQBAJ. Springer, Mar. 2007. ISBN: 978-3-540-48399-1.
- [23] VMWare. *How VMware HA Works - VMware vSphere 4 - ESX and vCenter Server*. 2018. URL: https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.availability.doc_41/c_useha_works.html (visited on 22/05/2018).
- [24] Federico Calzolari. 'High availability using virtualization'. In: *arXiv:0910.1719 [cs]* (Oct. 2009). arXiv: 0910.1719. URL: <http://arxiv.org/abs/0910.1719> (visited on 11/04/2018).
- [25] Pace Eugenio. *JWT_IO - JSON Web Tokens Introduction*. en. Tech. rep. Auth0, Inc, 2018. URL: <http://jwt.io/> (visited on 14/05/2018).
- [26] Hildegunn Vada. *Feide integration guide*. en. Tech. rep. UNINETT, A/S., 2016, p. 29.
- [27] Oracle LLC. *About VMware Tools - vSphere Documentation Center*. 2018. URL: https://pubs.vmware.com/vsphere-50/index.jsp?topic=%2Fcom.vmware.vsphere.upgrade.doc_50%2FGUID-28C39A00-743B-4222-B697-6632E94A8E72.html (visited on 05/05/2018).
- [28] Steve Faulkner (The Paciello Group) et al. *HTML 4.01 Specification*. en. Specification. World Wide Web Consortium, 1999, p. 389. URL: <http://webx.ubi.pt/~hgil/utills/HTML/html40.pdf>.
- [29] Brendan Eich, Rand Mckinney and Oracle LLC. *JavaScript Language Specification*. Wesley Publishing Company, Nov. 1996.
- [30] Matthew B. Miles, A. Michael Huberman and Johnny Saldana. *Qualitative Data Analysis*. en. SAGE, Apr. 2013. ISBN: 978-1-4522-5787-7.
- [31] *RabbitMQ - Messaging that just works*. 2018. URL: <https://www.rabbitmq.com/> (visited on 26/05/2018).

- [32] *Distributed Messaging - zeromq*. 2017. URL: <http://zeromq.org/> (visited on 26/05/2018).
- [33] Daniel Gicklhorn. *Azure Virtual Machine Agent Overview*. en-us. 2018. URL: <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/agent-user-guide> (visited on 05/05/2018).